AWS Certified Developer Associate

By Stéphane Maarek

COURSE →

EXTRA PRACTICE EXAMS

Disclaimer: These slides are copyrighted and strictly for personal use only

- This document is reserved for people enrolled into the AWS Certified Developer course by Stephane Maarek
- **Please do not share this document**, it is intended for personal use and exam preparation only, thank you.
- If you've obtained these slides for free on a website that is not the course's website, please reach out to piracy@datacumulus.com. Thanks!
- Best of luck for the exam and happy learning!

Table of Contents

Table of Contents

Table of Contents

AWS Certified Developer Associate Course

DVA-C02

Welcome! We're starting in 5 minutes

- We're going to prepare for the Certified Developer exam – DVA-C02
- It's a challenging certification, so this course will be long and interesting

- We will cover over 30 AWS services
- AWS / IT Beginners welcome! (but take your time, it's not a race)
- You don't need to be a developer to pass this exam

- Even if you've done AWS Certified Solutions Architect, don't skip lectures.

What's AWS?

- AWS (Amazon Web Services) is a Cloud Provider
- They provide you with servers and services that you can use on demand and scale easily

- AWS has revolutionized IT over time
- AWS powers some of the biggest websites in the world
  - Amazon.com
  - Netflix

What we'll learn in this course


Amazon EC2


Amazon ECR


Amazon ECS


AWS Elastic Beanstalk


AWS Lambda


Elastic Load Balancing


Amazon CloudFront


Amazon Kinesis

Amazon Route 53


Amazon S3


Amazon RDS


Amazon Aurora


Amazon DynamoDB


Amazon ElastiCache


Amazon SQS


Amazon SNS


AWS Step Functions


Auto Scaling


Amazon API Gateway


Amazon SES


Amazon Cognito


IAM


Amazon CloudWatch


Amazon EC2 Systems Manager


AWS CloudFormation


AWS CloudTrail


AWS CodeCommit


AWS CodeBuild


AWS CodeDeploy


AWS CodePipeline

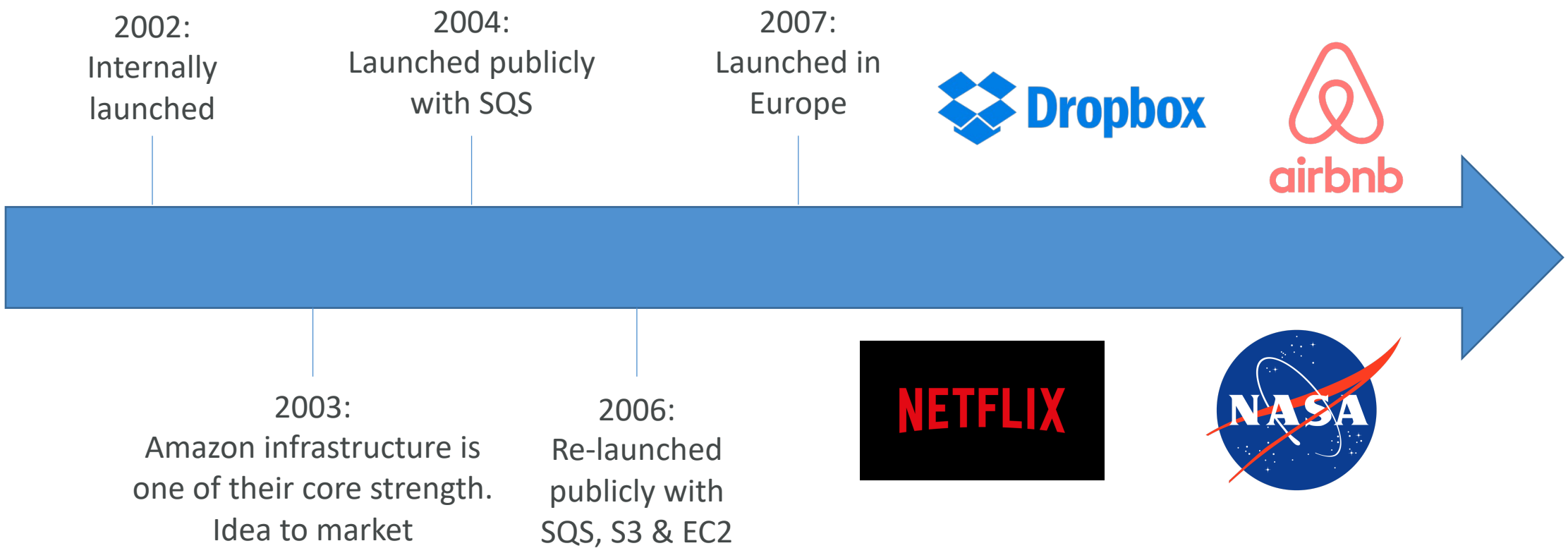

AWS X-Ray


AWS KMS

Navigating the AWS spaghetti bowl

Getting started with AWS

AWS Cloud History



AWS Cloud Number Facts

- In 2023, AWS had $90 billion in annual revenue
- AWS accounts for 31% of the market in Q1 2024 (Microsoft is 2nd with 25%)
- Pioneer and Leader of the AWS Cloud Market for the 13th consecutive year
- Over 1,000,000 active users

Figure 1: Magic Quadrant for Strategic Cloud Platform Services

Gartner Magic Quadrant

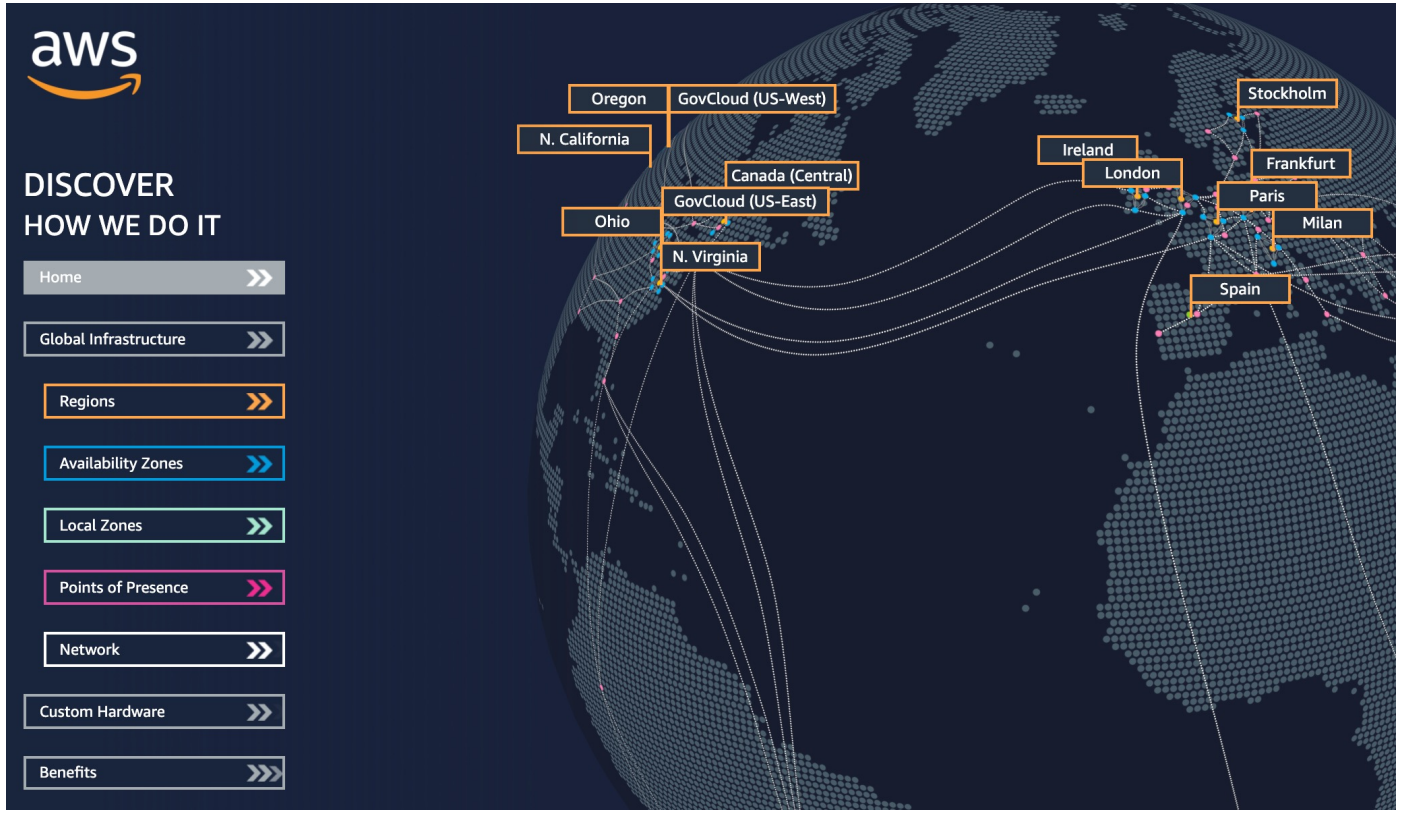
AWS Cloud Use Cases

- AWS enables you to build sophisticated, scalable applications
- Applicable to a diverse set of industries
- Use cases include
  - Enterprise IT, Backup & Storage, Big Data analytics
  - Website hosting, Mobile & Social Apps
  - Gaming

AWS Global Infrastructure

- AWS Regions
- AWS Availability Zones
- AWS Data Centers
- AWS Edge Locations / Points of Presence

- https://infrastructure.aws/



AWS Regions

- AWS has **Regions** all around the world
- Names can be us-east-1, eu-west-3…
- A region is a **cluster of data centers**
- Most AWS services are region-scoped

https://aws.amazon.com/about-aws/global-infrastructure/

US East (N. Virginia)	us-east-1
US East (Ohio)	us-east-2
US West (N. California)	us-west-1
US West (Oregon)	us-west-2
Africa (Cape Town)	af-south-1
Asia Pacific (Hong Kong)	ap-east-1
Asia Pacific (Mumbai)	ap-south-1
Asia Pacific (Seoul)	ap-northeast-2
Asia Pacific (Singapore)	ap-southeast-1
Asia Pacific (Sydney)	ap-southeast-2
Asia Pacific (Tokyo)	ap-northeast-1
Canada (Central)	ca-central-1
Europe (Frankfurt)	eu-central-1
Europe (Ireland)	eu-west-1
Europe (London)	eu-west-2
Europe (Paris)	eu-west-3
Europe (Stockholm)	eu-north-1
Middle East (Bahrain)	me-south-1
South America (São Paulo)	sa-east-1

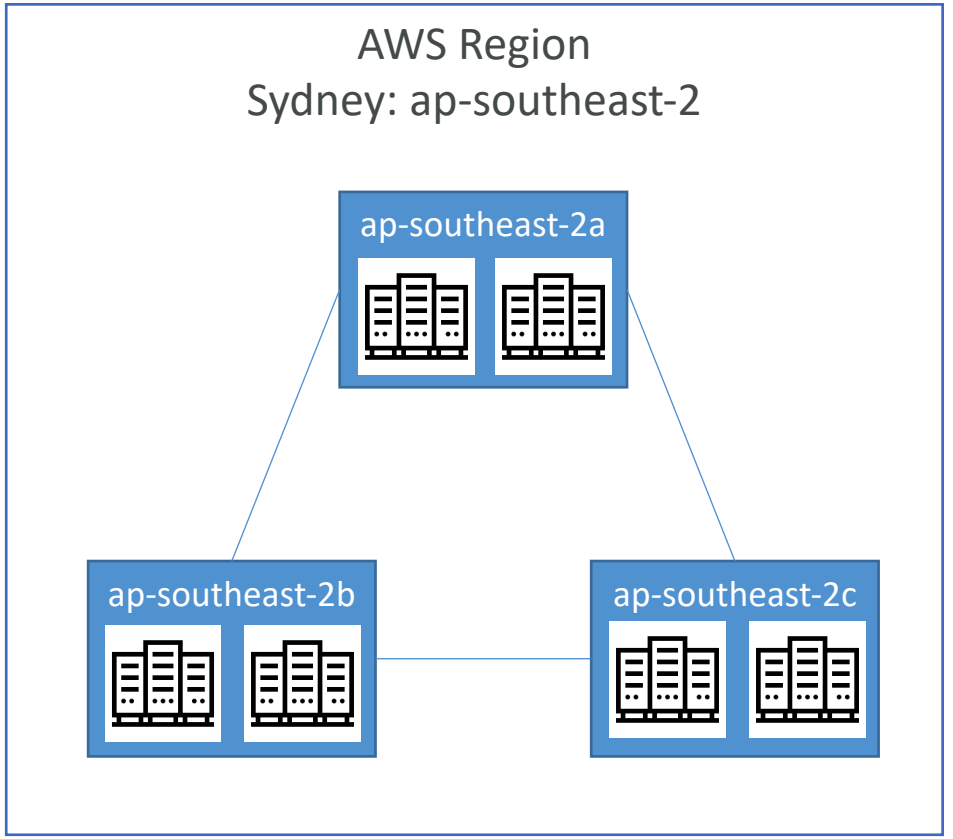
How to choose an AWS Region?

If you need to launch a new application, where should you do it?

- Compliance with data governance and legal requirements: data never leaves a region without your explicit permission
- Proximity to customers: reduced latency
- Available services within a Region: new services and new features aren't available in every Region
- Pricing: pricing varies region to region and is transparent in the service pricing page

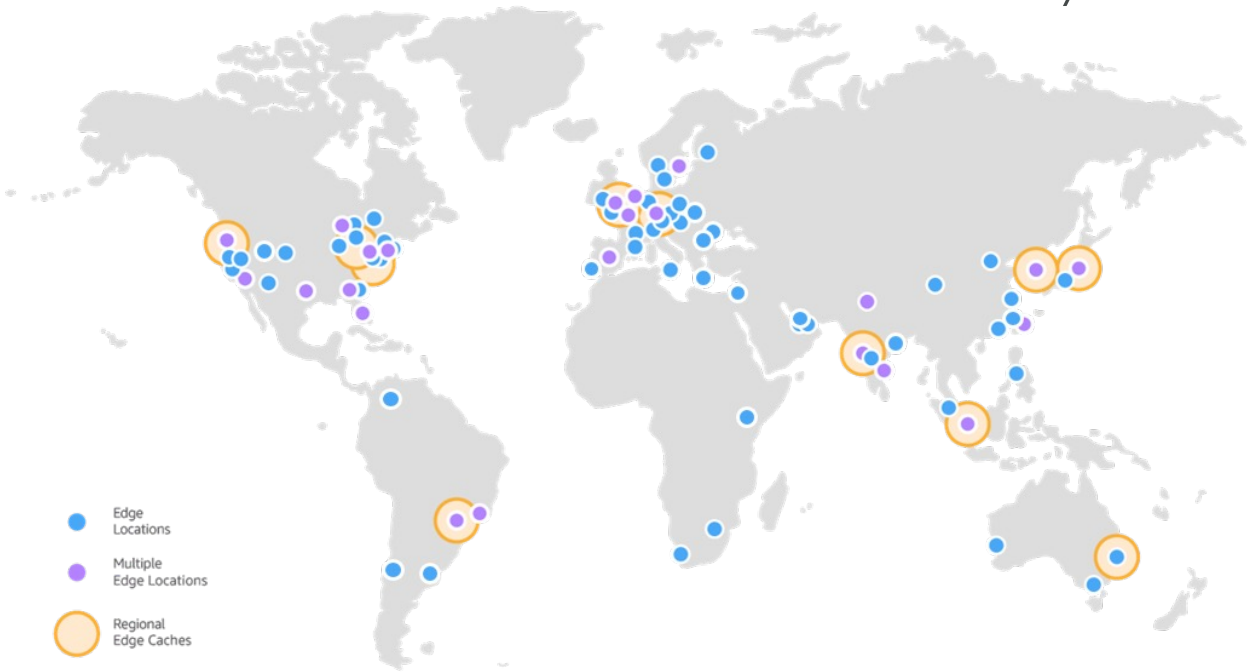
AWS Availability Zones

- Each region has many availability zones (usually 3, min is 3, max is 6). Example:
  - ap-southeast-2a
  - ap-southeast-2b
  - ap-southeast-2c
- Each availability zone (AZ) is one or more discrete data centers with redundant power, networking, and connectivity
- They're separate from each other, so that they're isolated from disasters
- They're connected with high bandwidth, ultra-low latency networking



AWS Points of Presence (Edge Locations)

- Amazon has 400+ Points of Presence (400+ Edge Locations & 10+ Regional Caches) in 90+ cities across 40+ countries
- Content is delivered to end users with lower latency

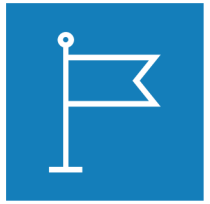


https://aws.amazon.com/cloudfront/features/

Tour of the AWS Console

- AWS has Global Services:
  - Identity and Access Management (IAM)
  - Route 53 (DNS service)
  - CloudFront (Content Delivery Network)
  - WAF (Web Application Firewall)

- Most AWS services are Region-scoped:
  - Amazon EC2 (Infrastructure as a Service)
  - Elastic Beanstalk (Platform as a Service)
  - Lambda (Function as a Service)
  - Rekognition (Software as a Service)

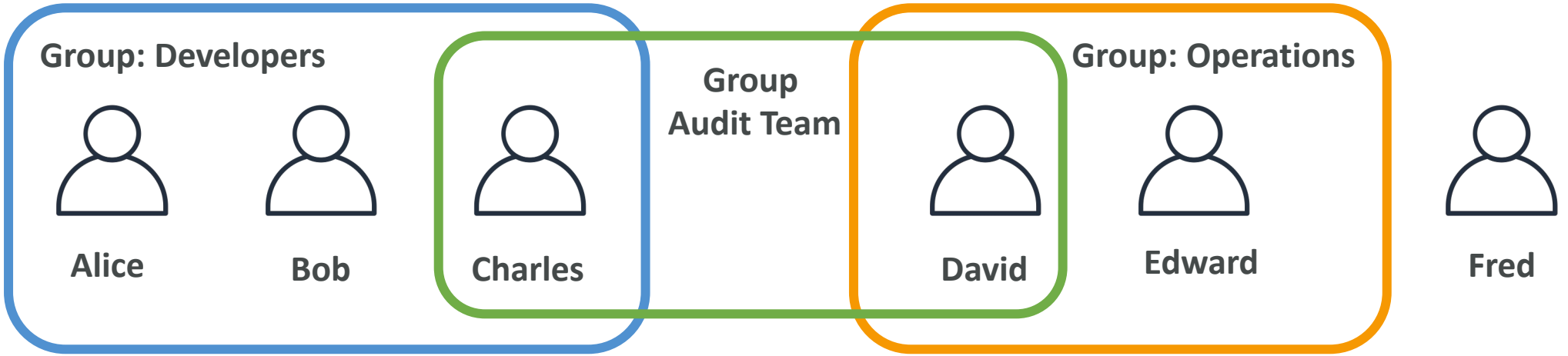


- Region Table: https://aws.amazon.com/about-aws/global-infrastructure/regional-product-services

AWS Identity & Access Management (AWS IAM)

IAM: Users & Groups

- IAM = Identity and Access Management, **Global** service
- **Root account** created by default, shouldn't be used or shared
- **Users** are people within your organization, and can be grouped
- **Groups** only contain users, not other groups
- Users don't have to belong to a group, and user can belong to multiple groups

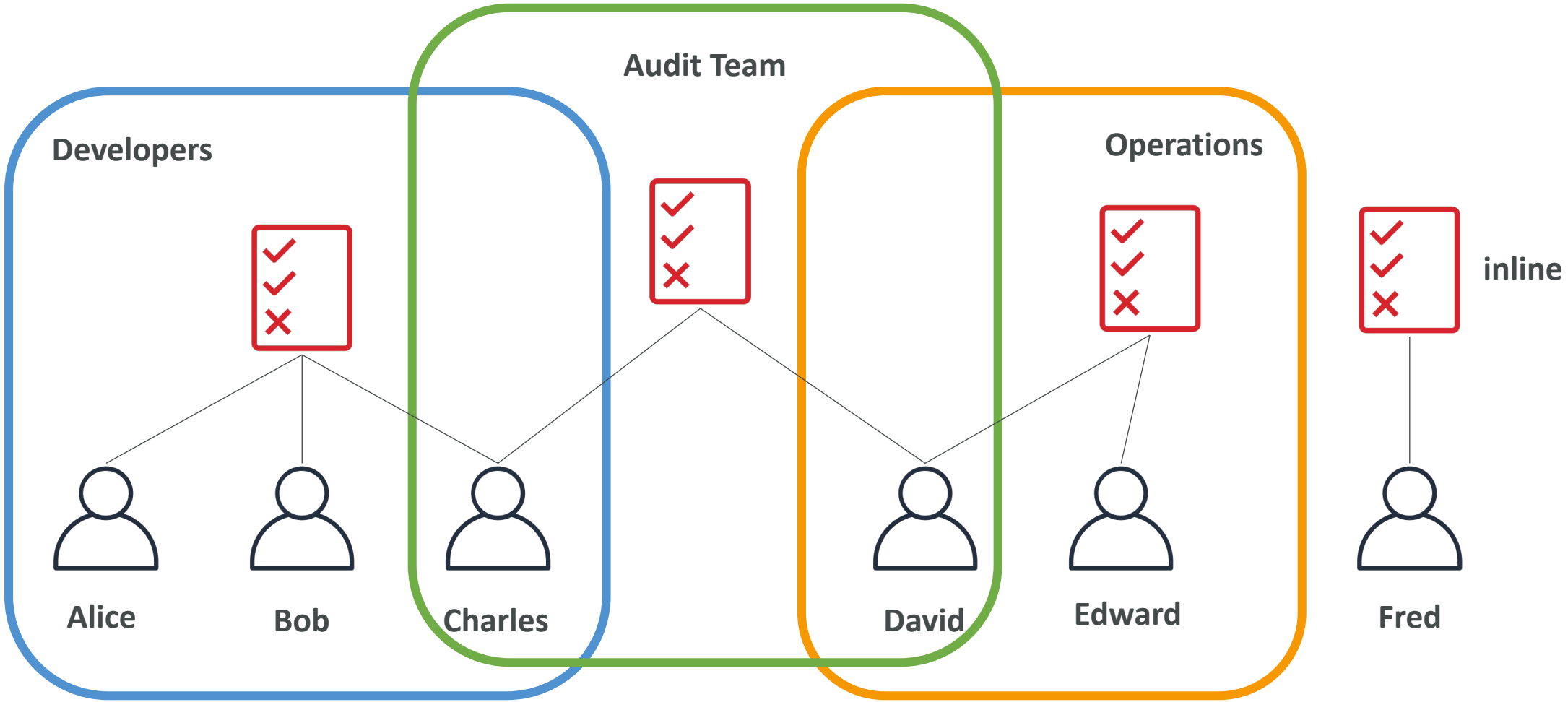


IAM: Permissions

- **Users or Groups** can be assigned JSON documents called policies
- These policies define the **permissions** of the users
- In AWS you apply the **least privilege principle**: don't give more permissions than a user needs

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "ec2:Describe*",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "elasticloadbalancing:Describe*",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:ListMetrics",
                "cloudwatch:GetMetricStatistics",
                "cloudwatch:Describe*"
            ],
            "Resource": "*"
        }
    ]
}
```

IAM Policies inheritance



IAM Policies Structure

- Consists of
  - **Version:** policy language version, always include "2012-10-17"
  - **Id:** an identifier for the policy (optional)
  - **Statement:** one or more individual statements (required)
- Statements consists of
  - **Sid:** an identifier for the statement (optional)
  - **Effect:** whether the statement allows or denies access (Allow, Deny)
  - **Principal:** account/user/role to which this policy applied to
  - **Action:** list of actions this policy allows or denies
  - **Resource:** list of resources to which the actions applied to
  - **Condition:** conditions for when this policy is in effect (optional)

```
{
    "Version": "2012-10-17",
    "Id": "S3-Account-Permissions",
    "Statement": [
        {
            "Sid": "1",
            "Effect": "Allow",
            "Principal": {
                "AWS": ["arn:aws:iam::123456789012:root"]
            },
            "Action": [
                "s3:GetObject",
                "s3:PutObject"
            ],
            "Resource": ["arn:aws:s3:::mybucket/*"]
        }
    ]
}
```

IAM – Password Policy

- Strong passwords = higher security for your account
- In AWS, you can setup a password policy:
  - Set a minimum password length
  - Require specific character types:
    - including uppercase letters
    - lowercase letters
    - numbers
    - non-alphanumeric characters
  - Allow all IAM users to change their own passwords
  - Require users to change their password after some time (password expiration)
  - Prevent password re-use

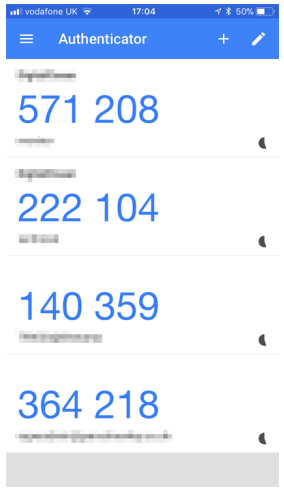
Multi Factor Authentication - MFA

- Users have access to your account and can possibly change configurations or delete resources in your AWS account
- You want to protect your Root Accounts and IAM users
- MFA = password *you know* + security device *you own*

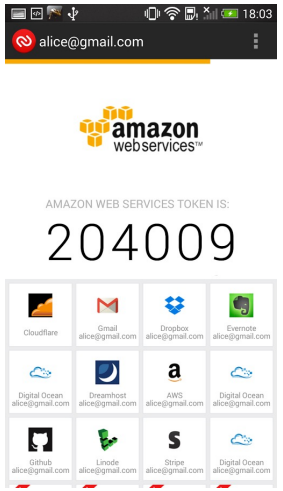
- Main benefit of MFA:
  if a password is stolen or hacked, the account is not compromised

MFA devices options in AWS

Virtual MFA device



Google Authenticator
(phone only)



Authy
(phone only)

Support for multiple tokens on a single device.

Universal 2nd Factor (U2F) Security Key

YubiKey by Yubico (3rd party)

Support for multiple root and IAM users using a single security key

MFA devices options in AWS

Hardware Key Fob MFA Device

Provided by Gemalto (3rd party)

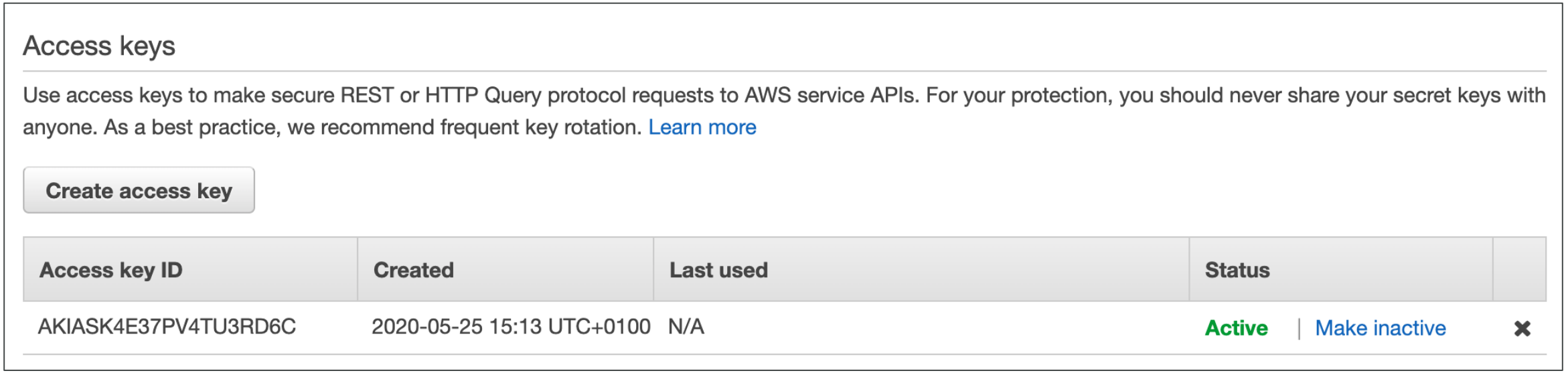
Hardware Key Fob MFA Device for AWS GovCloud (US)

Provided by SurePassID (3rd party)

How can users access AWS ?

- To access AWS, you have three options:
  - **AWS Management Console** (protected by password + MFA)
  - **AWS Command Line Interface (CLI):** protected by access keys
  - **AWS Software Developer Kit (SDK)** - for code: protected by access keys
- Access Keys are generated through the AWS Console
- Users manage their own access keys
- Access Keys are secret, just like a password. Don't share them
- Access Key ID ~= username
- Secret Access Key ~= password

Example (Fake) Access Keys



Access keys

Use access keys to make secure REST or HTTP Query protocol requests to AWS service APIs. For your protection, you should never share your secret keys with anyone. As a best practice, we recommend frequent key rotation. Learn more

Create access key

Access key ID	Created	Last used	Status	
AKIASK4E37PV4TU3RD6C	2020-05-25 15:13 UTC+0100	N/A	Active | Make inactive	✖

- Access key ID: AKIASK4E37PV4983d6C
- Secret Access Key: AZPN3zojWozWCndIjhB0Unh8239a1bzbzO5fqqkZq
- Remember: don't share your access keys

What's the AWS CLI?

- A tool that enables you to interact with AWS services using commands in your command-line shell
- Direct access to the public APIs of AWS services
- You can develop scripts to manage your resources
- It's open-source https://github.com/aws/aws-cli
- Alternative to using AWS Management Console

```
➜  ~ aws s3 cp myfile.txt s3://ccp-mybucket/myfile.txt
upload: ./myfile.txt to s3://ccp-mybucket/myfile.txt
➜  ~ aws s3 ls s3://ccp-mybucket
2021-05-14 03:22:52          0 myfile.txt
➜  ~
```

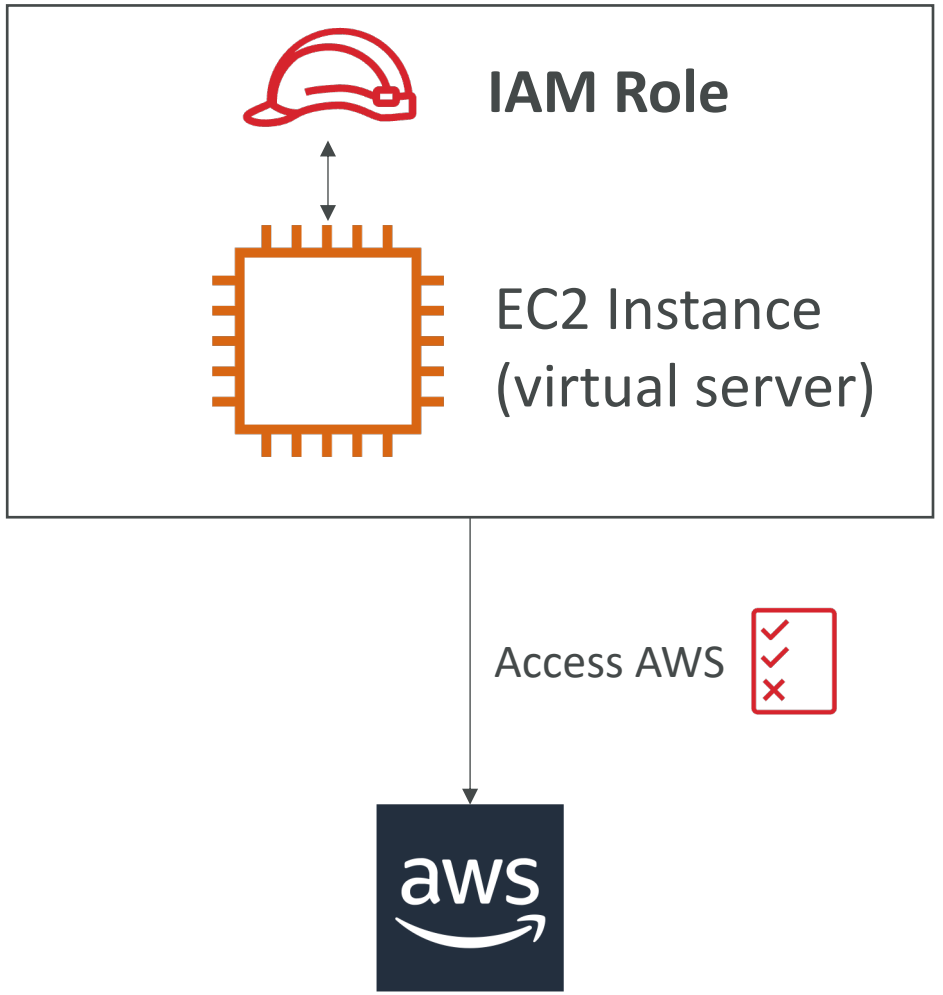
What's the AWS SDK?

- AWS Software Development Kit (AWS SDK)
- Language-specific APIs (set of libraries)
- Enables you to access and manage AWS services programmatically
- Embedded within your application
- Supports
  - SDKs (JavaScript, Python, PHP, .NET, Ruby, Java, Go, Node.js, C++)
  - Mobile SDKs (Android, iOS, …)
  - IoT Device SDKs (Embedded C, Arduino, …)
- Example: AWS CLI is built on AWS SDK for Python

Your Application

IAM Roles for Services

- Some AWS service will need to perform actions on your behalf
- To do so, we will assign **permissions** to AWS services with **IAM Roles**
- Common roles:
  - EC2 Instance Roles
  - Lambda Function Roles
  - Roles for CloudFormation



IAM Security Tools

- IAM Credentials Report (account-level)
  - a report that lists all your account's users and the status of their various credentials

- IAM Access Advisor (user-level)
  - Access advisor shows the service permissions granted to a user and when those services were last accessed.
  - You can use this information to revise your policies.

IAM Guidelines & Best Practices

- Don't use the root account except for AWS account setup
- One physical user = One AWS **user**
- **Assign users to groups** and assign permissions to groups
- Create a **strong password policy**
- Use and enforce the use of **Multi Factor Authentication (MFA)**
- Create and use **Roles** for giving permissions to AWS services
- Use Access Keys for Programmatic Access (CLI / SDK)
- Audit permissions of your account using IAM Credentials Report & IAM Access Advisor
- Never share IAM users & Access Keys

Shared Responsibility Model for IAM

- Infrastructure (global network security)
- Configuration and vulnerability analysis
- Compliance validation

You

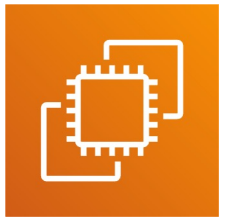
- Users, Groups, Roles, Policies management and monitoring
- Enable MFA on all accounts
- Rotate all your keys often
- Use IAM tools to apply appropriate permissions
- Analyze access patterns & review permissions

IAM Section – Summary

- **Users:** mapped to a physical user, has a password for AWS Console
- **Groups:** contains users only
- **Policies:** JSON document that outlines permissions for users or groups
- **Roles:** for EC2 instances or AWS services
- **Security:** MFA + Password Policy
- **AWS CLI:** manage your AWS services using the command-line
- **AWS SDK:** manage your AWS services using a programming language
- **Access Keys:** access AWS using the CLI or SDK
- **Audit:** IAM Credential Reports & IAM Access Advisor

Amazon EC2 – Basics

Amazon EC2



- EC2 is one of the most popular of AWS' offering
- EC2 = Elastic Compute Cloud = Infrastructure as a Service
- It mainly consists in the capability of :
  - Renting virtual machines (EC2)
  - Storing data on virtual drives (EBS)
  - Distributing load across machines (ELB)
  - Scaling the services using an auto-scaling group (ASG)

- Knowing EC2 is fundamental to understand how the Cloud works

EC2 sizing & configuration options

- Operating System (**OS**): Linux, Windows or Mac OS
- How much compute power & cores (**CPU**)
- How much random-access memory (**RAM**)
- How much storage space:
  - Network-attached (**EBS & EFS**)
  - hardware (**EC2 Instance Store**)
- Network card: speed of the card, Public IP address
- Firewall rules: **security group**
- Bootstrap script (configure at first launch): EC2 User Data

EC2 User Data

- It is possible to bootstrap our instances using an EC2 User data script.
- bootstrapping means launching commands when a machine starts
- That script is only run once at the instance first start
- EC2 user data is used to automate boot tasks such as:
  - Installing updates
  - Installing software
  - Downloading common files from the internet
  - Anything you can think of
- The EC2 User Data Script runs with the root user

Hands-On:
Launching an EC2 Instance running Linux

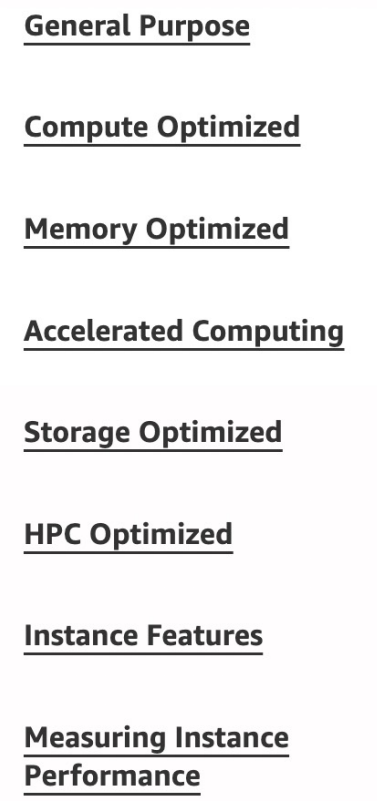
- We'll be launching our first virtual server using the AWS Console
- We'll get a first high-level approach to the various parameters
- We'll see that our web server is launched using EC2 user data
- We'll learn how to start / stop / terminate our instance.

EC2 Instance Types - Overview

- You can use different types of EC2 instances that are optimised for different use cases (https://aws.amazon.com/ec2/instance-types/)
- AWS has the following naming convention:

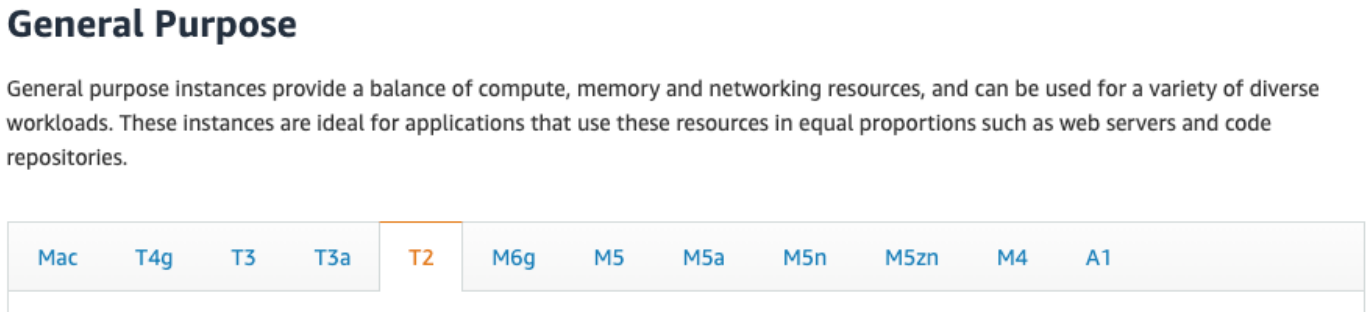
m5.2xlarge

- m: instance class
- 5: generation (AWS improves them over time)
- 2xlarge: size within the instance class



EC2 Instance Types – General Purpose

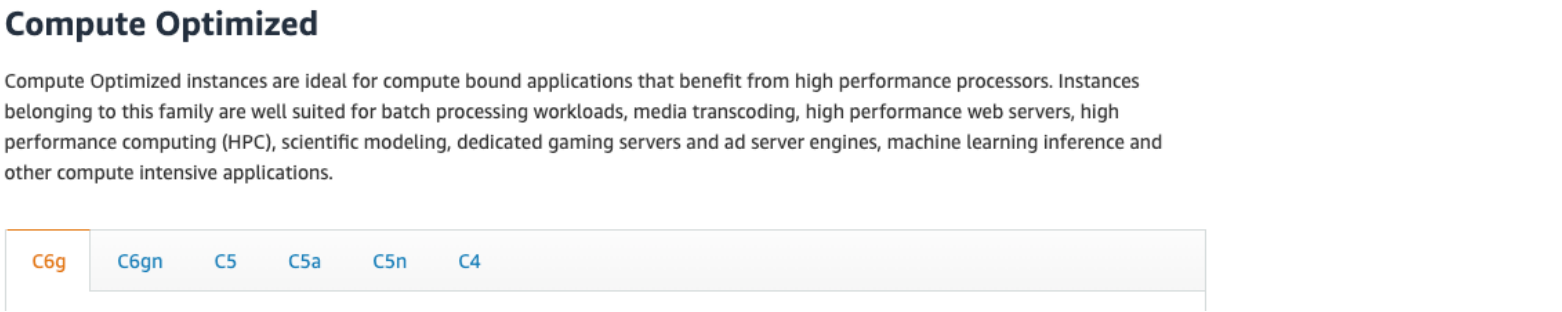
- Great for a diversity of workloads such as web servers or code repositories
- Balance between:
  - Compute
  - Memory
  - Networking
- In the course, we will be using the t2.micro which is a General Purpose EC2 instance



** this list will evolve over time, please check the AWS website for the latest information*

EC2 Instance Types – Compute Optimized

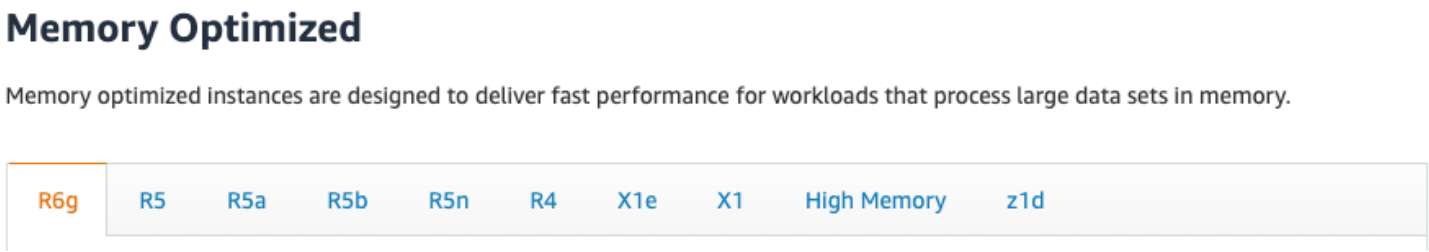
- Great for compute-intensive tasks that require high performance processors:
  - Batch processing workloads
  - Media transcoding
  - High performance web servers
  - High performance computing (HPC)
  - Scientific modeling & machine learning
  - Dedicated gaming servers



** this list will evolve over time, please check the AWS website for the latest information*

EC2 Instance Types – Memory Optimized

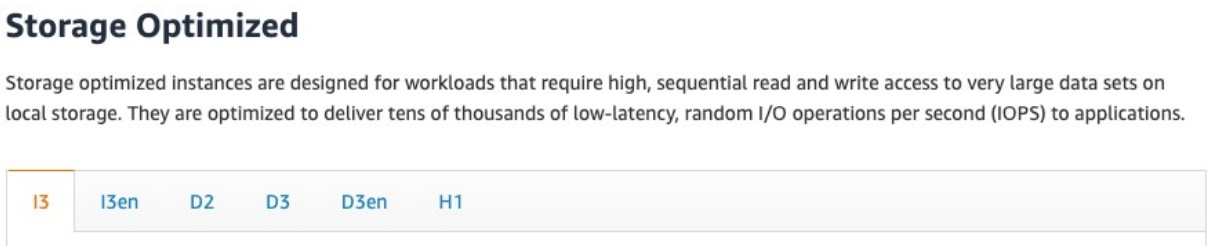
- Fast performance for workloads that process large data sets in memory
- Use cases:
  - High performance, relational/non-relational databases
  - Distributed web scale cache stores
  - In-memory databases optimized for BI (business intelligence)
  - Applications performing real-time processing of big unstructured data



** this list will evolve over time, please check the AWS website for the latest information*

EC2 Instance Types – Storage Optimized

- Great for storage-intensive tasks that require high, sequential read and write access to large data sets on local storage
- Use cases:
  - High frequency online transaction processing (OLTP) systems
  - Relational & NoSQL databases
  - Cache for in-memory databases (for example, Redis)
  - Data warehousing applications
  - Distributed file systems



** this list will evolve over time, please check the AWS website for the latest information*

EC2 Instance Types: example

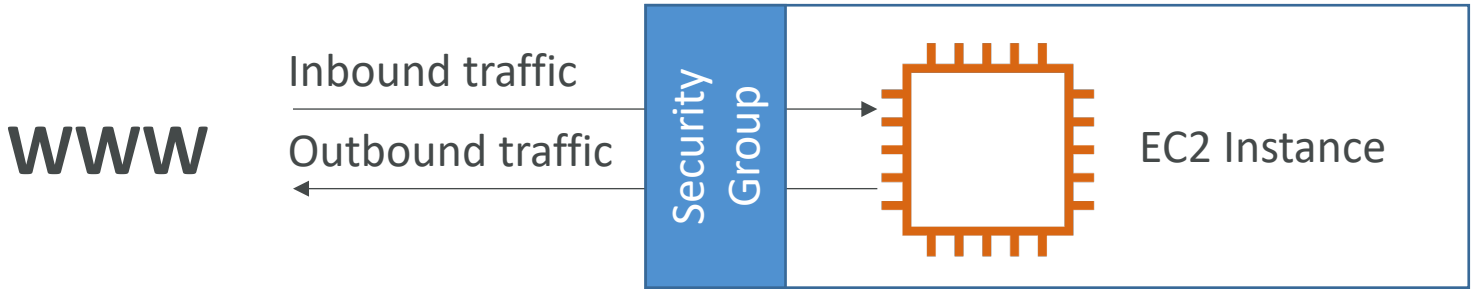
Instance	vCPU	Mem (GiB)	Storage	Network Performance	EBS Bandwidth (Mbps)
t2.micro	1	1	EBS-Only	Low to Moderate	
t2.xlarge	4	16	EBS-Only	Moderate	
c5d.4xlarge	16	32	1 x 400 NVMe SSD	Up to 10 Gbps	4,750
r5.16xlarge	64	512	EBS Only	20 Gbps	13,600
m5.8xlarge	32	128	EBS Only	10 Gbps	6,800

t2.micro is part of the AWS free tier (up to 750 hours per month)

Great website: https://instances.vantage.sh

Introduction to Security Groups

- Security Groups are the fundamental of network security in AWS
- They control how traffic is allowed into or out of our EC2 Instances.



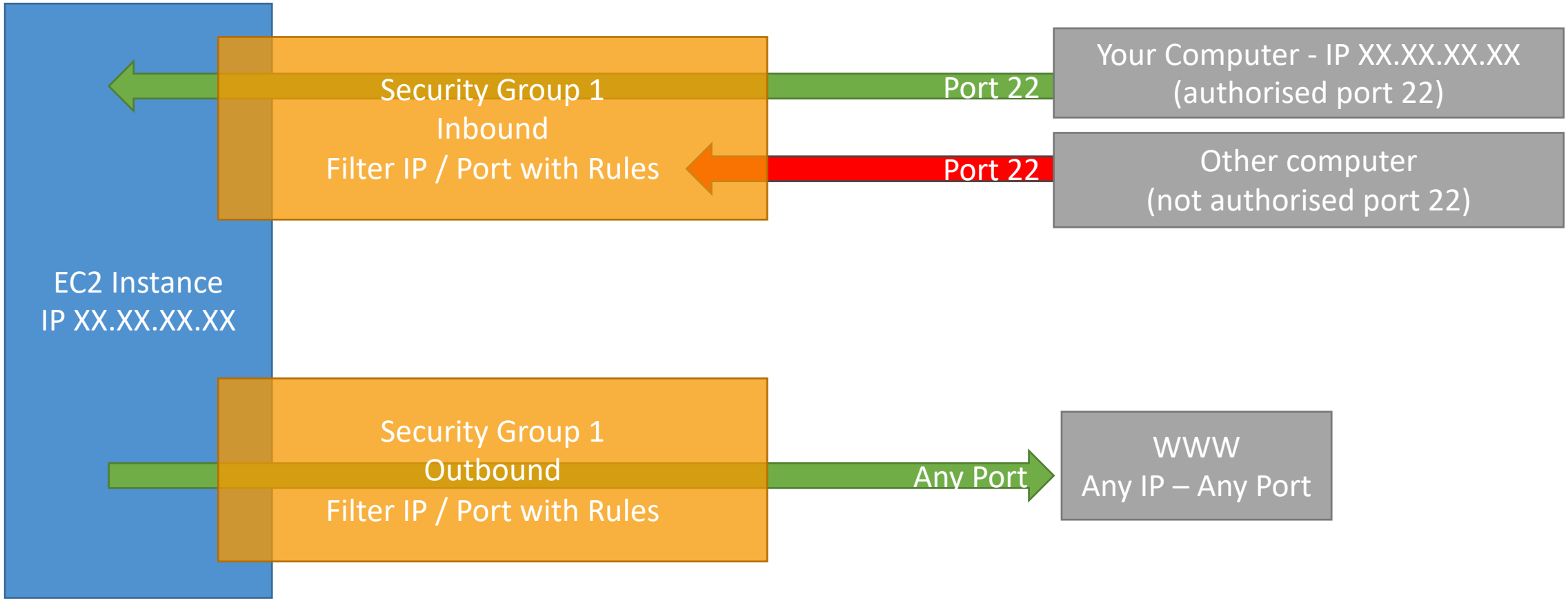
- Security groups only contain allow rules
- Security groups rules can reference by IP or by security group

Security Groups Deeper Dive

- Security groups are acting as a "firewall" on EC2 instances
- They regulate:
  - Access to Ports
  - Authorised IP ranges – IPv4 and IPv6
  - Control of inbound network (from other to the instance)
  - Control of outbound network (from the instance to other)

Type	Protocol	Port Range	Source	Description
HTTP	TCP	80	0.0.0.0/0	test http page
SSH	TCP	22	122.149.196.85/32	
Custom TCP Rule	TCP	4567	0.0.0.0/0	java app

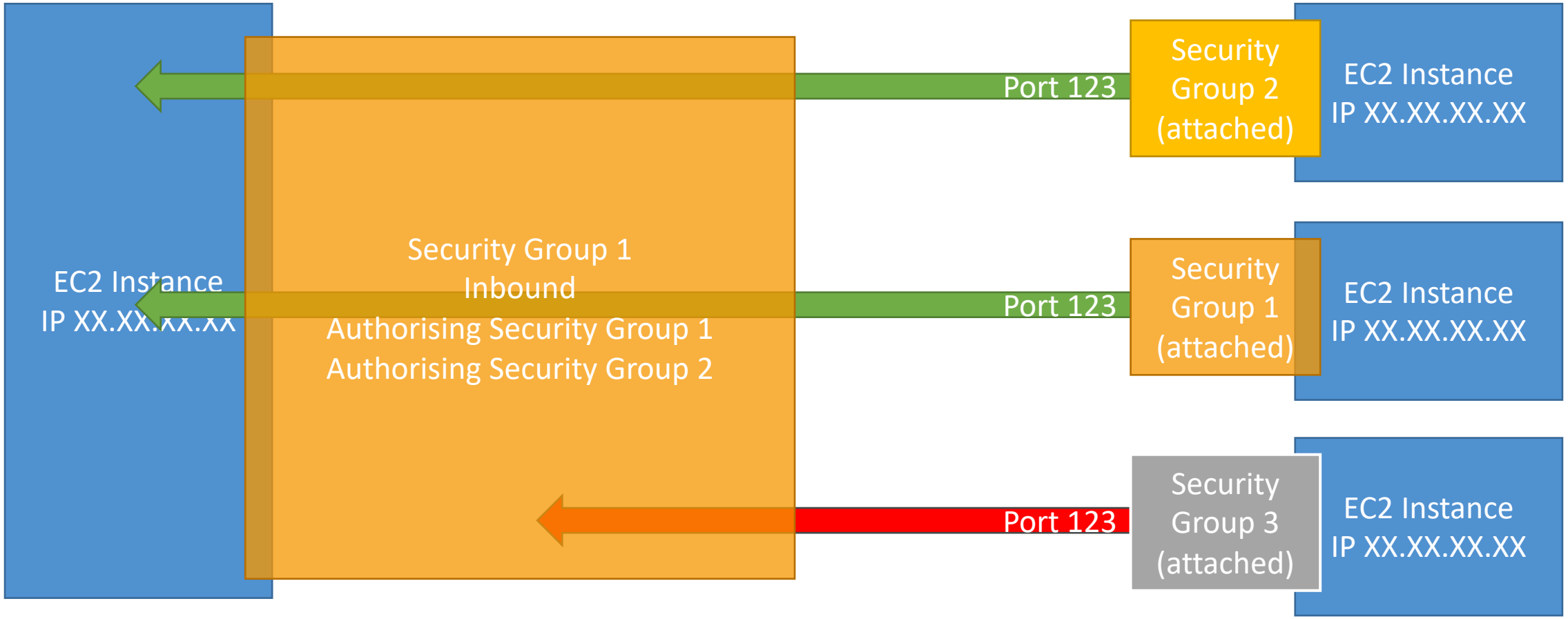
Security Groups Diagram



Security Groups
Good to know

- Can be attached to multiple instances
- Locked down to a region / VPC combination
- Does live "outside" the EC2 – if traffic is blocked the EC2 instance won't see it
- It's good to maintain one separate security group for SSH access
- If your application is not accessible (time out), then it's a security group issue
- If your application gives a "connection refused" error, then it's an application error or it's not launched
- All inbound traffic is blocked by default
- All outbound traffic is authorised by default

Referencing other security groups Diagram



Classic Ports to know

- 22 = SSH (Secure Shell) - log into a Linux instance
- 21 = FTP (File Transfer Protocol) – upload files into a file share
- 22 = SFTP (Secure File Transfer Protocol) – upload files using SSH
- 80 = HTTP – access unsecured websites
- 443 = HTTPS – access secured websites
- 3389 = RDP (Remote Desktop Protocol) – log into a Windows instance

SSH Summary Table

	SSH	Putty	EC2 Instance Connect
Mac	✓		✓
Linux	✓		✓
Windows < 10		✓	✓
Windows >= 10	✓	✓	✓

Which Lectures to watch

- Mac / Linux:
  - SSH on Mac/Linux lecture
- Windows:
  - Putty Lecture
  - If Windows 10: SSH on Windows 10 lecture
- All:
  - EC2 Instance Connect lecture

SSH troubleshooting

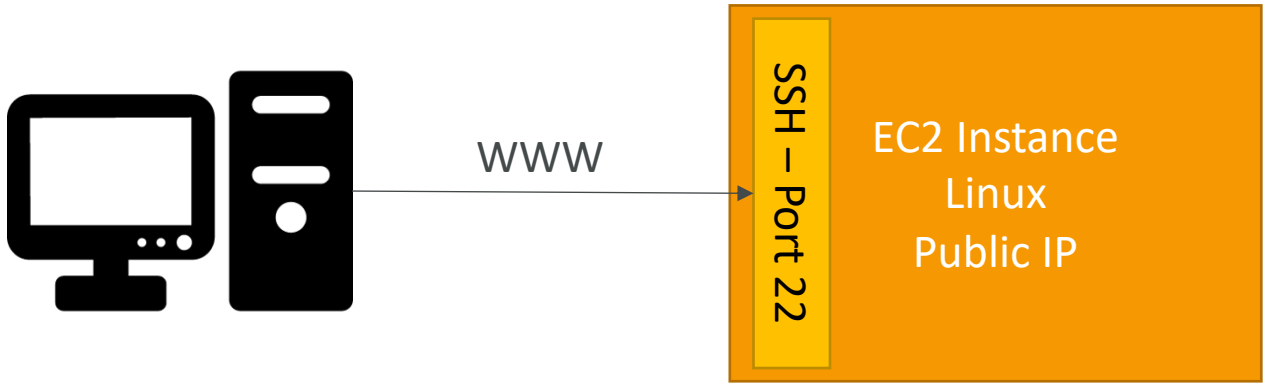
- Students have the most problems with SSH

- If things don't work…
  1. Re-watch the lecture. You may have missed something
  2. Read the troubleshooting guide
  3. Try EC2 Instance Connect

- If one method works (SSH, Putty or EC2 Instance Connect) you're good
- If no method works, that's okay, the course won't use SSH much

How to SSH into your EC2 Instance
Linux / Mac OS X

- We'll learn how to SSH into your EC2 instance using Linux / Mac
- SSH is one of the most important function. It allows you to control a remote machine, all using the command line.

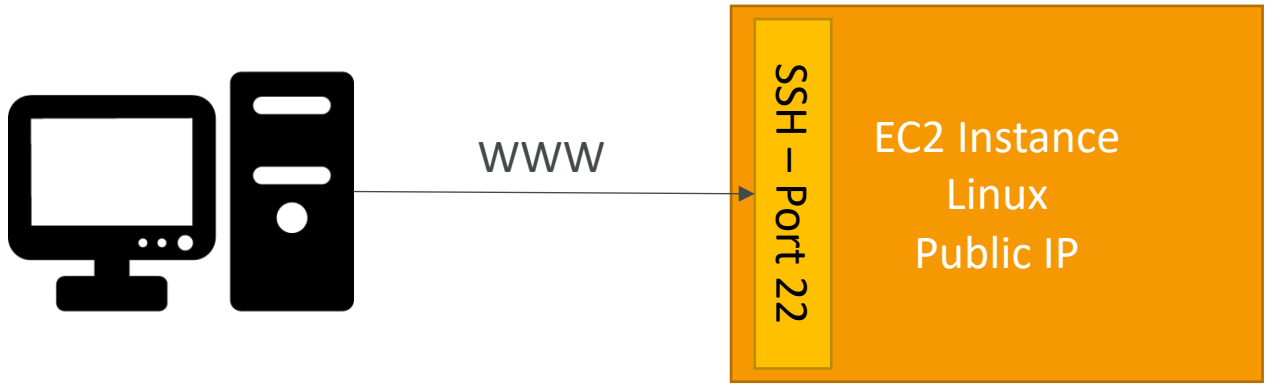


- We will see how we can configure OpenSSH ~/.ssh/config to facilitate the SSH into our EC2 instances

How to SSH into your EC2 Instance

Windows

- We'll learn how to SSH into your EC2 instance using Windows
- SSH is one of the most important function. It allows you to control a remote machine, all using the command line.



- We will configure all the required parameters necessary for doing SSH on Windows using the free tool Putty.

EC2 Instance Connect

- Connect to your EC2 instance within your browser
- No need to use your key file that was downloaded
- The "magic" is that a temporary key is uploaded onto EC2 by AWS

- Works only out-of-the-box with Amazon Linux 2

- Need to make sure the port 22 is still opened!

EC2 Instances Purchasing Options

- **On-Demand Instances** – short workload, predictable pricing, pay by second
- **Reserved** (1 & 3 years)
  - **Reserved Instances** – long workloads
  - **Convertible Reserved Instances** – long workloads with flexible instances
- **Savings Plans** (1 & 3 years) –commitment to an amount of usage, long workload
- **Spot Instances** – short workloads, cheap, can lose instances (less reliable)
- **Dedicated Hosts** – book an entire physical server, control instance placement
- **Dedicated Instances** – no other customers will share your hardware
- **Capacity Reservations** – reserve capacity in a specific AZ for any duration

EC2 On Demand

- Pay for what you use:
  - Linux or Windows - billing per second, after the first minute
  - All other operating systems - billing per hour
- Has the highest cost but no upfront payment
- No long-term commitment

- Recommended for **short-term** and **un-interrupted workloads**, where you can't predict how the application will behave

EC2 Reserved Instances

- Up to 72% discount compared to On-demand
- You reserve a specific instance attributes **(Instance Type, Region, Tenancy, OS)**
- **Reservation Period** – **1 year** (+discount) or **3 years** (+++discount)
- **Payment Options** – **No Upfront** (+), **Partial Upfront** (++), **All Upfront** (+++)
- **Reserved Instance's Scope** – **Regional** or **Zonal** (reserve capacity in an AZ)
- Recommended for steady-state usage applications (think database)
- You can buy and sell in the Reserved Instance Marketplace

- **Convertible Reserved Instance**
  - Can change the EC2 instance type, instance family, OS, scope and tenancy
  - Up to 66% discount

Note: the % discounts are different from the video as AWS change them over time – the exact numbers are not needed for the exam. This is just for illustrative purposes ☺

EC2 Savings Plans

- Get a discount based on long-term usage (up to 72% - same as RIs)
- Commit to a certain type of usage ($10/hour for 1 or 3 years)
- Usage beyond EC2 Savings Plans is billed at the On-Demand price

- Locked to a specific instance family & AWS region (e.g., M5 in us-east-1)
- Flexible across:
  - Instance Size (e.g., m5.xlarge, m5.2xlarge)
  - OS (e.g., Linux, Windows)
  - Tenancy (Host, Dedicated, Default)

EC2 Spot Instances

- Can get a **discount of up to 90%** compared to On-demand
- Instances that you can "lose" at any point of time if your max price is less than the current spot price
- The **MOST cost-efficient** instances in AWS

- Useful for workloads that are resilient to failure
  - Batch jobs
  - Data analysis
  - Image processing
  - Any **distributed** workloads
  - Workloads with a flexible start and end time

- Not suitable for critical jobs or databases

EC2 Dedicated Hosts

- A physical server with EC2 instance capacity fully dedicated to your use
- Allows you address **compliance requirements** and **use your existing server-bound software licenses** (per-socket, per-core, pe—VM software licenses)
- Purchasing Options:
  - **On-demand** – pay per second for active Dedicated Host
  - **Reserved** - 1 or 3 years (No Upfront, Partial Upfront, All Upfront)
- The most expensive option

- Useful for software that have complicated licensing model (BYOL – Bring Your Own License)
- Or for companies that have strong regulatory or compliance needs

EC2 Dedicated Instances

- Instances run on hardware that's dedicated to you
- May share hardware with other instances in same account
- No control over instance placement (can move hardware after Stop / Start)

Characteristic	Dedicated Instances	Dedicated Hosts
Enables the use of dedicated physical servers	X	X
Per instance billing (subject to a $2 per region fee)	X	
Per host billing		X
Visibility of sockets, cores, host ID		X
Affinity between a host and instance		X
Targeted instance placement		X
Automatic instance placement	X	X
Add capacity using an allocation request		X

EC2 Capacity Reservations

- Reserve **On-Demand** instances capacity in a specific AZ for any duration
- You always have access to EC2 capacity when you need it
- **No time commitment** (create/cancel anytime), **no billing discounts**
- Combine with Regional Reserved Instances and Savings Plans to benefit from billing discounts
- You're charged at On-Demand rate whether you run instances or not

- Suitable for short-term, uninterrupted workloads that needs to be in a specific AZ

Which purchasing option is right for me?

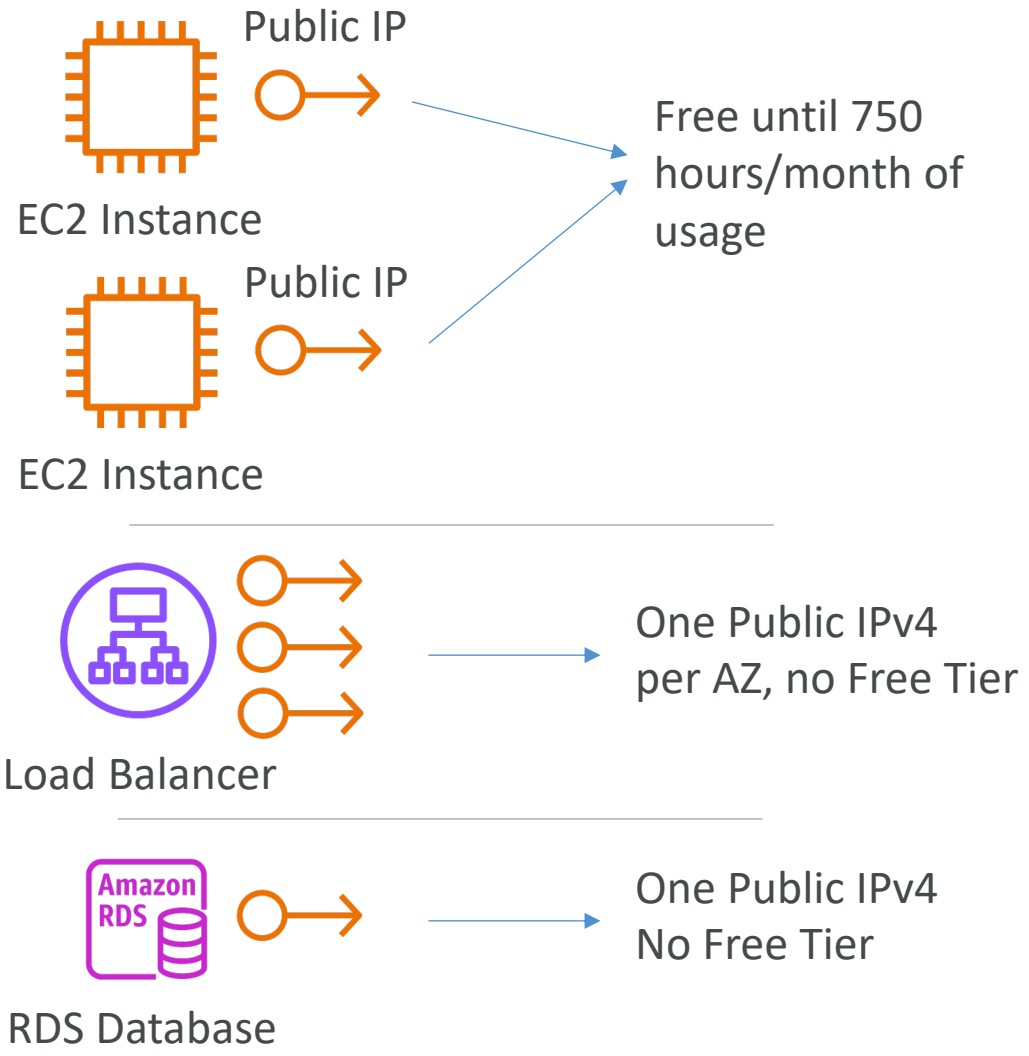
- **On demand:** coming and staying in resort whenever we like, we pay the full price
- **Reserved:** like planning ahead and if we plan to stay for a long time, we may get a good discount.
- **Savings Plans:** pay a certain amount per hour for certain period and stay in any room type (e.g., King, Suite, Sea View, …)
- **Spot instances:** the hotel allows people to bid for the empty rooms and the highest bidder keeps the rooms. You can get kicked out at any time
- **Dedicated Hosts:** We book an entire building of the resort
- **Capacity Reservations:** you book a room for a period with full price even you don't stay in it

Price Comparison
Example – m4.large – us-east-1

Price Type	Price (per hour)
On-Demand	$0.10
Spot Instance (Spot Price)	$0.038 - $0.039 (up to 61% off)
Reserved Instance (1 year)	$0.062 (No Upfront) - $0.058 (All Upfront)
Reserved Instance (3 years)	$0.043 (No Upfront) - $0.037 (All Upfront)
EC2 Savings Plan (1 year)	$0.062 (No Upfront) - $0.058 (All Upfront)
Reserved **Convertible** Instance (1 year)	$0.071 (No Upfront) - $0.066 (All Upfront)
Dedicated Host	On-Demand Price
Dedicated Host Reservation	Up to 70% off
Capacity Reservations	On-Demand Price

AWS charges for IPv4 addresses

- Starting February 1st 2024, there's a charge for all Public IPv4 created in your account
- **$0.005 per hour of Public IPv4 (~ $3.6 per month)**
- For new accounts in AWS, you have a free tier for the **EC2 service**: 750 hours of Public IPv4 per month for the first 12 months
- **For all other services there is no free tier**



AWS charges for IPv4 addresses

- What about IPv6?
  - Unfortunately, many Internet Service Provider (ISP) around the world don't support IPv6, so the course would not work for some of you
  - You can test IPv6 by going to https://test-ipv6.com/
  - If you use IPv6 in this course, you're on your own (security groups, networking…) but you can do it!
- How to troubleshoot charges?
  - Go into your AWS Bill
  - Look into the AWS Public IP Insights service
  - Nice article here: https://repost.aws/articles/ARknH_OR0cTvqoTfJrVGaB8A/why-am-i-seeing-charges-for-public-ipv4-addresses-when-i-am-under-the-aws-free-tier

Amazon EC2 – Instance Storage

What's an EBS Volume?

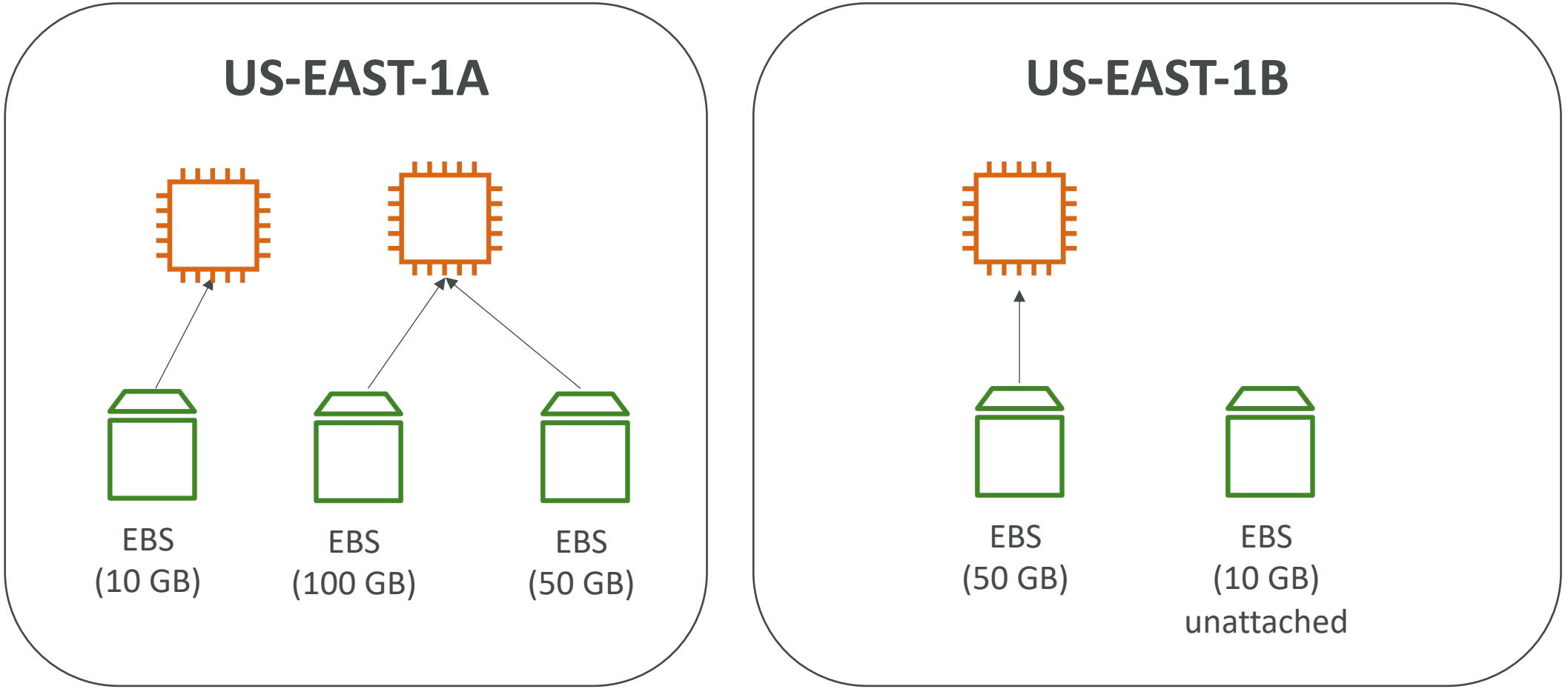
- An EBS (Elastic Block Store) Volume is a network drive you can attach to your instances while they run
- It allows your instances to persist data, even after their termination
- **They can only be mounted to one instance at a time** (at the CCP level)
- They are bound to **a specific availability zone**

- Analogy: Think of them as a "network USB stick"
- Free tier: 30 GB of free EBS storage of type General Purpose (SSD) or Magnetic per month

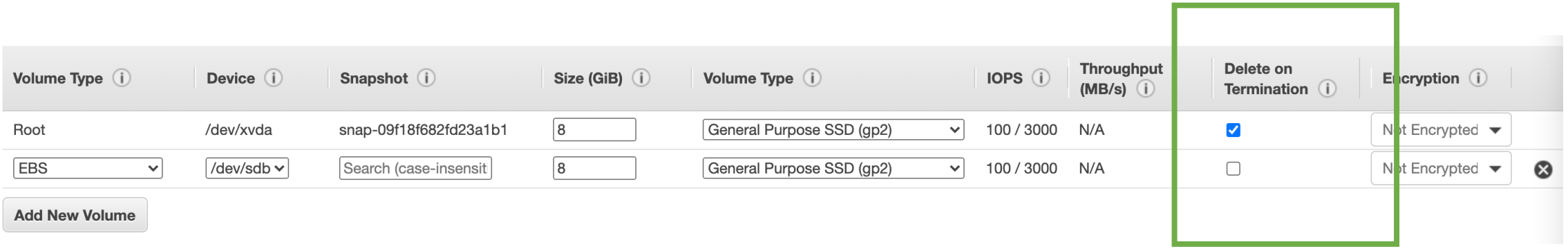
EBS Volume

- It's a network drive (i.e. not a physical drive)
  - It uses the network to communicate the instance, which means there might be a bit of latency
  - It can be detached from an EC2 instance and attached to another one quickly
- It's locked to an Availability Zone (AZ)
  - An EBS Volume in us-east-1a cannot be attached to us-east-1b
  - To move a volume across, you first need to snapshot it
- Have a provisioned capacity (size in GBs, and IOPS)
  - You get billed for all the provisioned capacity
  - You can increase the capacity of the drive over time

EBS Volume - Example



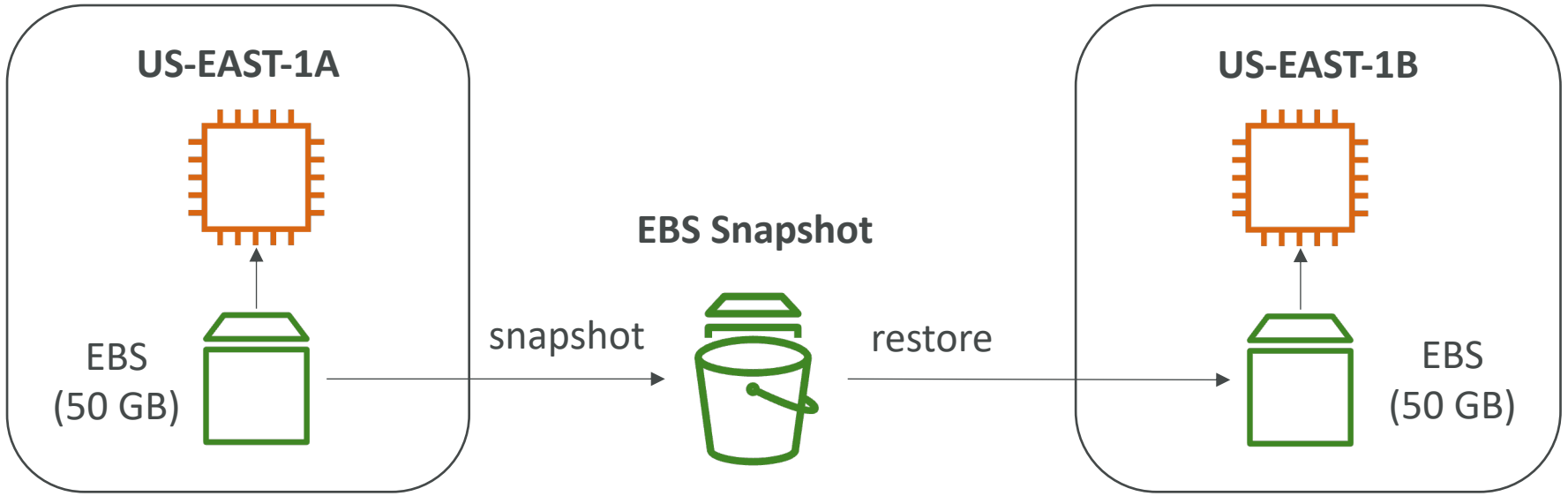
EBS – Delete on Termination attribute



- Controls the EBS behaviour when an EC2 instance terminates
  - By default, the root EBS volume is deleted (attribute enabled)
  - By default, any other attached EBS volume is not deleted (attribute disabled)
- This can be controlled by the AWS console / AWS CLI
- Use case: preserve root volume when instance is terminated

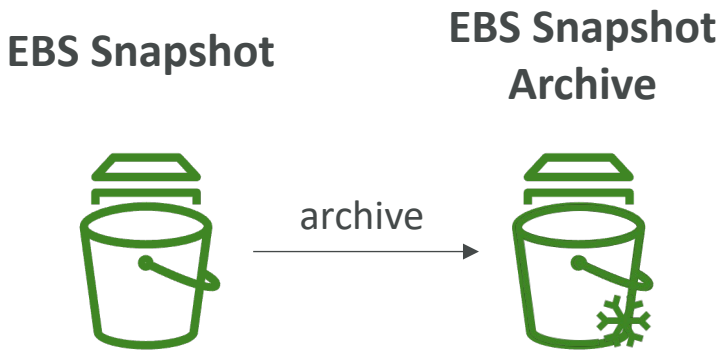
EBS Snapshots

- Make a backup (snapshot) of your EBS volume at a point in time
- Not necessary to detach volume to do snapshot, but recommended
- Can copy snapshots across AZ or Region

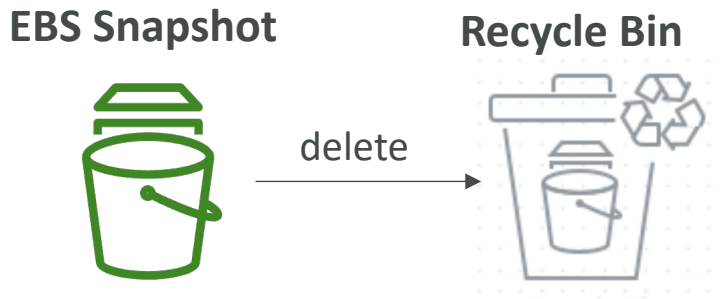


EBS Snapshots Features

- EBS Snapshot Archive
  - Move a Snapshot to an "archive tier" that is 75% cheaper
  - Takes within 24 to 72 hours for restoring the archive



- Recycle Bin for EBS Snapshots
  - Setup rules to retain deleted snapshots so you can recover them after an accidental deletion
  - Specify retention (from 1 day to 1 year)



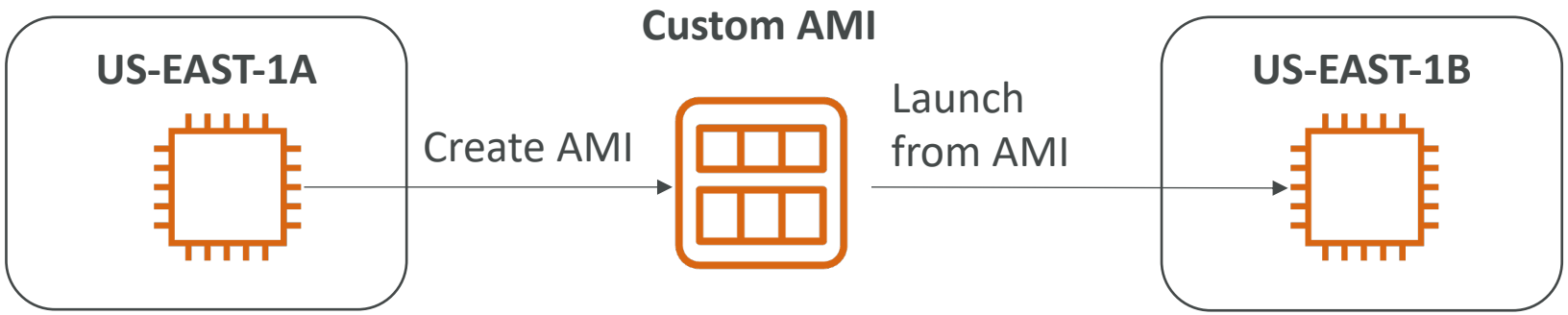
- Fast Snapshot Restore (FSR)
  - Force full initialization of snapshot to have no latency on the first use ($$$)

AMI Overview

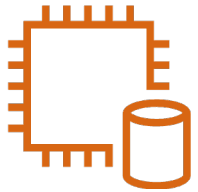
- AMI = Amazon Machine Image
- AMI are a **customization** of an EC2 instance
  - You add your own software, configuration, operating system, monitoring…
  - Faster boot / configuration time because all your software is pre-packaged
- AMI are built for a **specific region** (and can be copied across regions)
- You can launch EC2 instances from:
  - **A Public AMI:** AWS provided
  - **Your own AMI:** you make and maintain them yourself
  - **An AWS Marketplace AMI:** an AMI someone else made (and potentially sells)

AMI Process (from an EC2 instance)

- Start an EC2 instance and customize it
- Stop the instance (for data integrity)
- Build an AMI – this will also create EBS snapshots
- Launch instances from other AMIs



EC2 Instance Store



- EBS volumes are **network drives** with good but "limited" performance
- If you need a high-performance hardware disk, use EC2 Instance Store

- Better I/O performance
- EC2 Instance Store lose their storage if they're stopped (ephemeral)
- Good for buffer / cache / scratch data / temporary content
- Risk of data loss if hardware fails
- Backups and Replication are your responsibility

Local EC2 Instance Store

Very high IOPS

Instance Size	100% Random Read IOPS	Write IOPS
i3.large *	100,125	35,000
i3.xlarge *	206,250	70,000
i3.2xlarge	412,500	180,000
i3.4xlarge	825,000	360,000
i3.8xlarge	1.65 million	720,000
i3.16xlarge	3.3 million	1.4 million
i3.metal	3.3 million	1.4 million
i3en.large *	42,500	32,500
i3en.xlarge *	85,000	65,000
i3en.2xlarge *	170,000	130,000
i3en.3xlarge	250,000	200,000
i3en.6xlarge	500,000	400,000
i3en.12xlarge	1 million	800,000
i3en.24xlarge	2 million	1.6 million
i3en.metal	2 million	1.6 million

EBS Volume Types

- EBS Volumes come in 6 types
  - gp2 / gp3 (SSD): General purpose SSD volume that balances price and performance for a wide variety of workloads
  - io1 / io2 Block Express (SSD): Highest-performance SSD volume for mission-critical low-latency or high-throughput workloads
  - st1 (HDD): Low cost HDD volume designed for frequently accessed, throughput-intensive workloads
  - sc1 (HDD): Lowest cost HDD volume designed for less frequently accessed workloads
- EBS Volumes are characterized in Size | Throughput | IOPS (I/O Ops Per Sec)
- When in doubt always consult the AWS documentation – it's good!
- Only gp2/gp3 and io1/io2 Block Express can be used as boot volumes

EBS Volume Types Use cases

General Purpose SSD

- Cost effective storage, low-latency
- System boot volumes, Virtual desktops, Development and test environments
- 1 GiB - 16 TiB
- gp3:
  - Baseline of 3,000 IOPS and throughput of 125 MiB/s
  - Can increase IOPS up to 16,000 and throughput up to 1000 MiB/s independently
- gp2:
  - Small gp2 volumes can burst IOPS to 3,000
  - Size of the volume and IOPS are linked, max IOPS is 16,000
  - 3 IOPS per GB, means at 5,334 GB we are at the max IOPS

EBS Volume Types Use cases

Provisioned IOPS (PIOPS) SSD

- Critical business applications with sustained IOPS performance
- Or applications that need more than 16,000 IOPS
- Great for **databases workloads** (sensitive to storage perf and consistency)
- io1 (4 GiB - 16 TiB):
  - Max PIOPS: 64,000 for Nitro EC2 instances & 32,000 for other
  - Can increase PIOPS independently from storage size
- io2 Block Express (4 GiB – 64 TiB):
  - Sub-millisecond latency
  - Max PIOPS: 256,000 with an IOPS:GiB ratio of 1,000:1
- Supports EBS Multi-attach

EBS Volume Types Use cases

Hard Disk Drives (HDD)

- Cannot be a boot volume
- 125 GiB to 16 TiB
- Throughput Optimized HDD (st1)
  - Big Data, Data Warehouses, Log Processing
  - **Max throughput** 500 MiB/s – max IOPS 500
- Cold HDD (sc1):
  - For data that is infrequently accessed
  - Scenarios where lowest cost is important
  - **Max throughput** 250 MiB/s – max IOPS 250

EBS – Volume Types Summary

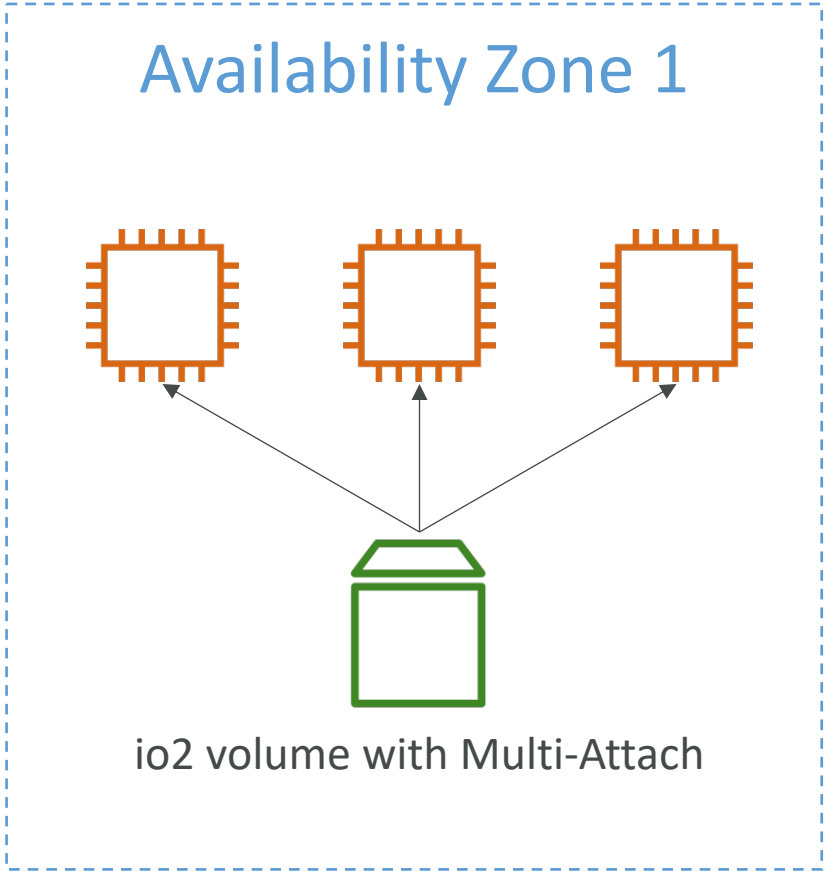
	General Purpose SSD volumes		Provisioned IOPS SSD volumes	
Volume type	gp3	gp2	io2 Block Express [3]	io1
Durability	99.8% - 99.9% durability (0.1% - 0.2% annual failure rate)		99.999% durability (0.001% annual failure rate)	99.8% - 99.9% durability (0.1% - 0.2% annual failure rate)
Use cases	• Transactional workloads • Virtual desktops • Medium-sized, single-instance databases • Low-latency interactive applications • Boot volumes • Development and test environments		Workloads that require: • Sub-millisecond latency • Sustained IOPS performance • More than 64,000 IOPS or 1,000 MiB/s of throughput	• Workloads that require sustained IOPS performance or more than 16,000 IOPS • I/O-intensive database workloads
Volume size	1 GiB - 16 TiB		4 GiB - 64 TiB [4]	4 GiB - 16 TiB
Max IOPS per volume (16 KiB I/O)	16,000		256,000 [5]	64,000
Max throughput per volume	1,000 MiB/s	250 MiB/s [1]	4,000 MiB/s	1,000 MiB/s [2]
Amazon EBS Multi-attach	Not supported		Supported	
NVMe reservations	Not supported		Supported	Not supported
Boot volume	Supported			

	Throughput Optimized HDD volumes	Cold HDD volumes
Volume type	st1	sc1
Durability	99.8% - 99.9% durability (0.1% - 0.2% annual failure rate)	
Use cases	• Big data • Data warehouses • Log processing	• Throughput-oriented storage for data that is infrequently accessed • Scenarios where the lowest storage cost is important
Volume size	125 GiB - 16 TiB	
Max IOPS per volume (1 MiB I/O)	500	250
Max throughput per volume	500 MiB/s	250 MiB/s
Amazon EBS Multi-attach	Not supported	
Boot volume	Not supported	

https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-volume-types.html#solid-state-drives

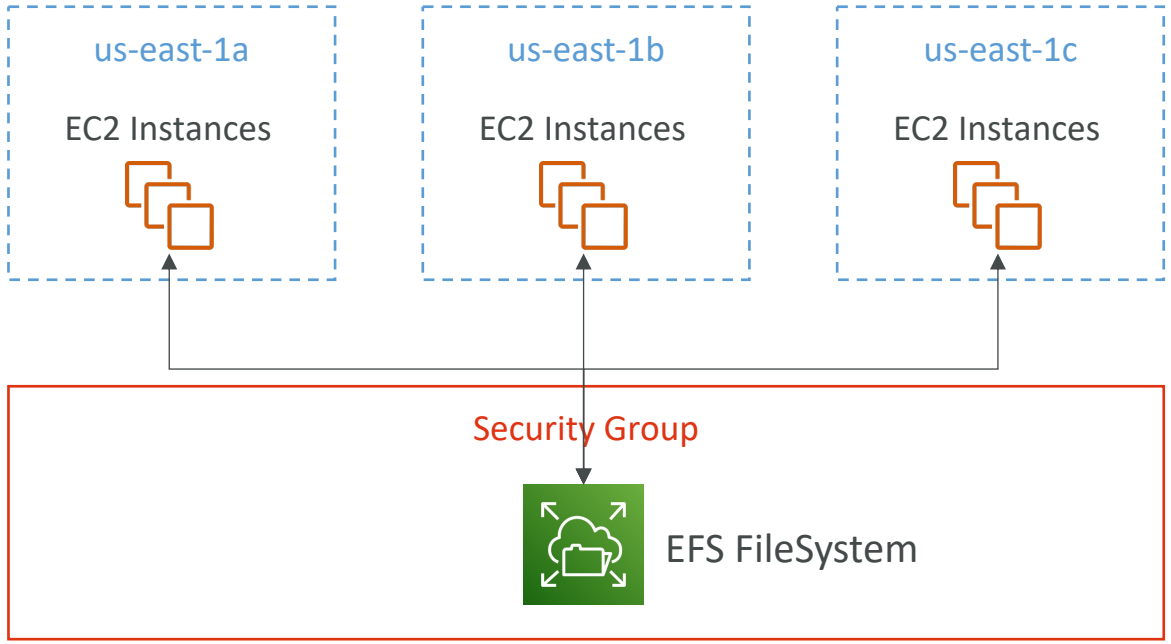
EBS Multi-Attach – io1/io2 family

- Attach the same EBS volume to multiple EC2 instances in the same AZ
- Each instance has full read & write permissions to the high-performance volume
- Use case:
  - Achieve **higher application availability** in clustered Linux applications (ex: Teradata)
  - Applications must manage concurrent write operations
- Up to 16 EC2 Instances at a time
- Must use a file system that's cluster-aware (not XFS, EXT4, etc…)



Amazon EFS – Elastic File System

- Managed NFS (network file system) that can be mounted on many EC2
- EFS works with EC2 instances in multi-AZ
- Highly available, scalable, expensive (3x gp2), pay per use



Amazon EFS – Elastic File System

- Use cases: content management, web serving, data sharing, Wordpress
- Uses NFSv4.1 protocol
- Uses security group to control access to EFS
- **Compatible with Linux based AMI (not Windows)**
- Encryption at rest using KMS

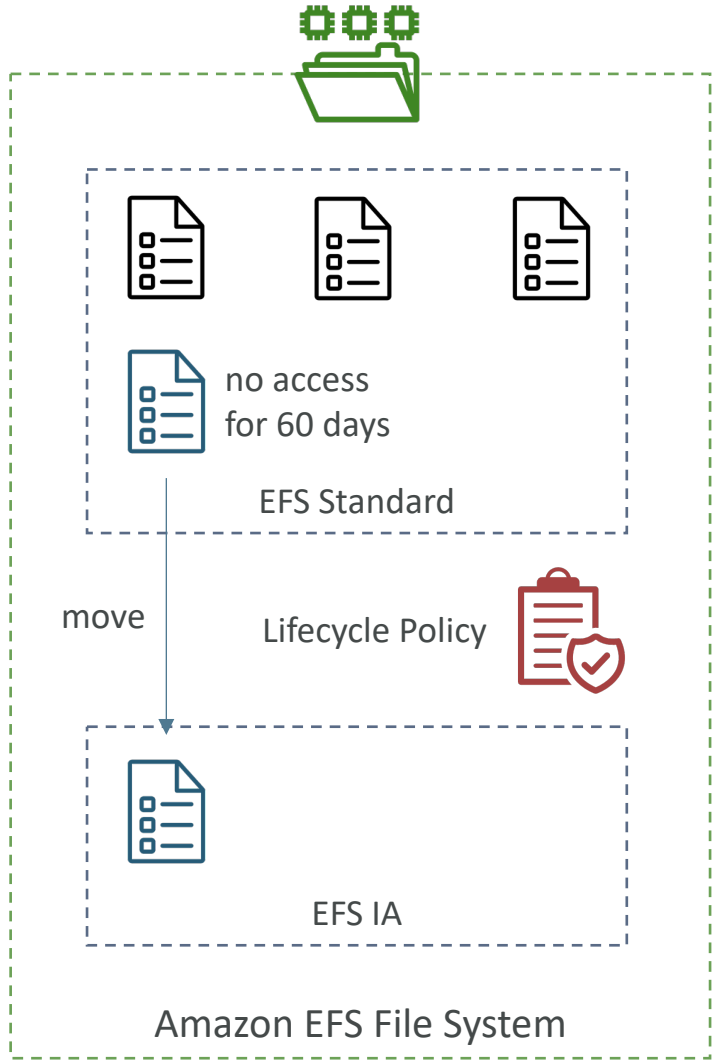
- POSIX file system (~Linux) that has a standard file API
- File system scales automatically, pay-per-use, no capacity planning!

EFS – Performance & Storage Classes

- EFS Scale
  - 1000s of concurrent NFS clients, 10 GB+ /s throughput
  - Grow to Petabyte-scale network file system, automatically
- Performance Mode (set at EFS creation time)
  - General Purpose (default) – latency-sensitive use cases (web server, CMS, etc…)
  - Max I/O – higher latency, throughput, highly parallel (big data, media processing)
- Throughput Mode
  - Bursting – 1 TB = 50MiB/s + burst of up to 100MiB/s
  - Provisioned – set your throughput regardless of storage size, ex: 1 GiB/s for 1 TB storage
  - Elastic – automatically scales throughput up or down based on your workloads
    - Up to 3GiB/s for reads and 1GiB/s for writes
    - Used for unpredictable workloads

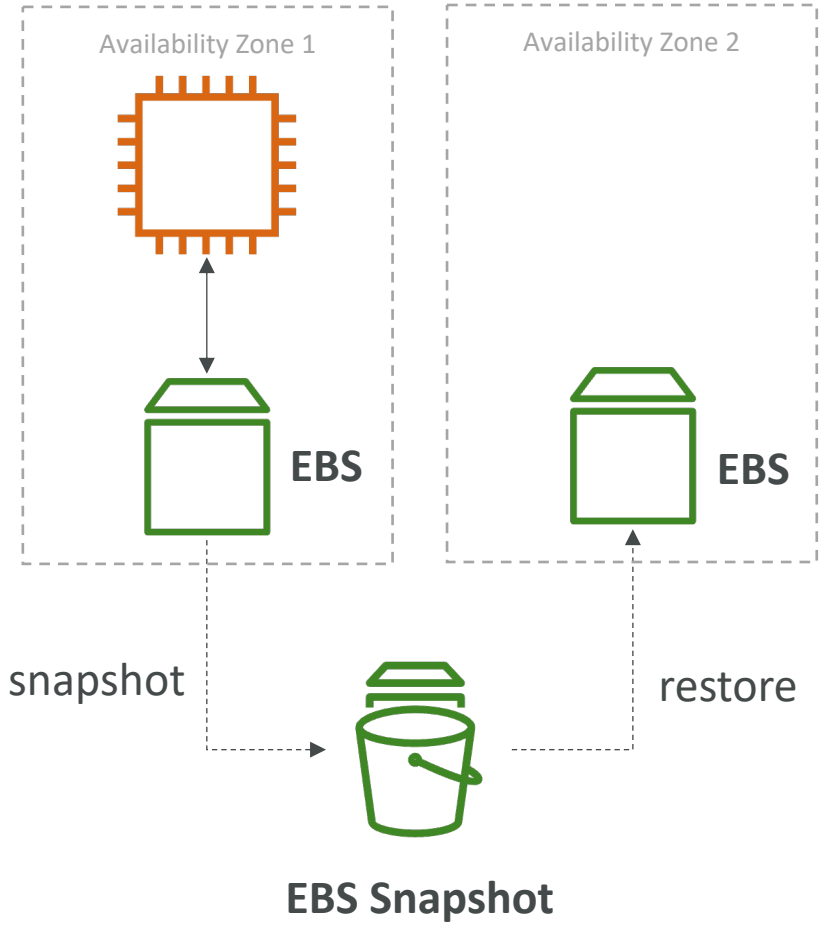
EFS – Storage Classes

- Storage Tiers (lifecycle management feature – move file after N days)
  - **Standard**: for frequently accessed files
  - **Infrequent access (EFS-IA):** cost to retrieve files, lower price to store.
  - **Archive:** rarely accessed data (few times each year), 50% cheaper
  - Implement **lifecycle policies** to move files between storage tiers
- Availability and durability
  - Standard: Multi-AZ, great for prod
  - One Zone: One AZ, great for dev, backup enabled by default, compatible with IA (EFS One Zone-IA)
- Over 90% in cost savings



EBS vs EFS – Elastic Block Storage

- EBS volumes…
  - one instance (except multi-attach io1/io2)
  - are locked at the Availability Zone (AZ) level
  - gp2: IO increases if the disk size increases
  - gp3 & io1: can increase IO independently
- To migrate an EBS volume across AZ
  - Take a snapshot
  - Restore the snapshot to another AZ
  - EBS backups use IO and you shouldn't run them while your application is handling a lot of traffic
- Root EBS Volumes of instances get terminated by default if the EC2 instance gets terminated. (you can disable that)

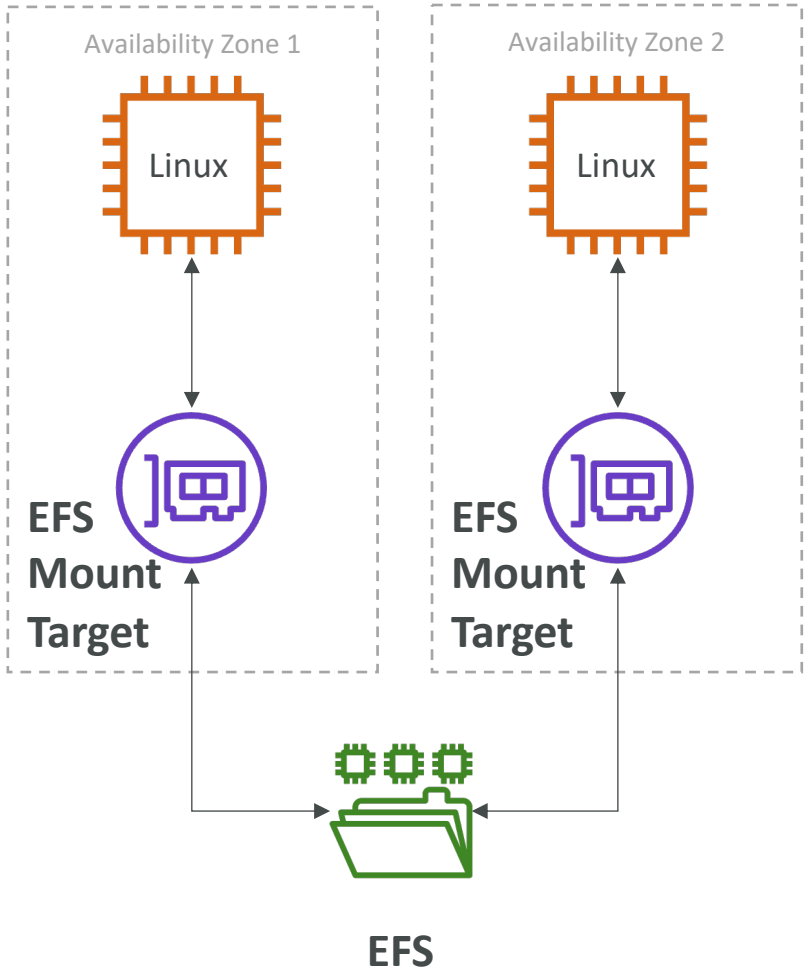


EBS vs EFS – Elastic File System

- Mounting 100s of instances across AZ
- EFS share website files (WordPress)
- Only for Linux Instances (POSIX)

- EFS has a higher price point than EBS
- Can leverage Storage Tiers for cost savings

- Remember: EFS vs EBS vs Instance Store



High Availability & Scalability

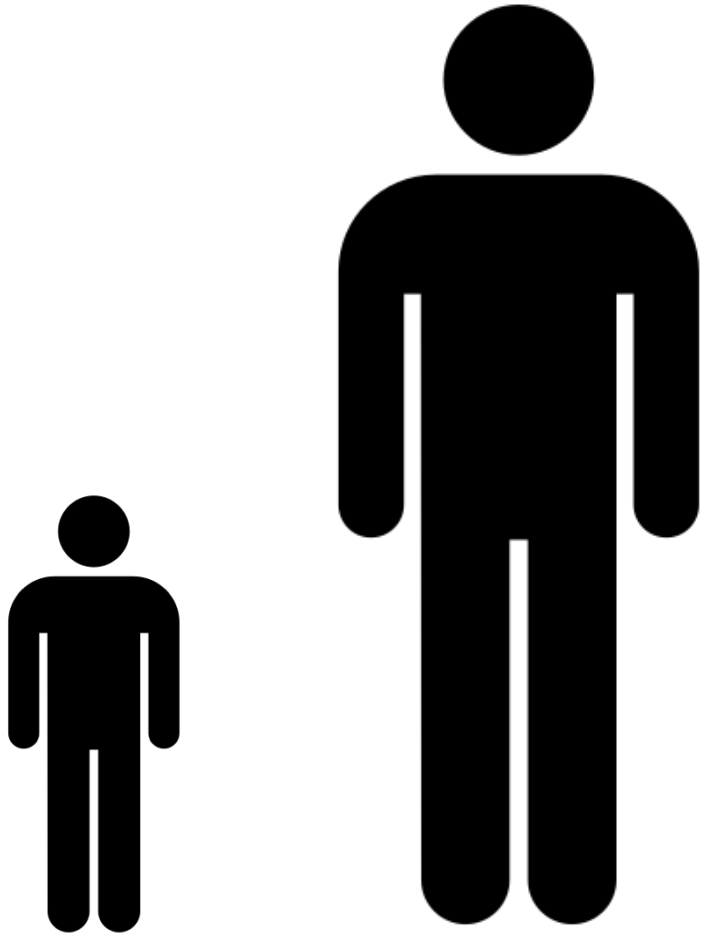
Scalability & High Availability

- Scalability means that an application / system can handle greater loads by adapting.
- There are two kinds of scalability:
  - Vertical Scalability
  - Horizontal Scalability (= elasticity)
- **Scalability is linked but different to High Availability**

- Let's deep dive into the distinction, using a call center as an example

Vertical Scalability

- Vertically scalability means increasing the size of the instance
- For example, your application runs on a t2.micro
- Scaling that application vertically means running it on a t2.large
- Vertical scalability is very common for non distributed systems, such as a database.
- RDS, ElastiCache are services that can scale vertically.
- There's usually a limit to how much you can vertically scale (hardware limit)



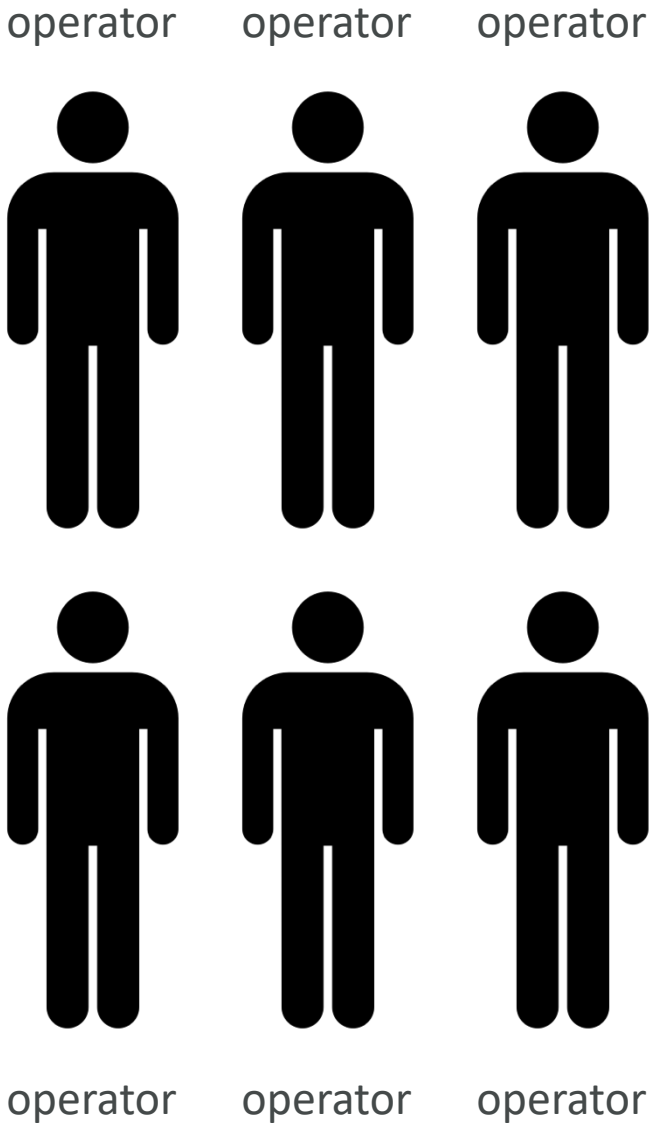
junior operator senior operator

Horizontal Scalability

- Horizontal Scalability means increasing the number of instances / systems for your application

- Horizontal scaling implies distributed systems.
- This is very common for web applications / modern applications

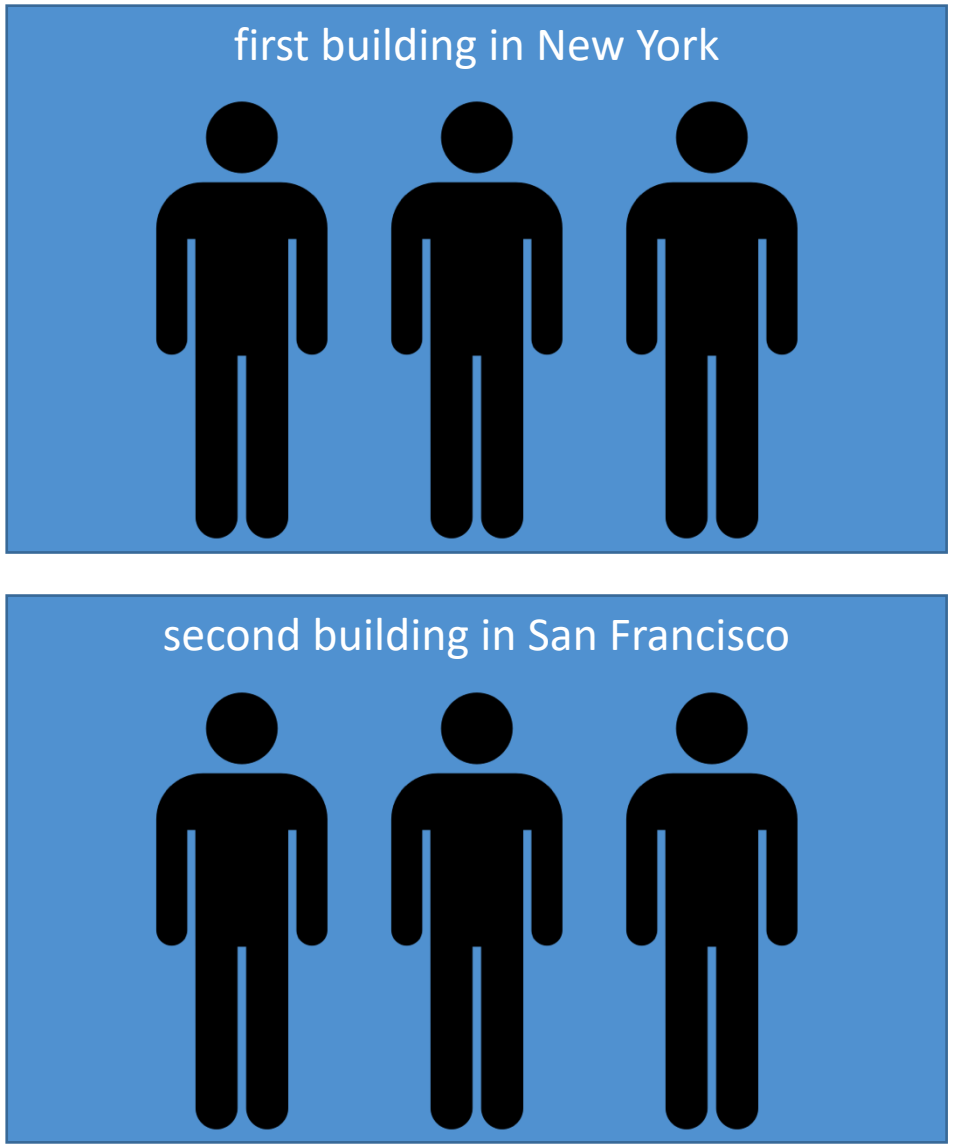
- It's easy to horizontally scale thanks the cloud offerings such as Amazon EC2



High Availability

- High Availability usually goes hand in hand with horizontal scaling
- High availability means running your application / system in at least 2 data centers (== Availability Zones)
- The goal of high availability is to survive a data center loss

- The high availability can be passive (for RDS Multi AZ for example)
- The high availability can be active (for horizontal scaling)



High Availability & Scalability For EC2

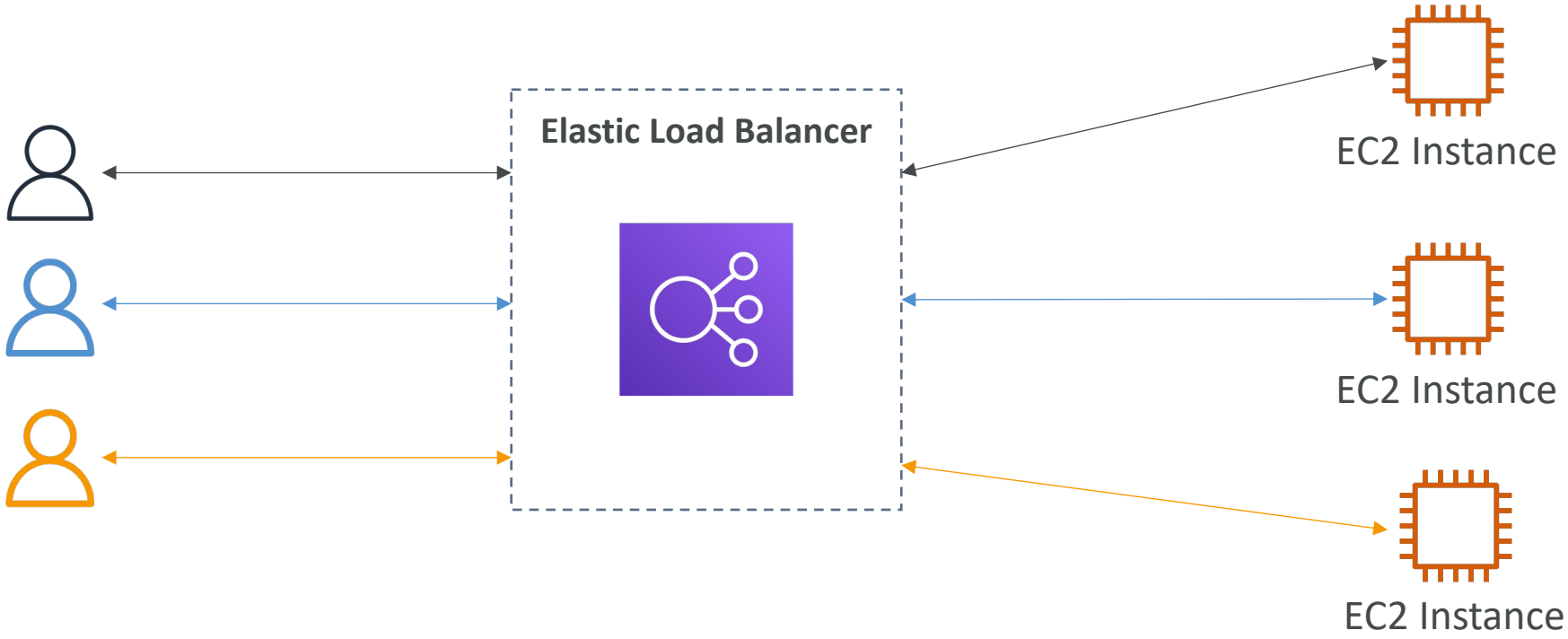
- Vertical Scaling: Increase instance size (= scale up / down)
  - From: t2.nano - 0.5G of RAM, 1 vCPU
  - To: u-12tb1.metal – 12.3 TB of RAM, 448 vCPUs

- Horizontal Scaling: Increase number of instances (= scale out / in)
  - Auto Scaling Group
  - Load Balancer

- High Availability: Run instances for the same application across multi AZ
  - Auto Scaling Group multi AZ
  - Load Balancer multi AZ

What is load balancing?

- Load Balances are servers that forward traffic to multiple servers (e.g., EC2 instances) downstream



Why use a load balancer?

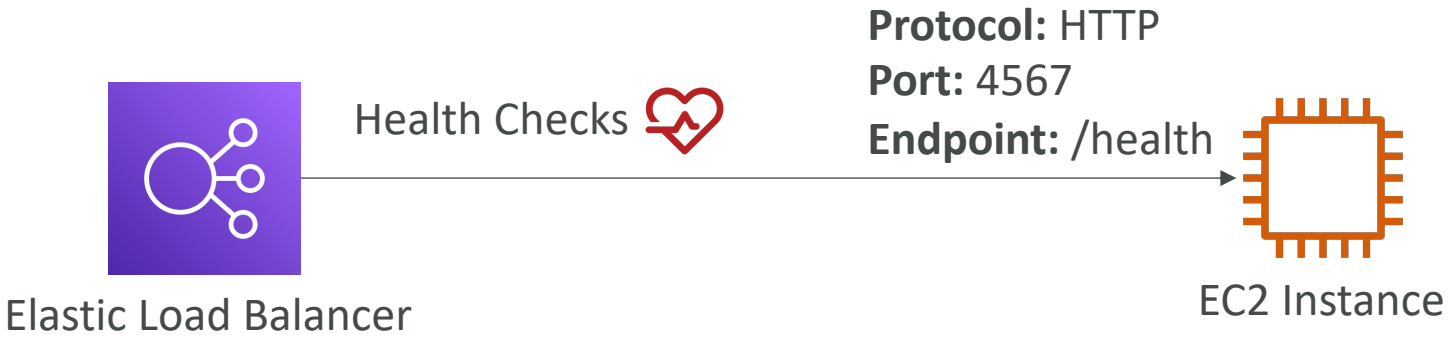
- Spread load across multiple downstream instances
- Expose a single point of access (DNS) to your application
- Seamlessly handle failures of downstream instances
- Do regular health checks to your instances
- Provide SSL termination (HTTPS) for your websites
- Enforce stickiness with cookies
- High availability across zones
- Separate public traffic from private traffic

Why use an Elastic Load Balancer?

- An Elastic Load Balancer is a managed load balancer
  - AWS guarantees that it will be working
  - AWS takes care of upgrades, maintenance, high availability
  - AWS provides only a few configuration knobs
- It costs less to setup your own load balancer but it will be a lot more effort on your end
- It is integrated with many AWS offerings / services
  - EC2, EC2 Auto Scaling Groups, Amazon ECS
  - AWS Certificate Manager (ACM), CloudWatch
  - Route 53, AWS WAF, AWS Global Accelerator

Health Checks

- Health Checks are crucial for Load Balancers
- They enable the load balancer to know if instances it forwards traffic to are available to reply to requests
- The health check is done on a port and a route (/health is common)
- If the response is not 200 (OK), then the instance is unhealthy

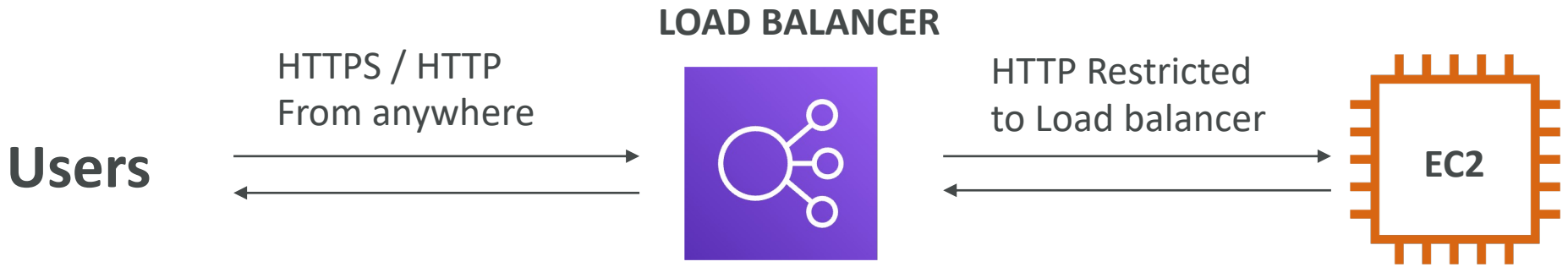


Types of load balancer on AWS

- AWS has 4 kinds of managed Load Balancers
- Classic Load Balancer (v1 - old generation) – 2009 – CLB
  - HTTP, HTTPS, TCP, SSL (secure TCP)
- Application Load Balancer (v2 - new generation) – 2016 – ALB
  - HTTP, HTTPS, WebSocket
- Network Load Balancer (v2 - new generation) – 2017 – NLB
  - TCP, TLS (secure TCP), UDP
- Gateway Load Balancer – 2020 – GWLB
  - Operates at layer 3 (Network layer) – IP Protocol

- Overall, it is recommended to use the newer generation load balancers as they provide more features
- Some load balancers can be setup as internal (private) or external (public) ELBs

Load Balancer Security Groups



Load Balancer Security Group:

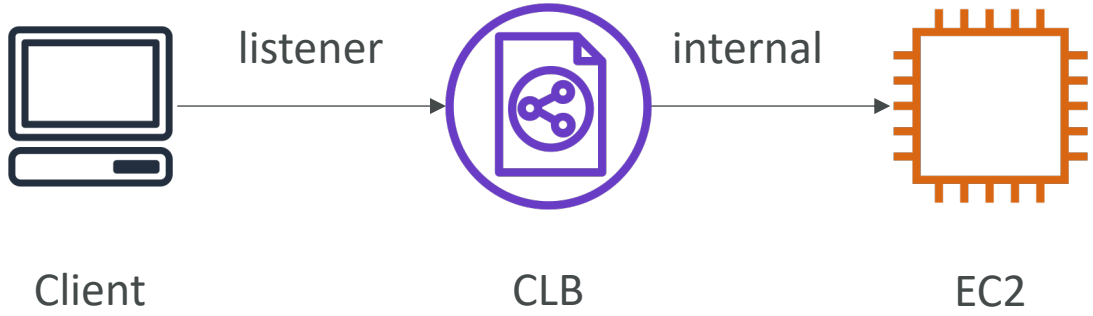
Type	Protocol	Port Range	Source	Description
HTTP	TCP	80	0.0.0.0/0	Allow HTTP from an...
HTTPS	TCP	443	0.0.0.0/0	Allow HTTPS from a...

Application Security Group: Allow traffic only from Load Balancer

Type	Protocol	Port Range	Source	Description
HTTP	TCP	80	sg-054b5ff5ea02f2b6e (load-b	Allow Traffic only...

Classic Load Balancers (v1)

- Supports TCP (Layer 4), HTTP & HTTPS (Layer 7)
- Health checks are TCP or HTTP based
- Fixed hostname XXX.region.elb.amazonaws.com



Application Load Balancer (v2)

- Application load balancers is Layer 7 (HTTP)

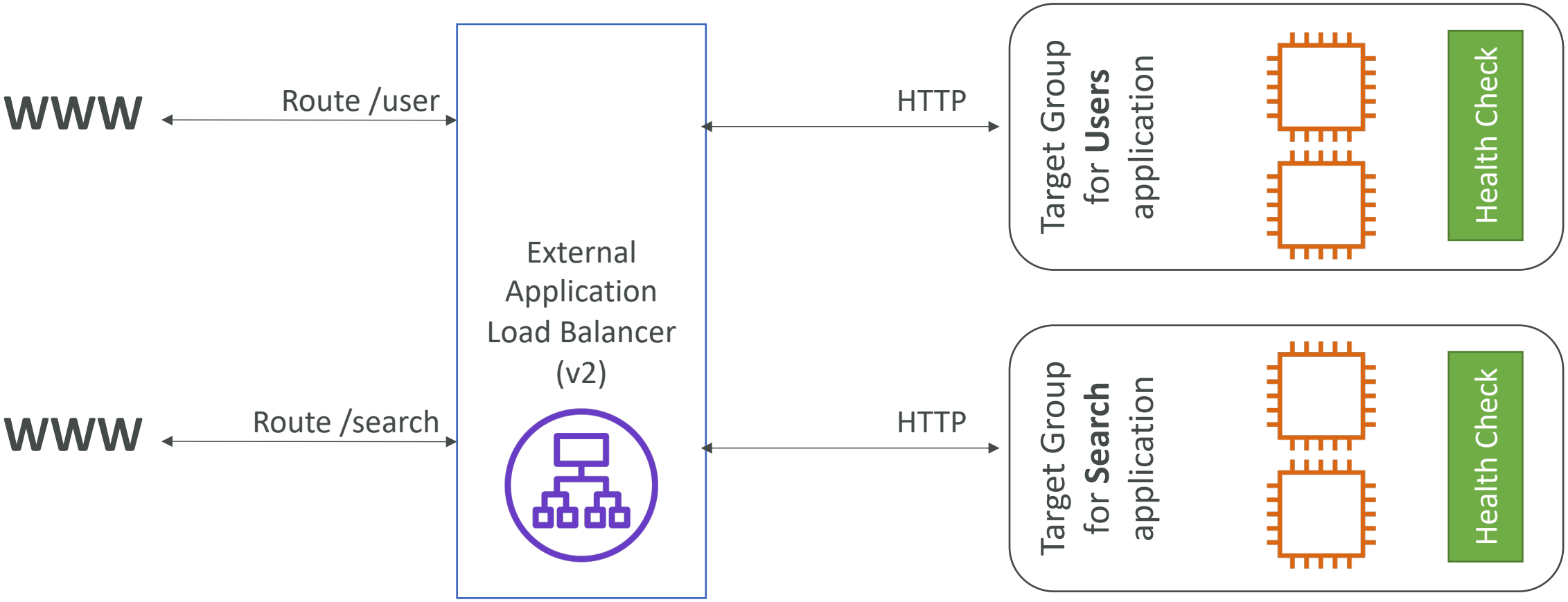
- Load balancing to multiple HTTP applications across machines (target groups)
- Load balancing to multiple applications on the same machine (ex: containers)
- Support for HTTP/2 and WebSocket
- Support redirects (from HTTP to HTTPS for example)

Application Load Balancer (v2)

- Routing tables to different target groups:
  - Routing based on path in URL (example.com/users & example.com/posts)
  - Routing based on hostname in URL (one.example.com & other.example.com)
  - Routing based on Query String, Headers (example.com/users?id=123&order=false)

- ALB are a great fit for micro services & container-based application (example: Docker & Amazon ECS)
- Has a port mapping feature to redirect to a dynamic port in ECS
- In comparison, we'd need multiple Classic Load Balancer per application

Application Load Balancer (v2)
HTTP Based Traffic

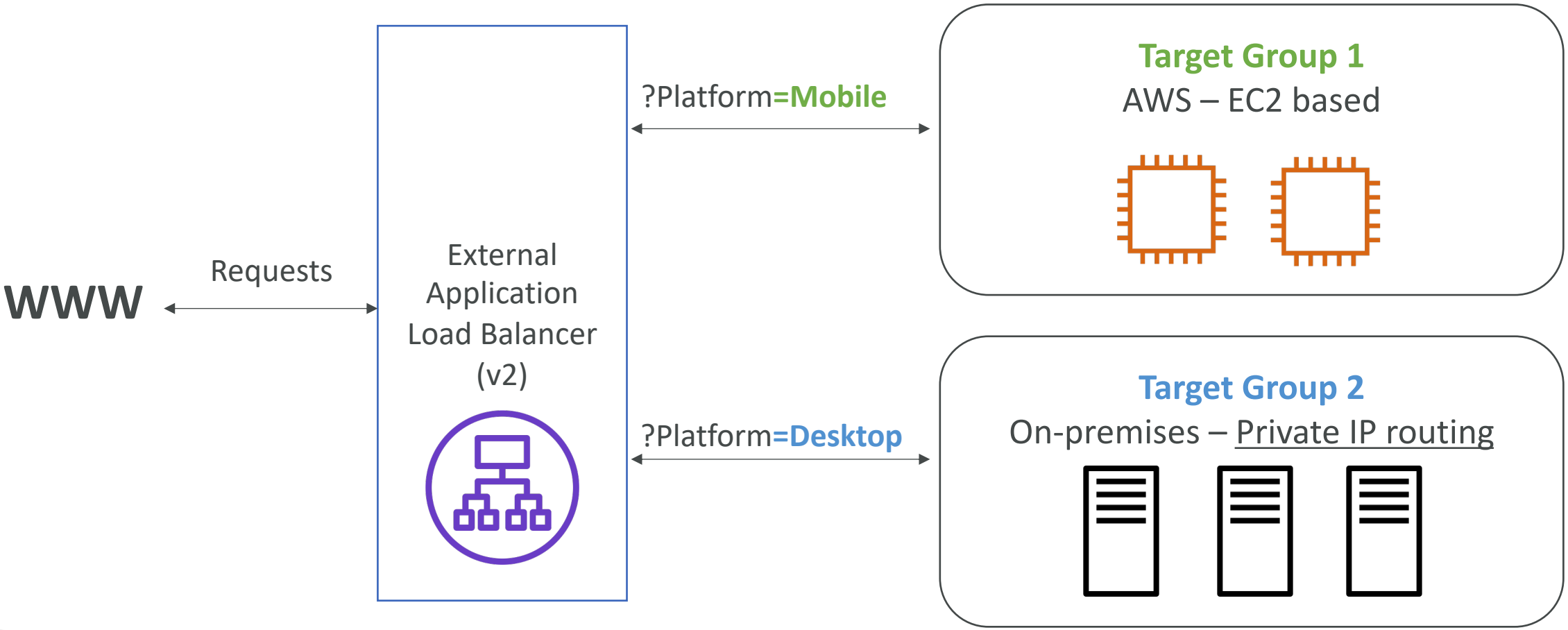


Application Load Balancer (v2) Target Groups

- EC2 instances (can be managed by an Auto Scaling Group) – HTTP
- ECS tasks (managed by ECS itself) – HTTP
- Lambda functions – HTTP request is translated into a JSON event
- IP Addresses – must be private IPs

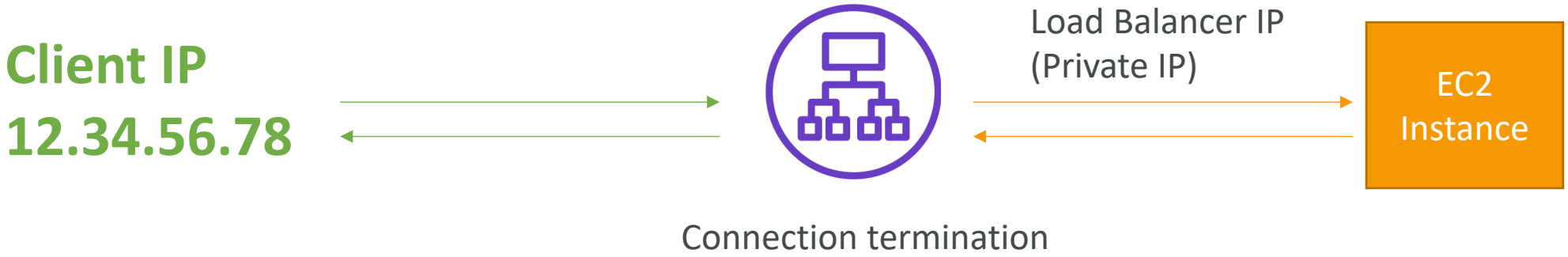
- ALB can route to multiple target groups
- Health checks are at the target group level

Application Load Balancer (v2) Query Strings/Parameters Routing

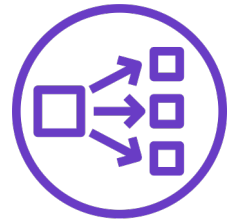


Application Load Balancer (v2) Good to Know

- Fixed hostname (XXX.region.elb.amazonaws.com)
- The application servers don't see the IP of the client directly
  - The true IP of the client is inserted in the header **X-Forwarded-For**
  - We can also get Port (X-Forwarded-Port) and proto (X-Forwarded-Proto)



Network Load Balancer (v2)

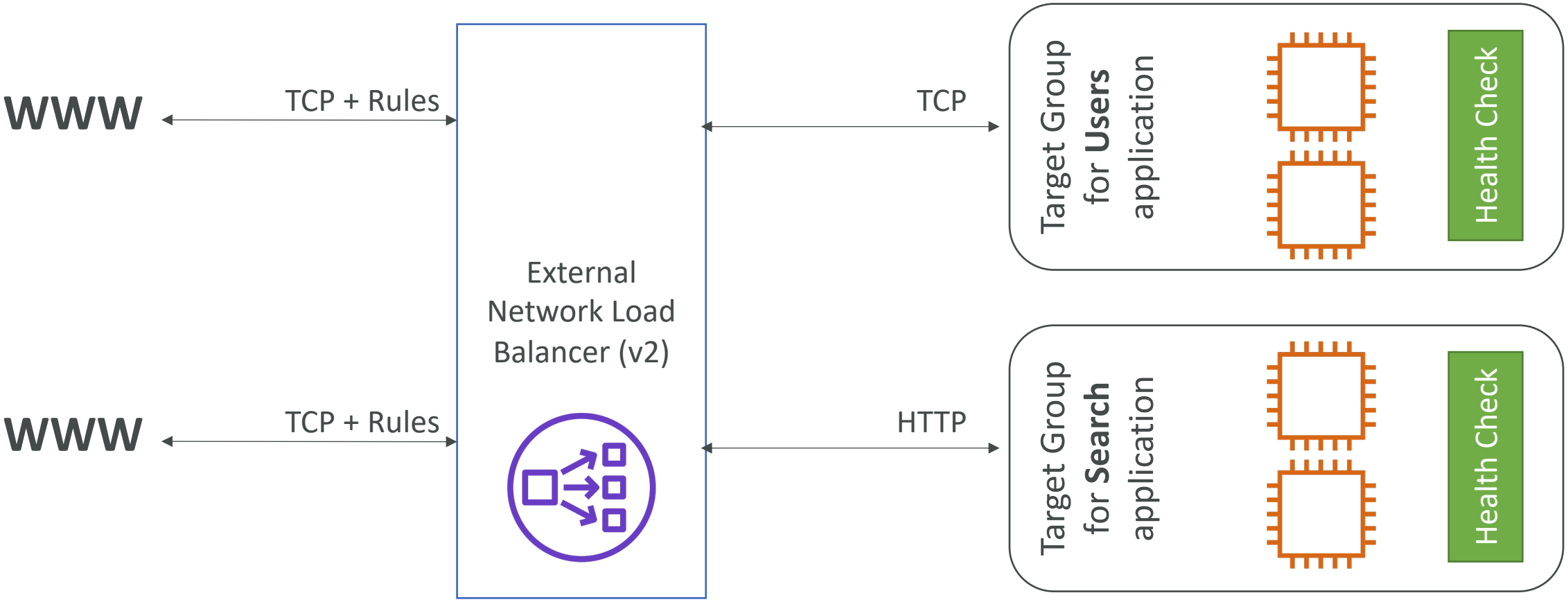


- Network load balancers (Layer 4) allow to:
  - **Forward TCP & UDP traffic to your instances**
  - Handle millions of request per seconds
  - Ultra-low latency

- **NLB has one static IP per AZ, and supports assigning Elastic IP** (helpful for whitelisting specific IP)

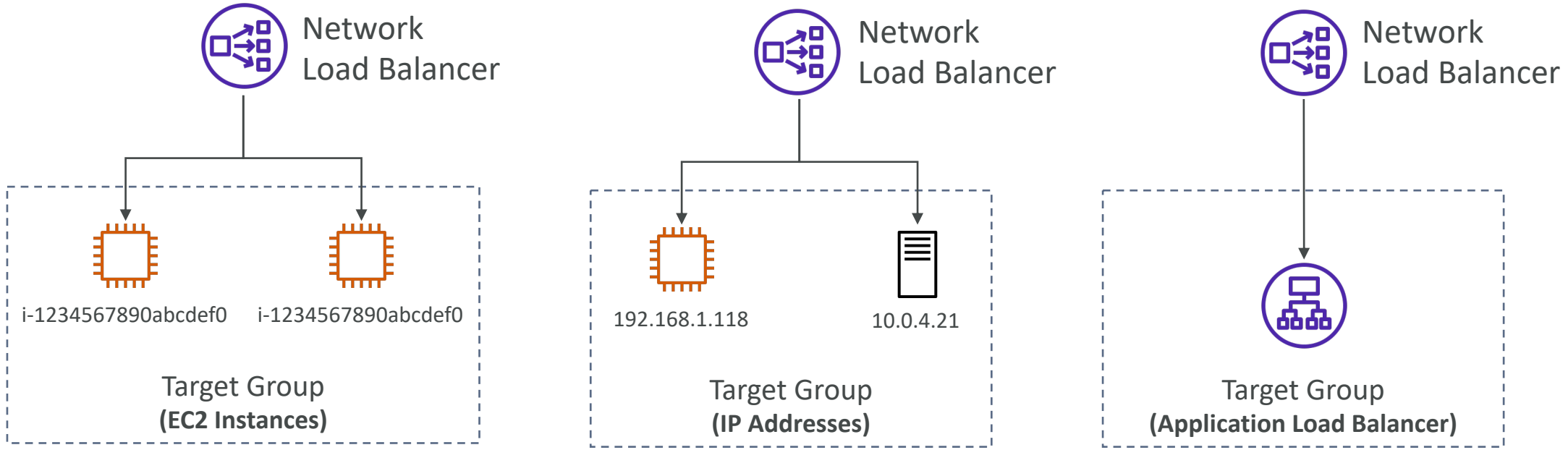
- NLB are used for extreme performance, TCP or UDP traffic
- Not included in the AWS free tier

Network Load Balancer (v2)
TCP (Layer 4) Based Traffic



Network Load Balancer – Target Groups

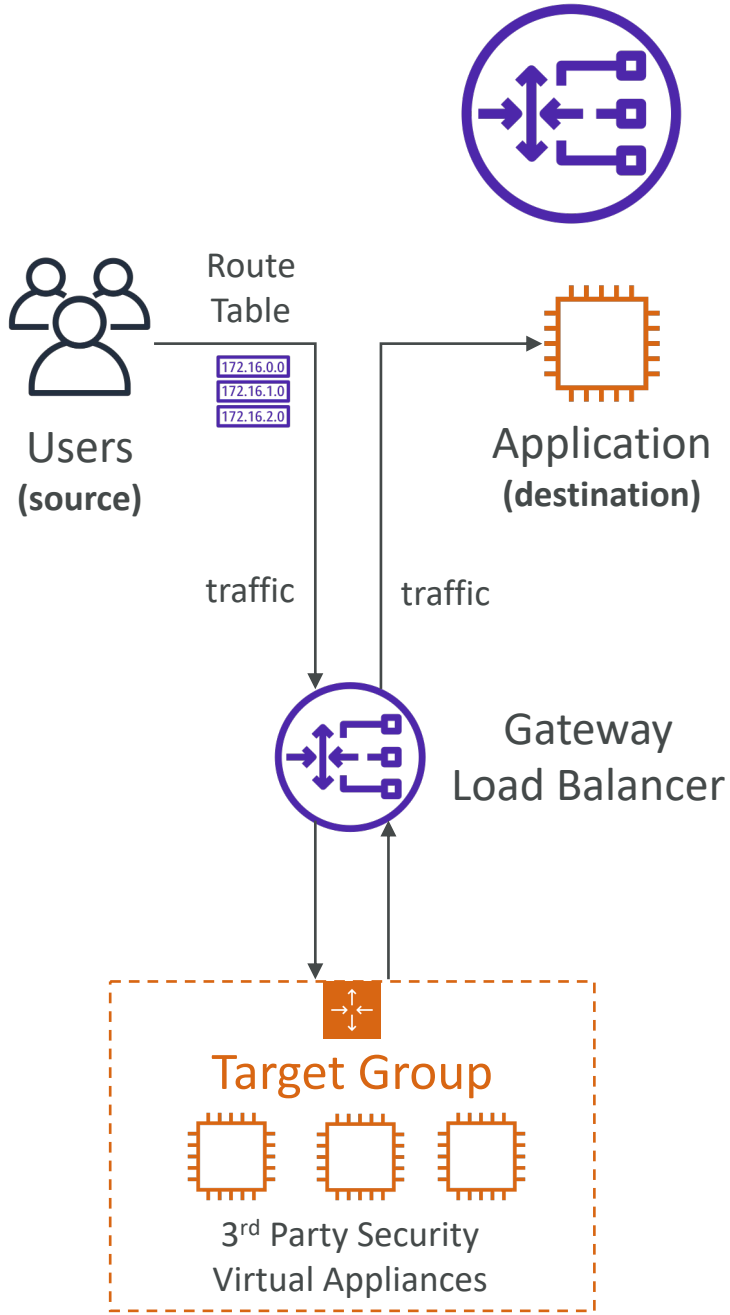
- EC2 instances
- IP Addresses – must be private IPs
- Application Load Balancer
- Health Checks support the TCP, HTTP and HTTPS Protocols



Gateway Load Balancer

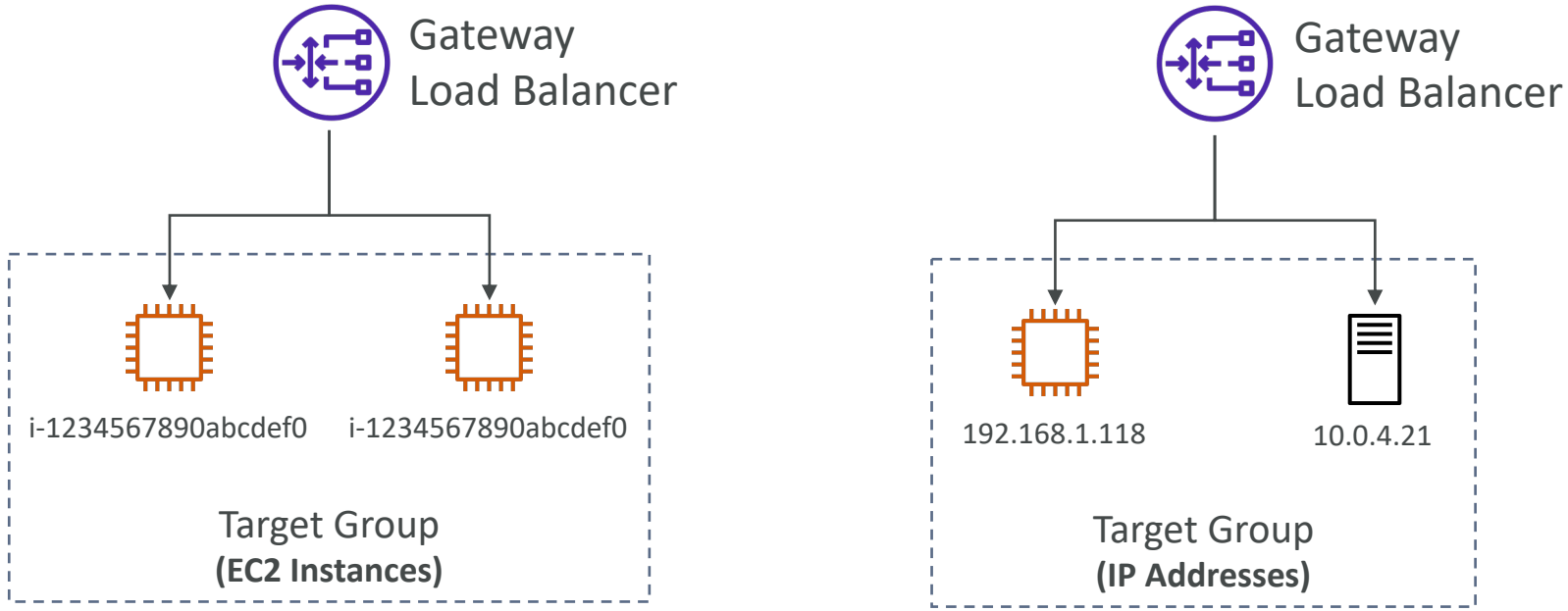
- Deploy, scale, and manage a fleet of 3rd party network virtual appliances in AWS
- Example: Firewalls, Intrusion Detection and Prevention Systems, Deep Packet Inspection Systems, payload manipulation, …

- Operates at Layer 3 (Network Layer) – IP Packets
- Combines the following functions:
  - **Transparent Network Gateway** – single entry/exit for all traffic
  - **Load Balancer** – distributes traffic to your virtual appliances
- Uses the **GENEVE** protocol on port **6081**



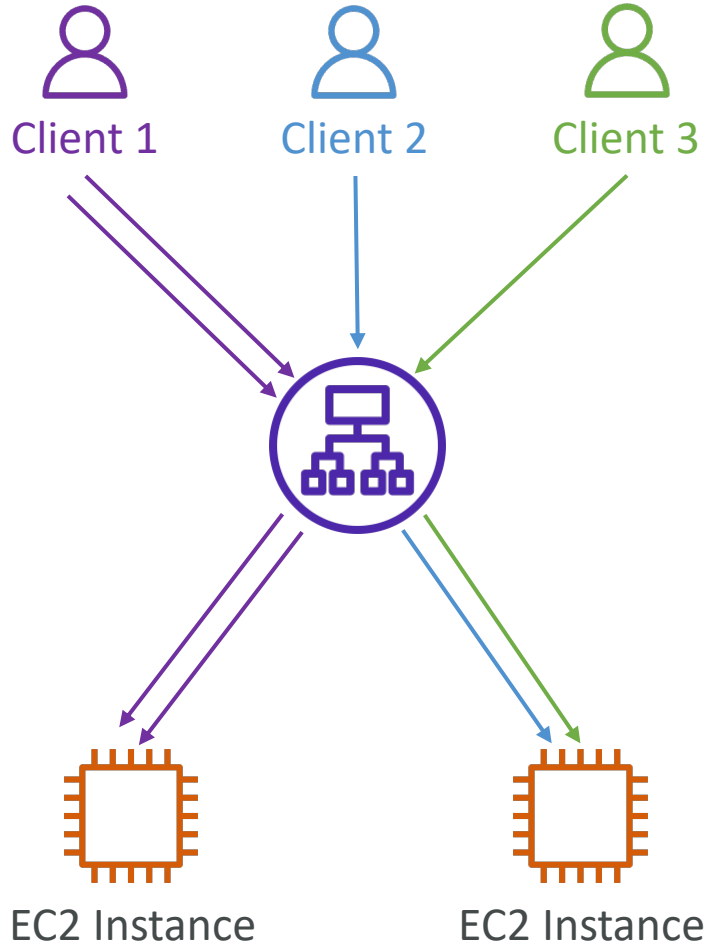
Gateway Load Balancer – Target Groups

- EC2 instances
- IP Addresses – must be private IPs



Sticky Sessions (Session Affinity)

- It is possible to implement stickiness so that the same client is always redirected to the same instance behind a load balancer
- This works for **Classic Load Balancer, Application Load Balancer, and Network Load Balancer**
- For both CLB & ALB, the "cookie" used for stickiness has an expiration date you control
- Use case: make sure the user doesn't lose his session data
- Enabling stickiness may bring imbalance to the load over the backend EC2 instances



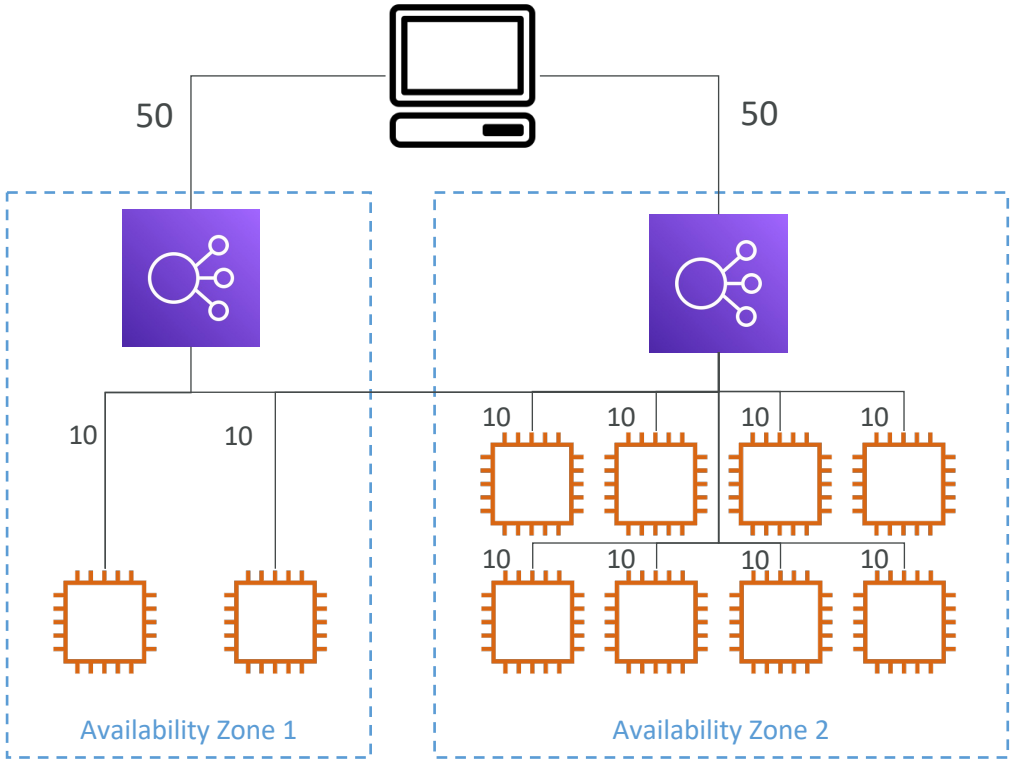
Sticky Sessions – Cookie Names

- Application-based Cookies
  - Custom cookie
    - Generated by the target
    - Can include any custom attributes required by the application
    - Cookie name must be specified individually for each target group
    - Don't use **AWSALB**, **AWSALBAPP**, or **AWSALBTG** (reserved for use by the ELB)
  - Application cookie
    - Generated by the load balancer
    - Cookie name is **AWSALBAPP**
- Duration-based Cookies
  - Cookie generated by the load balancer
  - Cookie name is **AWSALB** for ALB, **AWSELB** for CLB

Cross-Zone Load Balancing

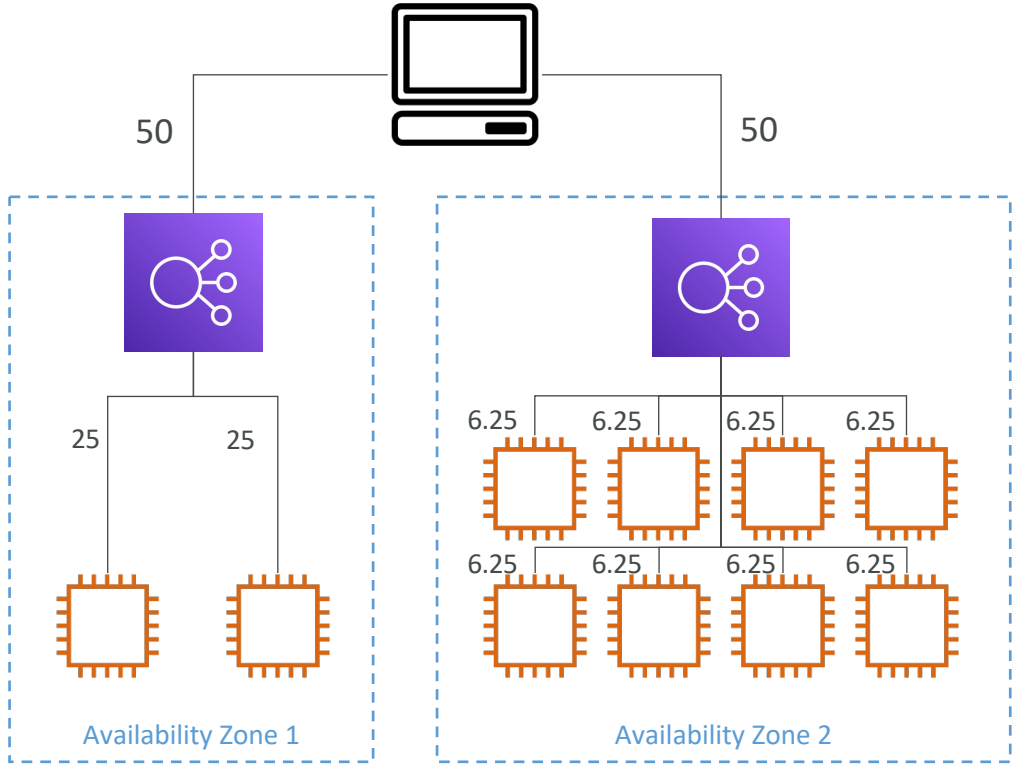
With Cross Zone Load Balancing:

each load balancer instance distributes evenly across all registered instances in all AZ



Without Cross Zone Load Balancing:

Requests are distributed in the instances of the node of the Elastic Load Balancer



Cross-Zone Load Balancing

- Application Load Balancer
  - Enabled by default (can be disabled at the Target Group level)
  - No charges for inter AZ data
- Network Load Balancer & Gateway Load Balancer
  - Disabled by default
  - You pay charges ($) for inter AZ data if enabled
- Classic Load Balancer
  - Disabled by default
  - No charges for inter AZ data if enabled

SSL/TLS - Basics

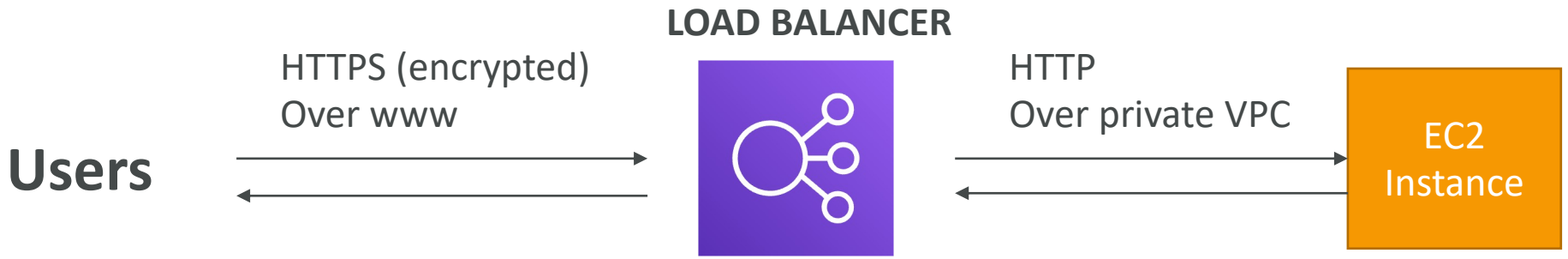
- An SSL Certificate allows traffic between your clients and your load balancer to be encrypted in transit (in-flight encryption)

- **SSL** refers to Secure Sockets Layer, used to encrypt connections
- **TLS** refers to Transport Layer Security, which is a newer version
- Nowadays, **TLS certificates are mainly used**, but people still refer as SSL

- Public SSL certificates are issued by Certificate Authorities (CA)
- Comodo, Symantec, GoDaddy, GlobalSign, Digicert, Letsencrypt, etc…

- SSL certificates have an expiration date (you set) and must be renewed

Load Balancer - SSL Certificates



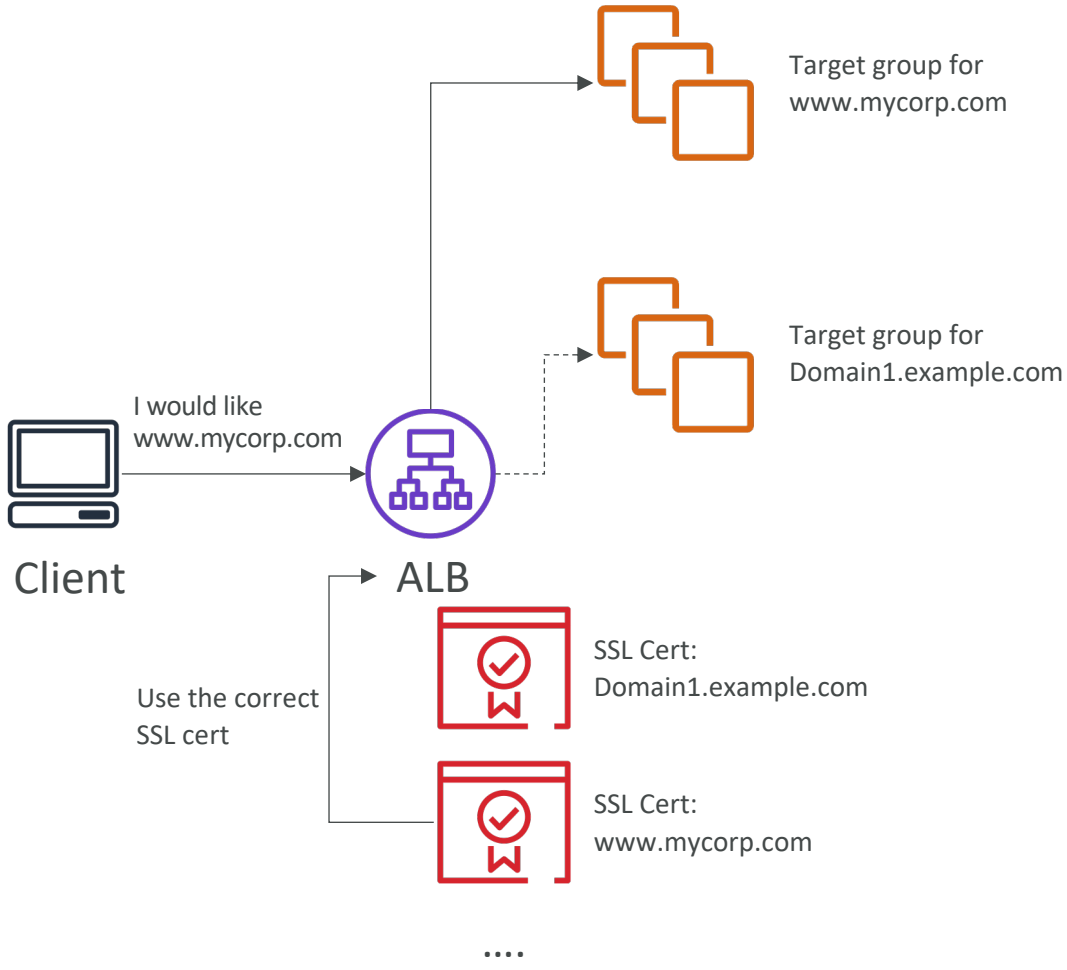
- The load balancer uses an X.509 certificate (SSL/TLS server certificate)
- You can manage certificates using ACM (AWS Certificate Manager)
- You can create upload your own certificates alternatively
- HTTPS listener:
  - You must specify a default certificate
  - You can add an optional list of certs to support multiple domains
  - Clients can use SNI (Server Name Indication) to specify the hostname they reach
  - Ability to specify a security policy to support older versions of SSL / TLS (legacy clients)

SSL – Server Name Indication (SNI)

- SNI solves the problem of loading **multiple SSL certificates onto one web server** (to serve multiple websites)
- It's a "newer" protocol, and requires the client to **indicate** the hostname of the target server in the initial SSL handshake
- The server will then find the correct certificate, or return the default one

Note:

- Only works for ALB & NLB (newer generation), CloudFront
- Does not work for CLB (older gen)



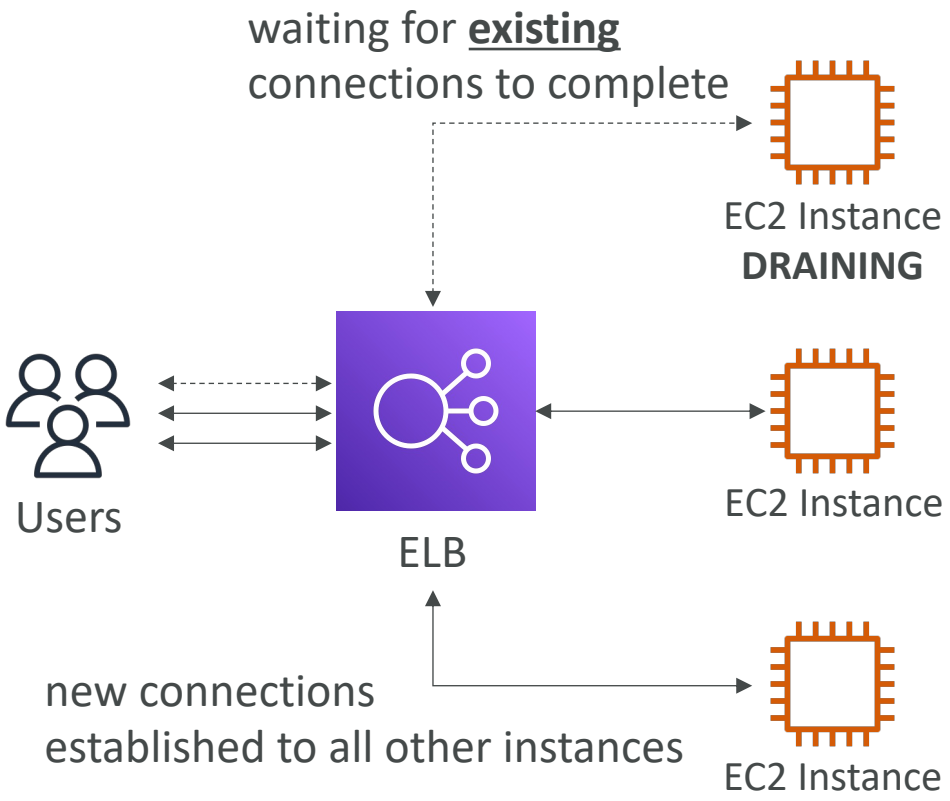
Elastic Load Balancers – SSL Certificates

- Classic Load Balancer (v1)
  - Support only one SSL certificate
  - Must use multiple CLB for multiple hostname with multiple SSL certificates
- Application Load Balancer (v2)
  - Supports multiple listeners with multiple SSL certificates
  - Uses Server Name Indication (SNI) to make it work
- Network Load Balancer (v2)
  - Supports multiple listeners with multiple SSL certificates
  - Uses Server Name Indication (SNI) to make it work

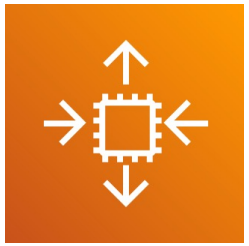
Connection Draining

- Feature naming
  - Connection Draining – for CLB
  - Deregistration Delay – for ALB & NLB

- Time to complete "in-flight requests" while the instance is de-registering or unhealthy
- Stops sending new requests to the EC2 instance which is de-registering
- Between 1 to 3600 seconds (default: 300 seconds)
- Can be disabled (set value to 0)
- Set to a low value if your requests are short



What's an Auto Scaling Group?

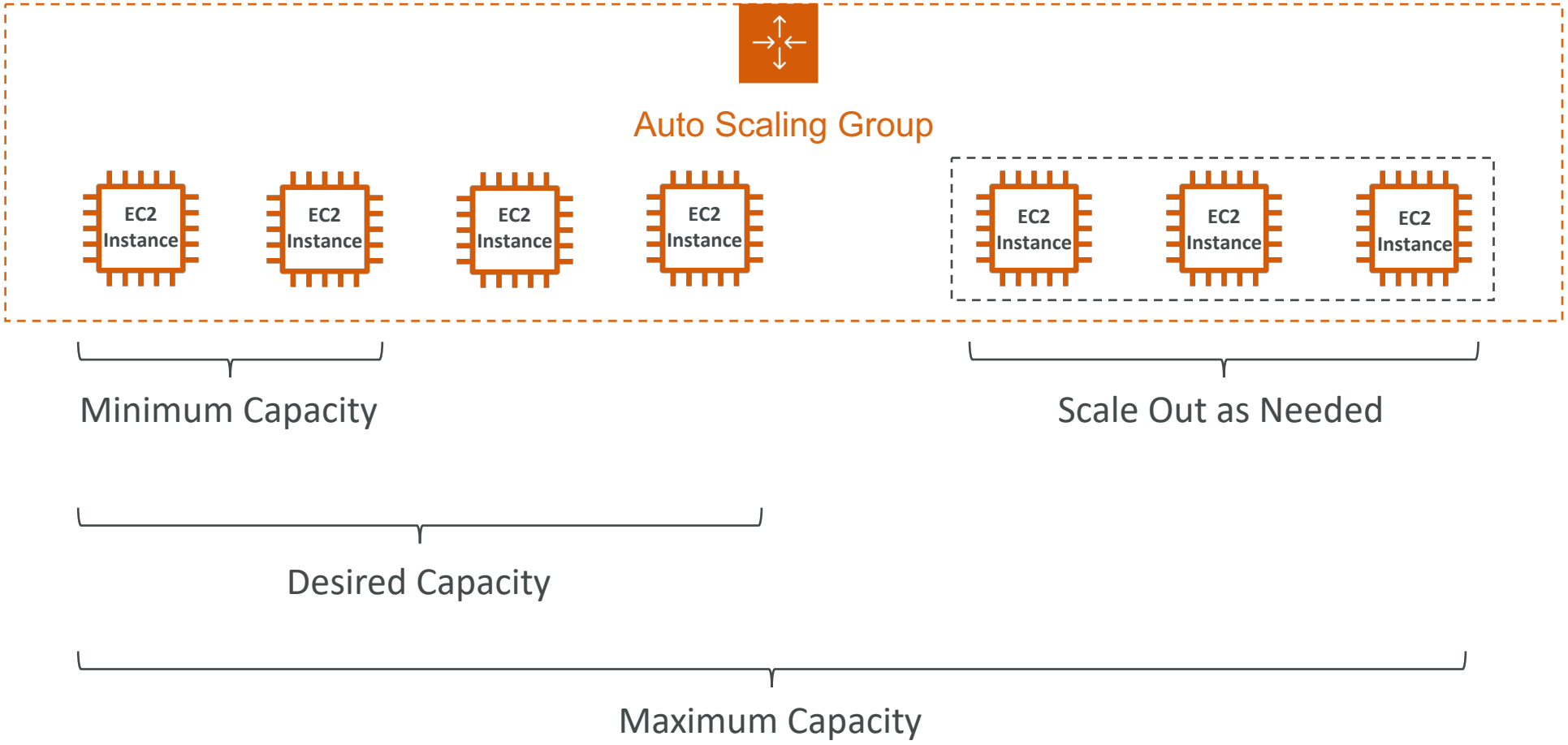


- In real-life, the load on your websites and application can change
- In the cloud, you can create and get rid of servers very quickly

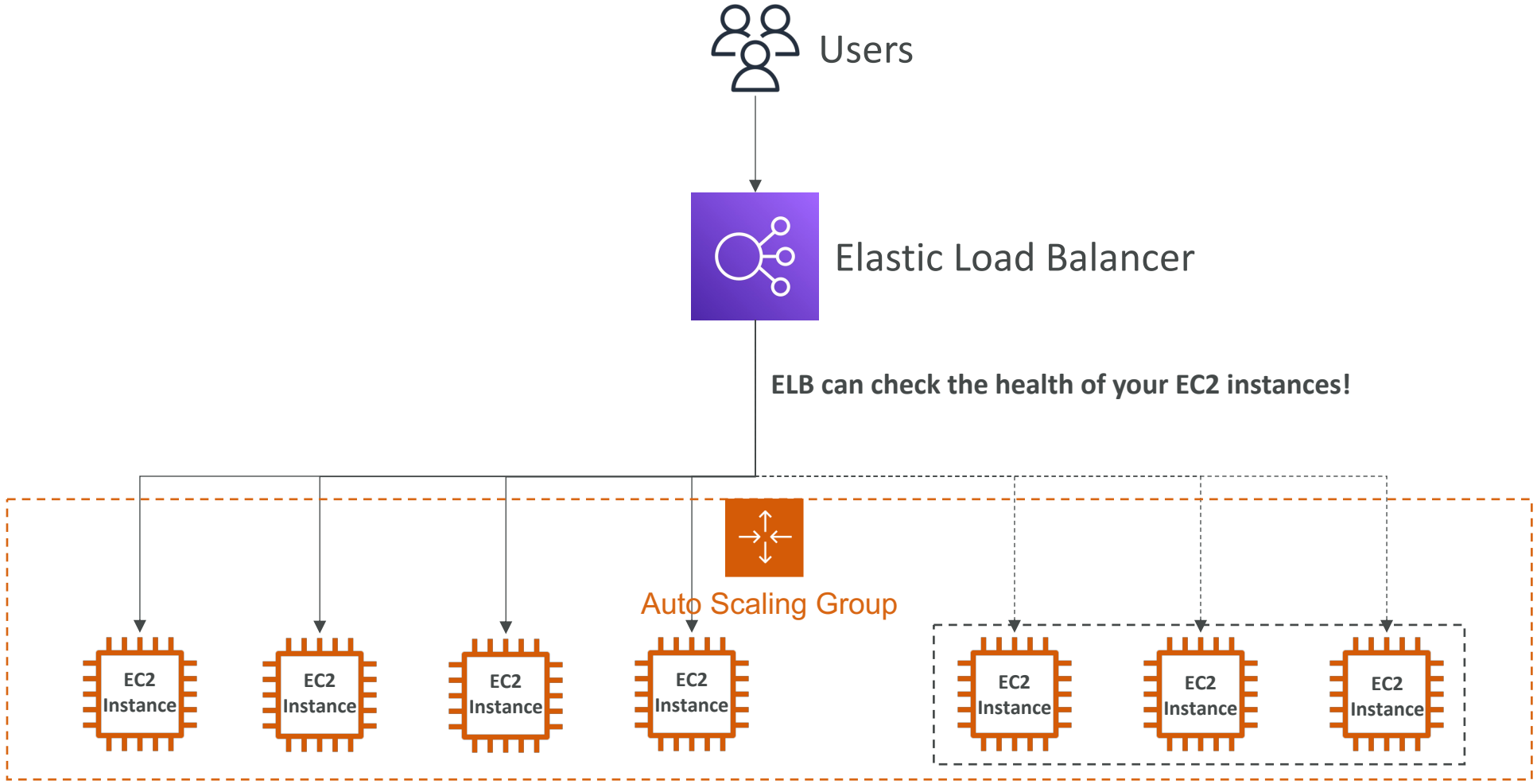
- The goal of an Auto Scaling Group (ASG) is to:
  - Scale out (add EC2 instances) to match an increased load
  - Scale in (remove EC2 instances) to match a decreased load
  - Ensure we have a minimum and a maximum number of EC2 instances running
  - Automatically register new instances to a load balancer
  - Re-create an EC2 instance in case a previous one is terminated (ex: if unhealthy)

- ASG are free (you only pay for the underlying EC2 instances)

Auto Scaling Group in AWS

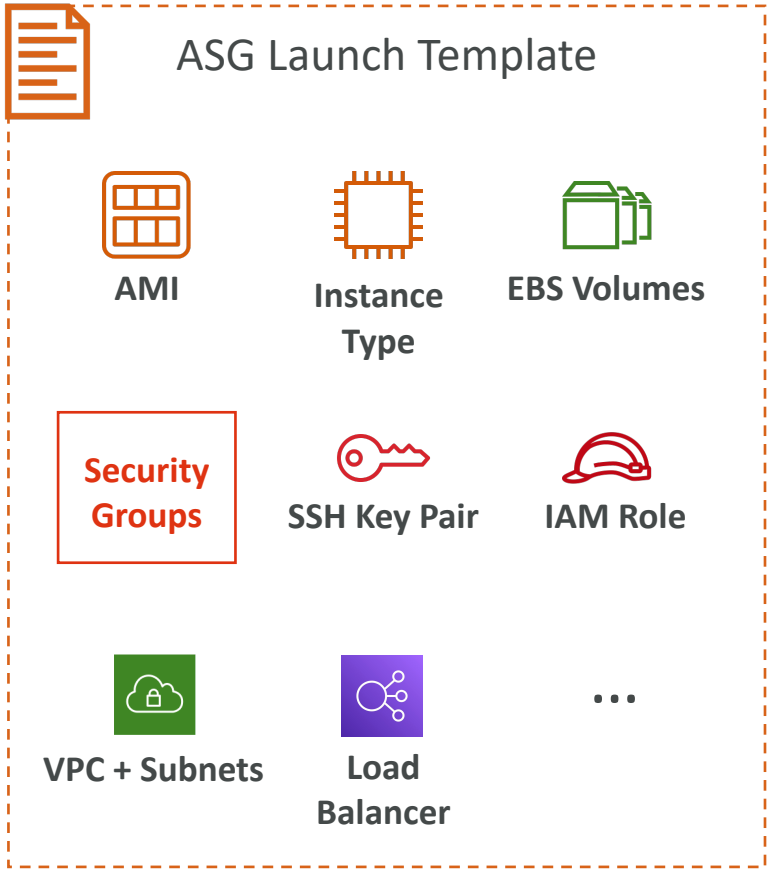


Auto Scaling Group in AWS With Load Balancer



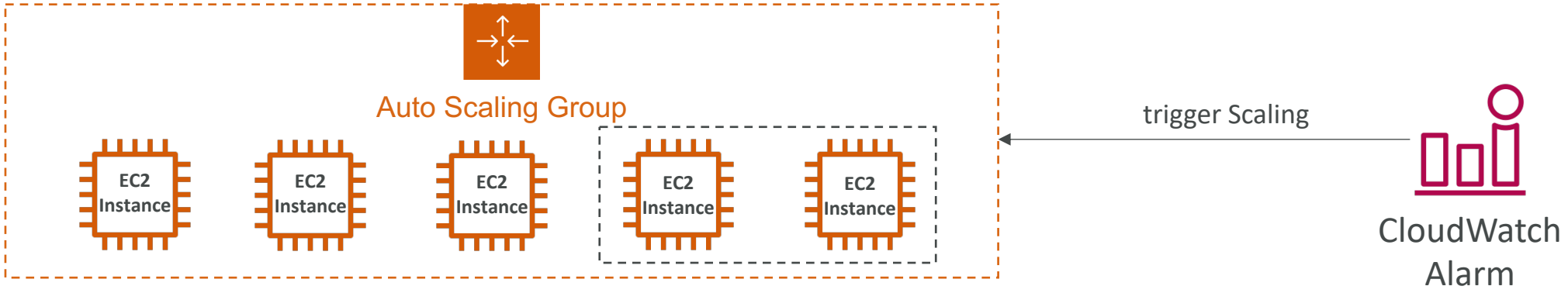
Auto Scaling Group Attributes

- A **Launch Template** (older "Launch Configurations" are deprecated)
  - AMI + Instance Type
  - EC2 User Data
  - EBS Volumes
  - Security Groups
  - SSH Key Pair
  - IAM Roles for your EC2 Instances
  - Network + Subnets Information
  - Load Balancer Information
- Min Size / Max Size / Initial Capacity
- Scaling Policies



Auto Scaling - CloudWatch Alarms & Scaling

- It is possible to scale an ASG based on CloudWatch alarms
- An alarm monitors a metric (such as **Average CPU**, or a **custom metric**)
- Metrics such as Average CPU are computed for the overall ASG instances
- Based on the alarm:
  - We can create scale-out policies (increase the number of instances)
  - We can create scale-in policies (decrease the number of instances)

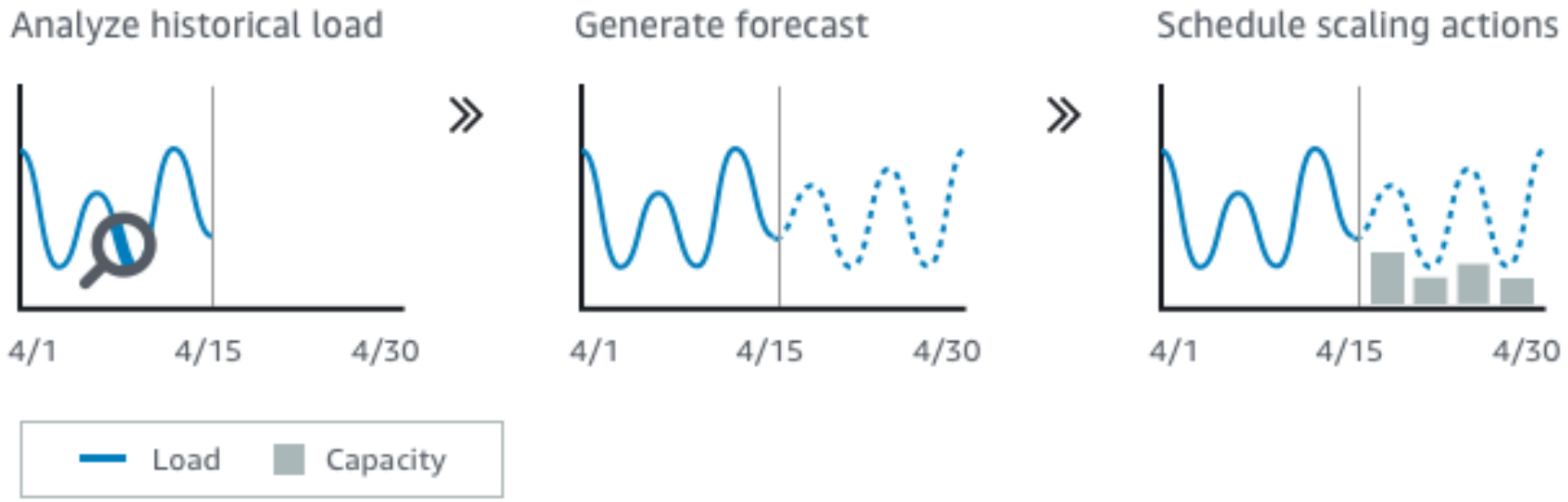


Auto Scaling Groups – Scaling Policies

- Dynamic Scaling
  - Target Tracking Scaling
    - Simple to set-up
    - Example: I want the average ASG CPU to stay at around 40%
  - Simple / Step Scaling
    - When a CloudWatch alarm is triggered (example CPU > 70%), then add 2 units
    - When a CloudWatch alarm is triggered (example CPU < 30%), then remove 1
- Scheduled Scaling
  - Anticipate a scaling based on known usage patterns
  - Example: increase the min capacity to 10 at 5 pm on Fridays

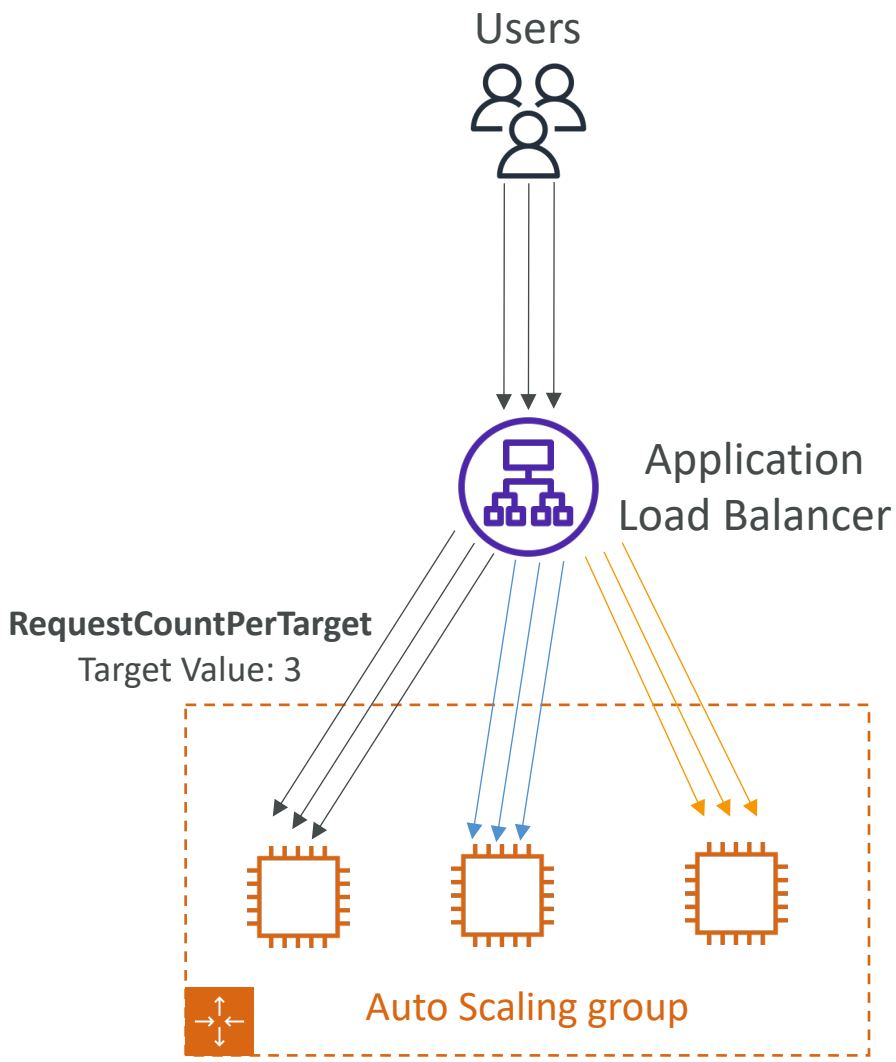
Auto Scaling Groups – Scaling Policies

- **Predictive scaling:** continuously forecast load and schedule scaling ahead



Good metrics to scale on

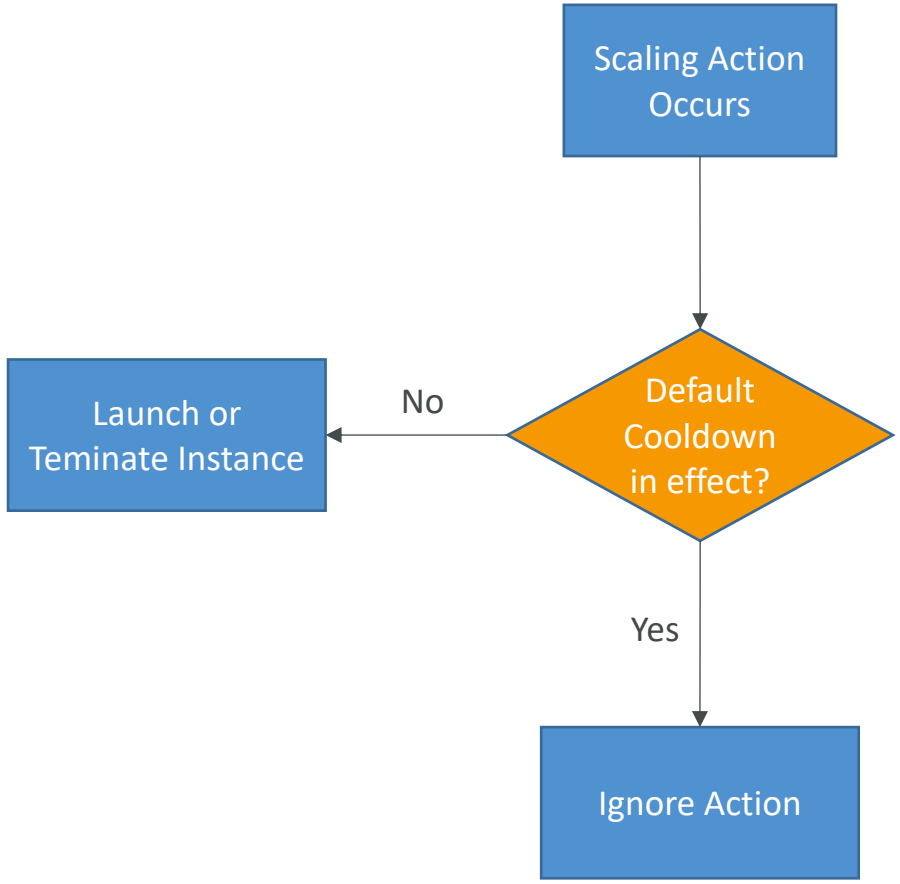
- **CPUUtilization**: Average CPU utilization across your instances
- **RequestCountPerTarget**: to make sure the number of requests per EC2 instances is stable
- **Average Network In / Out** (if you're application is network bound)
- **Any custom metric** (that you push using CloudWatch)



Auto Scaling Groups - Scaling Cooldowns

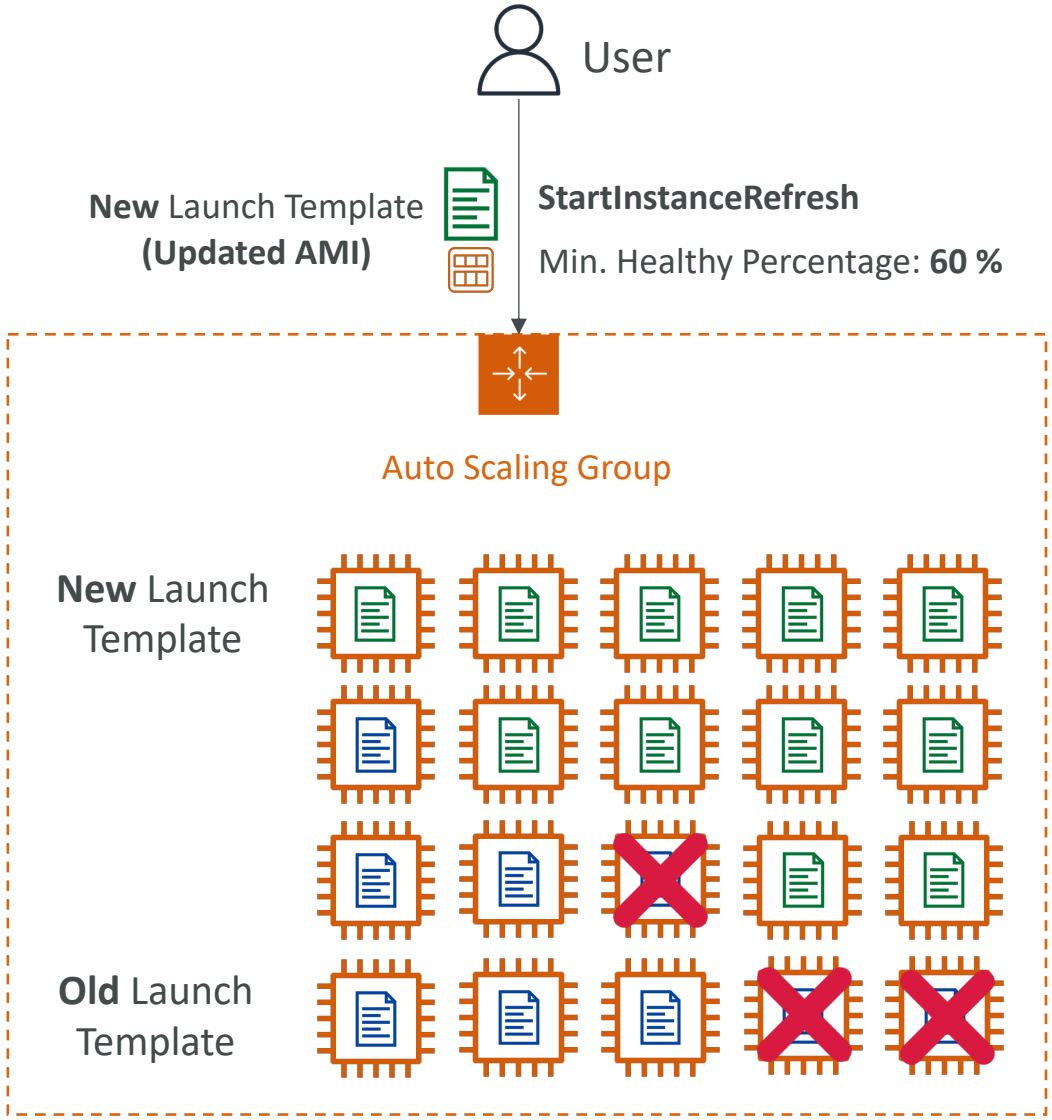
- After a scaling activity happens, you are in the **cooldown period (default 300 seconds)**
- During the cooldown period, the ASG will not launch or terminate additional instances (to allow for metrics to stabilize)

- Advice: Use a ready-to-use AMI to reduce configuration time in order to be serving request fasters and reduce the cooldown period



Auto Scaling – Instance Refresh

- Goal: update launch template and then re-creating all EC2 instances
- For this we can use the native feature of Instance Refresh
- Setting of minimum healthy percentage
- Specify warm-up time (how long until the instance is ready to use)



RDS, Aurora, & ElastiCache

Amazon RDS Overview

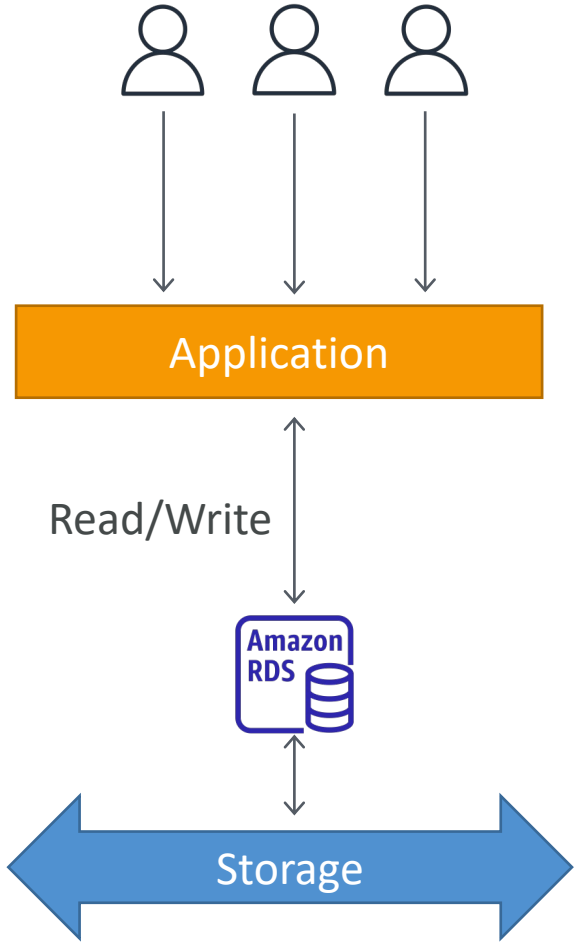
- RDS stands for Relational Database Service
- It's a managed DB service for DB use SQL as a query language.
- It allows you to create databases in the cloud that are managed by AWS
  - Postgres
  - MySQL
  - MariaDB
  - Oracle
  - Microsoft SQL Server
  - IBM DB2
  - Aurora (AWS Proprietary database)

Advantage over using RDS versus deploying DB on EC2

- RDS is a managed service:
  - Automated provisioning, OS patching
  - Continuous backups and restore to specific timestamp (Point in Time Restore)!
  - Monitoring dashboards
  - Read replicas for improved read performance
  - Multi AZ setup for DR (Disaster Recovery)
  - Maintenance windows for upgrades
  - Scaling capability (vertical and horizontal)
  - Storage backed by EBS
- BUT you can't SSH into your instances

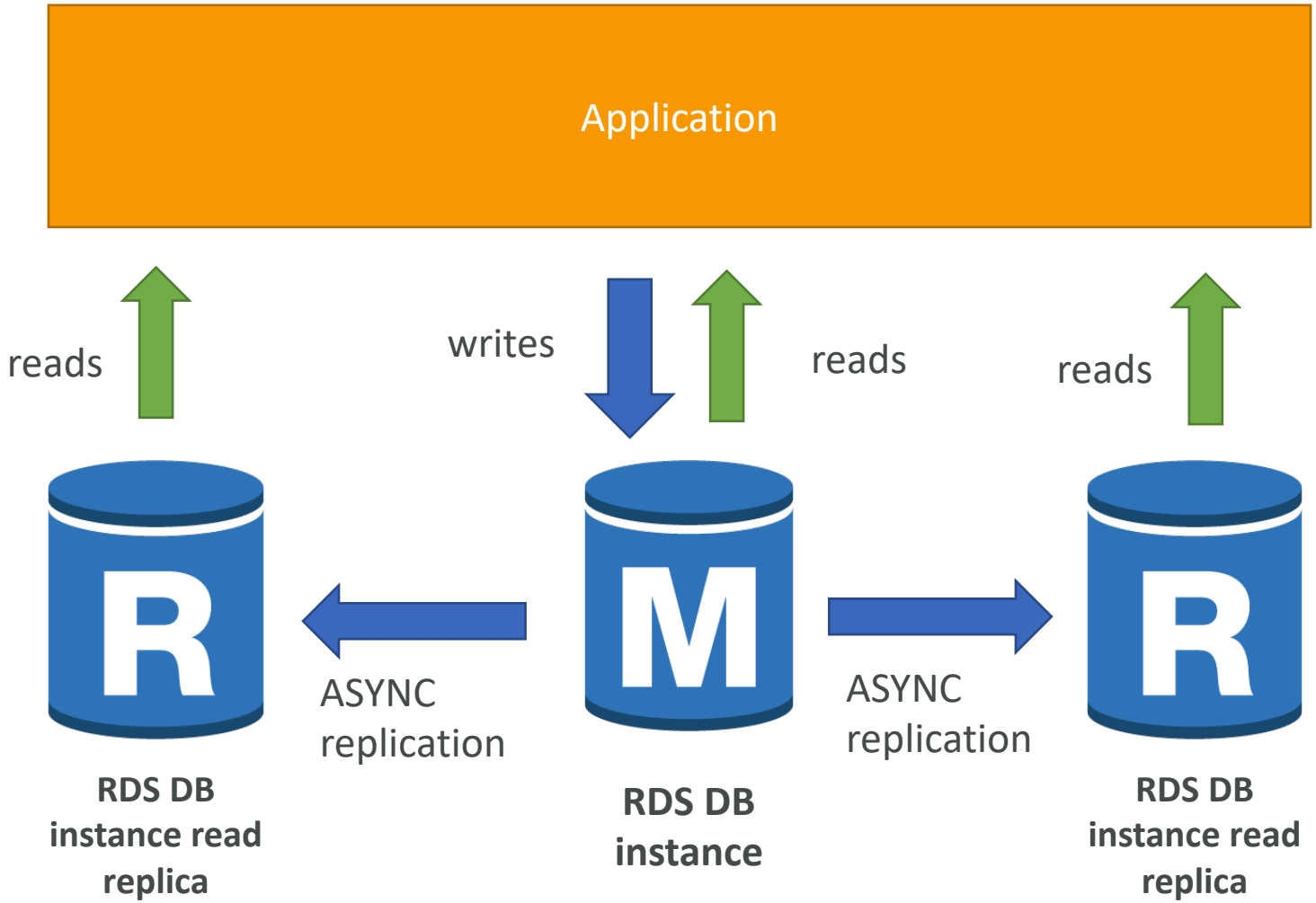
RDS – Storage Auto Scaling

- Helps you increase storage on your RDS DB instance dynamically
- When RDS detects you are running out of free database storage, it scales automatically
- Avoid manually scaling your database storage
- You have to set **Maximum Storage Threshold** (maximum limit for DB storage)
- Automatically modify storage if:
  - Free storage is less than 10% of allocated storage
  - Low-storage lasts at least 5 minutes
  - 6 hours have passed since last modification
- Useful for applications with **unpredictable workloads**
- Supports all RDS database engines



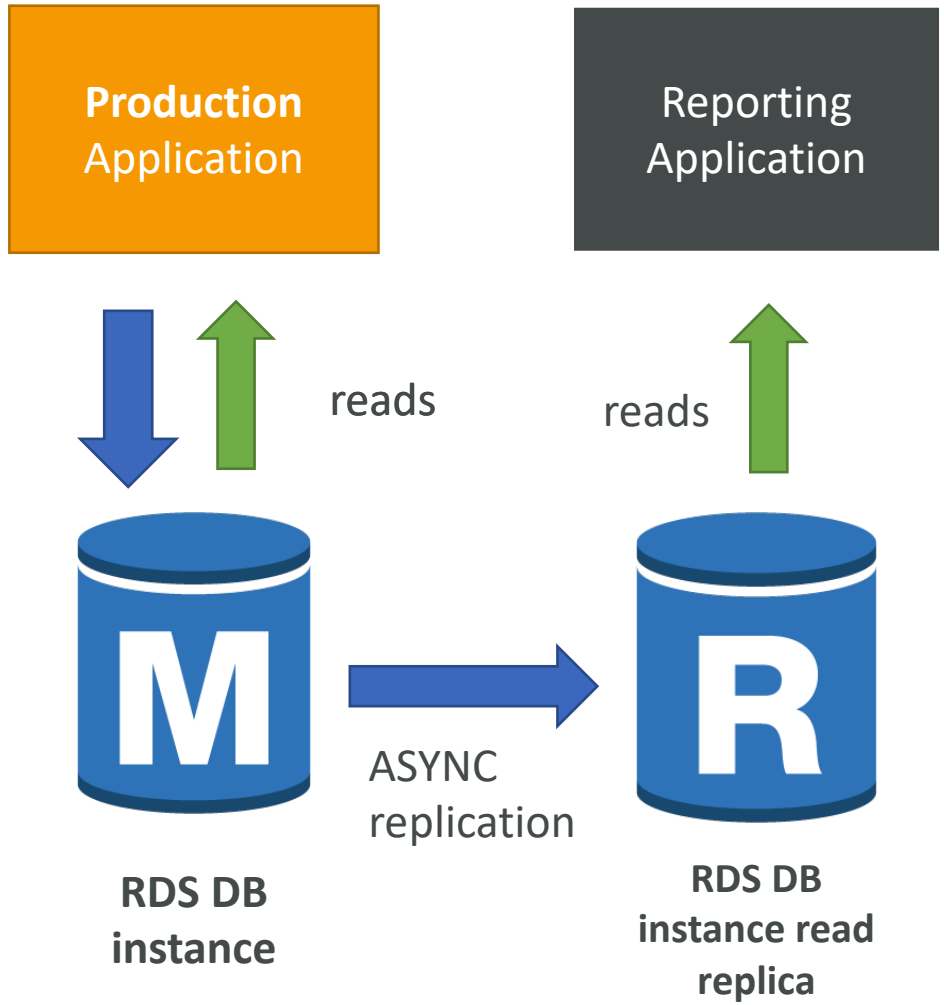
RDS Read Replicas for read scalability

- Up to 15 Read Replicas
- Within AZ, Cross AZ or Cross Region
- Replication is **ASYNC**, so reads are eventually consistent
- Replicas can be promoted to their own DB
- Applications must update the connection string to leverage read replicas



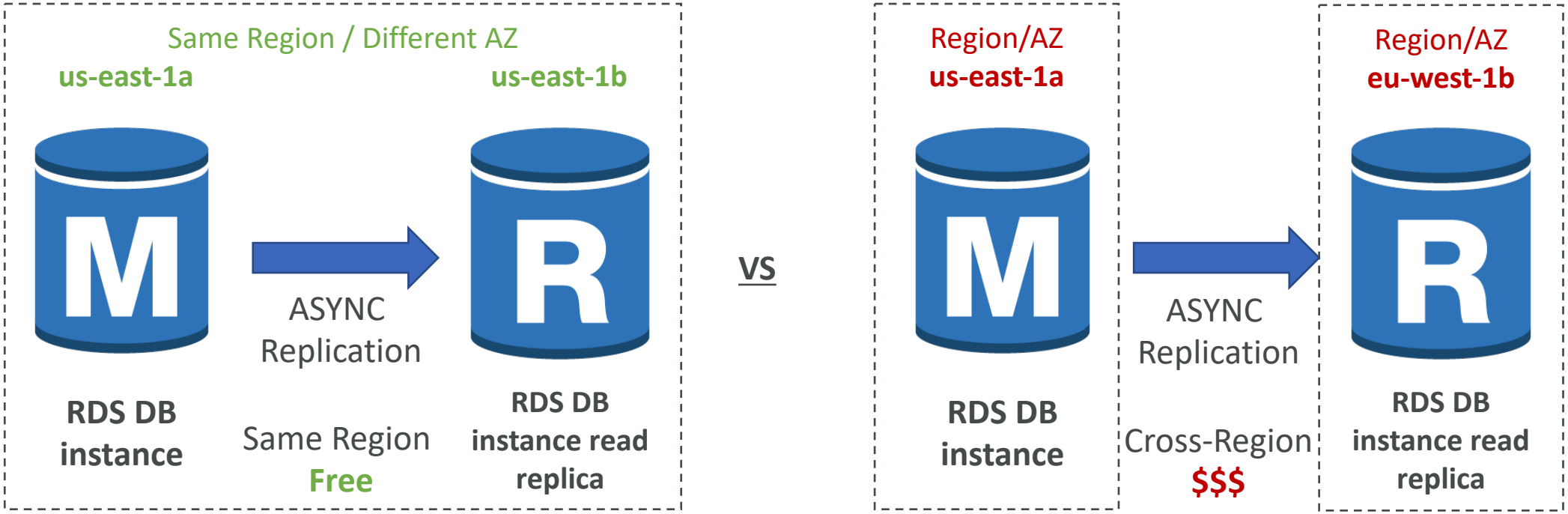
RDS Read Replicas – Use Cases

- You have a production database that is taking on normal load
- You want to run a reporting application to run some analytics
- You create a Read Replica to run the new workload there
- The production application is unaffected
- Read replicas are used for SELECT (=read) only kind of statements (not INSERT, UPDATE, DELETE)



RDS Read Replicas – Network Cost

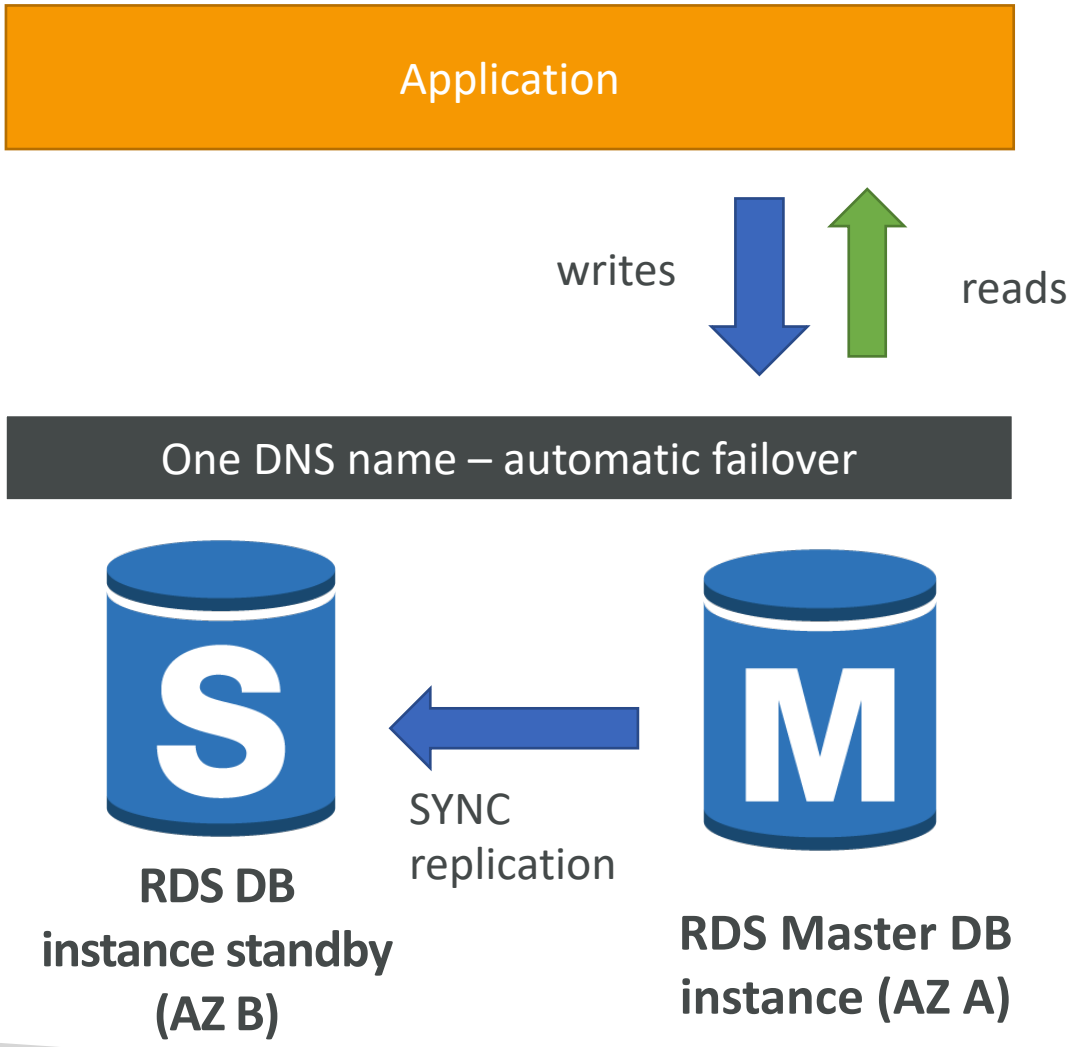
- In AWS there's a network cost when data goes from one AZ to another
- For RDS Read Replicas within the same region, you don't pay that fee



RDS Multi AZ (Disaster Recovery)

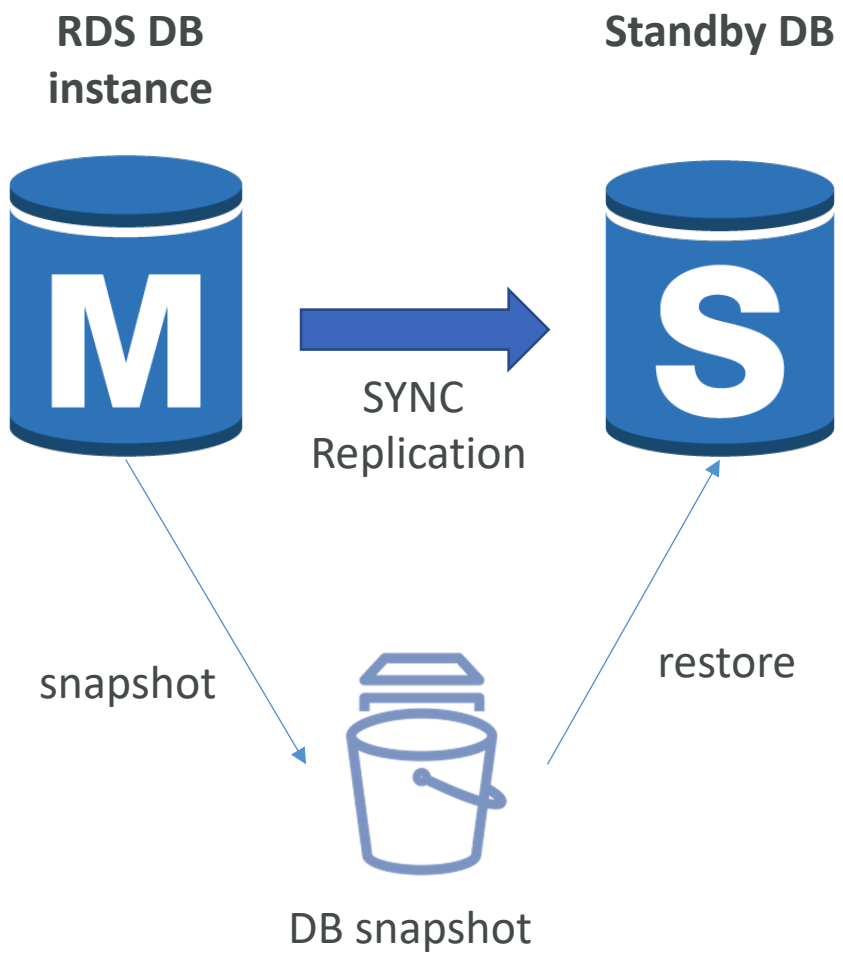
- **SYNC** replication
- One DNS name – automatic app failover to standby
- Increase **availability**
- Failover in case of loss of AZ, loss of network, instance or storage failure
- No manual intervention in apps
- Not used for scaling

- Note: The Read Replicas be setup as Multi AZ for Disaster Recovery (DR)



RDS – From Single-AZ to Multi-AZ

- Zero downtime operation (no need to stop the DB)
- Just click on "modify" for the database
- The following happens internally:
  - A snapshot is taken
  - A new DB is restored from the snapshot in a new AZ
  - Synchronization is established between the two databases

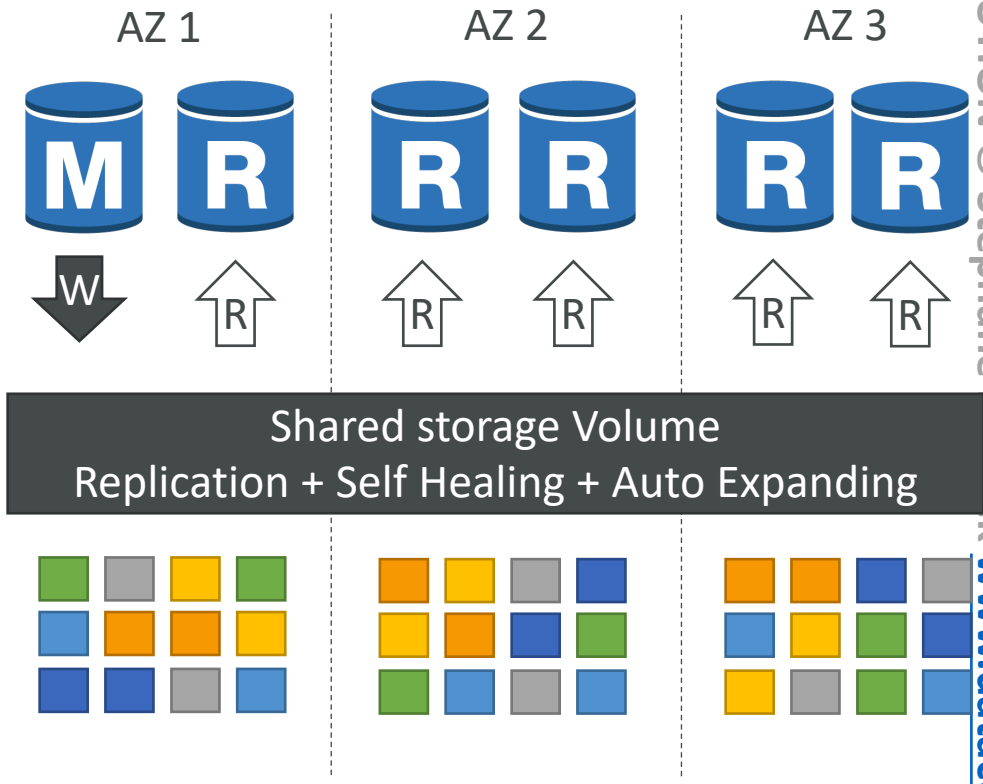


Amazon Aurora

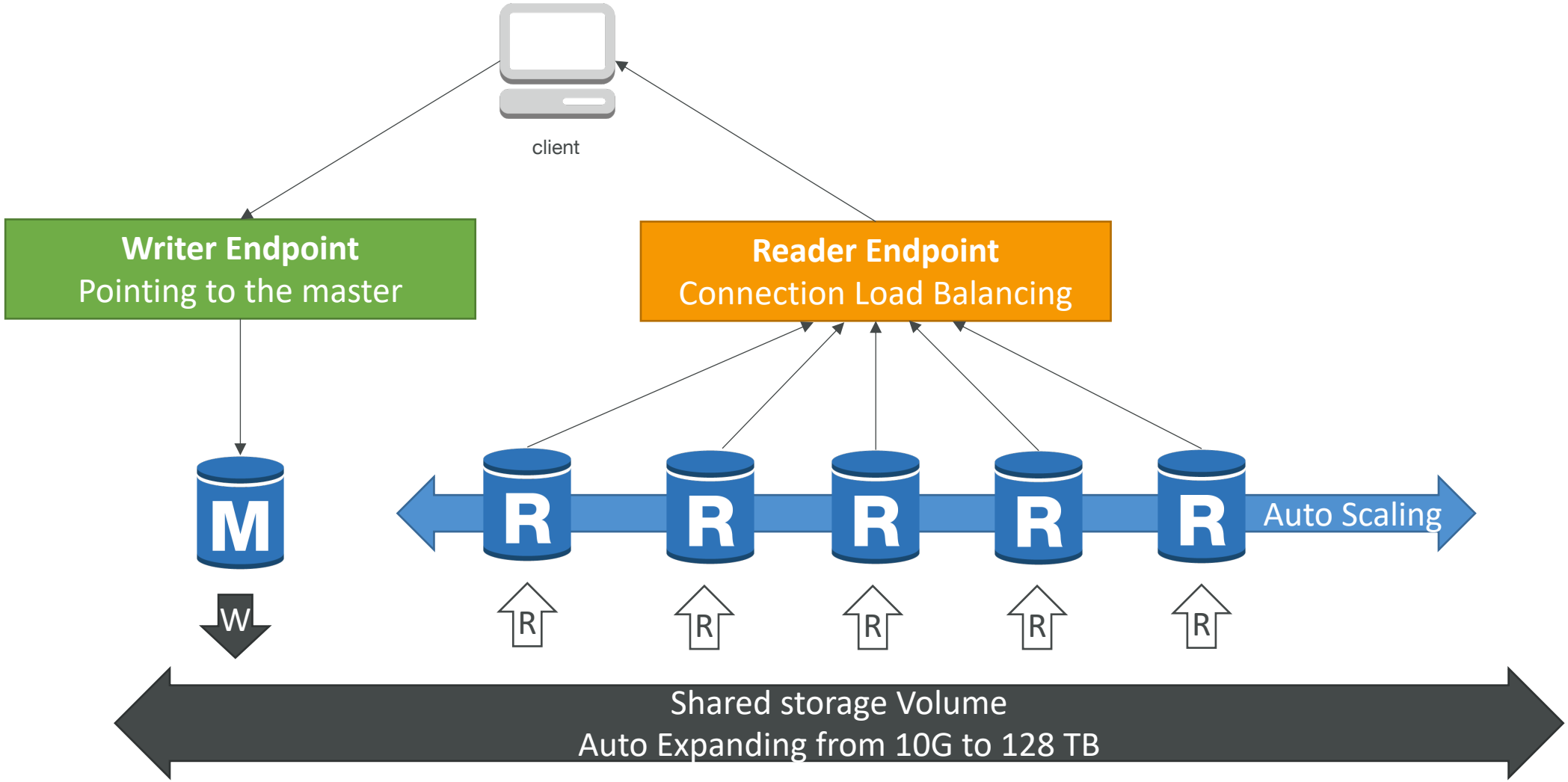
- Aurora is a proprietary technology from AWS (not open sourced)
- Postgres and MySQL are both supported as Aurora DB (that means your drivers will work as if Aurora was a Postgres or MySQL database)
- Aurora is "AWS cloud optimized" and claims 5x performance improvement over MySQL on RDS, over 3x the performance of Postgres on RDS
- Aurora storage automatically grows in increments of 10GB, up to 128 TB.
- Aurora can have up to 15 replicas and the replication process is faster than MySQL (sub 10 ms replica lag)
- Failover in Aurora is instantaneous. It's HA (High Availability) native.
- Aurora costs more than RDS (20% more) – but is more efficient

Aurora High Availability and Read Scaling

- 6 copies of your data across 3 AZ:
  - 4 copies out of 6 needed for writes
  - 3 copies out of 6 need for reads
  - Self healing with peer-to-peer replication
  - Storage is striped across 100s of volumes
- One Aurora Instance takes writes (master)
- Automated failover for master in less than 30 seconds
- Master + up to 15 Aurora Read Replicas serve reads
- Support for Cross Region Replication



Aurora DB Cluster



Features of Aurora

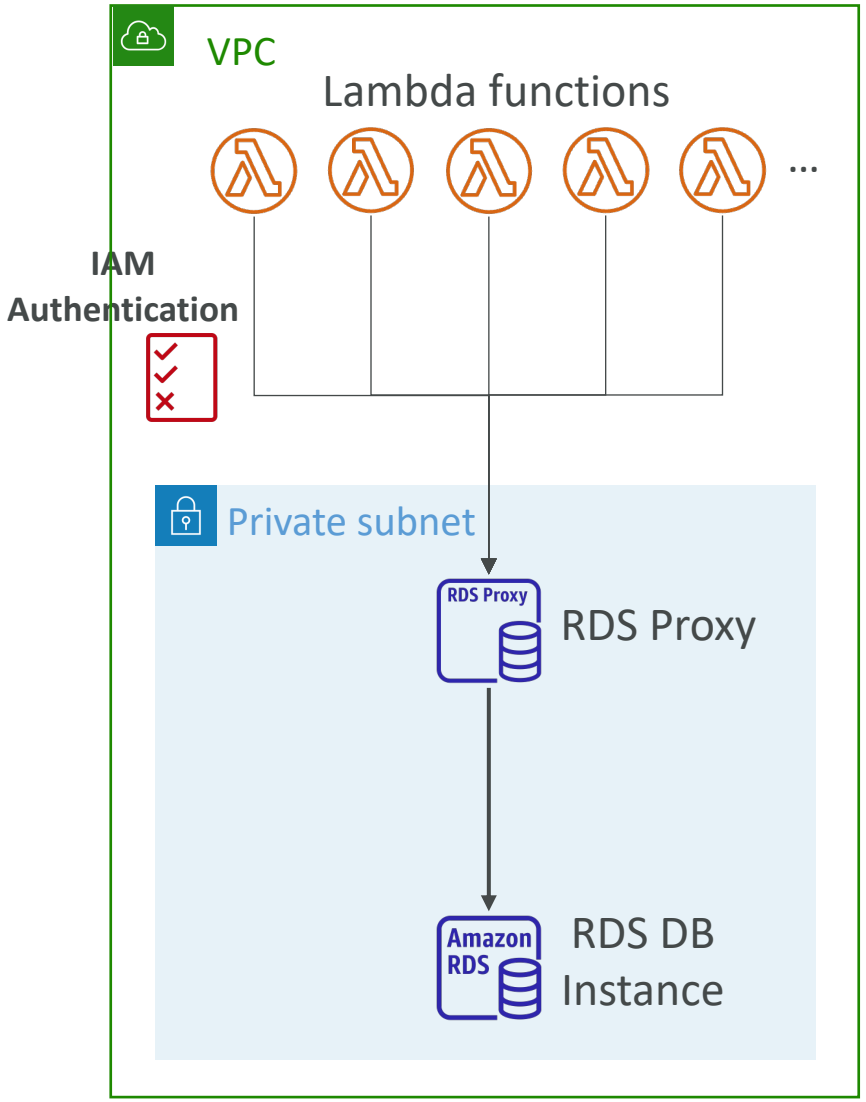
- Automatic fail-over
- Backup and Recovery
- Isolation and security
- Industry compliance
- Push-button scaling
- Automated Patching with Zero Downtime
- Advanced Monitoring
- Routine Maintenance
- Backtrack: restore data at any point of time without using backups

RDS & Aurora Security

- At-rest encryption:
  - Database master & replicas encryption using AWS KMS – must be defined as launch time
  - If the master is not encrypted, the read replicas cannot be encrypted
  - To encrypt an un-encrypted database, go through a DB snapshot & restore as encrypted
- In-flight encryption: TLS-ready by default, use the AWS TLS root certificates client-side
- IAM Authentication: IAM roles to connect to your database (instead of username/pw)
- Security Groups: Control Network access to your RDS / Aurora DB
- No SSH available except on RDS Custom
- Audit Logs can be enabled and sent to CloudWatch Logs for longer retention

Amazon RDS Proxy

- Fully managed database proxy for RDS
- Allows apps to pool and share DB connections established with the database
- Improving database efficiency by reducing the stress on database resources (e.g., CPU, RAM) and minimize open connections (and timeouts)
- Serverless, autoscaling, highly available (multi-AZ)
- Reduced RDS & Aurora failover time by up 66%
- Supports RDS (MySQL, PostgreSQL, MariaDB, MS SQL Server) and Aurora (MySQL, PostgreSQL)
- No code changes required for most apps
- Enforce IAM Authentication for DB, and securely store credentials in AWS Secrets Manager
- RDS Proxy is never publicly accessible (must be accessed from VPC)

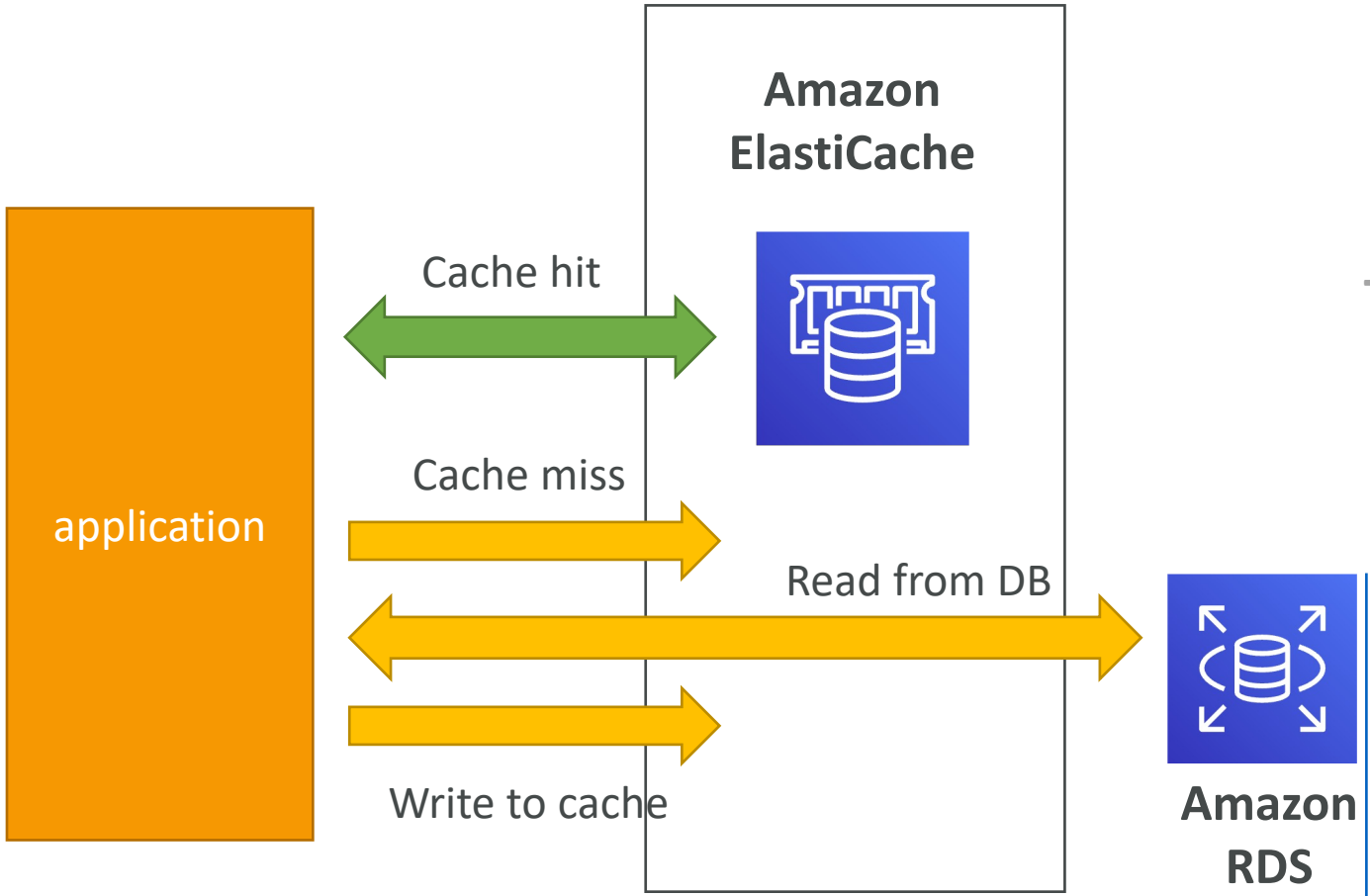


Amazon ElastiCache Overview

- The same way RDS is to get managed Relational Databases…
- ElastiCache is to get managed Redis or Memcached
- Caches are in-memory databases with really high performance, low latency
- Helps reduce load off of databases for read intensive workloads
- Helps make your application stateless
- AWS takes care of OS maintenance / patching, optimizations, setup, configuration, monitoring, failure recovery and backups
- Using ElastiCache involves heavy application code changes

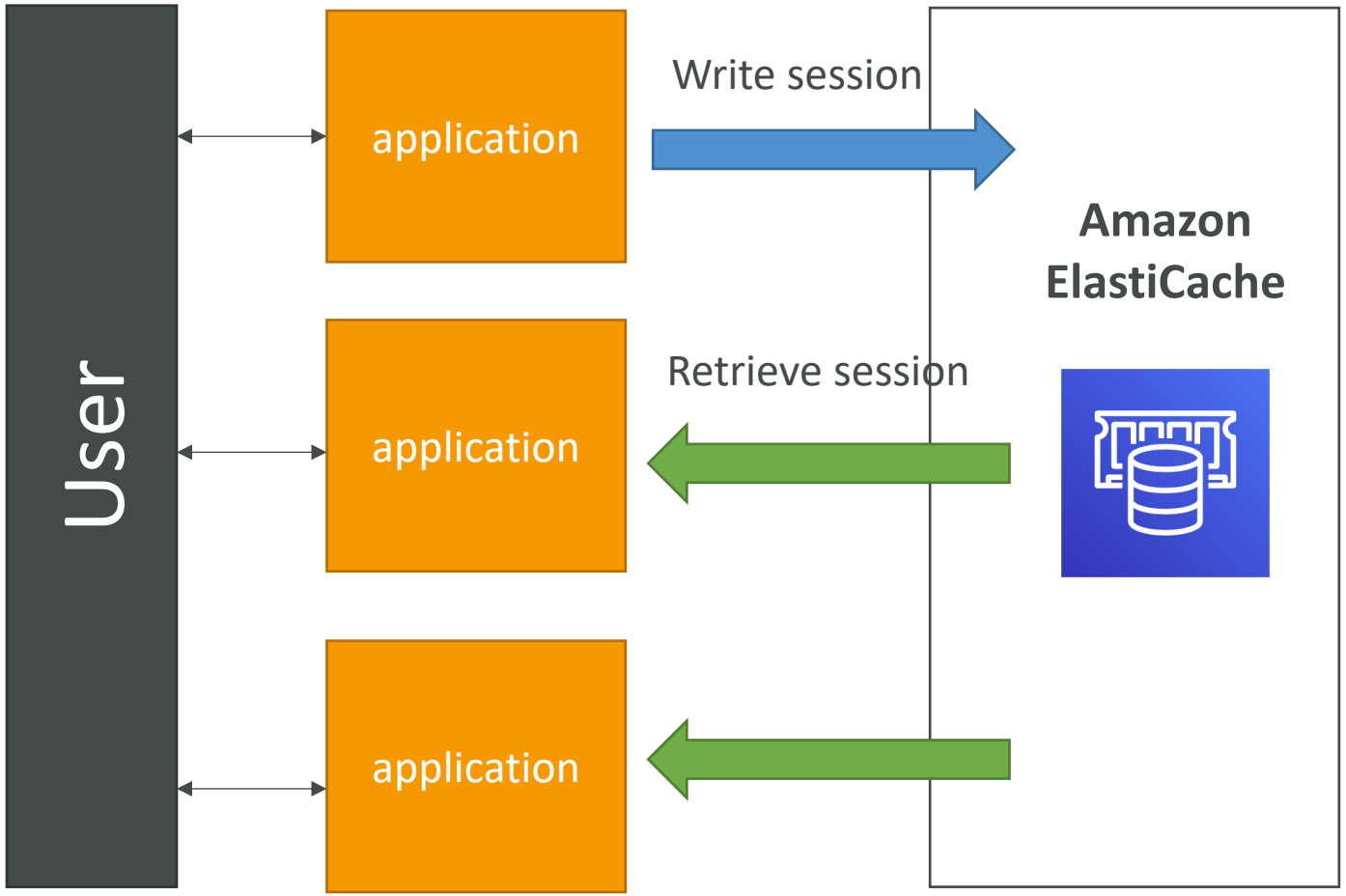
ElastiCache Solution Architecture - DB Cache

- Applications queries ElastiCache, if not available, get from RDS and store in ElastiCache.
- Helps relieve load in RDS
- Cache must have an invalidation strategy to make sure only the most current data is used in there.



ElastiCache
Solution Architecture – User Session Store

- User logs into any of the application
- The application writes the session data into ElastiCache
- The user hits another instance of our application
- The instance retrieves the data and the user is already logged in



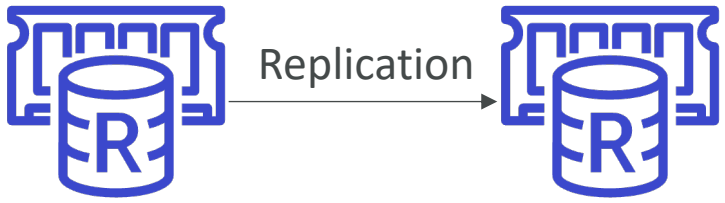
ElastiCache – Redis vs Memcached

REDIS

- **Multi AZ** with Auto-Failover
- **Read Replicas** to scale reads and have **high availability**
- Data Durability using AOF persistence
- **Backup and restore features**
- Supports Sets and Sorted Sets

MEMCACHED

- Multi-node for partitioning of data (sharding)
- No high availability (replication)
- Non persistent
- Backup and restore (Serverless)
- Multi-threaded architecture



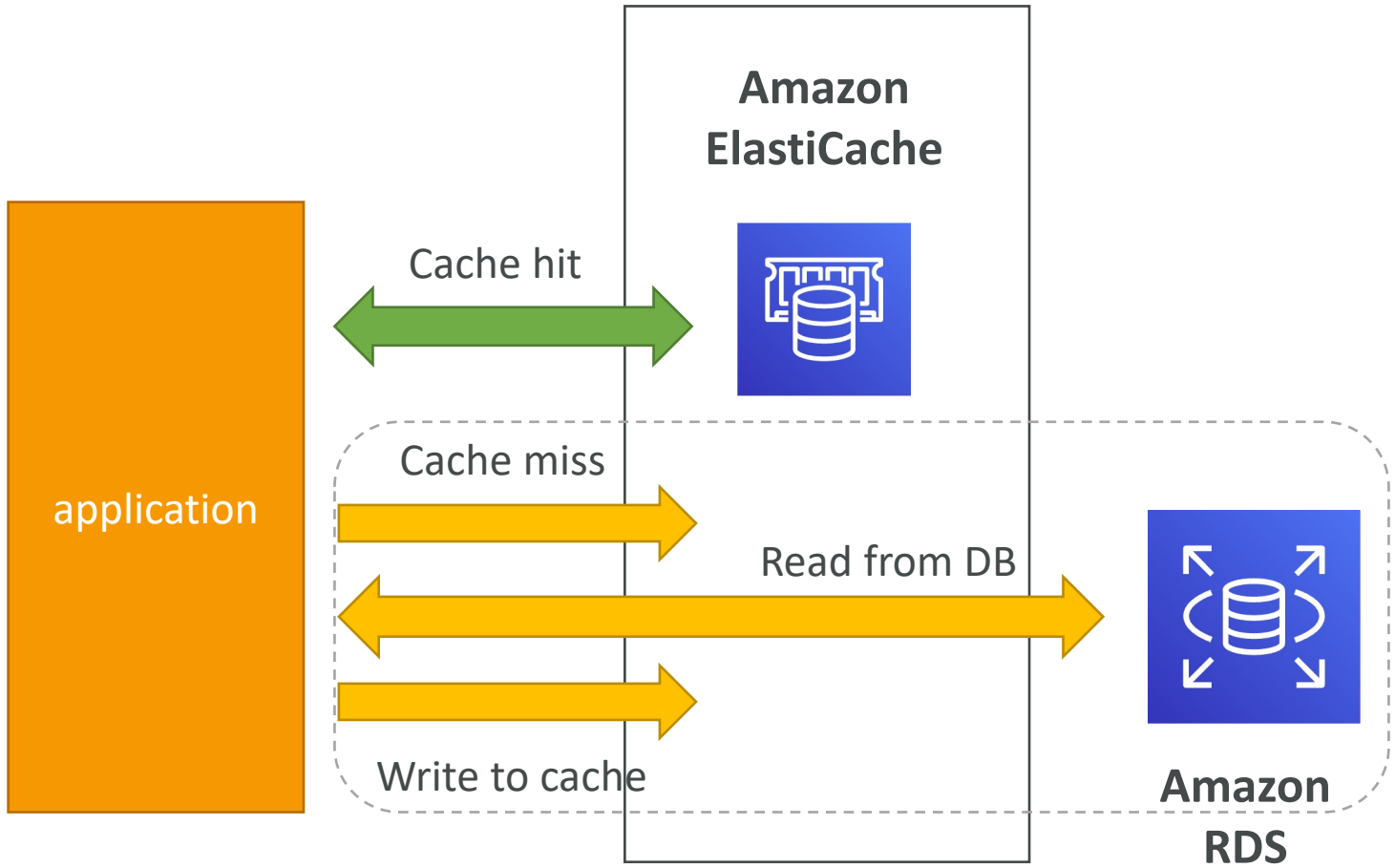
Caching Implementation Considerations

- Read more at: https://aws.amazon.com/caching/implementation-considerations/

- **Is it safe to cache data?** Data may be out of date, eventually consistent
- **Is caching effective for that data?**
  - Pattern: data changing slowly, few keys are frequently needed
  - Anti patterns: data changing rapidly, all large key space frequently needed
- **Is data structured well for caching?**
  - example: key value caching, or caching of aggregations results

- Which caching design pattern is the most appropriate?

Lazy Loading / Cache-Aside / Lazy Population

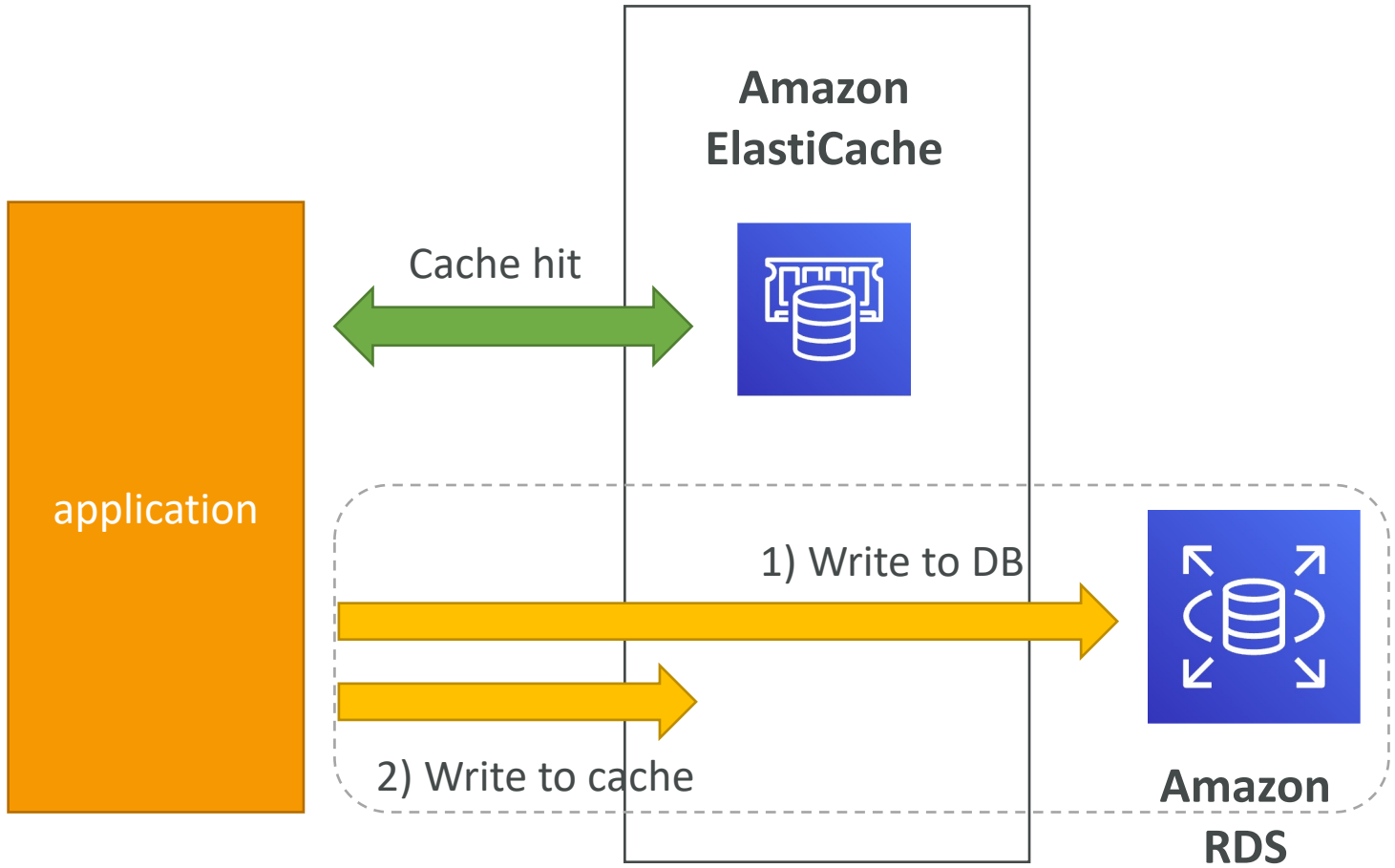


- Pros
  - Only requested data is cached (the cache isn't filled up with unused data)
  - Node failures are not fatal (just increased latency to warm the cache)
- Cons
  - Cache miss penalty that results in 3 round trips, noticeable delay for that request
  - Stale data: data can be updated in the database and outdated in the cache

Lazy Loading / Cache-Aside / Lazy Population Python Pseudocode

```python
# Python

def get_user(user_id):
    # Check the cache
    record = cache.get(user_id)

    if record is None:
        # Run a DB query
        record = db.query("select * from users where id = ?", user_id)
        # Populate the cache
        cache.set(user_id, record)
        return record
    else:
        return record

# App code
user = get_user(17)
```

Write Through – Add or Update cache when database is updated



- Pros:
  - Data in cache is never stale, **reads are quick**
  - **Write penalty** vs Read penalty (each write requires 2 calls)
- Cons:
  - Missing Data until it is added / updated in the DB. Mitigation is to implement Lazy Loading strategy as well
  - Cache churn – a lot of the data will never be read

Write-Through Python Pseudocode

```python
# Python

def save_user(user_id, values):

    # Save to DB

    record = db.query("update users ... where id = ?", user_id, values)

    # Push into cache

    cache.set(user_id, record)

    return record

# App code

user = save_user(17, {"name": "Nate Dogg"})
```

Cache Evictions and Time-to-live (TTL)

- Cache eviction can occur in three ways:
  - You delete the item explicitly in the cache
  - Item is evicted because the memory is full and it's not recently used (LRU)
  - You set an item **time-to-live (or TTL)**
- TTL are helpful for any kind of data:
  - Leaderboards
  - Comments
  - Activity streams
- TTL can range from few seconds to hours or days

- If too many evictions happen due to memory, you should scale up or out

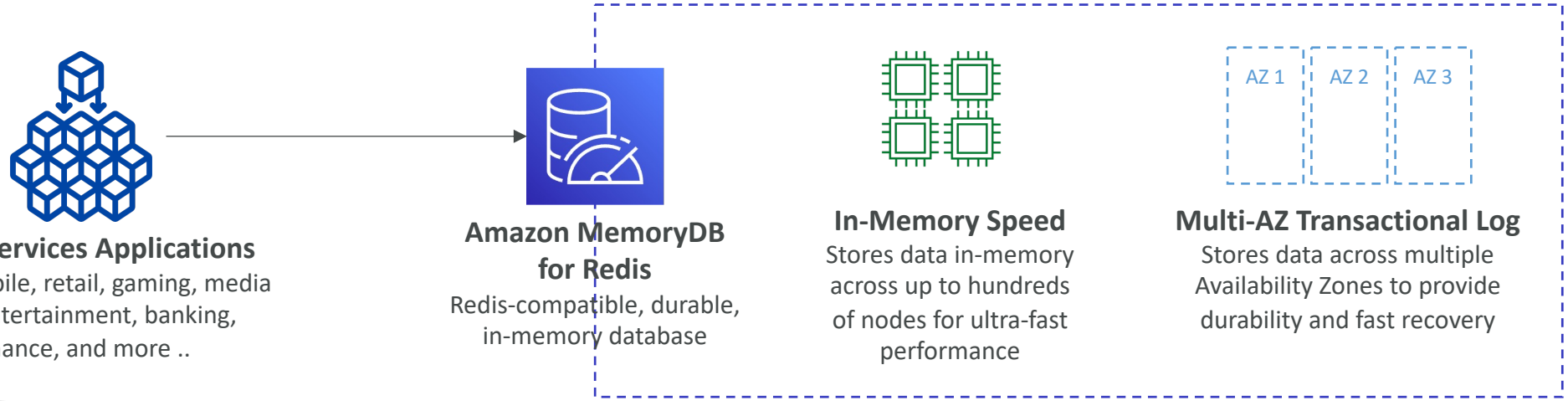
Final words of wisdom

- Lazy Loading / Cache aside is easy to implement and works for many situations as a foundation, especially on the read side
- Write-through is usually combined with Lazy Loading as targeted for the queries or workloads that benefit from this optimization
- Setting a TTL is usually not a bad idea, except when you're using Write-through. Set it to a sensible value for your application
- Only cache the data that makes sense (user profiles, blogs, etc…)

- *Quote: There are only two hard things in Computer Science: cache invalidation and naming things*

Amazon MemoryDB for Redis

- Redis-compatible, **durable, in-memory database service**
- **Ultra-fast performance with over 160 millions requests/second**
- Durable in-memory data storage with Multi-AZ transactional log
- Scale seamlessly from 10s GBs to 100s TBs of storage
- Use cases: web and mobile apps, online gaming, media streaming, …



Amazon Route 53

What is DNS?

- Domain Name System which translates the human friendly hostnames into the machine IP addresses
- www.google.com => 172.217.18.36
- DNS is the backbone of the Internet
- DNS uses hierarchical naming structure

.com

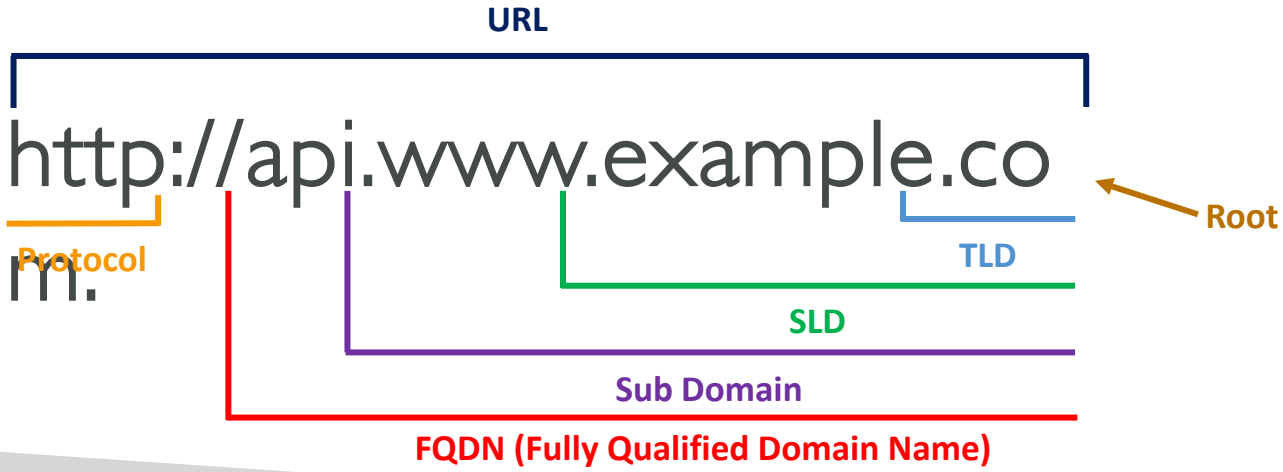
example.com

www.example.com

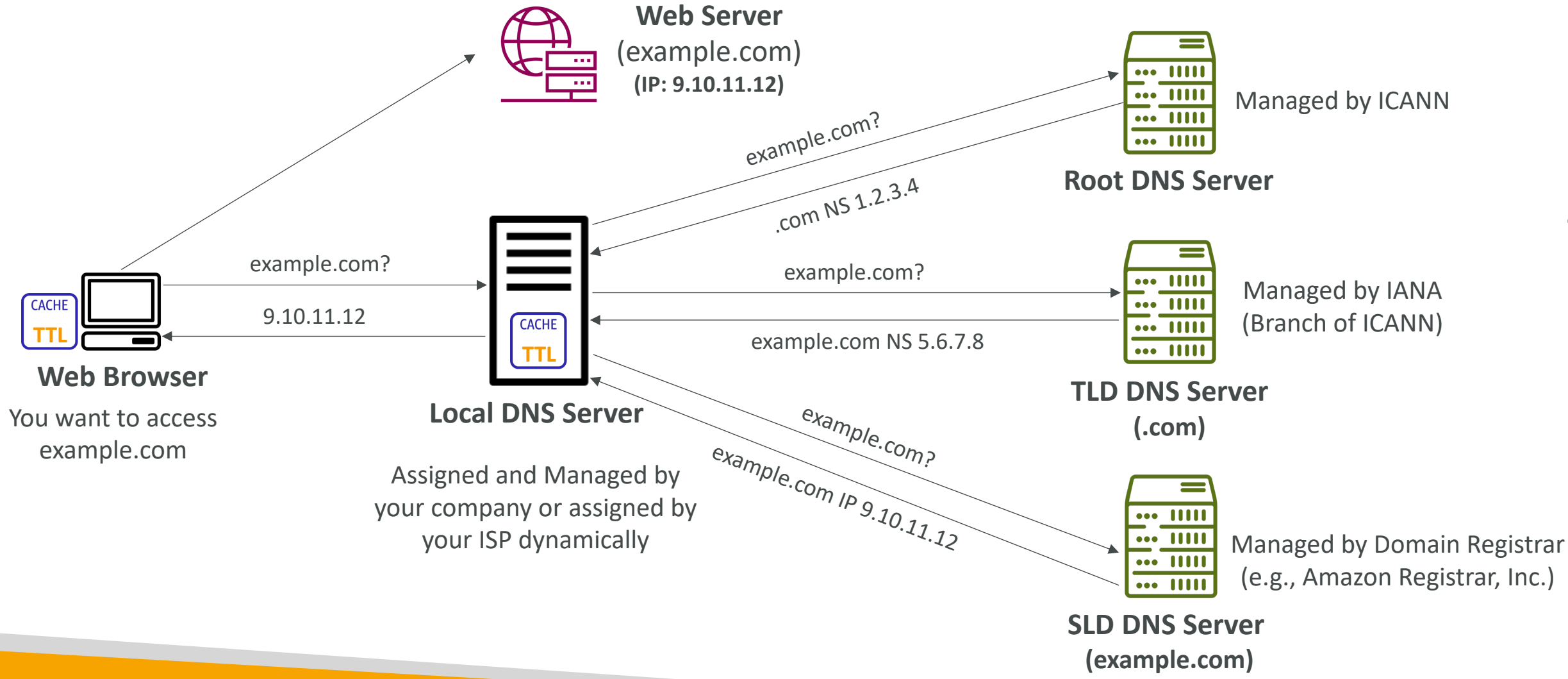
api.example.com

DNS Terminologies

- **Domain Registrar:** Amazon Route 53, GoDaddy, …
- **DNS Records:** A, AAAA, CNAME, NS, …
- **Zone File:** contains DNS records
- **Name Server:** resolves DNS queries (Authoritative or Non-Authoritative)
- **Top Level Domain (TLD):** .com, .us, .in, .gov, .org, …
- **Second Level Domain (SLD):** amazon.com, google.com, …

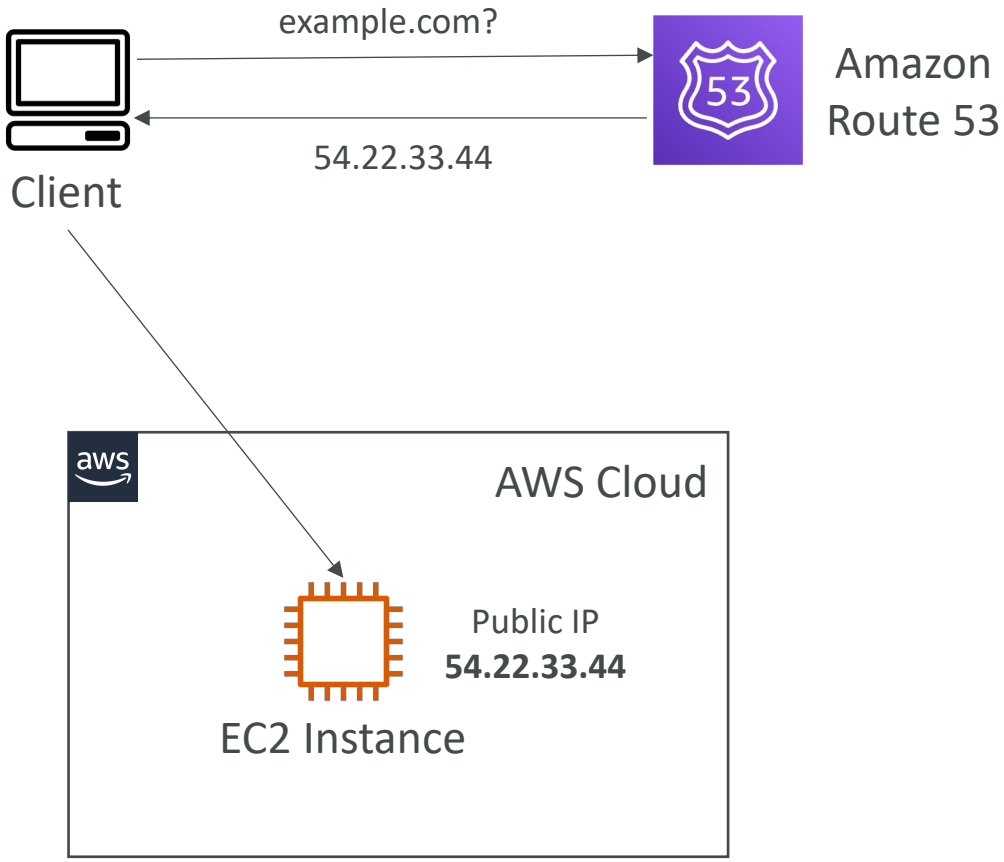


How DNS Works



Amazon Route 53

- A highly available, scalable, fully managed and *Authoritative* DNS
  - Authoritative = the customer (you) can update the DNS records
- Route 53 is also a Domain Registrar
- Ability to check the health of your resources
- The only AWS service which provides 100% availability SLA
- Why Route 53? 53 is a reference to the traditional DNS port



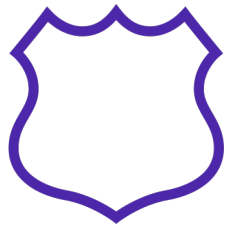
Route 53 – Records

- How you want to route traffic for a domain
- Each record contains:
  - **Domain/subdomain Name** – e.g., example.com
  - **Record Type** – e.g., A or AAAA
  - **Value** – e.g., 12.34.56.78
  - **Routing Policy** – how Route 53 responds to queries
  - **TTL** – amount of time the record cached at DNS Resolvers
- Route 53 supports the following DNS record types:
  - (must know) A / AAAA / CNAME / NS
  - (advanced) CAA / DS / MX / NAPTR / PTR / SOA / TXT / SPF / SRV

Route 53 – Record Types

- A – maps a hostname to IPv4
- AAAA – maps a hostname to IPv6
- CNAME – maps a hostname to another hostname
  - The target is a domain name which must have an A or AAAA record
  - Can't create a CNAME record for the top node of a DNS namespace (Zone Apex)
  - Example: you can't create for example.com, but you can create for www.example.com
- NS – Name Servers for the Hosted Zone
  - Control how traffic is routed for a domain

Route 53 – Hosted Zones



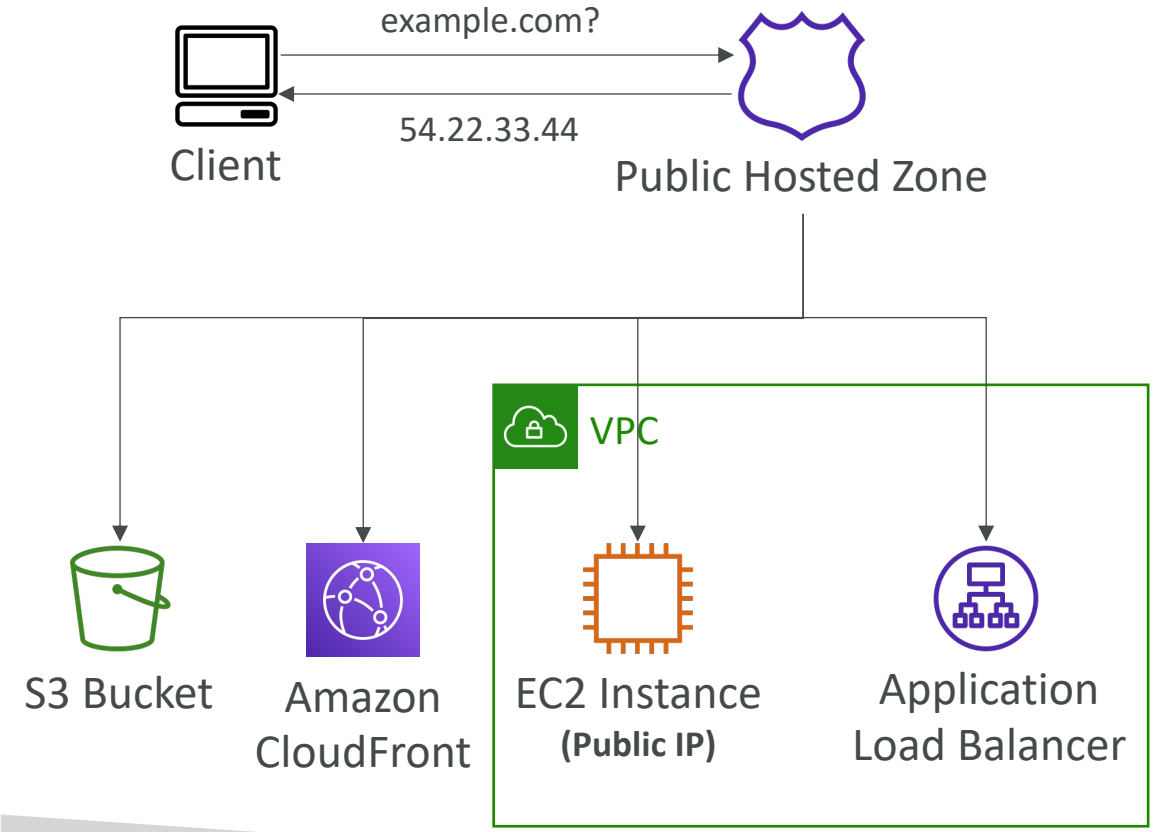
- A container for records that define how to route traffic to a domain and its subdomains

- **Public Hosted Zones** – contains records that specify how to route traffic on the Internet (public domain names)
  application1.mypublicdomain.com
- **Private Hosted Zones** – contain records that specify how you route traffic within one or more VPCs (private domain names)
  application1.company.internal

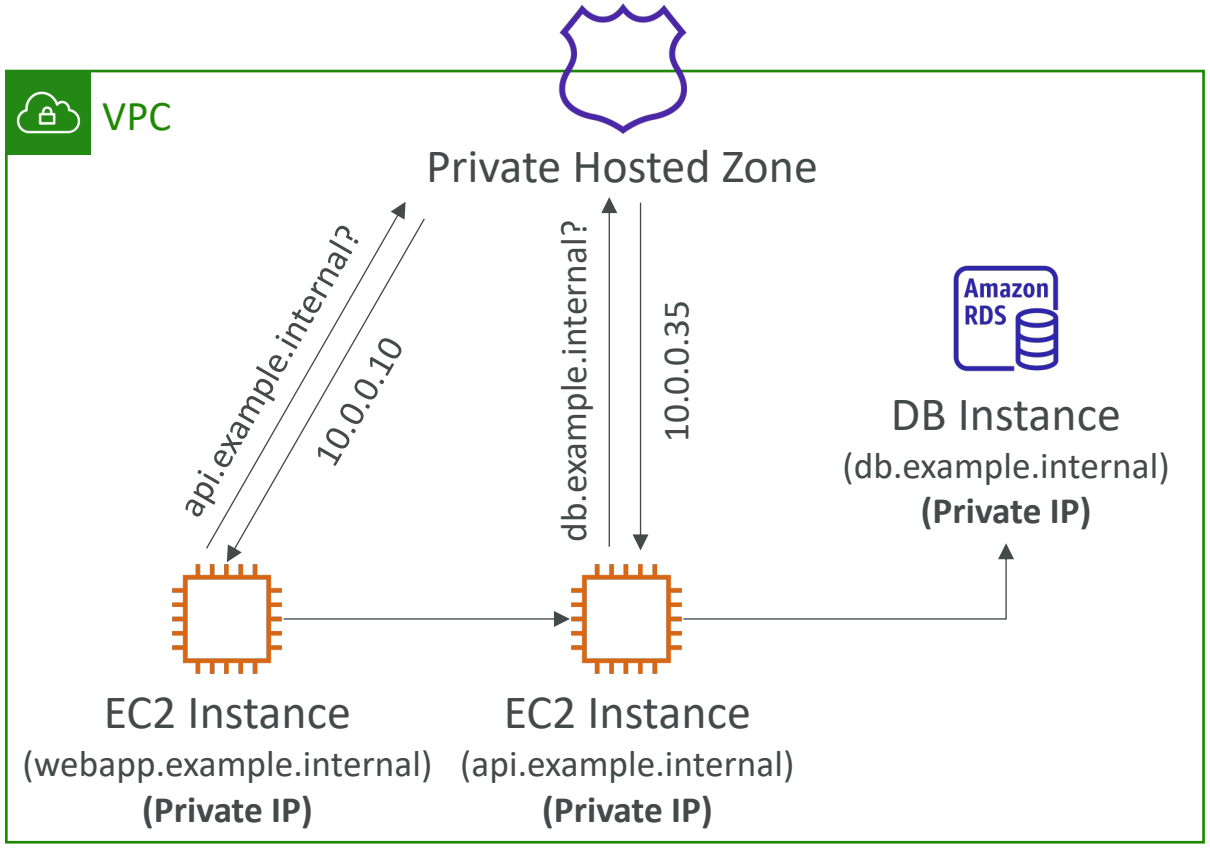
- You pay $0.50 per month per hosted zone

Route 53 – Public vs. Private Hosted Zones

Public Hosted Zone

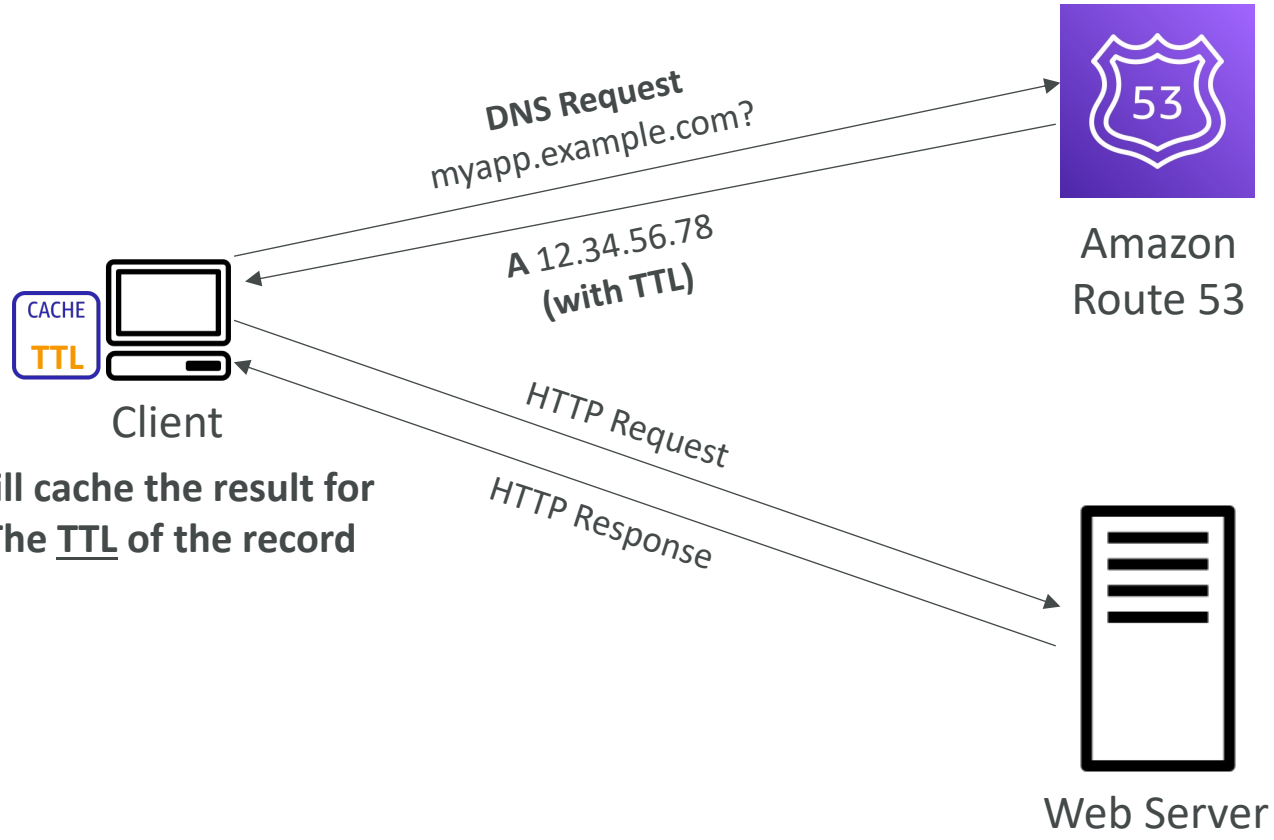


Private Hosted Zone



Route 53 – Records TTL (Time To Live)

- High TTL – e.g., 24 hr
  - Less traffic on Route 53
  - Possibly outdated records
- Low TTL – e.g., 60 sec.
  - More traffic on Route 53 ($$)
  - Records are outdated for less time
  - Easy to change records
- Except for Alias records, TTL is mandatory for each DNS record



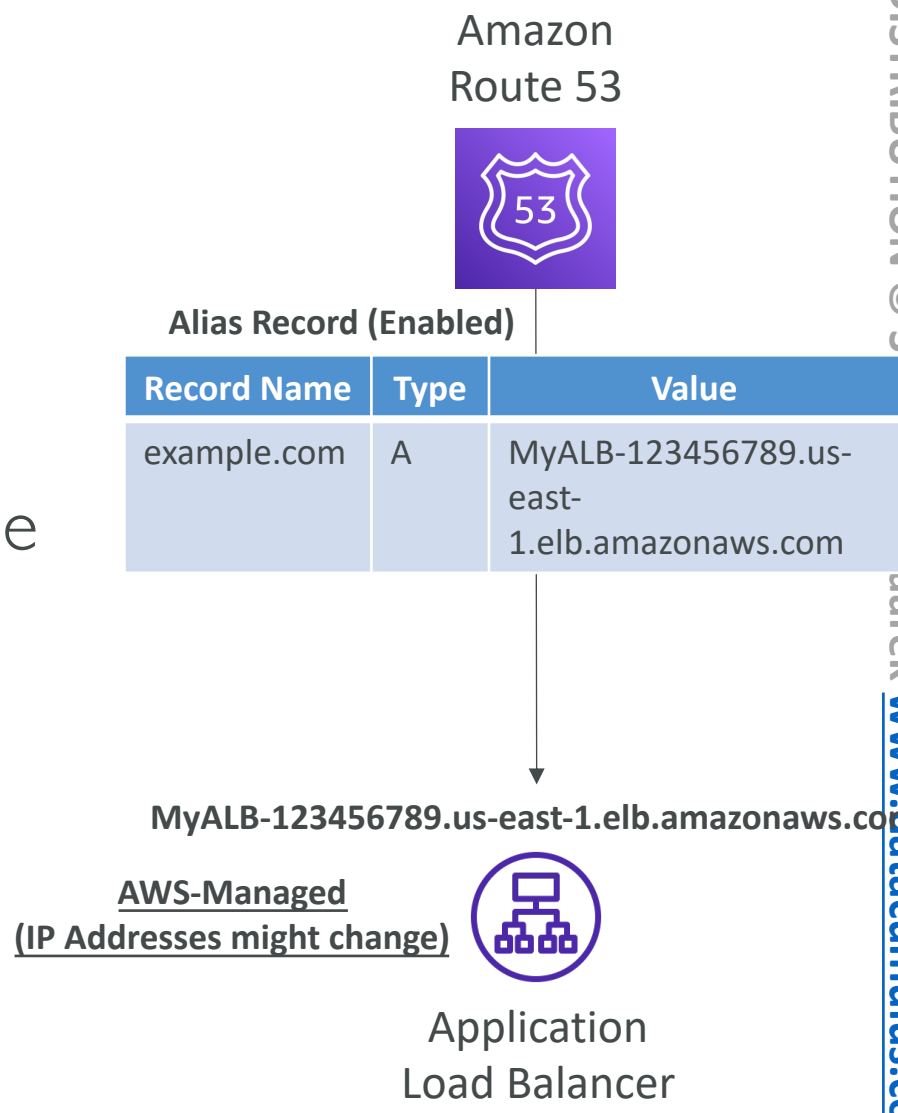
CNAME vs Alias

- AWS Resources (Load Balancer, CloudFront...) expose an AWS hostname:
  - lb1-1234.us-east-2.elb.amazonaws.com and you want myapp.mydomain.com

- CNAME:
  - Points a hostname to any other hostname. (app.mydomain.com => blabla.anything.com)
  - ONLY FOR NON ROOT DOMAIN (aka. something.mydomain.com)
- Alias:
  - Points a hostname to an AWS Resource (app.mydomain.com => blabla.amazonaws.com)
  - Works for ROOT DOMAIN and NON ROOT DOMAIN (aka mydomain.com)
  - Free of charge
  - Native health check

Route 53 – Alias Records

- Maps a hostname to an AWS resource
- An extension to DNS functionality
- Automatically recognizes changes in the resource's IP addresses
- Unlike CNAME, it can be used for the top node of a DNS namespace (Zone Apex), e.g.: example.com
- Alias Record is always of type A/AAAA for AWS resources (IPv4 / IPv6)
- You can't set the TTL

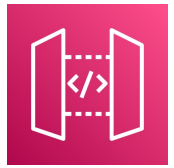


Route 53 – Alias Records Targets

- Elastic Load Balancers
- CloudFront Distributions
- API Gateway
- Elastic Beanstalk environments
- S3 Websites
- VPC Interface Endpoints
- Global Accelerator accelerator
- Route 53 record in the same hosted zone

Elastic
Load Balancer

Amazon
CloudFront



Amazon
API Gateway

Elastic Beanstalk

S3 Websites

VPC Interface
Endpoints

Global Accelerator

Route 53 Record
(same Hosted Zone)

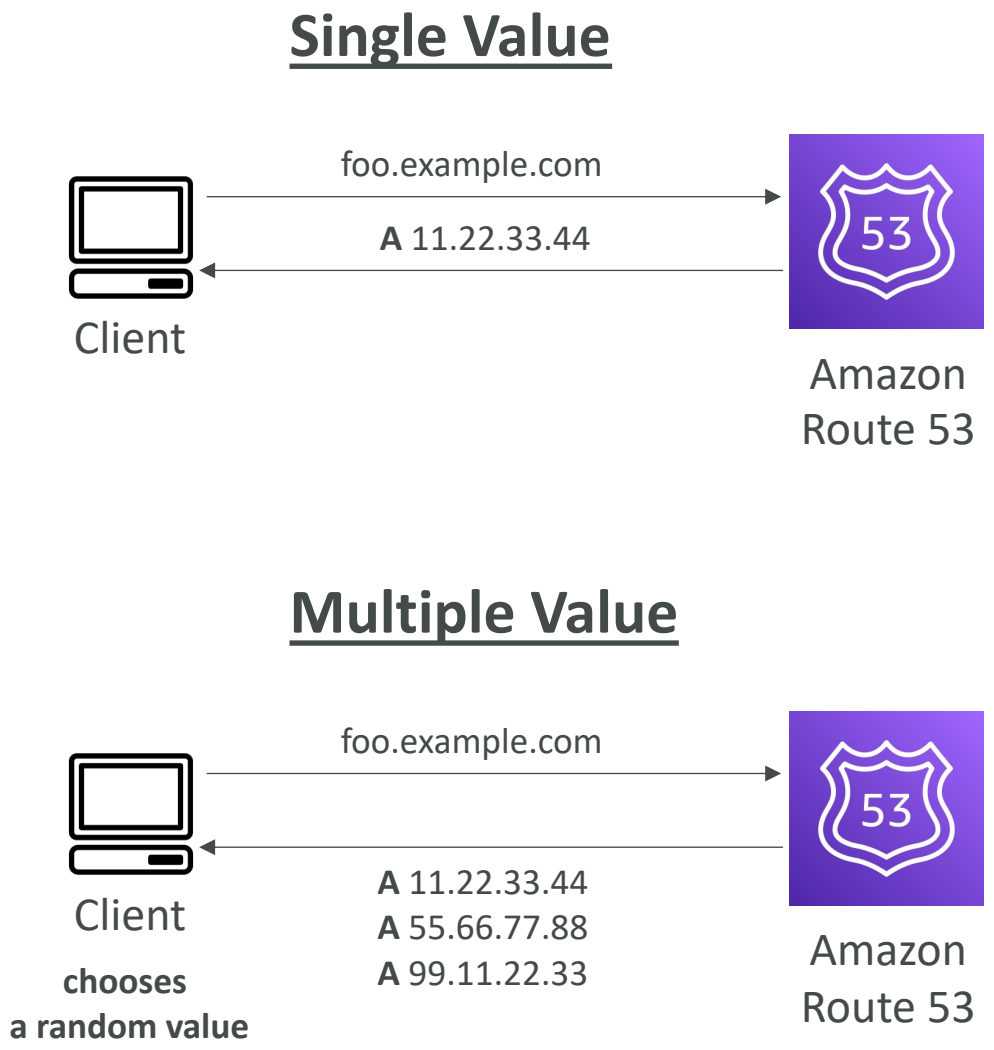
- You cannot set an ALIAS record for an EC2 DNS name

Route 53 – Routing Policies

- Define how Route 53 responds to DNS queries
- Don't get confused by the word *"Routing"*
  - It's not the same as Load balancer routing which routes the traffic
  - DNS does not route any traffic, it only responds to the DNS queries
- Route 53 Supports the following Routing Policies
  - Simple
  - Weighted
  - Failover
  - Latency based
  - Geolocation
  - Multi-Value Answer
  - Geoproximity (using Route 53 Traffic Flow feature)

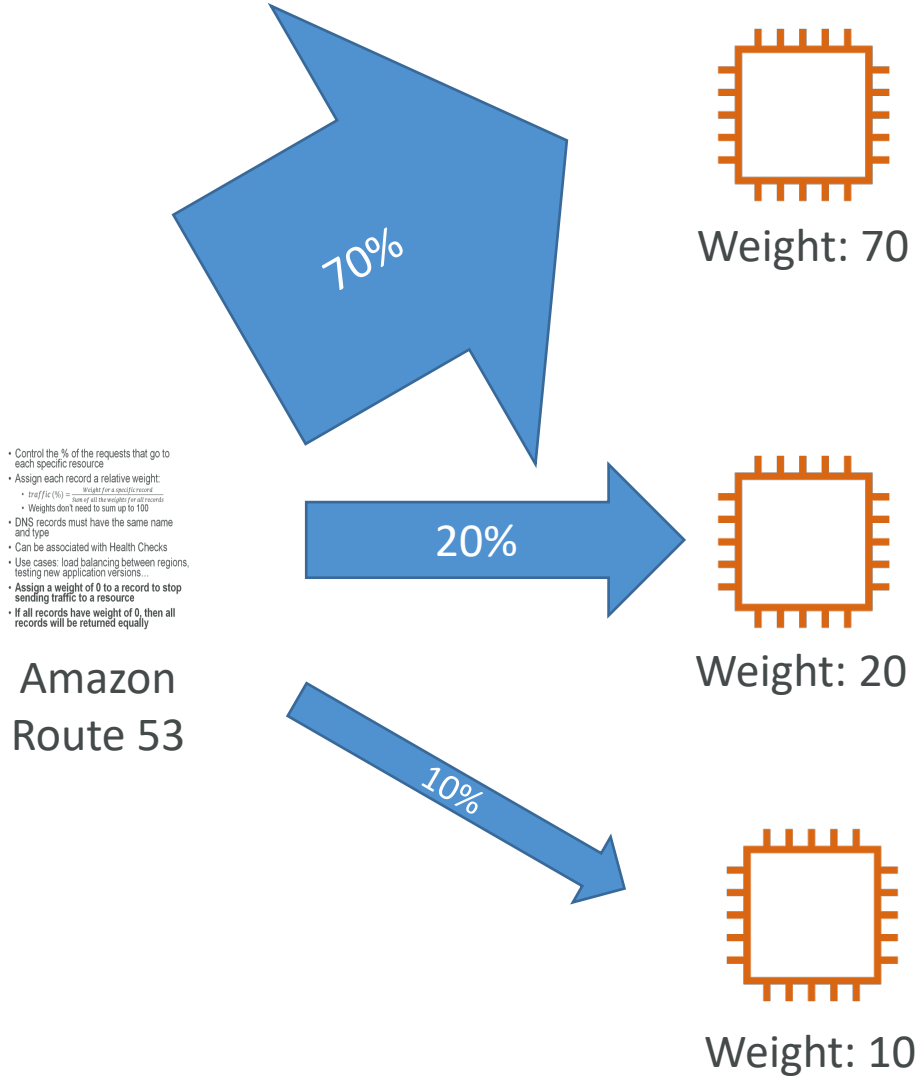
Routing Policies – Simple

- Typically, route traffic to a single resource
- Can specify multiple values in the same record
- **If multiple values are returned, a random one is chosen by the client**
- When Alias enabled, specify only one AWS resource
- Can't be associated with Health Checks



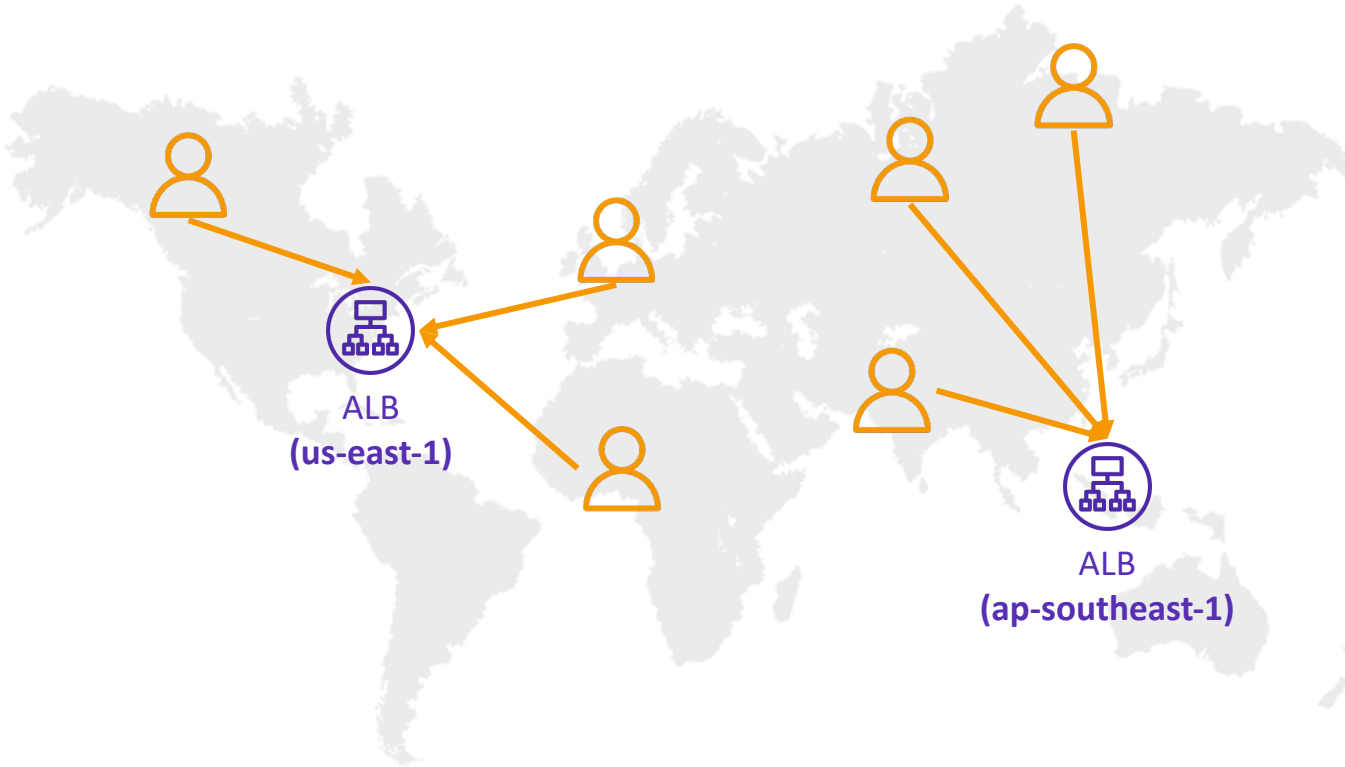
Routing Policies – Weighted

- Control the % of the requests that go to each specific resource
- Assign each record a relative weight:
  - $traffic\ (\%) = \frac{Weight\ for\ a\ specific\ record}{Sum\ of\ all\ the\ weights\ for\ all\ records}$
  - Weights don't need to sum up to 100
- DNS records must have the same name and type
- Can be associated with Health Checks
- Use cases: load balancing between regions, testing new application versions…
- Assign a weight of 0 to a record to stop sending traffic to a resource
- If all records have weight of 0, then all records will be returned equally



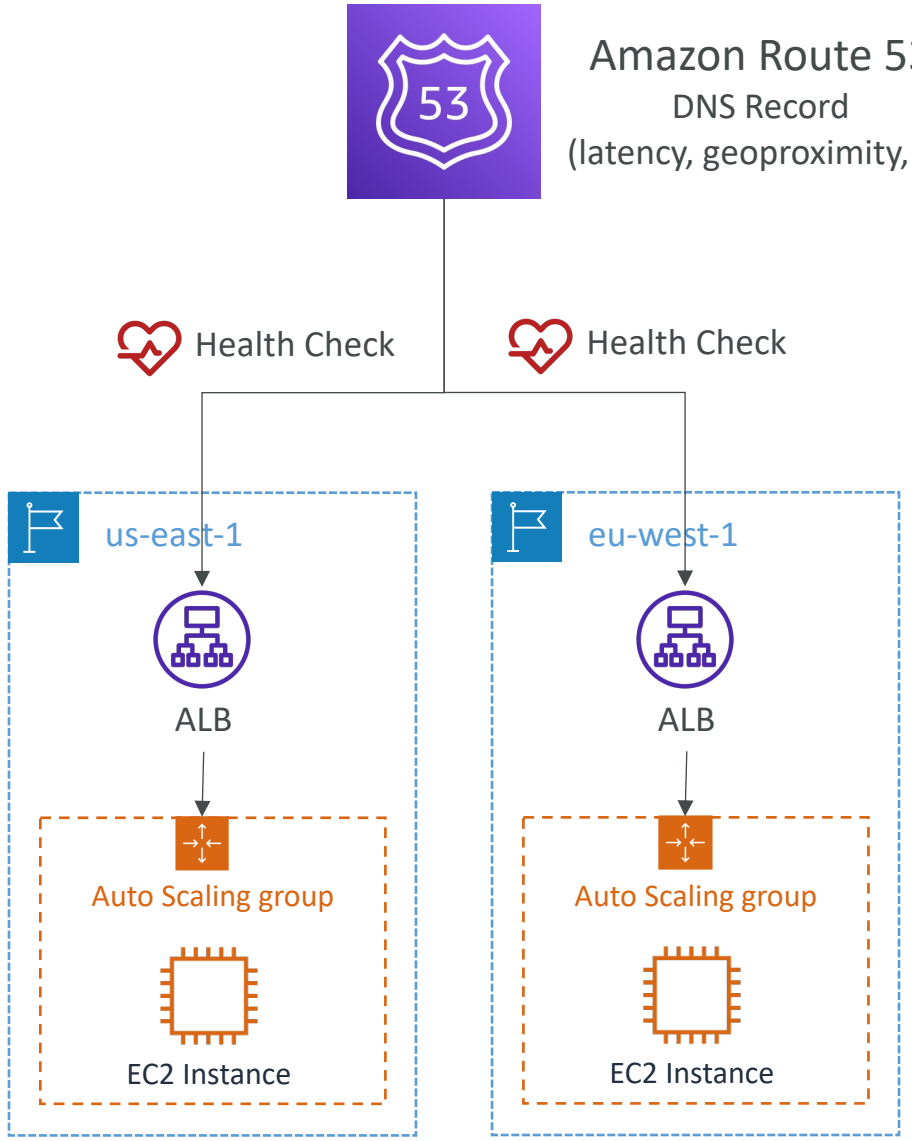
Routing Policies – Latency-based

- Redirect to the resource that has the least latency close to us
- Super helpful when latency for users is a priority
- **Latency is based on traffic between users and AWS Regions**
- Germany users may be directed to the US (if that's the lowest latency)
- Can be associated with Health Checks (has a failover capability)



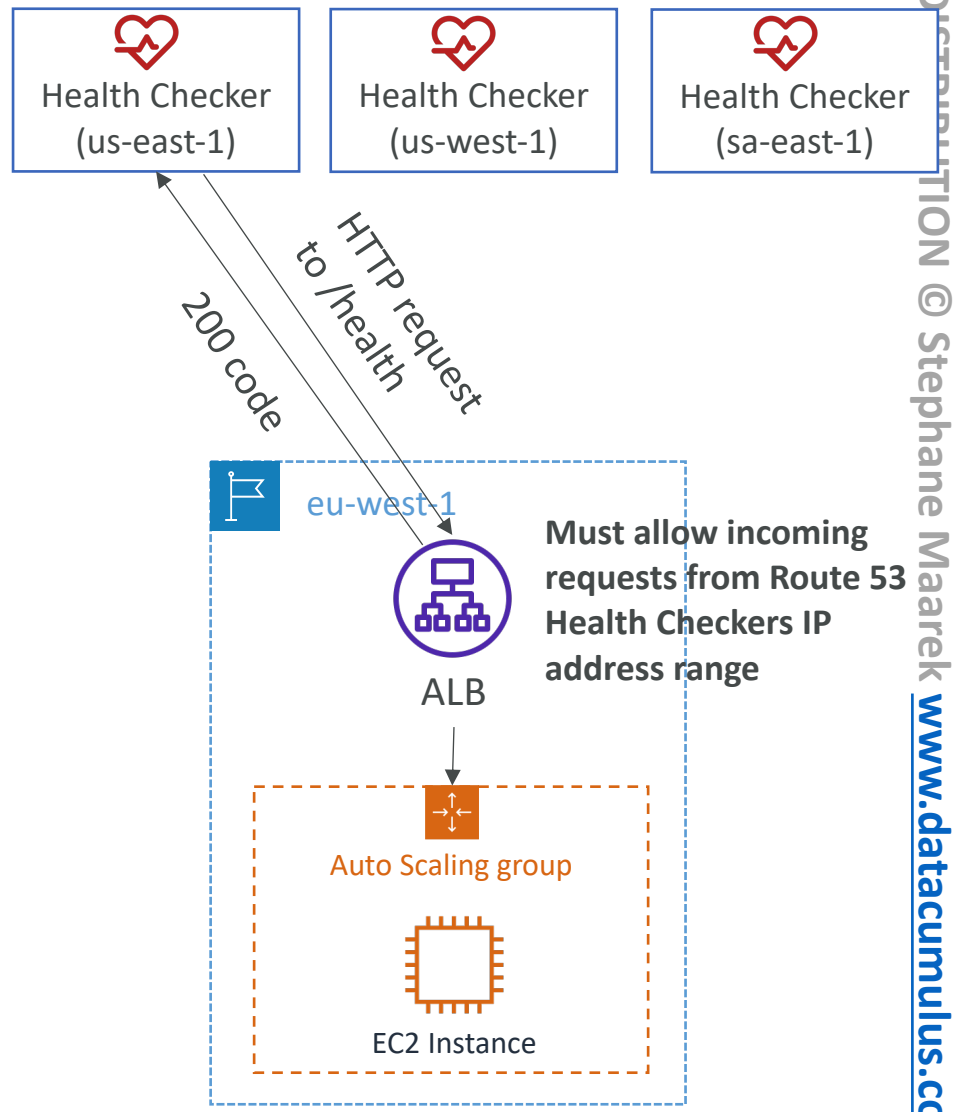
Route 53 – Health Checks

- HTTP Health Checks are only for **public resources**
- Health Check => Automated DNS Failover:
  1. Health checks that monitor an endpoint (application, server, other AWS resource)
  2. Health checks that monitor other health checks (Calculated Health Checks)
  3. Health checks that monitor CloudWatch Alarms (full control !!) – e.g., throttles of DynamoDB, alarms on RDS, custom metrics, … (helpful for private resources)
- Health Checks are integrated with CW metrics



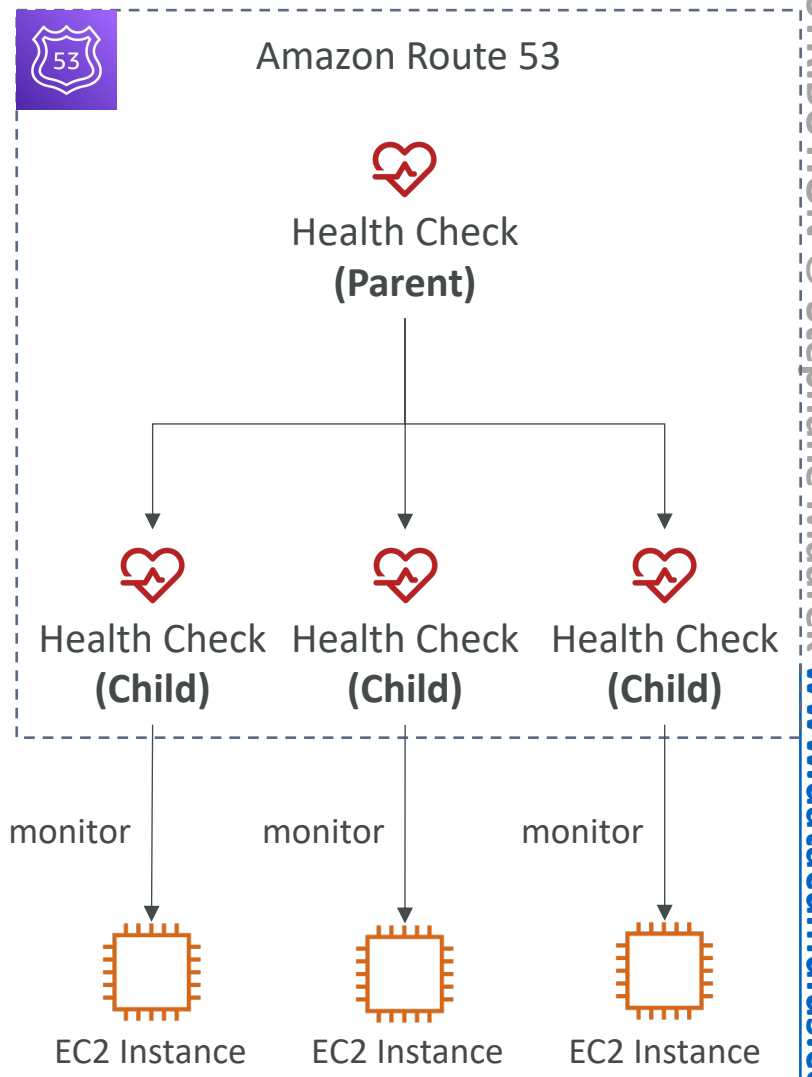
Health Checks – Monitor an Endpoint

- About 15 global health checkers will check the endpoint health
  - Healthy/Unhealthy Threshold – 3 (default)
  - Interval – 30 sec (can set to 10 sec – higher cost)
  - Supported protocol: HTTP, HTTPS and TCP
  - If > 18% of health checkers report the endpoint is healthy, Route 53 considers it **Healthy**. Otherwise, it's **Unhealthy**
  - Ability to choose which locations you want Route 53 to use
- Health Checks pass only when the endpoint responds with the 2xx and 3xx status codes
- Health Checks can be setup to pass / fail based on the text in the first **5120 bytes** of the response
- Configure you router/firewall to allow incoming requests from Route 53 Health Checkers



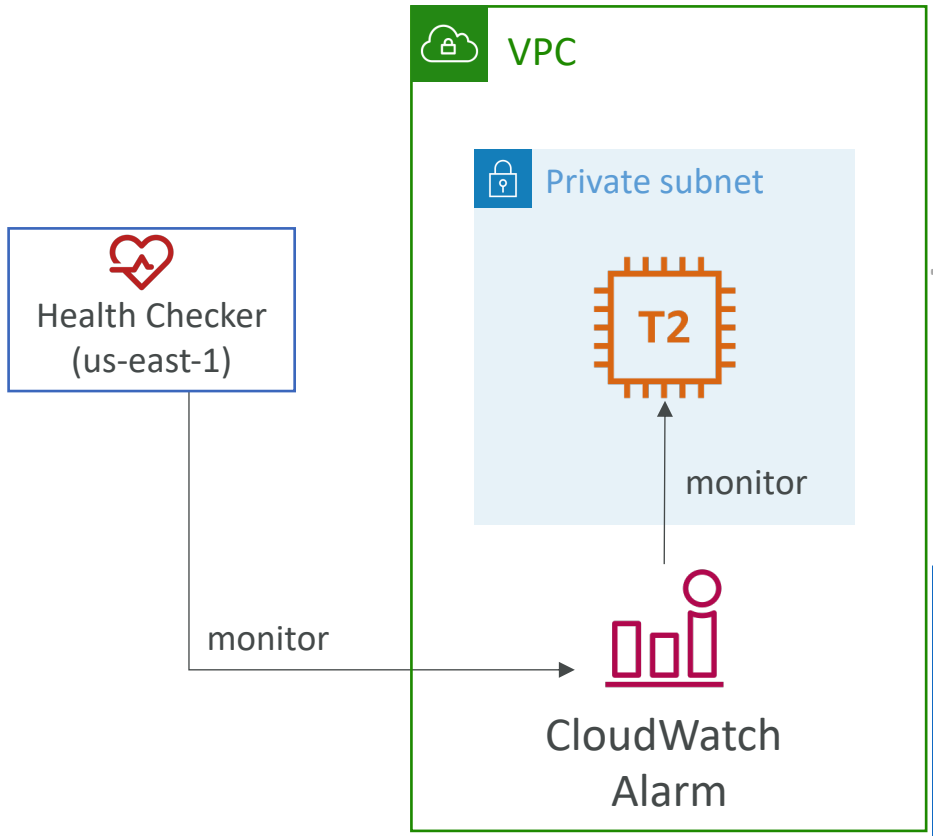
Route 53 – Calculated Health Checks

- Combine the results of multiple Health Checks into a single Health Check
- You can use **OR**, **AND**, or **NOT**
- Can monitor up to 256 Child Health Checks
- Specify how many of the health checks need to pass to make the parent pass
- Usage: perform maintenance to your website without causing all health checks to fail

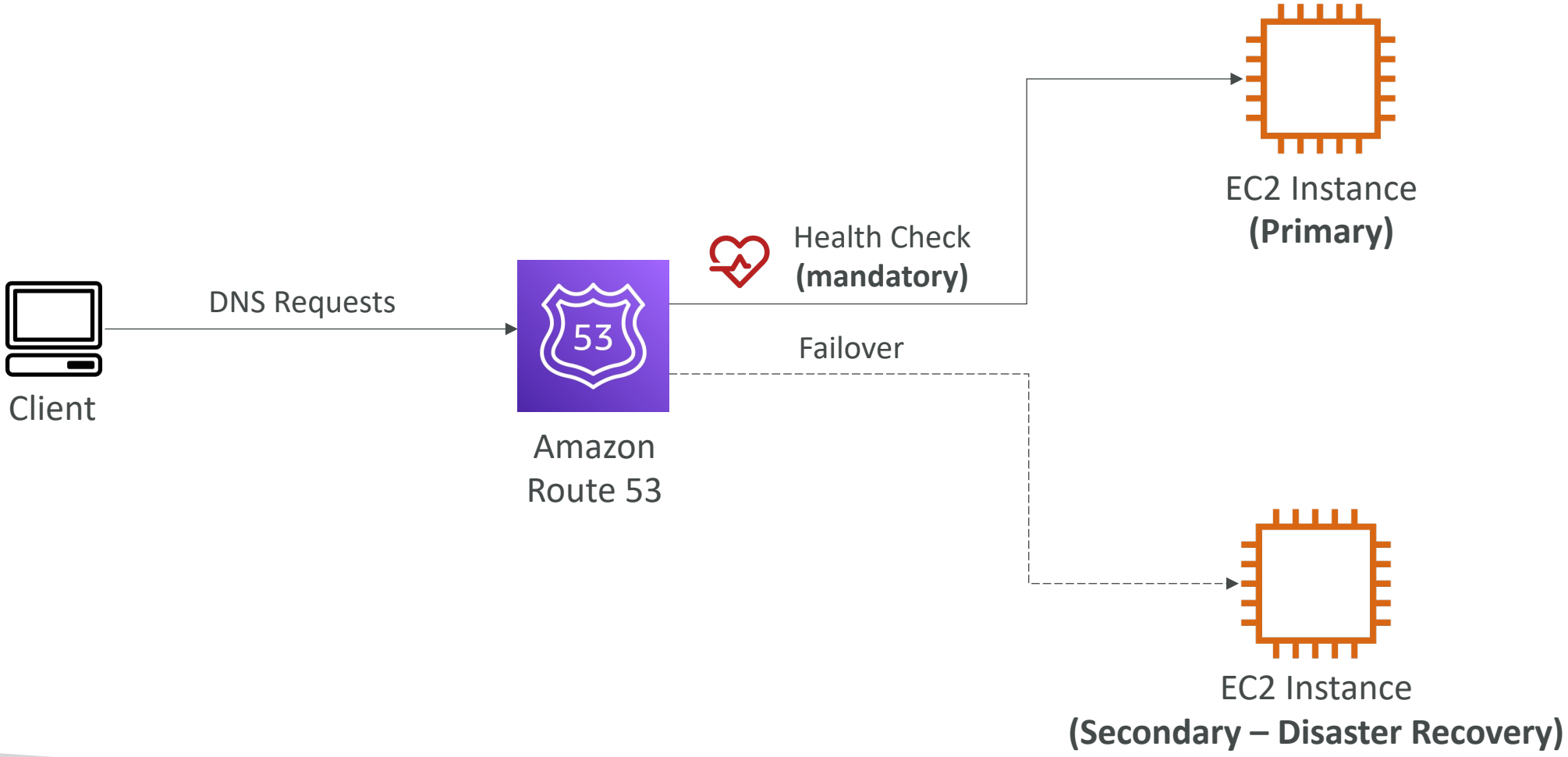


Health Checks – Private Hosted Zones

- Route 53 health checkers are outside the VPC
- They can't access **private** endpoints (private VPC or on-premises resource)
- You can create a **CloudWatch Metric** and associate a **CloudWatch Alarm**, then create a Health Check that checks the alarm itself

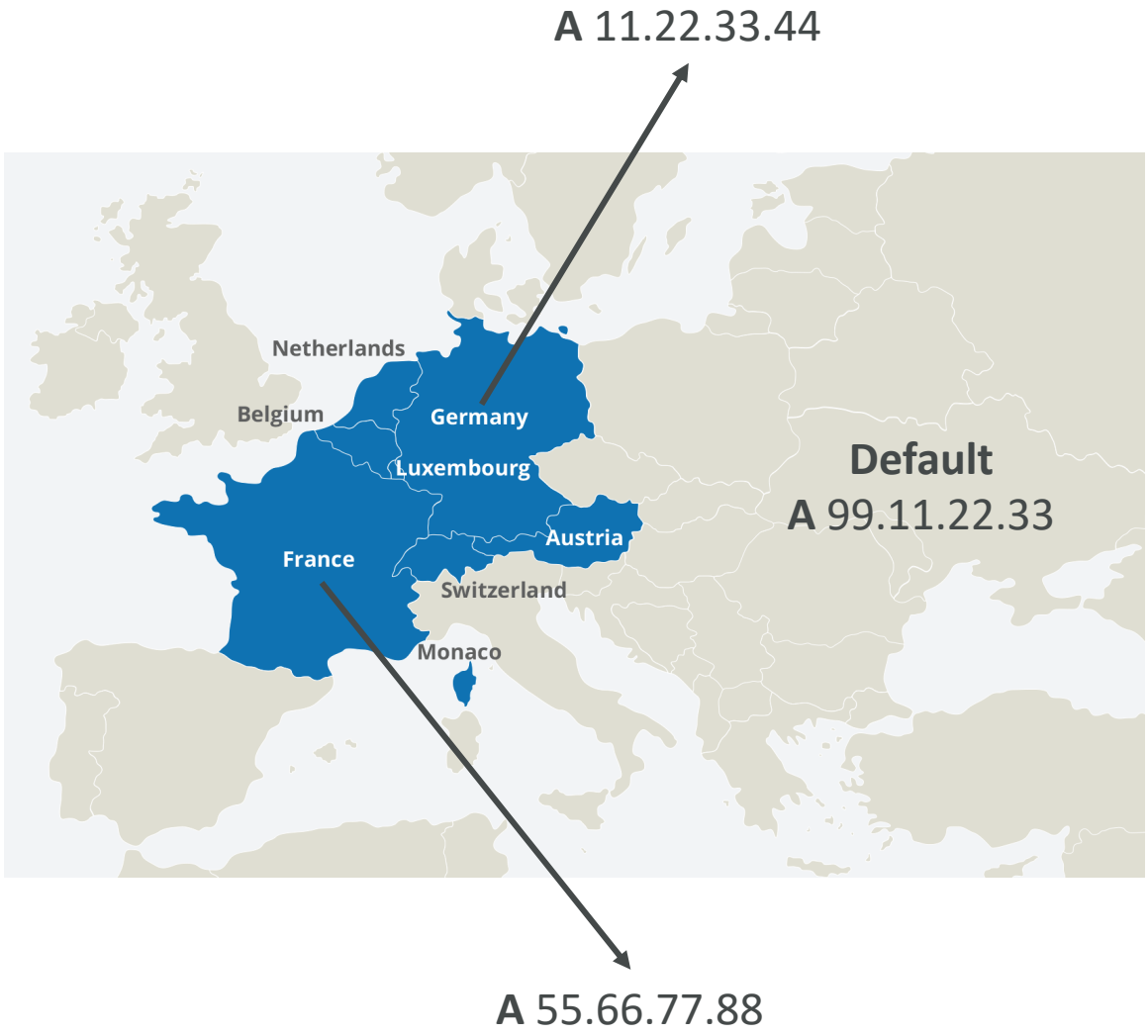


Routing Policies – Failover (Active-Passive)



Routing Policies – Geolocation

- Different from Latency-based!
- **This routing is based on user location**
- Specify location by Continent, Country or by US State (if there's overlapping, most precise location selected)
- Should create a **"Default"** record (in case there's no match on location)
- Use cases: website localization, restrict content distribution, load balancing, …
- Can be associated with Health Checks

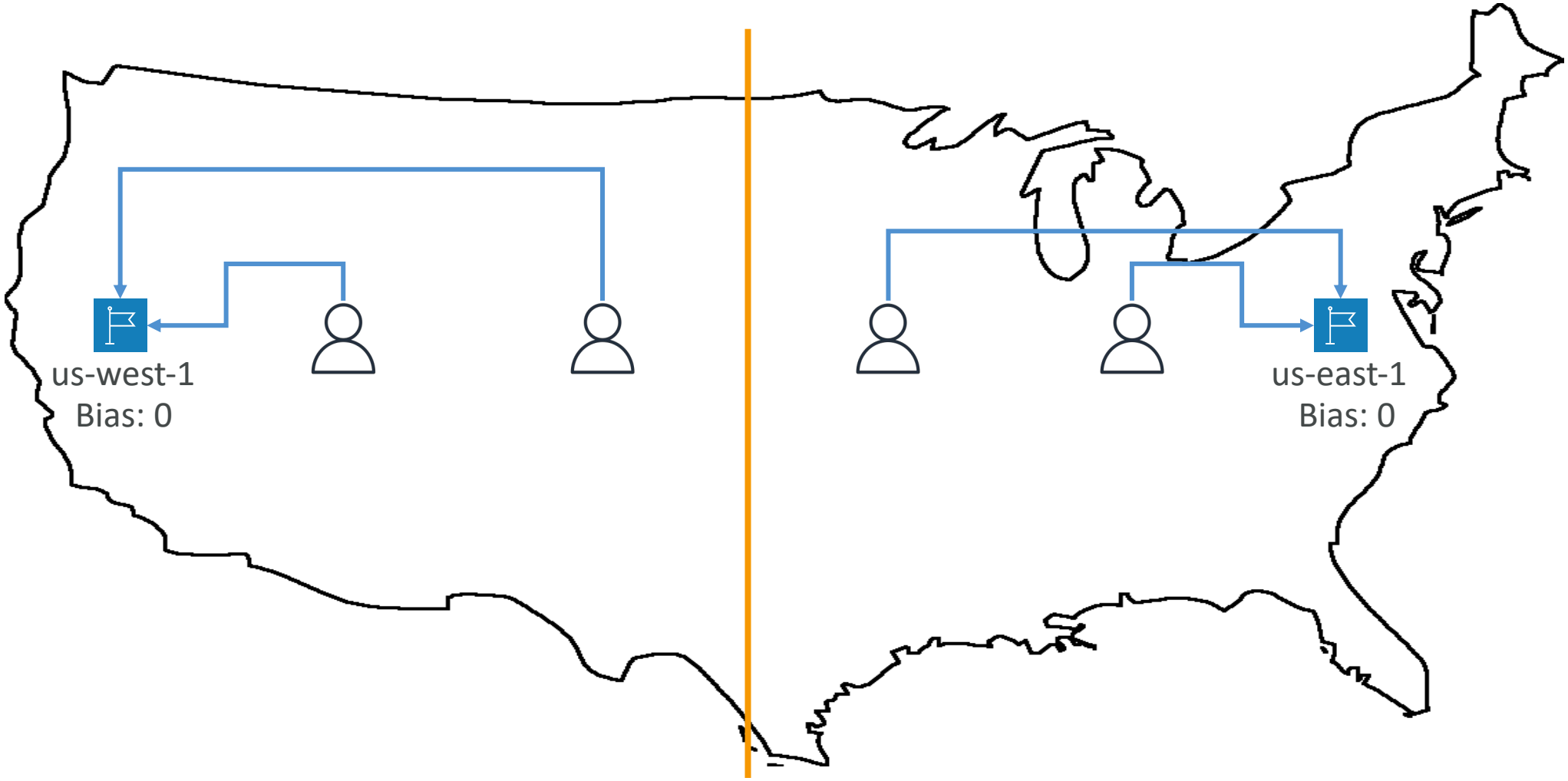


Routing Policies – Geoproximity

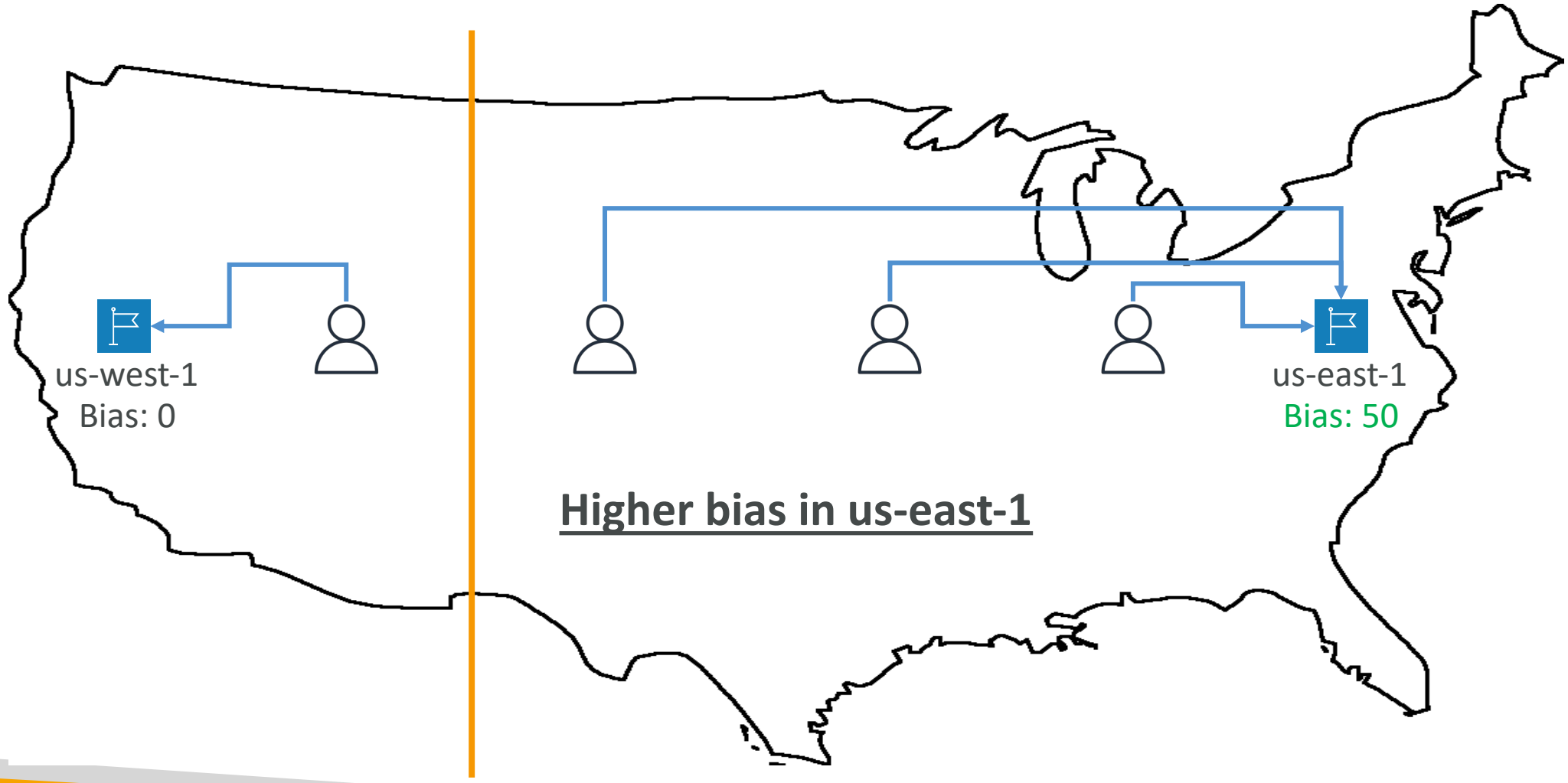
- Route traffic to your resources based on the geographic location of users and resources
- Ability **to shift more traffic to resources based** on the defined **bias**
- To change the size of the geographic region, specify **bias** values:
  - To expand (1 to 99) – more traffic to the resource
  - To shrink (-1 to -99) – less traffic to the resource

- Resources can be:
  - AWS resources (specify AWS region)
  - Non-AWS resources (specify Latitude and Longitude)
- You must use Route 53 **Traffic Flow** to use this feature

Routing Policies – Geoproximity

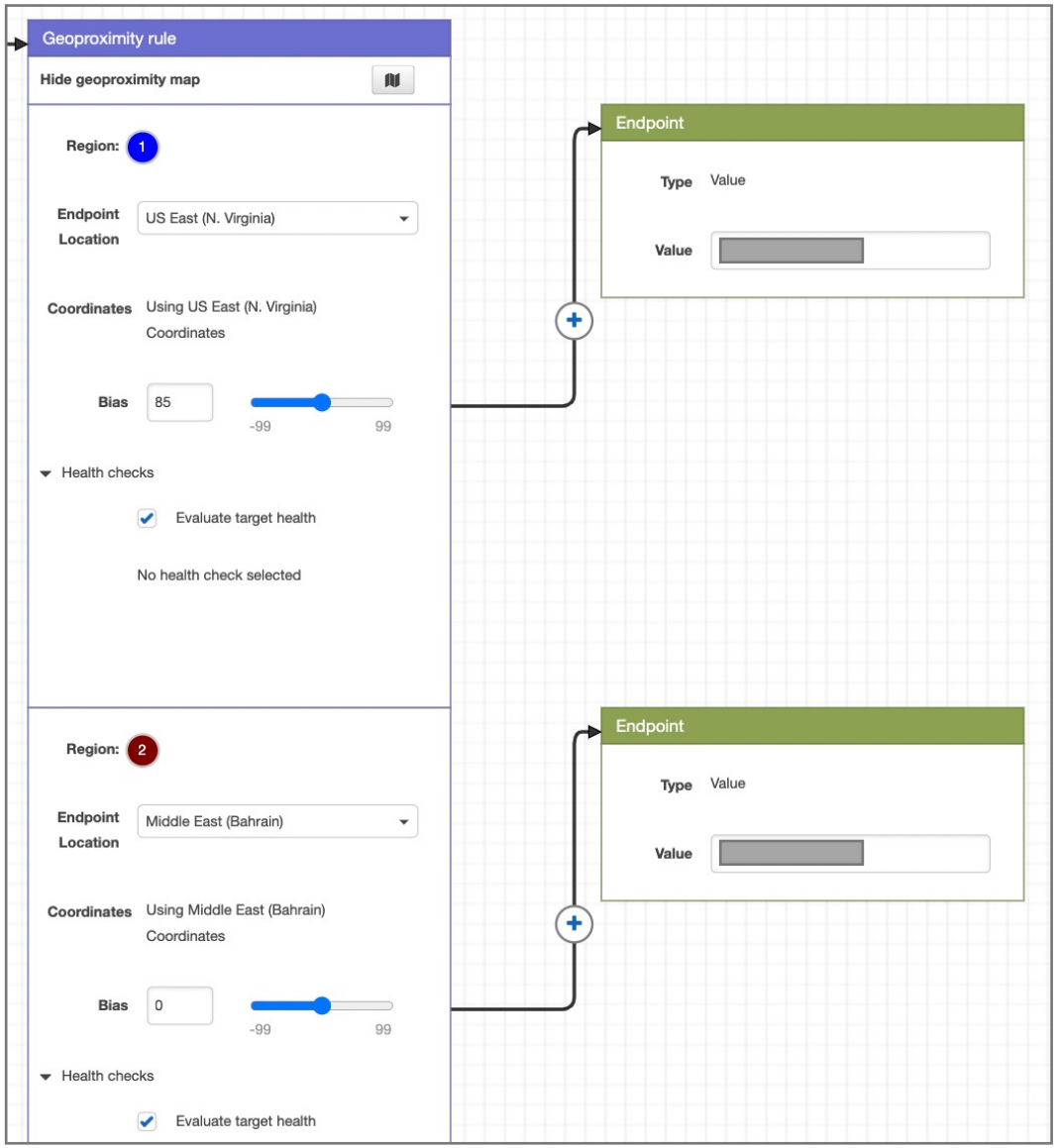


Routing Policies – Geoproximity



Route 53 – Traffic flow

- Simplify the process of creating and maintaining records in large and complex configurations
- Visual editor to manage complex routing decision trees
- Configurations can be saved as **Traffic Flow Policy**
  - Can be applied to different Route 53 Hosted Zones (different domain names)
  - Supports versioning



Routing Policies – IP-based Routing

- Routing is based on clients' IP addresses
- You provide a list of CIDRs for your clients and the corresponding endpoints/locations (user-IP-to-endpoint mappings)
- Use cases: Optimize performance, reduce network costs…
- Example: route end users from a particular ISP to a specific endpoint

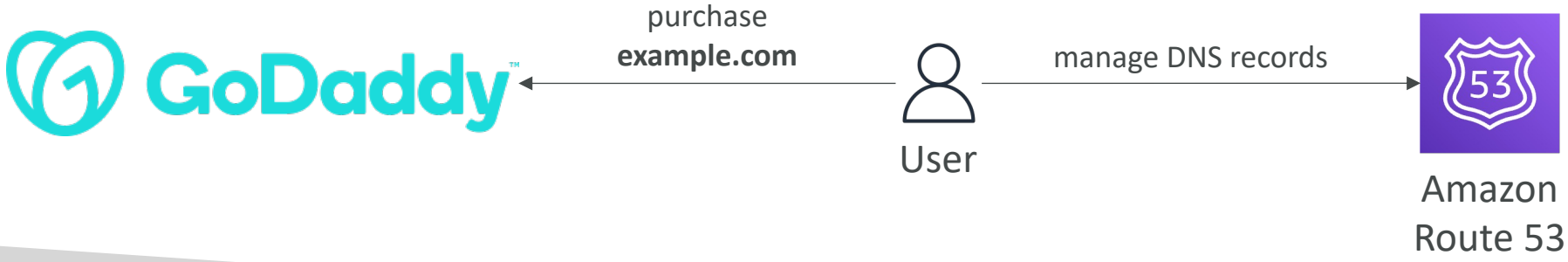
Routing Policies – Multi-Value

- Use when routing traffic to multiple resources
- Route 53 return multiple values/resources
- Can be associated with Health Checks (return only values for healthy resources)
- Up to 8 healthy records are returned for each Multi-Value query
- Multi-Value is not a substitute for having an ELB

Name	Type	Value	TTL	Set ID	Health Check
www.example.com	A Record	192.0.2.2	60	Web1	A
www.example.com	A Record	198.51.100.2	60	Web2	B
www.example.com	A Record	203.0.113.2	60	Web3	C

Domain Registar vs. DNS Service

- You buy or register your domain name with a Domain Registrar typically by paying annual charges (e.g., GoDaddy, Amazon Registrar Inc., …)
- The Domain Registrar usually provides you with a DNS service to manage your DNS records
- But you can use another DNS service to manage your DNS records
- Example: purchase the domain from GoDaddy and use Route 53 to manage your DNS records



GoDaddy as Registrar & Route 53 as DNS Service

GoDaddy

Records

We can't display your DNS information because your nameservers aren't managed by us.

Nameservers

Using custom nameservers **Change**

Nameserver
ns-1083.awsdns-07.org
ns-932.awsdns-52.net
ns-1911.awsdns-46.co.uk
ns-481.awsdns-60.com

Amazon Route 53

Public Hosted Zone
stephanetheteacher.com

Hosted zone details — Edit hosted zone

Hosted zone ID Z30IUJCCWPKZUV	Type Public hosted zone	Name servers ns-252.awsdns-31.com ns-1468.awsdns-55.org ns-633.awsdns-15.net ns-1800.awsdns-33.co.uk
Description HostedZone created by Route53 Registrar	Record count 22	
Query log		

3rd Party Registrar with Amazon Route 53

- If you buy your domain on a 3rd party registrar, you can still use Route 53 as the DNS Service provider

1. Create a Hosted Zone in Route 53
2. Update NS Records on 3rd party website to use Route 53 **Name Servers**

- Domain Registrar != DNS Service
- But every Domain Registrar usually comes with some DNS features

Amazon VPC – Basics

VPC – Crash Course

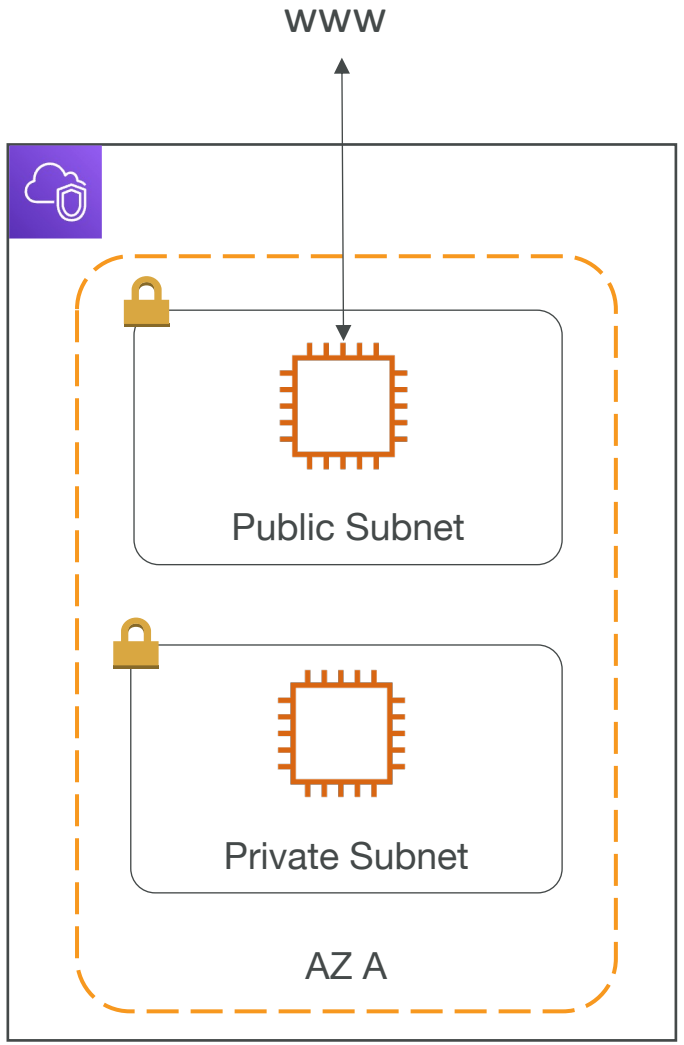
- VPC is something you should know in depth for the AWS Certified Solutions Architect Associate & AWS Certified SysOps Administrator

- At the AWS Certified Developer Level, you should know about:
  - VPC, Subnets, Internet Gateways & NAT Gateways
  - Security Groups, Network ACL (NACL), VPC Flow Logs
  - VPC Peering, VPC Endpoints
  - Site to Site VPN & Direct Connect

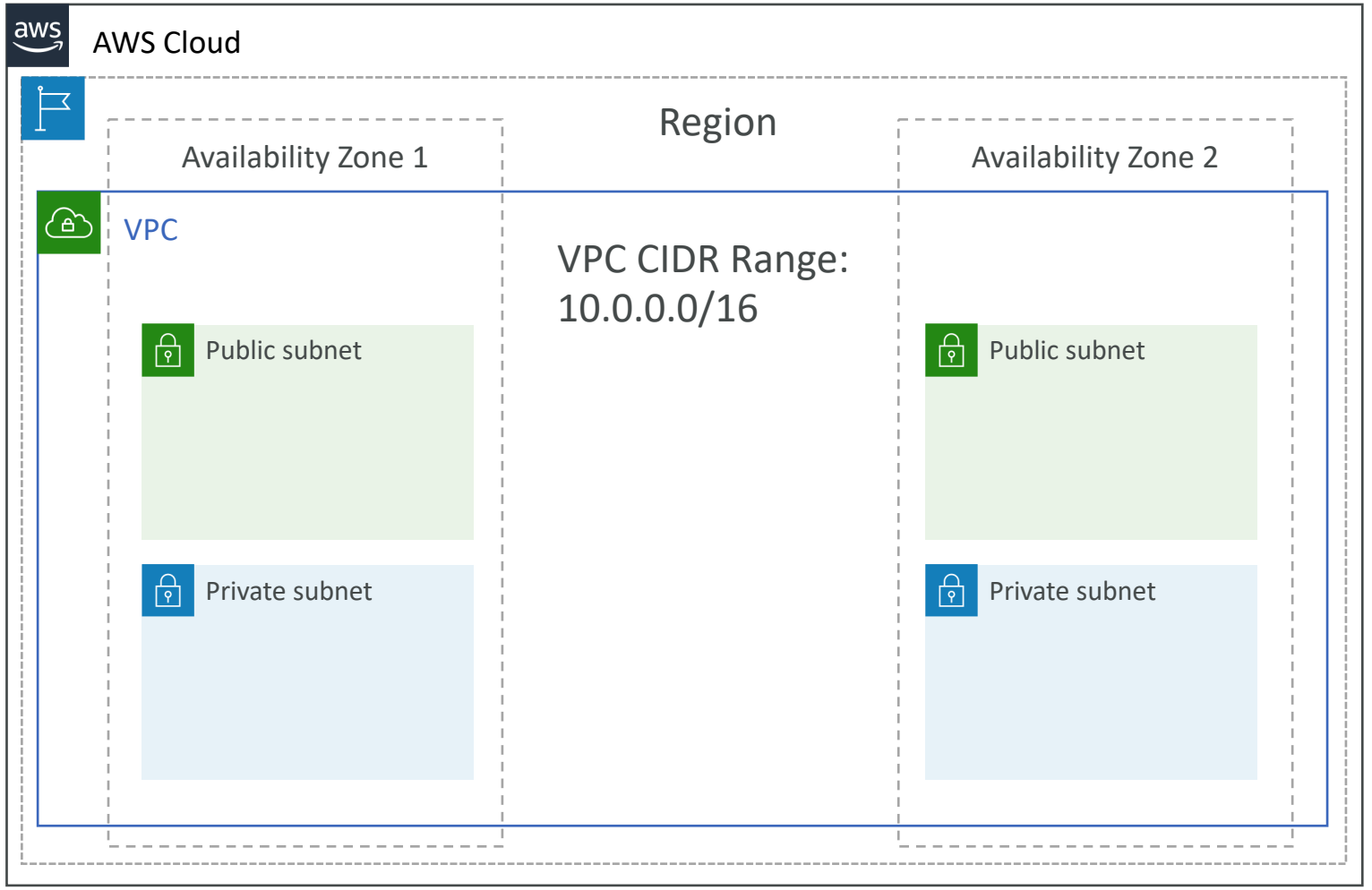
- I will just give you an overview, less than 1 or 2 questions at your exam.
- Later in the course, I will be highlighting when VPC concepts are helpful

VPC & Subnets Primer

- **VPC**: private network to deploy your resources (regional resource)
- **Subnets** allow you to partition your network inside your VPC (Availability Zone resource)
- A **public subnet** is a subnet that is accessible from the internet
- A **private subnet** is a subnet that is not accessible from the internet
- To define access to the internet and between subnets, we use **Route Tables**.



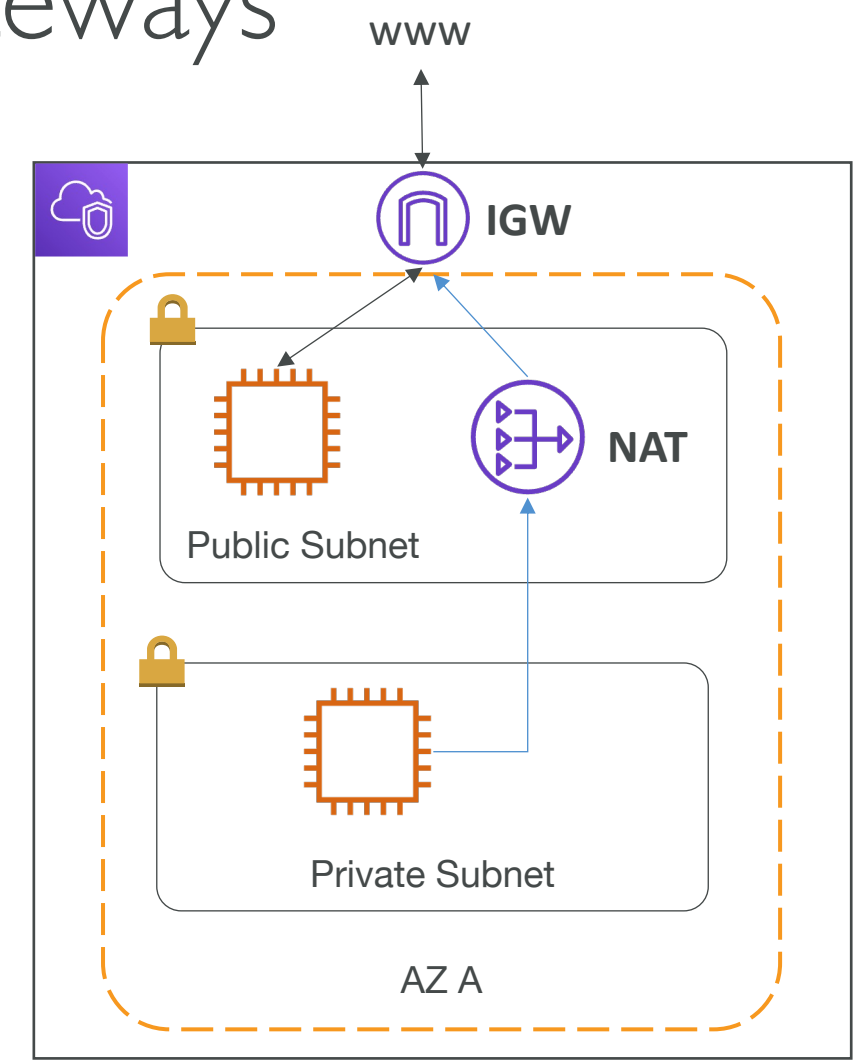
VPC Diagram



Internet Gateway & NAT Gateways

- **Internet Gateways** helps our VPC instances connect with the internet
- Public Subnets have a route to the internet gateway.

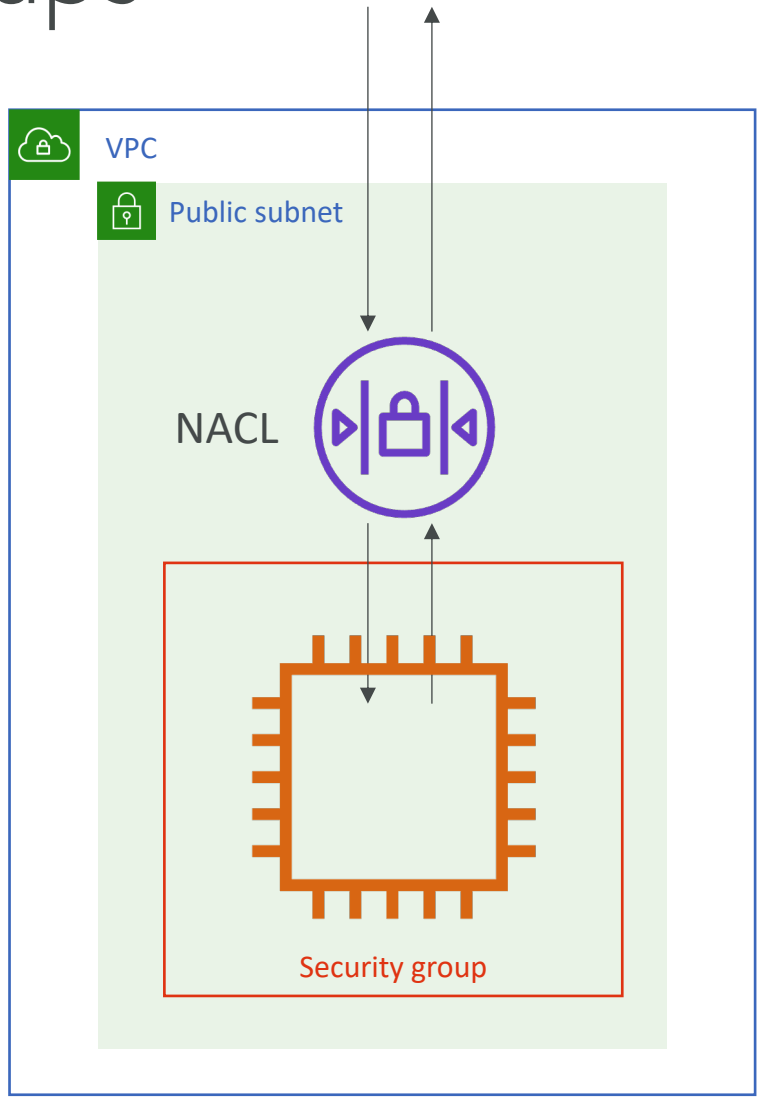
- **NAT Gateways** (AWS-managed) & **NAT Instances** (self-managed) allow your instances in your **Private Subnets** to access the internet while remaining private



Network ACL & Security Groups

- NACL (Network ACL)
  - A firewall which controls traffic from and to subnet
  - Can have ALLOW and DENY rules
  - Are attached at the **Subnet** level
  - Rules only include IP addresses

- Security Groups
  - A firewall that controls traffic to and from **an ENI / an EC2 Instance**
  - Can have only ALLOW rules
  - Rules include IP addresses and other security groups



Network ACLs vs Security Groups

Security Group	Network ACL
Operates at the instance level	Operates at the subnet level
Supports allow rules only	Supports allow rules and deny rules
Is stateful: Return traffic is automatically allowed, regardless of any rules	Is stateless: Return traffic must be explicitly allowed by rules
We evaluate all rules before deciding whether to allow traffic	We process rules in number order when deciding whether to allow traffic
Applies to an instance only if someone specifies the security group when launching the instance, or associates the security group with the instance later on	Automatically applies to all instances in the subnets it's associated with (therefore, you don't have to rely on users to specify the security group)

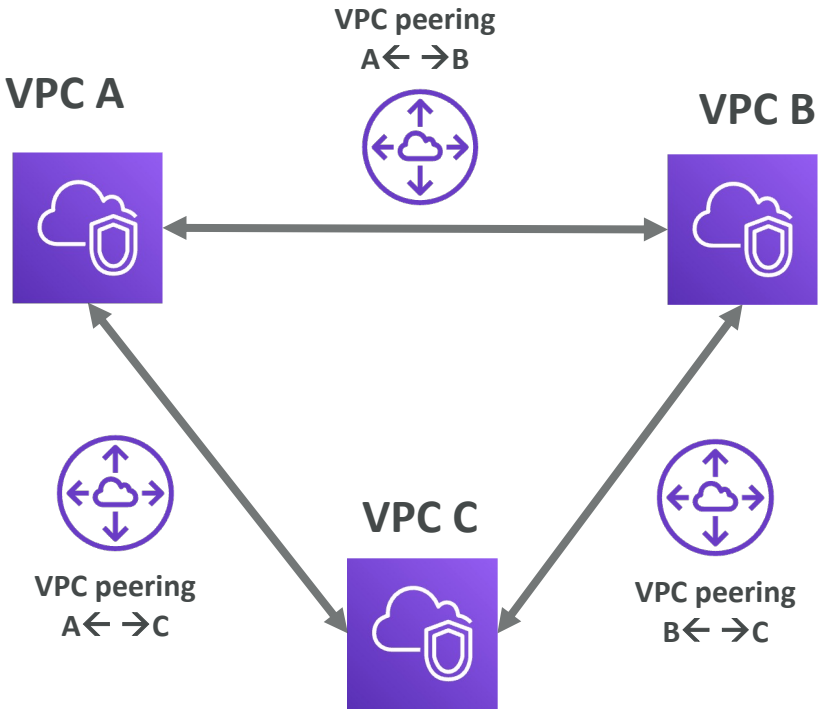
https://docs.aws.amazon.com/vpc/latest/userguide/VPC_Security.html#VPC_Security_Comparison

VPC Flow Logs

- Capture information about IP traffic going into your interfaces:
  - **VPC** Flow Logs
  - **Subnet** Flow Logs
  - **Elastic Network Interface** Flow Logs
- Helps to monitor & troubleshoot connectivity issues. Example:
  - Subnets to internet
  - Subnets to subnets
  - Internet to subnets
- Captures network information from AWS managed interfaces too: Elastic Load Balancers, ElastiCache, RDS, Aurora, etc…
- VPC Flow logs data can go to S3, CloudWatch Logs, and Kinesis Data Firehose

VPC Peering

- Connect two VPC, privately using AWS' network
- Make them behave as if they were in the same network
- Must not have overlapping CIDR (IP address range)
- VPC Peering connection is **not transitive** (must be established for each VPC that need to communicate with one another)

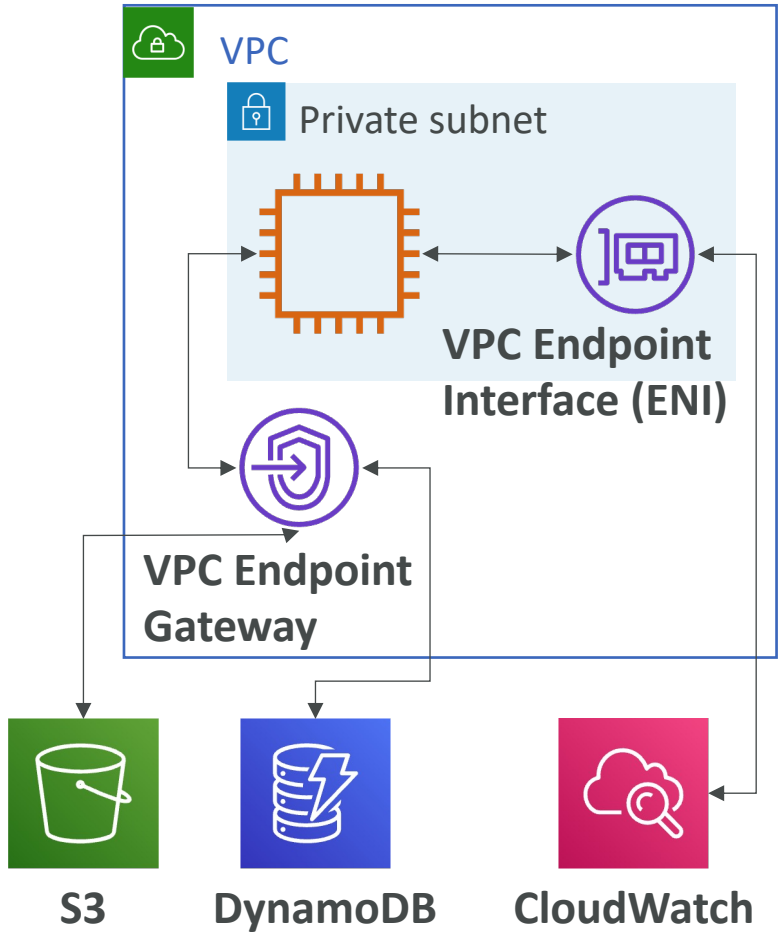


VPC Endpoints

- Endpoints allow you to connect to AWS Services **using a private network** instead of the public www network
- This gives you enhanced security and lower latency to access AWS services

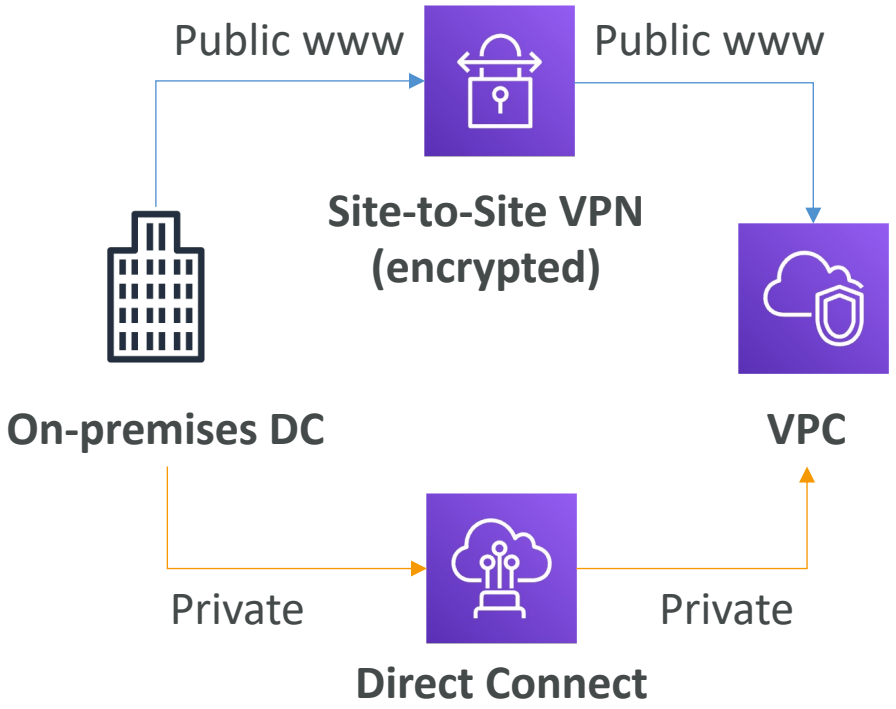
- VPC Endpoint Gateway: S3 & DynamoDB
- VPC Endpoint Interface: the rest

- Only used within your VPC



Site to Site VPN & Direct Connect

- Site to Site VPN
  - Connect an on-premises VPN to AWS
  - The connection is automatically encrypted
  - Goes over the public internet
- Direct Connect (DX)
  - Establish a physical connection between on-premises and AWS
  - The connection is private, secure and fast
  - Goes over a private network
  - Takes at least a month to establish



VPC Closing Comments

- **VPC:** Virtual Private Cloud
- **Subnets:** Tied to an AZ, network partition of the VPC
- **Internet Gateway:** at the VPC level, provide Internet Access
- **NAT Gateway / Instances:** give internet access to private subnets
- **NACL:** Stateless, subnet rules for inbound and outbound
- **Security Groups:** Stateful, operate at the EC2 instance level or ENI
- **VPC Peering:** Connect two VPC with non overlapping IP ranges, non transitive
- **VPC Endpoints:** Provide private access to AWS Services within VPC
- **VPC Flow Logs:** network traffic logs
- **Site to Site VPN:** VPN over public internet between on-premises DC and AWS
- **Direct Connect:** direct private connection to a AWS

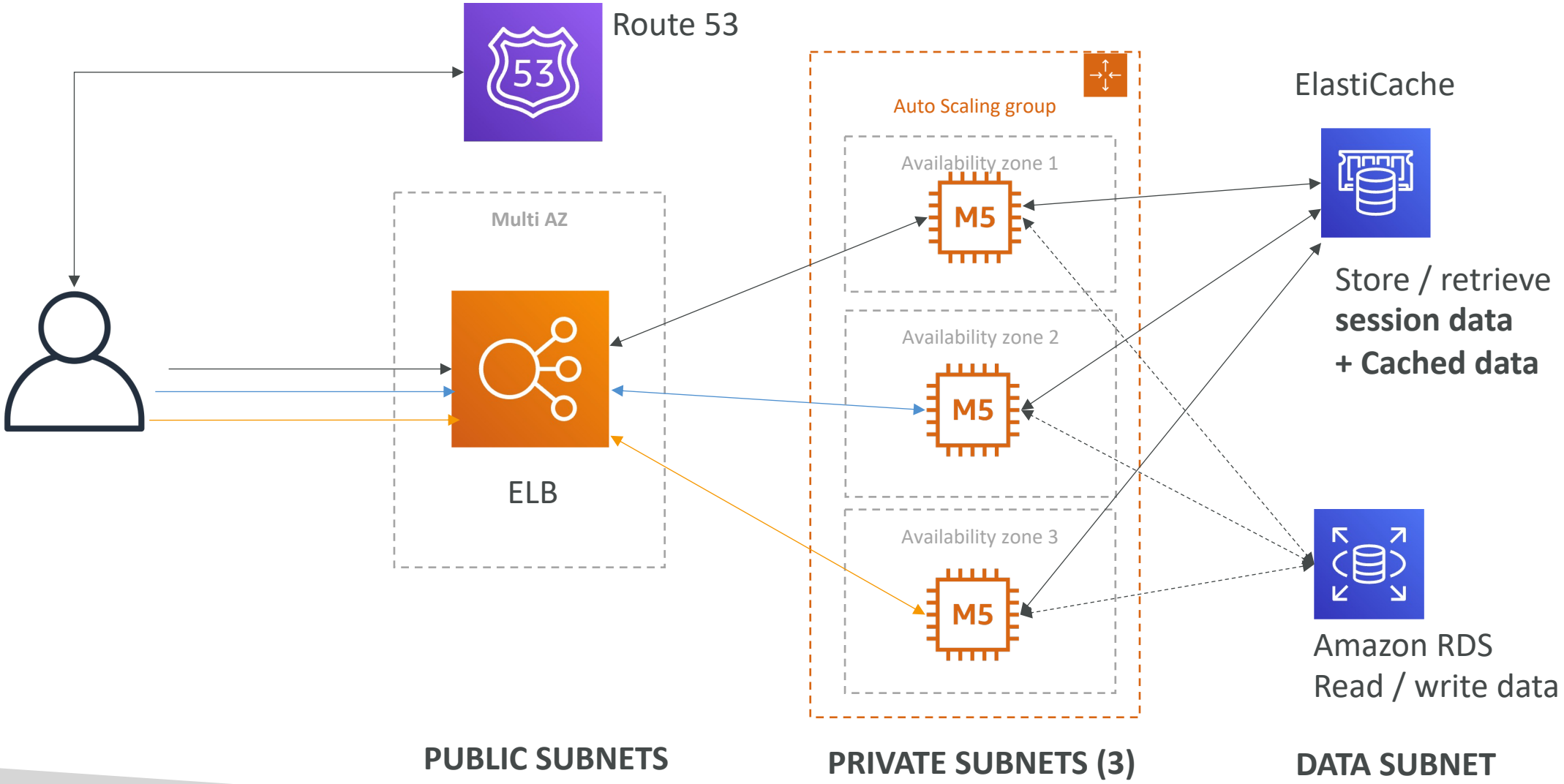
VPC note – AWS Certified Developer

- Don't stress if you didn't understand everything in that section
- I will be highlighting in the course the specific VPC features we need

- Feel free to revisit that section after you're done in the course !

- Moving on ☺

Typical 3 tier solution architecture

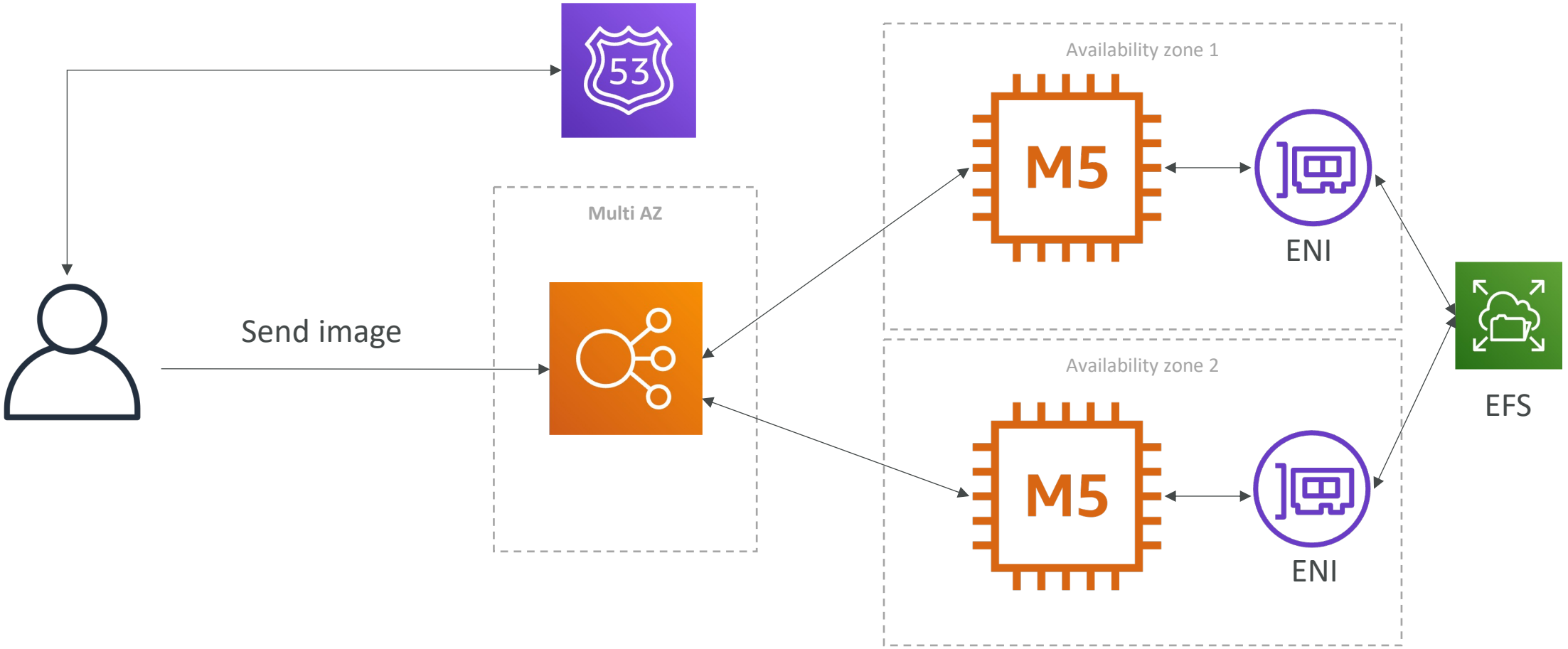


LAMP Stack on EC2

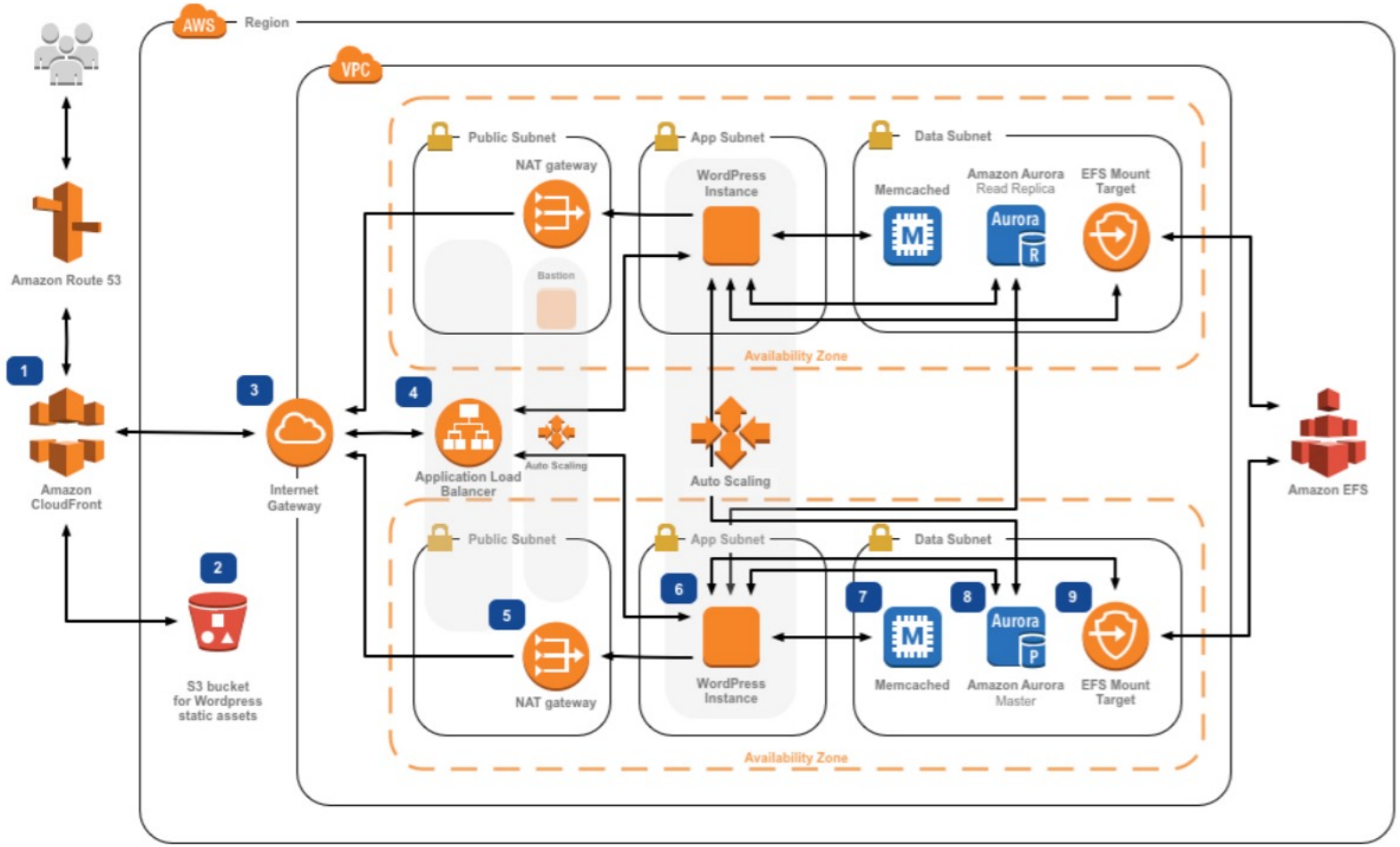
- Linux: OS for EC2 instances
- Apache: Web Server that run on Linux (EC2)
- MySQL: database on RDS
- PHP: Application logic (running on EC2)

- Can add Redis / Memcached (ElastiCache) to include a caching tech
- To store local application data & software: EBS drive (root)

Wordpress on AWS



WordPress on AWS (more complicated)



https://aws.amazon.com/blogs/architecture/wordpress-best-practices-on-aws/

Amazon S3

Section introduction

- Amazon S3 is one of the main building blocks of AWS
- It's advertised as "infinitely scaling" storage

- Many websites use Amazon S3 as a backbone
- Many AWS services use Amazon S3 as an integration as well

- We'll have a step-by-step approach to S3

Amazon S3 Use cases

- Backup and storage
- Disaster Recovery
- Archive
- Hybrid Cloud storage
- Application hosting
- Media hosting
- Data lakes & big data analytics
- Software delivery
- Static website

Nasdaq stores 7 years of data into S3 Glacier

Sysco runs analytics on its data and gain business insights

Amazon S3 - Buckets

- Amazon S3 allows people to store objects (files) in "buckets" (directories)
- Buckets must have a **globally unique name (across all regions all accounts)**
- Buckets are defined at the region level
- S3 looks like a global service but buckets are created in a region
- Naming convention
  - No uppercase, No underscore
  - 3-63 characters long
  - Not an IP
  - Must start with lowercase letter or number
  - Must NOT start with the prefix **xn--**
  - Must NOT end with the suffix **-s3alias**

S3 Bucket

Amazon S3 - Objects

- Objects (files) have a Key
- The key is the FULL path:
  - s3://my-bucket/my_file.txt
  - s3://my-bucket/my_folder1/another_folder/my_file.txt
- The key is composed of prefix + object name
  - s3://my-bucket/my_folder1/another_folder/my_file.txt
- There's no concept of "directories" within buckets (although the UI will trick you to think otherwise)
- Just keys with very long names that contain slashes ("/")

Object

S3 Bucket with Objects

Amazon S3 – Objects (cont.)

- Object values are the content of the body:
  - Max. Object Size is 5TB (5000GB)
  - If uploading more than 5GB, must use "multi-part upload"

- Metadata (list of text key / value pairs – system or user metadata)
- Tags (Unicode key / value pair – up to 10) – useful for security / lifecycle
- Version ID (if versioning is enabled)

Amazon S3 – Security

- User-Based
  - **IAM Policies** – which API calls should be allowed for a specific user from IAM
- Resource-Based
  - **Bucket Policies** – bucket wide rules from the S3 console - allows cross account
  - **Object Access Control List (ACL)** – finer grain (can be disabled)
  - **Bucket Access Control List (ACL)** – less common (can be disabled)
- **Note:** an IAM principal can access an S3 object if
  - The user IAM permissions ALLOW it OR the resource policy ALLOWS it
  - AND there's no explicit DENY
- **Encryption:** encrypt objects in Amazon S3 using encryption keys

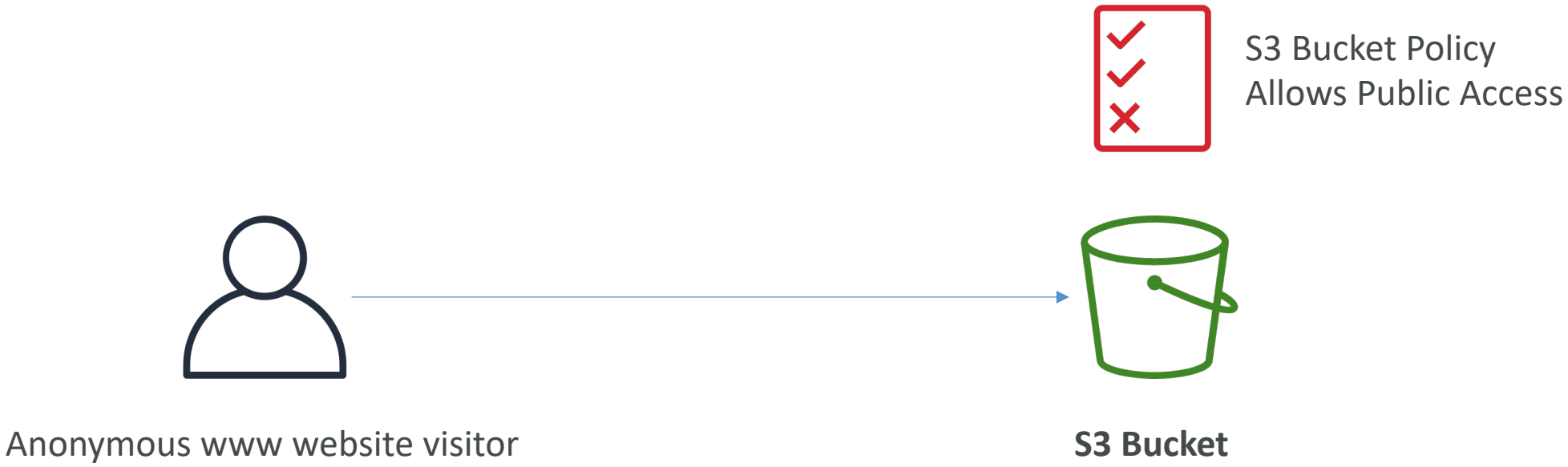
S3 Bucket Policies

- JSON based policies
  - Resources: buckets and objects
  - Effect: Allow / Deny
  - Actions: Set of API to Allow or Deny
  - Principal: The account or user to apply the policy to

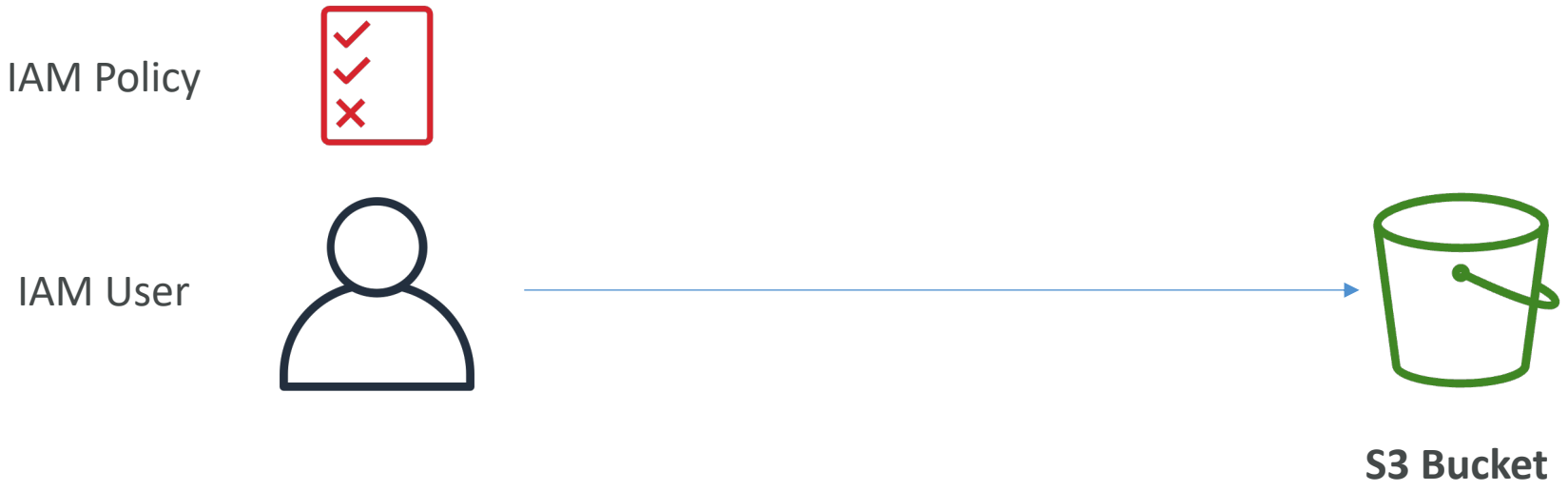
- Use S3 bucket for policy to:
  - Grant public access to the bucket
  - Force objects to be encrypted at upload
  - Grant access to another account (Cross Account)

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PublicRead",
            "Effect": "Allow",
            "Principal": "*",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::examplebucket/*"
            ]
        }
    ]
}
```

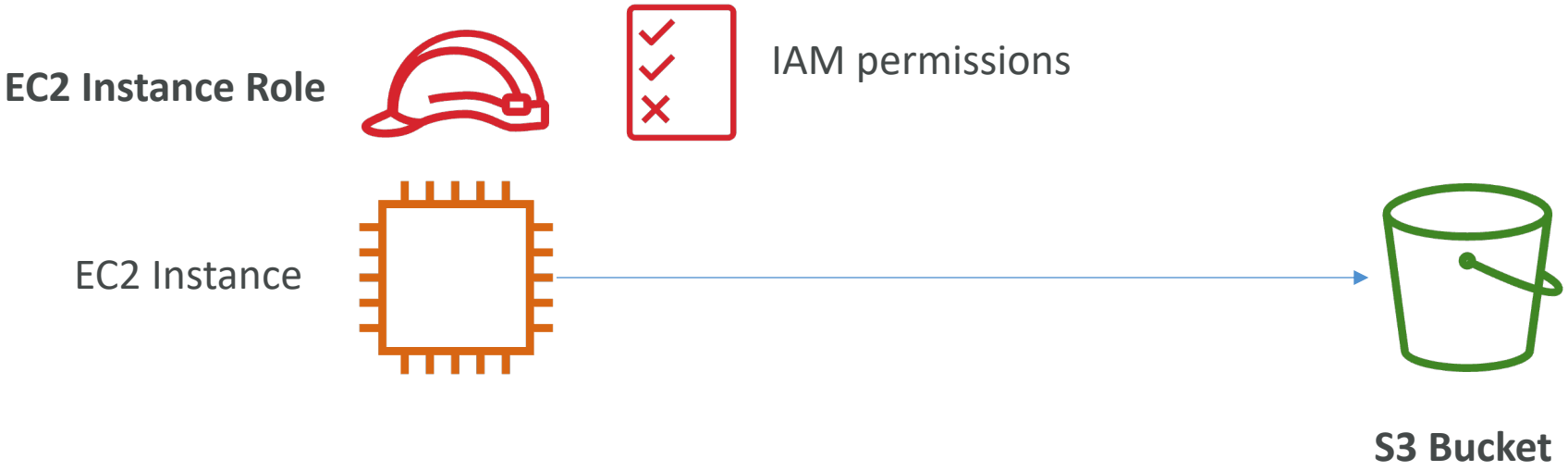
Example: Public Access - Use Bucket Policy



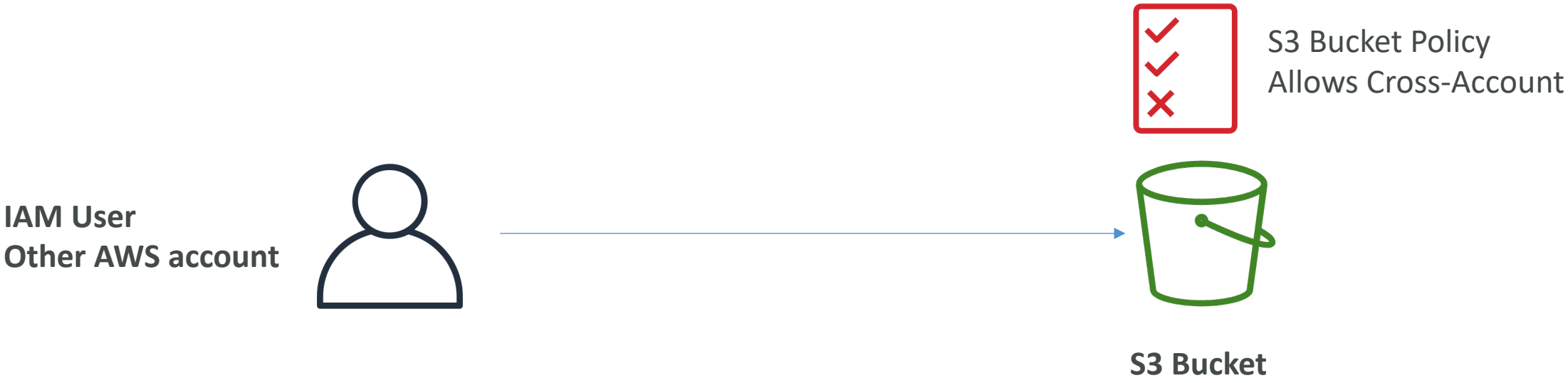
Example: User Access to S3 – IAM permissions



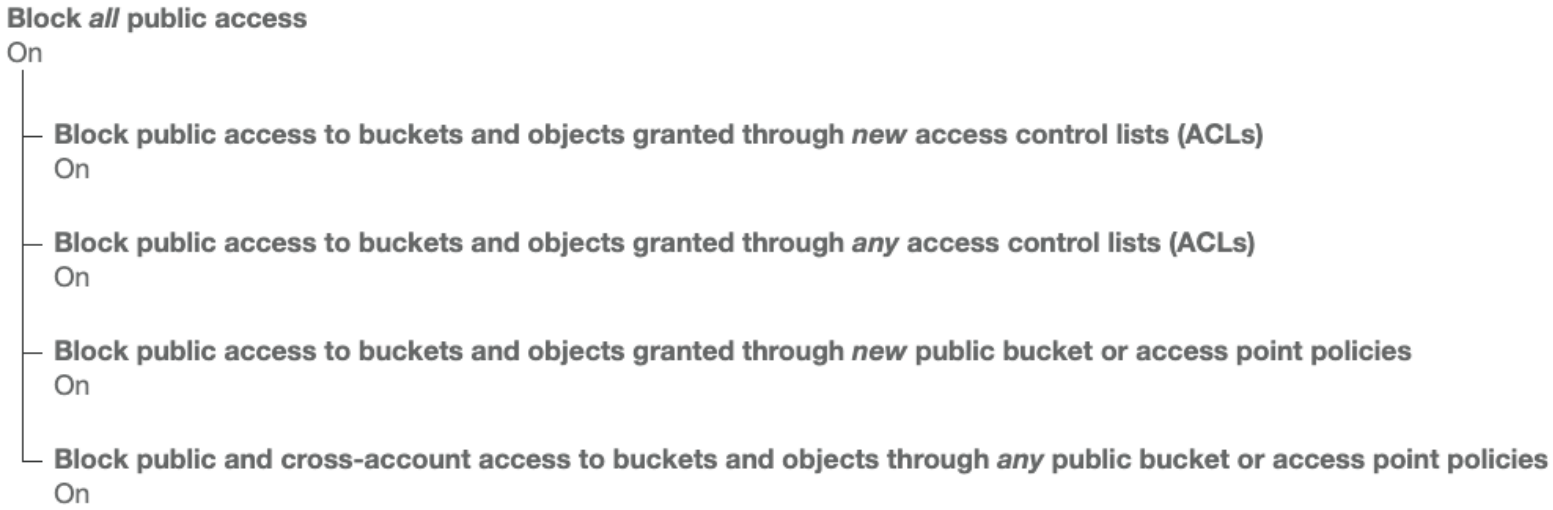
Example: EC2 instance access - Use IAM Roles



Advanced: Cross-Account Access – Use Bucket Policy



Bucket settings for Block Public Access



- These settings were created to prevent company data leaks
- If you know your bucket should never be public, leave these on
- Can be set at the account level

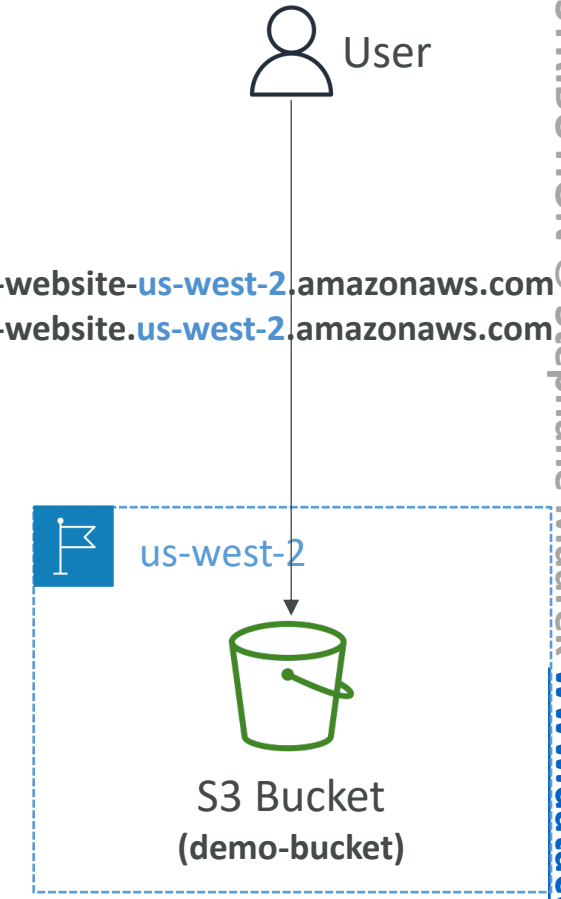
Amazon S3 – Static Website Hosting

- S3 can host static websites and have them accessible on the Internet

- The website URL will be (depending on the region)
  - http://*bucket-name*.s3-website-*aws-region*.amazonaws.com

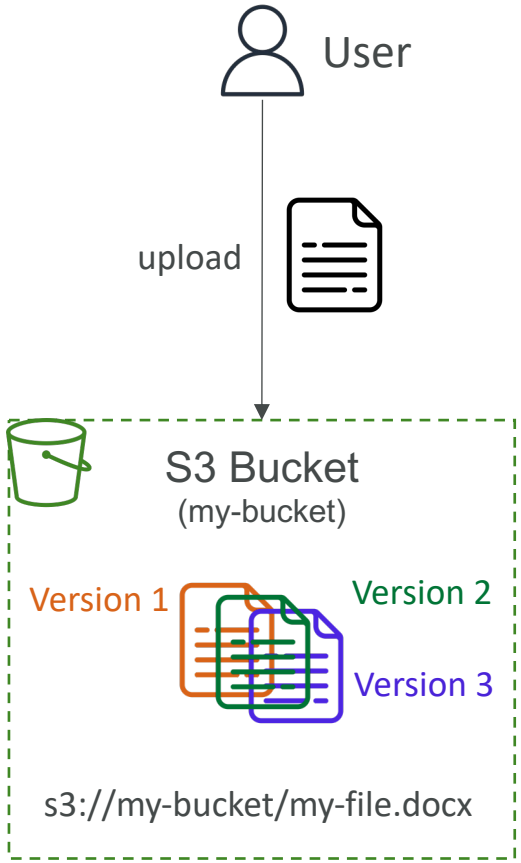
  OR
  - http://*bucket-name*.s3-website.*aws-region*.amazonaws.com

- If you get a **403 Forbidden** error, make sure the bucket policy allows public reads!



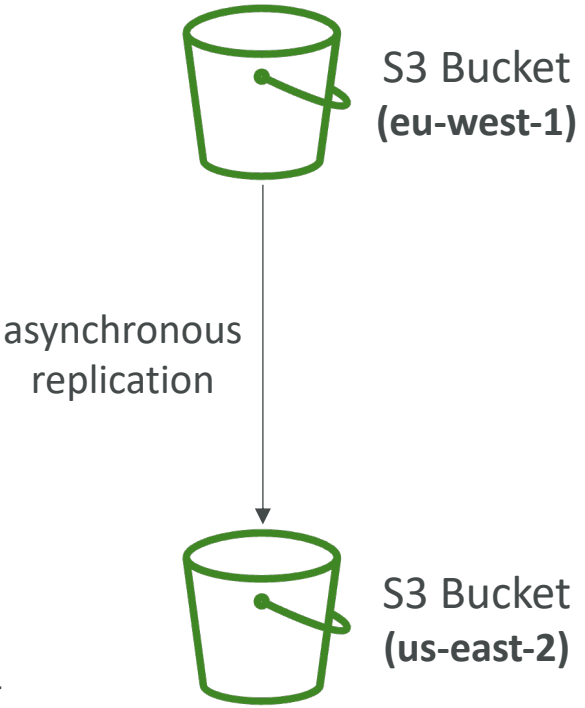
Amazon S3 - Versioning

- You can version your files in Amazon S3
- It is enabled at the **bucket level**
- Same key overwrite will change the "version": 1, 2, 3….
- It is best practice to version your buckets
  - Protect against unintended deletes (ability to restore a version)
  - Easy roll back to previous version
- Notes:
  - Any file that is not versioned prior to enabling versioning will have version "null"
  - Suspending versioning does not delete the previous versions



Amazon S3 – Replication (CRR & SRR)

- **Must enable Versioning** in source and destination buckets
- **Cross-Region Replication (CRR)**
- **Same-Region Replication (SRR)**
- Buckets can be in different AWS accounts
- Copying is asynchronous
- Must give proper IAM permissions to S3



- Use cases:
  - CRR – compliance, lower latency access, replication across accounts
  - SRR – log aggregation, live replication between production and test accounts

Amazon S3 – Replication (Notes)

- After you enable Replication, only new objects are replicated
- Optionally, you can replicate existing objects using **S3 Batch Replication**
  - Replicates existing objects and objects that failed replication

- For DELETE operations
  - **Can replicate delete markers** from source to target (optional setting)
  - Deletions with a version ID are not replicated (to avoid malicious deletes)

- There is no "chaining" of replication
  - If bucket 1 has replication into bucket 2, which has replication into bucket 3
  - Then objects created in bucket 1 are not replicated to bucket 3

S3 Storage Classes

- Amazon S3 Standard - General Purpose
- Amazon S3 Standard-Infrequent Access (IA)
- Amazon S3 One Zone-Infrequent Access
- Amazon S3 Glacier Instant Retrieval
- Amazon S3 Glacier Flexible Retrieval
- Amazon S3 Glacier Deep Archive
- Amazon S3 Intelligent Tiering

- Can move between classes manually or using S3 Lifecycle configurations

S3 Durability and Availability

- Durability:
  - High durability (99.999999999%, 11 9's) of objects across multiple AZ
  - If you store 10,000,000 objects with Amazon S3, you can on average expect to incur a loss of a single object once every 10,000 years
  - Same for all storage classes

- Availability:
  - Measures how readily available a service is
  - Varies depending on storage class
  - Example: S3 standard has 99.99% availability = not available 53 minutes a year

S3 Standard – General Purpose

- 99.99% Availability
- Used for frequently accessed data
- Low latency and high throughput
- Sustain 2 concurrent facility failures

- Use Cases: Big Data analytics, mobile & gaming applications, content distribution…

S3 Storage Classes – Infrequent Access

- For data that is less frequently accessed, but requires rapid access when needed
- Lower cost than S3 Standard

- Amazon S3 Standard-Infrequent Access (S3 Standard-IA)
  - 99.9% Availability
  - Use cases: Disaster Recovery, backups

- Amazon S3 One Zone-Infrequent Access (S3 One Zone-IA)
  - High durability (99.999999999%) in a single AZ; data lost when AZ is destroyed
  - 99.5% Availability
  - Use Cases: Storing secondary backup copies of on-premises data, or data you can recreate

Amazon S3 Glacier Storage Classes

- Low-cost object storage meant for archiving / backup
- Pricing: price for storage + object retrieval cost

- Amazon S3 Glacier Instant Retrieval
  - Millisecond retrieval, great for data accessed once a quarter
  - Minimum storage duration of 90 days
- Amazon S3 Glacier Flexible Retrieval (formerly Amazon S3 Glacier):
  - Expedited (1 to 5 minutes), Standard (3 to 5 hours), Bulk (5 to 12 hours) – free
  - Minimum storage duration of 90 days
- Amazon S3 Glacier Deep Archive – for long term storage:
  - Standard (12 hours), Bulk (48 hours)
  - Minimum storage duration of 180 days

S3 Intelligent-Tiering

- Small monthly monitoring and auto-tiering fee
- Moves objects automatically between Access Tiers based on usage
- There are no retrieval charges in S3 Intelligent-Tiering

- *Frequent Access tier (automatic):* default tier
- *Infrequent Access tier (automatic)*: objects not accessed for 30 days
- *Archive Instant Access tier (automatic):* objects not accessed for 90 days
- *Archive Access tier (optional):* configurable from 90 days to 700+ days
- *Deep Archive Access tier (optional):* config. from 180 days to 700+ days

S3 Storage Classes Comparison

	Standard	Intelligent-Tiering	Standard-IA	One Zone-IA	Glacier Instant Retrieval	Glacier Flexible Retrieval	Glacier Deep Archive
Durability	99.999999999% == (11 9's)						
Availability	99.99%	99.9%	99.9%	99.5%	99.9%	99.99%	99.99%
Availability SLA	99.9%	99%	99%	99%	99%	99.9%	99.9%
Availability Zones	>= 3	>= 3	>= 3	1	>= 3	>= 3	>= 3
Min. Storage Duration Charge	None	None	30 Days	30 Days	90 Days	90 Days	180 Days
Min. Billable Object Size	None	None	128 KB	128 KB	128 KB	40 KB	40 KB
Retrieval Fee	None	None	Per GB retrieved	Per GB retrieved	Per GB retrieved	Per GB retrieved	Per GB retrieved

https://aws.amazon.com/s3/storage-classes/

S3 Storage Classes – Price Comparison Example: us-east- l

	Standard	Intelligent-Tiering	Standard-IA	One Zone-IA	Glacier Instant Retrieval	Glacier Flexible Retrieval	Glacier Deep Archive
Storage Cost (per GB per month)	$0.023	$0.0025 - $0.023	$0.0125	$0.01	$0.004	$0.0036	$0.00099
Retrieval Cost (per 1000 request)	**GET:** $0.0004 **POST:** $0.005	**GET:** $0.0004 **POST:** $0.005	**GET:** $0.001 **POST:** $0.01	**GET:** $0.001 **POST:** $0.01	**GET:** $0.01 **POST:** $0.02	**GET:** $0.0004 **POST:** $0.03 **Expedited:** $10 **Standard:** $0.05 **Bulk:** free	**GET:** $0.0004 **POST:** $0.05 **Standard:** $0.10 **Bulk:** $0.025
Retrieval Time	Instantaneous					**Expedited** (1 – 5 mins) **Standard** (3 – 5 hours) Bulk (5 – 12 hours)	**Standard** (12 hours) **Bulk** (48 hours)
Monitoring Cost (pet 1000 objects)		$0.0025					

https://aws.amazon.com/s3/pricing/

AWS CLI, SDK, IAM Roles & Policies

EC2 Instance Metadata (IMDS)

- AWS EC2 Instance Metadata (IMDS) is powerful but one of the least known features to developers
- It allows AWS EC2 instances to "learn about themselves" **without using an IAM Role for that purpose.**
- The URL is http://169.254.169.254/latest/meta-data
- You can retrieve the IAM Role name from the metadata, but you CANNOT retrieve the IAM Policy.
- Metadata = Info about the EC2 instance
- Userdata = launch script of the EC2 instance

- Let's practice and see what we can do with it!

IMDSv2 vs. IMDSv1

- IMDSv1 is accessing http://169.254.169.254/latest/meta-data directly

- IMDSv2 is more secure and is done in two steps:
  1. Get Session Token (limited validity) – using headers & PUT

```
$ TOKEN=`curl -X PUT "http://169.254.169.254/latest/api/token"
-H "X-aws-ec2-metadata-token-ttl-seconds: 21600"`
```

  2. Use Session Token in IMDSv2 calls – using headers

```
$ curl http://169.254.169.254/latest/meta-data/profile
-H "X-aws-ec2-metadata-token: $TOKEN"
```

MFA with CLI

- To use MFA with the CLI, you must create a temporary session
- To do so, you must run the **STS GetSessionToken** API call

- **aws sts get-session-token** --serial-number arn-of-the-mfa-device --token-code code-from-token --duration-seconds 3600

```
{
    "Credentials": {
        "SecretAccessKey": "secret-access-key",
        "SessionToken": "temporary-session-token",
        "Expiration": "expiration-date-time",
        "AccessKeyId": "access-key-id"
    }
}
```

AWS SDK Overview

- What if you want to perform actions on AWS directly from your applications code ? (without using the CLI).
- You can use an SDK (software development kit) !
- Official SDKs are…
  - Java
  - .NET
  - Node.js
  - PHP
  - Python (named boto3 / botocore)
  - Go
  - Ruby
  - C++

AWS SDK Overview

- We have to use the AWS SDK when coding against AWS Services such as DynamoDB

- Fun fact… the AWS CLI uses the Python SDK (boto3)
- The exam expects you to know when you should use an SDK
- We'll practice the AWS SDK when we get to the Lambda functions

- Good to know: if you don't specify or configure a default region, then us-east-1 will be chosen by default

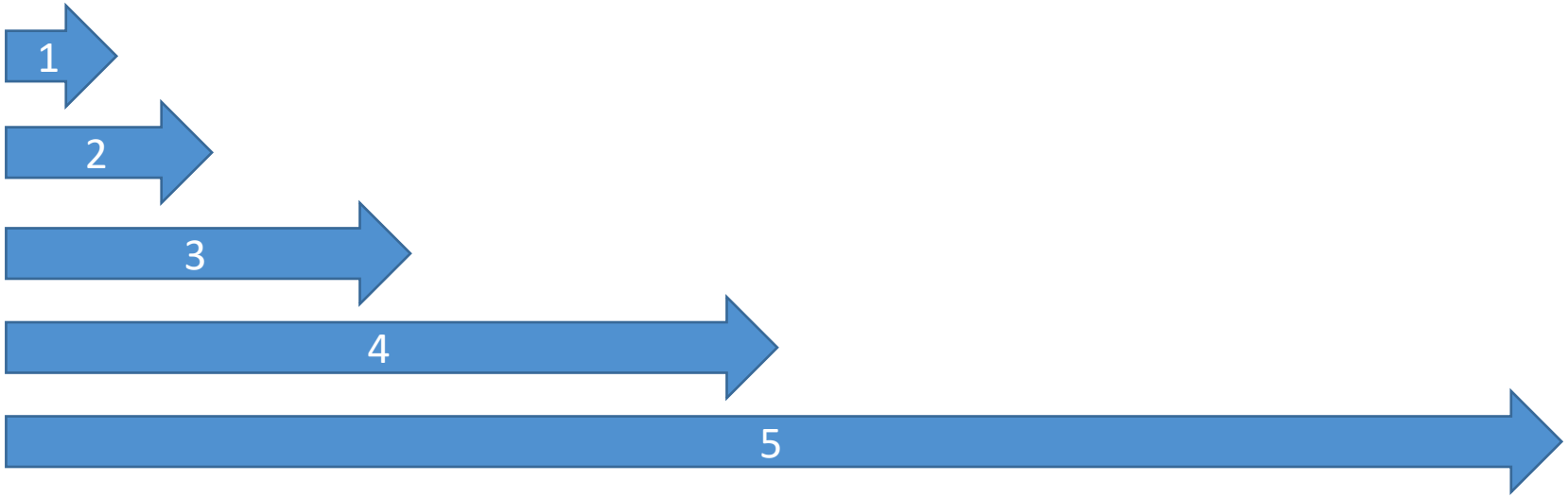
AWS Limits (Quotas)

- API Rate Limits
  - **DescribeInstances** API for EC2 has a limit of 100 calls per seconds
  - **GetObject** on S3 has a limit of 5500 GET per second per prefix
  - For Intermittent Errors: implement Exponential Backoff
  - For Consistent Errors: request an API throttling limit increase

- Service Quotas (Service Limits)
  - Running On-Demand Standard Instances: 1152 vCPU
  - You can request a service limit increase by **opening a ticket**
  - You can request a service quota increase by using the **Service Quotas API**

Exponential Backoff (any AWS service)

- If you get **ThrottlingException** intermittently, use exponential backoff
- Retry mechanism already included in AWS SDK API calls
- Must implement yourself if using the AWS API as-is or in specific cases
  - **Must only implement the retries on 5xx server errors and throttling**
  - Do not implement on the 4xx client errors



AWS CLI Credentials Provider Chain

- The CLI will look for credentials in this order

1. **Command line options** – --region, --output, and --profile
2. **Environment variables** – AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY, and AWS_SESSION_TOKEN
3. **CLI credentials file** –aws configure
   ~/.aws/credentials on Linux / Mac & C:\Users\user\.aws\credentials on Windows
4. **CLI configuration file** – aws configure
   ~/.aws/config on Linux / macOS & C:\Users\USERNAME\.aws\config on Windows
5. **Container credentials** – for ECS tasks
6. **Instance profile credentials** – for EC2 Instance Profiles

AWS SDK Default Credentials Provider Chain

- The Java SDK (example) will look for credentials in this order

1. **Java system properties** – aws.accessKeyId and aws.secretKey
2. **Environment variables** – AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY
3. **The default credential profiles file** – ex at: ~/.aws/credentials, shared by many SDK
4. **Amazon ECS container credentials** – for ECS containers
5. **Instance profile credentials**– used on EC2 instances

AWS Credentials Scenario

- An application deployed on an EC2 instance is using environment variables with credentials from an IAM user to call the Amazon S3 API.
- The IAM user has S3FullAccess permissions.
- The application only uses one S3 bucket, so according to best practices:
  - An IAM Role & EC2 Instance Profile was created for the EC2 instance
  - The Role was assigned the minimum permissions to access that one S3 bucket

- The IAM Instance Profile was assigned to the EC2 instance, but it still had access to all S3 buckets. Why?

the credentials chain is still giving priorities to the environment variables

AWS Credentials Best Practices

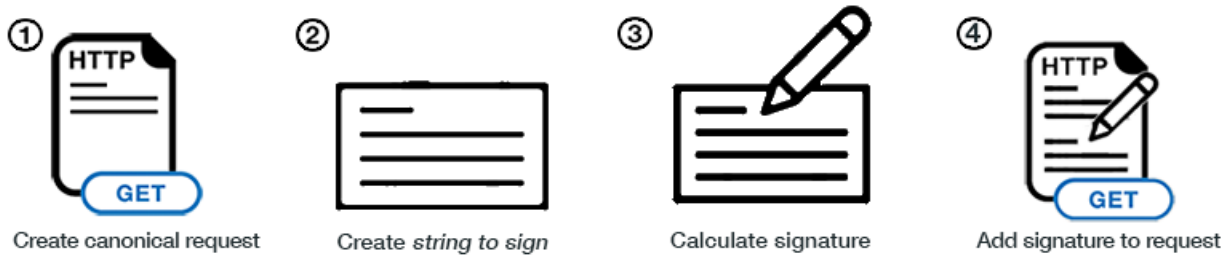
- Overall, NEVER EVER STORE AWS CREDENTIALS IN YOUR CODE
- Best practice is for credentials to be inherited from the credentials chain

- If using working within AWS, use IAM Roles
  - => EC2 Instances Roles for EC2 Instances
  - => ECS Roles for ECS tasks
  - => Lambda Roles for Lambda functions
- If working outside of AWS, use environment variables / named profiles

Signing AWS API requests

- When you call the AWS HTTP API, you sign the request so that AWS can identify you, using your AWS credentials (access key & secret key)
- Note: some requests to Amazon S3 don't need to be signed
- If you use the SDK or CLI, the HTTP requests are signed for you

- You should sign an AWS HTTP request using Signature v4 (SigV4)



SigV4 Request examples

- HTTP Header option (signature in Authorization header)

```
GET https://iam.amazonaws.com/?Action=ListUsers&Version=2010-05-08 HTTP/1.1
Authorization: AWS4-HMAC-SHA256 Credential=AKIDEXAMPLE/20150830/us-east-1/iam/aws4_request,
SignedHeaders=content-type;host;x-amz-date,
Signature=5d672d79c15b13162d9279b0855cfba6789a8edb4c82c400e06b5924a6f2b5d7
content-type: application/x-www-form-urlencoded; charset=utf-8
host: iam.amazonaws.com
x-amz-date: 20150830T123600Z
```

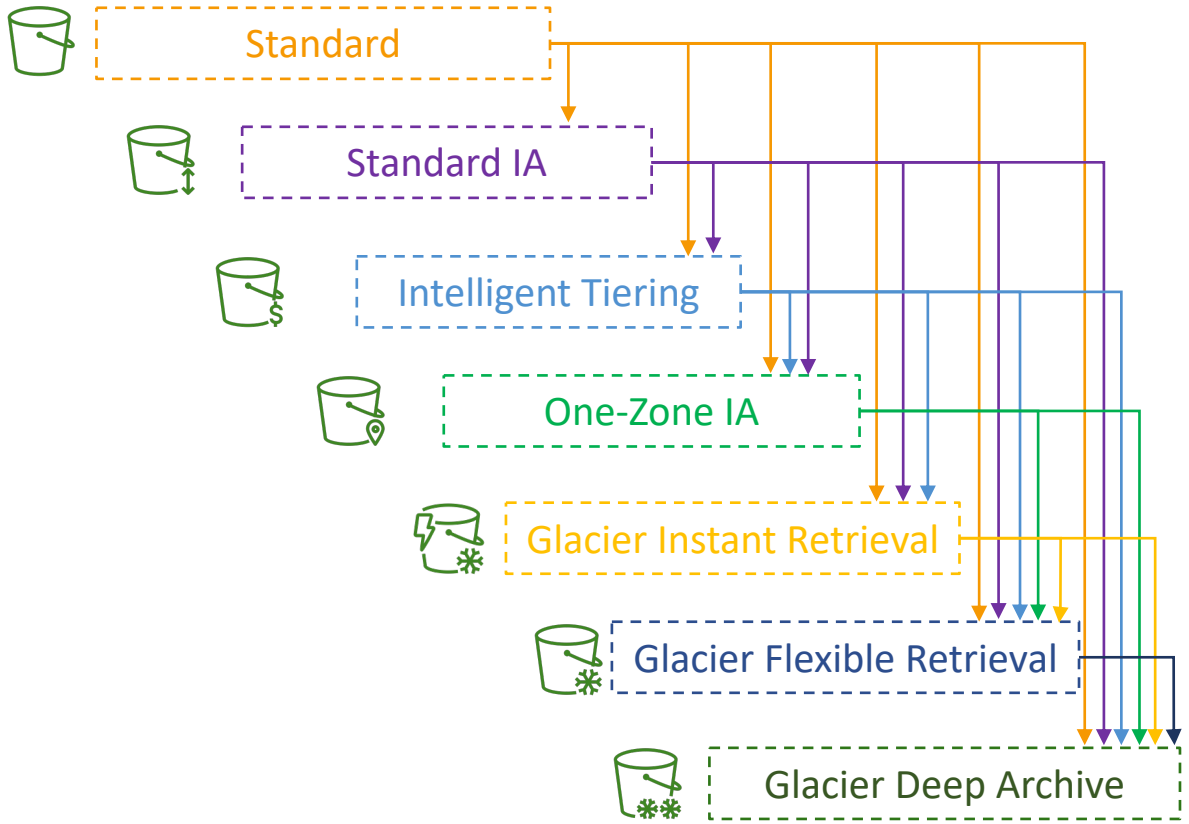
- Query String option, ex: S3 pre-signed URLs (signature in X-Amz-Signature)

```
GET https://iam.amazonaws.com?Action=ListUsers&Version=2010-05-08&
X-Amz-Algorithm=AWS4-HMAC-SHA256&
X-Amz-Credential=AKIDEXAMPLE%2F20150830%2Fus-east-1%2Fiam%2Faws4_request&
X-Amz-Date=20150830T123600Z&X-Amz-Expires=60&X-Amz-SignedHeaders=content-type%3Bhost&
X-Amz-Signature=37ac2f4fde00b0ac9bd9eadeb459b1bbee224158d66e7ae5fcadb70b2d181d02 HTTP/1.1
content-type: application/x-www-form-urlencoded; charset=utf-8
host: iam.amazonaws.com
```

Amazon S3 – Advanced

Amazon S3 – Moving between Storage Classes

- You can transition objects between storage classes
- For infrequently accessed object, move them to **Standard IA**
- For archive objects that you don't need fast access to, move them to **Glacier or Glacier Deep Archive**
- Moving objects can be automated using a **Lifecycle Rules**



Amazon S3 – Lifecycle Rules

- **Transition Actions** – configure objects to transition to another storage class
  - Move objects to Standard IA class 60 days after creation
  - Move to Glacier for archiving after 6 months

- **Expiration actions** – configure objects to expire (delete) after some time
  - Access log files can be set to delete after a 365 days
  - **Can be used to delete old versions of files (if versioning is enabled)**
  - Can be used to delete incomplete Multi-Part uploads

- Rules can be created for a certain prefix (example: *s3://mybucket/mp3/**)
- Rules can be created for certain objects Tags (example: *Department: Finance*)

Amazon S3 – Lifecycle Rules (Scenario 1)

- Your application on EC2 creates images thumbnails after profile photos are uploaded to Amazon S3. These thumbnails can be easily recreated, and only need to be kept for 60 days. The source images should be able to be immediately retrieved for these 60 days, and afterwards, the user can wait up to 6 hours. How would you design this?

- S3 source images can be on **Standard**, with a lifecycle configuration to transition them to **Glacier** after 60 days
- S3 thumbnails can be on **One-Zone IA**, with a lifecycle configuration to expire them (delete them) after 60 days

Amazon S3 – Lifecycle Rules (Scenario 2)

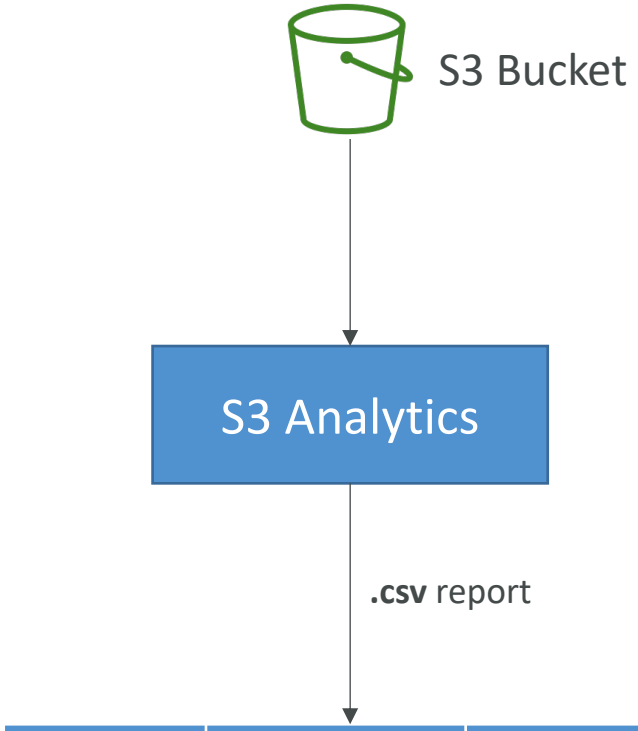
- A rule in your company states that you should be able to recover your deleted S3 objects immediately for 30 days, although this may happen rarely. After this time, and for up to 365 days, deleted objects should be recoverable within 48 hours.

- Enable S3 Versioning in order to have object versions, so that "deleted objects" are in fact hidden by a "delete marker" and can be recovered
- Transition the "noncurrent versions" of the object to Standard IA
- Transition afterwards the "noncurrent versions" to Glacier Deep Archive

Amazon S3 Analytics – Storage Class Analysis

- Help you decide when to transition objects to the right storage class
- Recommendations for **Standard** and **Standard IA**
  - Does NOT work for One-Zone IA or Glacier
- Report is updated daily
- 24 to 48 hours to start seeing data analysis

- Good first step to put together Lifecycle Rules (or improve them)!

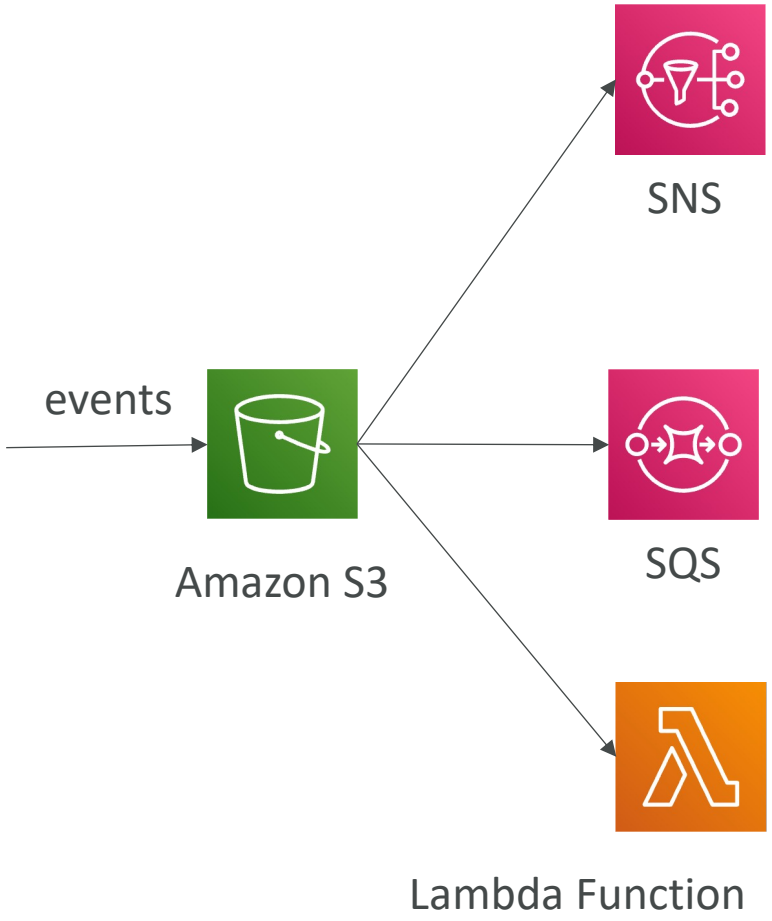


Date	StorageClass	ObjectAge
8/22/2022	STANDARD	000-014
8/25/2022	STANDARD	030-044
9/6/2022	STANDARD	120-149

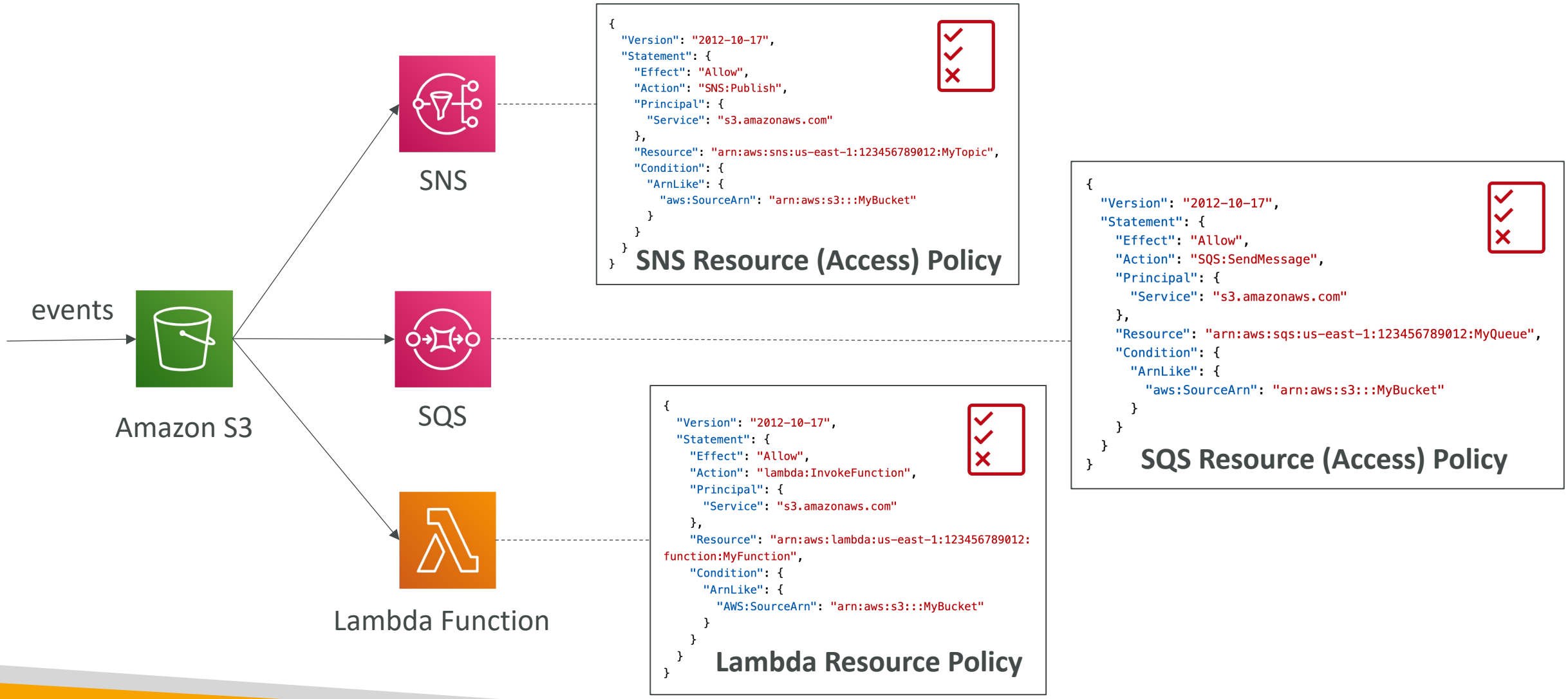
S3 Event Notifications

- S3:ObjectCreated, S3:ObjectRemoved, S3:ObjectRestore, S3:Replication…
- Object name filtering possible (*.jpg)
- Use case: generate thumbnails of images uploaded to S3
- **Can create as many "S3 events" as desired**

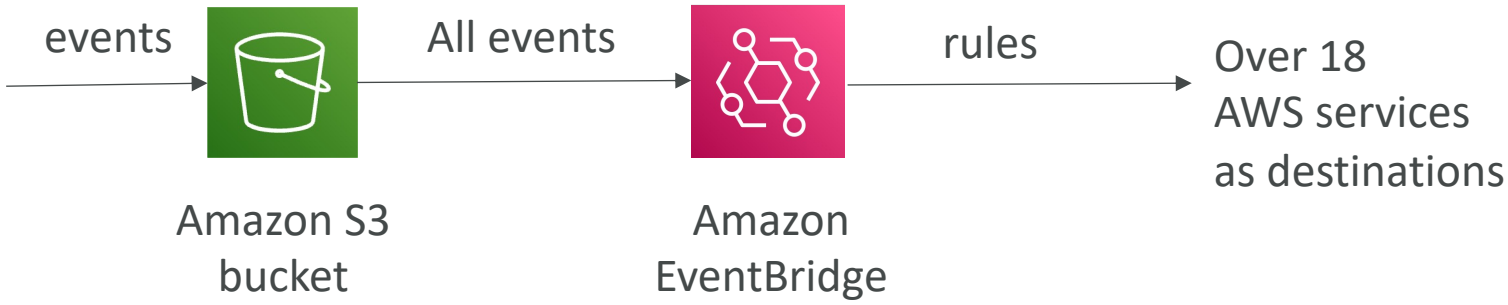
- S3 event notifications typically deliver events in seconds but can sometimes take a minute or longer



S3 Event Notifications – IAM Permissions



S3 Event Notifications with Amazon EventBridge



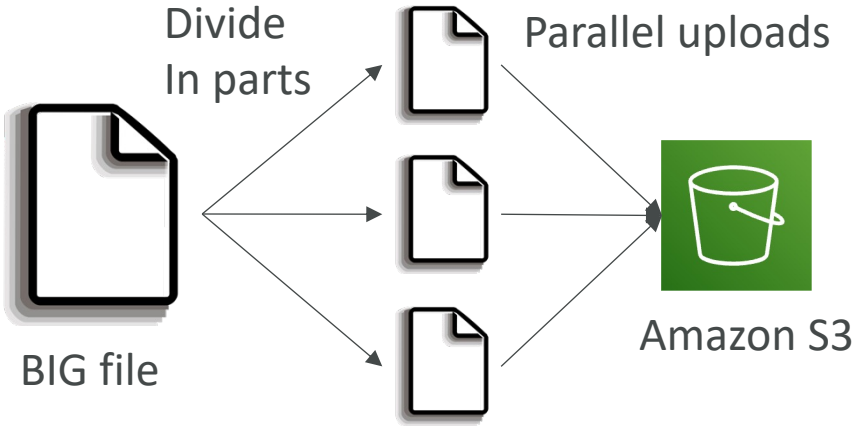
- **Advanced filtering** options with JSON rules (metadata, object size, name...)
- **Multiple Destinations** – ex Step Functions, Kinesis Streams / Firehose…
- **EventBridge Capabilities** – Archive, Replay Events, Reliable delivery

S3 – Baseline Performance

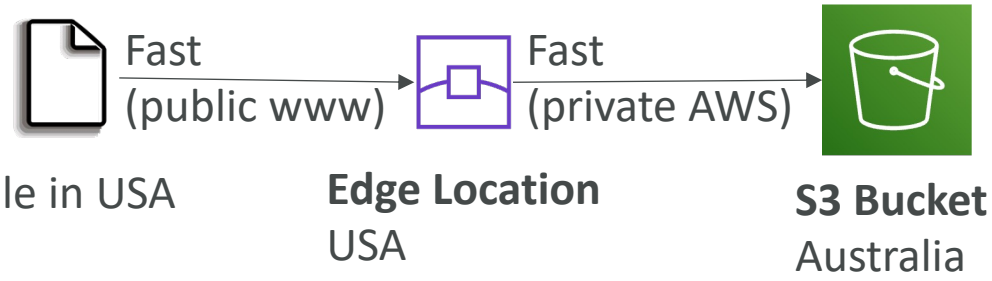
- Amazon S3 automatically scales to high request rates, latency 100-200 ms
- Your application can achieve at least **3,500 PUT/COPY/POST/DELETE or 5,500 GET/HEAD requests per second per prefix in a bucket.**
- There are no limits to the number of prefixes in a bucket.
- Example (object path => prefix):
  - bucket/folder1/sub1/file => /folder1/sub1/
  - bucket/folder1/sub2/file => /folder1/sub2/
  - bucket/1/file => /1/
  - bucket/2/file => /2/
- If you spread reads across all four prefixes evenly, you can achieve 22,000 requests per second for GET and HEAD

S3 Performance

- Multi-Part upload:
  - recommended for files > 100MB, must use for files > 5GB
  - Can help parallelize uploads (speed up transfers)



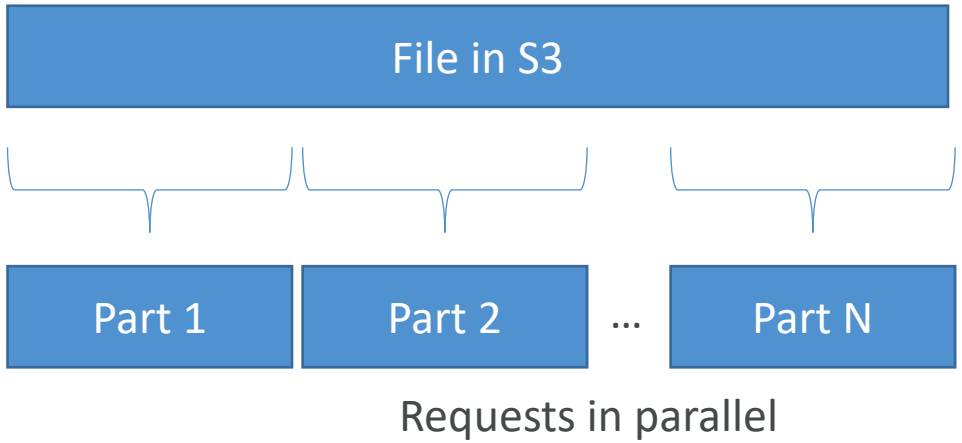
- S3 Transfer Acceleration
  - Increase transfer speed by transferring file to an AWS edge location which will forward the data to the S3 bucket in the target region
  - Compatible with multi-part upload



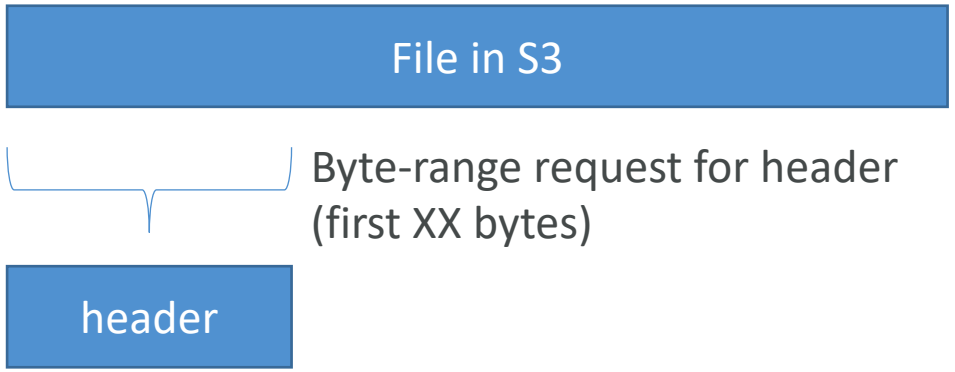
S3 Performance – S3 Byte-Range Fetches

- Parallelize GETs by requesting specific byte ranges
- Better resilience in case of failures

Can be used to speed up downloads



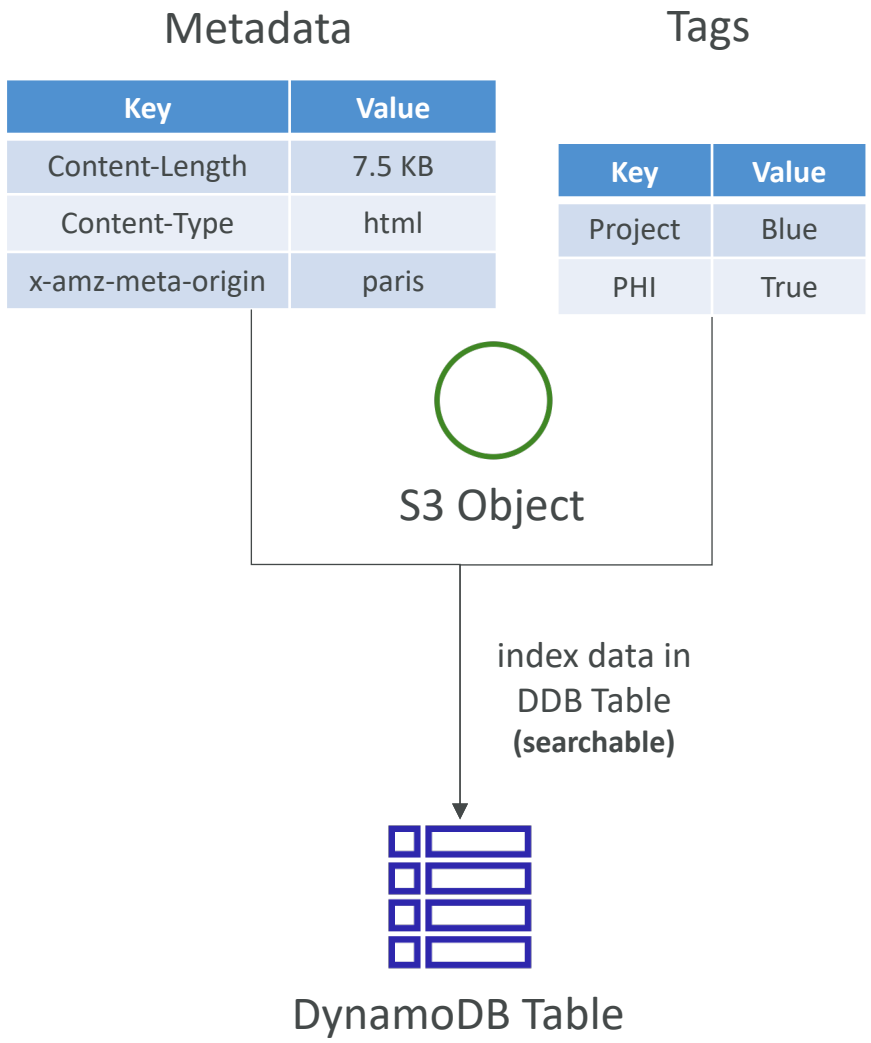
Can be used to retrieve only partial data (for example the head of a file)



S3 User-Defined Object Metadata & S3 Object Tags

- S3 User-Defined Object Metadata
  - When uploading an object, you can also assign metadata
  - Name-value (key-value) pairs
  - User-defined metadata names must begin with "x-amz-meta-"
  - Amazon S3 stores user-defined metadata keys in lowercase
  - Metadata can be retrieved while retrieving the object
- S3 Object Tags
  - Key-value pairs for objects in Amazon S3
  - Useful for fine-grained permissions (only access specific objects with specific tags)
  - Useful for analytics purposes (using S3 Analytics to group by tags)

- You cannot search the object metadata or object tags
- Instead, you must use an external DB as a search index such as DynamoDB



Amazon S3 – Security

Amazon S3 – Object Encryption

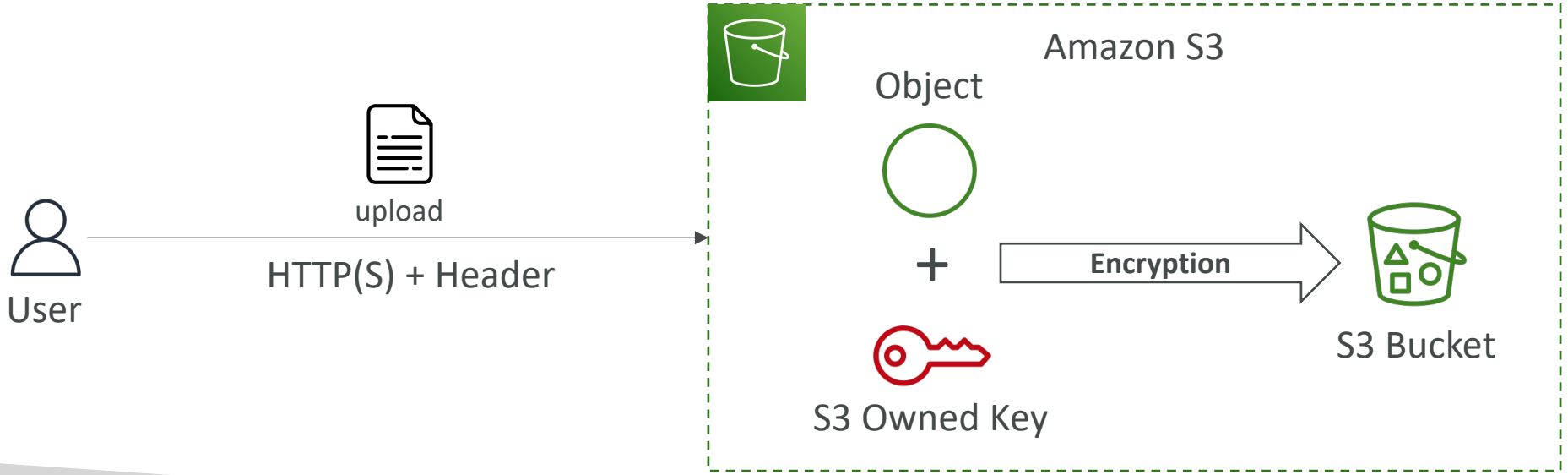
- You can encrypt objects in S3 buckets using one of 4 methods

- Server-Side Encryption (SSE)
  - Server-Side Encryption with Amazon S3-Managed Keys (SSE-S3) – Enabled by Default
    - Encrypts S3 objects using keys handled, managed, and owned by AWS
  - Server-Side Encryption with KMS Keys stored in AWS KMS (SSE-KMS)
    - Leverage AWS Key Management Service (AWS KMS) to manage encryption keys
  - Server-Side Encryption with Customer-Provided Keys (SSE-C)
    - When you want to manage your own encryption keys
- Client-Side Encryption

- It's important to understand which ones are for which situation for the exam

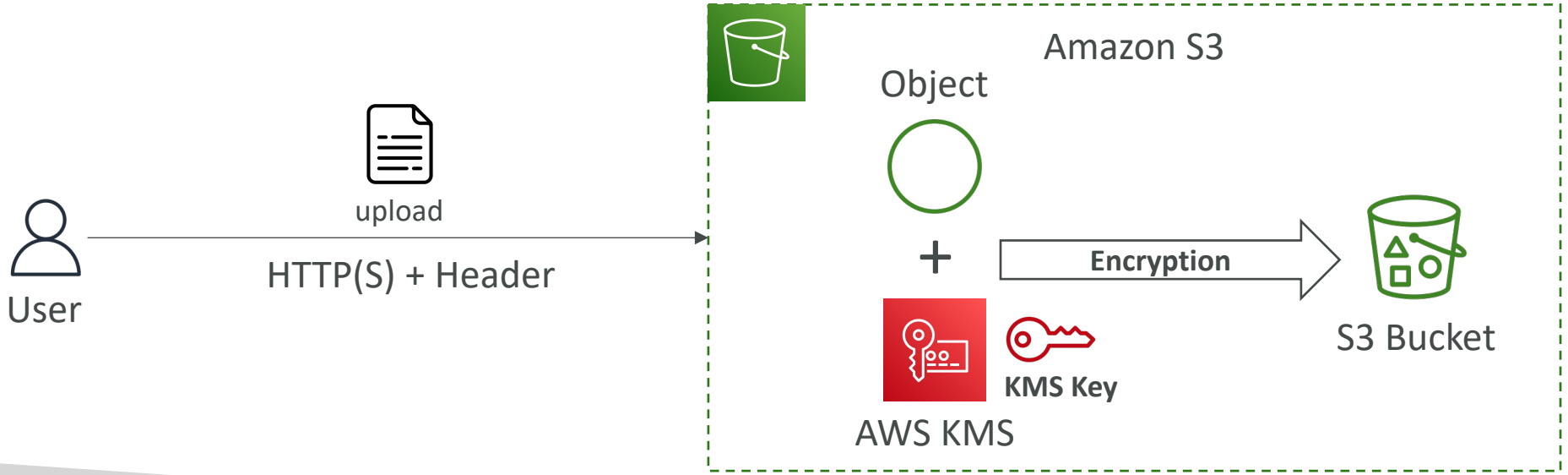
Amazon S3 Encryption – SSE-S3

- Encryption using keys handled, managed, and owned by AWS
- Object is encrypted server-side
- Encryption type is AES-256
- Must set header "x-amz-server-side-encryption": "AES256"
- Enabled by default for new buckets & new objects



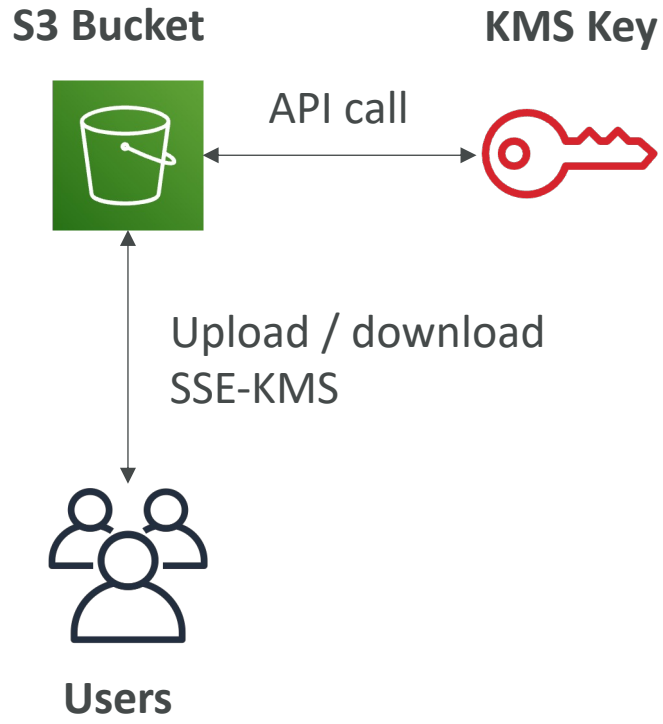
Amazon S3 Encryption – SSE-KMS

- Encryption using keys handled and managed by AWS KMS (Key Management Service)
- KMS advantages: user control + audit key usage using CloudTrail
- Object is encrypted server side
- Must set header **"x-amz-server-side-encryption": "aws:kms"**



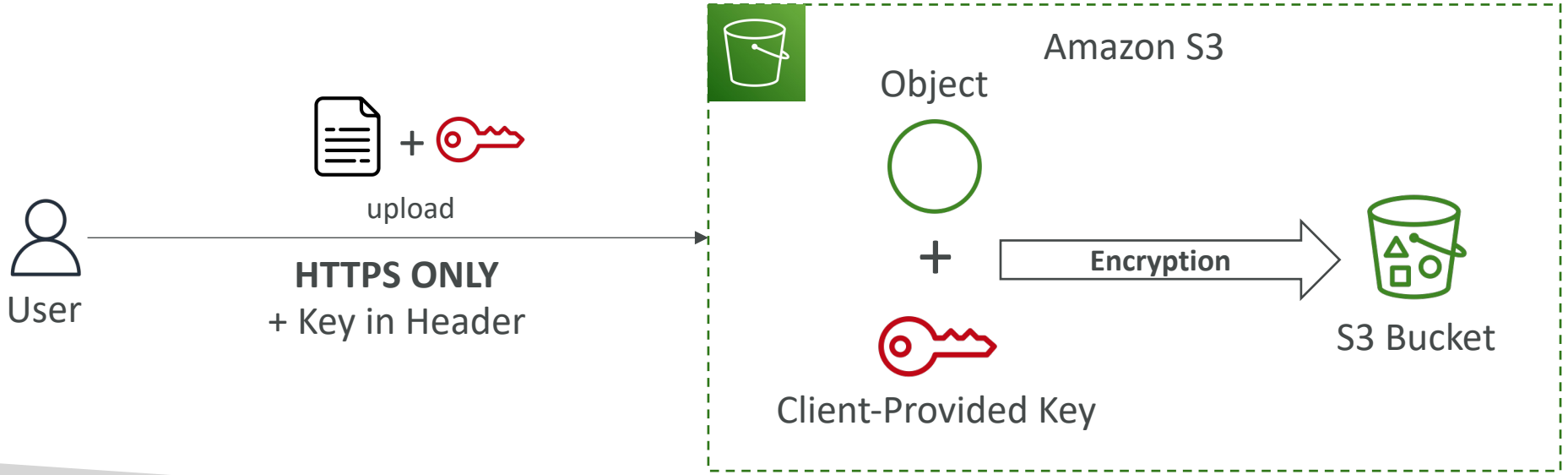
SSE-KMS Limitation

- If you use SSE-KMS, you may be impacted by the KMS limits
- When you upload, it calls the **GenerateDataKey** KMS API
- When you download, it calls the **Decrypt** KMS API
- Count towards the KMS quota per second (5500, 10000, 30000 req/s based on region)
- You can request a quota increase using the Service Quotas Console



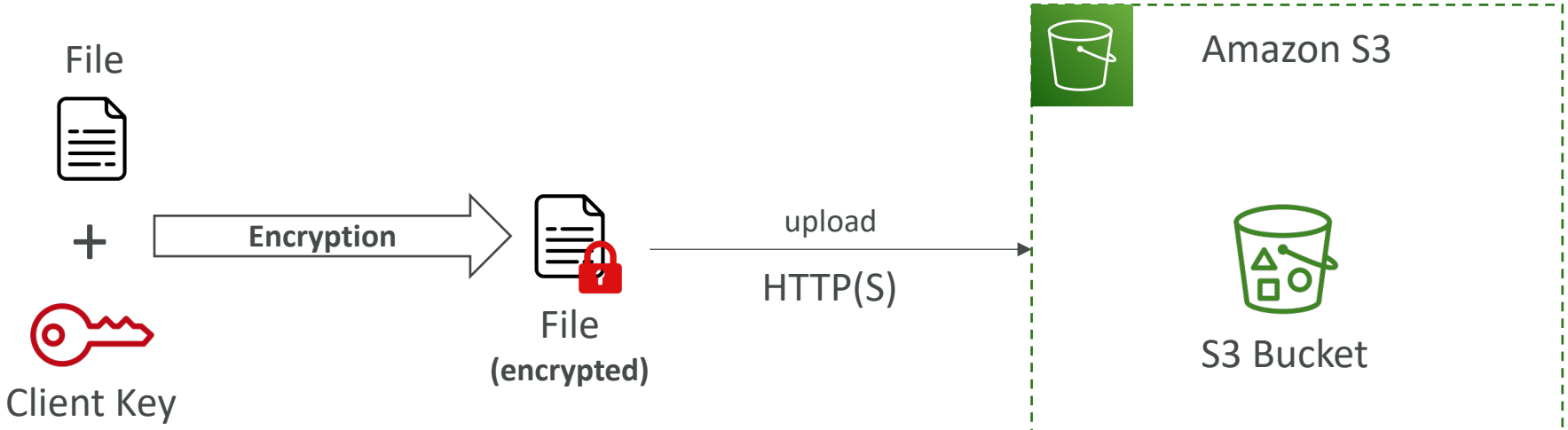
Amazon S3 Encryption – SSE-C

- Server-Side Encryption using keys fully managed by the customer outside of AWS
- Amazon S3 does **NOT** store the encryption key you provide
- **HTTPS must be used**
- Encryption key must provided in HTTP headers, for every HTTP request made



Amazon S3 Encryption – Client-Side Encryption

- Use client libraries such as **Amazon S3 Client-Side Encryption Library**
- Clients must encrypt data themselves before sending to Amazon S3
- Clients must decrypt data themselves when retrieving from Amazon S3
- Customer fully manages the keys and encryption cycle



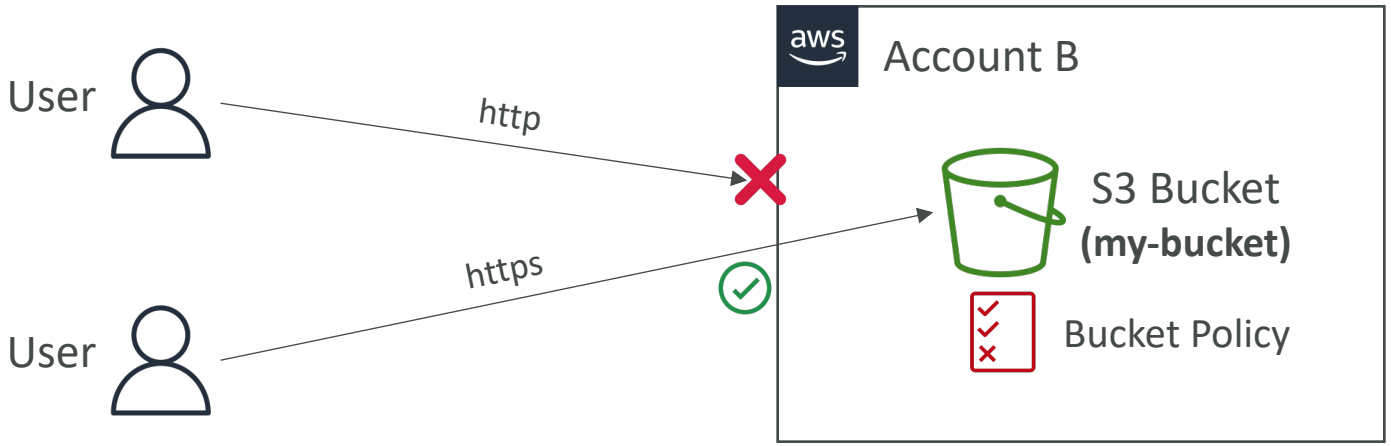
Amazon S3 – Encryption in transit (SSL/TLS)

- Encryption in flight is also called SSL/TLS

- Amazon S3 exposes two endpoints:
  - HTTP Endpoint – non encrypted
  - HTTPS Endpoint – encryption in flight

- HTTPS is recommended
- HTTPS is mandatory for SSE-C
- Most clients would use the HTTPS endpoint by default

Amazon S3 – Force Encryption in Transit aws:SecureTransport




```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::my-bucket/*",
            "Condition": {
                "Bool": {
                    "aws:SecureTransport": "false"
                }
            }
        }
    ]
}
```

Amazon S3 – Default Encryption vs. Bucket Policies

- SSE-S3 encryption is automatically applied to new objects stored in S3 bucket
- Optionally, you can "force encryption" using a bucket policy and refuse any API call to PUT an S3 object without encryption headers (SSE-KMS or SSE-C)

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": "s3:PutObject",
            "Principal": "*",
            "Resource": "arn:aws:s3:::my-bucket/*",
            "Condition": {
                "StringNotEquals": {
                    "s3:x-amz-server-side-encryption": "aws:kms"
                }
            }
        }
    ]
}
```

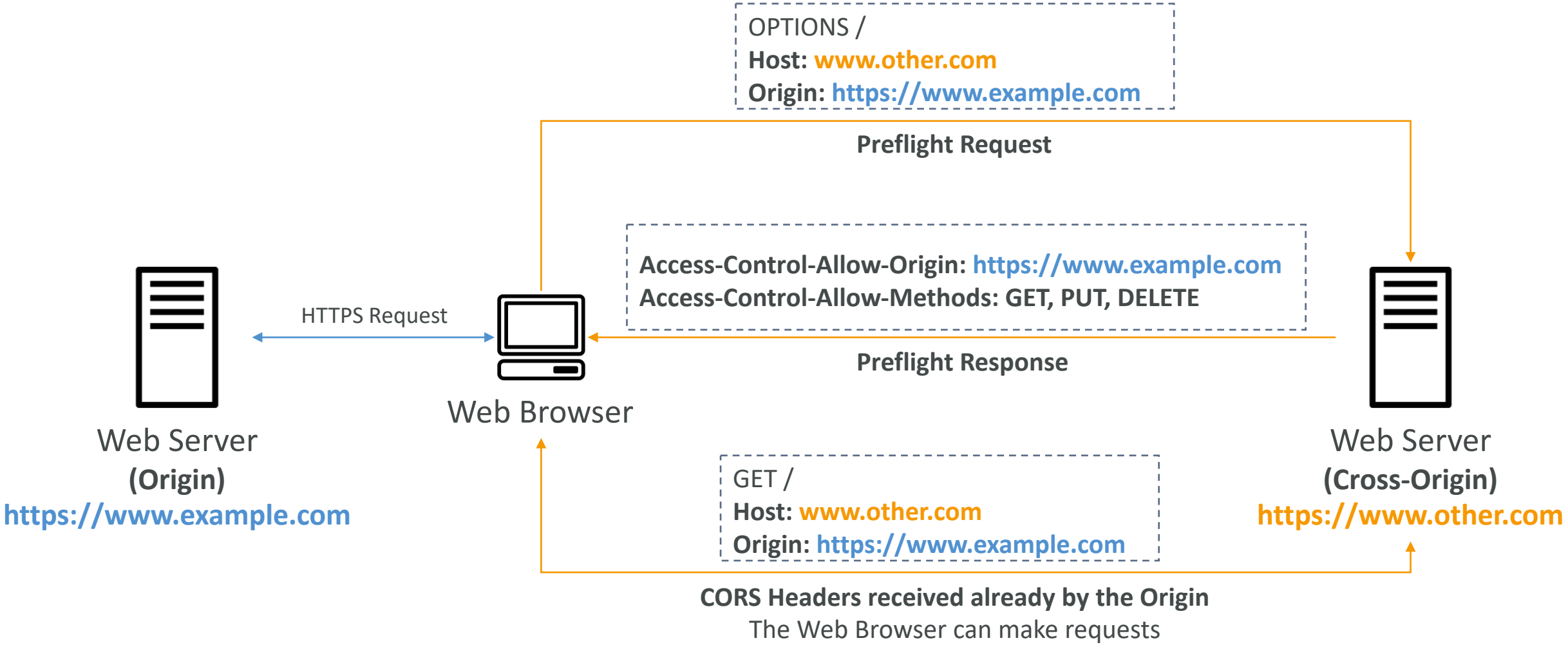
```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": "s3:PutObject",
            "Principal": "*",
            "Resource": "arn:aws:s3:::my-bucket/*",
            "Condition": {
                "Null": {
                    "s3:x-amz-server-side-encryption-customer-algorithm": "true"
                }
            }
        }
    ]
}
```

- Note: Bucket Policies are evaluated before "Default Encryption"

What is CORS?

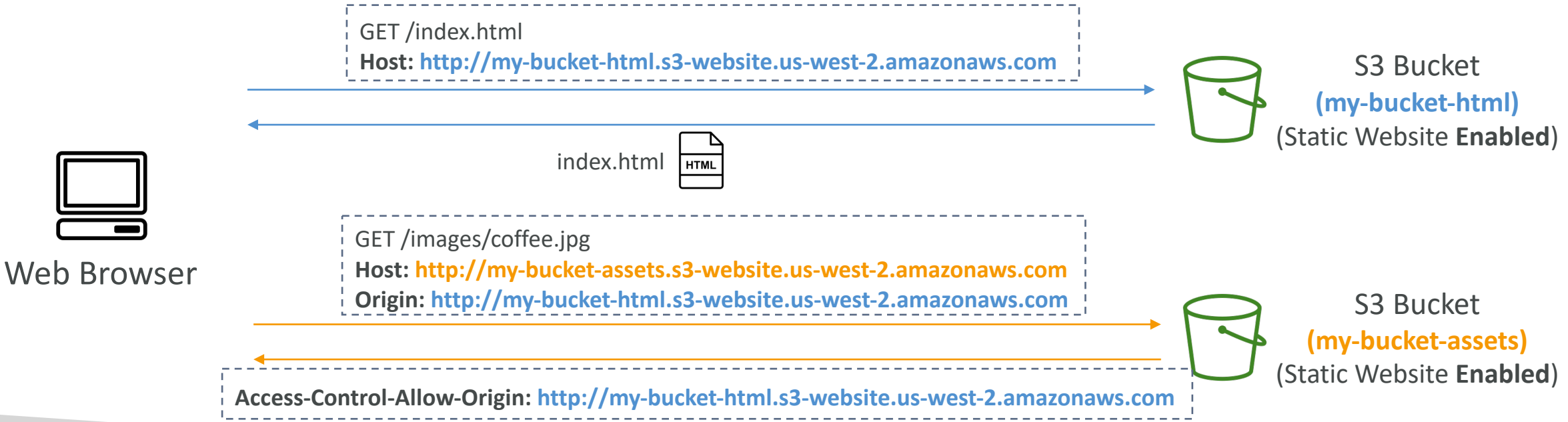
- Cross-Origin Resource Sharing (CORS)
- Origin = scheme (protocol) + host (domain) + port
  - example: https://www.example.com (implied port is 443 for HTTPS, 80 for HTTP)
- **Web Browser** based mechanism to allow requests to other origins while visiting the main origin
- Same origin: http://example.com/app1 & http://example.com/app2
- Different origins: http://www.example.com & http://other.example.com
- The requests won't be fulfilled unless the other origin allows for the requests, using **CORS Headers** (example: **Access-Control-Allow-Origin**)

What is CORS?



Amazon S3 – CORS

- If a client makes a cross-origin request on our S3 bucket, we need to enable the correct CORS headers
- It's a popular exam question
- You can allow for a specific origin or for * (all origins)



Amazon S3 – MFA Delete

- **MFA (Multi-Factor Authentication)** – force users to generate a code on a device (usually a mobile phone or hardware) before doing important operations on S3
- MFA will be required to:
  - Permanently delete an object version
  - Suspend Versioning on the bucket
- MFA won't be required to:
  - Enable Versioning
  - List deleted versions
- To use MFA Delete, **Versioning must be enabled** on the bucket
- **Only the bucket owner (root account) can enable/disable MFA Delete**

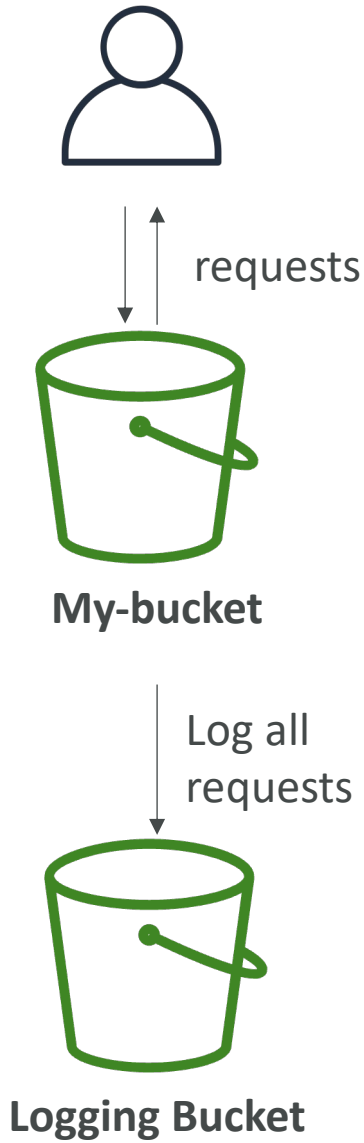
Google Authenticator

MFA Hardware Device

S3 Access Logs

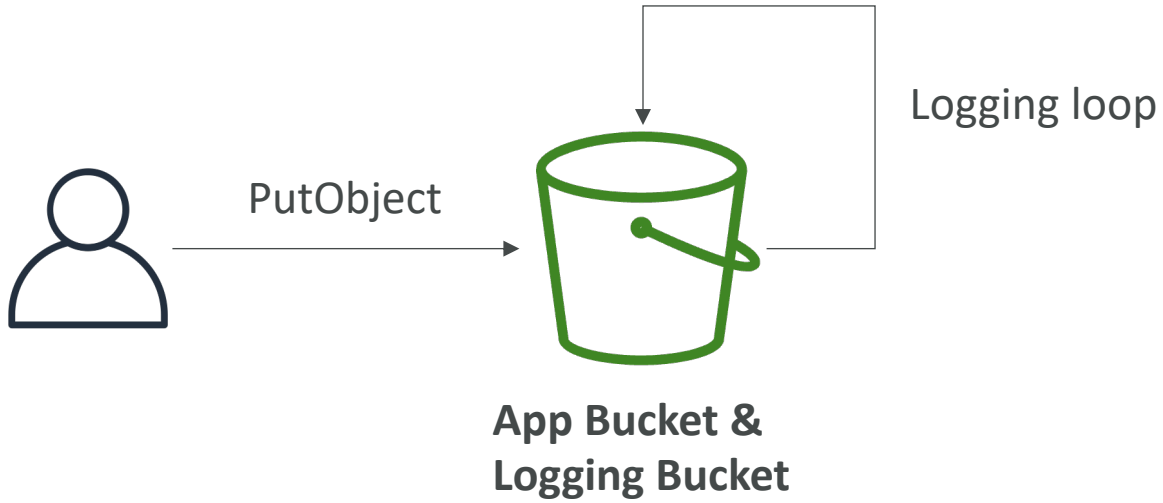
- For audit purpose, you may want to log all access to S3 buckets
- Any request made to S3, from any account, authorized or denied, will be logged into another S3 bucket
- That data can be analyzed using data analysis tools…
- The target logging bucket must be in the same AWS region

- The log format is at: https://docs.aws.amazon.com/AmazonS3/latest/dev/LogFormat.html



S3 Access Logs: Warning

- Do not set your logging bucket to be the monitored bucket
- It will create a logging loop, and **your bucket will grow exponentially**

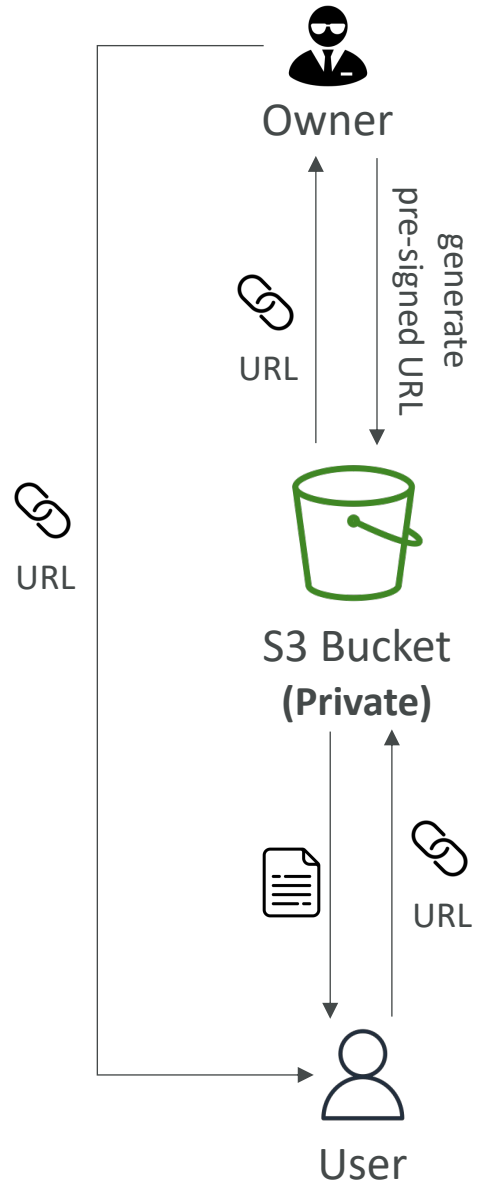


Do not try this at home ☺

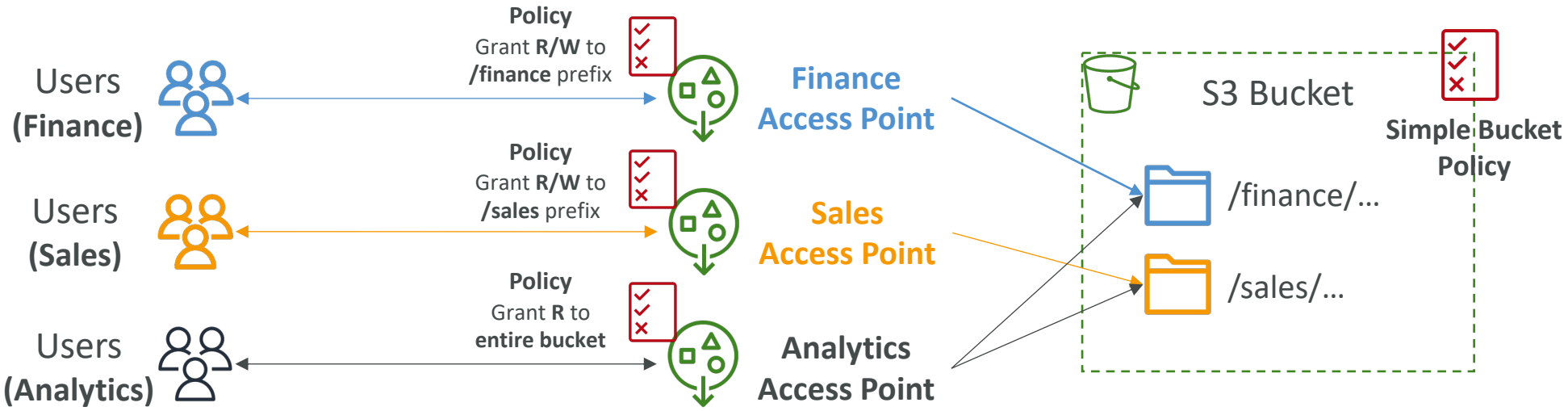
Amazon S3 – Pre-Signed URLs

- Generate pre-signed URLs using the **S3 Console, AWS CLI or SDK**
- **URL Expiration**
  - **S3 Console** – 1 min up to 720 mins (12 hours)
  - **AWS CLI** – configure expiration with *--expires-in* parameter in seconds (default 3600 secs, max. 604800 secs ~ 168 hours)
- Users given a pre-signed URL inherit the permissions of the user that generated the URL for GET / PUT

- Examples:
  - Allow only logged-in users to download a premium video from your S3 bucket
  - Allow an ever-changing list of users to download files by generating URLs dynamically
  - Allow temporarily a user to upload a file to a precise location in your S3 bucket



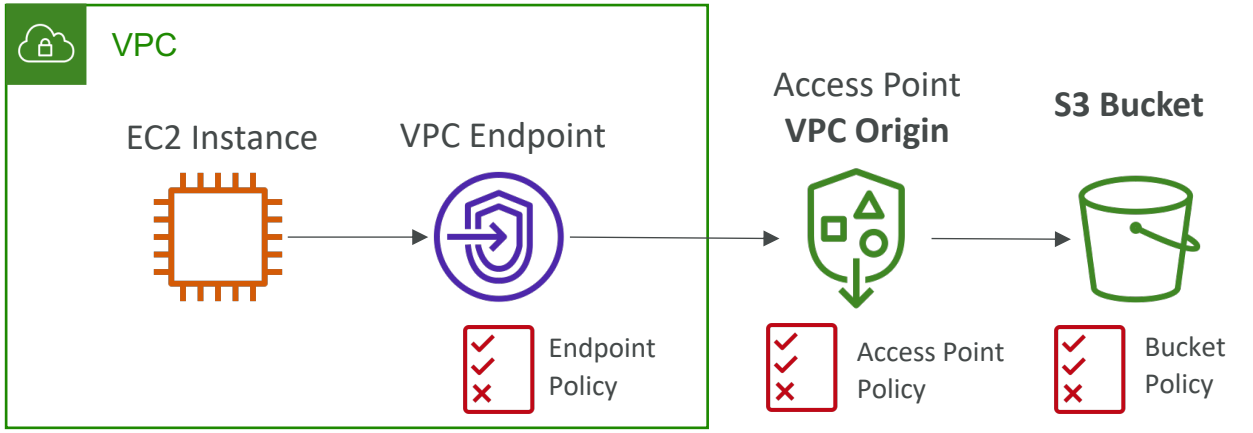
S3 – Access Points



- Access Points simplify security management for S3 Buckets
- Each Access Point has:
  - its own DNS name (Internet Origin or VPC Origin)
  - an access point policy (similar to bucket policy) – manage security at scale

S3 – Access Points – VPC Origin

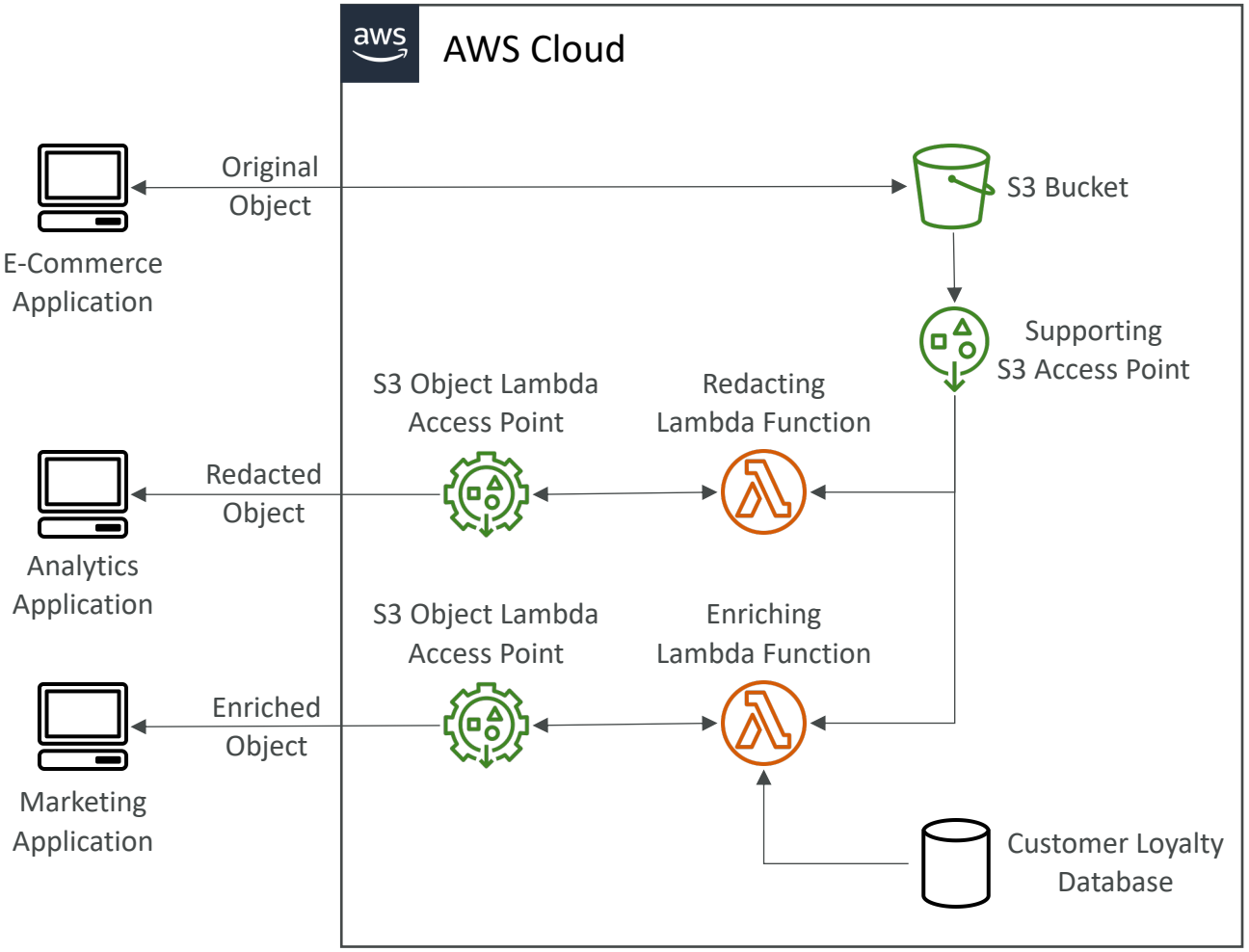
- We can define the access point to be accessible only from within the VPC
- You must create a VPC Endpoint to access the Access Point (Gateway or Interface Endpoint)
- The VPC Endpoint Policy must allow access to the target bucket and Access Point




```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Principal": "*",
            "Action": [
                "s3:GetObject"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:s3:::awsexamplebucket1/*",
                "arn:aws:s3:us-west-2:123456789012:accesspoint/example-vpc-ap/object/*"
            ]
        }
    ]
}
```

S3 Object Lambda

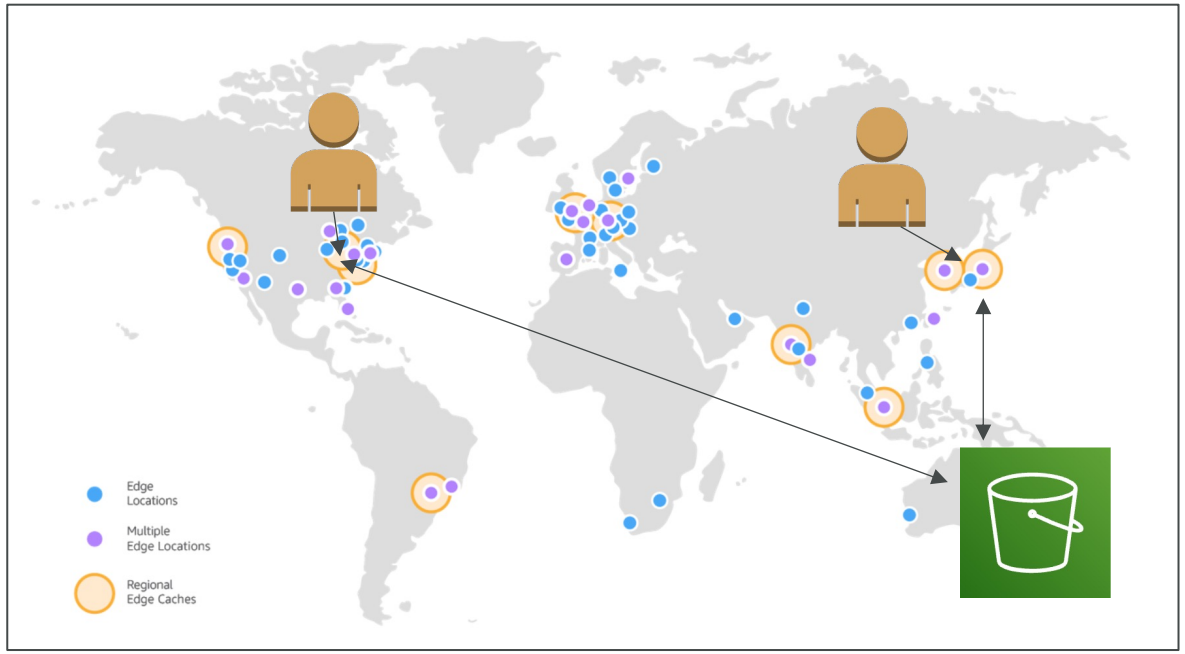
- Use AWS Lambda Functions to change the object before it is retrieved by the caller application
- Only one S3 bucket is needed, on top of which we create **S3 Access Point** and **S3 Object Lambda Access Points.**
- Use Cases:
  - Redacting personally identifiable information for analytics or non-production environments.
  - Converting across data formats, such as converting XML to JSON.
  - Resizing and watermarking images on the fly using caller-specific details, such as the user who requested the object.



Amazon CloudFront

Amazon CloudFront

- Content Delivery Network (CDN)
- Improves read performance, content is cached at the edge
- Improves users experience
- 216 Point of Presence globally (edge locations)
- DDoS protection (because worldwide), integration with Shield, AWS Web Application Firewall

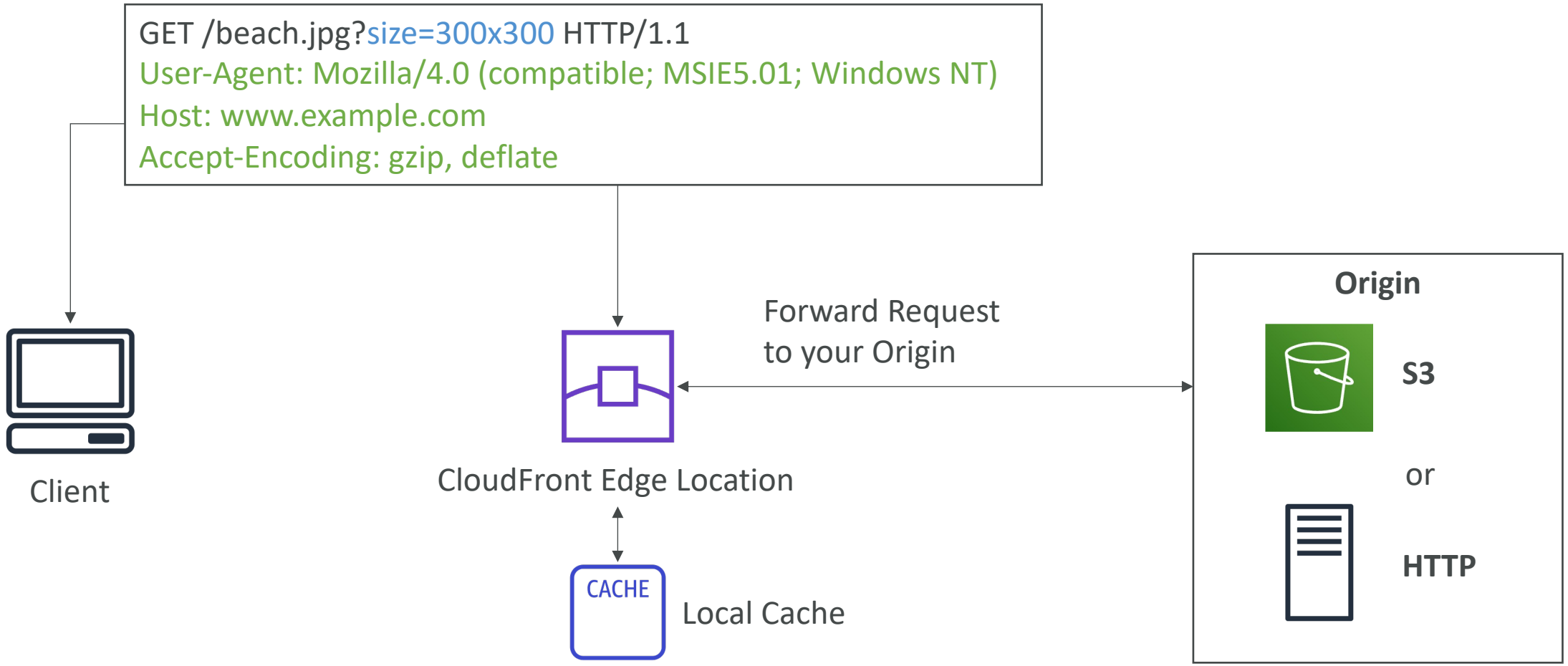


Source: https://aws.amazon.com/cloudfront/features/?nc=sn&loc=2

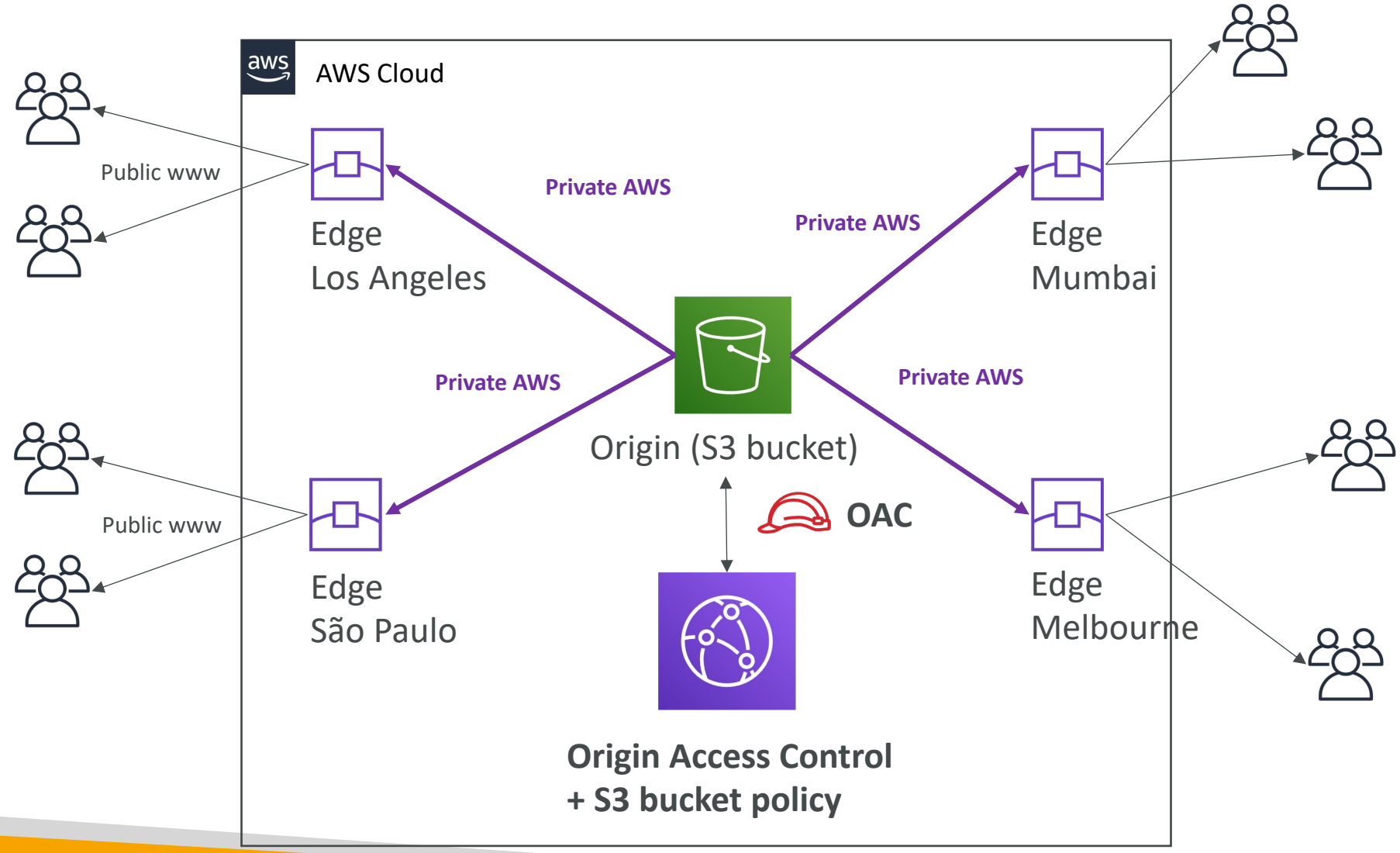
CloudFront – Origins

- S3 bucket
  - For distributing files and caching them at the edge
  - Enhanced security with CloudFront **Origin Access Control** (OAC)
  - OAC is replacing Origin Access Identity (OAI)
  - CloudFront can be used as an ingress (to upload files to S3)
- Custom Origin (HTTP)
  - Application Load Balancer
  - EC2 instance
  - S3 website (must first enable the bucket as a static S3 website)
  - Any HTTP backend you want

CloudFront at a high level



CloudFront – S3 as an Origin



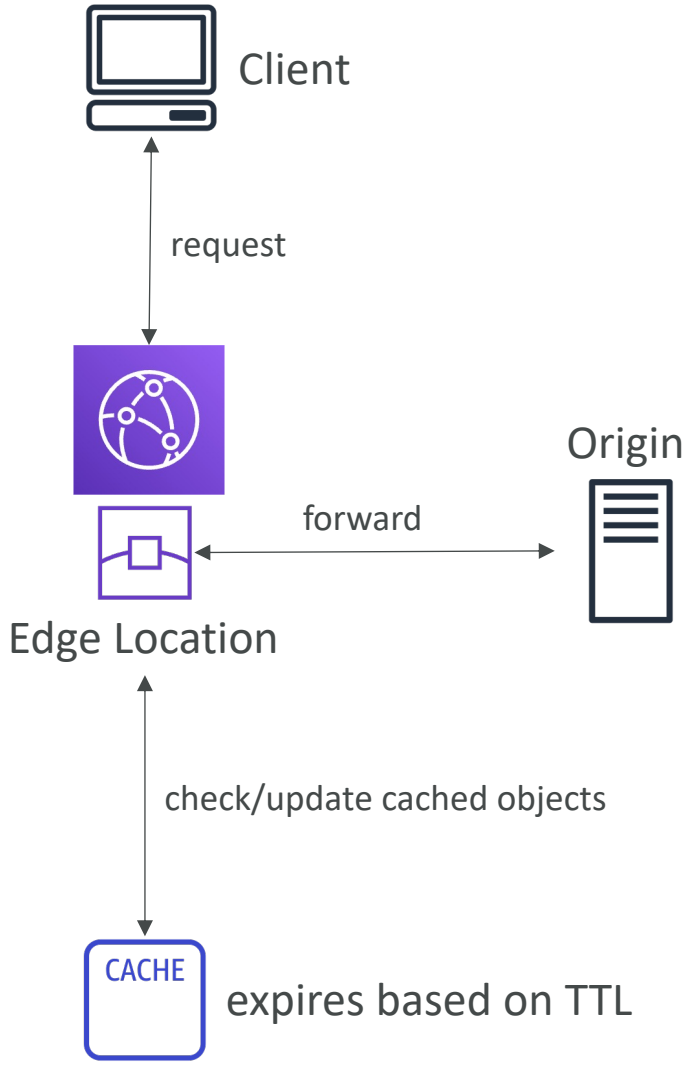
CloudFront vs S3 Cross Region Replication

- CloudFront:
  - Global Edge network
  - Files are cached for a TTL (maybe a day)
  - **Great for static content that must be available everywhere**

- S3 Cross Region Replication:
  - Must be setup for each region you want replication to happen
  - Files are updated in near real-time
  - Read only
  - **Great for dynamic content that needs to be available at low-latency in few regions**

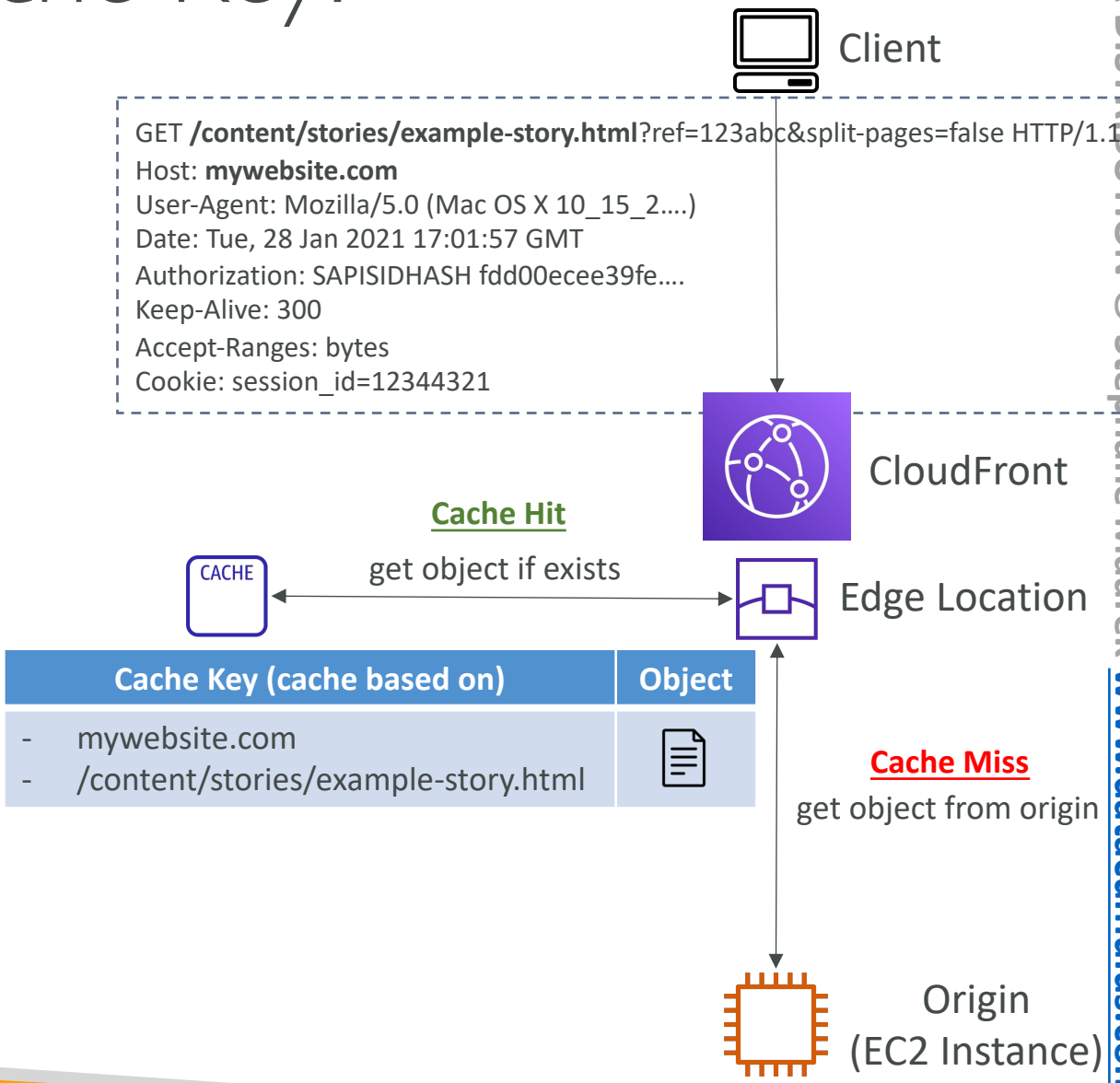
CloudFront Caching

- The cache lives at each CloudFront **Edge Location**
- CloudFront identifies each object in the cache using the **Cache Key** (see next slide)
- You want to maximize the Cache Hit ratio to minimize requests to the origin
- You can invalidate part of the cache using the **CreateInvalidation** API



What is CloudFront Cache Key?

- A unique identifier for every object in the cache
- By default, consists of **hostname + resource portion of the URL**
- If you have an application that serves up content that varies based on user, device, language, location…
- You can add other elements (HTTP headers, cookies, query strings) to the Cache Key using **CloudFront Cache Policies**



CloudFront Policies – Cache Policy

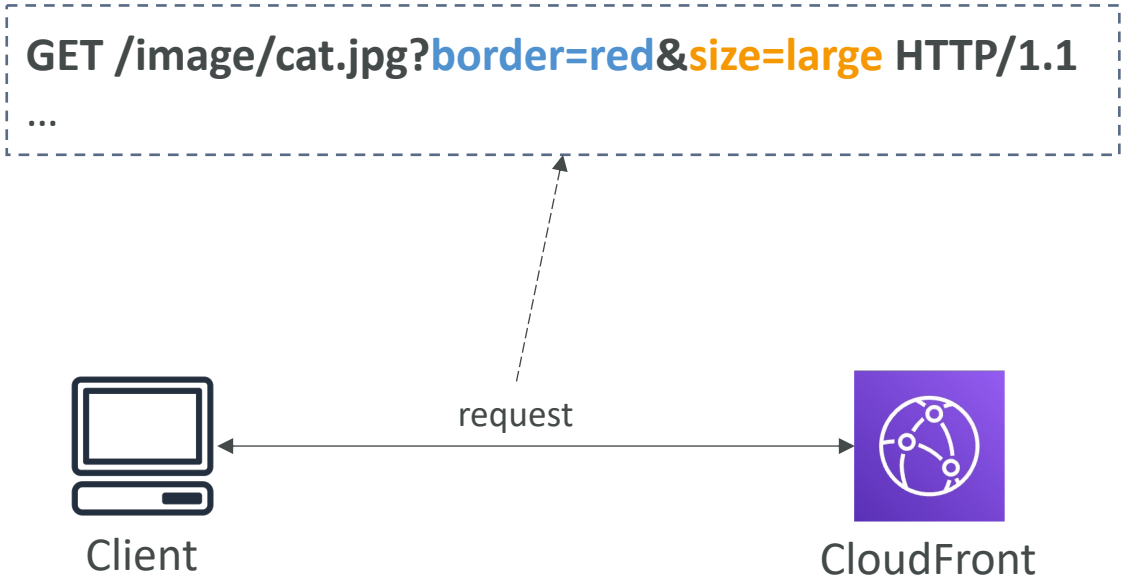
- Cache based on:
  - **HTTP Headers:** None – Whitelist
  - **Cookies:** None – Whitelist – Include All-Except – All
  - **Query Strings:** None – Whitelist – Include All-Except – All
- Control the TTL (0 seconds to 1 year), can be set by the origin using the **Cache-Control** header, **Expires** header…
- Create your own policy or use Predefined Managed Policies
- All HTTP headers, cookies, and query strings that you include in the Cache Key are automatically included in origin requests

CloudFront Caching – Cache Policy HTTP Headers

GET /blogs/myblog.html HTTP/1.1
Host: mywebsite.com
User-Agent: Mozilla/5.0 (Mac OS X 10_15_2….)
Date: Tue, 28 Jan 2021 17:01:57 GMT
Authorization: SAPISIDHASH fdd00ecee39fe….
Keep-Alive: 300
Language: fr-fr

- None:
  - Don't include any headers in the Cache Key (except default)
  - Headers are not forwarded (except default)
  - Best caching performance
- Whitelist:
  - only specified headers included in the Cache Key
  - Specified headers are also forwarded to Origin

CloudFront Cache – Cache Policy Query Strings



- None
  - Don't include any query strings in the Cache Key
  - Query strings are not forwarded
- Whitelist
  - Only specified query strings included in the Cache Key
  - Only specified query strings are forwarded
- Include All-Except
  - Include all query strings in the Cache Key except the specified list
  - All query strings are forwarded except the specified list
- All
  - Include all query strings in the Cache Key
  - All query strings are forwarded
  - Worst caching performance

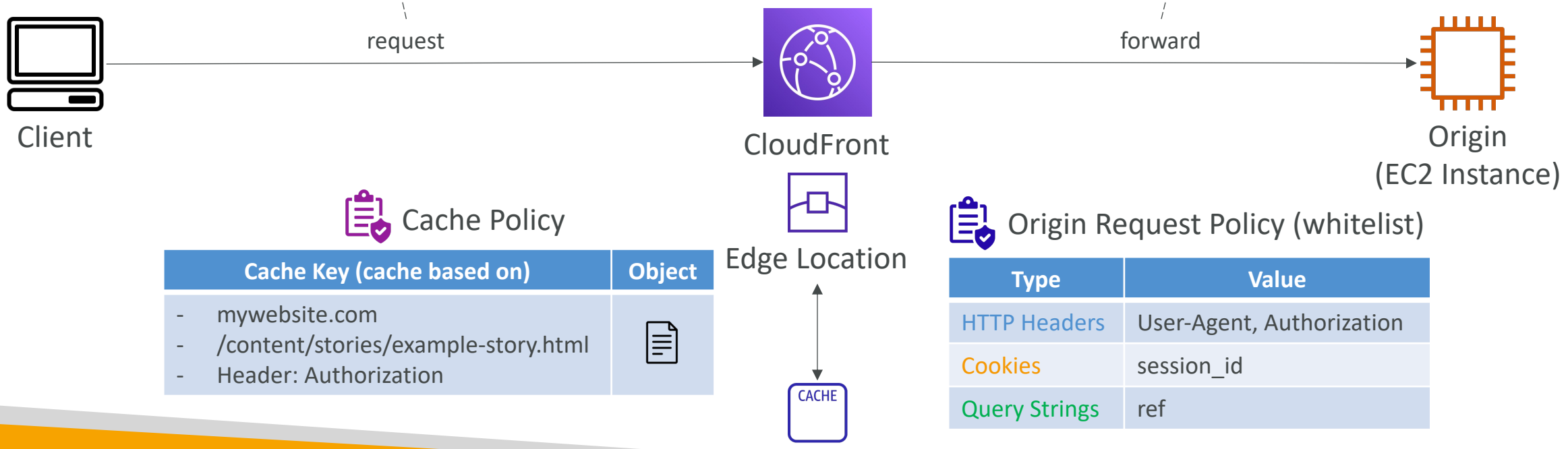
CloudFront Policies – Origin Request Policy

- Specify values that you want to include in origin requests **without including them in the Cache Key (no duplicated cached content)**
- You can include:
  - **HTTP headers:** None – Whitelist – All viewer headers options
  - **Cookies:** None – Whitelist – All
  - **Query Strings:** None – Whitelist – All
- Ability to add CloudFront HTTP headers and Custom Headers to an origin request that were not included in the viewer request
- Create your own policy or use Predefined Managed Policies

Cache Policy vs. Origin Request Policy

GET **/content/stories/example-story.html**?ref=123abc&split-pages=false HTTP/1.1
Host: **mywebsite.com**
User-Agent: Mozilla/5.0 (Mac OS X 10_15_2….)
Date: Tue, 28 Jan 2021 17:01:57 GMT
Authorization: SAPISIDHASH fdd00ecee39fe….
Keep-Alive: 300
Accept-Ranges: bytes
Cookie: session_id=12344321

GET **/content/stories/example-story.html**?ref=123abc HTTP/1.1
Host: **mywebsite.com**
User-Agent: Mozilla/5.0 (Mac OS X 10_15_2….)
Authorization: SAPISIDHASH fdd00ecee39fe….
Cookie: session_id=12344321



Cache Policy

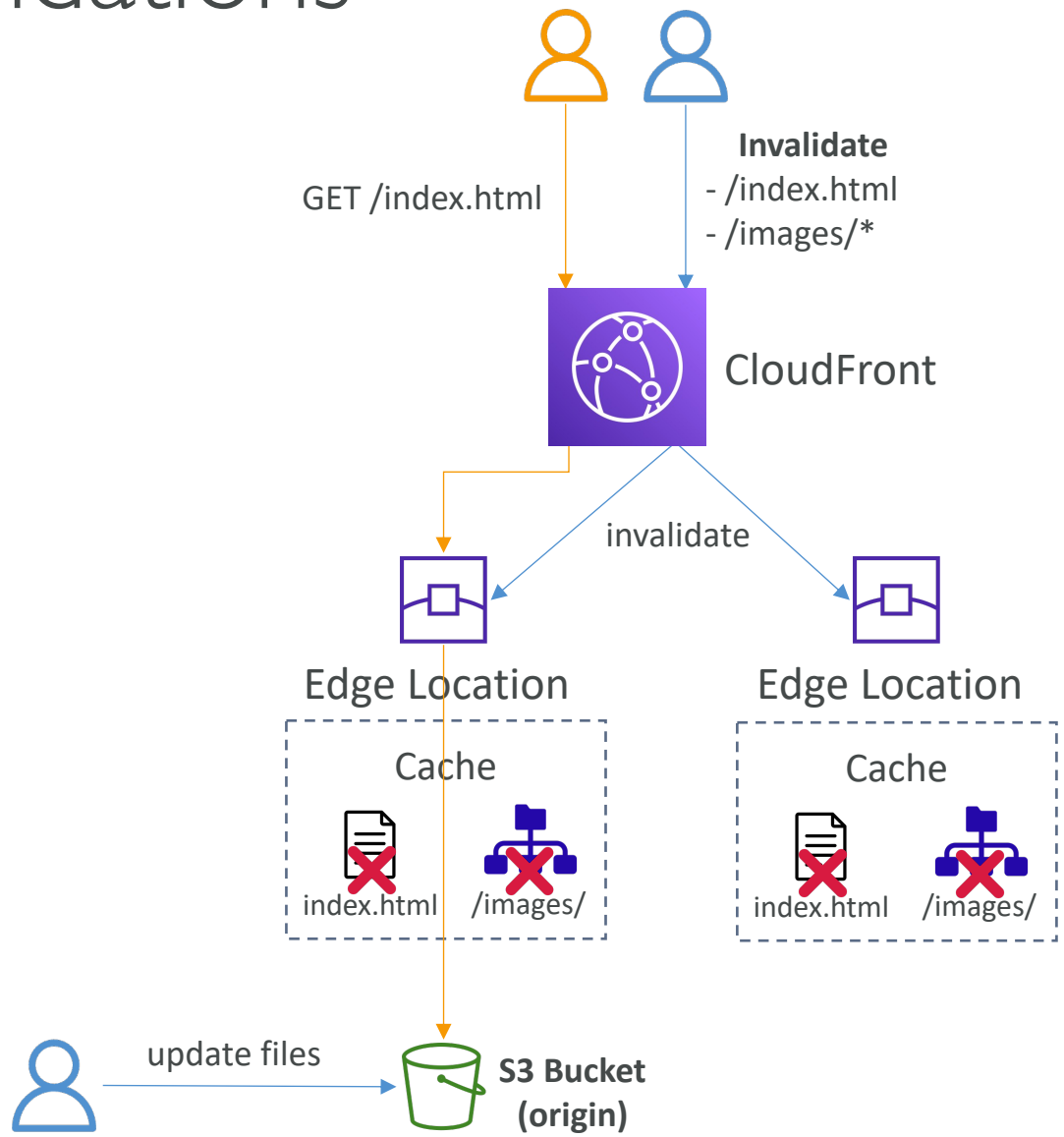
Cache Key (cache based on)	Object
- mywebsite.com - /content/stories/example-story.html - Header: Authorization	

Origin Request Policy (whitelist)

Type	Value
HTTP Headers	User-Agent, Authorization
Cookies	session_id
Query Strings	ref

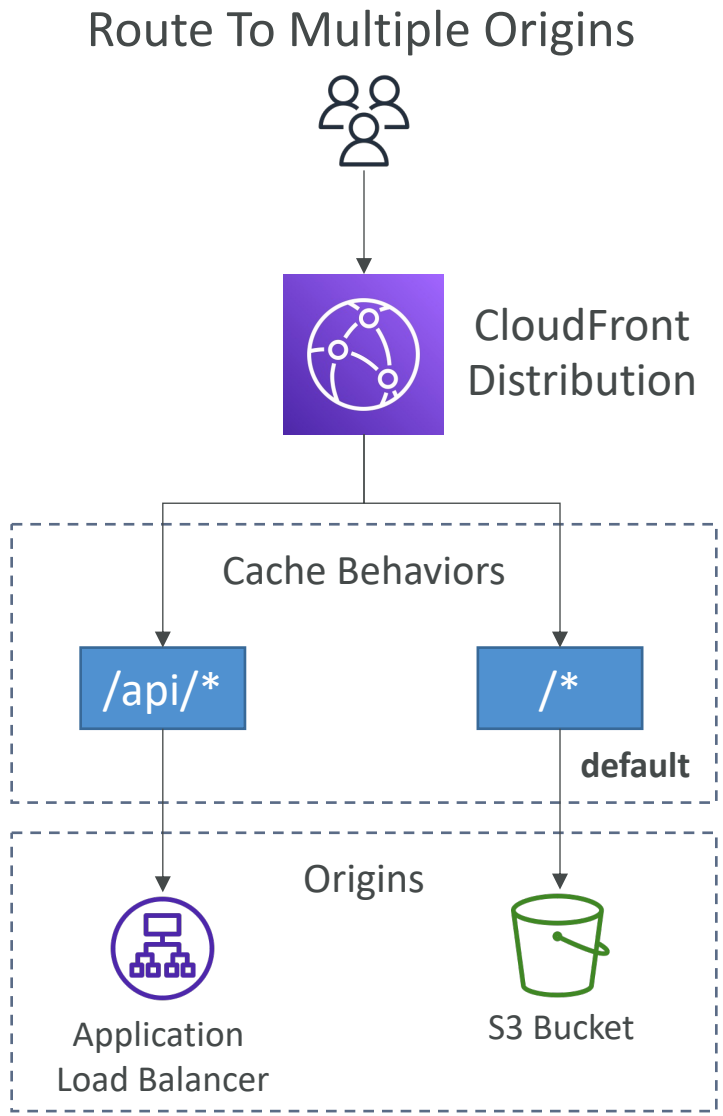
CloudFront – Cache Invalidations

- In case you update the back-end origin, CloudFront doesn't know about it and will only get the refreshed content after the TTL has expired
- However, you can force an entire or partial cache refresh (thus bypassing the TTL) by performing a **CloudFront Invalidation**
- You can invalidate all files (*) or a special path (/images/*)

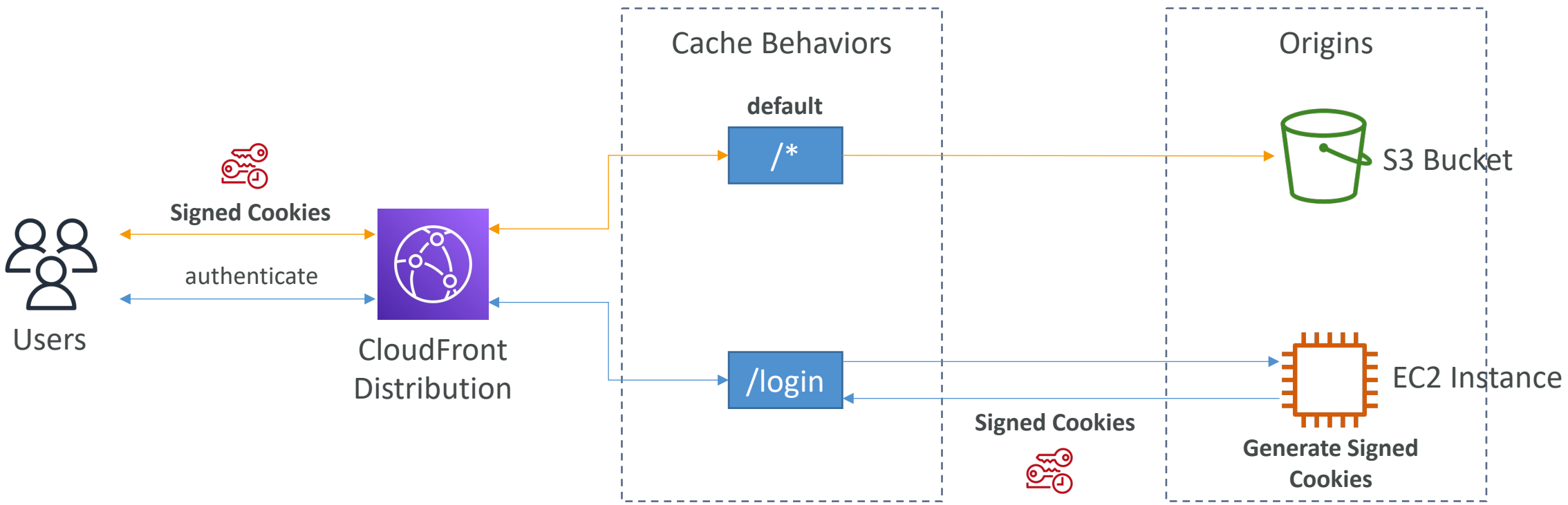


CloudFront – Cache Behaviors

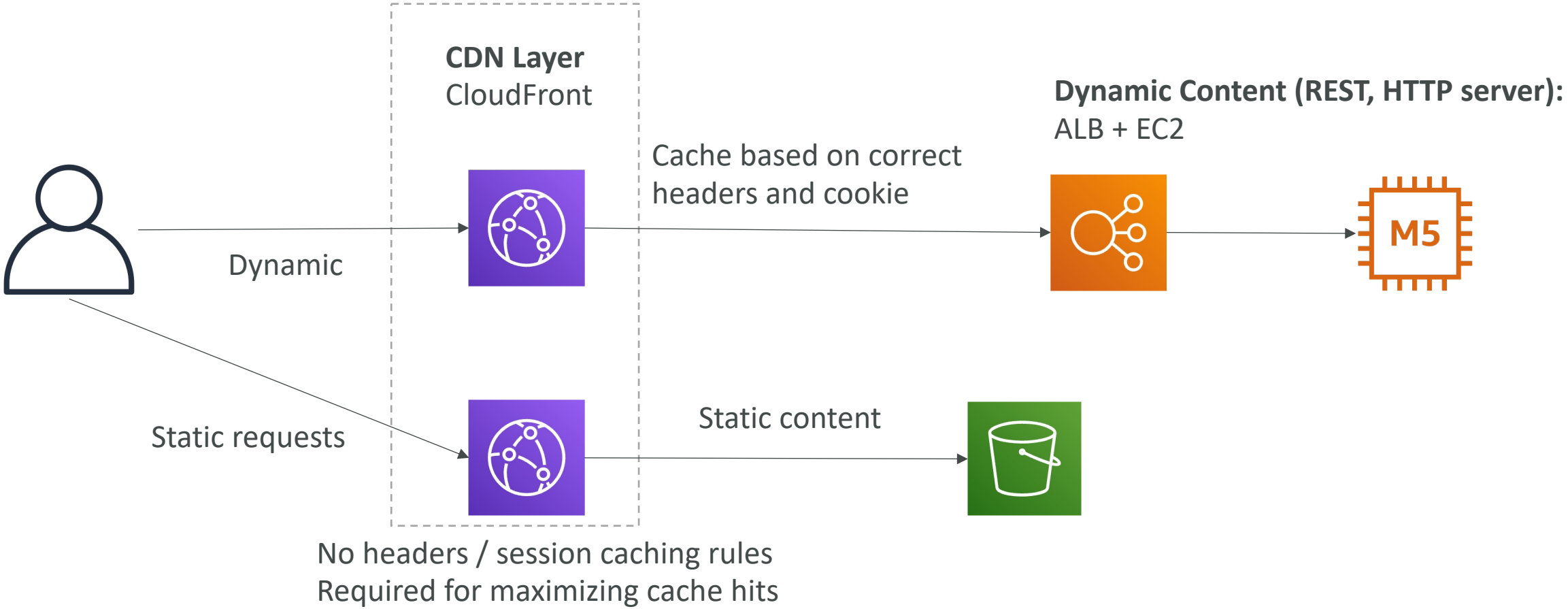
- Configure different settings for a given URL path pattern
- Example: one specific cache behavior to **images/*.jpg** files on your origin web server
- Route to different kind of origins/origin groups based on the content type or path pattern
  - /images/*
  - /api/*
  - /* (default cache behavior)
- When adding additional Cache Behaviors, the Default Cache Behavior is always the last to be processed **and is always /***



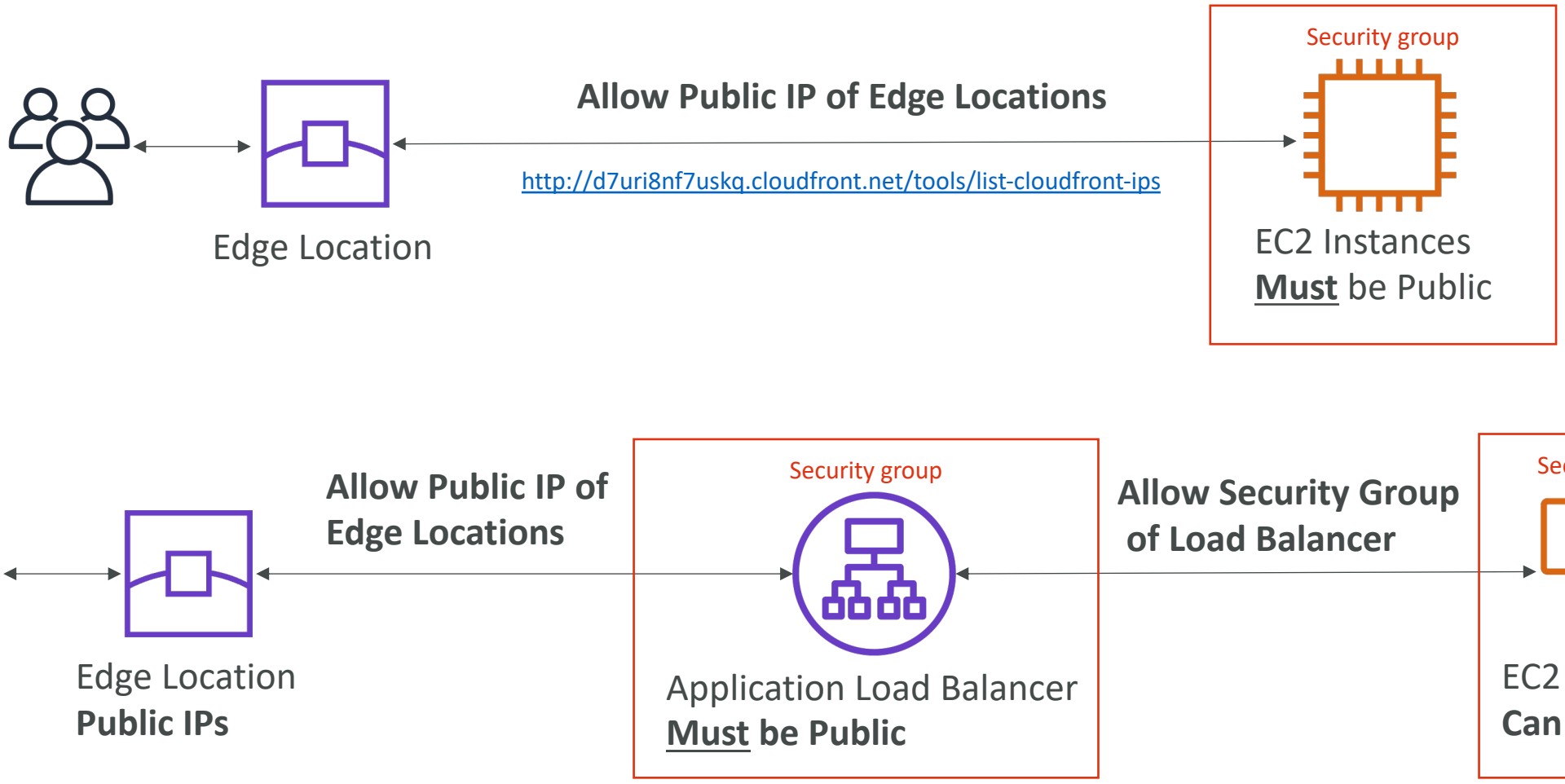
CloudFront – Cache Behaviors – Sign In Page



CloudFront – Maximize cache hits by separating static and dynamic distributions



CloudFront – ALB or EC2 as an origin



CloudFront Geo Restriction

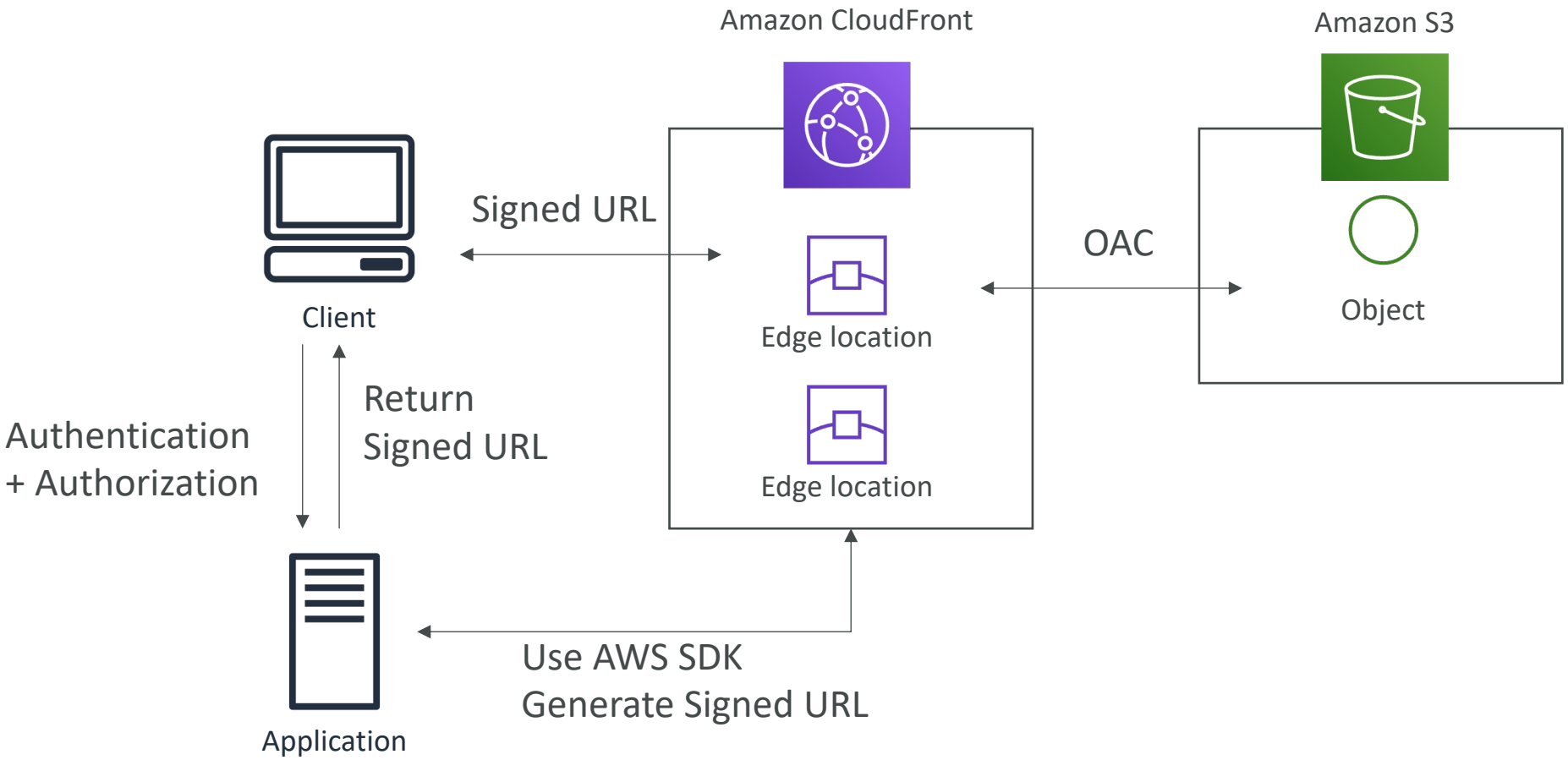
- You can restrict who can access your distribution
  - **Allowlist:** Allow your users to access your content only if they're in one of the countries on a list of approved countries.
  - **Blocklist:** Prevent your users from accessing your content if they're in one of the countries on a list of banned countries.
- The "country" is determined using a 3rd party Geo-IP database
- Use case: Copyright Laws to control access to content

CloudFront Signed URL / Signed Cookies

- You want to distribute paid shared content to premium users over the world
- We can use CloudFront Signed URL / Cookie. We attach a policy with:
  - Includes URL expiration
  - Includes IP ranges to access the data from
  - Trusted signers (which AWS accounts can create signed URLs)
- How long should the URL be valid for?
  - Shared content (movie, music): make it short (a few minutes)
  - Private content (private to the user): you can make it last for years

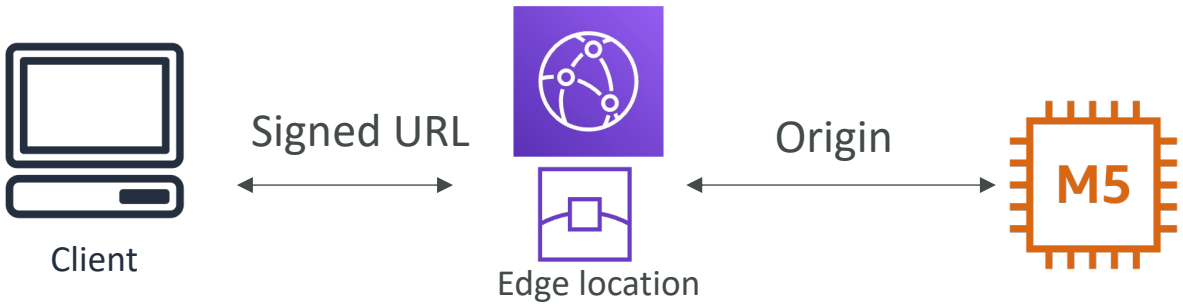
- Signed URL = access to individual files (one signed URL per file)
- Signed Cookies = access to multiple files (one signed cookie for many files)

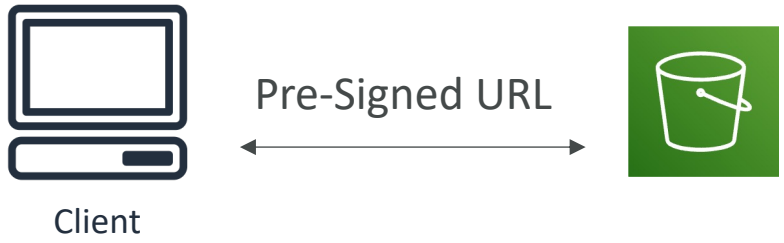
CloudFront Signed URL Diagram



CloudFront Signed URL vs S3 Pre-Signed URL

- CloudFront Signed URL:
  - Allow access to a path, no matter the origin
  - Account wide key-pair, only the root can manage it
  - Can filter by IP, path, date, expiration
  - Can leverage caching features
- S3 Pre-Signed URL:
  - Issue a request as the person who pre-signed the URL
  - Uses the IAM key of the signing IAM principal
  - Limited lifetime





CloudFront Signed URL Process

- Two types of signers:
  - Either a trusted key group (recommended)
    - Can leverage APIs to create and rotate keys (and IAM for API security)
  - An AWS Account that contains a CloudFront Key Pair
    - Need to manage keys using **the root account and the AWS console**
    - Not recommended because you shouldn't use the root account for this
- In your CloudFront distribution, create one or more **trusted key groups**
- You generate your own public / private key
  - The private key is used by your applications (e.g. EC2) to sign URLs
  - The public key (uploaded) is used by CloudFront to verify URLs

CloudFront - Pricing

- CloudFront Edge locations are all around the world
- The cost of data out per edge location varies

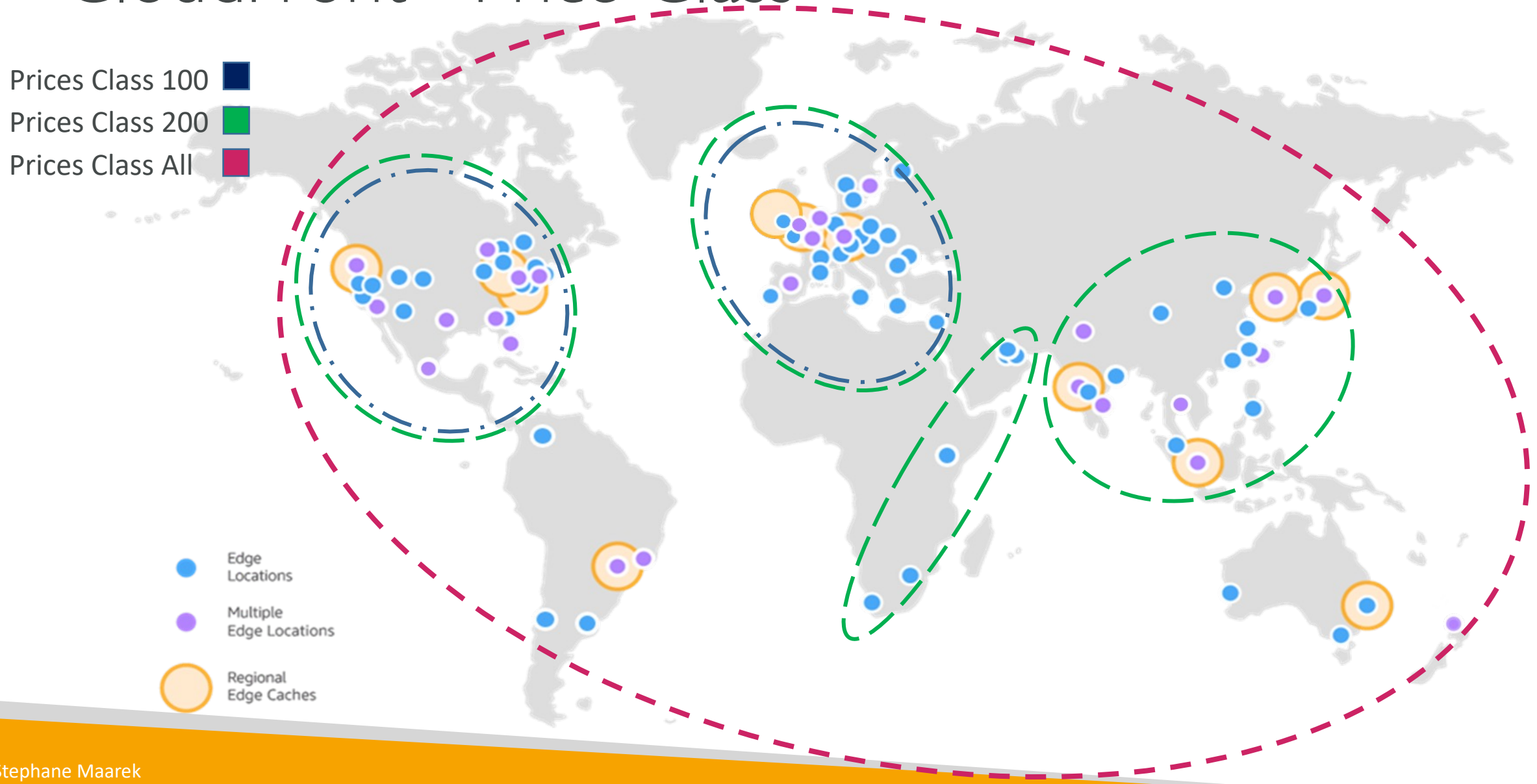
Per Month	United States, Mexico, & Canada	Europe & Israel	South Africa, Kenya, & Middle East	South America	Japan	Australia & New Zealand	Hong Kong, Philippines, Singapore, South Korea, Taiwan, & Thailand	India
First 10TB	$0.085	$0.085	$0.110	$0.110	$0.114	$0.114	$0.140	$0.170
Next 40TB	$0.080	$0.080	$0.105	$0.105	$0.089	$0.098	$0.135	$0.130
Next 100TB	$0.060	$0.060	$0.090	$0.090	$0.086	$0.094	$0.120	$0.110
Next 350TB	$0.040	$0.040	$0.080	$0.080	$0.084	$0.092	$0.100	$0.100
Next 524TB	$0.030	$0.030	$0.060	$0.060	$0.080	$0.090	$0.080	$0.100
Next 4PB	$0.025	$0.025	$0.050	$0.050	$0.070	$0.085	$0.070	$0.100
Over 5PB	$0.020	$0.020	$0.040	$0.040	$0.060	$0.080	$0.060	$0.100

CloudFront – Price Classes

- You can reduce the number of edge locations for **cost reduction**
- Three price classes:
  1. Price Class All: all regions – best performance
  2. Price Class 200: most regions, but excludes the most expensive regions
  3. Price Class 100: only the least expensive regions

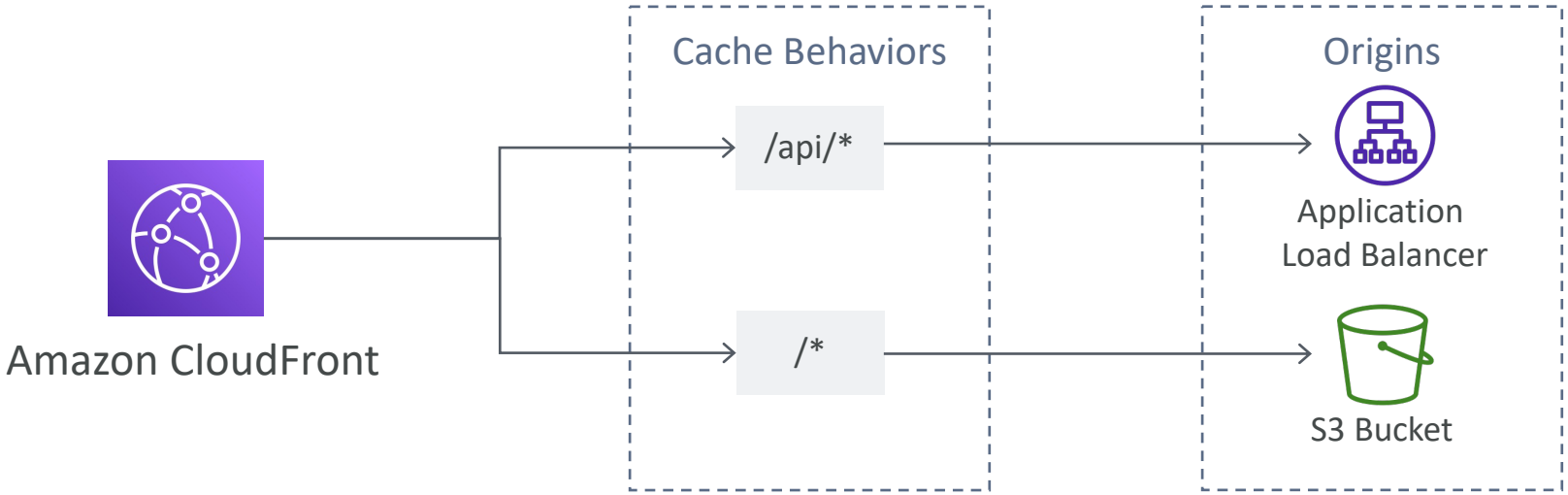
Edge Locations Included Within	**United States, Mexico, & Canada**	**Europe & Israel**	**South Africa, Kenya, & Middle East**	**South America**	**Japan**	**Australia & New Zealand**	**Hong Kong, Philippines, Singapore, South Korea, Taiwan, & Thailand**	**India**
Price Class All	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Price Class 200	Yes	Yes	Yes	x	Yes	x	Yes	Yes
Price Class 100	Yes	Yes	x	x	x	x	x	x

CloudFront - Price Class



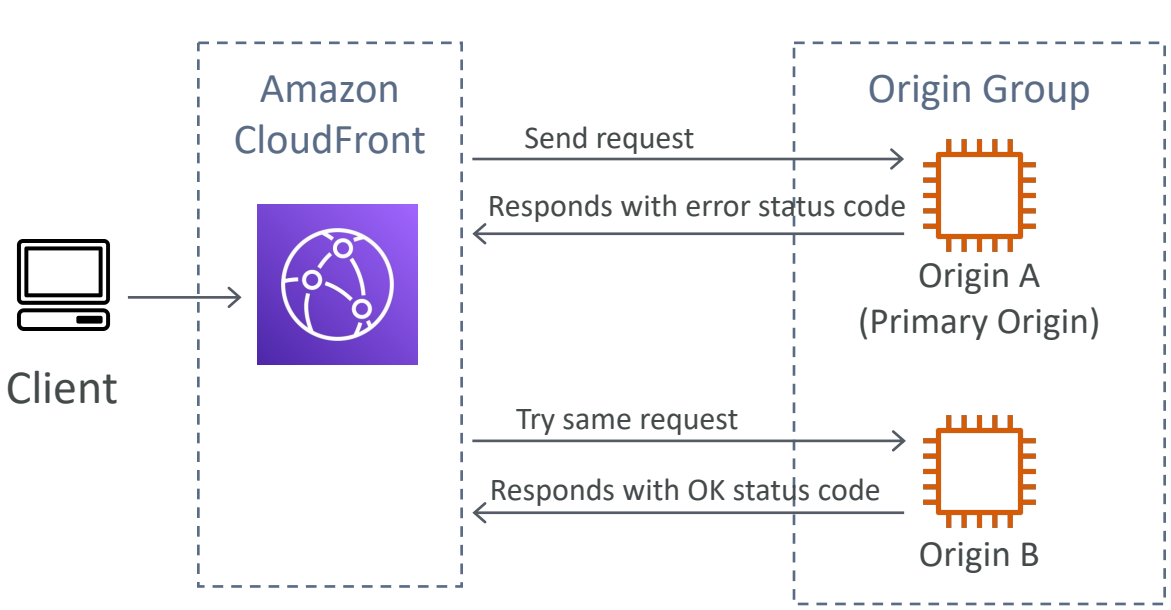
CloudFront – Multiple Origin

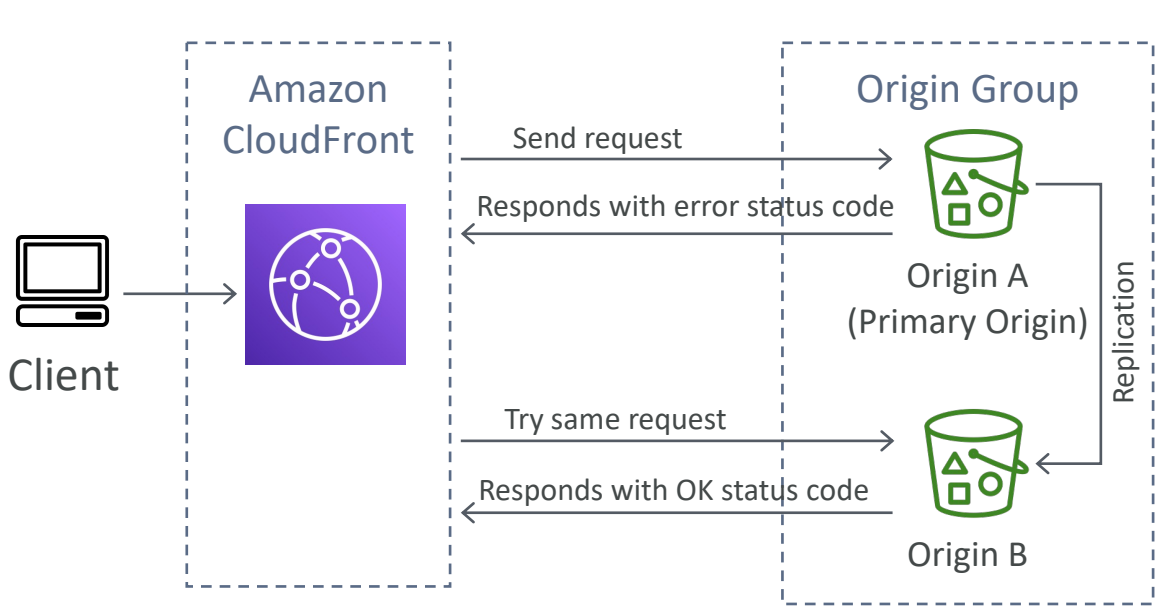
- To route to different kind of origins based on the content type
- Based on path pattern:
  - /images/*
  - /api/*
  - /*



CloudFront – Origin Groups

- To increase high-availability and do failover
- Origin Group: one primary and one secondary origin
- If the primary origin fails, the second one is used

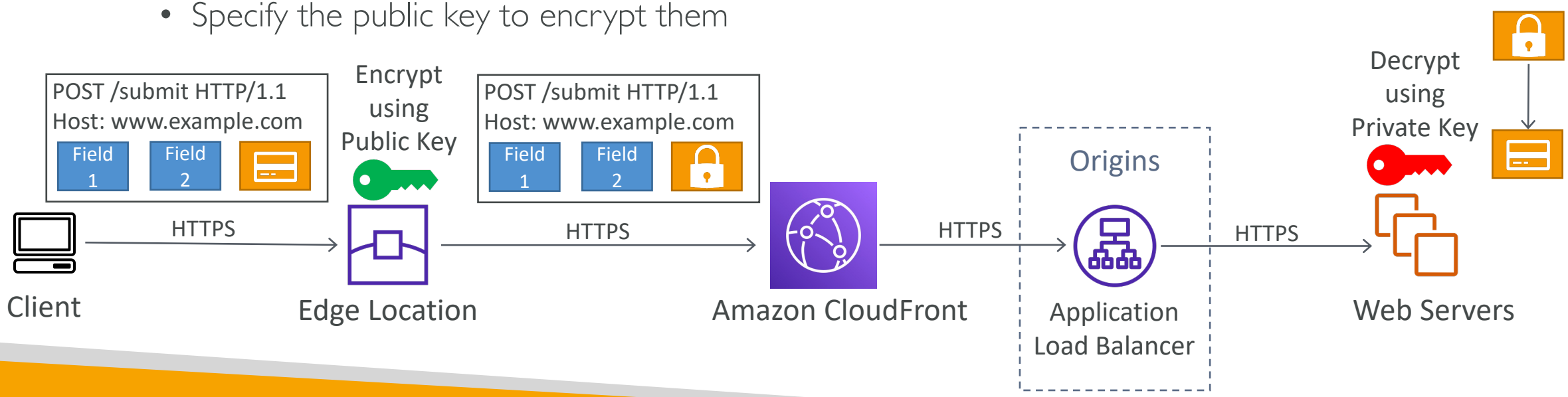




S3 + CloudFront – Region-level High Availability

CloudFront – Field Level Encryption

- Protect user sensitive information through application stack
- Adds an additional layer of security along with HTTPS
- Sensitive information encrypted at the edge close to user
- Uses asymmetric encryption
- Usage:
  - Specify set of fields in POST requests that you want to be encrypted (up to 10 fields)
  - Specify the public key to encrypt them



CloudFront – Real Time Logs

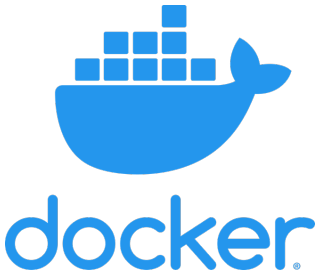
- Get real-time requests received by CloudFront sent to Kinesis Data Streams
- Monitor, analyze, and take actions based on content delivery performance
- Allows you to choose:
  - **Sampling Rate** – percentage of requests for which you want to receive
  - Specific fields and specific Cache Behaviors (path patterns)

Real-time Processing

Near Real-time Processing

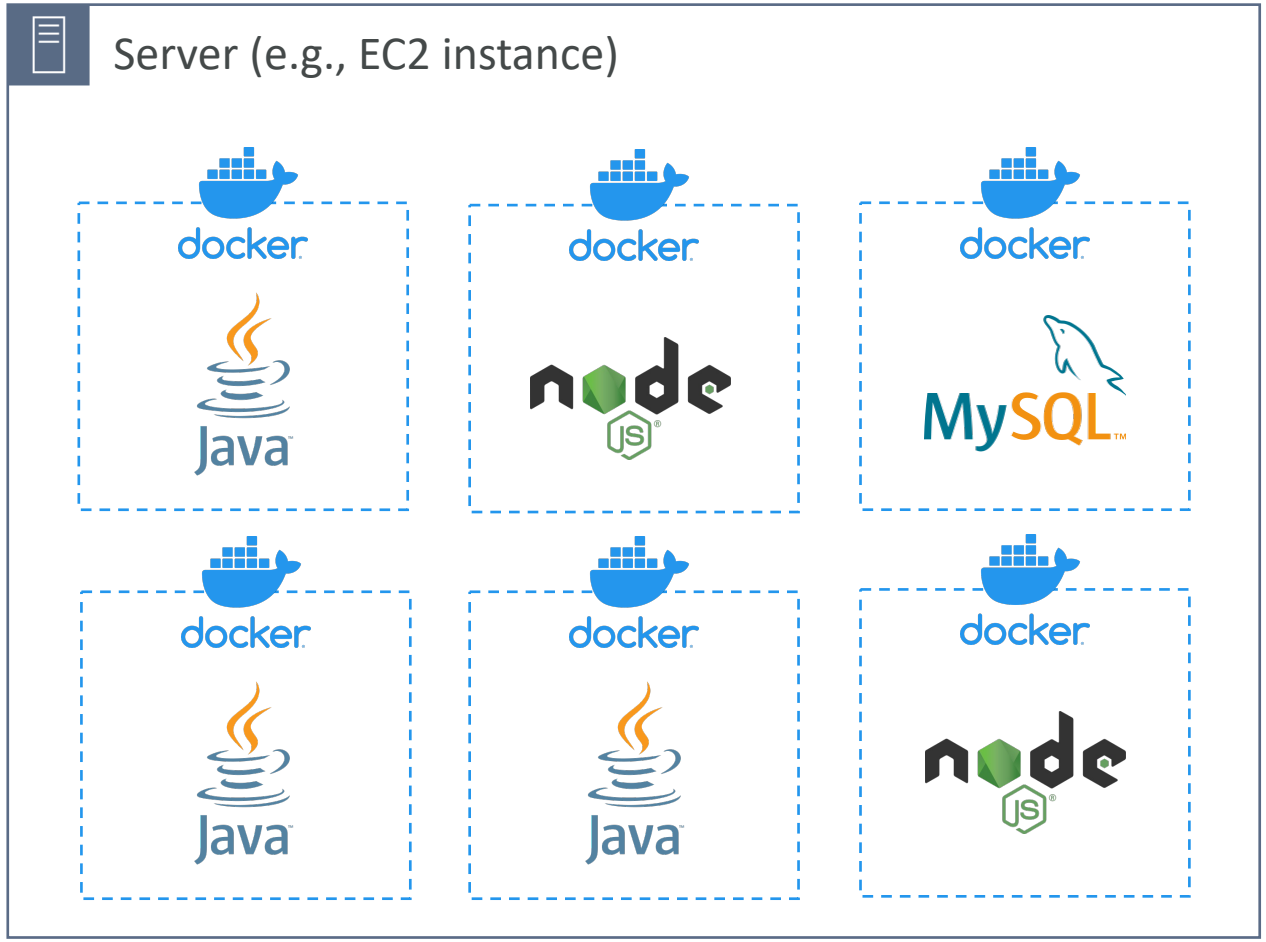
Containers on AWS

What is Docker?



- Docker is a software development platform to deploy apps
- Apps are packaged in **containers** that can be run on any OS
- Apps run the same, regardless of where they're run
  - Any machine
  - No compatibility issues
  - Predictable behavior
  - Less work
  - Easier to maintain and deploy
  - Works with any language, any OS, any technology
- Use cases: microservices architecture, lift-and-shift apps from on-premises to the AWS cloud, …

Docker on an OS



Where are Docker images stored?

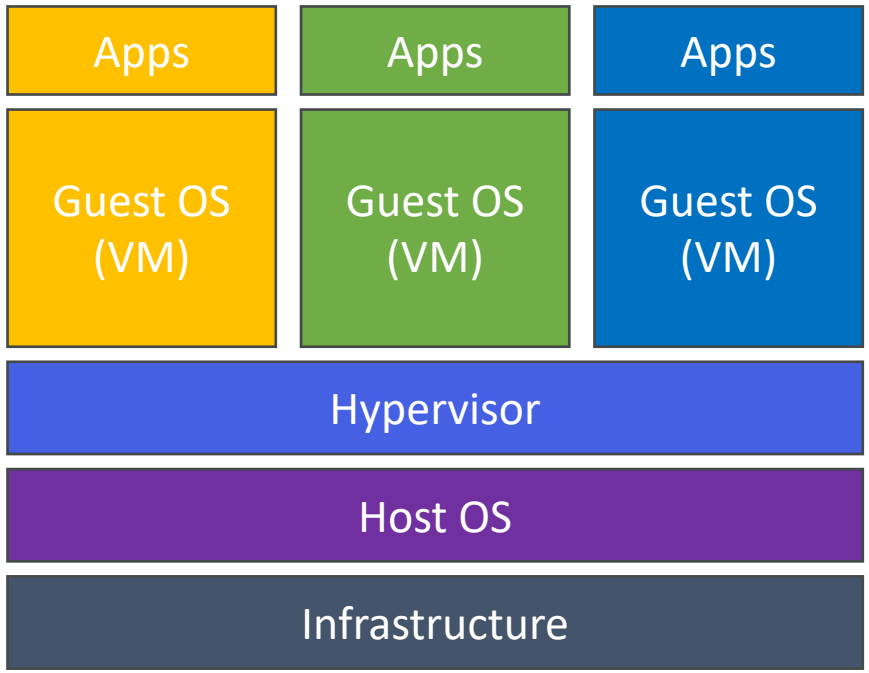
- Docker images are stored in Docker Repositories

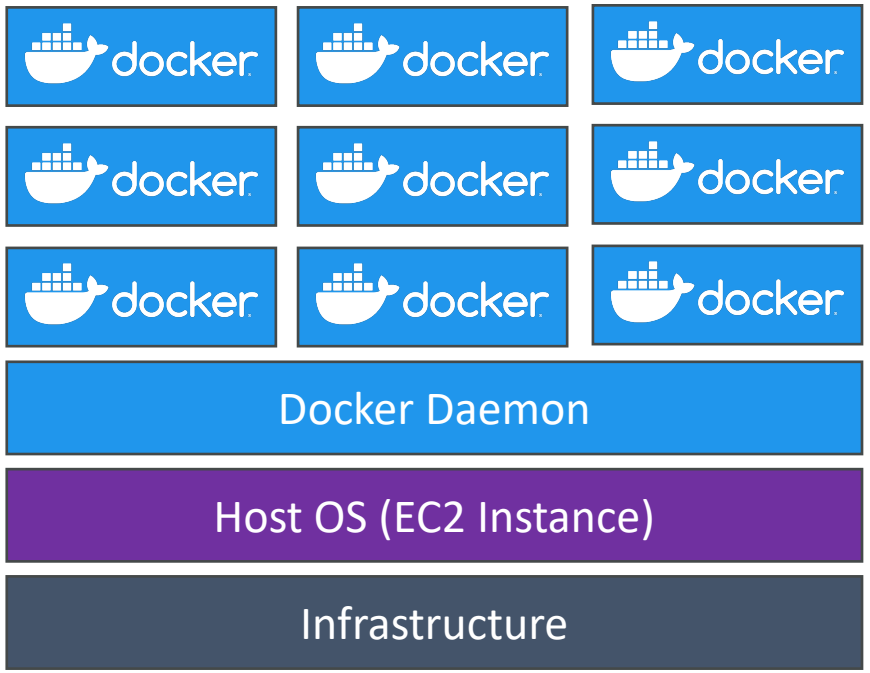
- Docker Hub (https://hub.docker.com)
  - **Public** repository
  - Find base images for many technologies or OS (e.g., Ubuntu, MySQL, …)

- Amazon ECR (Amazon Elastic Container Registry)
  - **Private** repository
  - **Public** repository (**Amazon ECR Public Gallery** https://gallery.ecr.aws)

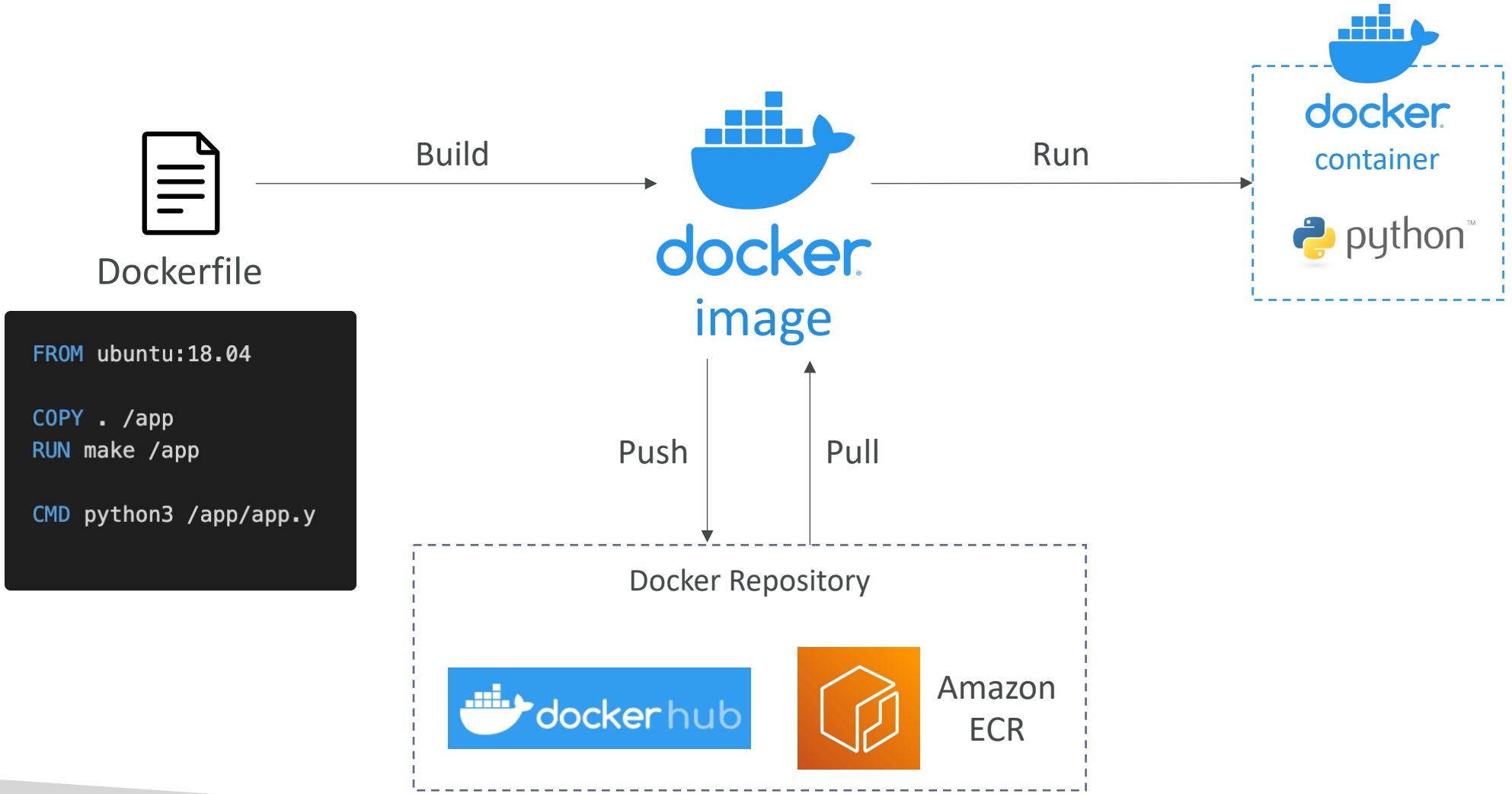
Docker vs. Virtual Machines

- Docker is "sort of" a virtualization technology, but not exactly
- Resources are shared with the host => many containers on one server





Getting Started with Docker



Docker Containers Management on AWS

- Amazon Elastic Container Service (Amazon ECS)
  - Amazon's own container platform

Amazon ECS

- Amazon Elastic Kubernetes Service (Amazon EKS)
  - Amazon's managed Kubernetes (open source)

Amazon EKS

- AWS Fargate
  - Amazon's own Serverless container platform
  - Works with ECS and with EKS

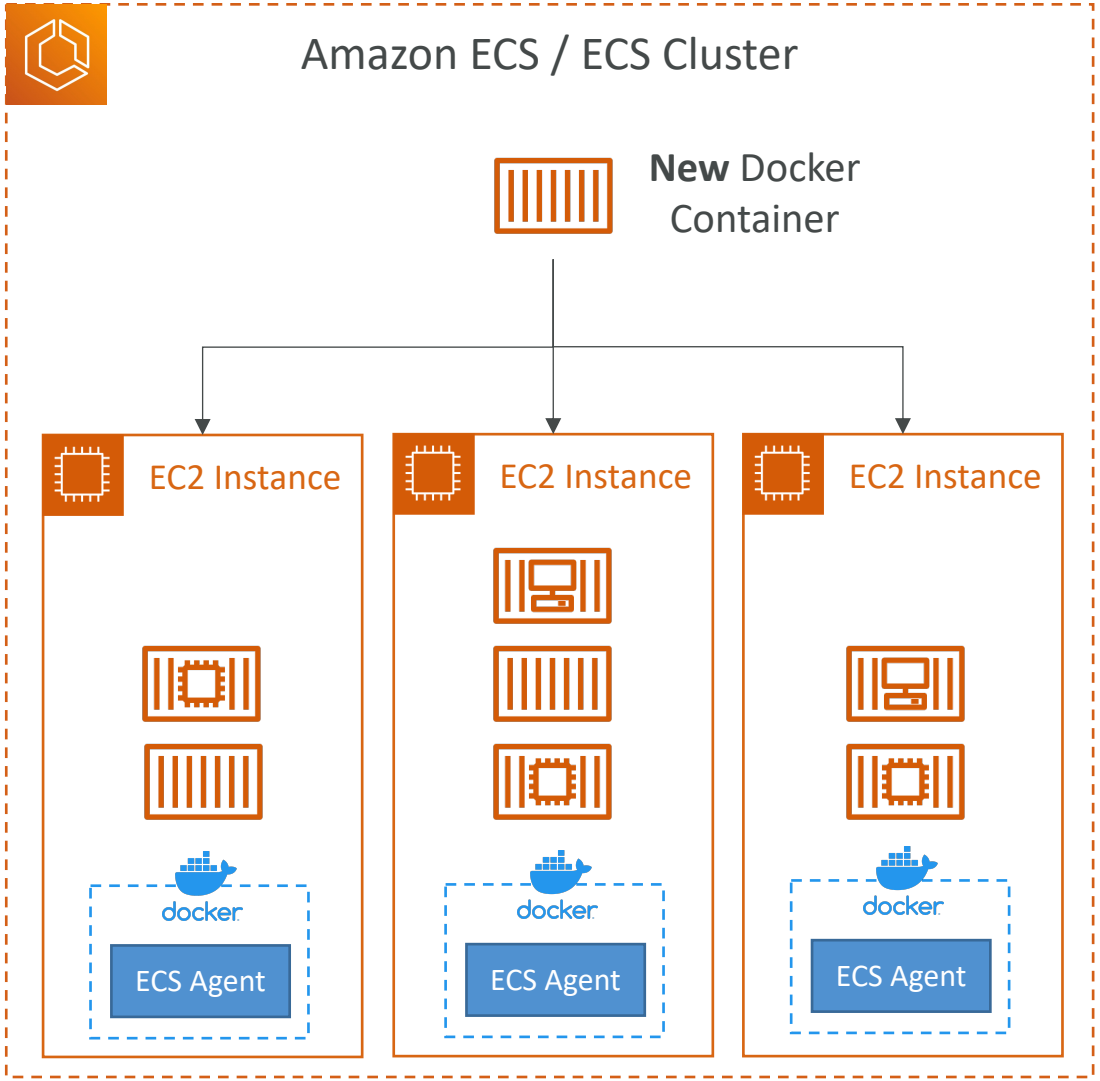
AWS Fargate

- Amazon ECR:
  - Store container images

Amazon ECR

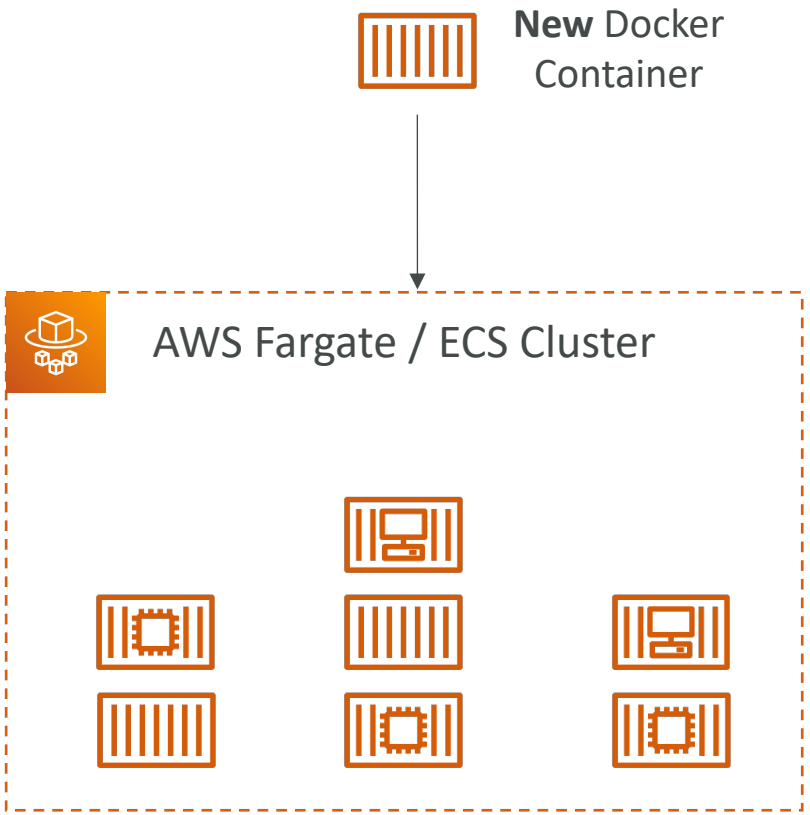
Amazon ECS - EC2 Launch Type

- ECS = Elastic Container Service
- Launch Docker containers on AWS = Launch **ECS Tasks** on ECS Clusters
- **EC2 Launch Type: you must provision & maintain the infrastructure (the EC2 instances)**
- Each EC2 Instance must run the ECS Agent to register in the ECS Cluster
- AWS takes care of starting / stopping containers



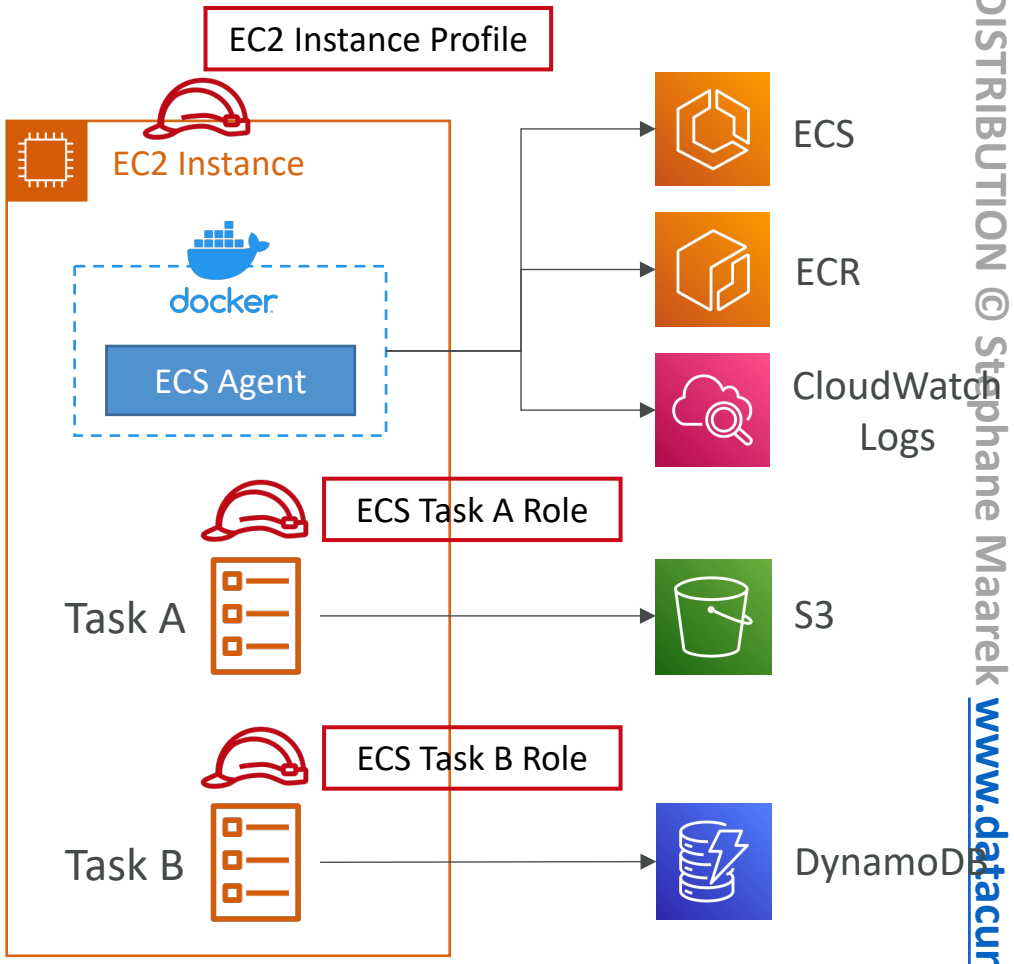
Amazon ECS – Fargate Launch Type

- Launch Docker containers on AWS
- **You do not provision the infrastructure (no EC2 instances to manage)**
- **It's all Serverless!**
- You just create task definitions
- AWS just runs ECS Tasks for you based on the CPU / RAM you need
- To scale, just increase the number of tasks. Simple - no more EC2 instances



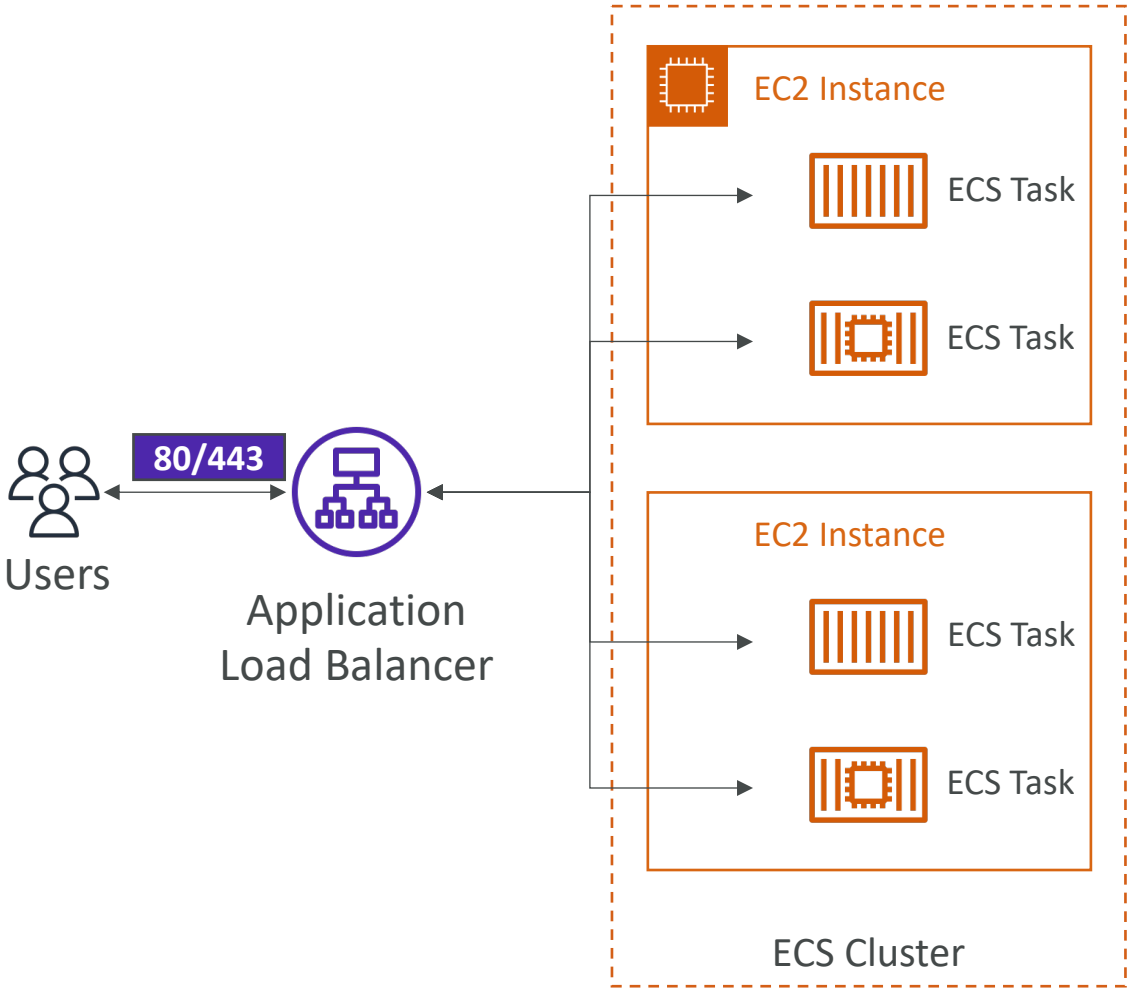
Amazon ECS – IAM Roles for ECS

- EC2 Instance Profile (EC2 Launch Type only):
  - Used by the ECS agent
  - Makes API calls to ECS service
  - Send container logs to CloudWatch Logs
  - Pull Docker image from ECR
  - Reference sensitive data in Secrets Manager or SSM Parameter Store
- ECS Task Role:
  - Allows each task to have a specific role
  - Use different roles for the different ECS Services you run
  - Task Role is defined in the task definition



Amazon ECS – Load Balancer Integrations

- Application Load Balancer supported and works for most use cases
- Network Load Balancer recommended only for high throughput / high performance use cases, or to pair it with AWS Private Link
- Classic Load Balancer supported but not recommended (no advanced features – no Fargate)

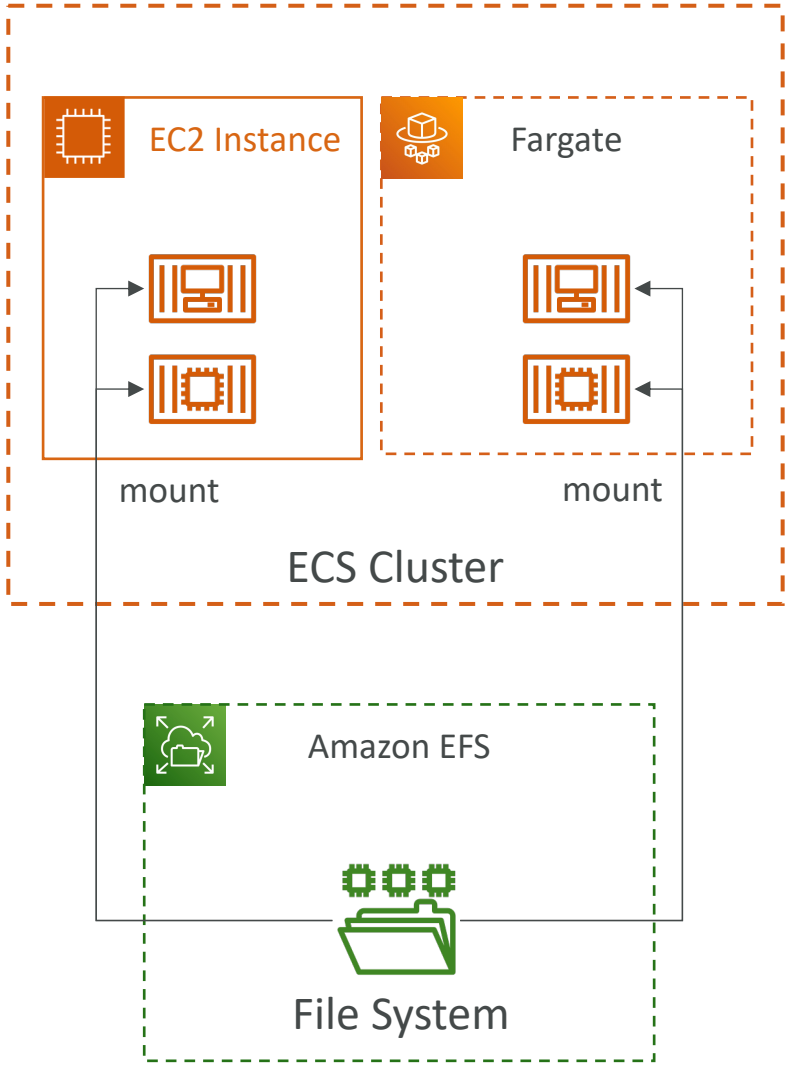


Amazon ECS – Data Volumes (EFS)

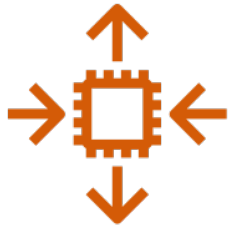
- Mount EFS file systems onto ECS tasks
- Works for both **EC2** and **Fargate** launch types
- Tasks running in any AZ will share the same data in the EFS file system
- **Fargate + EFS = Serverless**

- Use cases: persistent multi-AZ shared storage for your containers

- Note:
  - Amazon S3 cannot be mounted as a file system



ECS Service Auto Scaling



- Automatically increase/decrease the desired number of ECS tasks

- Amazon ECS Auto Scaling uses **AWS Application Auto Scaling**
  - ECS Service Average CPU Utilization
  - ECS Service Average Memory Utilization - Scale on RAM
  - ALB Request Count Per Target – metric coming from the ALB

- **Target Tracking** – scale based on target value for a specific CloudWatch metric
- **Step Scaling** – scale based on a specified CloudWatch Alarm
- **Scheduled Scaling** – scale based on a specified date/time (predictable changes)

- ECS Service Auto Scaling (task level) ≠ EC2 Auto Scaling (EC2 instance level)
- Fargate Auto Scaling is much easier to setup (because **Serverless**)

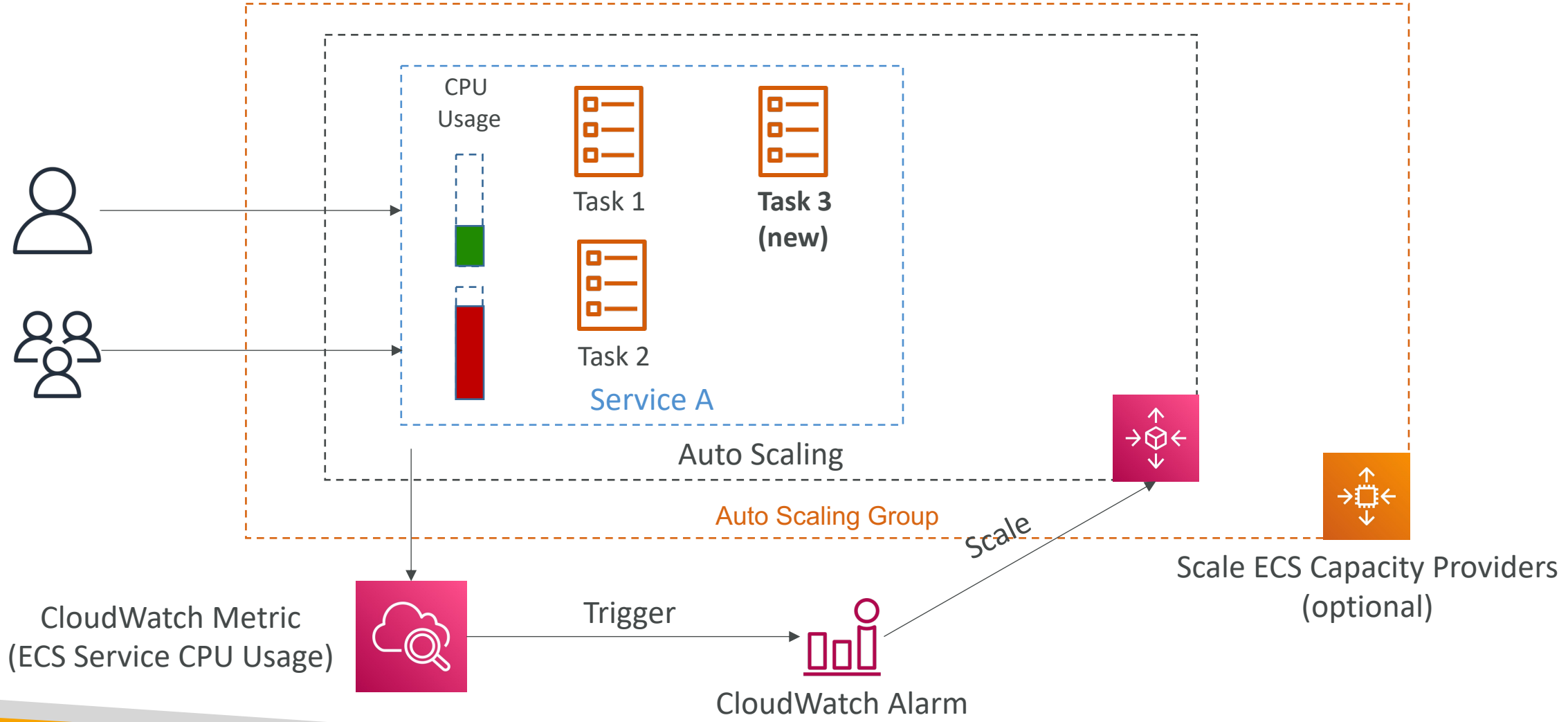
EC2 Launch Type – Auto Scaling EC2 Instances

- Accommodate ECS Service Scaling by adding underlying EC2 Instances

- Auto Scaling Group Scaling
  - Scale your ASG based on CPU Utilization
  - Add EC2 instances over time

- ECS Cluster Capacity Provider
  - Used to automatically provision and scale the infrastructure for your ECS Tasks
  - Capacity Provider paired with an Auto Scaling Group
  - Add EC2 Instances when you're missing capacity (CPU, RAM…)

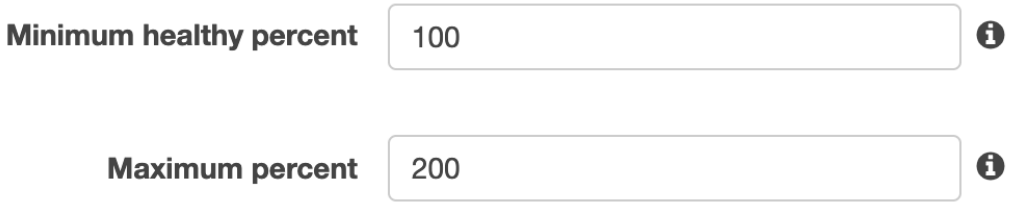
ECS Scaling – Service CPU Usage Example

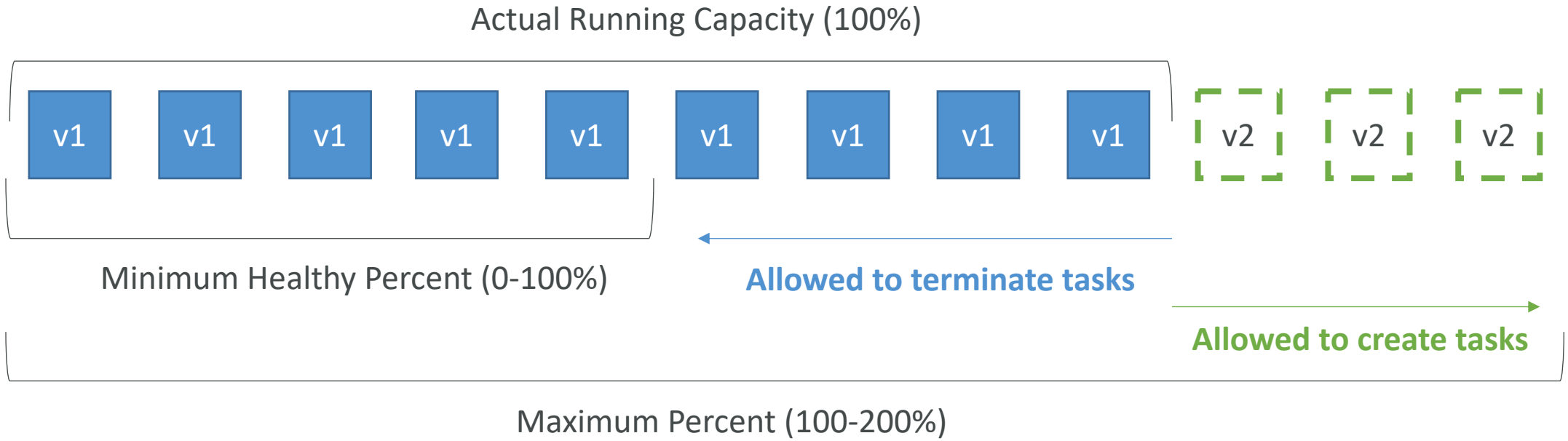


ECS Rolling Updates

- When updating from v1 to v2, we can control how many tasks can be started and stopped, and in which order

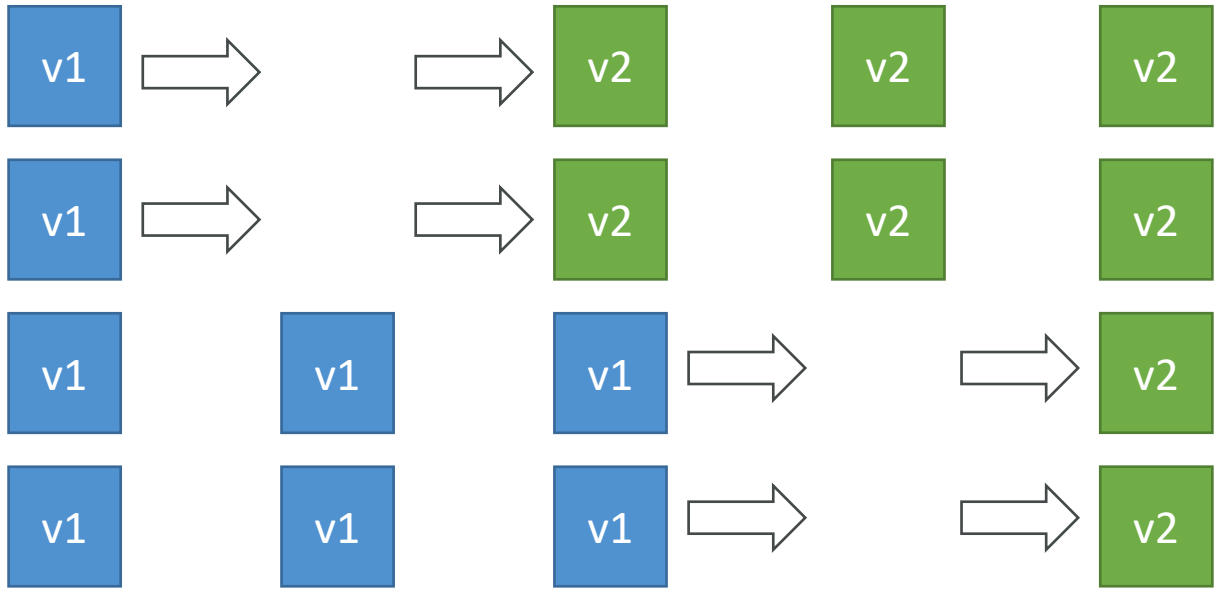
ECS Service update screen





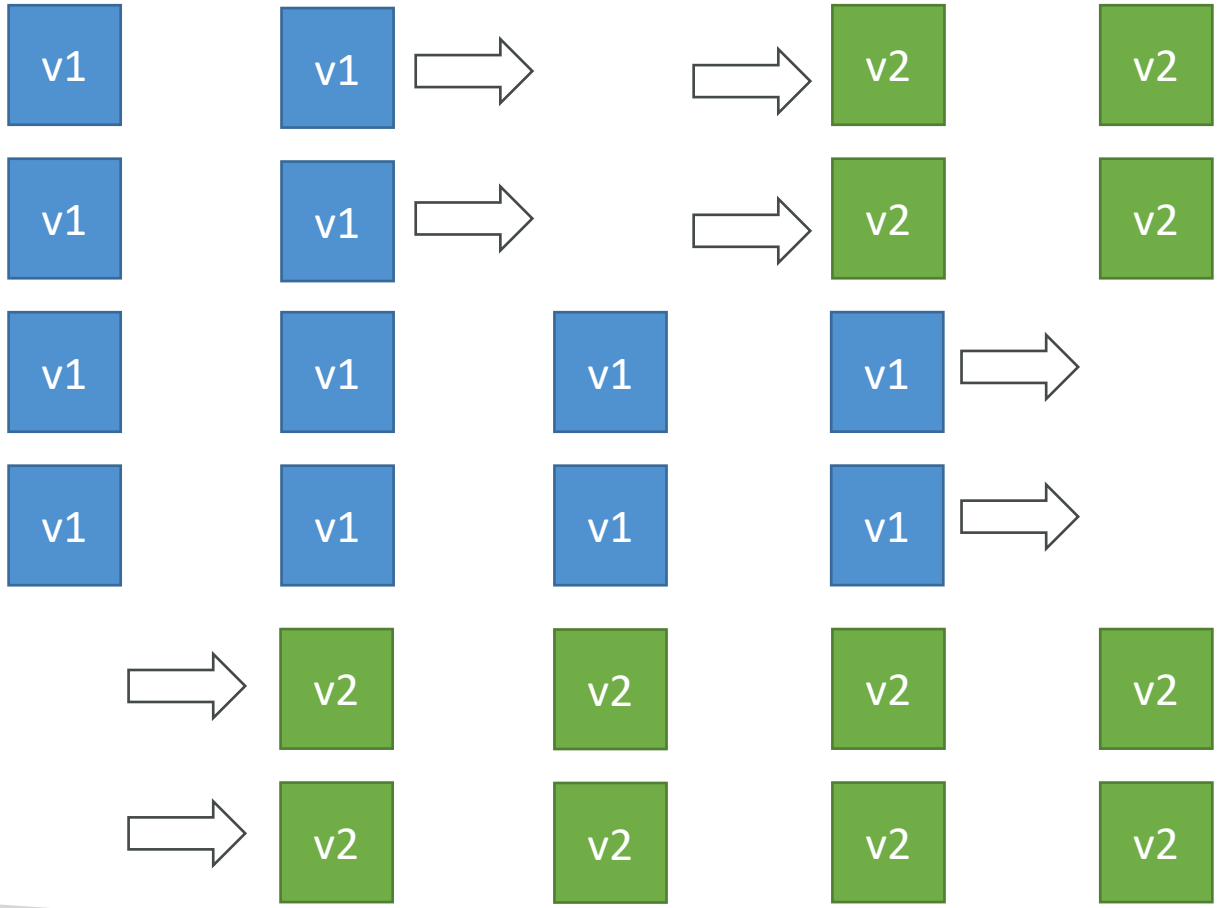
ECS Rolling Update – **Min 50%**, Max 100%

- Starting number of tasks: 4

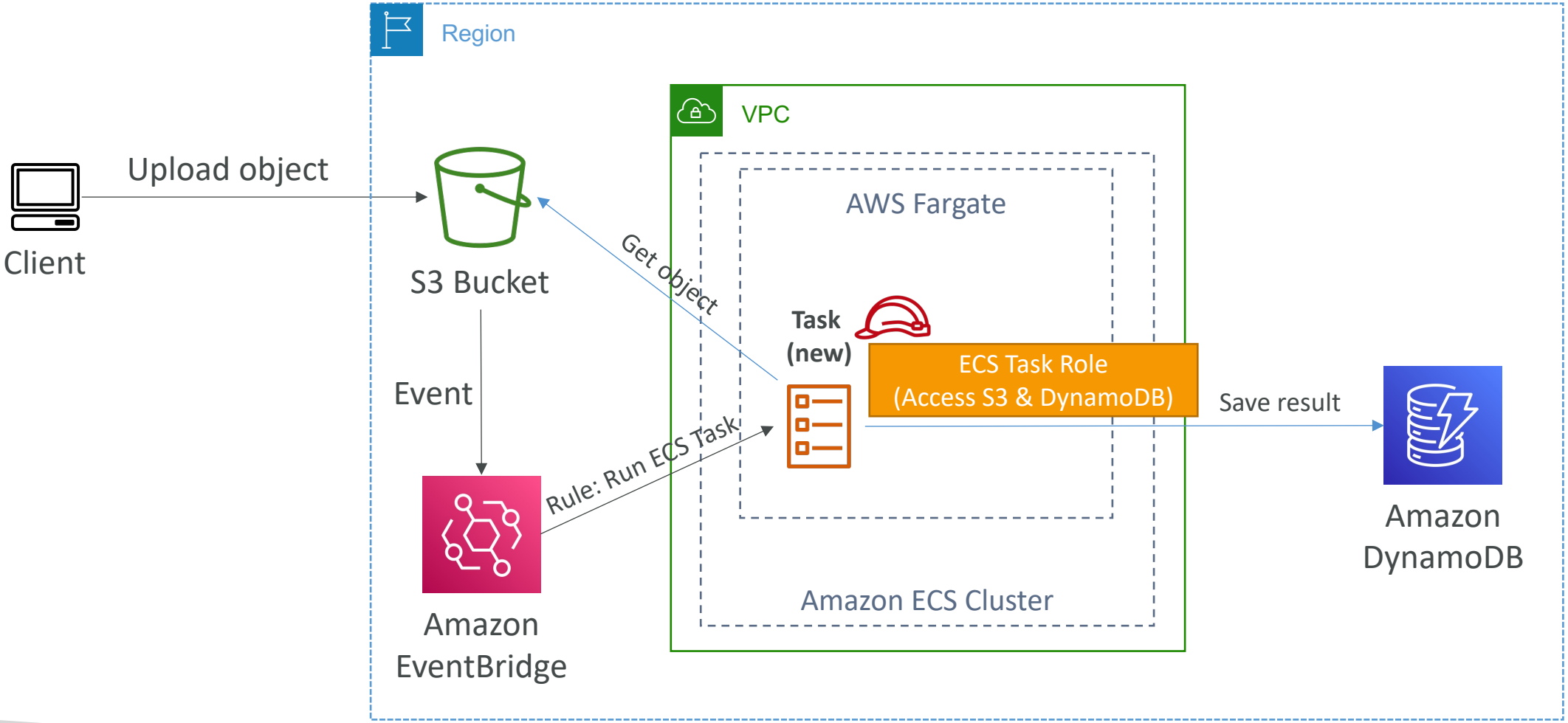


ECS Rolling Update – Min 100%, Max 150%

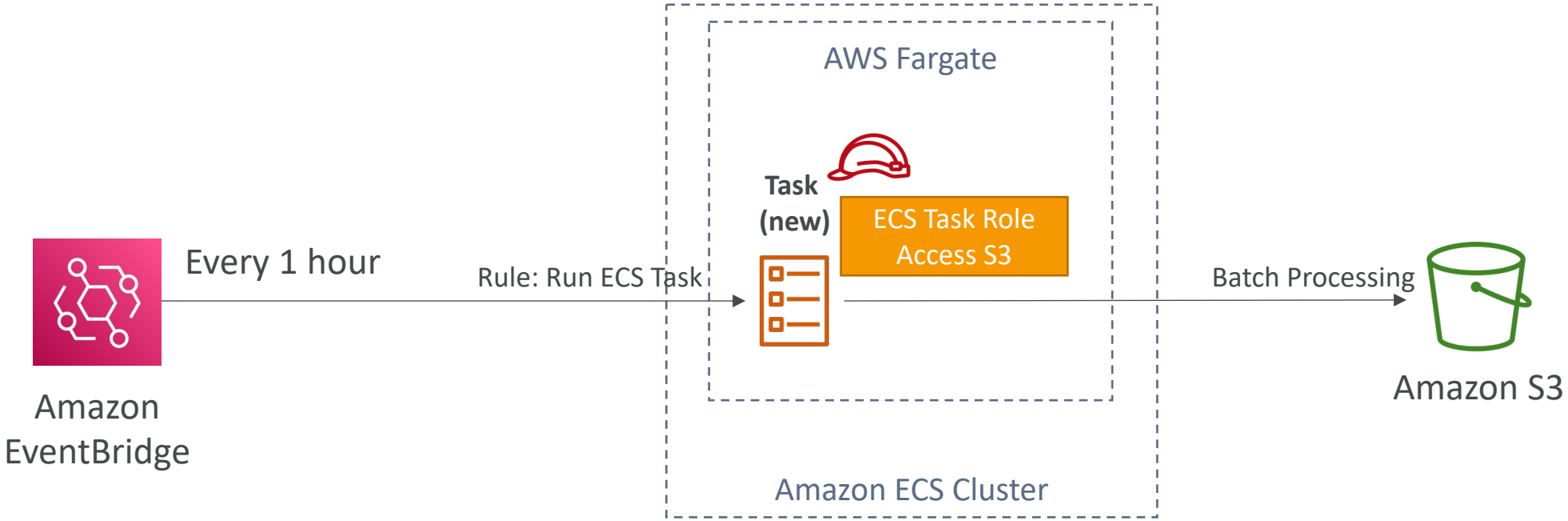
- Starting number of tasks: 4



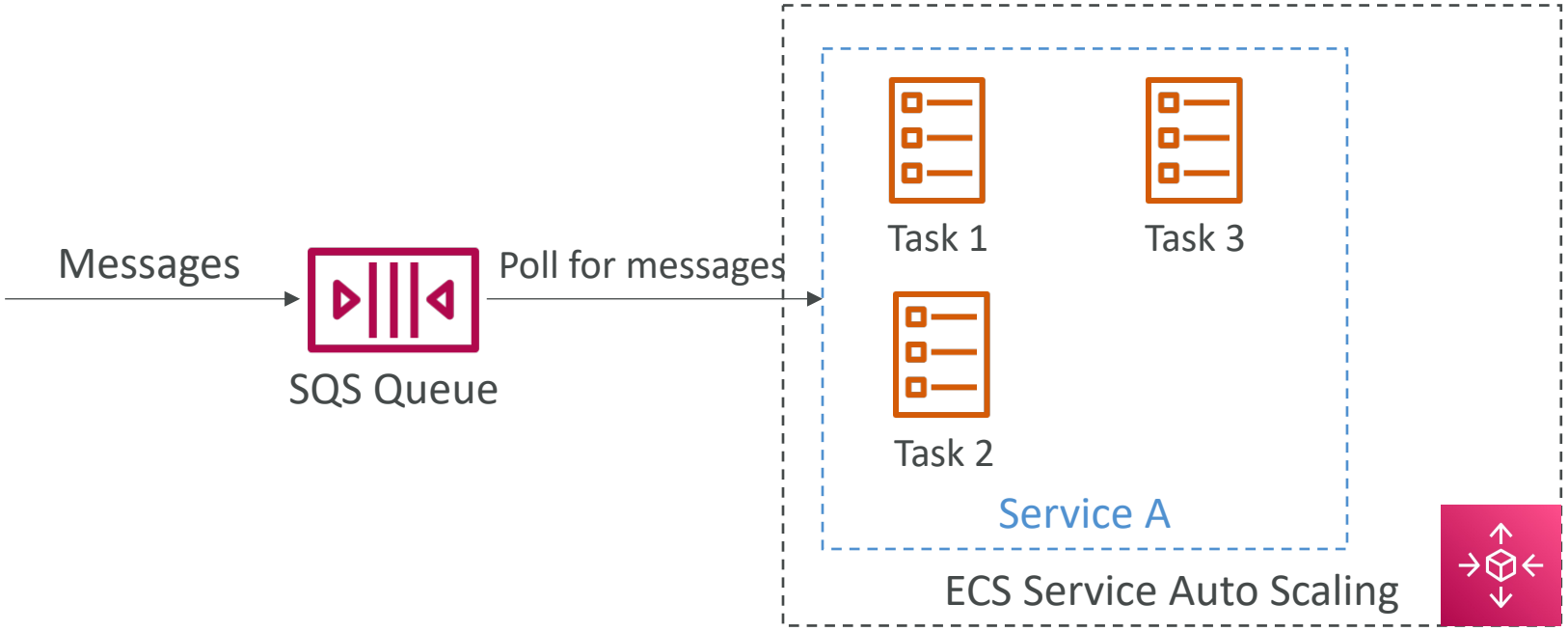
ECS tasks invoked by Event Bridge



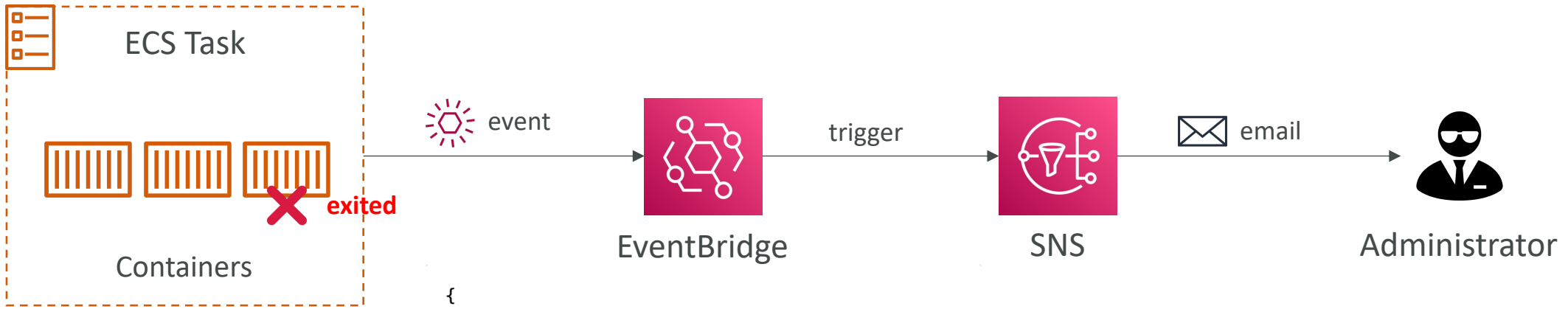
ECS tasks invoked by Event Bridge Schedule



ECS – SQS Queue Example



ECS – Intercept Stopped Tasks using EventBridge

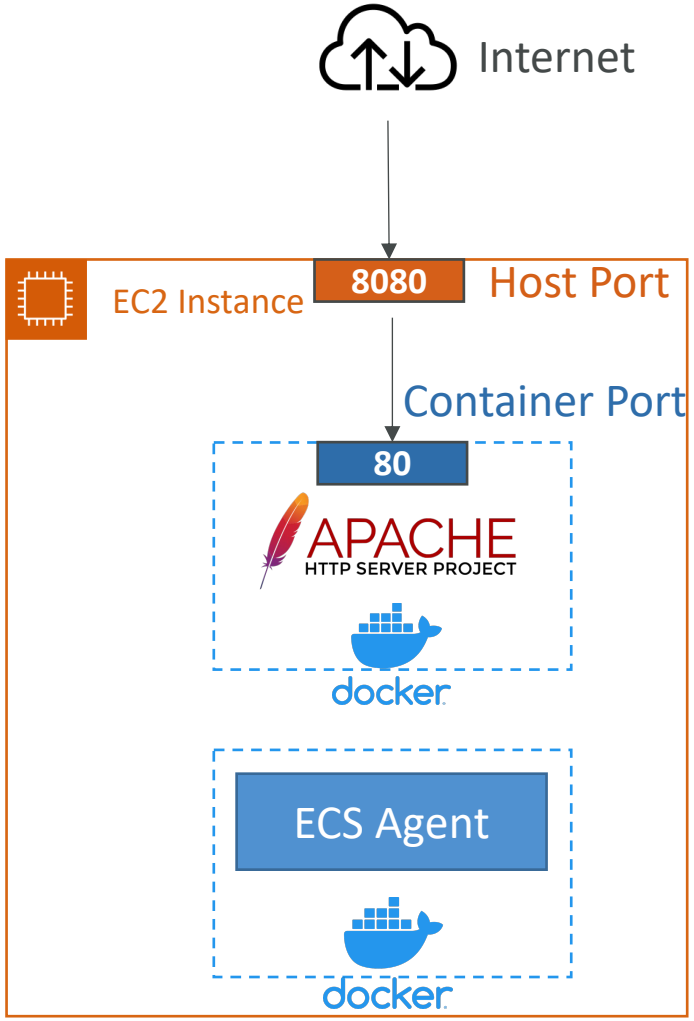



```
{
    "source": [
        "aws.ecs"
    ],
    "detail-type": [
        "ECS Task State Change"
    ],
    "detail": {
        "lastStatus": [
            "STOPPED"
        ],
        "stoppedReason": [
            "Essential container in task exited"
        ]
    }
}
```

Event Pattern

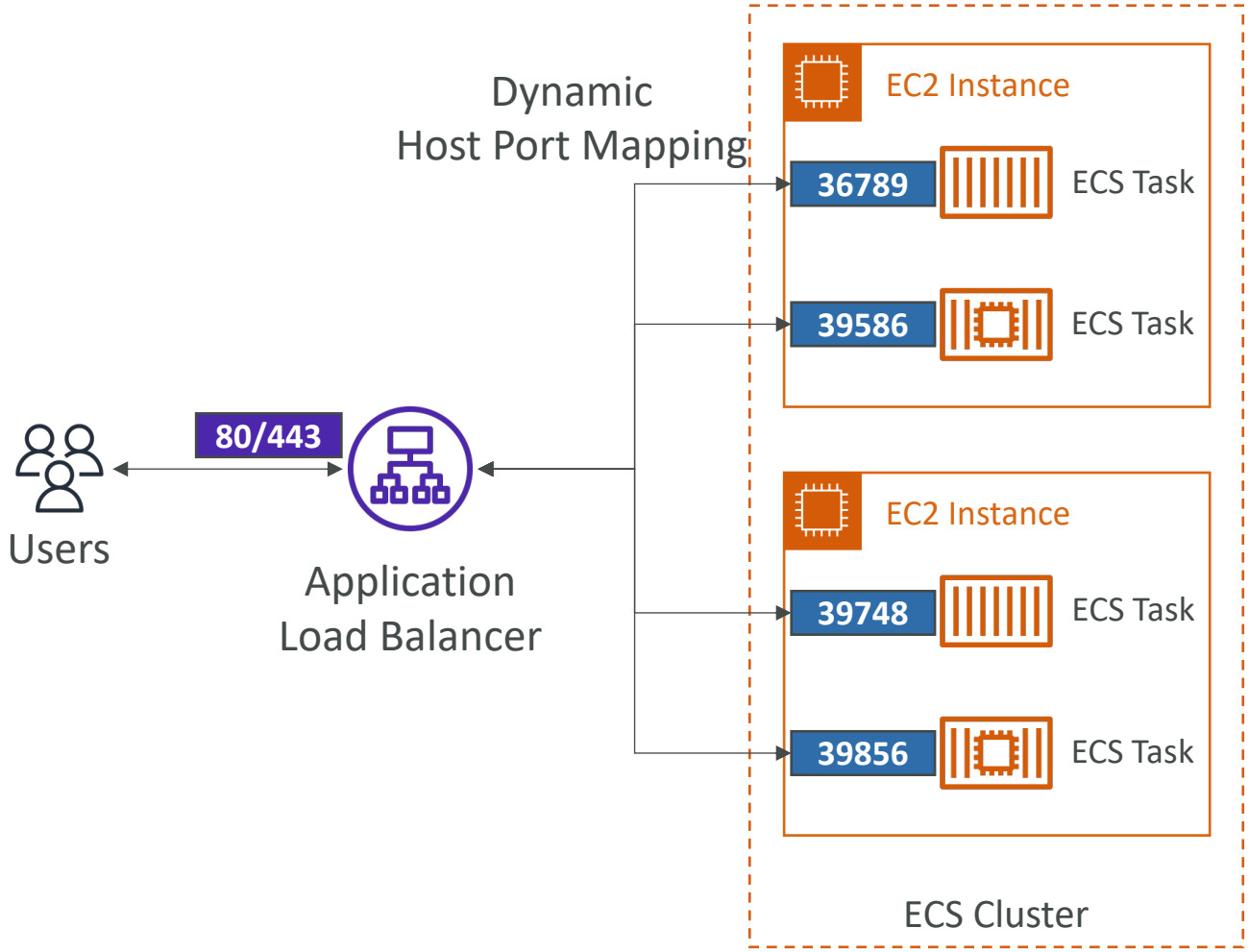
Amazon ECS – Task Definitions

- Task definitions are metadata in **JSON form** to tell ECS how to run a Docker container
- It contains crucial information, such as:
  - Image Name
  - Port Binding for Container and Host
  - Memory and CPU required
  - Environment variables
  - Networking information
  - IAM Role
  - Logging configuration (ex CloudWatch)
- Can define up to 10 containers in a Task Definition



Amazon ECS – Load Balancing (EC2 Launch Type)

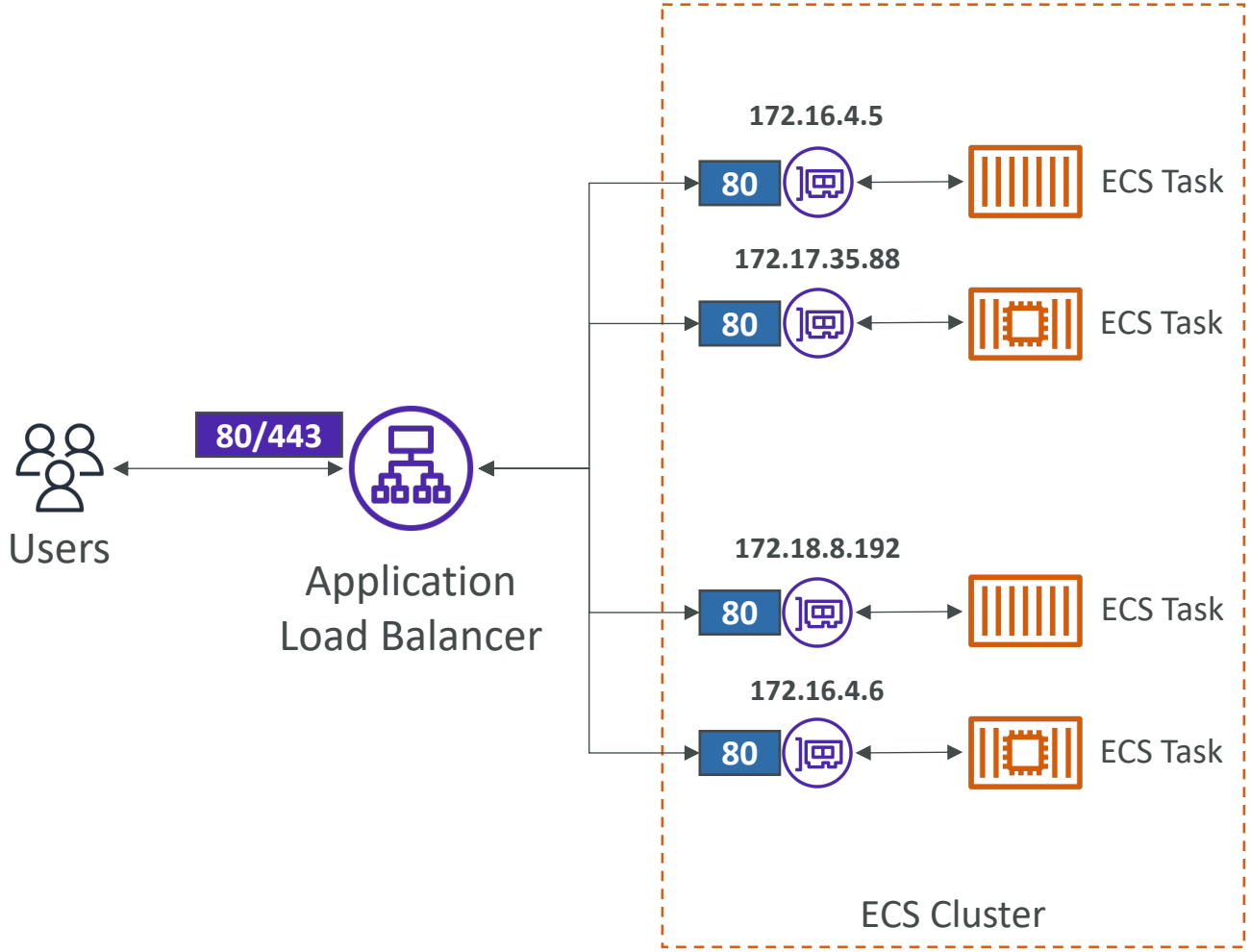
- We get a **Dynamic Host Port Mapping** if you define only the container port in the task definition
- The ALB finds the right port on your EC2 Instances
- You must allow on the EC2 instance's Security Group any port from the ALB's Security Group



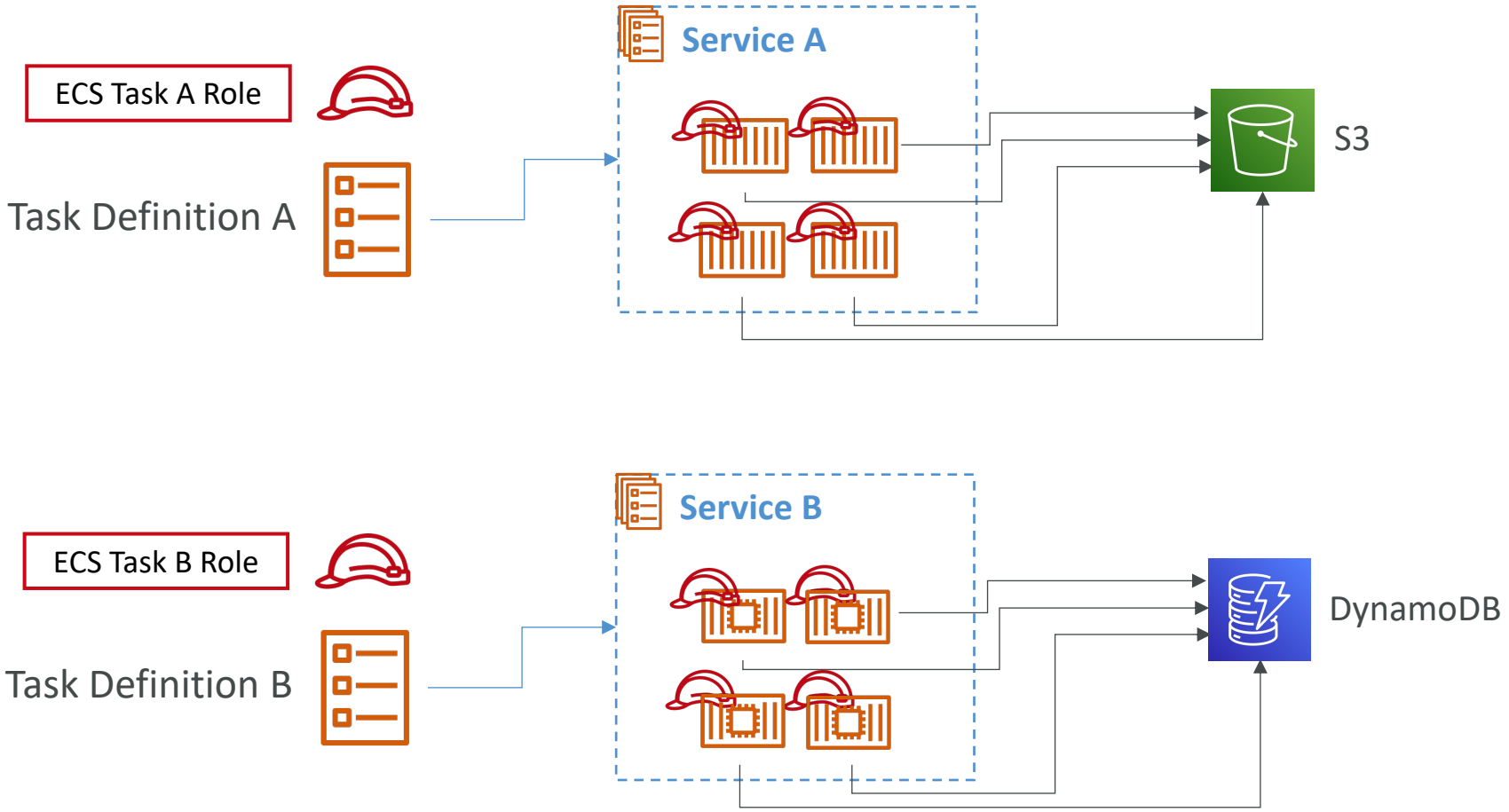
Amazon ECS – Load Balancing (Fargate)

- Each task has a **unique private IP**
- **Only define the container port** (host port is not applicable)

- Example
  - ECS ENI Security Group
    - Allow port 80 from the ALB
  - ALB Security Group
    - Allow port 80/443 from web

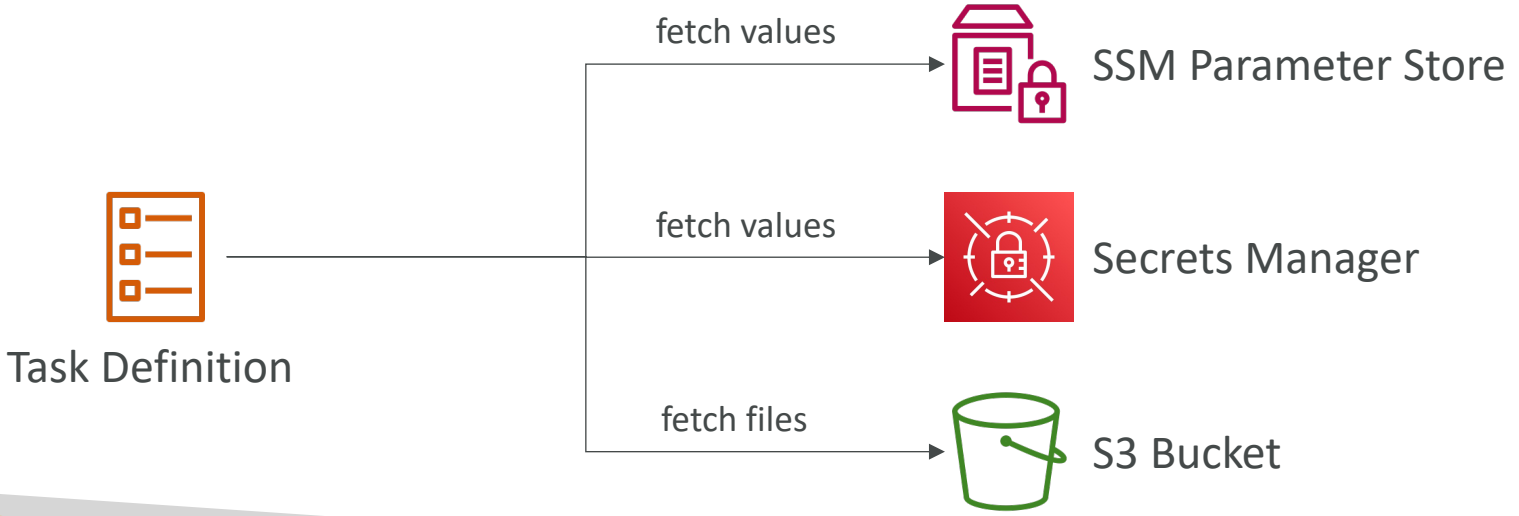


Amazon ECS
One IAM Role per Task Definition



Amazon ECS – Environment Variables

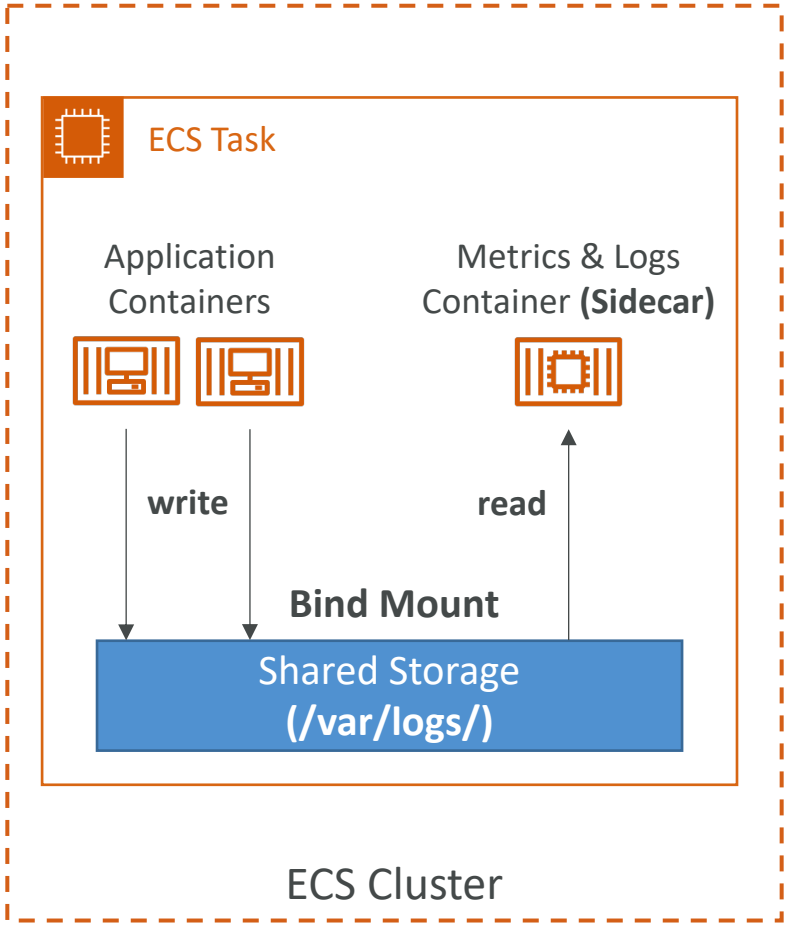
- Environment Variable
  - **Hardcoded** – e.g., URLs
  - **SSM Parameter Store** – sensitive variables (e.g., API keys, shared configs)
  - **Secrets Manager** – sensitive variables (e.g., DB passwords)
- Environment Files (bulk) – Amazon S3



Amazon ECS – Data Volumes (Bind Mounts)

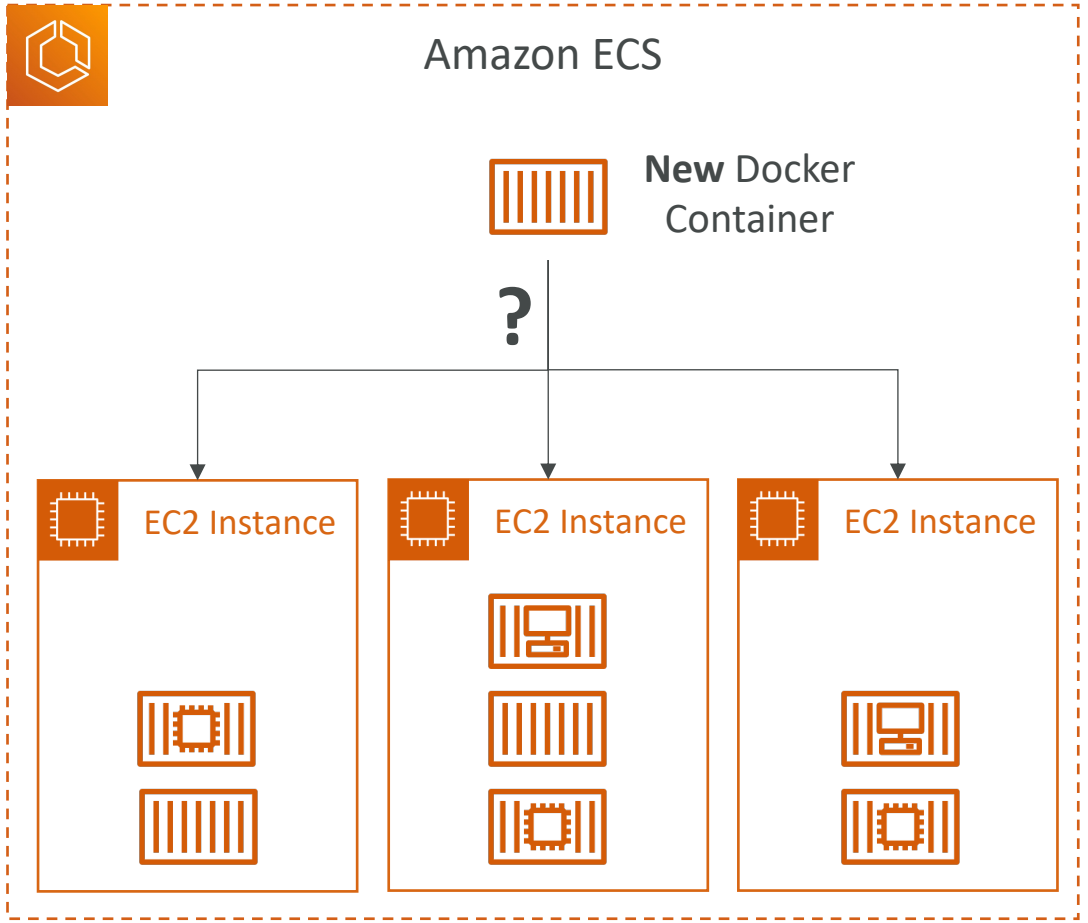
- Share data between multiple containers in the same Task Definition
- Works for both **EC2** and **Fargate** tasks
- **EC2 Tasks** – using EC2 instance storage
  - Data are tied to the lifecycle of the EC2 instance
- **Fargate Tasks** – using ephemeral storage
  - Data are tied to the container(s) using them
  - 20 GiB – 200 GiB (default 20 GiB)

- Use cases:
  - Share ephemeral data between multiple containers
  - "Sidecar" container pattern, where the "sidecar" container used to send metrics/logs to other destinations (separation of conerns)



Amazon ECS – Task Placement

- When an ECS task is started with EC2 Launch Type, ECS must determine where to place it, with the constraints of **CPU** and **memory (RAM)**
- Similarly, when a service scales in, ECS needs to determine which task to terminate
- You can define:
  - Task Placement Strategy
  - Task Placement Constraints
- Note: only for ECS Tasks with EC2 Launch Type **(Fargate not supported)**



Amazon ECS – Task Placement Process

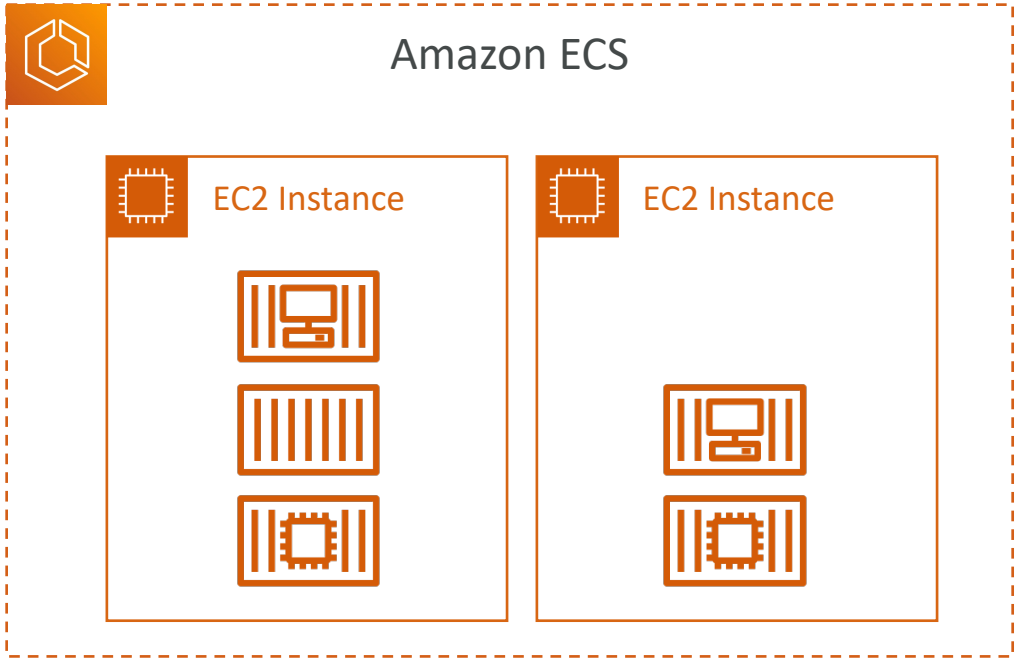
- Task Placement Strategies are a **best effort**

- When Amazon ECS places a task, it uses the following process to select the appropriate EC2 Container instance:
  1. Identify which instances that satisfy the **CPU, memory, and port** requirements
  2. Identify which instances that satisfy the **Task Placement Constraints**
  3. Identify which instances that satisfy the **Task Placement Strategies**
  4. Select the instances

Amazon ECS – Task Placement Strategies

- Binpack
  - Tasks are placed on the least available amount of CPU and Memory
  - Minimizes the number of EC2 instances in use (cost savings)

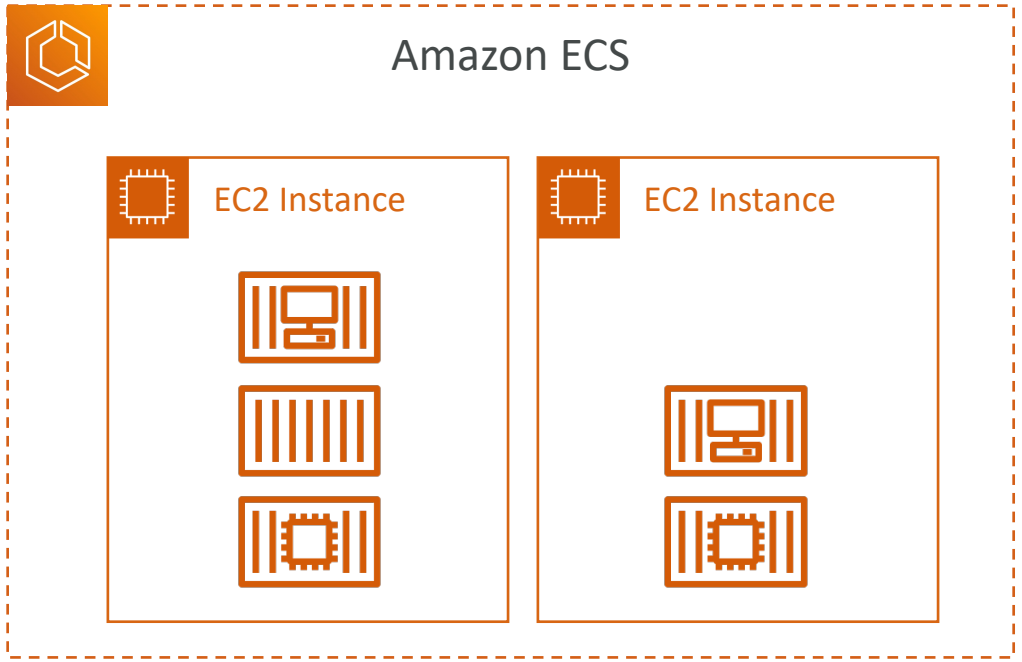
```
"placementStrategy": [
    {
        "type": "binpack",
        "field": "memory"
    }
]
```



Amazon ECS – Task Placement Strategies

- Random
  - Tasks are placed randomly

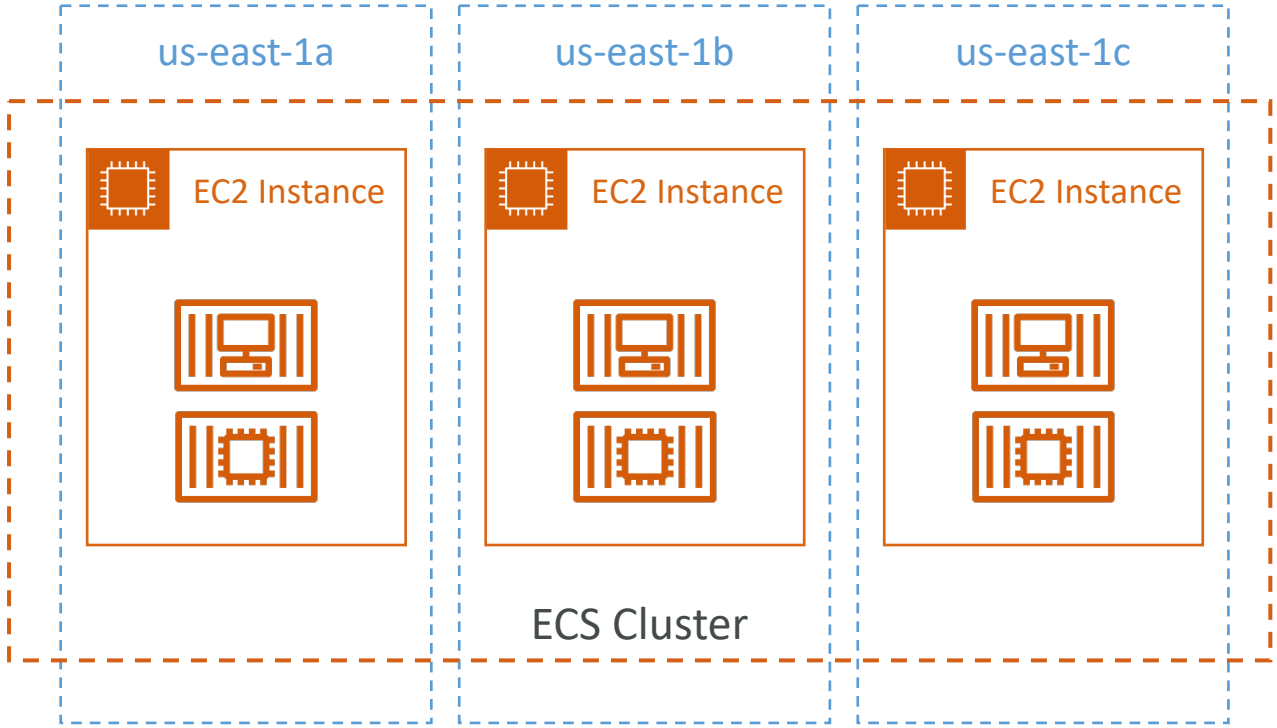
```
"placementStrategy": [
    {
        "type": "random"
    }
]
```



Amazon ECS – Task Placement Strategies

- Spread
  - Tasks are placed evenly based on the specified value
  - Example: **instanceId**, **attribute:ecs.availability-zone**, …

```
"placementStrategy": [
    {
        "type": "spread",
        "field": "attribute:ecs.availability-zone"
    }
]
```



Amazon ECS – Task Placement Strategies

- You can mix them together

```
"placementStrategy": [
    {
        "type": "spread",
        "field": "attribute:ecs.availability-zone"
    },
    {
        "type": "spread",
        "field": "instanceId"
    }
]
```

```
"placementStrategy": [
    {
        "type": "spread",
        "field": "attribute:ecs.availability-zone"
    },
    {
        "type": "binpack",
        "field": "memory"
    }
]
```

Amazon ECS – Task Placement Constraints

- distinctInstance
  - Tasks are placed on a different EC2 instance

```
"placementConstraints": [
    {
        "type": "distinctInstance"
    }
]
```

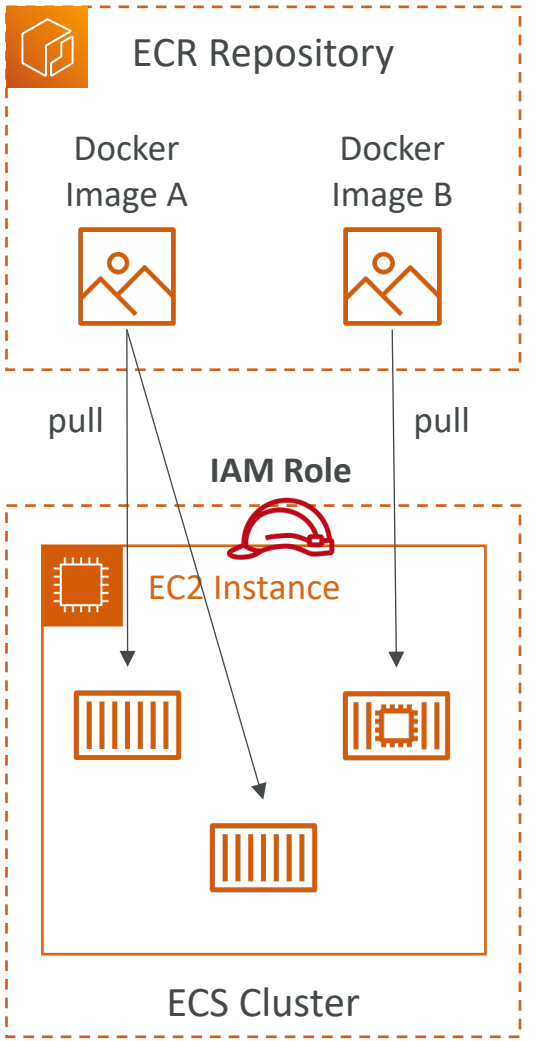
- memberOf
  - Tasks are placed on EC2 instances that satisfy a specified expression
  - Uses the **Cluster Query Language** (advanced)

```
"placementConstraints": [
    {
        "type": "memberOf",
        "expression": "attribute:ecs.instance-type =~ t2.*"
    }
]
```

```
"placementConstraints": [
    {
        "type": "memberOf",
        "expression": "attribute:ecs.availability-zone in
 [eu-west-2a, eu-west-2b]"
    }
]
```

Amazon ECR

- ECR = Elastic Container Registry
- Store and manage Docker images on AWS
- **Private** and **Public** repository (**Amazon ECR Public Gallery** https://gallery.ecr.aws)
- Fully integrated with ECS, backed by Amazon S3
- Access is controlled through IAM (permission errors => policy)
- Supports image vulnerability scanning, versioning, image tags, image lifecycle, …



Amazon ECR – Using AWS CLI

- Login Command
  - AWS CLI v2

```
aws ecr get-login-password --region region | docker login --username AWS
      --password-stdin aws_account_id.dkr.ecr.region.amazonaws.com
```

- Docker Commands
  - Push

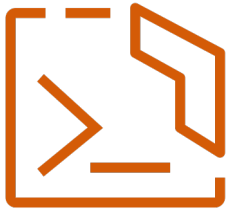
```
docker push aws_account_id.dkr.ecr.region.amazonaws.com/demo:latest
```

  - Pull

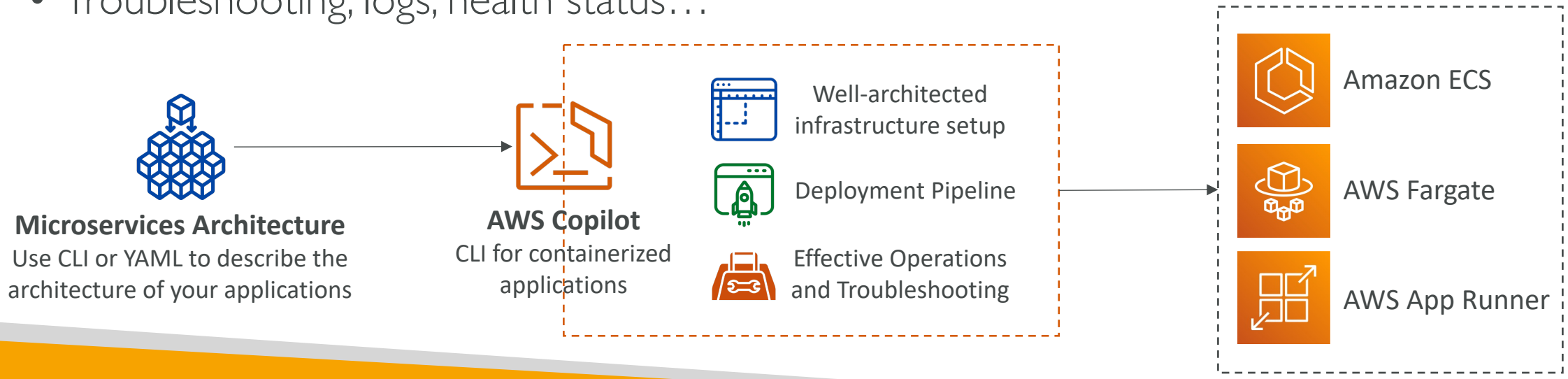
```
docker pull aws_account_id.dkr.ecr.region.amazonaws.com/demo:latest
```

  - In case an EC2 instance (or you) can't pull a Docker image, check IAM permissions

AWS Copilot



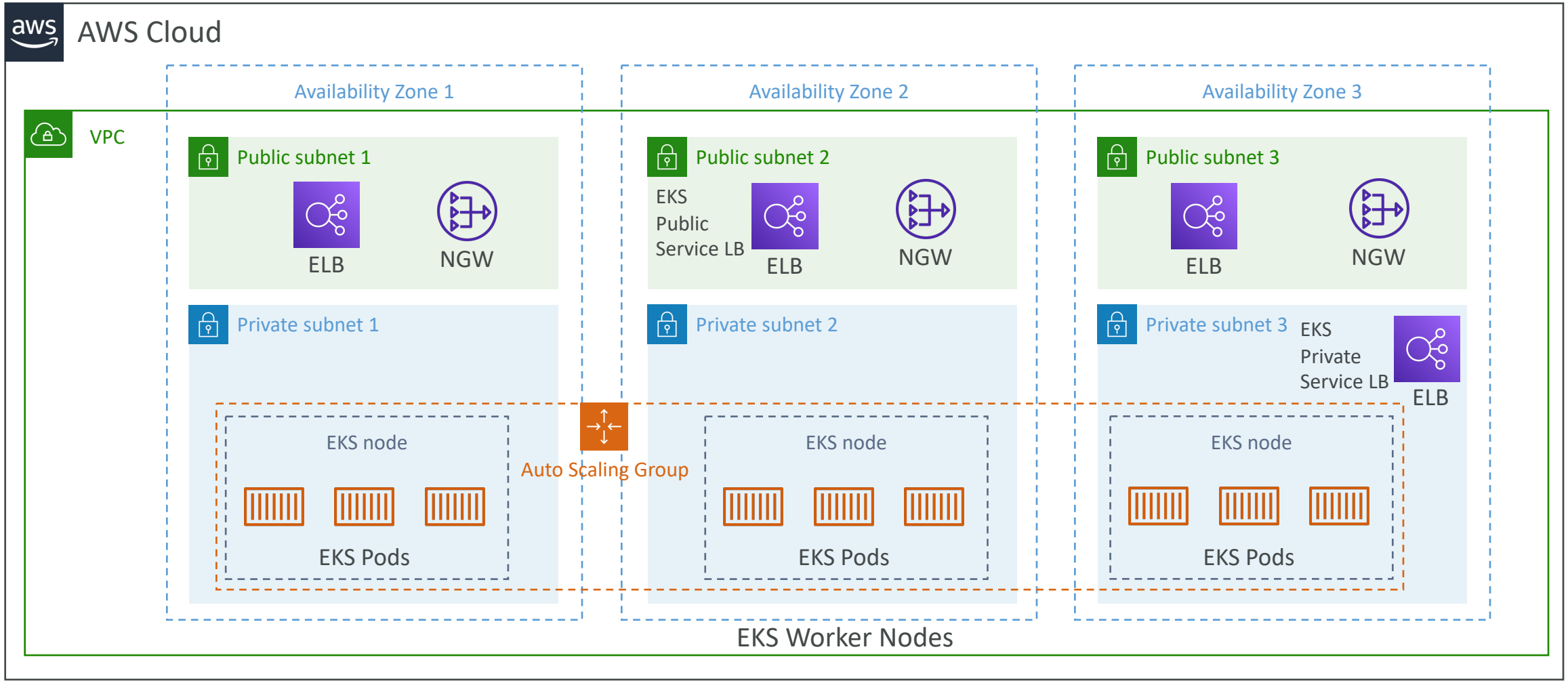
- CLI tool to build, release, and operate production-ready containerized apps
- Run your apps on **AppRunner**, **ECS**, and **Fargate**
- Helps you focus on building apps rather than setting up infrastructure
- Provisions all required infrastructure for containerized apps (ECS, VPC, ELB, ECR…)
- Automated deployments with one command using CodePipeline
- Deploy to multiple environments
- Troubleshooting, logs, health status…



Amazon EKS Overview

- Amazon EKS = Amazon Elastic **Kubernetes** Service
- It is a way to launch **managed Kubernetes clusters on AWS**
- Kubernetes is an **open-source system** for automatic deployment, scaling and management of containerized (usually Docker) application
- It's an alternative to ECS, similar goal but different API
- EKS supports **EC2** if you want to deploy worker nodes or **Fargate** to deploy serverless containers
- **Use case:** if your company is already using Kubernetes on-premises or in another cloud, and wants to migrate to AWS using Kubernetes
- **Kubernetes is cloud-agnostic** (can be used in any cloud – Azure, GCP…)
- For multiple regions, deploy one EKS cluster per region
- Collect logs and metrics using **CloudWatch Container Insights**

Amazon EKS - Diagram



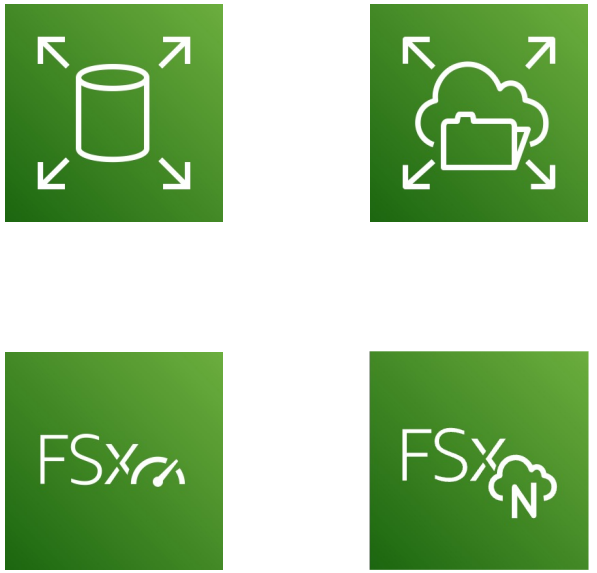
Amazon EKS – Node Types

- Managed Node Groups
  - Creates and manages Nodes (EC2 instances) for you
  - Nodes are part of an ASG managed by EKS
  - Supports On-Demand or Spot Instances
- Self-Managed Nodes
  - Nodes created by you and registered to the EKS cluster and managed by an ASG
  - You can use prebuilt AMI - Amazon EKS Optimized AMI
  - Supports On-Demand or Spot Instances
- AWS Fargate
  - No maintenance required; no nodes managed

Amazon EKS – Data Volumes

- Need to specify **StorageClass** manifest on your EKS cluster
- Leverages a **Container Storage Interface (CSI)** compliant driver

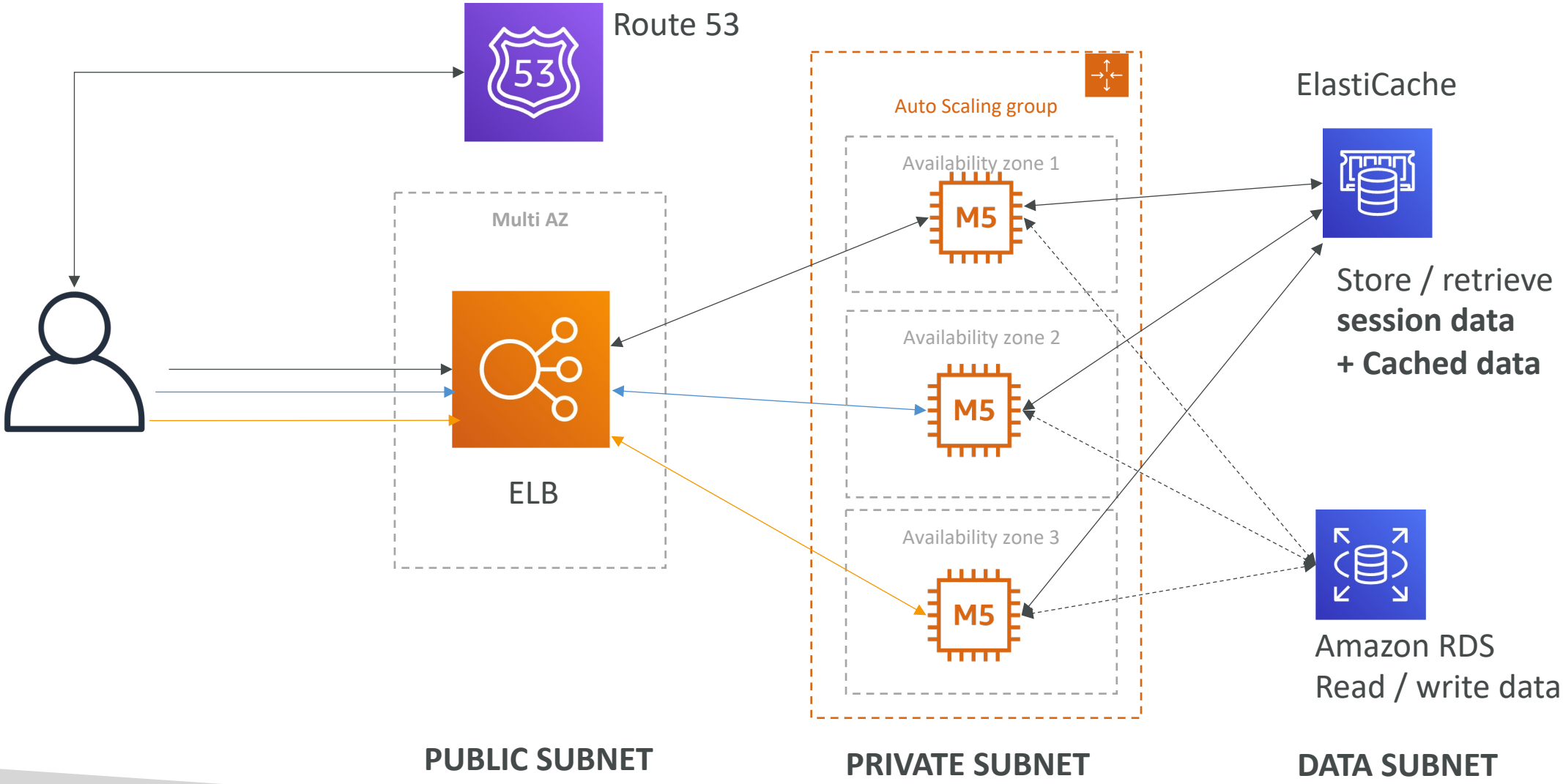
- Support for…
- Amazon EBS
- Amazon EFS (works with Fargate)
- Amazon FSx for Lustre
- Amazon FSx for NetApp ONTAP



AWS Elastic Beanstalk

Deploying applications in AWS safely and predictably

Typical architecture: Web App 3-tier



Developer problems on AWS

- Managing infrastructure
- Deploying Code
- Configuring all the databases, load balancers, etc
- Scaling concerns

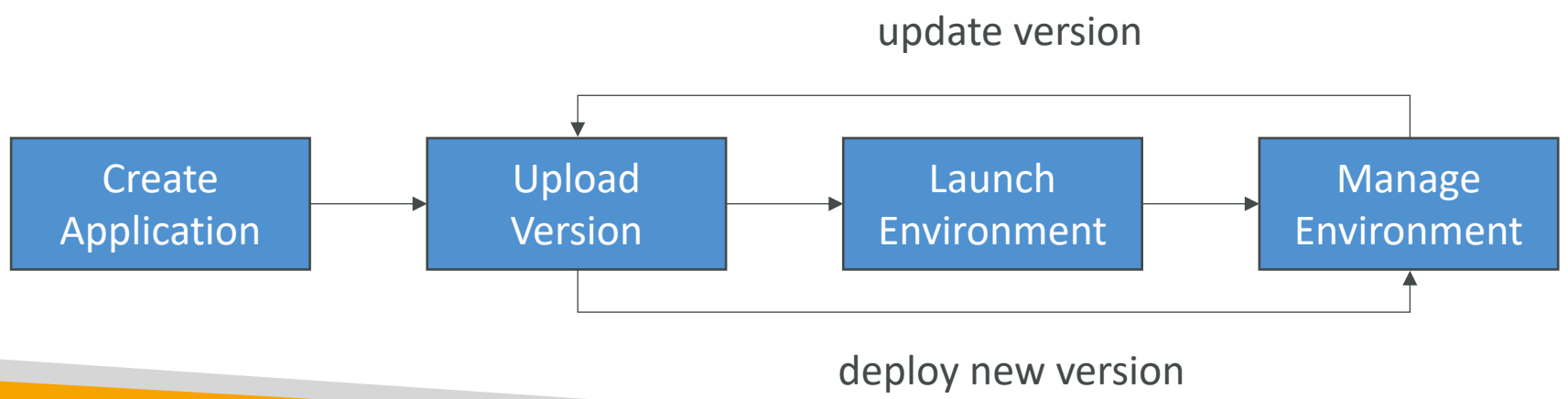
- Most web apps have the same architecture (ALB + ASG)
- All the developers want is for their code to run!
- Possibly, consistently across different applications and environments

Elastic Beanstalk – Overview

- Elastic Beanstalk is a developer centric view of deploying an application on AWS
- It uses all the component's we've seen before: EC2, ASG, ELB, RDS, …
- Managed service
  - Automatically handles capacity provisioning, load balancing, scaling, application health monitoring, instance configuration, …
  - Just the application code is the responsibility of the developer
- We still have full control over the configuration
- Beanstalk is free but you pay for the underlying instances

Elastic Beanstalk – Components

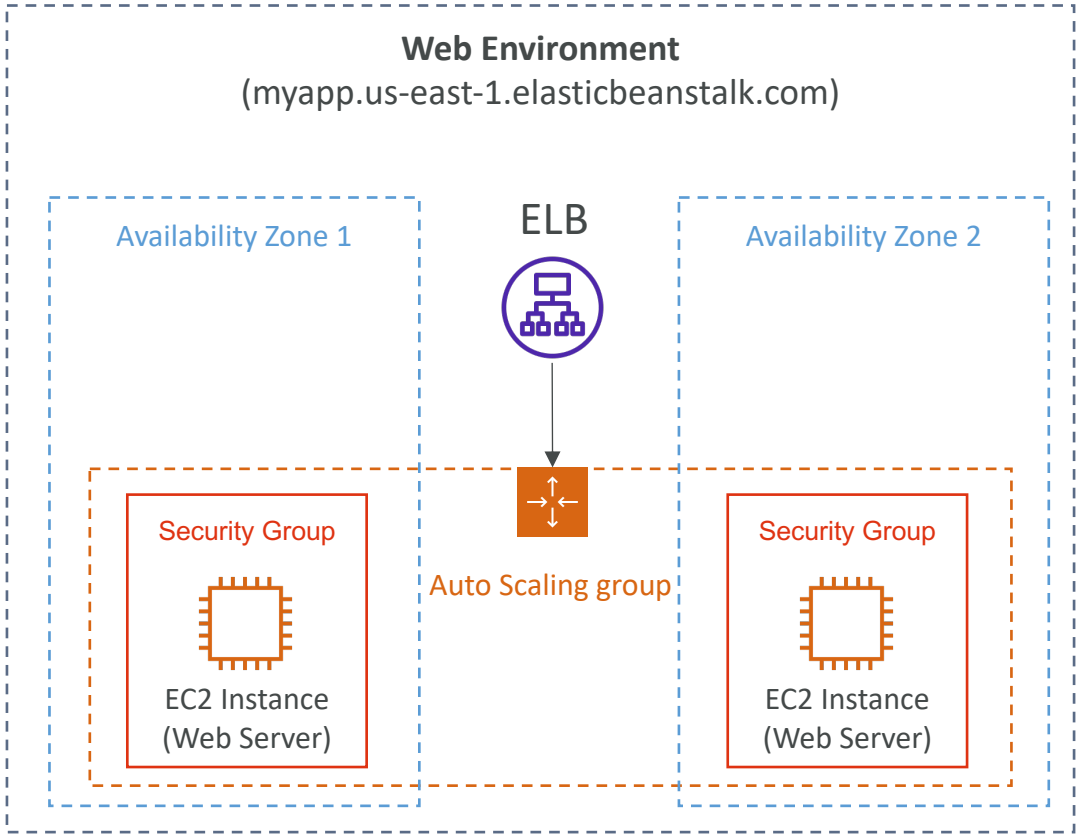
- **Application:** collection of Elastic Beanstalk components (environments, versions, configurations, …)
- **Application Version:** an iteration of your application code
- **Environment**
  - Collection of AWS resources running an application version (only one application version at a time)
  - **Tiers:** Web Server Environment Tier & Worker Environment Tier
  - You can create multiple environments (dev, test, prod, …)

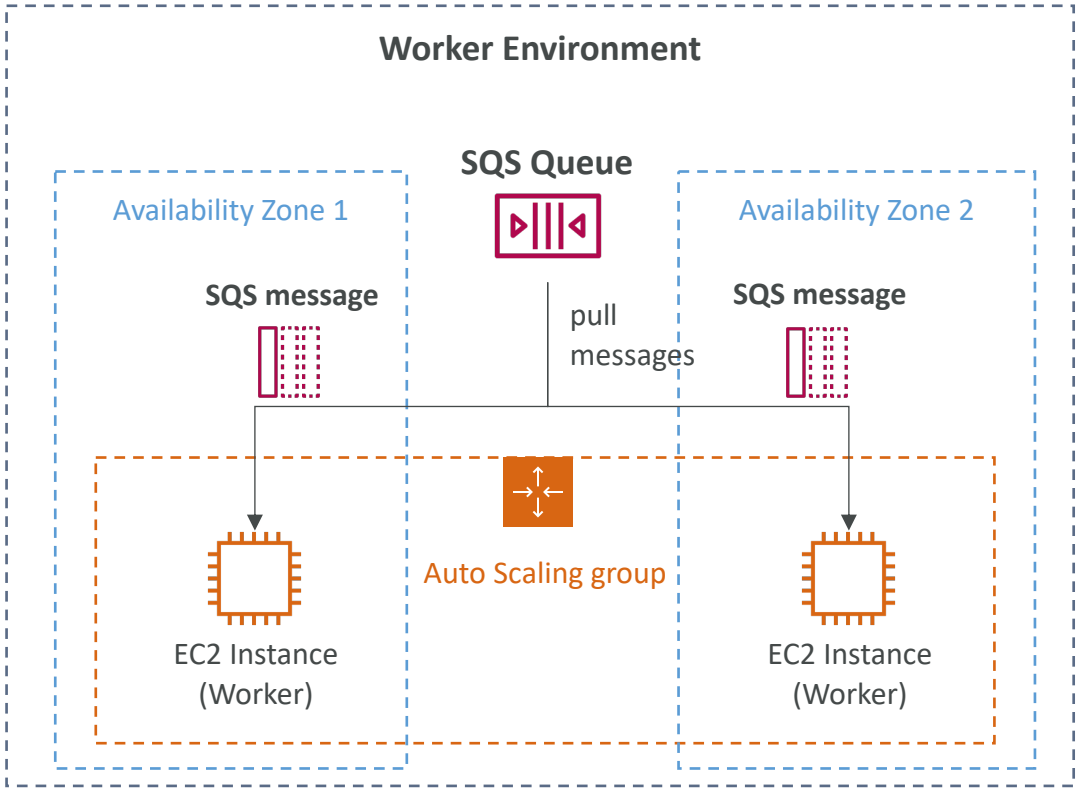


Elastic Beanstalk – Supported Platforms

- Go
- Java SE
- Java with Tomcat
- .NET Core on Linux
- .NET on Windows Server
- Node.js
- PHP
- Python
- Ruby
- Packer Builder
- Single Container Docker
- Multi-container Docker
- Preconfigured Docker

Web Server Tier vs. Worker Tier

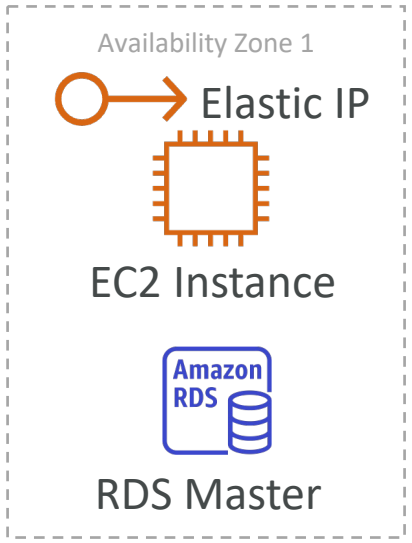




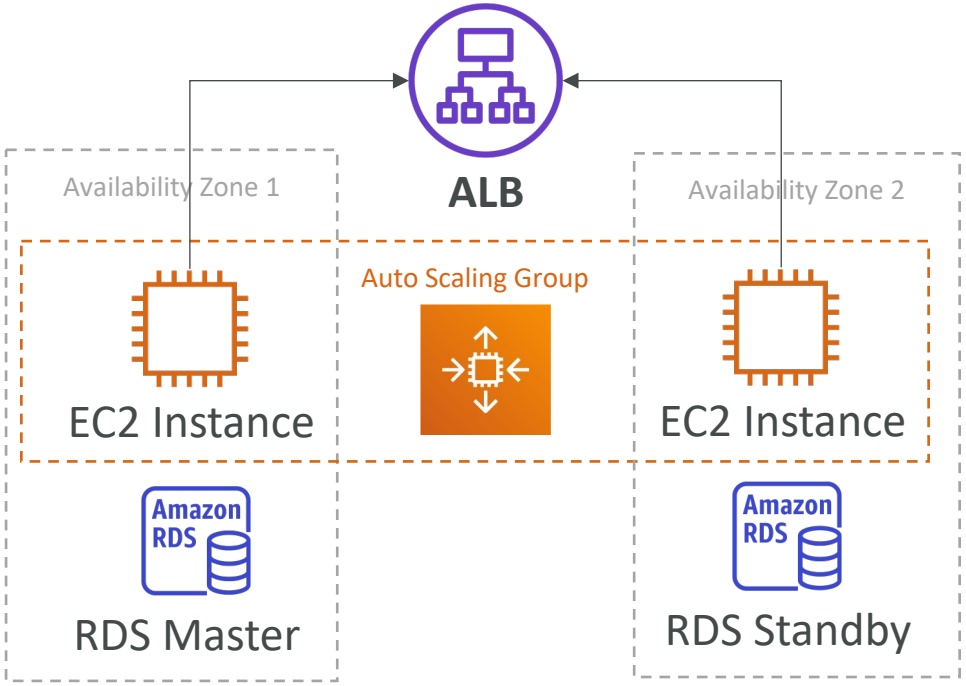
- Scale based on the number of SQS messages
- Can push messages to SQS queue from another Web Server Tier

Elastic Beanstalk Deployment Modes

Single Instance
Great for dev



High Availability with Load Balancer
Great for prod

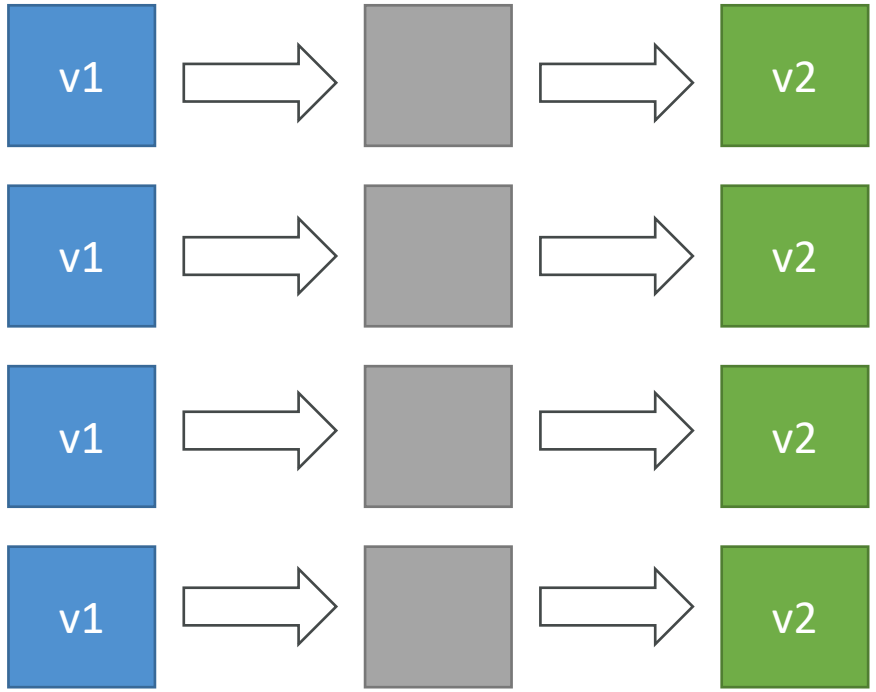


Beanstalk Deployment Options for Updates

- **All at once (deploy all in one go)** – fastest, but instances aren't available to serve traffic for a bit (downtime)
- **Rolling:** update a few instances at a time (bucket), and then move onto the next bucket once the first bucket is healthy
- **Rolling with additional batches:** like rolling, but spins up new instances to move the batch (so that the old application is still available)
- **Immutable:** spins up new instances in a new ASG, deploys version to these instances, and then swaps all the instances when everything is healthy
- **Blue Green:** create a new environment and switch over when ready
- **Traffic Splitting:** canary testing – send a small % of traffic to new deployment

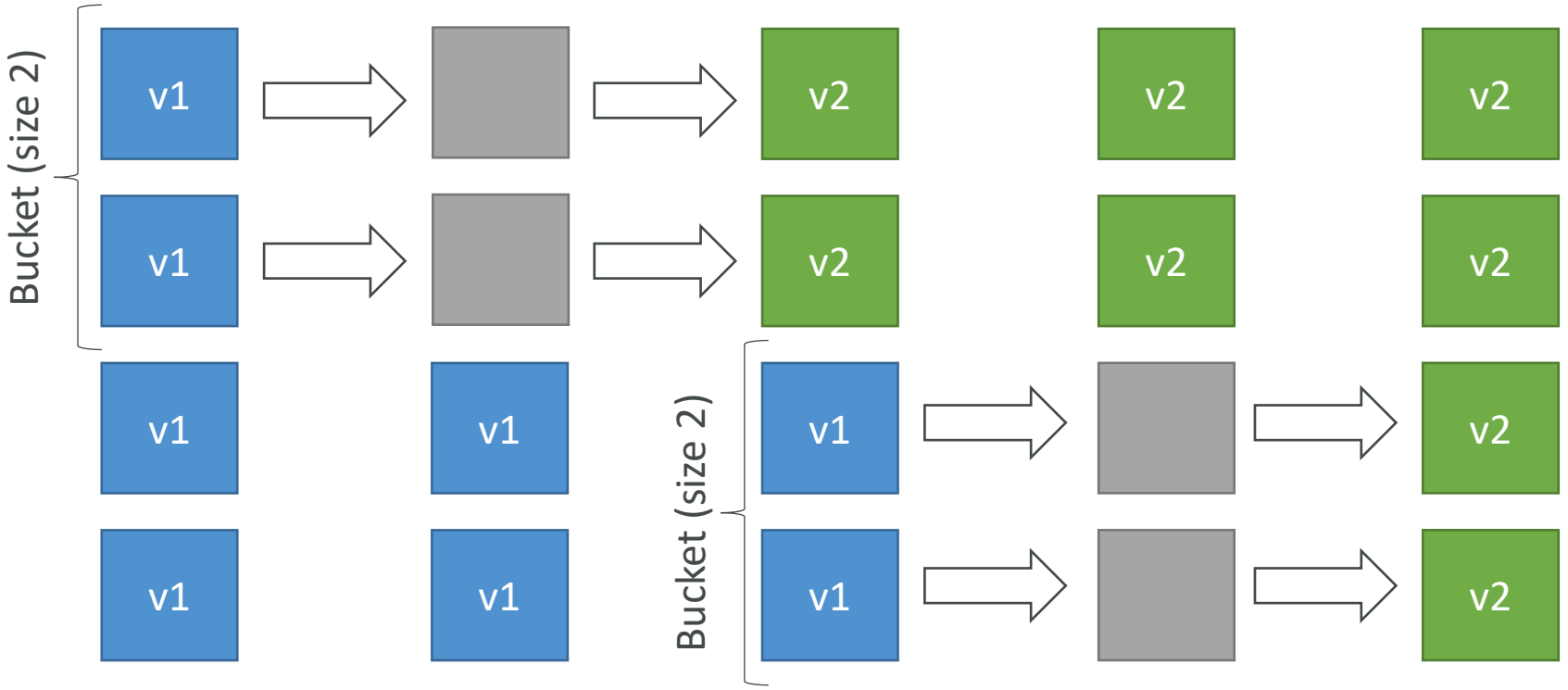
Elastic Beanstalk Deployment
All at once

- Fastest deployment
- Application has downtime
- Great for quick iterations in development environment
- No additional cost



Elastic Beanstalk Deployment Rolling

- Application is running below capacity
- Can set the bucket size
- Application is running both versions simultaneously
- No additional cost
- Long deployment

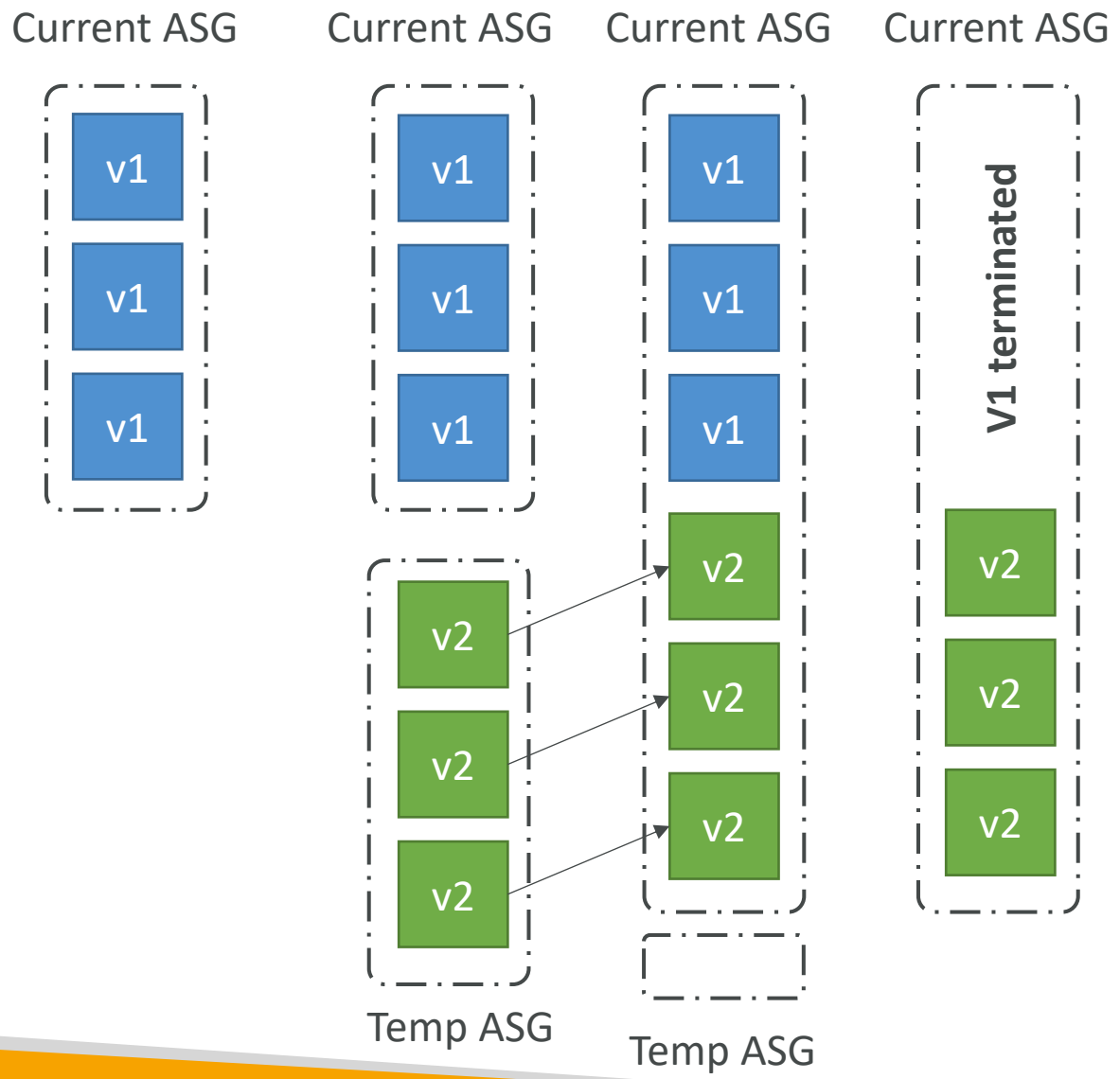


Elastic Beanstalk Deployment Rolling with additional batches

- Application is running at capacity
- Can set the bucket size
- Application is running both versions simultaneously
- Small additional cost
- Additional batch is removed at the end of the deployment
- Longer deployment
- Good for prod

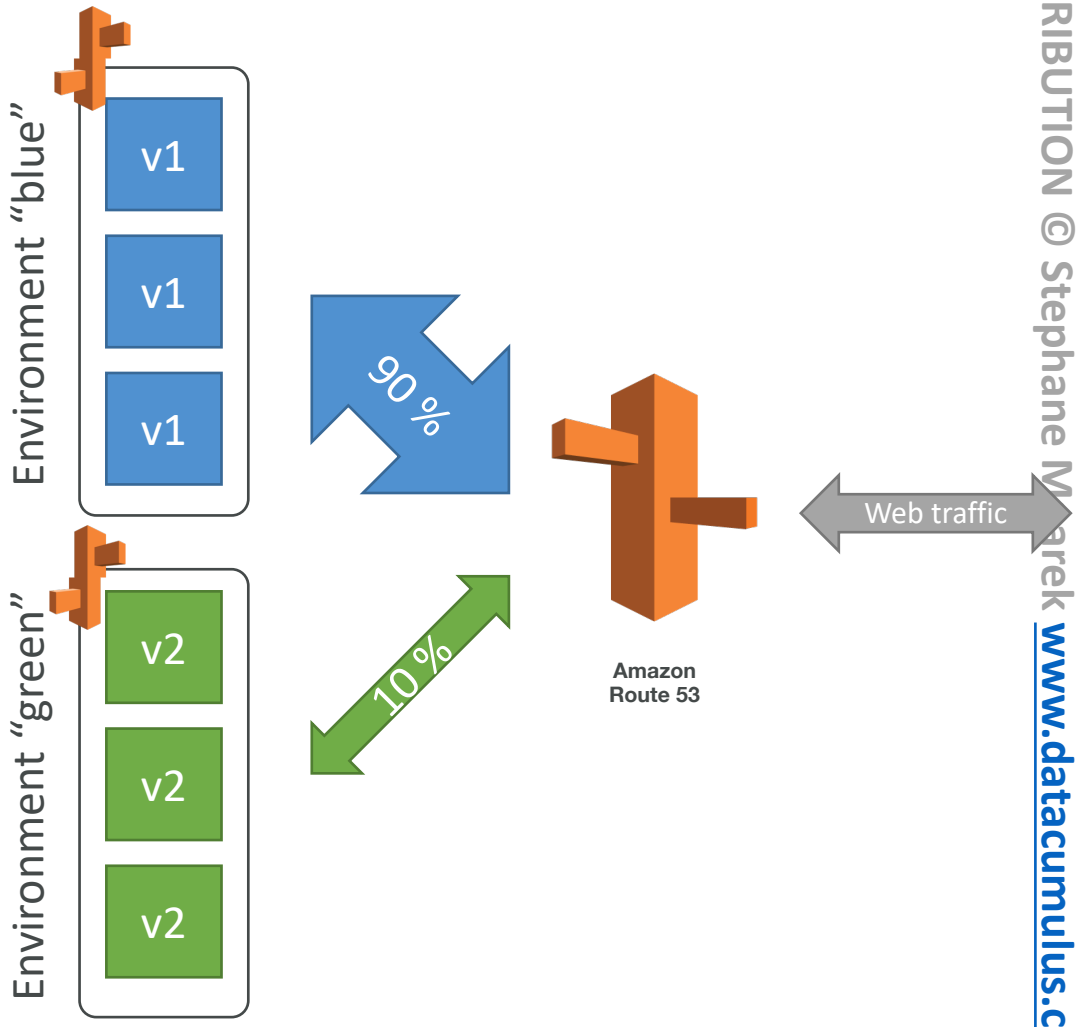
Elastic Beanstalk Deployment Immutable

- Zero downtime
- New Code is deployed to new instances on a temporary ASG
- High cost, double capacity
- Longest deployment
- Quick rollback in case of failures (just terminate new ASG)
- Great for prod



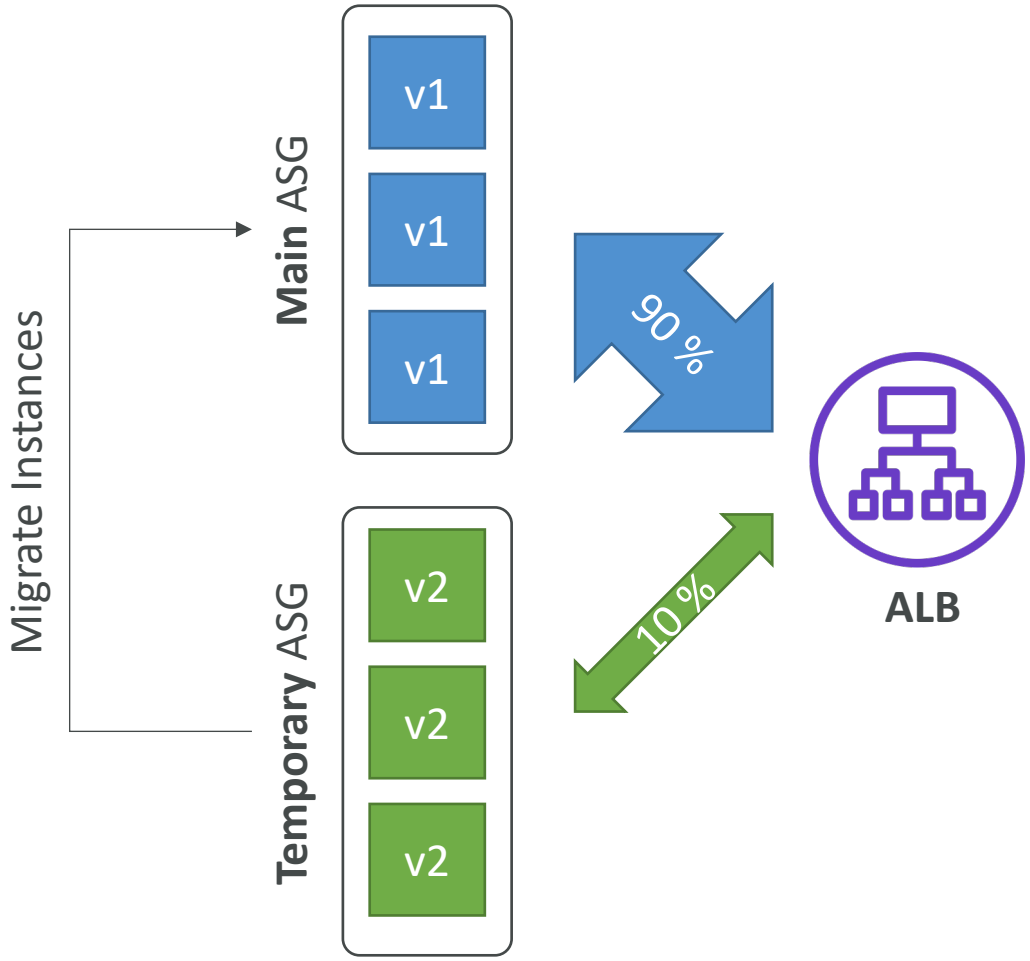
Elastic Beanstalk Deployment Blue / Green

- Not a "direct feature" of Elastic Beanstalk
- Zero downtime and release facility
- Create a new "stage" environment and deploy v2 there
- The new environment (green) can be validated independently and roll back if issues
- Route 53 can be setup using weighted policies to redirect a little bit of traffic to the stage environment
- Using Beanstalk, "swap URLs" when done with the environment test



Elastic Beanstalk - Traffic Splitting

- Canary Testing
- New application version is deployed to a temporary ASG with the same capacity
- A small % of traffic is sent to the temporary ASG for a configurable amount of time
- Deployment health is monitored
- If there's a deployment failure, this triggers an **automated rollback (very quick)**
- No application downtime
- New instances are migrated from the temporary to the original ASG
- Old application version is then terminated



Elastic Beanstalk Deployment Summary from AWS Doc

- https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features.deploy-existing-version.html

Deployment methods

Method	Impact of failed deployment	Deploy time	Zero downtime	No DNS change	Rollback process	Code deployed to
All at once	Downtime	🕒	x	✓	Manual redeploy	Existing instances
Rolling	Single batch out of service; any successful batches before failure running new application version	🕒🕒 †	✓	✓	Manual redeploy	Existing instances
Rolling with an additional batch	Minimal if first batch fails; otherwise, similar to **Rolling**	🕒🕒🕒 †	✓	✓	Manual redeploy	New and existing instances
Immutable	Minimal	🕒🕒🕒🕒	✓	✓	Terminate new instances	New instances
Traffic splitting	Percentage of client traffic routed to new version temporarily impacted	🕒🕒🕒🕒 ††	✓	✓	Reroute traffic and terminate new instances	New instances
Blue/green	Minimal	🕒🕒🕒🕒	✓	x	Swap URL	New instances

Elastic Beanstalk CLI

- We can install an additional CLI called the "EB cli" which makes working with Beanstalk from the CLI easier
- Basic commands are:
  - eb create
  - eb status
  - eb health
  - eb events
  - eb logs
  - eb open
  - eb deploy
  - eb config
  - eb terminate
- It's helpful for your automated deployment pipelines!

Elastic Beanstalk Deployment Process

- Describe dependencies
  (requirements.txt for Python, package.json for Node.js)
- Package code as zip, and describe dependencies
  - Python: requirements.txt
  - Node.js: package.json
- **Console:** upload zip file (creates new app version), and then deploy
- **CLI:** create new app version using CLI (uploads zip), and then deploy

- Elastic Beanstalk will deploy the zip on each EC2 instance, resolve dependencies and start the application

Beanstalk Lifecycle Policy

- Elastic Beanstalk can store at most 1000 application versions
- If you don't remove old versions, you won't be able to deploy anymore
- To phase out old application versions, use a **lifecycle policy**
  - Based on time (old versions are removed)
  - Based on space (when you have too many versions)
- Versions that are currently used won't be deleted
- Option not to delete the source bundle in S3 to prevent data loss

Elastic Beanstalk Extensions

- A zip file containing our code must be deployed to Elastic Beanstalk
- All the parameters set in the UI can be configured with code using files
- Requirements:
  - in the .ebextensions/ directory in the root of source code
  - YAML / JSON format
  - **.config** extensions (example: logging.config)
  - Able to modify some default settings using: option_settings
  - Ability to add resources such as RDS, ElastiCache, DynamoDB, etc…
- Resources managed by .ebextensions get deleted if the environment goes away

Elastic Beanstalk Under the Hood

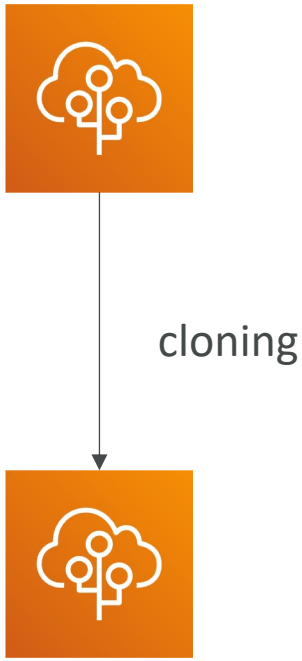
- Under the hood, Elastic Beanstalk relies on CloudFormation
- CloudFormation is used to provision other AWS services (we'll see later)

Elastic Beanstalk **CloudFormation**

- Use case: you can define CloudFormation resources in your **.ebextensions** to provision ElastiCache, an S3 bucket, anything you want!
- Let's have a sneak peak into it!

Elastic Beanstalk Cloning

- Clone an environment with the exact same configuration
- Useful for deploying a "test" version of your application

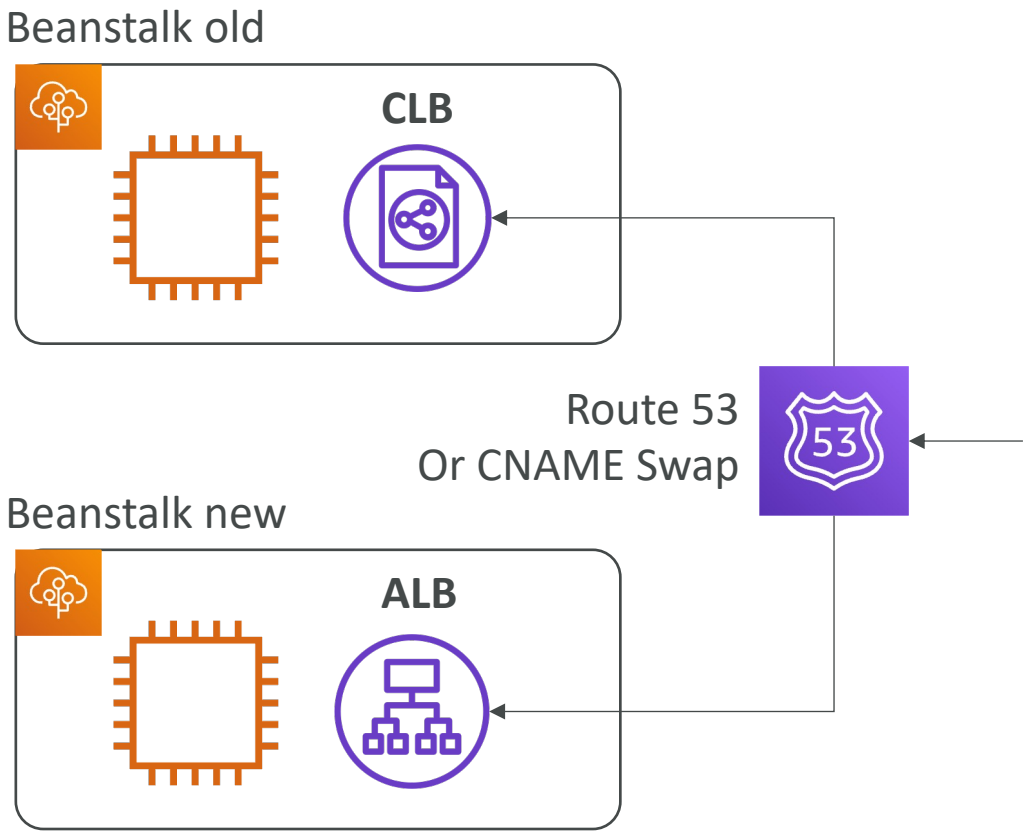


- All resources and configuration are preserved:
  - Load Balancer type and configuration
  - RDS database type (but the data is not preserved)
  - Environment variables

- After cloning an environment, you can change settings

Elastic Beanstalk Migration: Load Balancer

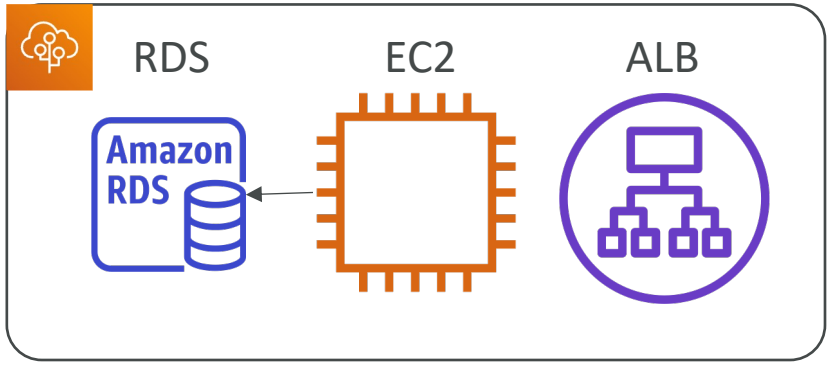
- After creating an Elastic Beanstalk environment, **you cannot change the Elastic Load Balancer type** (only the configuration)
- To migrate:
  1. create a new environment with the same configuration except LB (can't clone)
  2. deploy your application onto the new environment
  3. perform a CNAME swap or Route 53 update



RDS with Elastic Beanstalk

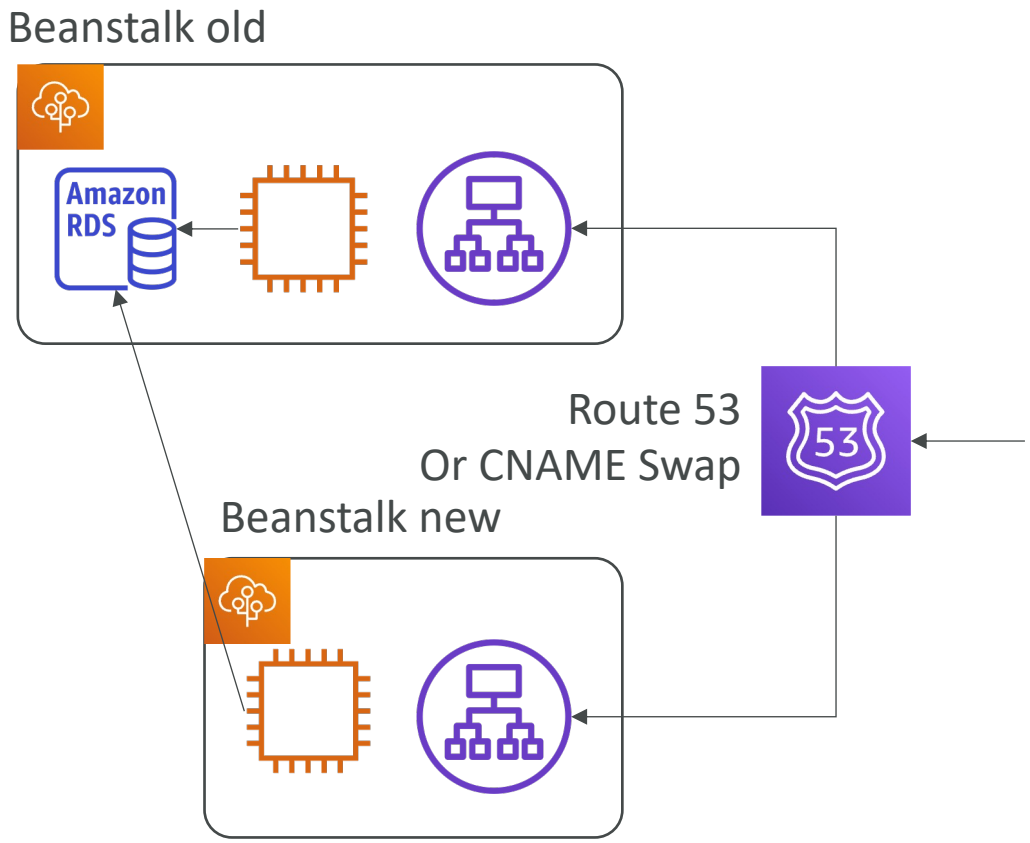
- RDS can be provisioned with Beanstalk, which is great for dev / test
- This is not great for prod as the database lifecycle is tied to the Beanstalk environment lifecycle
- The best for prod is to separately create an RDS database and provide our EB application with the connection string

Beanstalk with RDS



Elastic Beanstalk Migration: Decouple RDS

1. Create a snapshot of RDS DB (as a safeguard)
2. Go to the RDS console and protect the RDS database from deletion
3. Create a new Elastic Beanstalk environment, without RDS, point your application to existing RDS
4. perform a CNAME swap (blue/green) or Route 53 update, confirm working
5. Terminate the old environment (RDS won't be deleted)
6. Delete CloudFormation stack (in DELETE_FAILED state)



AWS CloudFormation

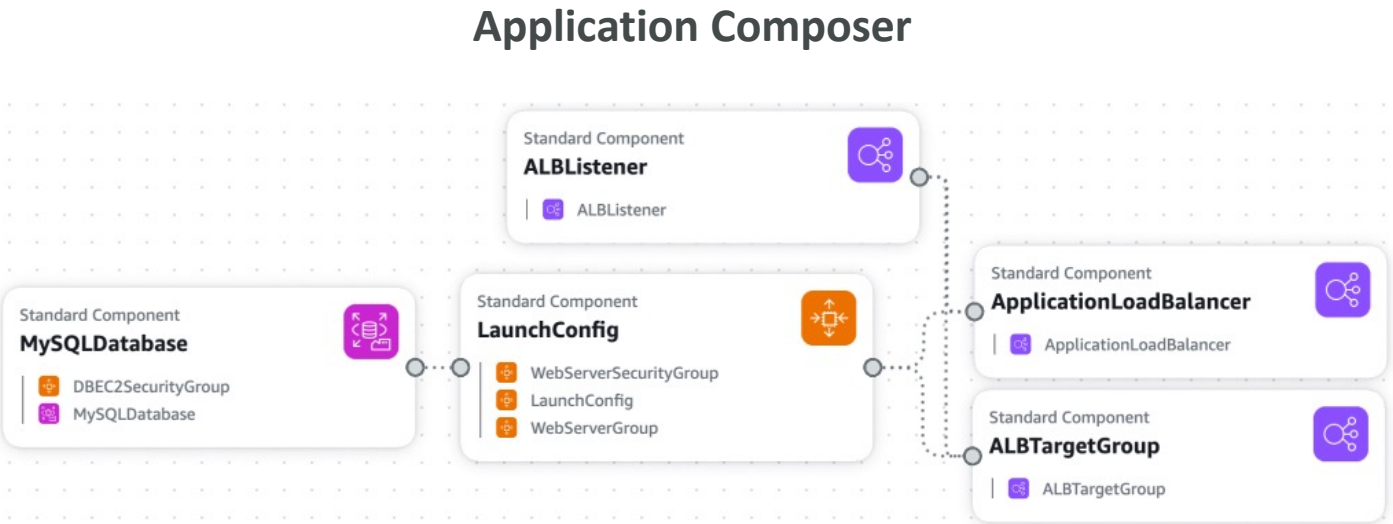
Managing your infrastructure as code

AWS CloudFormation

- CloudFormation is a declarative way of outlining your AWS Infrastructure, for any resources (most of them are supported)
- For example, within a CloudFormation template, you say:
  - I want a security group
  - I want two EC2 instances using this security group
  - I want two Elastic IPs for these EC2 instances
  - I want an S3 bucket
  - I want a load balancer (ELB) in front of these EC2 instances
- Then CloudFormation creates those for you, in the **right order**, with the **exact configuration** that you specify

CloudFormation – Template Example

```
1   AWSTemplateFormatVersion: '2010-09-09'
2 > Description: 'AWS CloudFormation Sample Template LAMP_Multi_AZ: C
10 > Parameters: ⋯
175 > Mappings: ⋯
481 Resources:
482   ApplicationLoadBalancer:
483     Type: AWS::ElasticLoadBalancingV2::LoadBalancer
484     Properties:
485       Subnets: !Ref Subnets
486   ALBListener:
487     Type: AWS::ElasticLoadBalancingV2::Listener
488     Properties:
489       DefaultActions:
490         - Type: forward
491           TargetGroupArn: !Ref ALBTargetGroup
492       LoadBalancerArn: !Ref ApplicationLoadBalancer
493       Port: '80'
494       Protocol: HTTP
495   ALBTargetGroup:
496     Type: AWS::ElasticLoadBalancingV2::TargetGroup
497     Properties:
498       HealthCheckIntervalSeconds: 10
499       HealthCheckTimeoutSeconds: 5
500       HealthyThresholdCount: 2
501       Port: 80
502       Protocol: HTTP
503       UnhealthyThresholdCount: 5
504       VpcId: !Ref VpcId
505       TargetGroupAttributes:
506         - Key: stickiness.enabled
507           Value: 'true'
```



Benefits of AWS CloudFormation (1/2)

- Infrastructure as code
  - No resources are manually created, which is excellent for control
  - The code can be version controlled for example using Git
  - Changes to the infrastructure are reviewed through code

- Cost
  - Each resources within the stack is tagged with an identifier so you can easily see how much a stack costs you
  - You can estimate the costs of your resources using the CloudFormation template
  - Savings strategy: In Dev, you could automation deletion of templates at 5 PM and recreated at 8 AM, safely

Benefits of AWS CloudFormation (2/2)

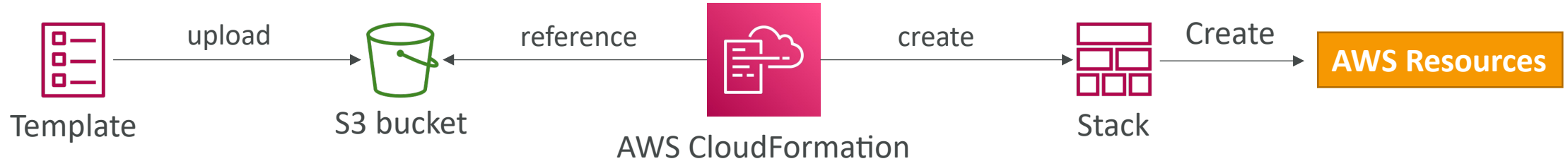
- Productivity
  - Ability to destroy and re-create an infrastructure on the cloud on the fly
  - Automated generation of Diagram for your templates!
  - Declarative programming (no need to figure out ordering and orchestration)

- Separation of concern: create many stacks for many apps, and many layers. Ex:
  - VPC stacks
  - Network stacks
  - App stacks

- Don't re-invent the wheel
  - Leverage existing templates on the web!
  - Leverage the documentation

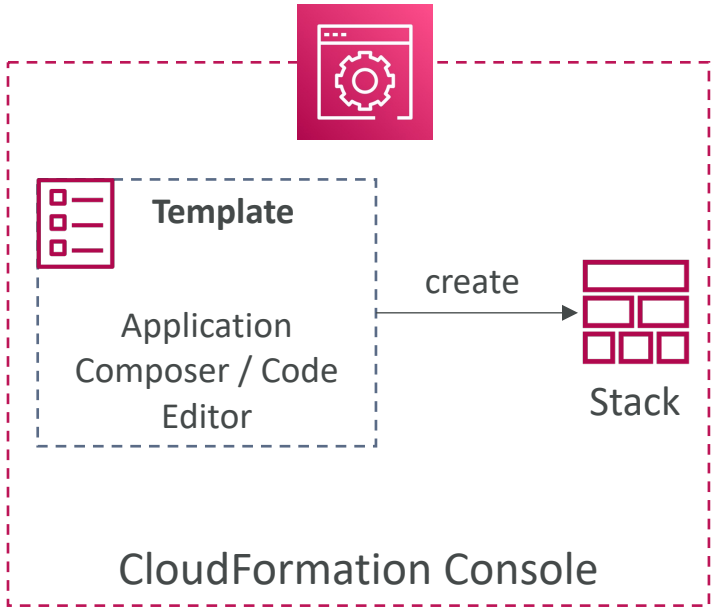
How CloudFormation Works

- Templates must be uploaded in S3 and then referenced in CloudFormation
- To update a template, we can't edit previous ones. We have to re-upload a new version of the template to AWS
- Stacks are identified by a name
- Deleting a stack deletes every single artifact that was created by CloudFormation.

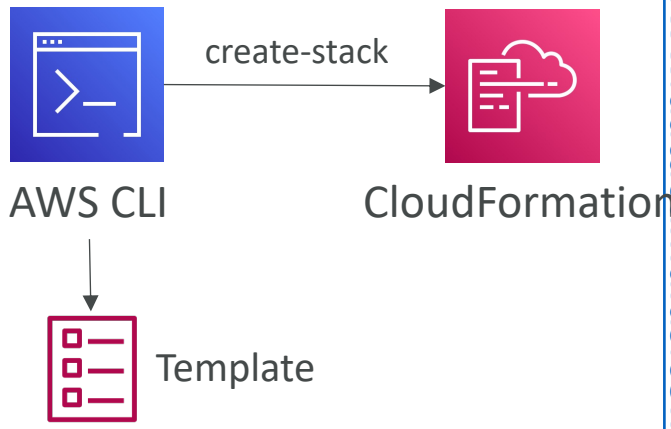


Deploying CloudFormation Templates

- Manual way
  - Editing templates in Application Composer or code editor
  - Using the console to input parameters, etc…
  - We'll mostly do this way in the course for learning purposes



- Automated way
  - Editing templates in a YAML file
  - Using the AWS CLI (Command Line Interface) to deploy the templates, or using a Continuous Delivery (CD) tool
  - Recommended way when you fully want to automate your flow



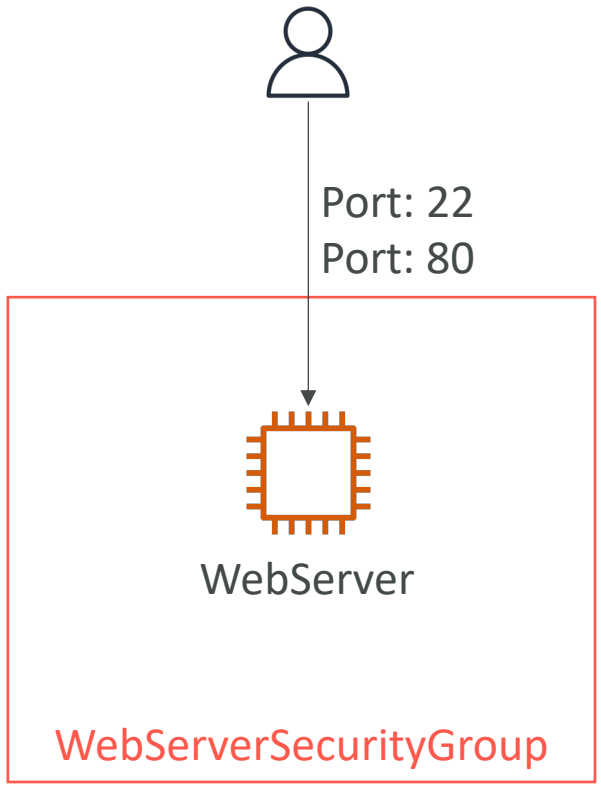
CloudFormation – Building Blocks

- Template's Components
  - AWSTemplateFormatVersion – identifies the capabilities of the template "2010-09-09"
  - Description – comments about the template
  - Resources (MANDATORY) – your AWS resources declared in the template
  - Parameters – the dynamic inputs for your template
  - Mappings – the static variables for your template
  - Outputs – references to what has been created
  - Conditionals – list of conditions to perform resource creation

- Template's Helpers
  - References
  - Functions

Introductory Example

- We're going to create a simple EC2 instance
- And we're going to add security group to it
- For now, forget about the code syntax
- We'll look at the structure of the files later



- We'll see how in no-time, we are able to get started with CloudFormation!

YAML Crash Course

```yaml
invoice: 34843
date: 2001-01-23
bill-to:
  given: Chris
  family: Dumars
  address:
    lines: |
      458 Walkman Dr.
      Suite #292
    city: Royal Oak
    state: MI
    postal: 48046
products:
  - sku: BL394D
    quantity: 4
    description: Basketball
    price: 450.00
  - sku: BL4438H
    quantity: 1
    description: Super Hoop
    price: 2392.00
```

- YAML and JSON are the languages you can use for CloudFormation
- JSON is horrible for CF
- YAML is great in so many ways
- Let's learn a bit about it!

- Key value Pairs
- Nested objects
- Support Arrays
- Multi line strings
- Can include comments!

CloudFormation – Resources

- Resources are the core of your CloudFormation template (MANDATORY)
- They represent the different AWS Components that will be created and configured
- Resources are declared and can reference each other

- AWS figures out creation, updates and deletes of resources for us
- There are over 700 types of resources (!)
- Resource types identifiers are of the form:

```
service-provider::service-name::data-type-name
```

How do I find Resources documentation?

- I can't teach you all the 700+ resources, but I can teach you how to learn how to use them
- All the resources can be found here: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-template-resource-type-ref.html
- Then, we just read the docs ☺
- Example here (for an EC2 instance): https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-ec2-instance.html

Analysis of CloudFormation Template

- Going back to the example of the introductory lecture, let's learn why it was written this way.
- Relevant documentation can be found here:
  - https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-ec2-instance.html
  - https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-ec2-securitygroup.html
  - http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-ec2-eip.html

CloudFormation – Resources FAQ

- Can I create a dynamic number of resources?

➢ Yes, you can by using CloudFormation Macros and Transform

➢ It is not in the scope of this course

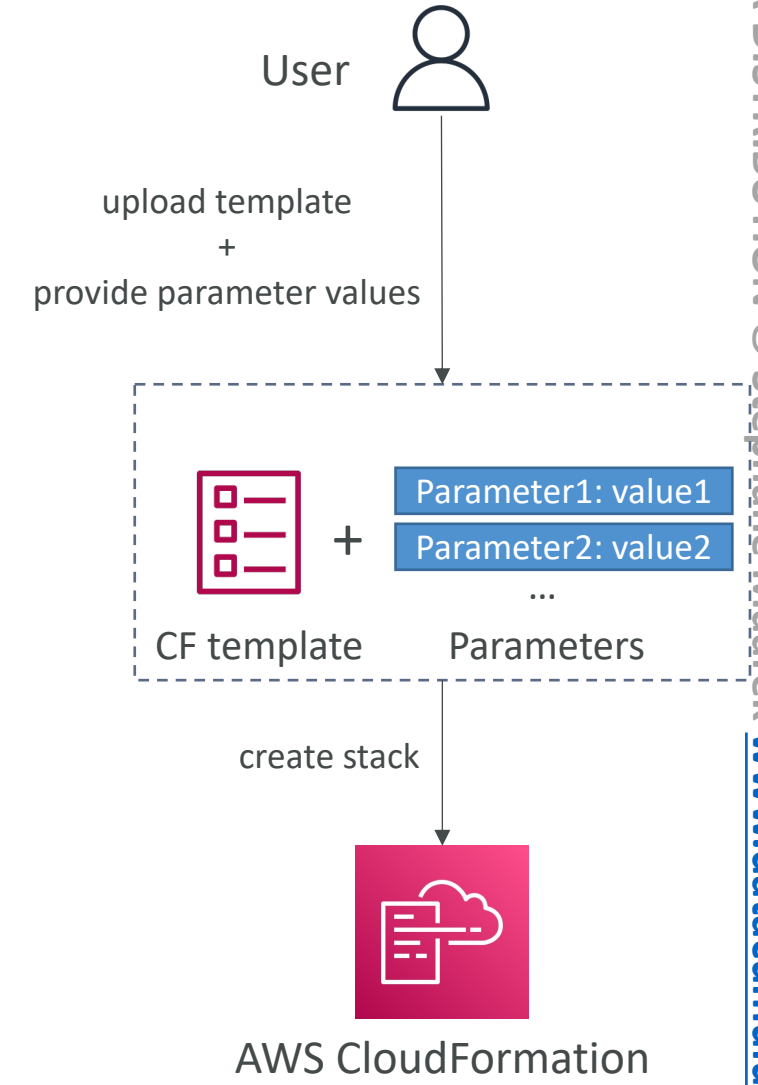
- Is every AWS Service supported?

➢ Almost. Only a select few niches are not there yet

➢ You can work around that using CloudFormation Custom Resources

CloudFormation – Parameters

- Parameters are a way to provide inputs to your AWS CloudFormation template
- They're important to know about if:
  - You want to reuse your templates across the company
  - Some inputs can not be determined ahead of time
- Parameters are extremely powerful, controlled, and can prevent errors from happening in your templates, thanks to types



When should you use a Parameter?

```
Parameters:
  SecurityGroupDescription:
    Description: Security Group Description
    Type: String
```

- Ask yourself this:
  - Is this CloudFormation resource configuration likely to change in the future?
  - If so, make it a parameter
- You won't have to re-upload a template to change its content ☺

CloudFormation – Parameters Settings

- Parameters can be controlled by all these settings:
  - Type:
    - String
    - Number
    - CommaDelimitedList
    - List<Number>
    - AWS-Specific Parameter (to help catch invalid values – match against existing values in the AWS account)
    - List<AWS-Specific Parameter>
    - SSM Parameter (get parameter value from SSM Parameter store)
  - Description
  - ConstraintDescription (String)
  - Min/MaxLength
  - Min/MaxValue
  - Default
  - AllowedValues (array)
  - AllowedPattern (regex)
  - NoEcho (Boolean)

CloudFormation – Parameters Example

AllowedValues

```yaml
Parameters:
  InstanceType:
    Description: Choose an EC2 instance type
    Type: String
    AllowedValues:
      - t2.micro
      - t2.small
      - t2.medium
    Default: t2.micro

Resources:
  MyEC2Instance:
    Type: AWS::EC2::Instance
    Properties:
      InstanceType: !Ref InstanceType
      ImageId: ami-0c02fb55956c7d316
```

NoEcho

```yaml
Parameters:
  DBPassword:
    Description: The database admin password
    Type: String
    NoEcho: true

Resources:
  MyDBInstance:
    Type: AWS::RDS::DBInstance
    Properties:
      DBInstanceClass: db.t2.micro
      AllocatedStorage: 20
      Engine: mysql
      MasterUsername: admin
      MasterUserPassword: !Ref DBPassword
      DBInstanceIdentifier: mydbinstance
```

How to Reference a Parameter?

```
Resources:
  DBSubnet1:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref MyVPC
```

- The Fn::Ref function can be leveraged to reference parameters
- Parameters can be used anywhere in a template
- The shorthand for this in YAML is !Ref
- The function can also reference other elements within the template

CloudFormation – Pseudo Parameters

- AWS offers us Pseudo Parameters in any CloudFormation template
- These can be used at any time and are enabled by default
- Important pseudo parameters:

Reference Value	Example Returned Value
AWS::AccountId	123456789012
AWS::Region	us-east-1
AWS::StackId	arn:aws:cloudformation:us-east-1:123456789012:stack/MyStack/1c2fa620-982a-11e3-aff7-50e2416294e0
AWS::StackName	MyStack
AWS::NotificationARNs	[arn:aws:sns:us-east-1:123456789012:MyTopic]
AWS::NoValue	Doesn't return a value

CloudFormation – Mappings

- Mappings are fixed variables within your CloudFormation template
- They're very handy to differentiate between different environments (dev vs prod), regions (AWS regions), AMI types…
- All the values are hardcoded within the template

```
Mappings:
  Mapping01:
    Key01:
      Name: Value01
    Key02:
      Name: Value02
    Key03:
      Name: Value03
```

```
RegionMap:
  us-east-1:
    HVM64: ami-0ff8a91507f77f867
    HVMG2: ami-0a584ac55a7631c0c
  us-west-1:
    HVM64: ami-0bdb828fd58c52235
    HVMG2: ami-066ee5fd4a9ef77f1
  eu-west-1:
    HVM64: ami-047bb4163c506cd98
    HVMG2: ami-0a7c483d527806435
```

Accessing Mapping Values (Fn::FindInMap)

- We use Fn::FindInMap to return a named value from a specific key
- !FindInMap [MapName, TopLevelKey, SecondLevelKey]

```
Mappings:
  RegionMap:
    us-east-1:
      HVM64: ami-0ff8a91507f77f867
      HVMG2: ami-0a584ac55a7631c0c
    us-west-1:
      HVM64: ami-0bdb828fd58c52235
      HVMG2: ami-066ee5fd4a9ef77f1

Resources:
  MyEC2Instance:
    Type: AWS::EC2::Instance
    Properties:
      ImageId: !FindInMap [RegionMap, !Ref "AWS::Region", HVM64]
      InstanceType: t2.micro
```

Mappings work great for AMIs
Because AMIs are region-specific!

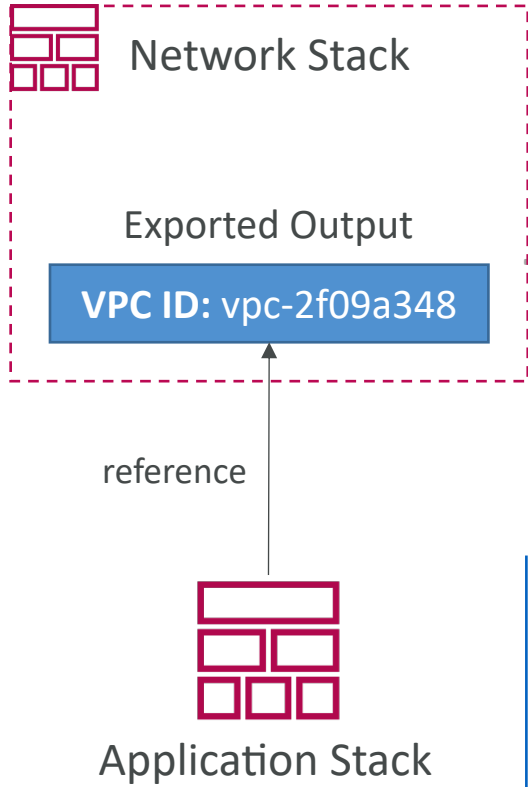
When would you use Mappings vs. Parameters?

- Mappings are great when you know in advance all the values that can be taken and that they can be deduced from variables such as
  - Region
  - Availability Zone
  - AWS Account
  - Environment (dev vs prod)
  - etc…
- They allow safer control over the template

- Use parameters when the values are really user specific

CloudFormation – Outputs

- The Outputs section declares *optional* outputs values that we can import into other stacks (if you export them first)!
- You can also view the outputs in the AWS Console or in using the AWS CLI
- They're very useful for example if you define a network CloudFormation, and output the variables such as VPC ID and your Subnet IDs
- It's the best way to perform some collaboration cross stack, as you let expert handle their own part of the stack



CloudFormation – Outputs

- Creating a SSH Security Group as part of one template
- We create an output that references that security group

```
Outputs:
  StackSSHSecurityGroup:
    Description: The SSH Security Group for our Company
    Value: !Ref MyCompanyWideSSHSecurityGroup
    Export:
      Name: SSHSecurityGroup
```

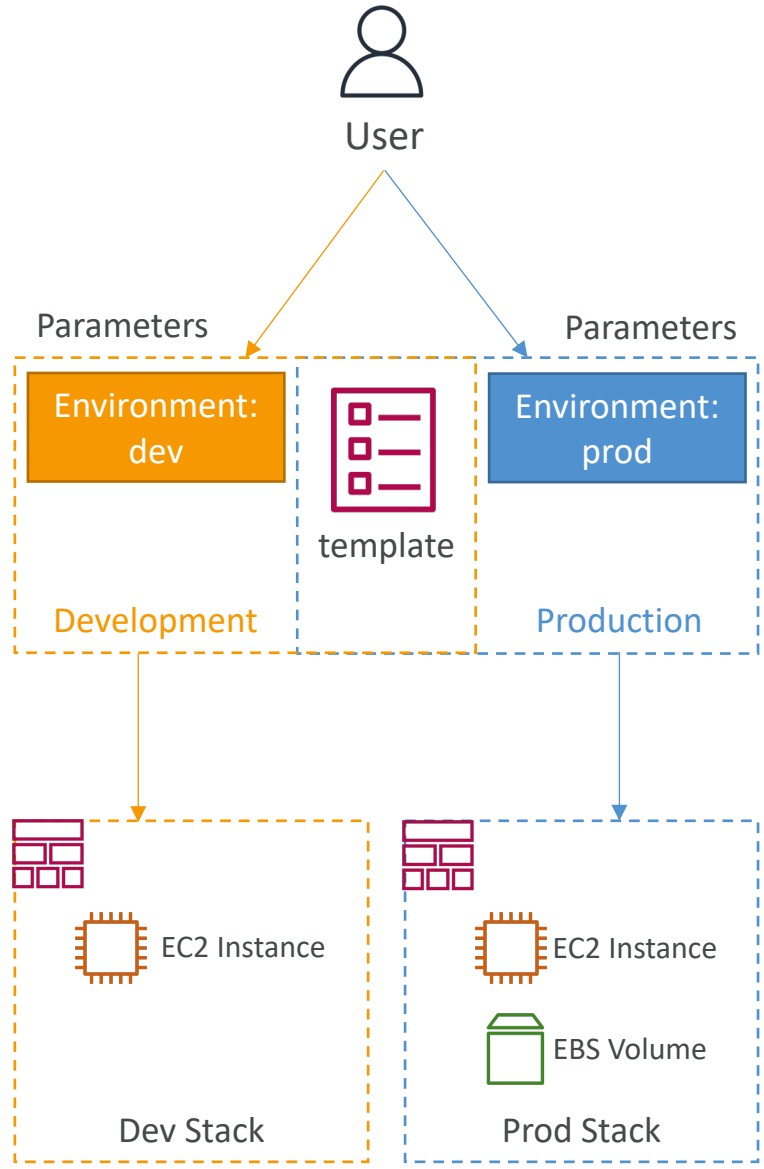
CloudFormation – Outputs Cross-Stack Reference

- We then create a second template that leverages that security group
- For this, we use the Fn::ImportValue function
- You can't delete the underlying stack until all the references are deleted

```
Resources:
  MySecureInstance:
    Type: AWS::EC2::Instance
    Properties:
      ImageId: ami-0742b4e673072066f
      InstanceType: t2.micro
      AvailabilityZone: us-east-1a
      SecurityGroups:
        - !ImportValue SSHSecurityGroup
```

CloudFormation – Conditions

- Conditions are used to control the creation of resources or outputs based on a condition
- Conditions can be whatever you want them to be, but common ones are:
  - Environment (dev / test / prod)
  - AWS Region
  - Any parameter value
- Each condition can reference another condition, parameter value or mapping



How to define a Condition

```
Conditions:
  CreateProdResources: !Equals [ !Ref EnvType, prod ]
```

- The logical ID is for you to choose. It's how you name condition
- The intrinsic function (logical) can be any of the following:
  - Fn::And
  - Fn::Equals
  - Fn::If
  - Fn::Not
  - Fn::Or

How to use a Condition

- Conditions can be applied to resources / outputs / etc…

```
Resources:
  MountPoint:
    Type: AWS::EC2::VolumeAttachment
    Condition: CreateProdResources
```

CloudFormation – Intrinsic Functions

Blue = must know

- Ref
- Fn::GetAtt
- Fn::FindInMap
- Fn::ImportValue
- Fn::Join
- Fn::Sub
- Fn::ForEach
- Fn::ToJsonString
- Fn::Base64
- Fn::Cidr
- Fn::GetAZs
- Fn::Select
- Fn::Split
- Fn::Transform
- Fn::Length
- Condition Functions (Fn::If, Fn::Not, Fn::Equals, etc…)

Intrinsic Functions – Fn::Ref

- The Fn::Ref function can be leveraged to reference
  - **Parameters** – returns the value of the parameter
  - **Resources** – returns the physical ID of the underlying resource (e.g., EC2 ID)
- The shorthand for this in YAML is !Ref

```
Resources:
  DBSubnet1:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref MyVPC
```

Intrinsic Functions – Fn::GetAtt

- Attributes are attached to any resources you create
- To know the attributes of your resources, the best place to look at is the documentation
- Example: the AZ of an EC2 instance!

```
Resources:
  EC2Instance:
    Type: AWS::EC2::Instance
    Properties:
      ImageId: ami-0742b4e673072066f
      InstanceType: t2.micro
```

```
EBSVolume:
  Type: AWS::EC2::Volume
  Condition: CreateProdResources
  Properties:
    Size: 100
    AvailabilityZone: !GetAtt EC2Instance.AvailabilityZone
```

Intrinsic Functions – Fn::FindInMap

- We use Fn::FindInMap to return a named value from a specific key
- !FindInMap [MapName, TopLevelKey, SecondLevelKey]

```
Mappings:
  RegionMap:
    us-east-1:
      HVM64: ami-0ff8a91507f77f867
      HVMG2: ami-0a584ac55a7631c0c
    us-west-1:
      HVM64: ami-0bdb828fd58c52235
      HVMG2: ami-066ee5fd4a9ef77f1

Resources:
  MyEC2Instance:
    Type: AWS::EC2::Instance
    Properties:
      ImageId: !FindInMap [RegionMap, !Ref "AWS::Region", HVM64]
      InstanceType: t2.micro
```

Intrinsic Functions – Fn::ImportValue

- Import values that are exported in other stacks
- For this, we use the Fn::ImportValue function

```
Resources:
  MySecureInstance:
    Type: AWS::EC2::Instance
    Properties:
      ImageId: ami-0742b4e673072066f
      InstanceType: t2.micro
      AvailabilityZone: us-east-1a
      SecurityGroups:
        - !ImportValue SSHSecurityGroup
```

Intrinsic Functions – Fn::Base64

- Convert **String** to it's **Base64** representation

```
!Base64 "ValueToEncode"
```

- Example: pass encoded data to EC2 Instance's **UserData** property

```
Resources:
  WebServer:
    Type: AWS::EC2::Instance
    Properties:
      ...
      UserData:
        Fn::Base64: |
          #!/bin/bash
          dnf update -y
          dnf install -y httpd
```

Intrinsic Functions – Condition Functions

```
Conditions:
  CreateProdResources: !Equals [ !Ref EnvType, prod ]
```

- The logical ID is for you to choose. It's how you name condition
- The intrinsic function (logical) can be any of the following:
  - Fn::And
  - Fn::Equals
  - Fn::If
  - Fn::Not
  - Fn::Or

CloudFormation – Rollbacks

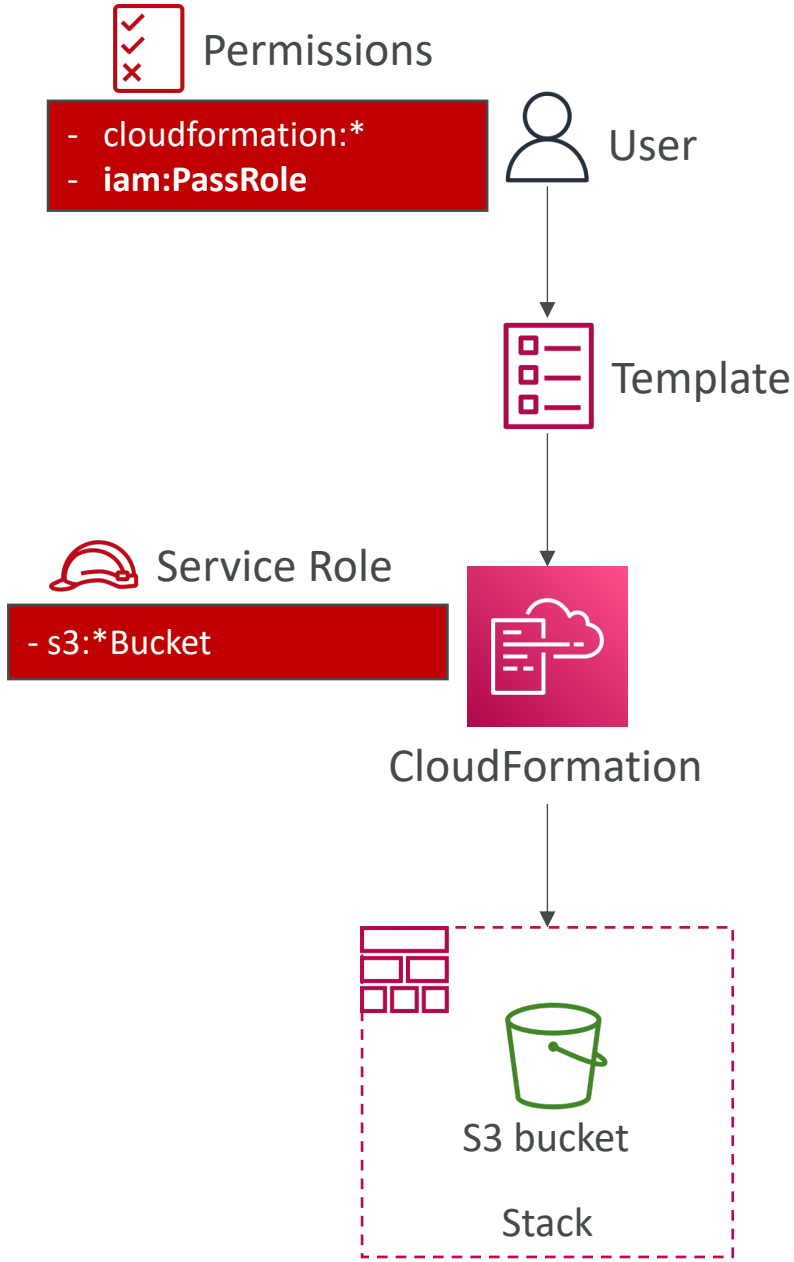
- Stack Creation Fails:
  - Default: everything rolls back (gets deleted). We can look at the log
  - Option to disable rollback and troubleshoot what happened

- Stack Update Fails:
  - The stack automatically rolls back to the previous known working state
  - Ability to see in the log what happened and error messages

- Rollback Failure? Fix resources manually then issue **ContinueUpdateRollback API** from Console
  - Or from the CLI using continue-update-rollback API call

CloudFormation – Service Role

- IAM role that allows CloudFormation to create/update/delete stack resources on your behalf
- Give ability to users to create/update/delete the stack resources even if they don't have permissions to work with the resources in the stack
- Use cases:
  - You want to achieve the least privilege principle
  - But you don't want to give the user all the required permissions to create the stack resources
- User must have **iam:PassRole** permissions



CloudFormation Capabilities

- CAPABILITY_NAMED_IAM and CAPABILITY_IAM
  - Necessary to enable when you CloudFormation template is creating or updating IAM resources (IAM User, Role, Group, Policy, Access Keys, Instance Profile…)
  - Specify **CAPABILITY_NAMED_IAM** if the resources are named
- CAPABILITY_AUTO_EXPAND
  - Necessary when your CloudFormation template includes Macros or Nested Stacks (stacks within stacks) to perform dynamic transformations
  - You're acknowledging that your template may change before deploying
- InsufficientCapabilitiesException
  - Exception that will be thrown by CloudFormation if the capabilities haven't been acknowledged when deploying a template (security measure)

CloudFormation – DeletionPolicy Delete

- DeletionPolicy:
  - Control what happens when the CloudFormation template is deleted or when a resource is removed from a CloudFormation template
  - Extra safety measure to preserve and backup resources
- Default **DeletionPolicy=Delete**
  - ⚠ **Delete** won't work on an S3 bucket if the bucket is not empty

```yaml
Resources:
  MyEC2Instance:
    Type: AWS::EC2::Instance
    Properties:
      ImageId: ami-12345678
      InstanceType: t2.micro
      KeyName: my-key-pair
      SecurityGroupIds:
        - sg-12345678
    DeletionPolicy: Delete
```

```yaml
Resources:
  MyS3Bucket:
    Type: AWS::S3::Bucket
    DeletionPolicy: Delete
```

⚠

CloudFormation – DeletionPolicy Retain

- DeletionPolicy=Retain:
  - Specify on resources to preserve in case of CloudFormation deletes
- Works with any resources

```
Resources:
  MyDynamoDBTable:
    Type: AWS::DynamoDB::Table
    Properties:
      TableName: MyTable
      AttributeDefinitions:
        - AttributeName: ID
          AttributeType: S
      KeySchema:
        - AttributeName: ID
          KeyType: HASH
      ProvisionedThroughput:
        ReadCapacityUnits: 5
        WriteCapacityUnits: 5
    DeletionPolicy: Retain
```

CloudFormation – DeletionPolicy Snapshot

- DeletionPolicy=Snapshot
- Create one final snapshot before deleting the resource
- Examples of supported resources:
  - EBS Volume, ElastiCache Cluster, ElastiCache ReplicationGroup
  - RDS DBInstance, RDS DBCluster, Redshift Cluster, Neptune DBCluster, DocumentDB DBCluster

```yaml
Resources:
  MyDBInstance:
    Type: AWS::RDS::DBInstance
    Properties:
      DBInstanceClass: db.t2.micro
      AllocatedStorage: 20
      Engine: mysql
      MasterUsername: admin
      MasterUserPassword: "ExamplePassword"
    DeletionPolicy: Snapshot
```

CloudFormation – Stack Policies

- During a CloudFormation Stack update, all update actions are allowed on all resources (default)

- A Stack Policy is a JSON document that defines the update actions that are allowed on specific resources during Stack updates
- Protect resources from unintentional updates
- When you set a Stack Policy, all resources in the Stack are protected by default
- Specify an explicit ALLOW for the resources you want to be allowed to be updated

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "Update:*",
            "Principal": "*",
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": "Update:*",
            "Principal": "*",
            "Resource": "LogicalResourceId/ProductionDatabase"
        }
    ]
}
```

Allow updates on all resources **except** the ProductionDatabase

CloudFormation – Termination Protection

- To prevent accidental deletes of CloudFormation Stacks, use **TerminationProtection**

CloudFormation – Custom Resources

- Used to
  - define resources not yet supported by CloudFormation
  - define custom provisioning logic for resources can that be outside of CloudFormation (on-premises resources, 3rd party resources…)
  - have custom scripts run during create / update / delete through Lambda functions (running a Lambda function to empty an S3 bucket before being deleted)
- Defined in the template using AWS::CloudFormation::CustomResource or Custom::*MyCustomResourceTypeName* (recommended)
- Backed by a Lambda function (most common) or an SNS topic

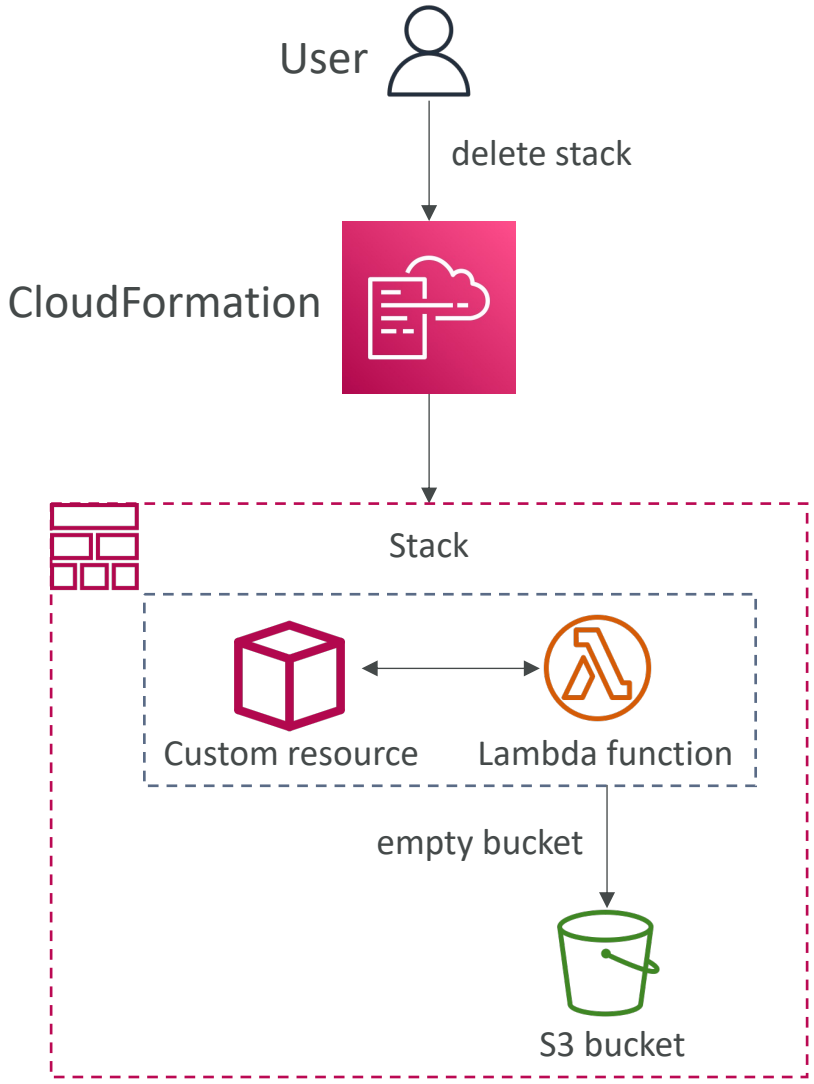
How to define a Custom Resource?

- **ServiceToken** specifies where CloudFormation sends requests to, such as Lambda ARN or SNS ARN (required & must be in the same region)
- Input data parameters (optional)

```yaml
Resources:
  MyCustomResourceUsingLambda:
    Type: Custom::MyLambdaResource
    Properties:
      ServiceToken: arn:aws:lambda:REGION:ACCOUNT_ID:function:FUNCTION_NAME
      # Input values (optional)
      ExampleProperty: "ExampleValue"
```

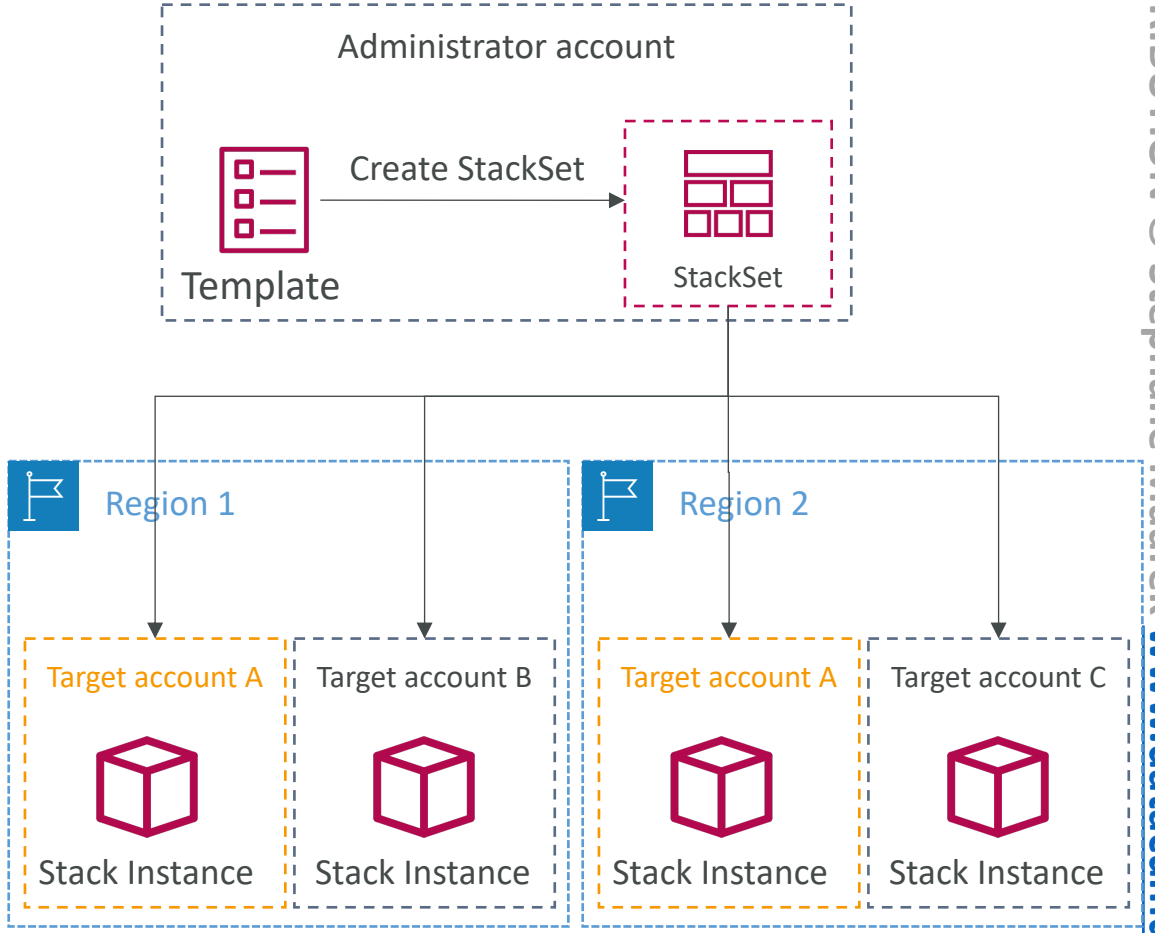
Use Case – Delete content from an S3 bucket

- You can't delete a non-empty S3 bucket
- To delete a non-empty S3 bucket, you must first delete all the objects inside it
- We can use a custom resource to empty an S3 bucket before it gets deleted by CloudFormation



CloudFormation – StackSets

- Create, update, or delete stacks across **multiple accounts and regions** with a single operation/template
- Target accounts to create, update, delete stack instances from StackSets
- When you update a stack set, *all* associated stack instances are updated throughout all accounts and regions
- **Can be applied into all accounts of an AWS Organization**
- Only Administrator account (or Delegated Administrator) can create StackSets



AWS Integration & Messaging

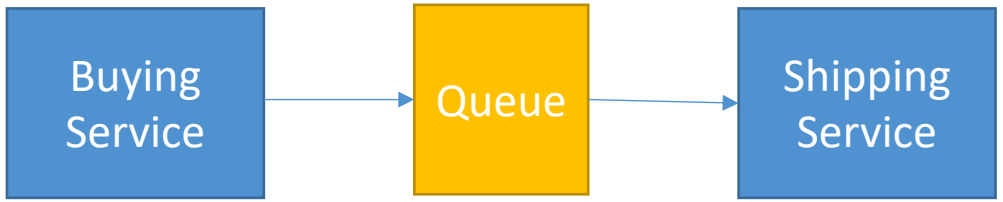
SQS, SNS & Kinesis

Section Introduction

- When we start deploying multiple applications, they will inevitably need to communicate with one another
- There are two patterns of application communication

1) Synchronous communications (application to application)

2) Asynchronous / Event based (application to queue to application)

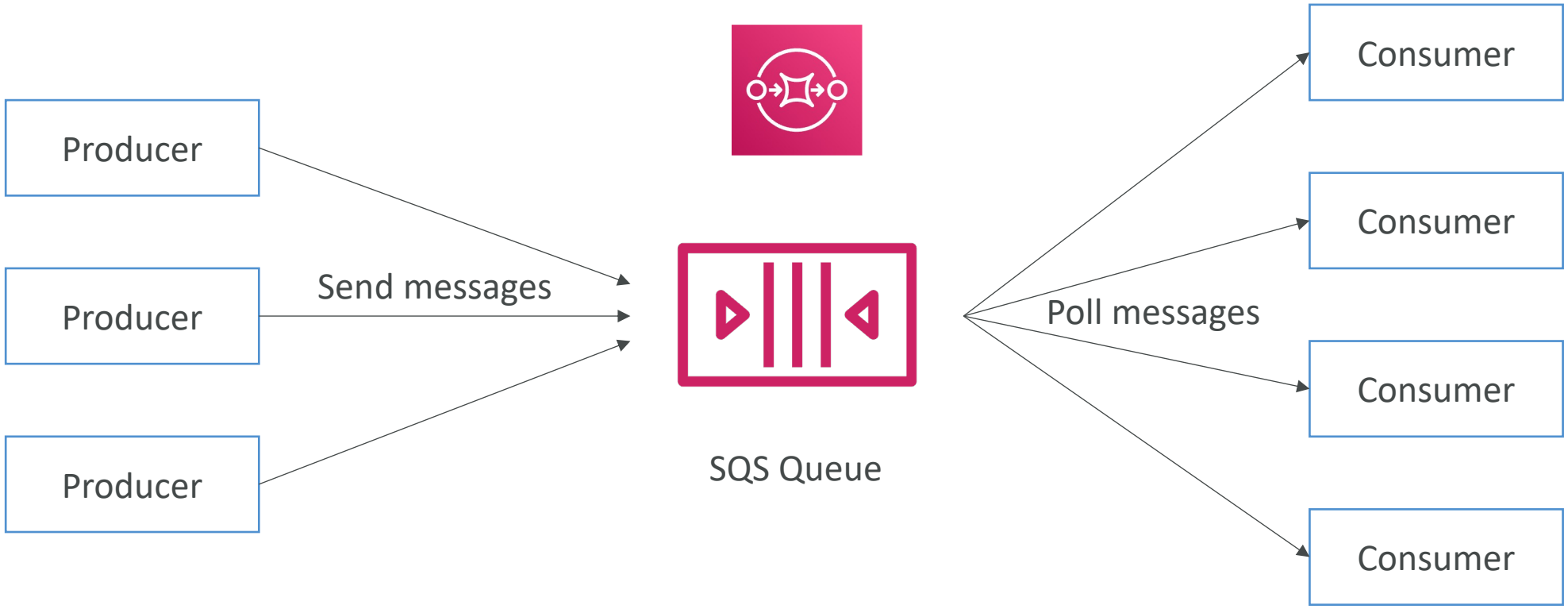


Section Introduction

- Synchronous between applications can be problematic if there are sudden spikes of traffic
- What if you need to suddenly encode 1000 videos but usually it's 10?

- In that case, it's better to **decouple** your applications,
  - using SQS: queue model
  - using SNS: pub/sub model
  - using Kinesis: real-time streaming model
- These services can scale independently from our application!

Amazon SQS
What's a queue?



Amazon SQS – Standard Queue

- Oldest offering (over 10 years old)
- Fully managed service, used to **decouple applications**

- Attributes:
  - Unlimited throughput, unlimited number of messages in queue
  - Default retention of messages: 4 days, maximum of 14 days
  - Low latency (<10 ms on publish and receive)
  - Limitation of 256KB per message sent

- Can have duplicate messages (at least once delivery, occasionally)
- Can have out of order messages (best effort ordering)

SQS – Producing Messages

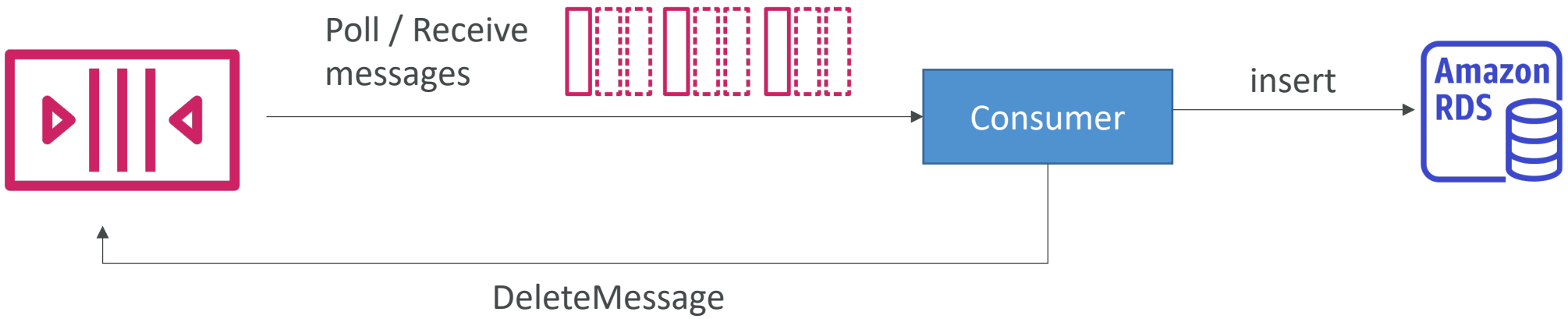
- Produced to SQS using the SDK (SendMessage API)
- The message is **persisted** in SQS until a consumer deletes it
- Message retention: default 4 days, up to 14 days

- Example: send an order to be processed
  - Order id
  - Customer id
  - Any attributes you want

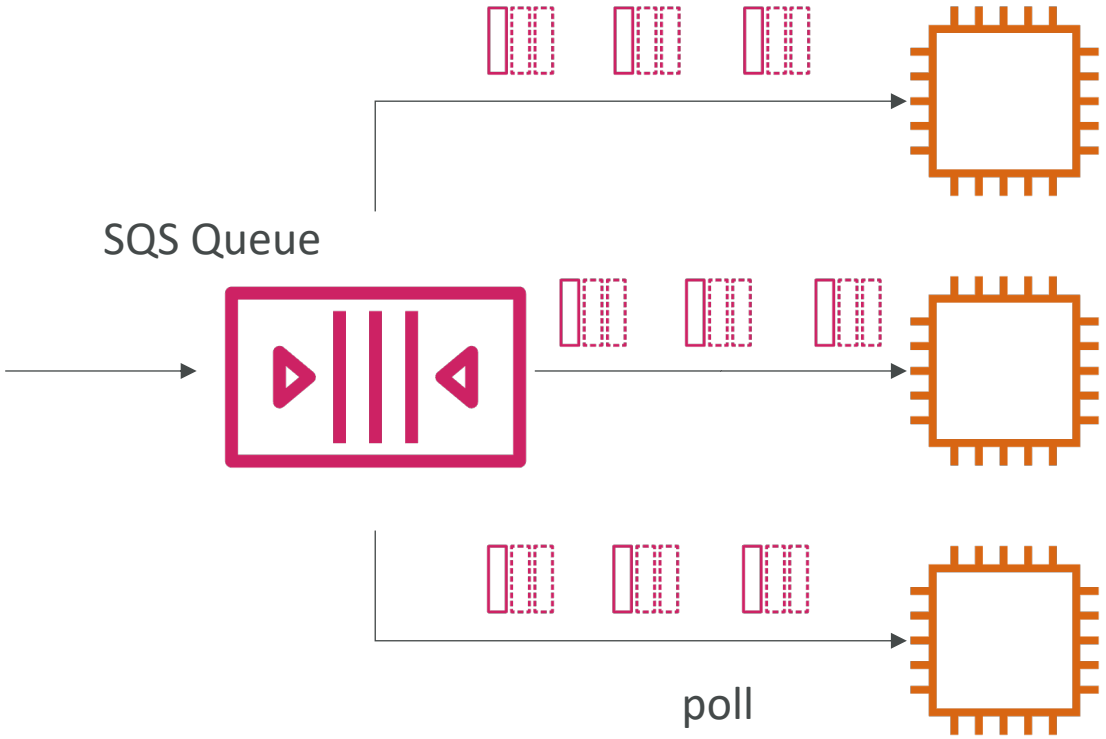
- SQS standard: unlimited throughput

SQS – Consuming Messages

- Consumers (running on EC2 instances, servers, or AWS Lambda)…
- Poll SQS for messages (receive up to 10 messages at a time)
- Process the messages (example: insert the message into an RDS database)
- Delete the messages using the DeleteMessage API

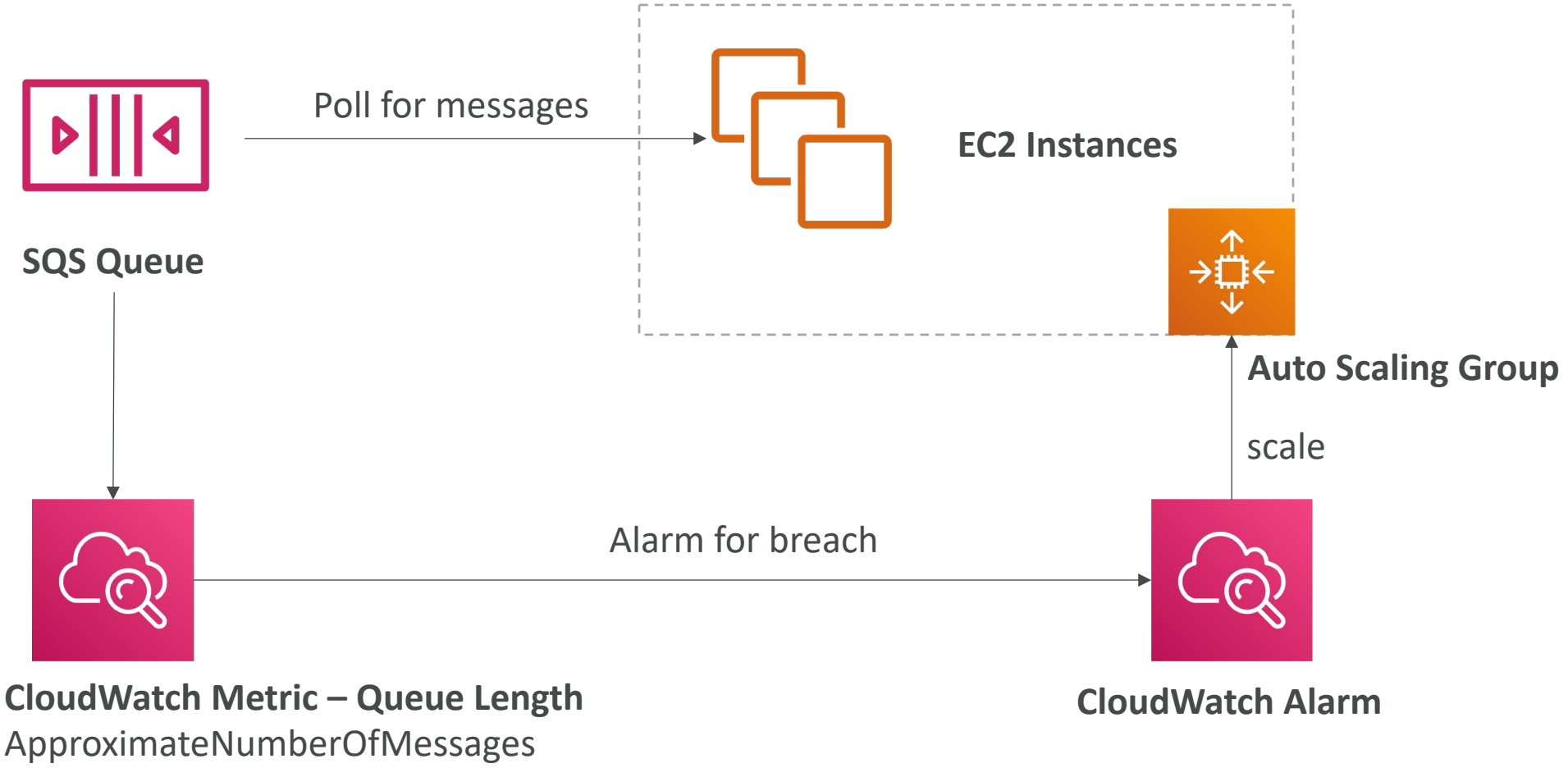


SQS – Multiple EC2 Instances Consumers

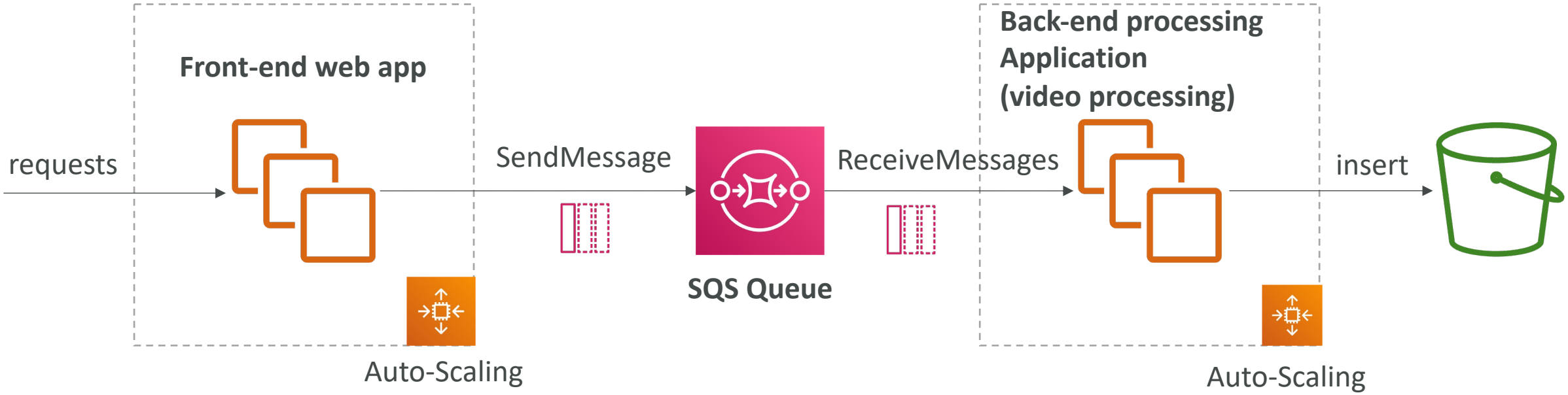


- Consumers receive and process messages in parallel
- At least once delivery
- Best-effort message ordering
- Consumers delete messages after processing them
- We can scale consumers horizontally to improve throughput of processing

SQS with Auto Scaling Group (ASG)



SQS to **decouple** between application tiers

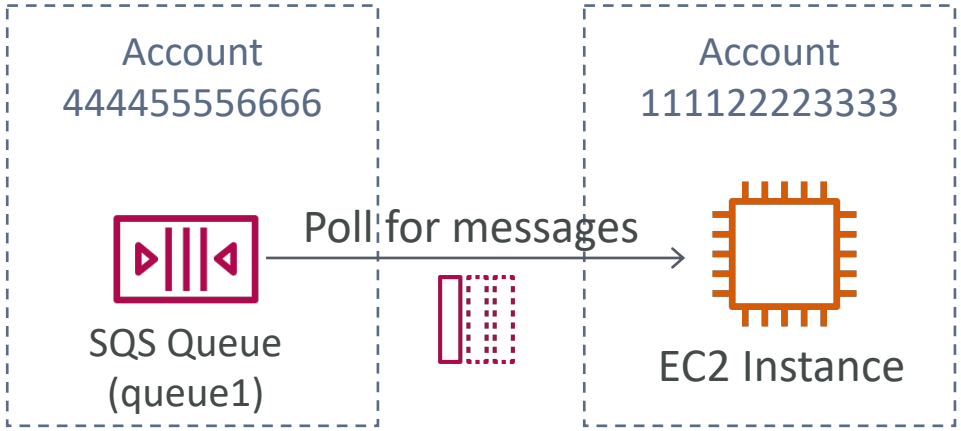


Amazon SQS - Security

- Encryption:
  - In-flight encryption using HTTPS API
  - At-rest encryption using KMS keys
  - Client-side encryption if the client wants to perform encryption/decryption itself
- Access Controls: IAM policies to regulate access to the SQS API
- SQS Access Policies (similar to S3 bucket policies)
  - Useful for cross-account access to SQS queues
  - Useful for allowing other services (SNS, S3…) to write to an SQS queue

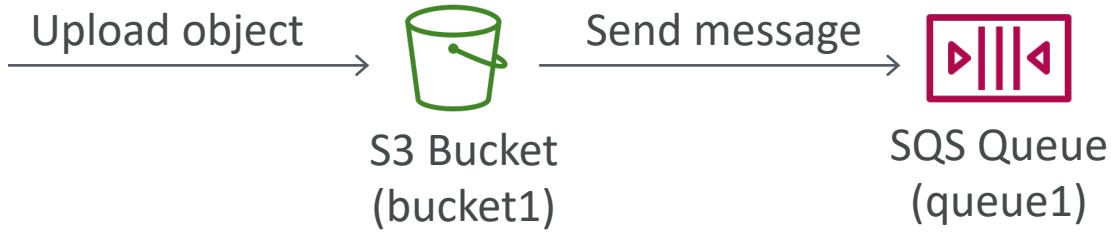
SQS Queue Access Policy

Cross Account Access




```
{
   "Version": "2012-10-17",
   "Statement" : [{
      "Effect": "Allow",
      "Principal": { "AWS": [ "111122223333" ] },
      "Action": [ "sqs:ReceiveMessage" ],
      "Resource": "arn:aws:sqs:us-east-1:444455556666:queue1"
   }]
}
```

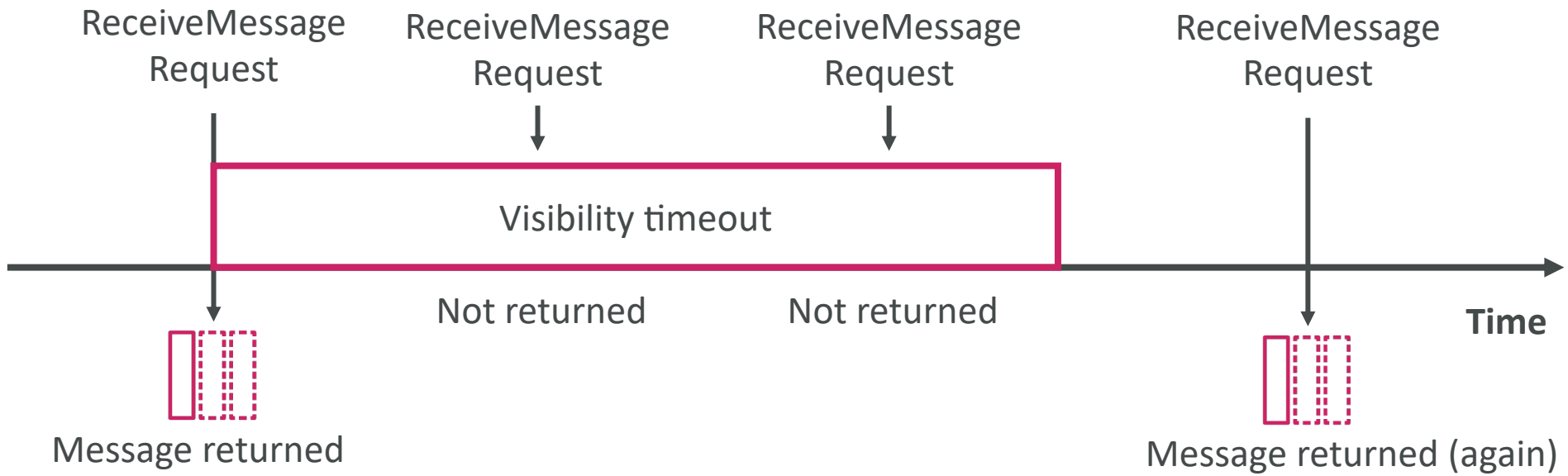
Publish S3 Event Notifications To SQS Queue



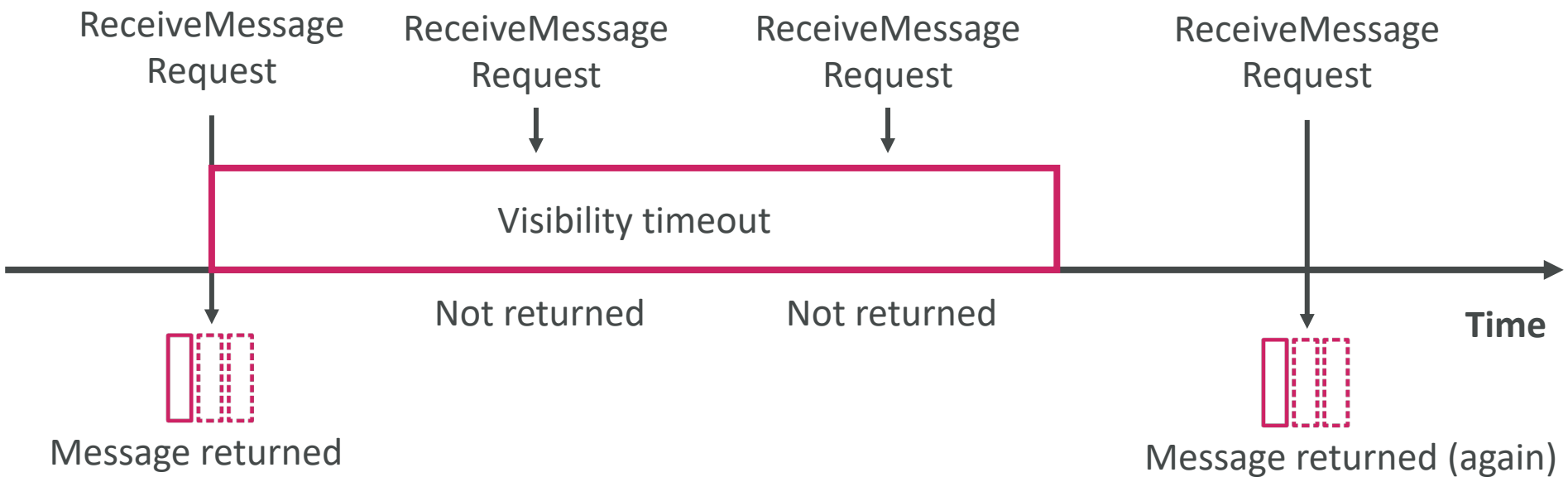

```
{
   "Version": "2012-10-17",
   "Statement" : [{
      "Effect": "Allow",
      "Principal": { "AWS": "*"},
      "Action": [ "sqs:SendMessage" ],
      "Resource": "arn:aws:sqs:us-east-1:444455556666:queue1",
      "Condition": {
         "ArnLike": { "aws:SourceArn": "arn:aws:s3:*:*:bucket1" },
         "StringEquals": { "aws:SourceAccount": "<bucket1_owner_account_id>" },
      }
   }]
}
```

SQS – Message Visibility Timeout

- After a message is polled by a consumer, it becomes **invisible** to other consumers
- By default, the "message visibility timeout" is **30 seconds**
- That means the message has 30 seconds to be processed
- After the message visibility timeout is over, the message is "visible" in SQS



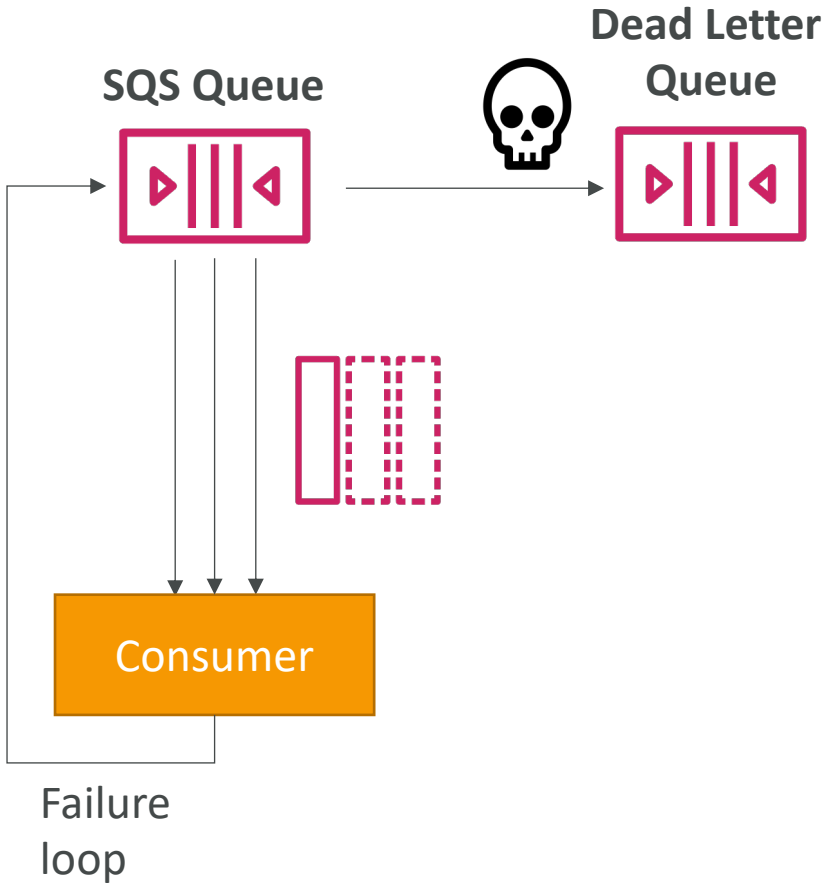
SQS – Message Visibility Timeout



- If a message is not processed within the visibility timeout, it will be processed **twice**
- A consumer could call the **ChangeMessageVisibility** API to get more time
- If visibility timeout is high (hours), and consumer crashes, re-processing will take time
- If visibility timeout is too low (seconds), we may get duplicates

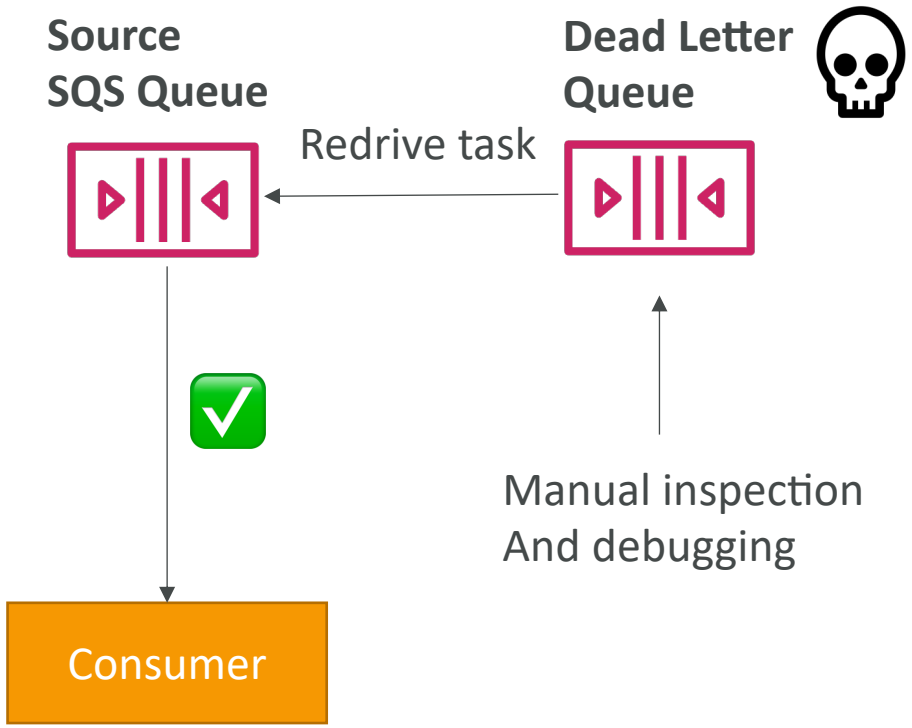
Amazon SQS – Dead Letter Queue (DLQ)

- If a consumer fails to process a message within the Visibility Timeout…
  the message goes back to the queue!
- We can set a threshold of how many times a message can go back to the queue
- After the **MaximumReceives** threshold is exceeded, the message goes into a Dead Letter Queue (DLQ)
- Useful for debugging!
- **DLQ of a FIFO queue must also be a FIFO queue**
- **DLQ of a Standard queue must also be a Standard queue**
- Make sure to process the messages in the DLQ before they expire:
  - Good to set a retention of 14 days in the DLQ



SQS DLQ – Redrive to Source

- Feature to help consume messages in the DLQ to understand what is wrong with them
- When our code is fixed, we can redrive the messages from the DLQ back into the source queue (or any other queue) in batches without writing custom code

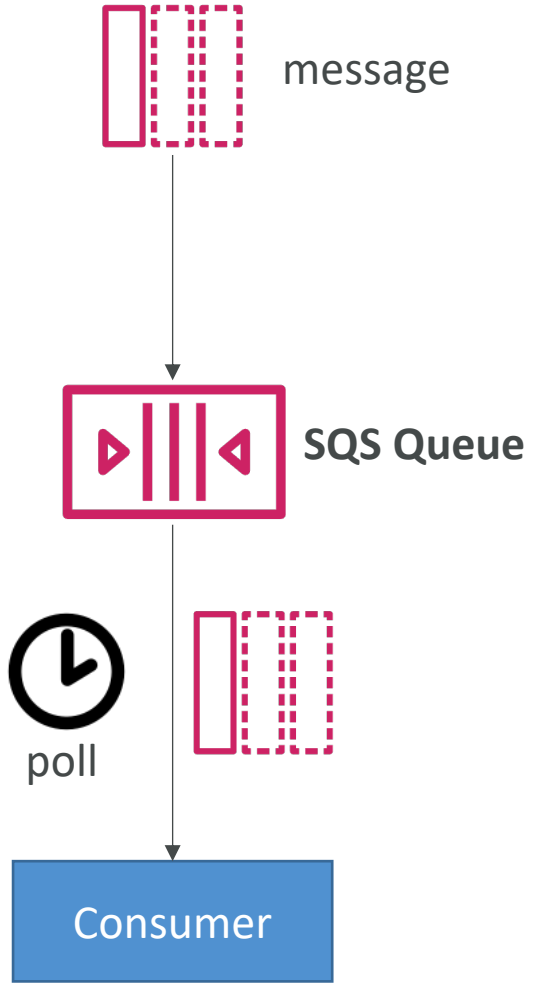


Amazon SQS – Delay Queue

- Delay a message (consumers don't see it immediately) up to 15 minutes
- Default is 0 seconds (message is available right away)
- Can set a default at queue level
- Can override the default on send using the **DelaySeconds** parameter

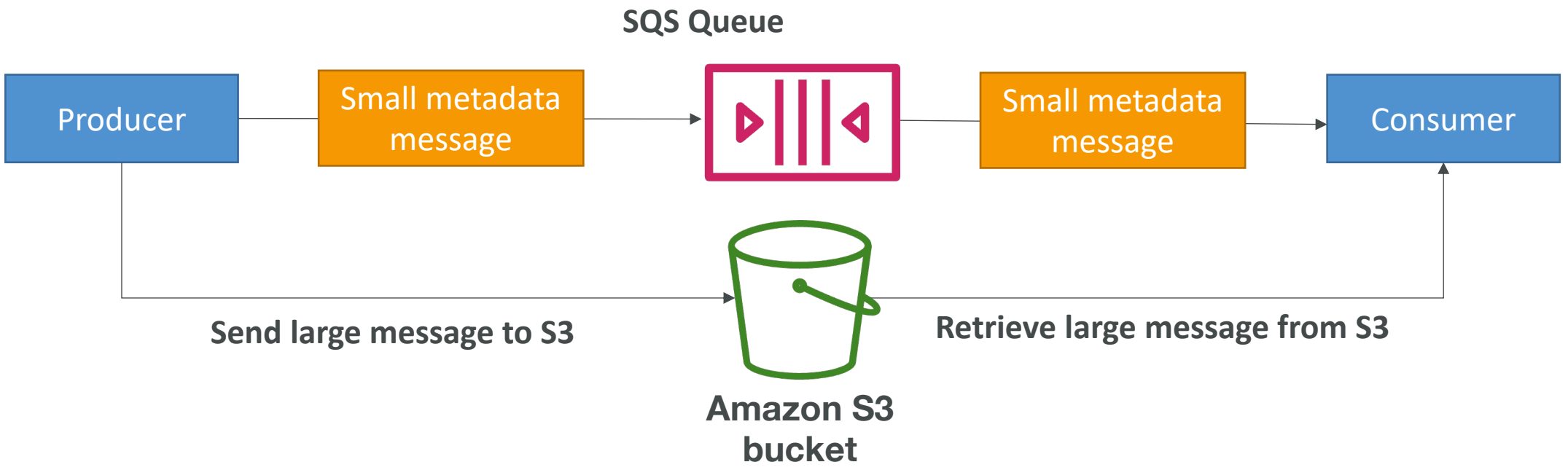
Amazon SQS - Long Polling

- When a consumer requests messages from the queue, it can optionally "wait" for messages to arrive if there are none in the queue
- This is called Long Polling
- **LongPolling decreases the number of API calls made to SQS while increasing the efficiency and decreasing the latency of your application.**
- The wait time can be between 1 sec to 20 sec (20 sec preferable)
- Long Polling is preferable to Short Polling
- Long polling can be enabled at the queue level or at the API level using **ReceiveMessageWaitTimeSeconds**



SQS Extended Client

- Message size limit is 256KB, how to send large messages, e.g. 1GB?
- Using the SQS Extended Client (Java Library)



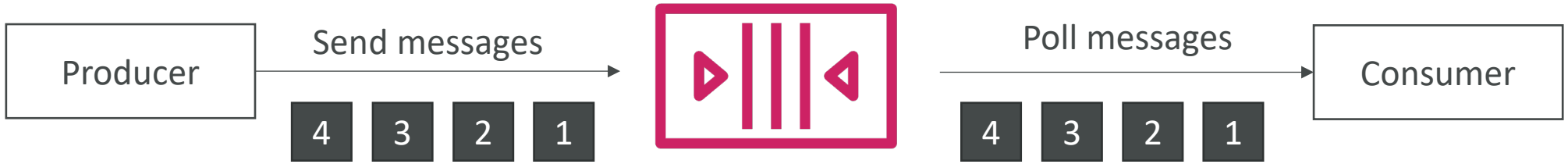
SQS – Must know API

- CreateQueue (MessageRetentionPeriod), DeleteQueue
- PurgeQueue: delete all the messages in queue
- SendMessage (DelaySeconds), ReceiveMessage, DeleteMessage
- MaxNumberOfMessages: default 1, max 10 (for ReceiveMessage API)
- ReceiveMessageWaitTimeSeconds: Long Polling
- ChangeMessageVisibility: change the message timeout

- Batch APIs for SendMessage, DeleteMessage, ChangeMessageVisibility helps decrease your costs

Amazon SQS – FIFO Queue

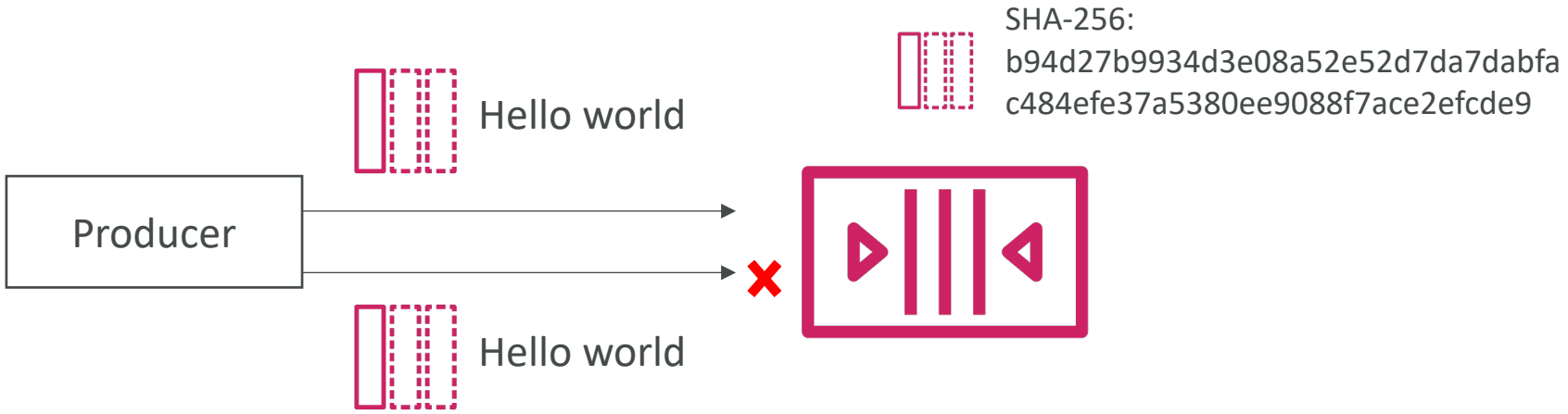
- FIFO = First In First Out (ordering of messages in the queue)



- Limited throughput: 300 msg/s without batching, 3000 msg/s with
- Exactly-once send capability (by removing duplicates)
- Messages are processed in order by the consumer

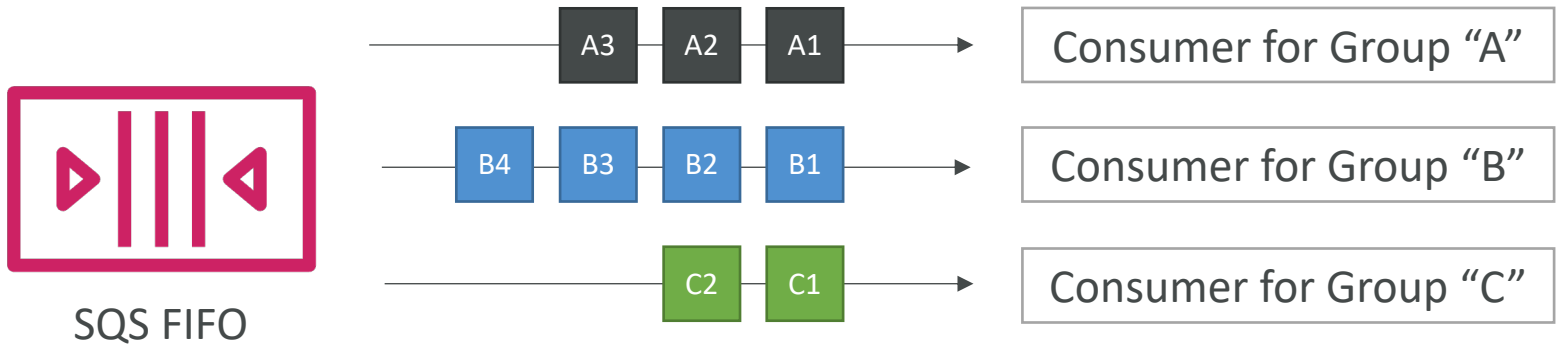
SQS FIFO – Deduplication

- De-duplication interval is 5 minutes
- Two de-duplication methods:
  - Content-based deduplication: will do a SHA-256 hash of the message body
  - Explicitly provide a Message Deduplication ID



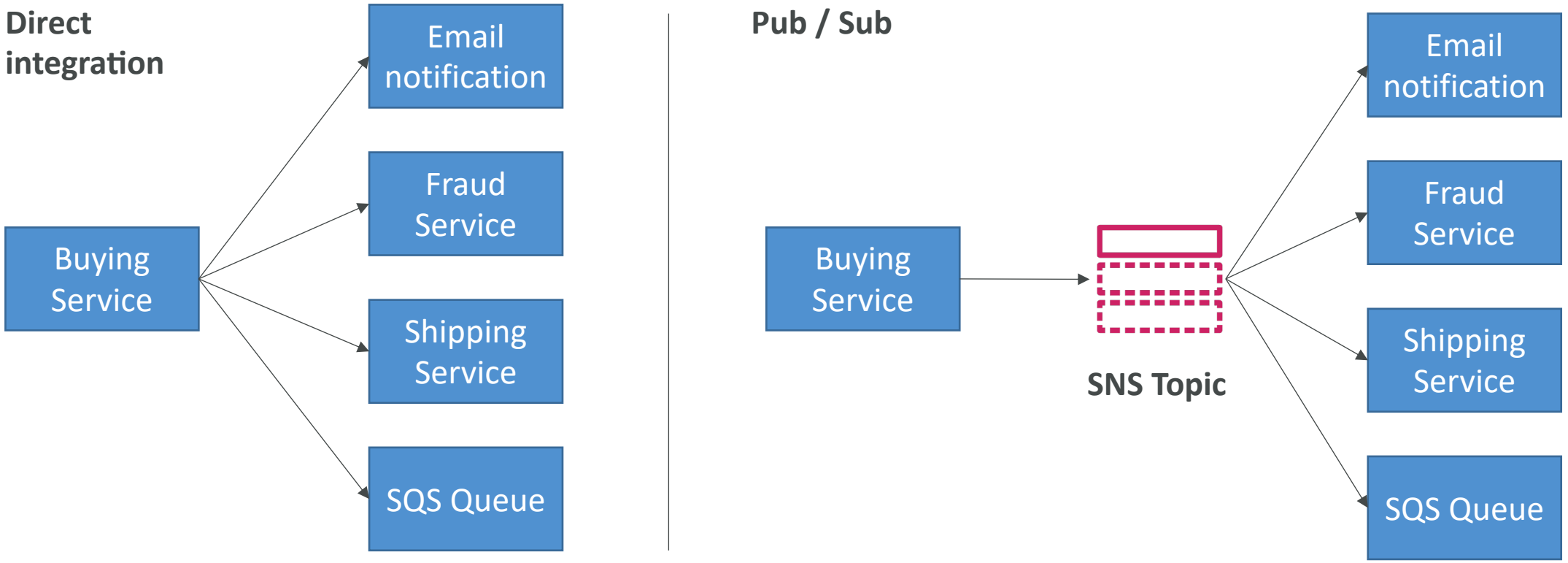
SQS FIFO – Message Grouping

- If you specify the same value of **MessageGroupID** in an SQS FIFO queue, you can only have one consumer, and all the messages are in order
- To get ordering at the level of a subset of messages, specify different values for **MessageGroupID**
  - Messages that share a common Message Group ID will be in order within the group
  - Each Group ID can have a different consumer (parallel processing!)
  - Ordering across groups is not guaranteed



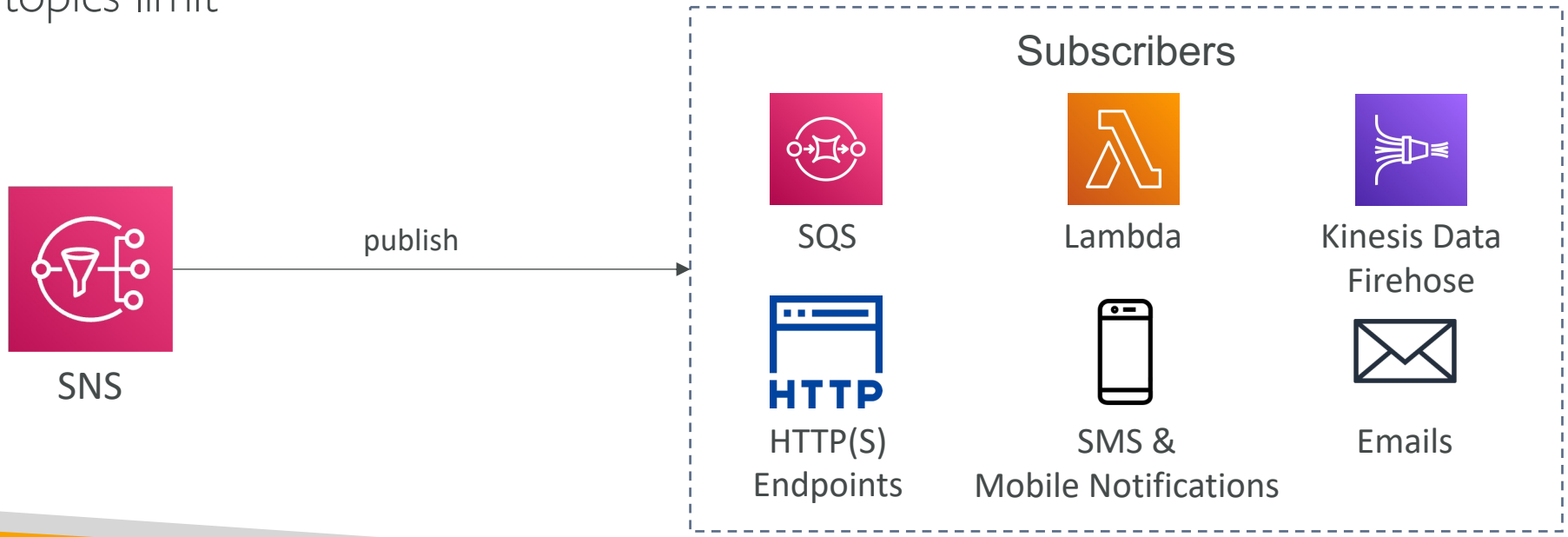
Amazon SNS

- What if you want to send one message to many receivers?



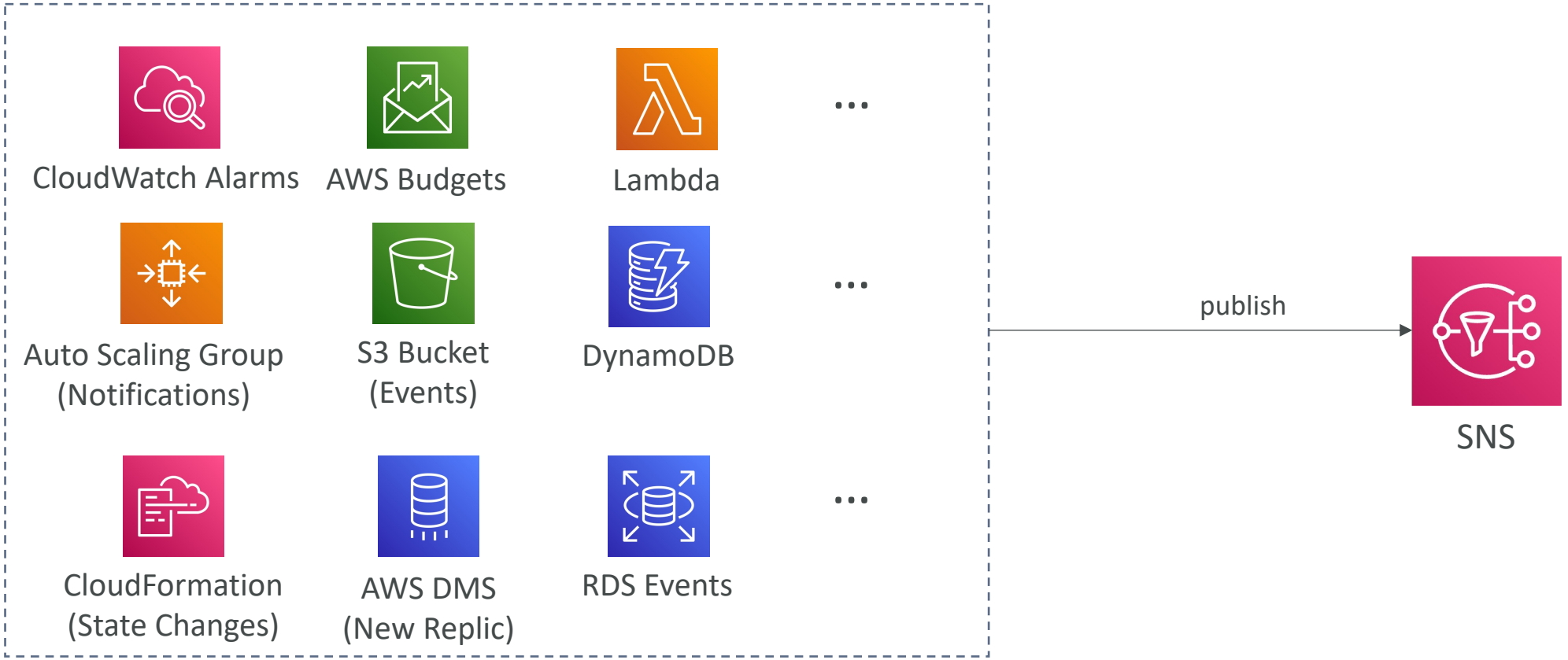
Amazon SNS

- The "event producer" only sends message to one SNS topic
- As many "event receivers" (subscriptions) as we want to listen to the SNS topic notifications
- Each subscriber to the topic will get all the messages (note: new feature to filter messages)
- Up to 12,500,000 subscriptions per topic
- 100,000 topics limit



SNS integrates with a lot of AWS services

- Many AWS services can send data directly to SNS for notifications



Amazon SNS – How to publish

- Topic Publish (using the SDK)
  - Create a topic
  - Create a subscription (or many)
  - Publish to the topic

- Direct Publish (for mobile apps SDK)
  - Create a platform application
  - Create a platform endpoint
  - Publish to the platform endpoint
  - Works with Google GCM, Apple APNS, Amazon ADM…

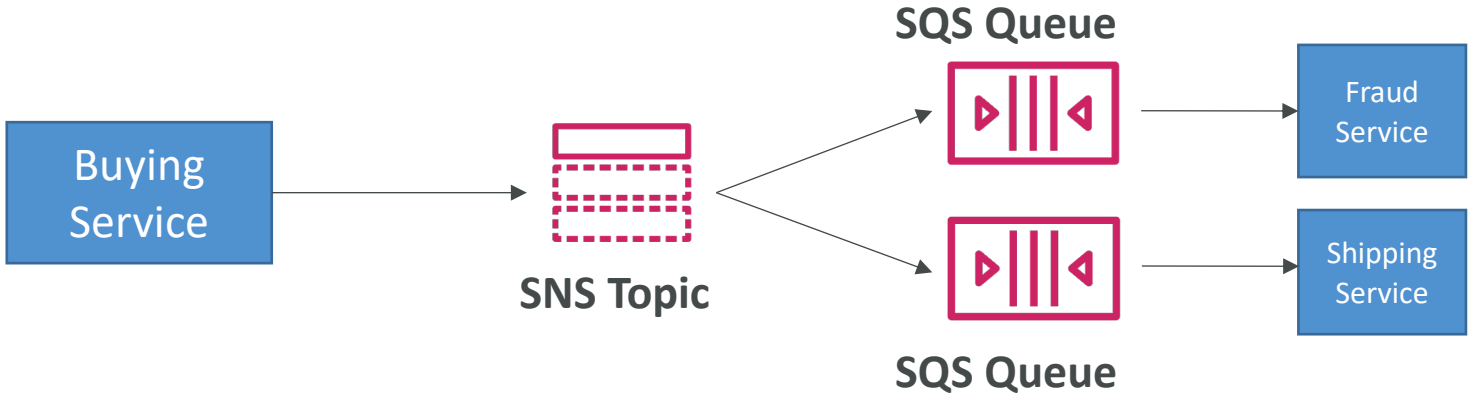
Amazon SNS – Security

- Encryption:
  - In-flight encryption using HTTPS API
  - At-rest encryption using KMS keys
  - Client-side encryption if the client wants to perform encryption/decryption itself

- Access Controls: IAM policies to regulate access to the SNS API

- SNS Access Policies (similar to S3 bucket policies)
  - Useful for cross-account access to SNS topics
  - Useful for allowing other services (S3…) to write to an SNS topic

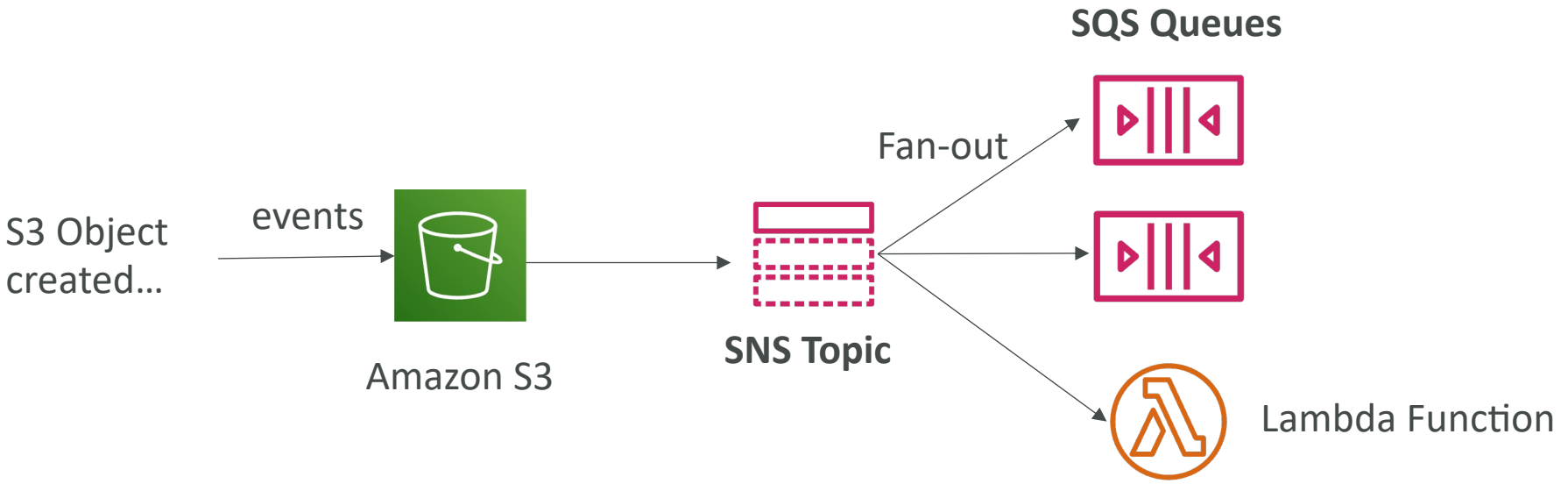
SNS + SQS: Fan Out



- Push once in SNS, receive in all SQS queues that are subscribers
- Fully decoupled, no data loss
- SQS allows for: data persistence, delayed processing and retries of work
- Ability to add more SQS subscribers over time
- Make sure your SQS queue **access policy** allows for SNS to write
- Cross-Region Delivery: works with SQS Queues in other regions

Application: S3 Events to multiple queues

- For the same combination of: **event type** (e.g. object create) and **prefix** (e.g. images/) you can only have one S3 Event rule
- If you want to send the same S3 event to many SQS queues, use fan-out

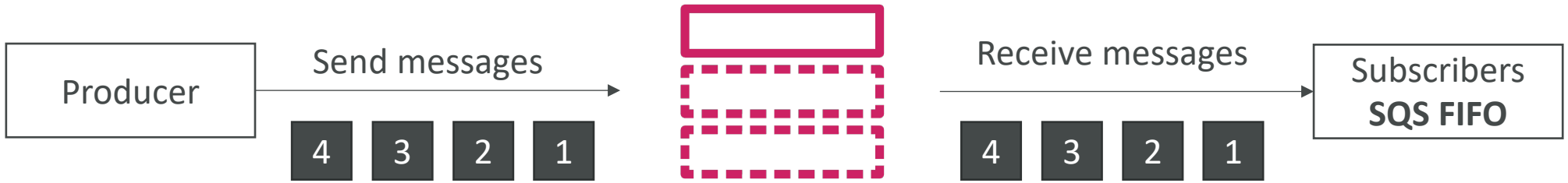


Application: SNS to Amazon S3 through Kinesis Data Firehose

- SNS can send to Kinesis and therefore we can have the following solutions architecture:

Amazon SNS – FIFO Topic

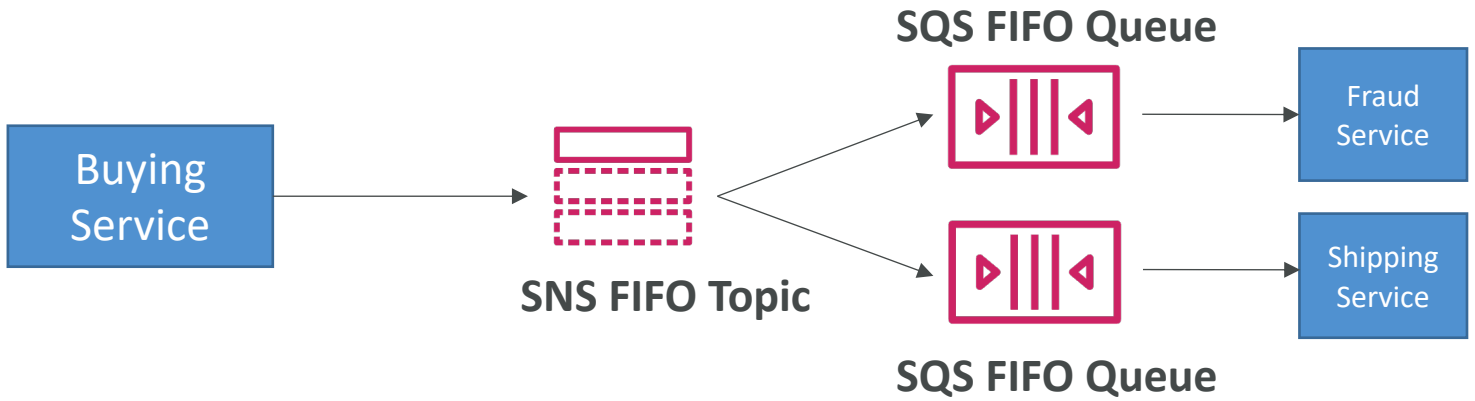
- FIFO = First In First Out (ordering of messages in the topic)



- Similar features as SQS FIFO:
  - **Ordering** by Message Group ID (all messages in the same group are ordered)
  - **Deduplication** using a Deduplication ID or Content Based Deduplication
- **Can have SQS Standard and FIFO queues as subscribers**
- Limited throughput (same throughput as SQS FIFO)

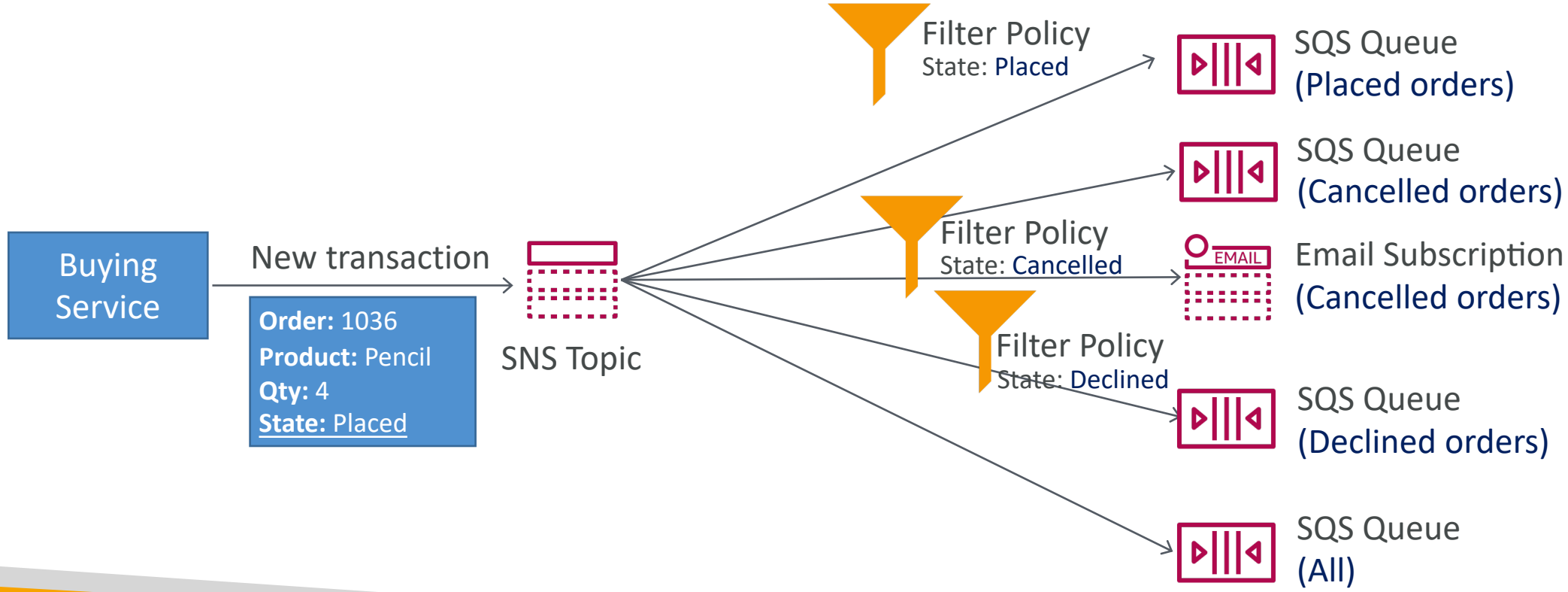
SNS FIFO + SQS FIFO: Fan Out

- In case you need fan out + ordering + deduplication



SNS – Message Filtering

- JSON policy used to filter messages sent to SNS topic's subscriptions
- If a subscription doesn't have a filter policy, it receives every message



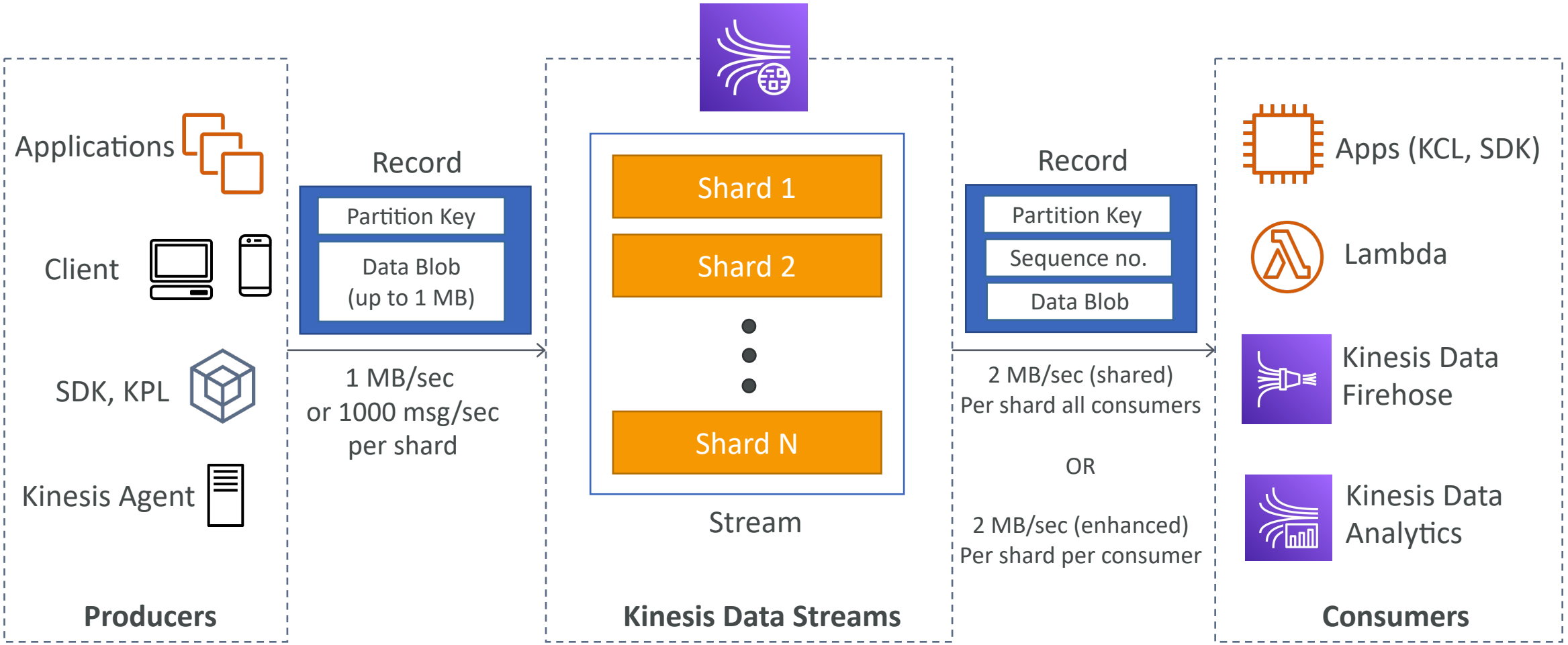
Kinesis Overview

- Makes it easy to **collect**, **process**, and **analyze** streaming data in real-time
- Ingest real-time data such as: Application logs, Metrics, Website clickstreams, IoT telemetry data…




- **Kinesis Data Streams:** capture, process, and store data streams
- **Kinesis Data Firehose:** load data streams into AWS data stores
- **Kinesis Data Analytics:** analyze data streams with SQL or Apache Flink
- **Kinesis Video Streams:** capture, process, and store video streams

Kinesis Data Streams



Can scale # of shards

Kinesis Data Streams

- Retention between 1 day to 365 days
- Ability to reprocess (replay) data
- Once data is inserted in Kinesis, it can't be deleted (immutability)
- Data that shares the same partition goes to the same shard (ordering)
- Producers: AWS SDK, Kinesis Producer Library (KPL), Kinesis Agent
- Consumers:
  - Write your own: Kinesis Client Library (KCL), AWS SDK
  - Managed: AWS Lambda, Kinesis Data Firehose, Kinesis Data Analytics,

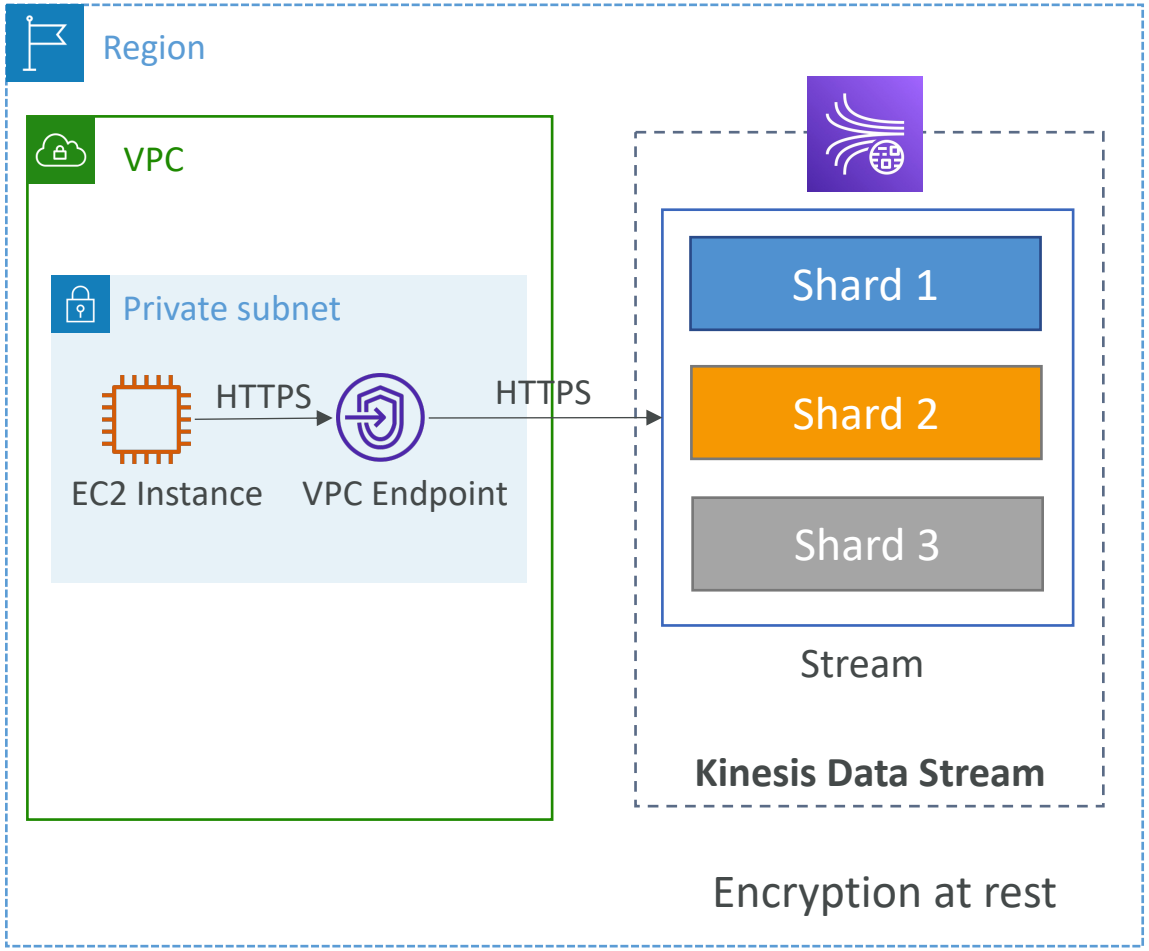
Kinesis Data Streams – Capacity Modes

- Provisioned mode:
  - You choose the number of shards provisioned, scale manually or using API
  - Each shard gets 1MB/s in (or 1000 records per second)
  - Each shard gets 2MB/s out (classic or enhanced fan-out consumer)
  - You pay per shard provisioned per hour

- On-demand mode:
  - No need to provision or manage the capacity
  - Default capacity provisioned (4 MB/s in or 4000 records per second)
  - Scales automatically based on observed throughput peak during the last 30 days
  - Pay per stream per hour & data in/out per GB

Kinesis Data Streams Security

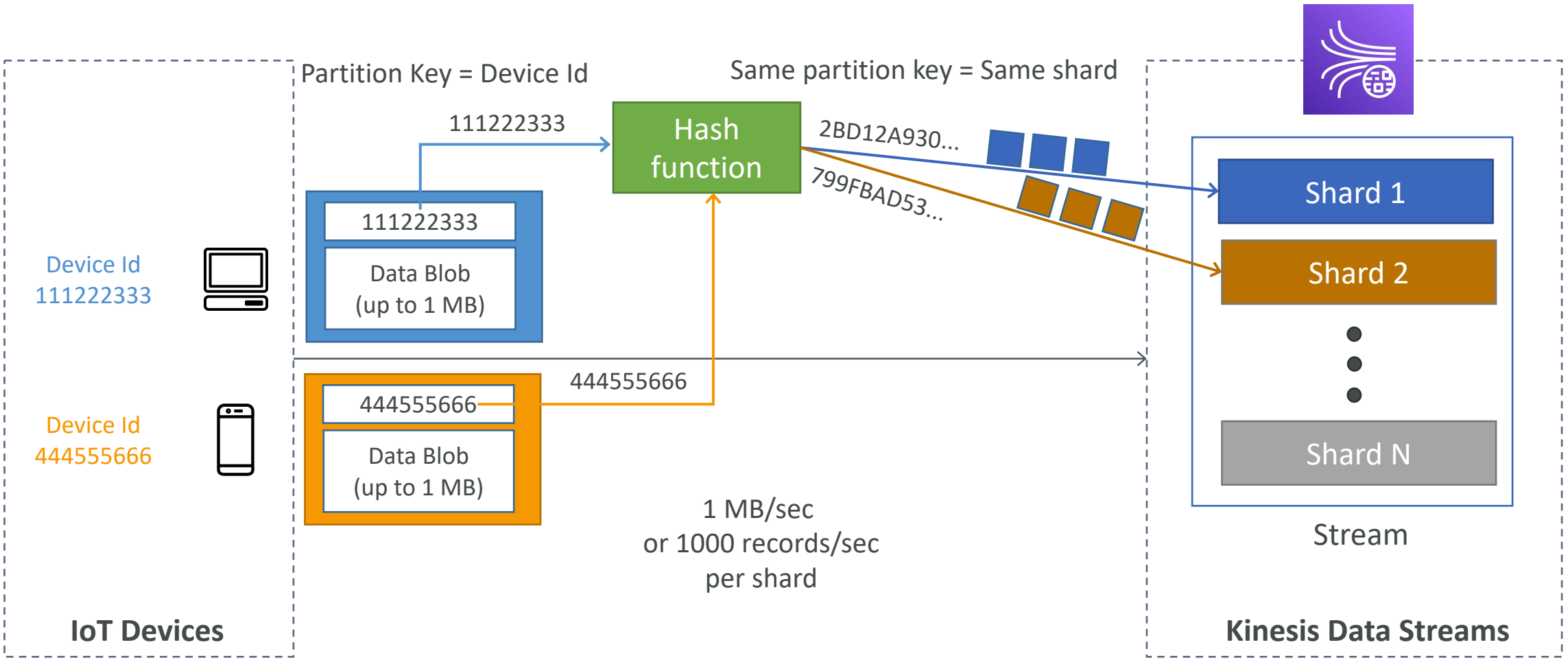
- Control access / authorization using IAM policies
- Encryption in flight using HTTPS endpoints
- Encryption at rest using KMS
- You can implement encryption/decryption of data on client side (harder)
- VPC Endpoints available for Kinesis to access within VPC
- Monitor API calls using CloudTrail



Kinesis Producers

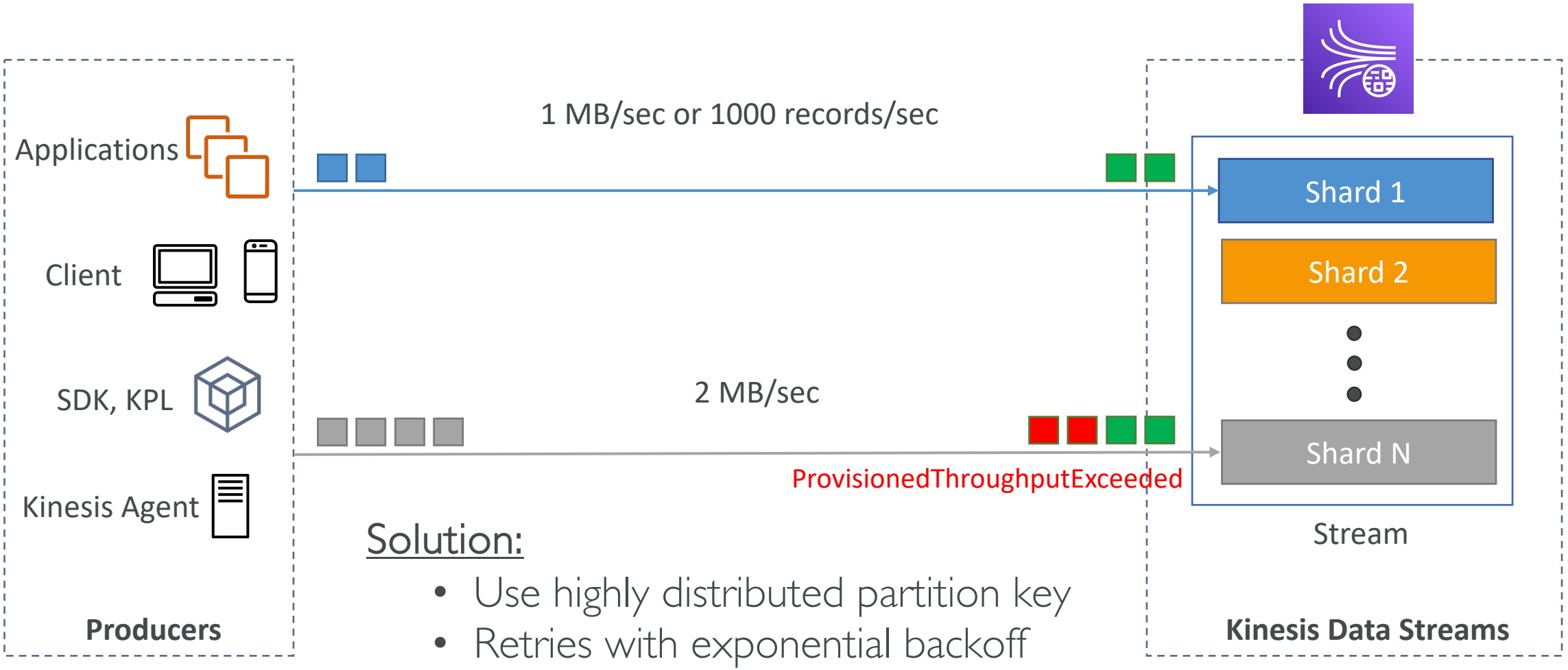
- Puts data records into data streams
- Data record consists of:
  - Sequence number (unique per partition-key within shard)
  - Partition key (must specify while put records into stream)
  - Data blob (up to 1 MB)
- Producers:
  - **AWS SDK:** simple producer
  - **Kinesis Producer Library (KPL):** C++, Java, batch, compression, retries
  - **Kinesis Agent:** monitor log files
- Write throughput: 1 MB/sec or 1000 records/sec per shard
- PutRecord API
- Use batching with PutRecords API to reduce costs & increase throughput

Kinesis Producers



Use highly distributed partition key to avoid “hot partition”

Kinesis - ProvisionedThroughputExceeded



Solution:

- Use highly distributed partition key
- Retries with exponential backoff
- Increase shards (scaling)

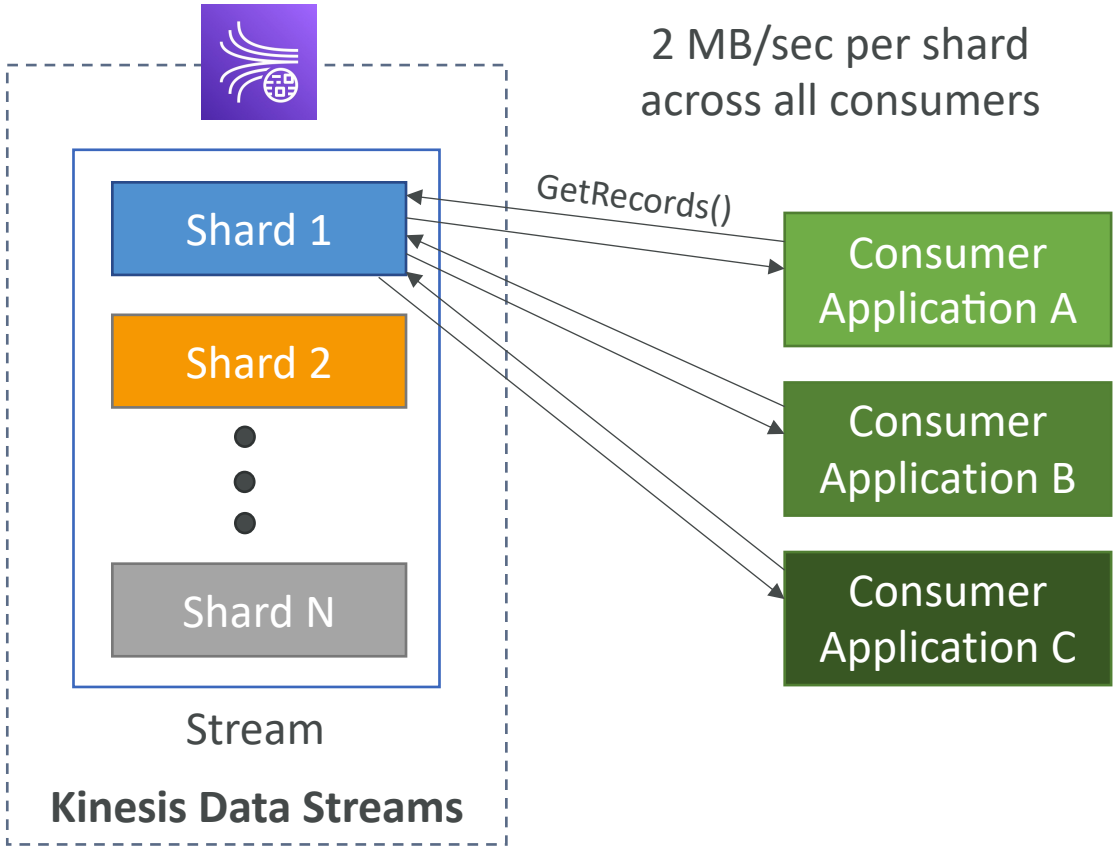
Kinesis Data Streams Consumers

- Get data records from data streams and process them

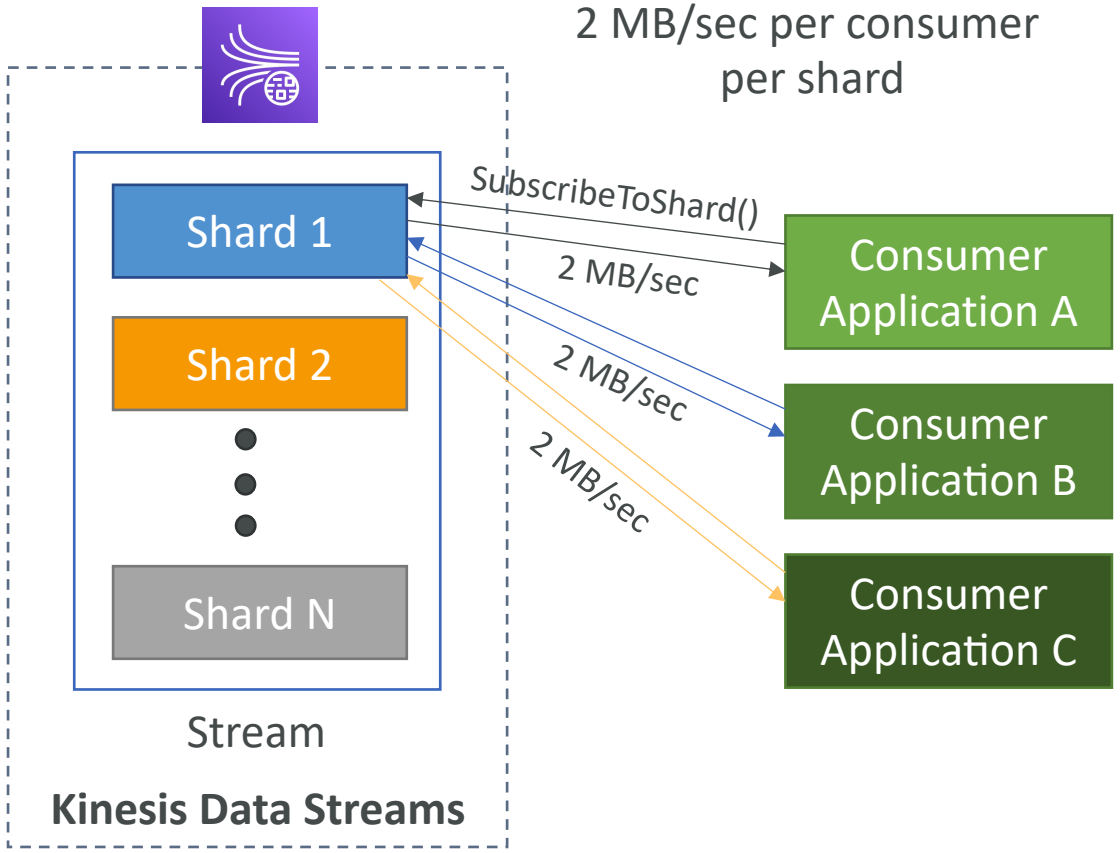
- AWS Lambda
- Kinesis Data Analytics
- Kinesis Data Firehose
- Custom Consumer (AWS SDK) – Classic or Enhanced Fan-Out
- Kinesis Client Library (KCL): library to simplify reading from data stream

Kinesis Consumers – Custom Consumer

Shared (Classic) Fan-out Consumer



Enhanced Fan-out Consumer



Kinesis Consumers Types

Shared (Classic) Fan-out Consumer - pull

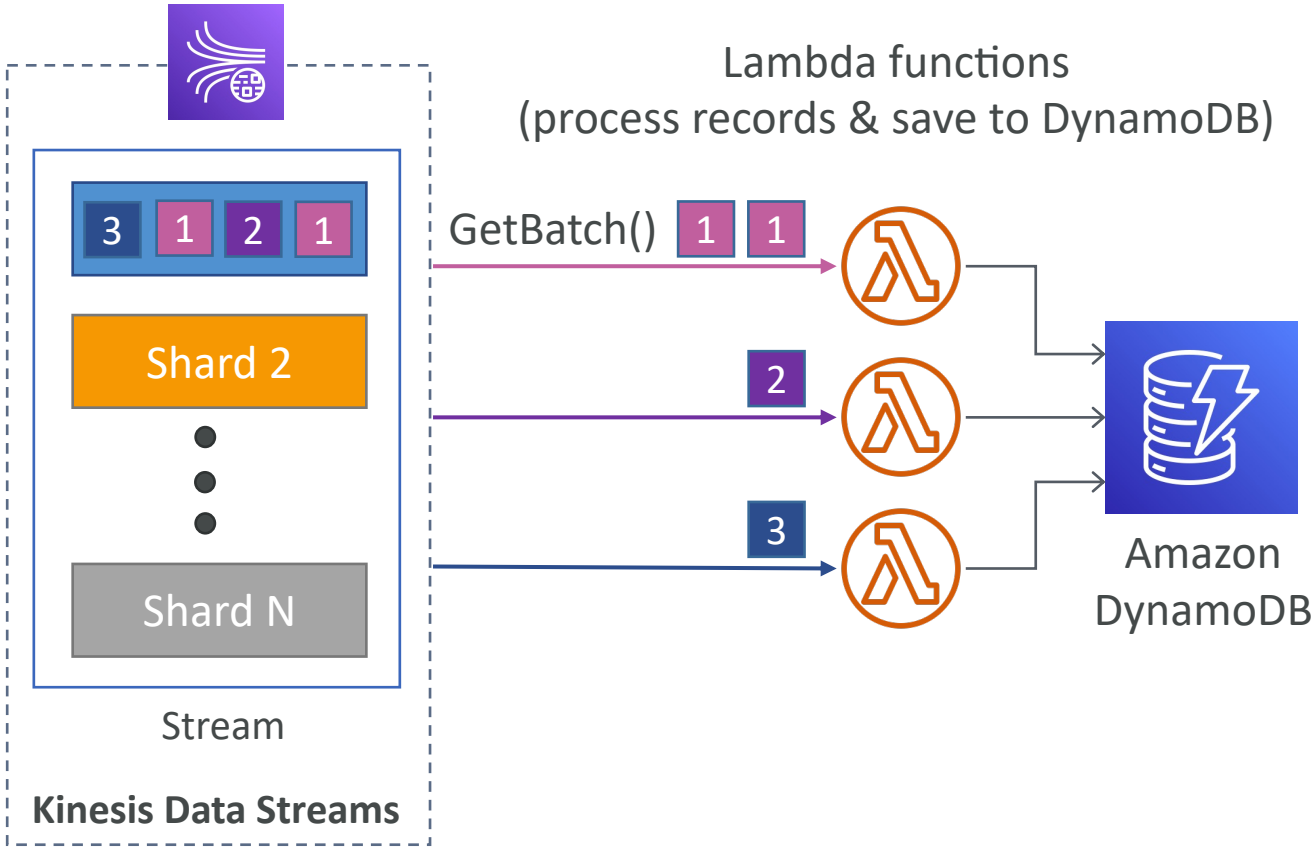
- Low number of consuming applications
- Read throughput: 2 MB/sec per shard across all consumers
- Max. 5 GetRecords API calls/sec
- Latency ~200 ms
- Minimize cost ($)
- Consumers poll data from Kinesis using GetRecords API call
- Returns up to 10 MB (then throttle for 5 seconds) or up to 10000 records

Enhanced Fan-out Consumer - push

- Multiple consuming applications for the same stream
- 2 MB/sec per consumer per shard
- Latency ~70 ms
- Higher costs ($$$)
- Kinesis pushes data to consumers over HTTP/2 (SubscribeToShard API)
- Soft limit of 5 consumer applications (KCL) per data stream (default)

Kinesis Consumers – AWS Lambda

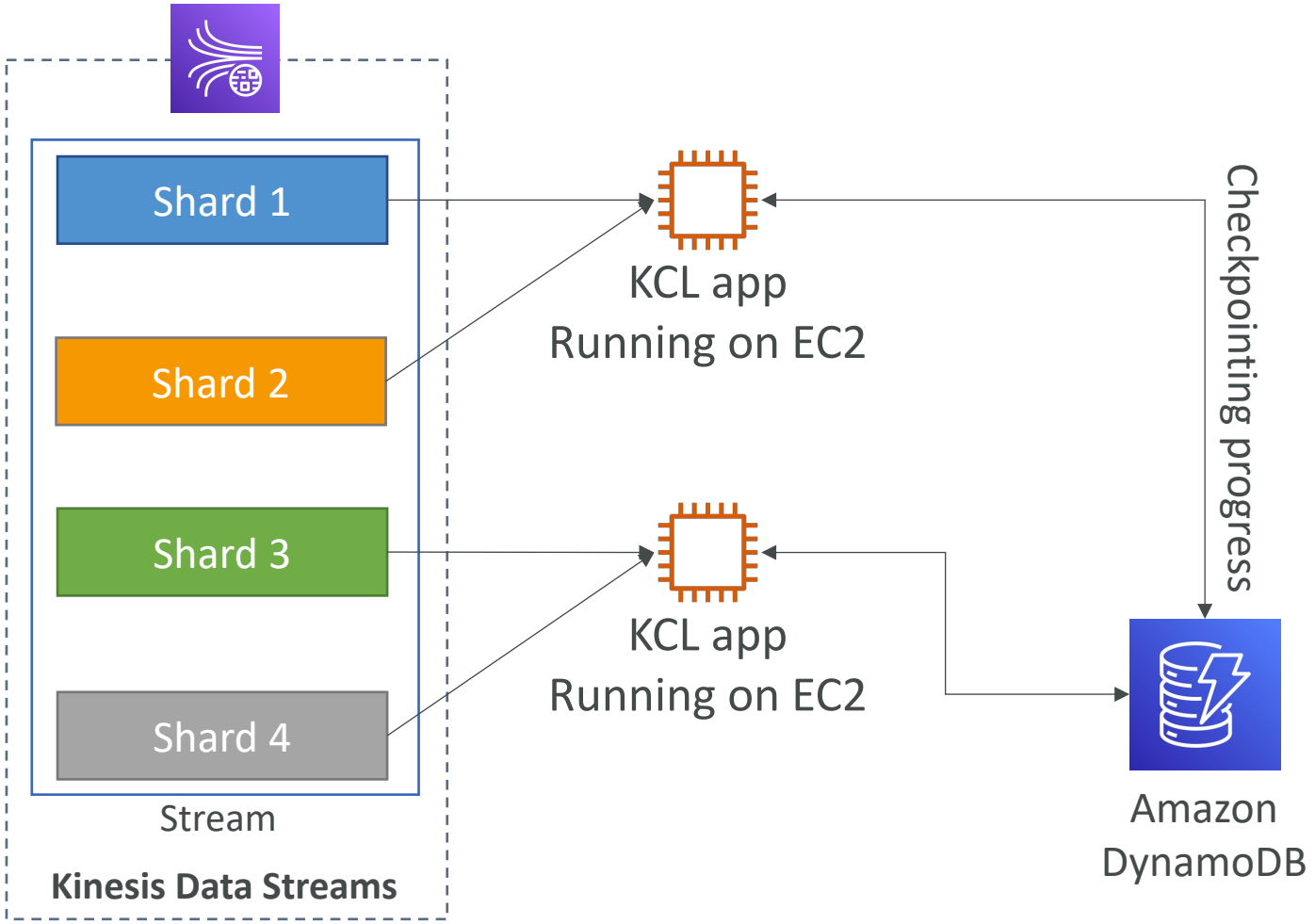
- Supports Classic & Enhanced fan-out consumers
- Read records in batches
- Can configure **batch size** and **batch window**
- If error occurs, Lambda retries until succeeds or data expired
- Can process up to 10 batches per shard simultaneously



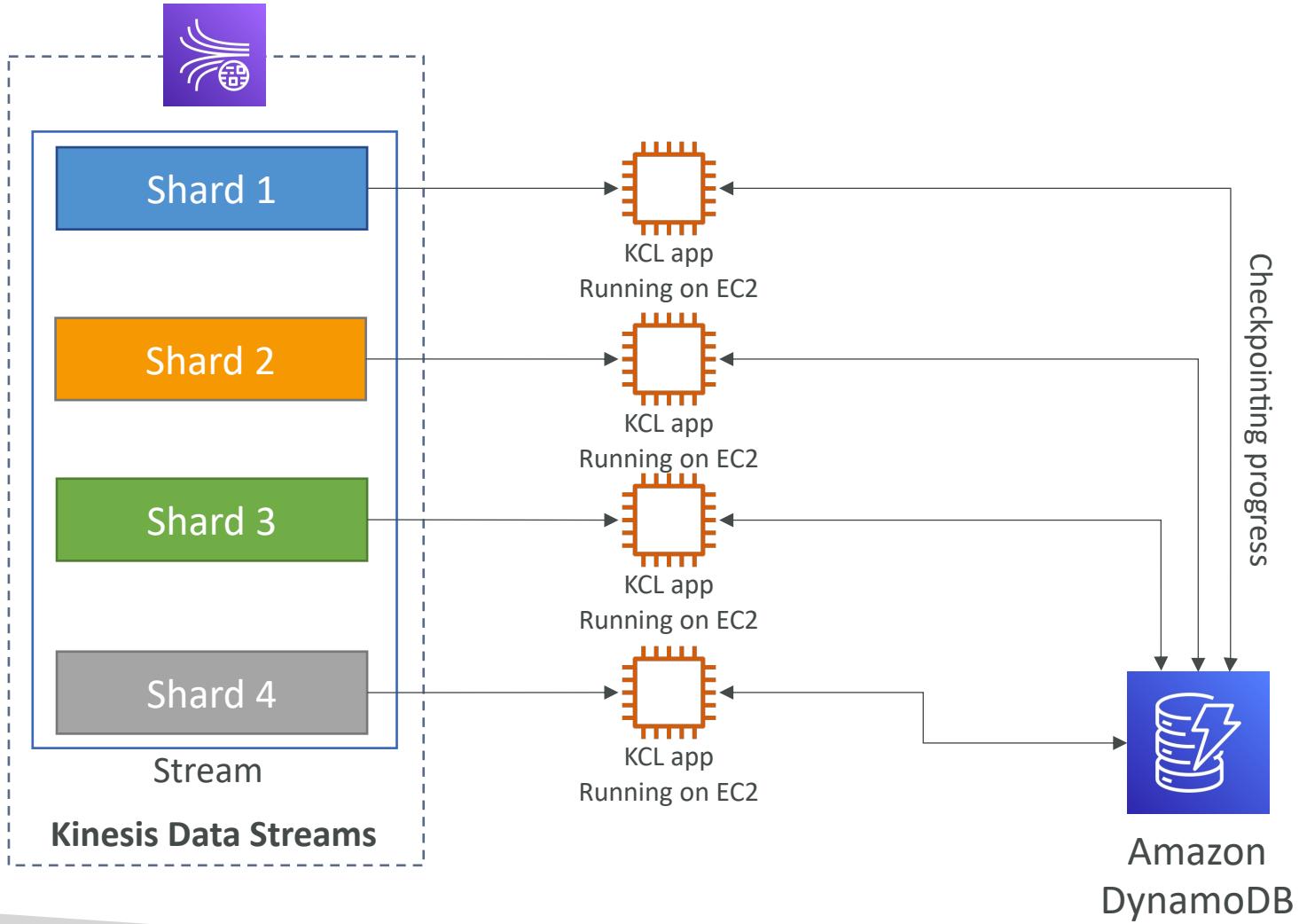
Kinesis Client Library (KCL)

- A Java library that helps read record from a Kinesis Data Stream with distributed applications sharing the read workload
- Each shard is to be read by only one KCL instance
  - 4 shards = max. 4 KCL instances
  - 6 shards = max. 6 KCL instances
- Progress is checkpointed into DynamoDB (needs IAM access)
- Track other workers and share the work amongst shards using DynamoDB
- KCL can run on EC2, Elastic Beanstalk, and on-premises
- Records are read in order at the shard level
- Versions:
  - KCL 1.x (supports shared consumer)
  - KCL 2.x (supports shared & enhanced fan-out consumer)

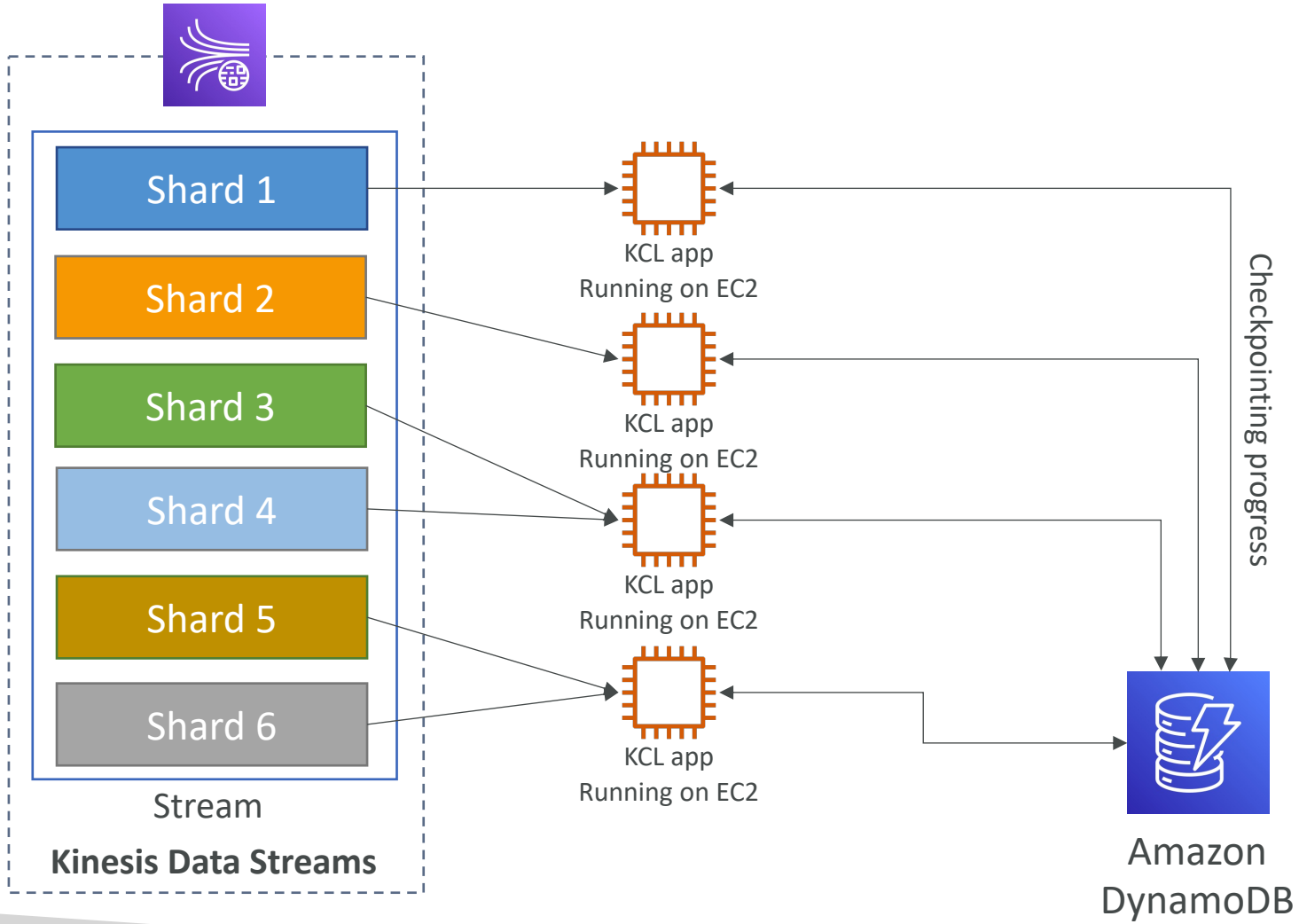
KCL Example: 4 shards



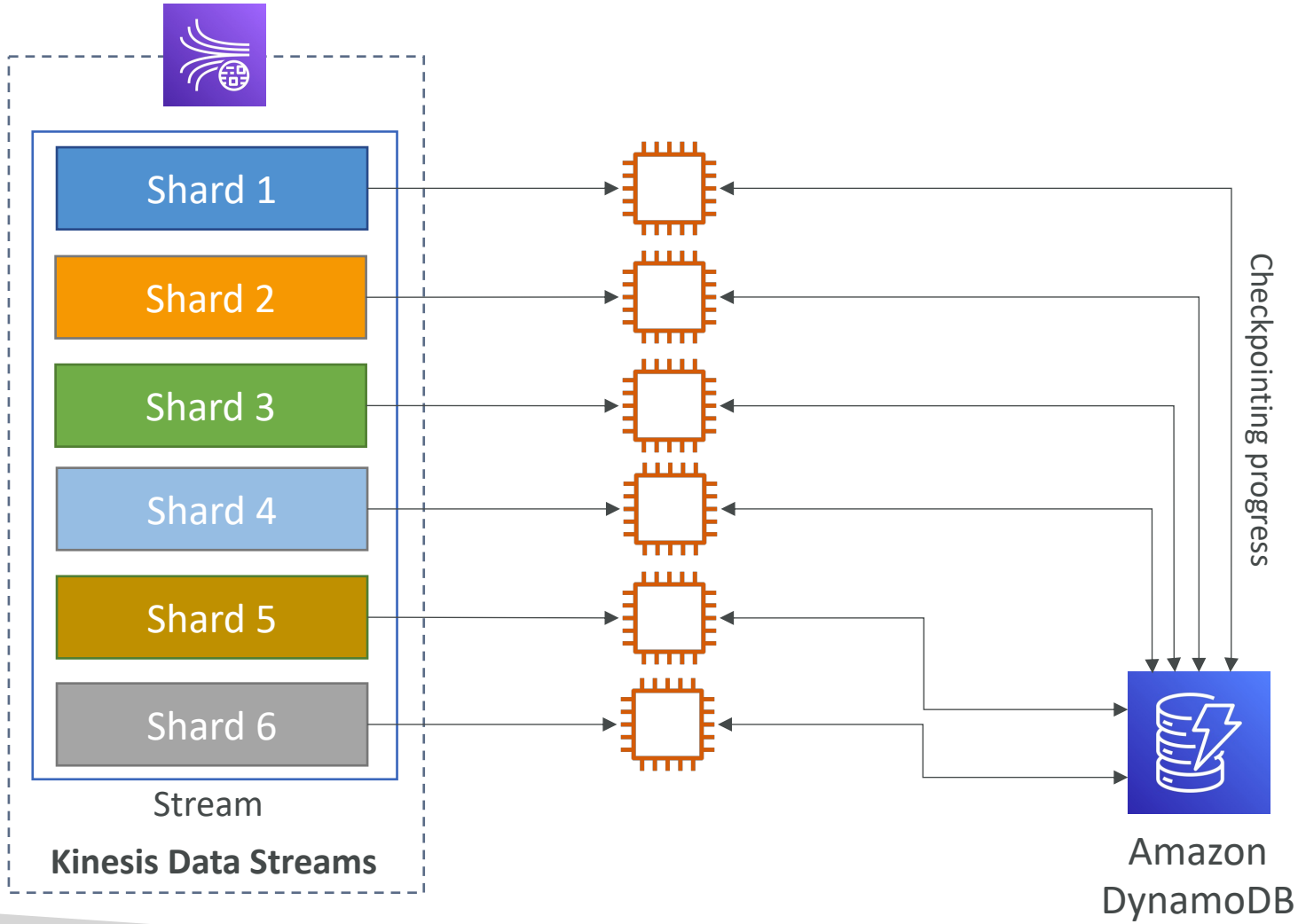
KCL Example: 4 shards, Scaling KCL App



KCL Example: 6 shards, Scaling Kinesis

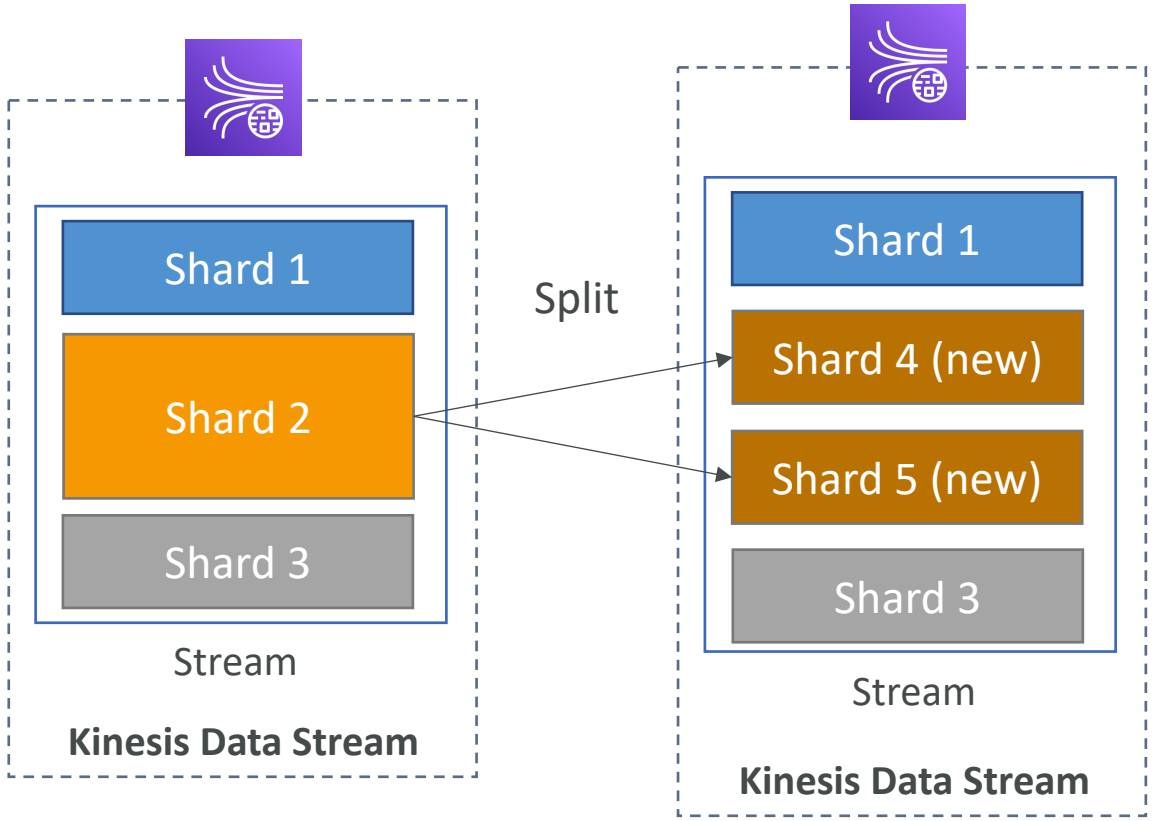


KCL Example: 6 shards, Scaling KCL App



Kinesis Operation – Shard Splitting

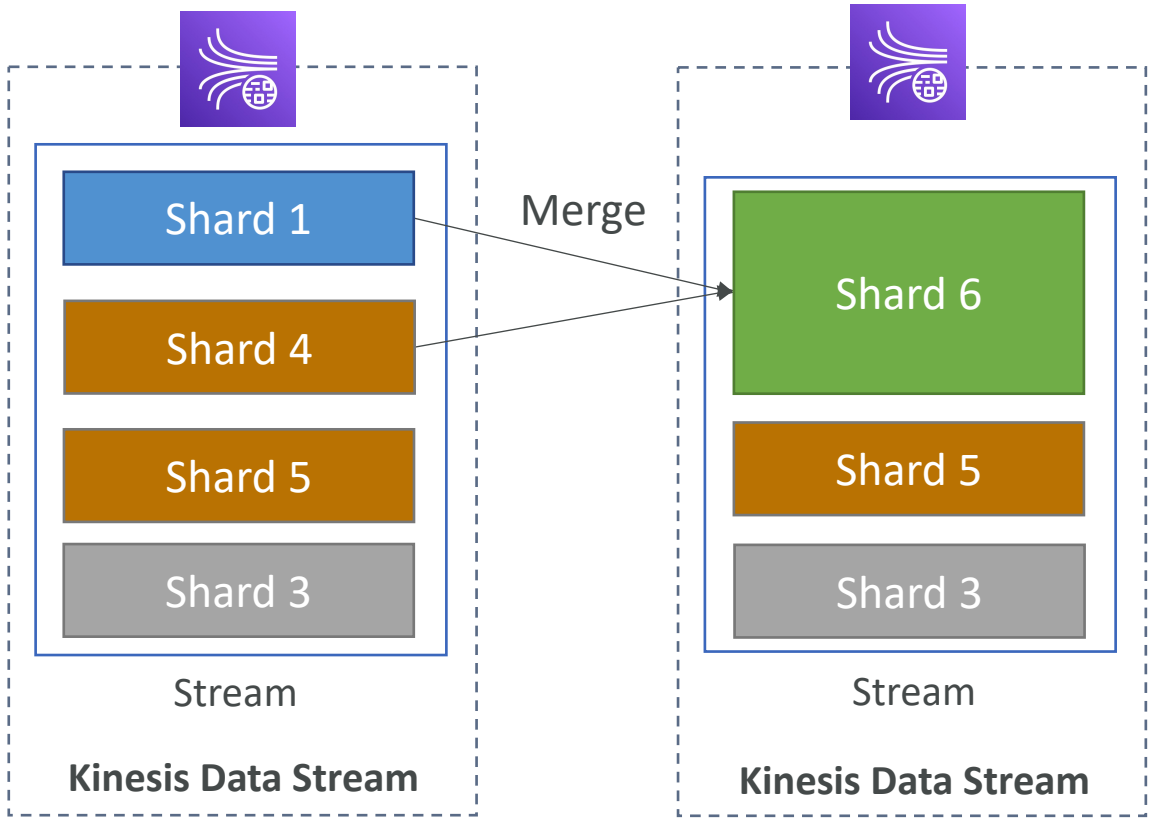
- Used to increase the Stream capacity (1 MB/s data in per shard)
- Used to divide a "hot shard"
- The old shard is closed and will be deleted once the data is expired
- No automatic scaling (manually increase/decrease capacity)
- Can't split into more than two shards in a single operation



Increase capacity and cost

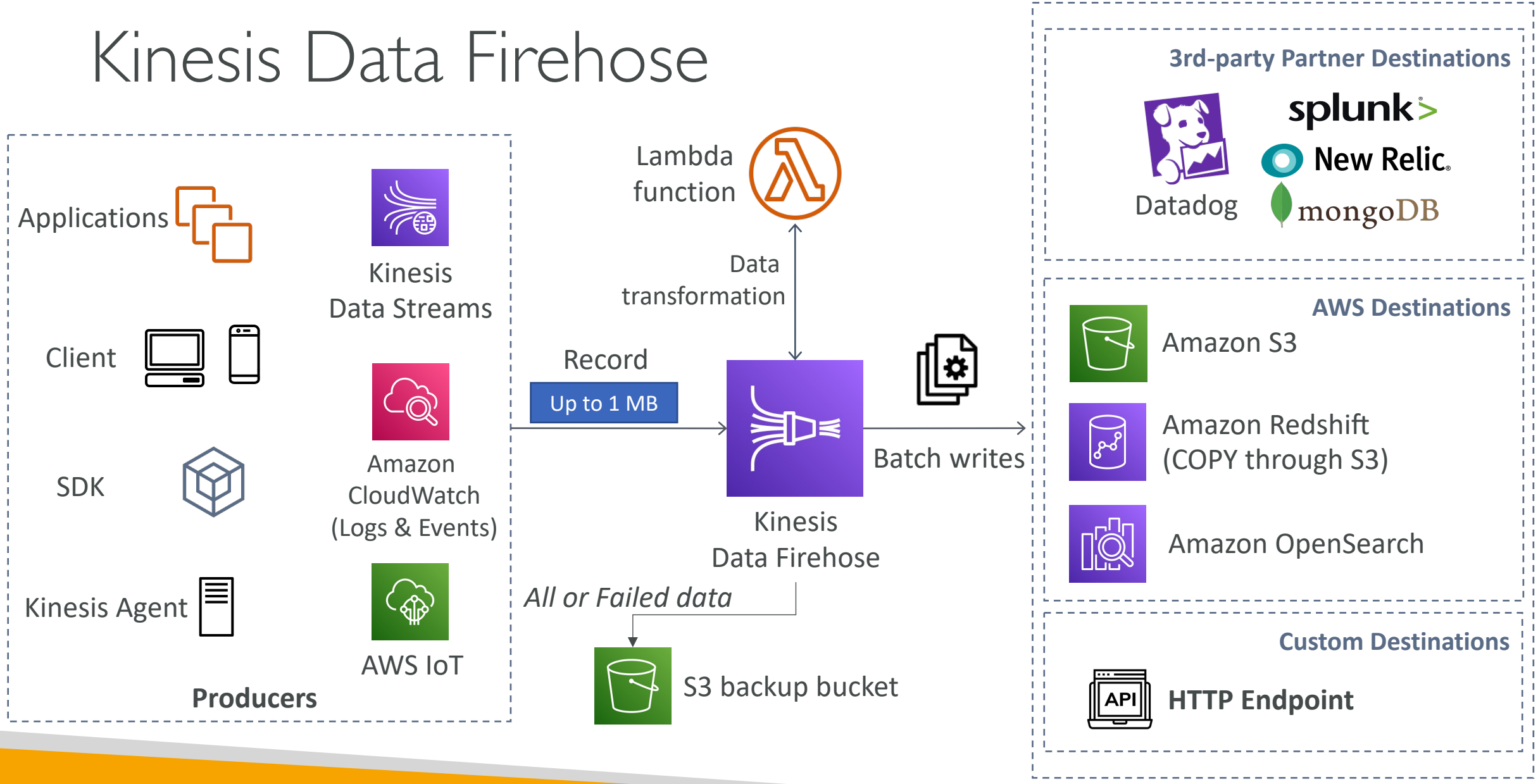
Kinesis Operation – Merging Shards

- Decrease the Stream capacity and save costs
- Can be used to group two shards with low traffic (cold shards)
- Old shards are closed and will be deleted once the data is expired
- Can't merge more than two shards in a single operation

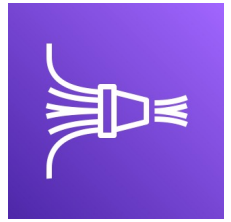


Decrease capacity and cost

Kinesis Data Firehose



Kinesis Data Firehose

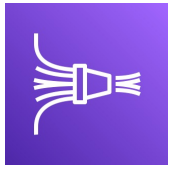


- Fully Managed Service, no administration, automatic scaling, serverless
  - AWS: Redshift / Amazon S3 / OpenSearch
  - 3rd party partner: Splunk / MongoDB / DataDog / NewRelic / …
  - Custom: send to any HTTP endpoint
- Pay for data going through Firehose
- Near Real Time
  - Buffer interval: 0 seconds (no buffering) to 900 seconds
  - Buffer size: minimum 1MB
- Supports many data formats, conversions, transformations, compression
- Supports custom data transformations using AWS Lambda
- Can send failed or all data to a backup S3 bucket

Kinesis Data Streams vs Firehose

Kinesis Data Streams

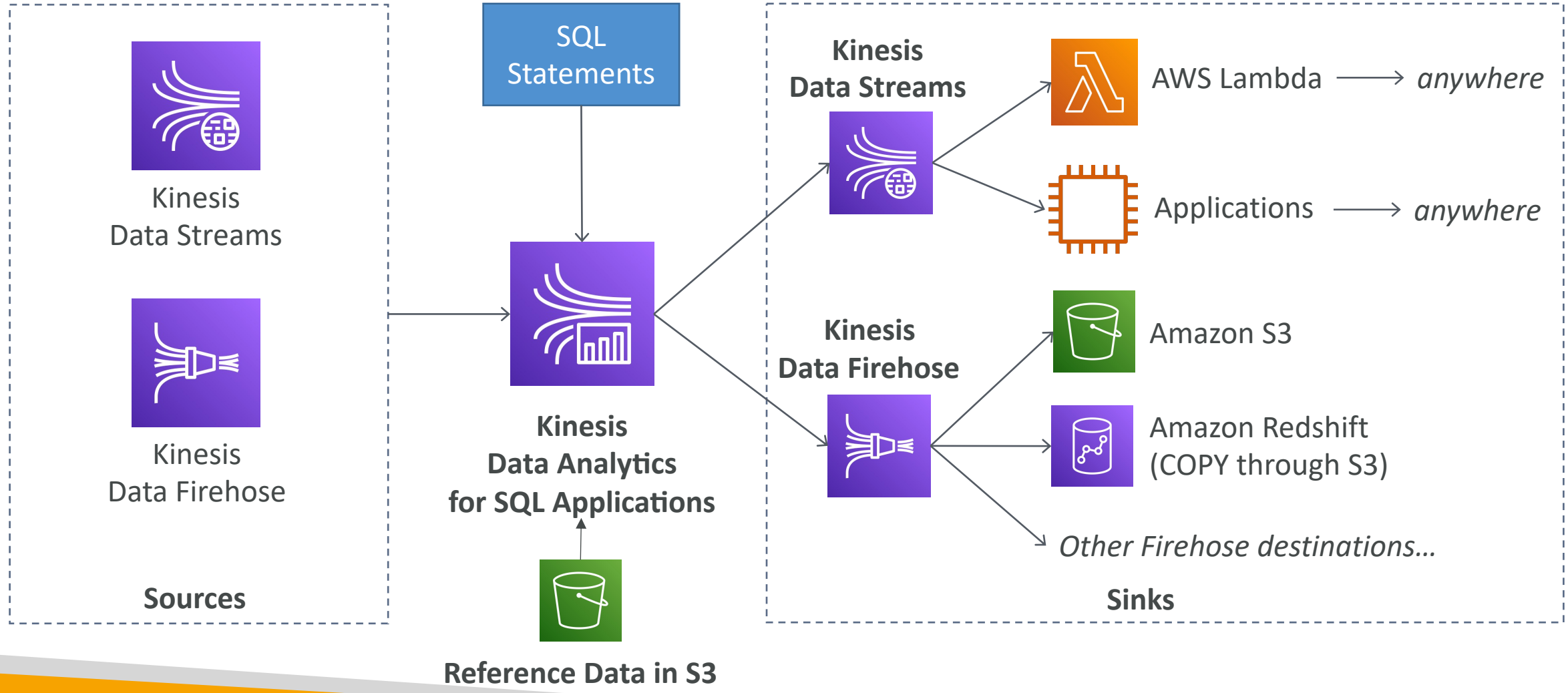
- Streaming service for ingest at scale
- Write custom code (producer / consumer)
- Real-time (~200 ms)
- Manage scaling (shard splitting / merging)
- Data storage for 1 to 365 days
- Supports replay capability



Kinesis Data Firehose

- Load streaming data into S3 / Redshift / OpenSearch / 3rd party / custom HTTP
- Fully managed
- Near real-time
- Automatic scaling
- No data storage
- Doesn't support replay capability

Kinesis Data Analytics for SQL applications

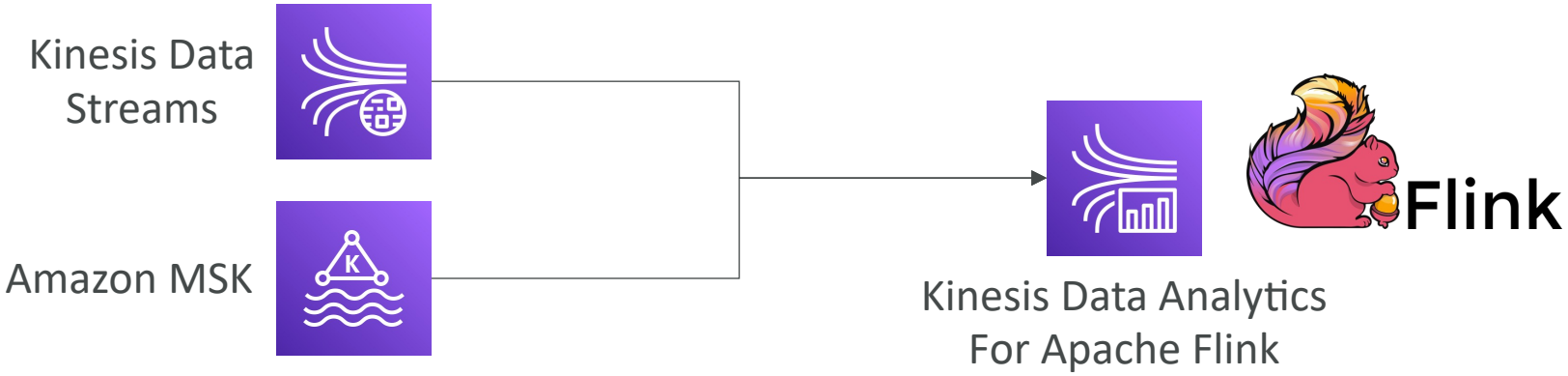


Kinesis Data Analytics (SQL application)

- Real-time analytics on **Kinesis Data Streams & Firehose** using SQL
- Add reference data from Amazon S3 to enrich streaming data
- Fully managed, no servers to provision
- Automatic scaling
- Pay for actual consumption rate
- Output:
  - Kinesis Data Streams: create streams out of the real-time analytics queries
  - Kinesis Data Firehose: send analytics query results to destinations
- Use cases:
  - Time-series analytics
  - Real-time dashboards
  - Real-time metrics

Kinesis Data Analytics for Apache Flink

- Use Flink (Java, Scala or SQL) to process and analyze streaming data

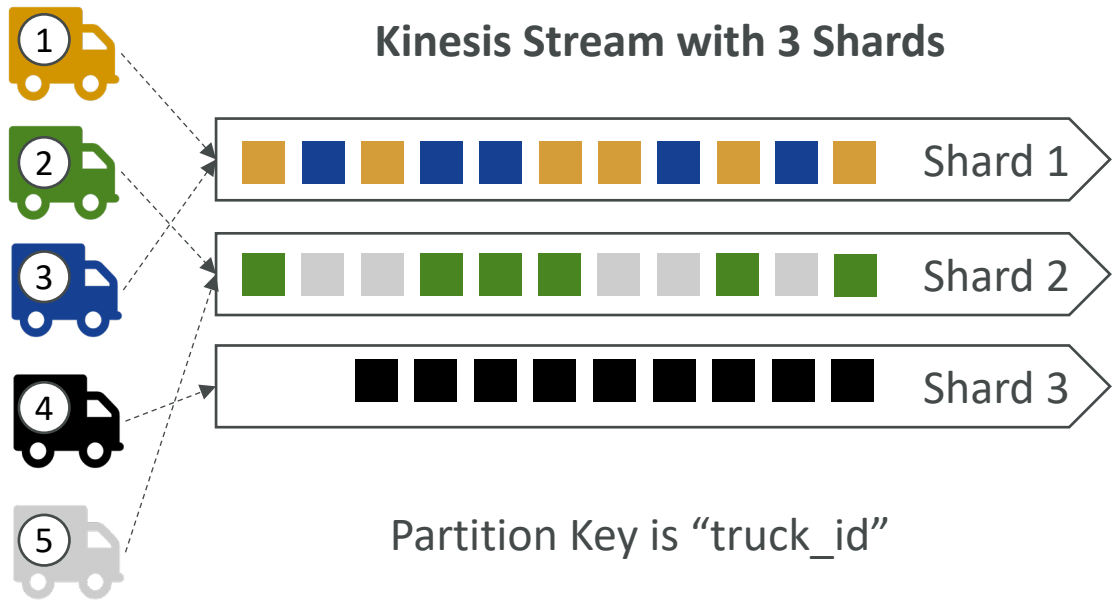


- Run any Apache Flink application on a managed cluster on AWS
  - provisioning compute resources, parallel computation, automatic scaling
  - application backups (implemented as checkpoints and snapshots)
  - Use any Apache Flink programming features
  - Flink does not read from Firehose (use Kinesis Analytics for SQL instead)

Ordering data into Kinesis

- Imagine you have 100 trucks (truck_1, truck_2, … truck_100) on the road sending their GPS positions regularly into AWS.
- You want to consume the data in order for each truck, so that you can track their movement accurately.
- How should you send that data into Kinesis?

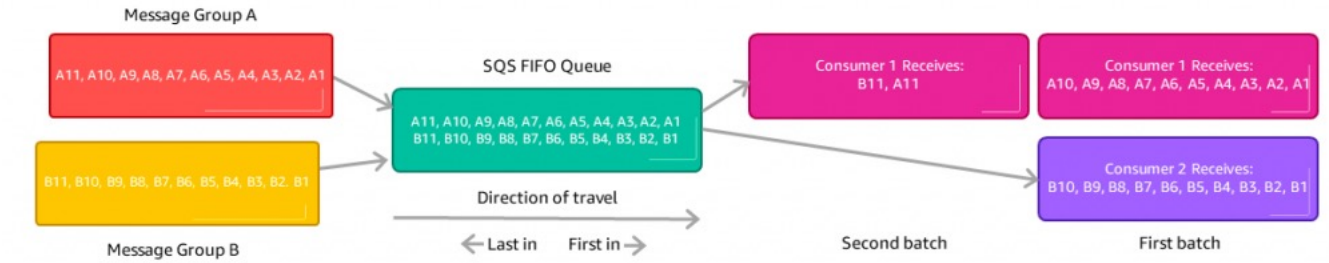
- Answer: send using a "Partition Key" value of the "truck_id"
- The same key will always go to the same shard



Ordering data into SQS

- For SQS standard, there is no ordering.
- For SQS FIFO, if you don't use a Group ID, messages are consumed in the order they are sent, **with only one consumer**

- You want to scale the number of consumers, but you want messages to be "grouped" when they are related to each other
- Then you use a Group ID (similar to Partition Key in Kinesis)



https://aws.amazon.com/blogs/compute/solving-complex-ordering-challenges-with-amazon-sqs-fifo-queues/

Kinesis vs SQS ordering

- Let's assume 100 trucks, 5 kinesis shards, 1 SQS FIFO
- Kinesis Data Streams:
  - On average you'll have 20 trucks per shard
  - Trucks will have their data ordered within each shard
  - The maximum amount of consumers in parallel we can have is 5
  - Can receive up to 5 MB/s of data
- SQS FIFO
  - You only have one SQS FIFO queue
  - You will have 100 Group ID
  - You can have up to 100 Consumers (due to the 100 Group ID)
  - You have up to 300 messages per second (or 3000 if using batching)

SQS vs SNS vs Kinesis

SQS:

- Consumer “pull data”
- Data is deleted after being consumed
- Can have as many workers (consumers) as we want
- No need to provision throughput
- Ordering guarantees only on FIFO queues
- Individual message delay capability

SNS:

- Push data to many subscribers
- Up to 12,500,000 subscribers
- Data is not persisted (lost if not delivered)
- Pub/Sub
- Up to 100,000 topics
- No need to provision throughput
- Integrates with SQS for fan-out architecture pattern
- FIFO capability for SQS FIFO

Kinesis:

- Standard: pull data
  - 2 MB per shard
- Enhanced-fan out: push data
  - 2 MB per shard per consumer
- Possibility to replay data
- Meant for real-time big data, analytics and ETL
- Ordering at the shard level
- Data expires after X days
- Provisioned mode or on-demand capacity mode

AWS Monitoring, Troubleshooting & Audit

CloudWatch, X-Ray and CloudTrail

Why Monitoring is Important

- We know how to deploy applications
  - Safely
  - Automatically
  - Using Infrastructure as Code
  - Leveraging the best AWS components!
- Our applications are deployed, and our users don't care how we did it…
- Our users only care that the application is working!
  - Application latency: will it increase over time?
  - Application outages: customer experience should not be degraded
  - Users contacting the IT department or complaining is not a good outcome
  - Troubleshooting and remediation
- Internal monitoring:
  - Can we prevent issues before they happen?
  - Performance and Cost
  - Trends (scaling patterns)
  - Learning and Improvement

Monitoring in AWS

- AWS CloudWatch:
  - Metrics: Collect and track key metrics
  - Logs: Collect, monitor, analyze and store log files
  - Events: Send notifications when certain events happen in your AWS
  - Alarms: React in real-time to metrics / events
- AWS X-Ray:
  - Troubleshooting application performance and errors
  - Distributed tracing of microservices
- AWS CloudTrail:
  - Internal monitoring of API calls being made
  - Audit changes to AWS Resources by your users

AWS CloudWatch Metrics

- CloudWatch provides metrics for *every* services in AWS
- **Metric** is a variable to monitor (CPUUtilization, NetworkIn…)
- Metrics belong to **namespaces**
- **Dimension** is an attribute of a metric (instance id, environment, etc…).
- Up to 30 dimensions per metric
- Metrics have **timestamps**
- Can create CloudWatch dashboards of metrics

EC2 Detailed monitoring

- EC2 instance metrics have metrics "every 5 minutes"
- With detailed monitoring (for a cost), you get data "every 1 minute"
- Use detailed monitoring if you want to scale faster for your ASG!

- The AWS Free Tier allows us to have 10 detailed monitoring metrics

- Note: EC2 Memory usage is by default not pushed (must be pushed from inside the instance as a custom metric)

CloudWatch Custom Metrics

- Possibility to define and send your own custom metrics to CloudWatch
- Example: memory (RAM) usage, disk space, number of logged in users …
- Use API call **PutMetricData**
- Ability to use dimensions (attributes) to segment metrics
  - Instance.id
  - Environment.name
- Metric resolution (**StorageResolution** API parameter – two possible value):
  - Standard: 1 minute (60 seconds)
  - High Resolution: 1/5/10/30 second(s) – Higher cost
- **Important:** Accepts metric data points two weeks in the past and two hours in the future (make sure to configure your EC2 instance time correctly)

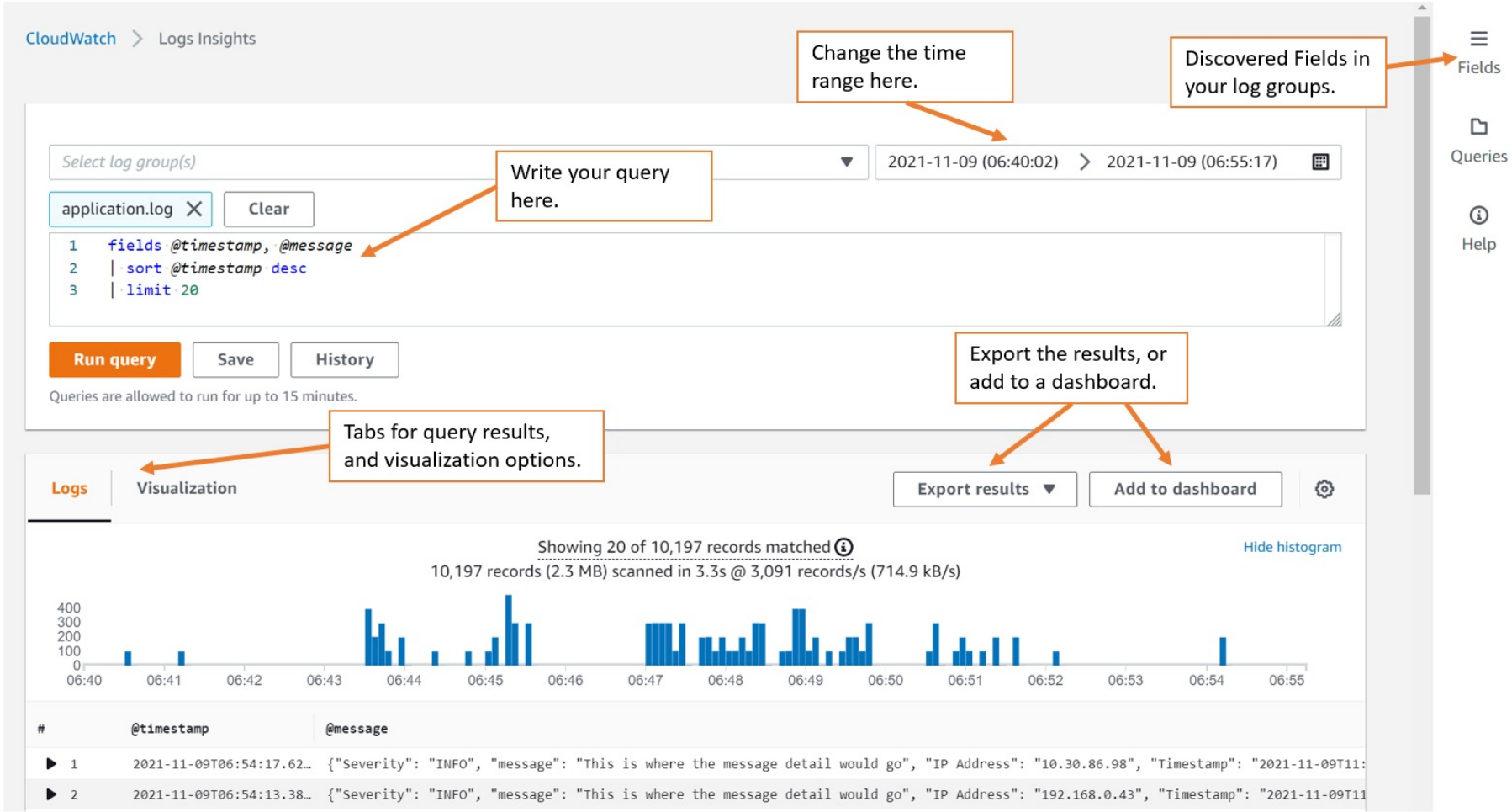
CloudWatch Logs

- **Log groups:** arbitrary name, usually representing an application
- **Log stream:** instances within application / log files / containers
- Can define log expiration policies (never expire, 1 day to 10 years…)
- **CloudWatch Logs can send logs to:**
  - Amazon S3 (exports)
  - Kinesis Data Streams
  - Kinesis Data Firehose
  - AWS Lambda
  - OpenSearch
- Logs are encrypted by default
- Can setup KMS-based encryption with your own keys

CloudWatch Logs - Sources

- SDK, CloudWatch Logs Agent, CloudWatch Unified Agent
- Elastic Beanstalk: collection of logs from application
- ECS: collection from containers
- AWS Lambda: collection from function logs
- VPC Flow Logs: VPC specific logs
- API Gateway
- CloudTrail based on filter
- Route53: Log DNS queries

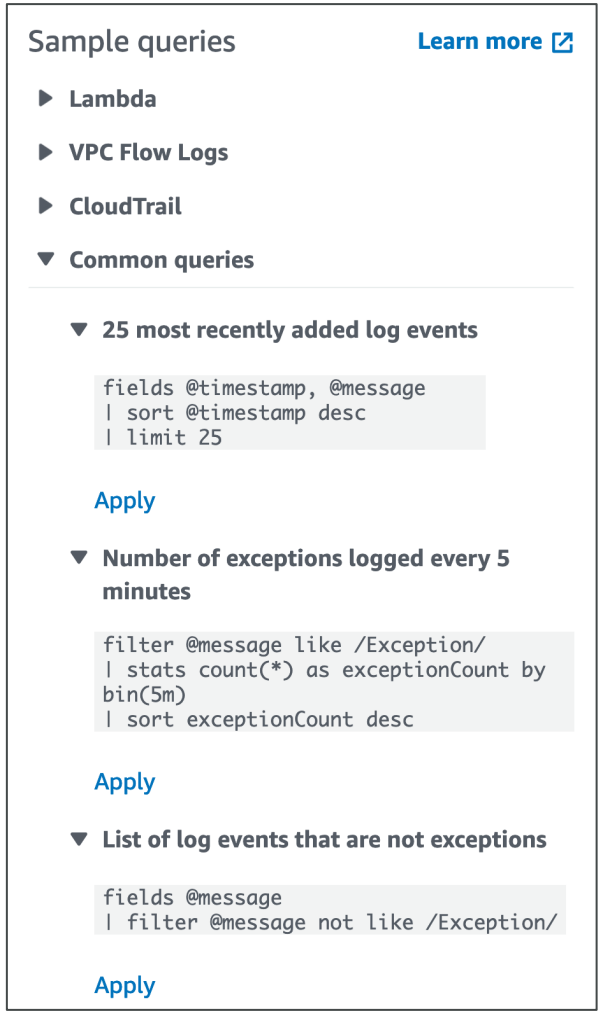
CloudWatch Logs Insights



https://mng.workshop.aws/operations-2022/detect/cwlogs.html

CloudWatch Logs Insights

- Search and analyze log data stored in CloudWatch Logs
- Example: find a specific IP inside a log, count occurrences of "ERROR" in your logs…
- Provides a purpose-built query language
  - Automatically discovers fields from AWS services and JSON log events
  - Fetch desired event fields, filter based on conditions, calculate aggregate statistics, sort events, limit number of events…
  - Can save queries and add them to CloudWatch Dashboards
- Can query multiple Log Groups in different AWS accounts
- It's a query engine, not a real-time engine



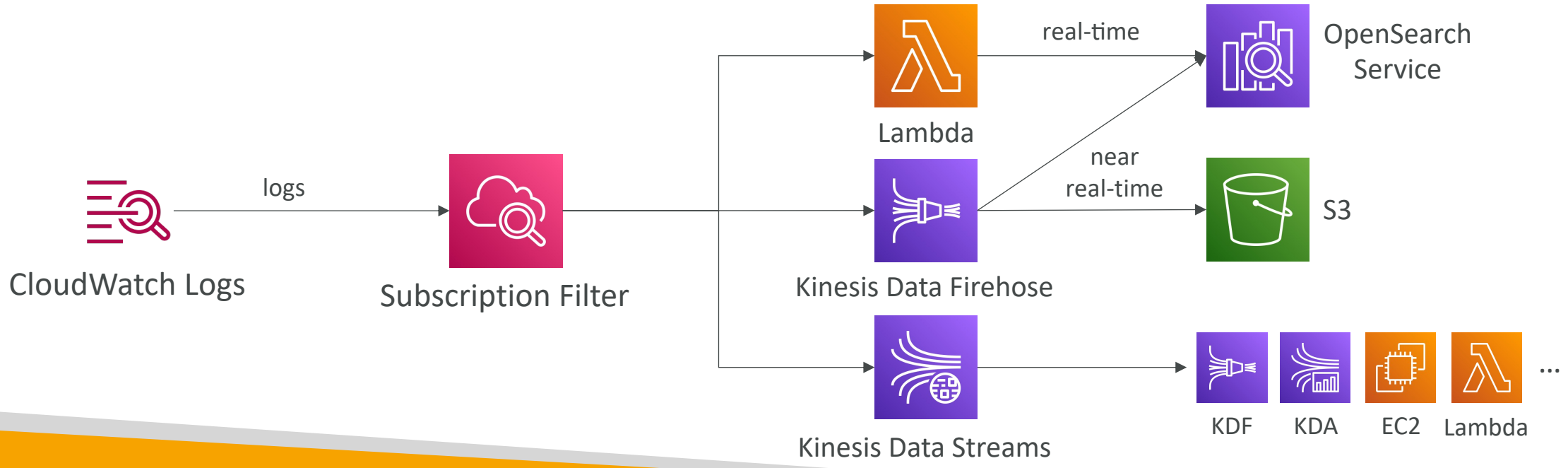
CloudWatch Logs – S3 Export

- Log data can take **up to 12 hours** to become available for export
- The API call is **CreateExportTask**

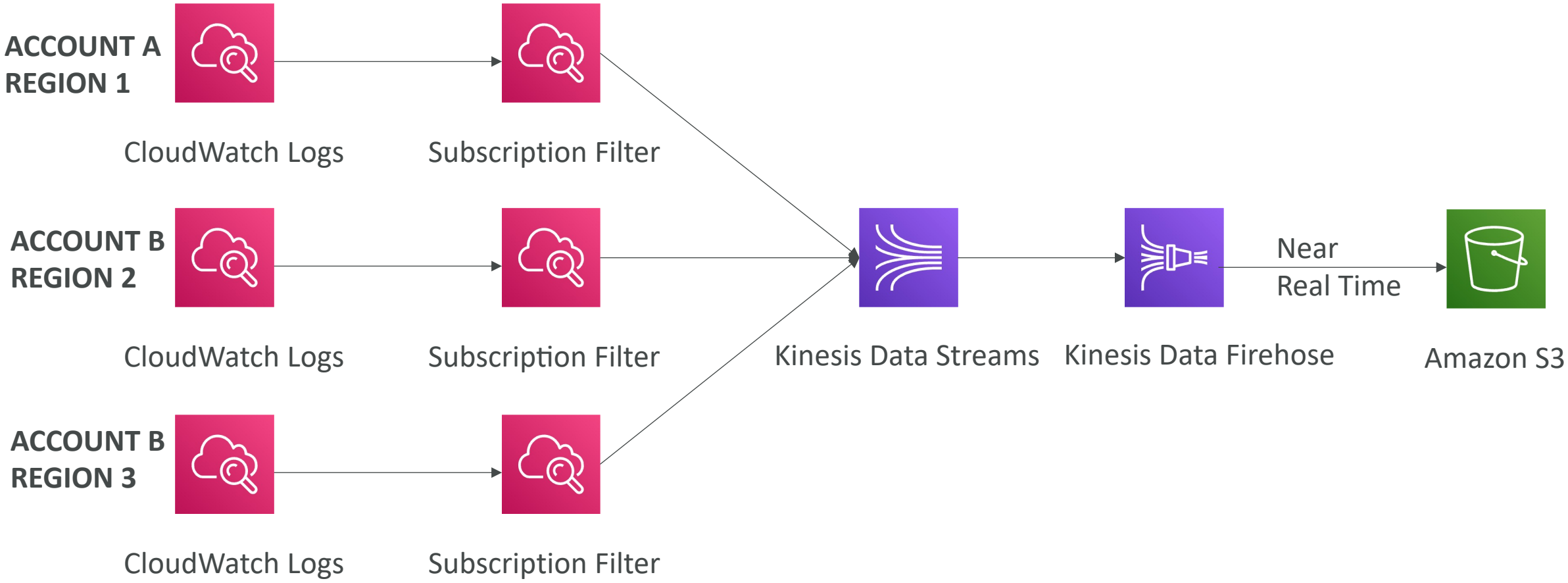
- Not near-real time or real-time… use Logs Subscriptions instead

CloudWatch Logs Subscriptions

- Get a real-time log events from CloudWatch Logs for processing and analysis
- Send to Kinesis Data Streams, Kinesis Data Firehose, or Lambda
- **Subscription Filter** – filter which logs are events delivered to your destination

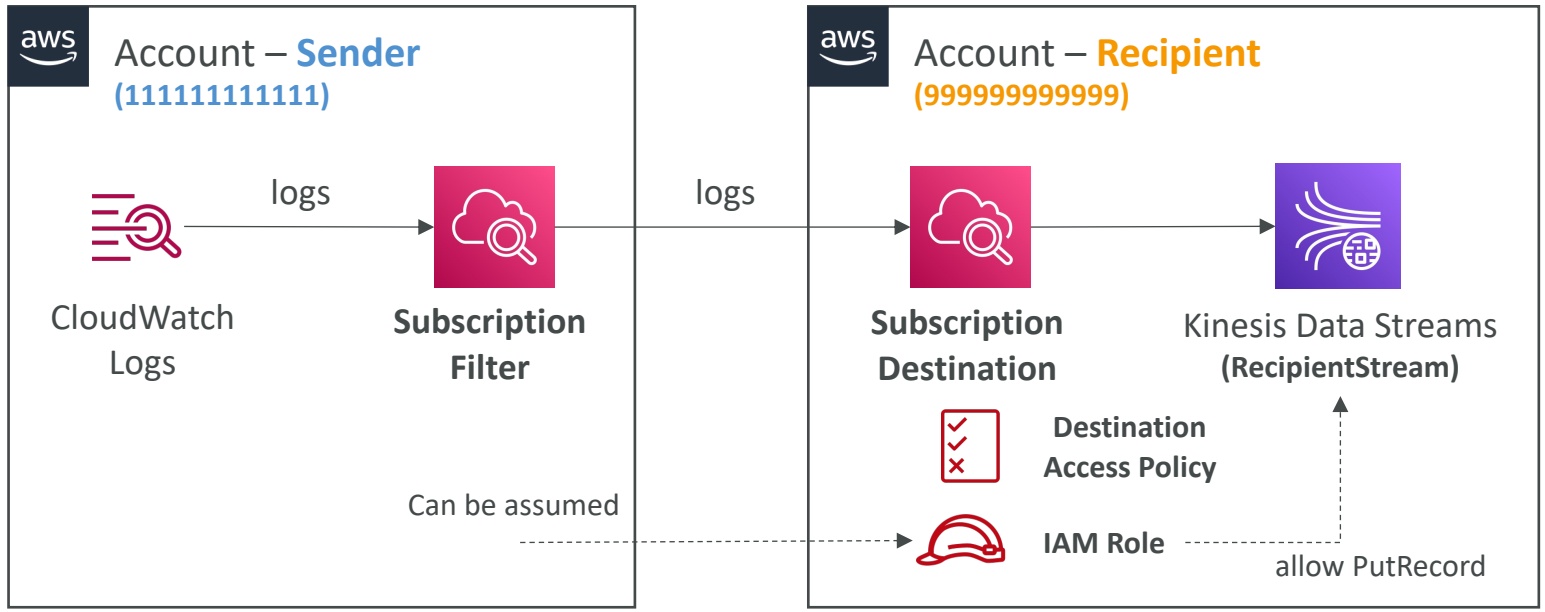


CloudWatch Logs Aggregation Multi-Account & Multi Region



CloudWatch Logs Subscriptions

- **Cross-Account Subscription** – send log events to resources in a different AWS account (KDS, KDF)



IAM Role (Cross-Account)

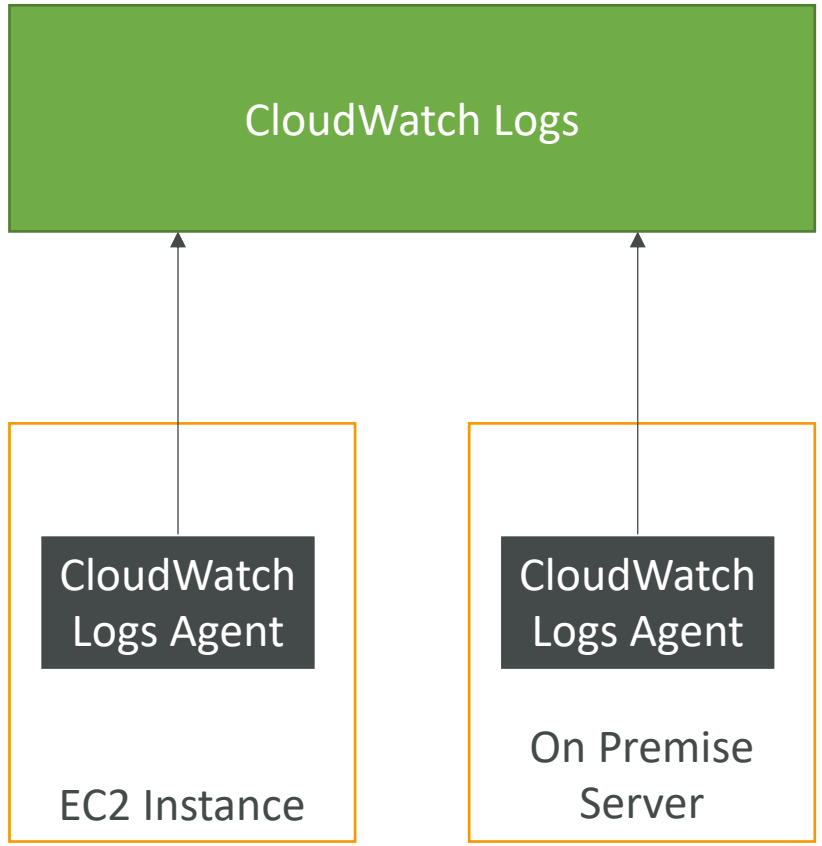
```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "kinesis:PutRecord",
      "Resource": "arn:aws:kinesis:us-east-1:
999999999999:stream/RecipientStream"
    }
  ]
}
```

Destination Access Policy

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "AWS": "111111111111"
      },
      "Action": "logs:PutSubscriptionFilter",
      "Resource": "arn:aws:logs:us-east-1:999999999999:
destination:testDestination"
    }
  ]
}
```

CloudWatch Logs for EC2

- By default, no logs from your EC2 machine will go to CloudWatch
- You need to run a CloudWatch agent on EC2 to push the log files you want
- Make sure IAM permissions are correct
- The CloudWatch log agent can be setup on-premises too



CloudWatch Logs Agent & Unified Agent

- For virtual servers (EC2 instances, on-premise servers…)
- CloudWatch Logs Agent
  - Old version of the agent
  - Can only send to CloudWatch Logs
- CloudWatch Unified Agent
  - Collect additional system-level metrics such as RAM, processes, etc…
  - Collect logs to send to CloudWatch Logs
  - Centralized configuration using SSM Parameter Store

CloudWatch Unified Agent – Metrics

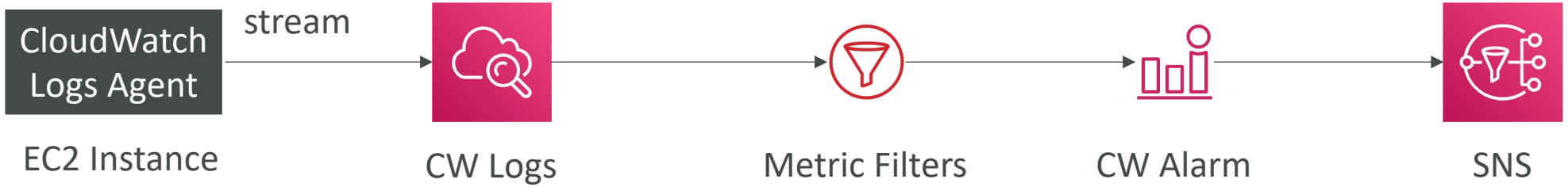
- Collected directly on your Linux server / EC2 instance

- **CPU** (active, guest, idle, system, user, steal)
- **Disk metrics** (free, used, total), Disk IO (writes, reads, bytes, iops)
- **RAM** (free, inactive, used, total, cached)
- **Netstat** (number of TCP and UDP connections, net packets, bytes)
- **Processes** (total, dead, bloqued, idle, running, sleep)
- **Swap Space** (free, used, used %)

- Reminder: out-of-the box metrics for EC2 – disk, CPU, network (high level)

CloudWatch Logs Metric Filter

- CloudWatch Logs can use filter expressions
  - For example, find a specific IP inside of a log
  - Or count occurrences of "ERROR" in your logs
  - Metric filters can be used to trigger alarms
- **Filters do not retroactively filter data. Filters only publish the metric data points for events that happen after the filter was created.**
- Ability to specify up to 3 Dimensions for the Metric Filter (optional)

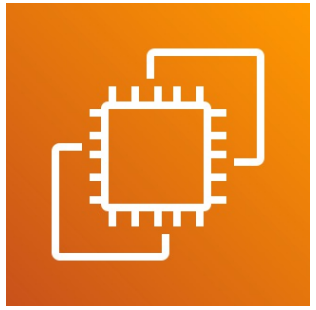


CloudWatch Alarms

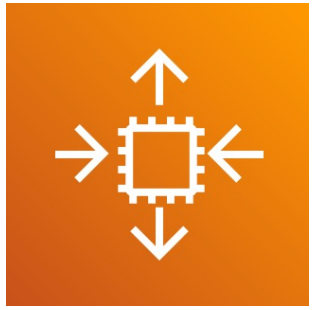
- Alarms are used to trigger notifications for any metric
- Various options (sampling, %, max, min, etc…)
- Alarm States:
  - OK
  - INSUFFICIENT_DATA
  - ALARM
- Period:
  - Length of time in seconds to evaluate the metric
  - High resolution custom metrics: 10 sec, 30 sec or multiples of 60 sec

CloudWatch Alarm Targets

- Stop, Terminate, Reboot, or Recover an EC2 Instance
- Trigger Auto Scaling Action
- Send notification to SNS (from which you can do pretty much anything)



Amazon EC2

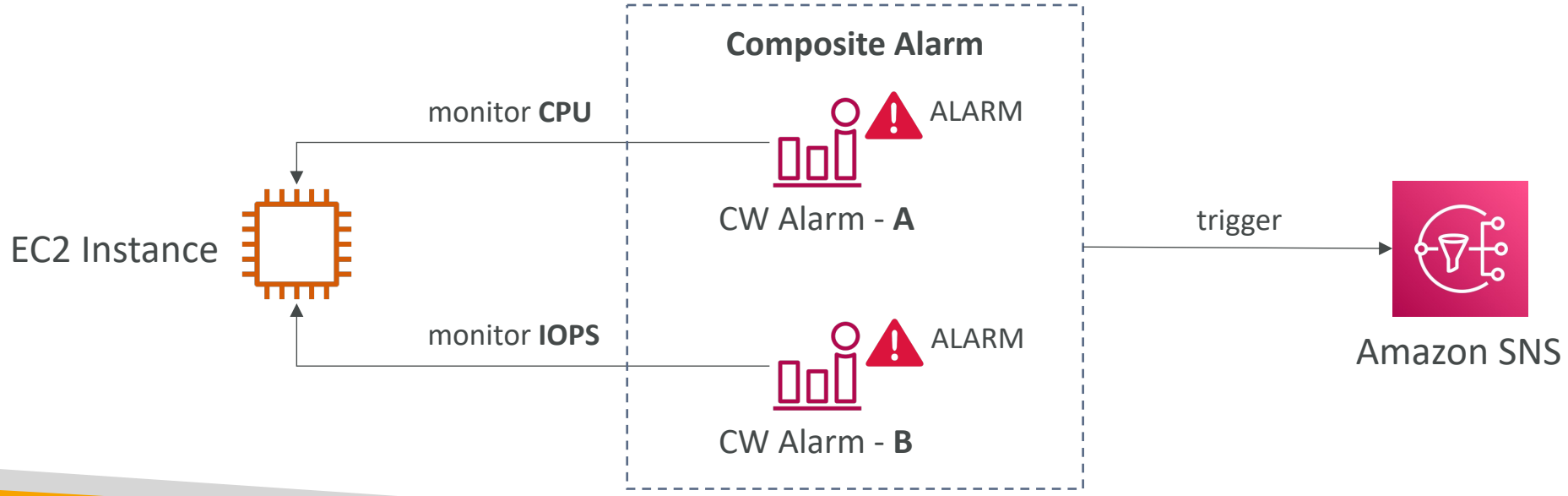


EC2 Auto Scaling

Amazon SNS

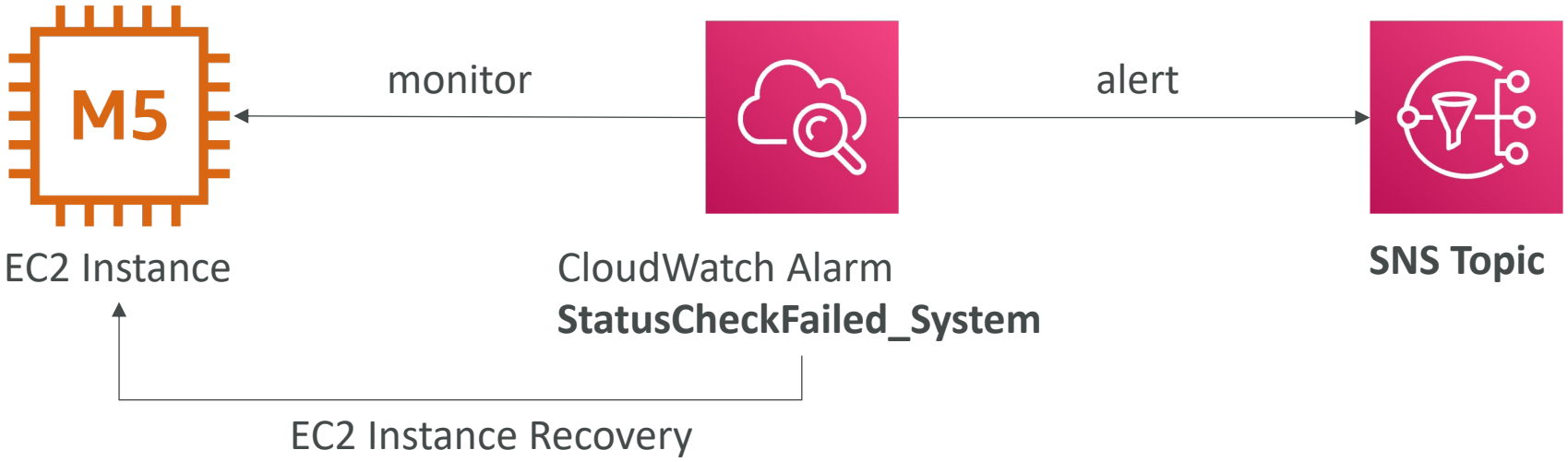
CloudWatch Alarms – Composite Alarms

- CloudWatch Alarms are on a single metric
- **Composite Alarms are monitoring the states of multiple other alarms**
- AND and OR conditions
- Helpful to reduce "alarm noise" by creating complex composite alarms



EC2 Instance Recovery

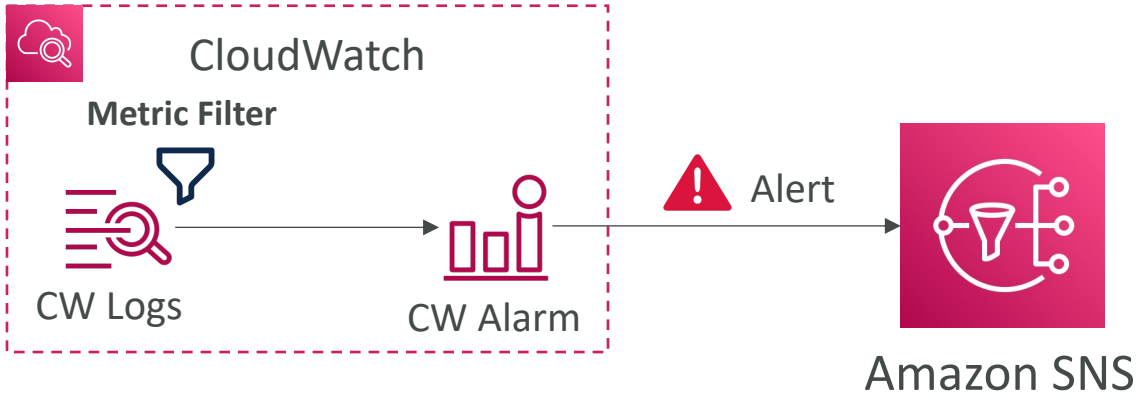
- Status Check:
  - Instance status = check the EC2 VM
  - System status = check the underlying hardware
  - Attached EBS status = check attached EBS volumes



- **Recovery:** Same Private, Public, Elastic IP, metadata, placement group

CloudWatch Alarm: good to know

- Alarms can be created based on CloudWatch Logs Metrics Filters

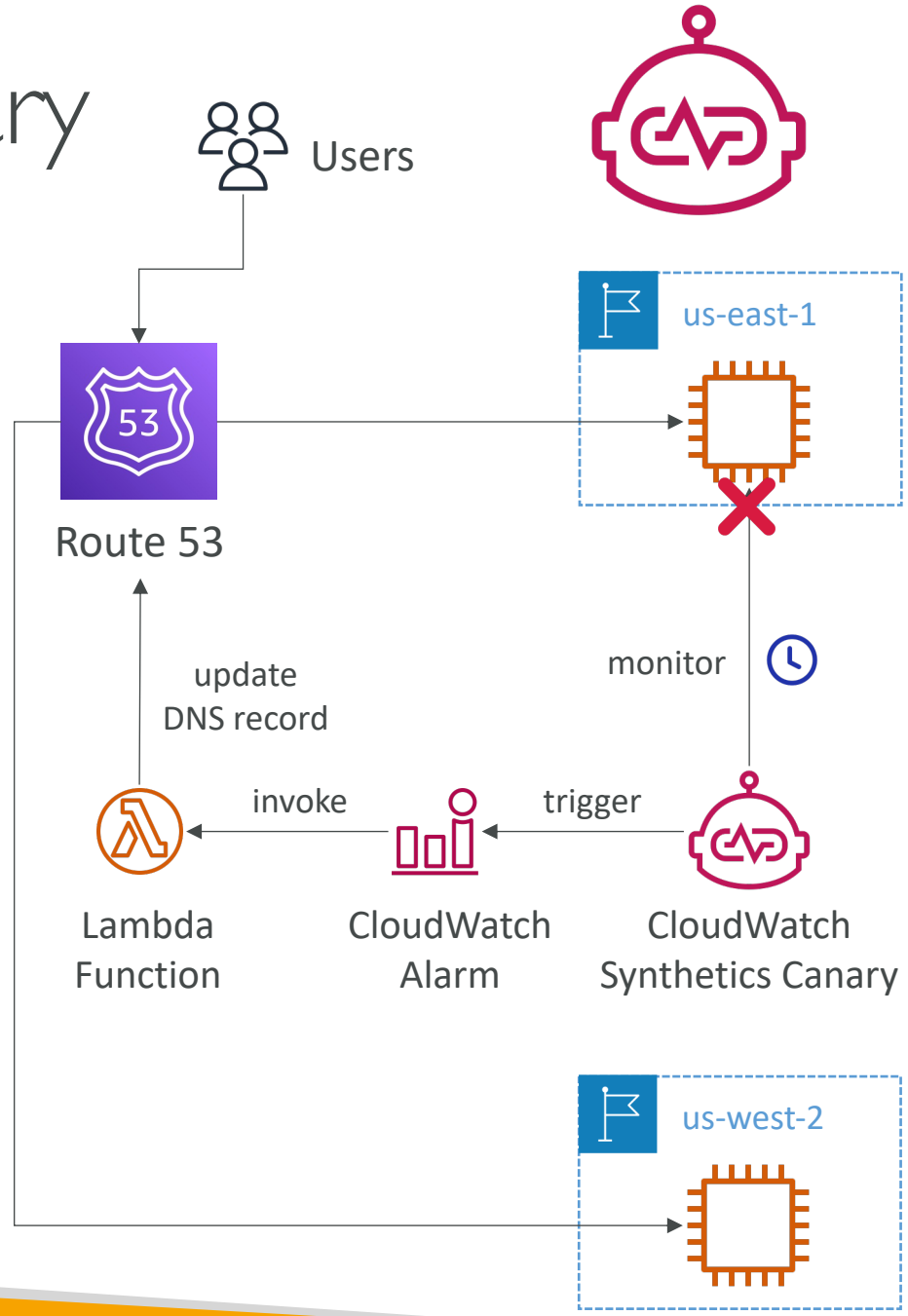


- To test alarms and notifications, set the alarm state to Alarm using CLI

```
aws cloudwatch set-alarm-state --alarm-name "myalarm" --state-value ALARM --state-reason "testing purposes"
```

CloudWatch Synthetics Canary

- Configurable script that monitor your APIs, URLs, Websites, …
- Reproduce what your customers do programmatically to find issues before customers are impacted
- Checks the availability and latency of your endpoints and can store load time data and screenshots of the UI
- Integration with CloudWatch Alarms
- Scripts written in Node.js or Python
- Programmatic access to a headless Google Chrome browser
- Can run once or on a regular schedule

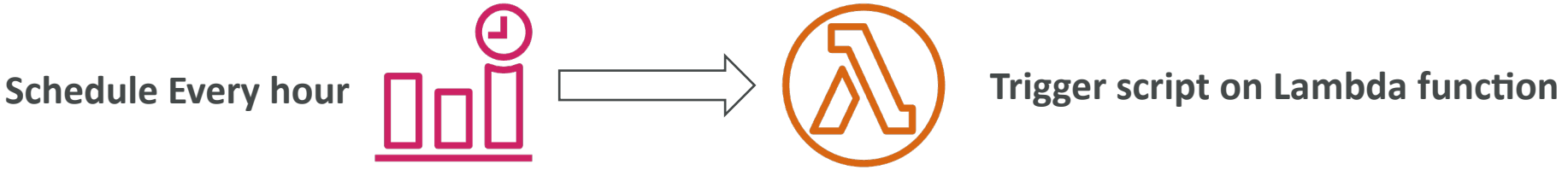


CloudWatch Synthetics Canary Blueprints

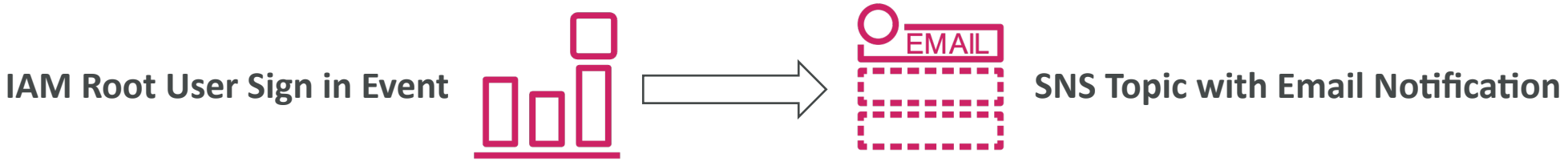
- **Heartbeat Monitor** – load URL, store screenshot and an HTTP archive file
- **API Canary** – test basic read and write functions of REST APIs
- **Broken Link Checker** – check all links inside the URL that you are testing
- **Visual Monitoring** – compare a screenshot taken during a canary run with a baseline screenshot
- **Canary Recorder** – used with CloudWatch Synthetics Recorder (record your actions on a website and automatically generates a script for that)
- **GUI Workflow Builder** – verifies that actions can be taken on your webpage (e.g., test a webpage with a login form)

Amazon EventBridge (formerly CloudWatch Events)

- Schedule: Cron jobs (scheduled scripts)

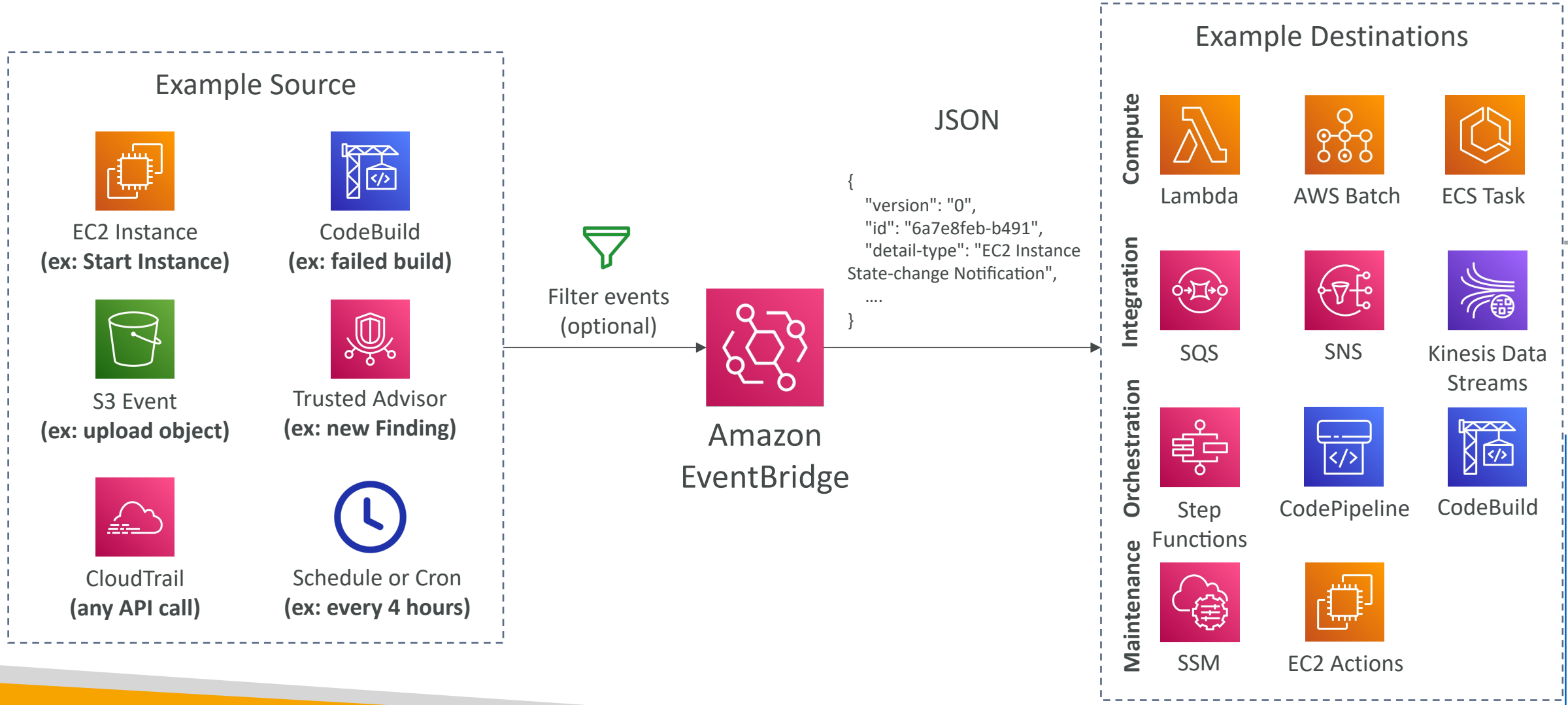


- Event Pattern: Event rules to react to a service doing something

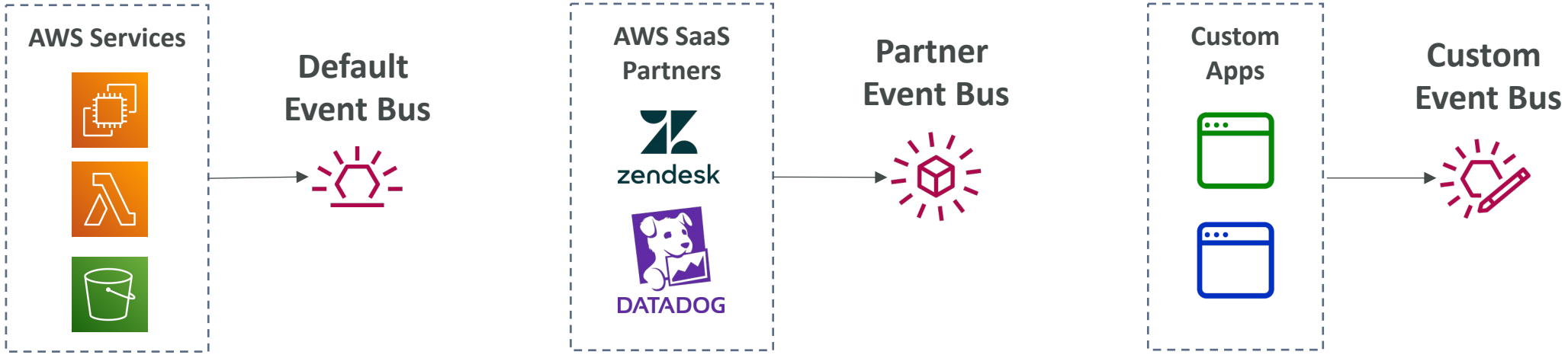


- Trigger Lambda functions, send SQS/SNS messages…

Amazon EventBridge Rules



Amazon EventBridge

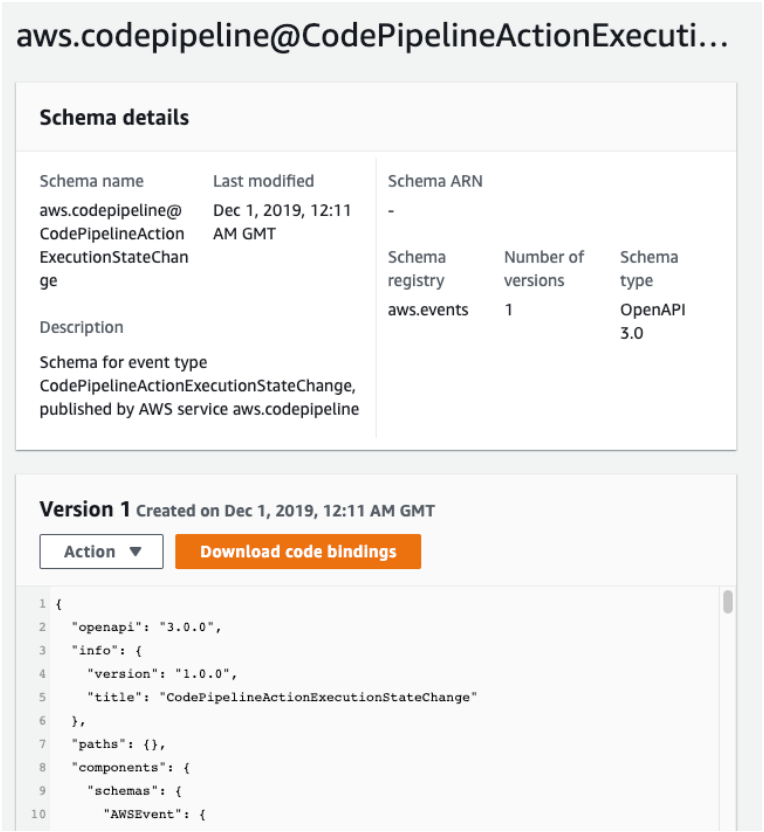


- Event buses can be accessed by other AWS accounts using Resource-based Policies

- You can **archive events** (all/filter) sent to an event bus (indefinitely or set period)
- Ability to **replay archived events**

Amazon EventBridge – Schema Registry

- EventBridge can analyze the events in your bus and infer the **schema**
- The **Schema Registry** allows you to generate code for your application, that will know in advance how data is structured in the event bus
- Schema can be versioned

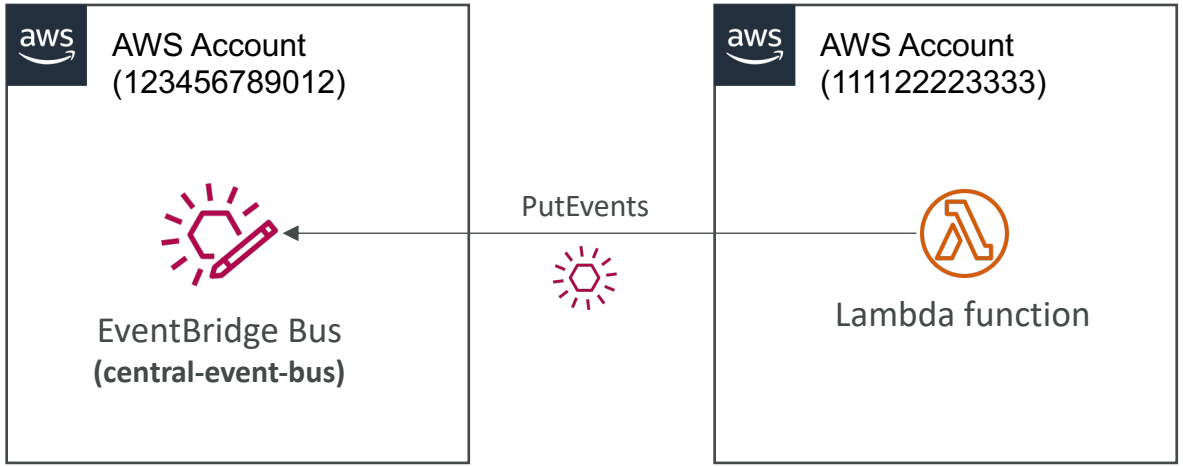


Amazon EventBridge – Resource-based Policy

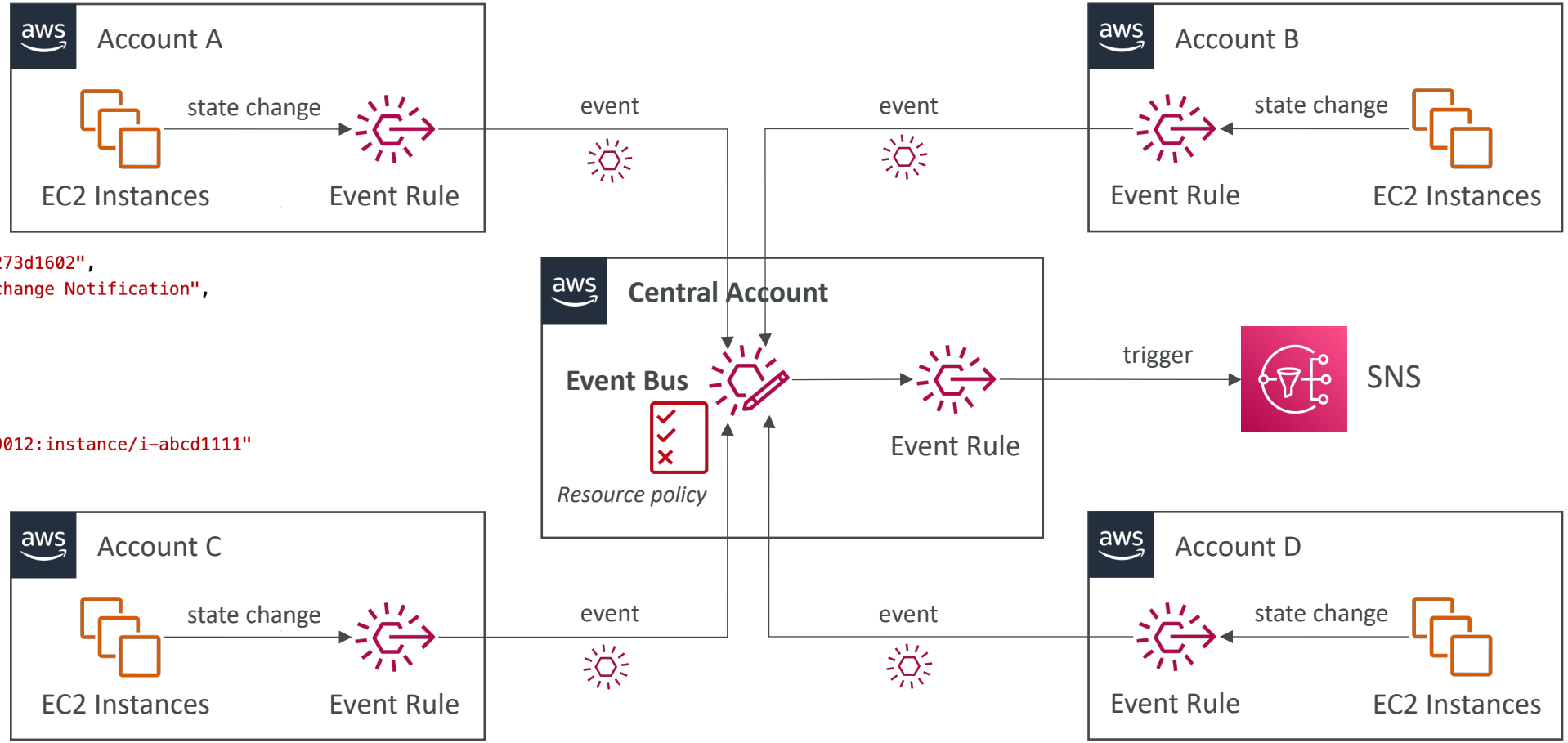
- Manage permissions for a specific Event Bus
- Example: allow/deny events from another AWS account or AWS region
- Use case: aggregate all events from your AWS Organization in a single AWS account or AWS region

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "events:PutEvents",
            "Principal": { "AWS": "111122223333" },
            "Resource": "arn:aws:events:us-east-1:123456789012:
event-bus/central-event-bus"
        }
    ]
}
```

Allow **events** from another AWS account



EventBridge – Multi-account Aggregation



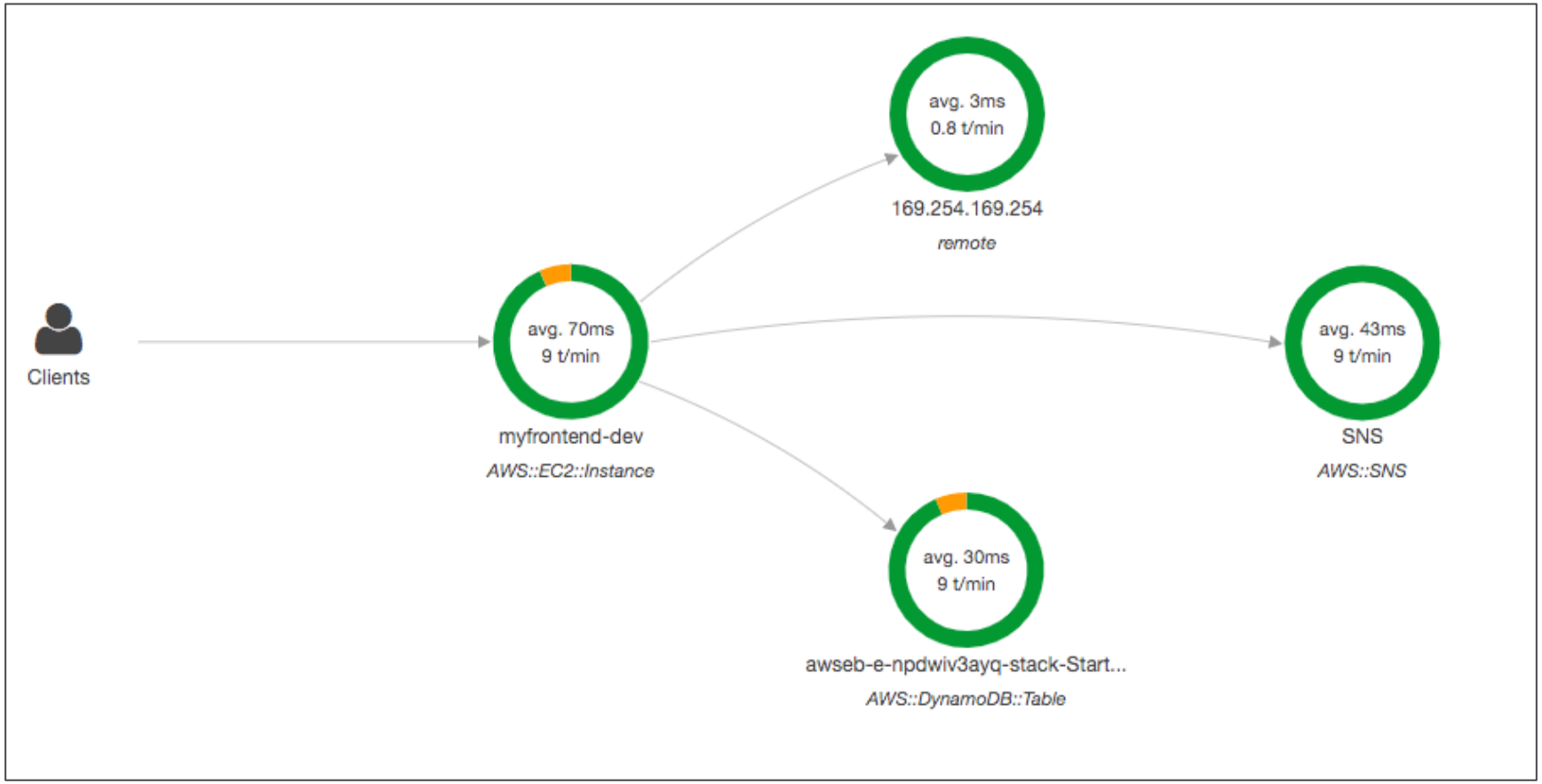

```
{
    "id": "7bf73129-1428-4cd3-a780-95db273d1602",
    "detail-type": "EC2 Instance State-change Notification",
    "source": "aws.ec2",
    "account": "123456789012",
    "time": "2021-11-11T21:29:54Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:ec2:us-east-1:123456789012:instance/i-abcd1111"
    ],
    "detail": {
        "instance-id": "i-abcd1111",
        "state": "pending"
    }
}
```

AWS X-Ray

- Debugging in Production, the good old way:
  - Test locally
  - Add log statements everywhere
  - Re-deploy in production
- Log formats differ across applications using CloudWatch and analytics is hard.
- Debugging: monolith "easy", distributed services "hard"
- No common views of your entire architecture!

- Enter… AWS X-Ray!

AWS X-Ray
Visual analysis of our applications



AWS X-Ray advantages

- Troubleshooting performance (bottlenecks)
- Understand dependencies in a microservice architecture
- Pinpoint service issues
- Review request behavior
- Find errors and exceptions
- Are we meeting time SLA?
- Where I am throttled?
- Identify users that are impacted

X-Ray compatibility

- AWS Lambda
- Elastic Beanstalk
- ECS
- ELB
- API Gateway
- EC2 Instances or any application server (even on premise)

AWS X-Ray Leverages Tracing

- Tracing is an end to end way to following a "request"
- Each component dealing with the request adds its own "trace"
- Tracing is made of segments (+ sub segments)
- Annotations can be added to traces to provide extra-information
- Ability to trace:
  - Every request
  - Sample request (as a % for example or a rate per minute)
- X-Ray Security:
  - IAM for authorization
  - KMS for encryption at rest

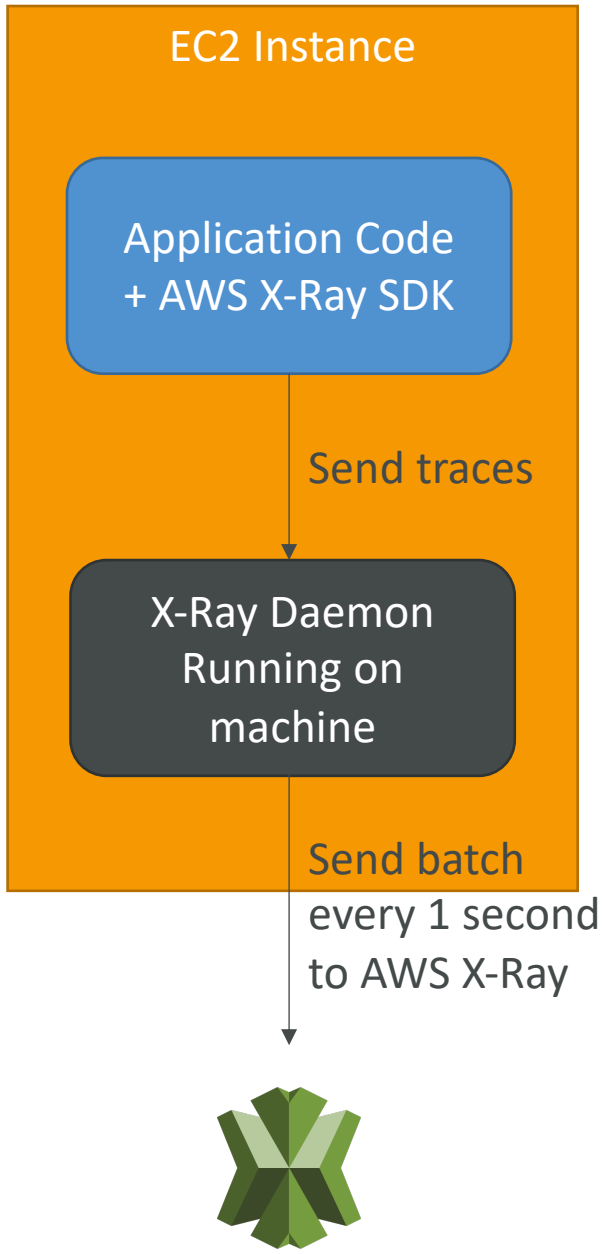
AWS X-Ray
How to enable it?

1) Your code (Java, Python, Go, Node.js, .NET) must import the AWS X-Ray SDK

- Very little code modification needed
- The application SDK will then capture:
  - Calls to AWS services
  - HTTP / HTTPS requests
  - Database Calls (MySQL, PostgreSQL, DynamoDB)
  - Queue calls (SQS)

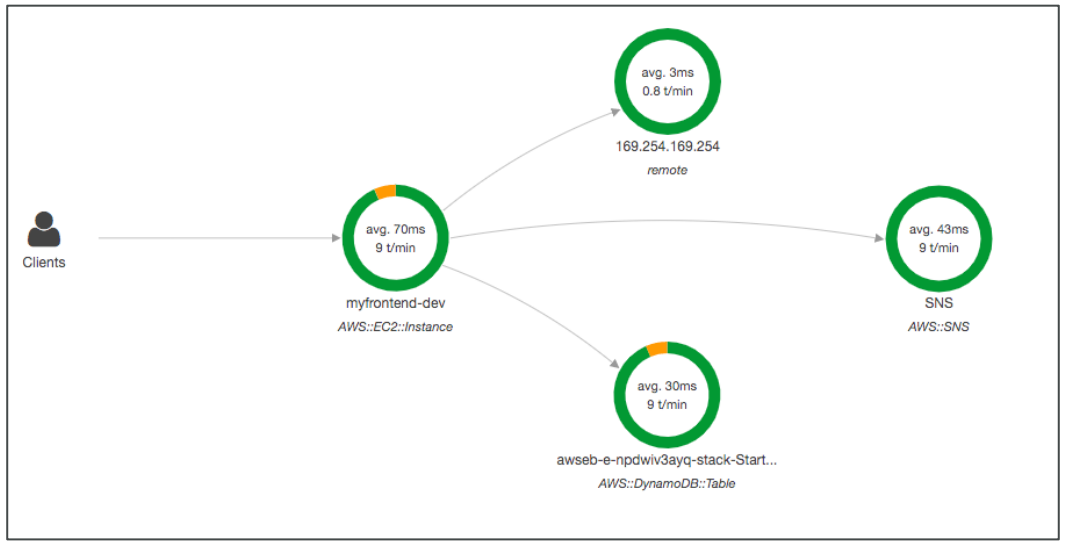
2) Install the X-Ray daemon or enable X-Ray AWS Integration

- X-Ray daemon works as a low level UDP packet interceptor (Linux / Windows / Mac…)
- AWS Lambda / other AWS services already run the X-Ray daemon for you
- Each application must have the IAM rights to write data to X-Ray



The X-Ray magic

- X-Ray service collects data from all the different services
- Service map is computed from all the segments and traces
- X-Ray is graphical, so even non technical people can help troubleshoot



AWS X-Ray Troubleshooting

- If X-Ray is not working on EC2
  - Ensure the EC2 IAM Role has the proper permissions
  - Ensure the EC2 instance is running the X-Ray Daemon

- To enable on AWS Lambda:
  - Ensure it has an IAM execution role with proper policy (AWSX-RayWriteOnlyAccess)
  - Ensure that X-Ray is imported in the code
  - Enable **Lambda X-Ray Active Tracing**

X-Ray Instrumentation in your code

- **Instrumentation** means the measure of product's performance, diagnose errors, and to write trace information.
- To instrument your application code, you use the **X-Ray SDK**
- Many SDK require only configuration changes
- You can modify your application code to customize and annotation the data that the SDK sends to X-Ray, using **interceptors**, **filters**, **handlers**, **middleware**…

Example for Node.js & Express

```
var app = express();

var AWSXRay = require('aws-xray-sdk');
app.use(AWSXRay.express.openSegment('MyApp'));

app.get('/', function (req, res) {
  res.render('index');
});

app.use(AWSXRay.express.closeSegment());
```

X-Ray Concepts

- Segments: each application / service will send them
- Subsegments: if you need more details in your segment
- Trace: segments collected together to form an end-to-end trace
- Sampling: decrease the amount of requests sent to X-Ray, reduce cost
- Annotations: Key Value pairs used to **index** traces and use with **filters**
- Metadata: Key Value pairs, **not indexed**, not used for searching

- The X-Ray daemon / agent has a config to send traces cross account:
  - make sure the IAM permissions are correct – the agent will assume the role
  - This allows to have a central account for all your application tracing

X-Ray Sampling Rules

- With sampling rules, you control the amount of data that you record
- You can modify sampling rules without changing your code

- By default, the X-Ray SDK records the first request each second, and five percent of any additional requests.
- One request per second is the *reservoir*, which ensures that at least one trace is recorded each second as long the service is serving requests.
- Five percent is the *rate* at which additional requests beyond the reservoir size are sampled.

X-Ray Custom Sampling Rules

- You can create your own rules with the reservoir and rate

Example Higher minimum rate for POSTs

- **Rule name** – `POST minimum`
- **Priority** – `100`
- **Reservoir** – `10`
- **Rate** – `0.10`
- **Service name** – `*`
- **Service type** – `*`
- **Host** – `*`
- **HTTP method** – `POST`
- **URL path** – `*`
- **Resource ARN** – `*`

Example Debugging rule to trace all requests for a problematic route

A high-priority rule applied temporarily for debugging.

- **Rule name** – `DEBUG - history updates`
- **Priority** – `1`
- **Reservoir** – `1`
- **Rate** – `1`
- **Service name** – `Scorekeep`
- **Service type** – `*`
- **Host** – `*`
- **HTTP method** – `PUT`
- **URL path** – `/history/*`
- **Resource ARN** – `*`

X-Ray Write APIs (used by the X-Ray daemon)

```
"Effect": "Allow",
"Action": [
    "xray:PutTraceSegments",
    "xray:PutTelemetryRecords",
    "xray:GetSamplingRules",
    "xray:GetSamplingTargets",
    "xray:GetSamplingStatisticSummaries"
],
"Resource": [
    "*"
]
```

arn:aws:iam::aws:policy/AWSXrayWriteOnlyAccess

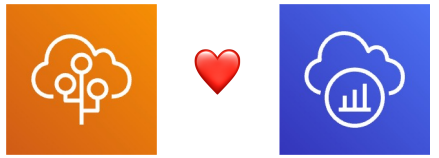
- **PutTraceSegments:** Uploads segment documents to AWS X-Ray
- **PutTelemetryRecords:** Used by the AWS X-Ray daemon to upload telemetry.
  - SegmentsReceivedCount, SegmentsRejectedCounts, BackendConnectionErrors…
- **GetSamplingRules:** Retrieve all sampling rules (to know what/when to send)
- GetSamplingTargets & GetSamplingStatisticSummaries: advanced
- The X-Ray daemon needs to have an IAM policy authorizing the correct API calls to function correctly

X-Ray Read APIs – continued

```
"Effect": "Allow",
"Action": [
    "xray:GetSamplingRules",
    "xray:GetSamplingTargets",
    "xray:GetSamplingStatisticSummaries",
    "xray:BatchGetTraces",
    "xray:GetServiceGraph",
    "xray:GetTraceGraph",
    "xray:GetTraceSummaries",
    "xray:GetGroups",
    "xray:GetGroup",
    "xray:GetTimeSeriesServiceStatistics"
],
"Resource": [
    "*"
]
```

- **GetServiceGraph:** main graph
- **BatchGetTraces:** Retrieves a list of traces specified by ID. Each trace is a collection of segment documents that originates from a single request.
- **GetTraceSummaries:** Retrieves IDs and annotations for traces available for a specified time frame using an optional filter. To get the full traces, pass the trace IDs to BatchGetTraces.
- **GetTraceGraph:** Retrieves a service graph for one or more specific trace IDs.

X-Ray with Elastic Beanstalk

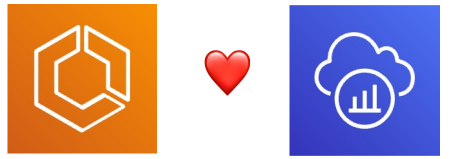


- AWS Elastic Beanstalk platforms include the X-Ray daemon
- You can run the daemon by setting an option in the Elastic Beanstalk console or with a configuration file (in .ebextensions/xray-daemon.config)

```
option_settings:
  aws:elasticbeanstalk:xray:
    XRayEnabled: true
```

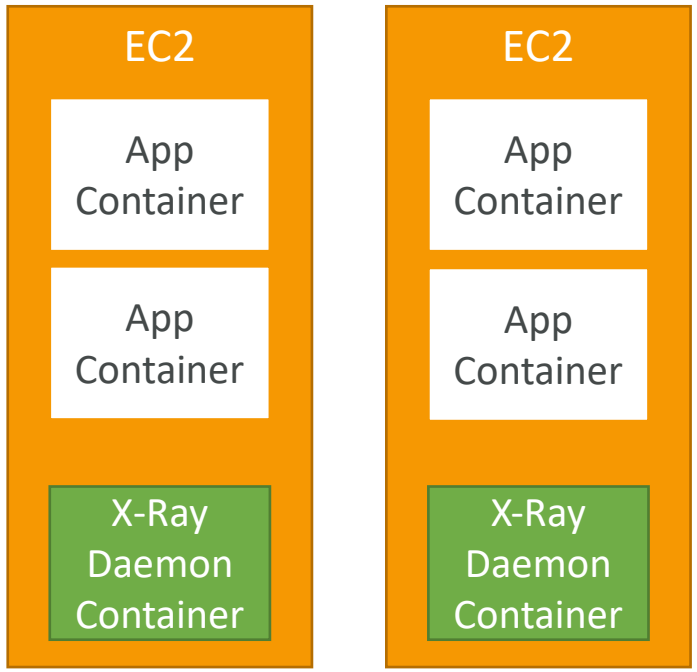
- Make sure to give your instance profile the correct IAM permissions so that the X-Ray daemon can function correctly
- Then make sure your application code is instrumented with the X-Ray SDK
- Note: The X-Ray daemon is not provided for Multicontainer Docker

ECS + X-Ray integration options



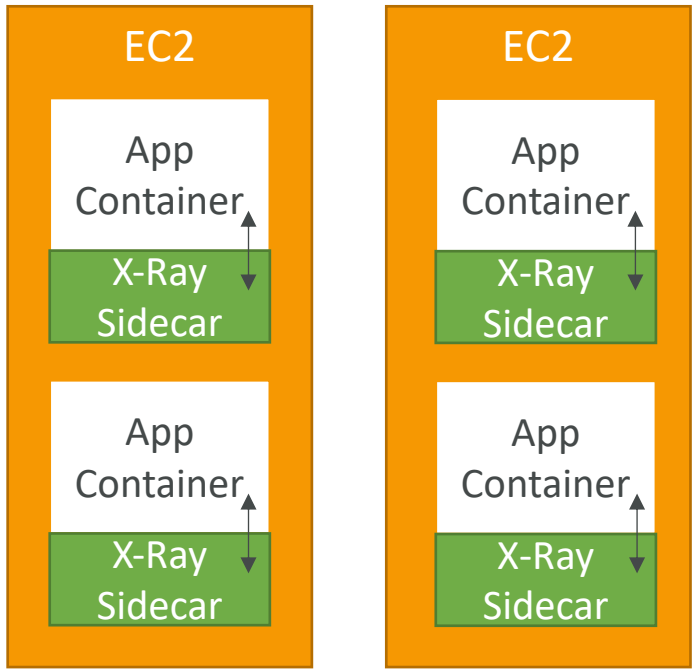
ECS Cluster

X-Ray Container as a Daemon



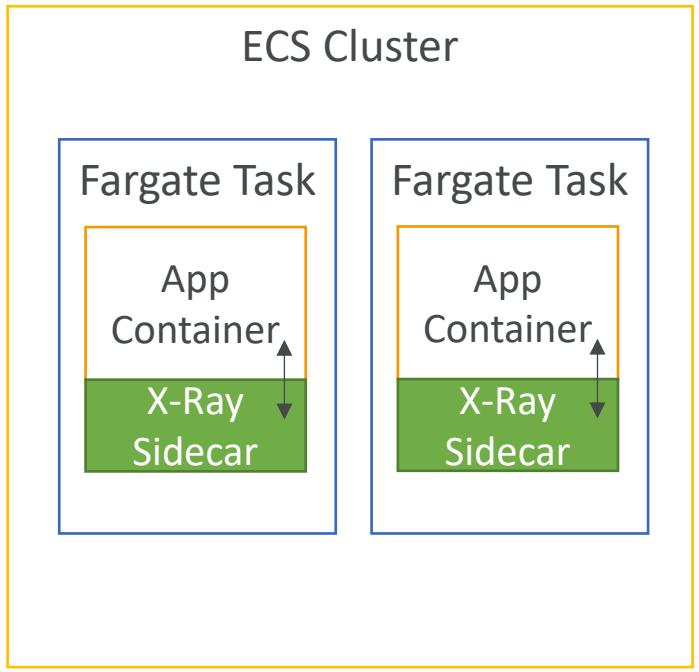
ECS Cluster

X-Ray Container as a "Side Car"



Fargate Cluster

X-Ray Container as a "Side Car"



ECS + X-Ray: Example Task Definition

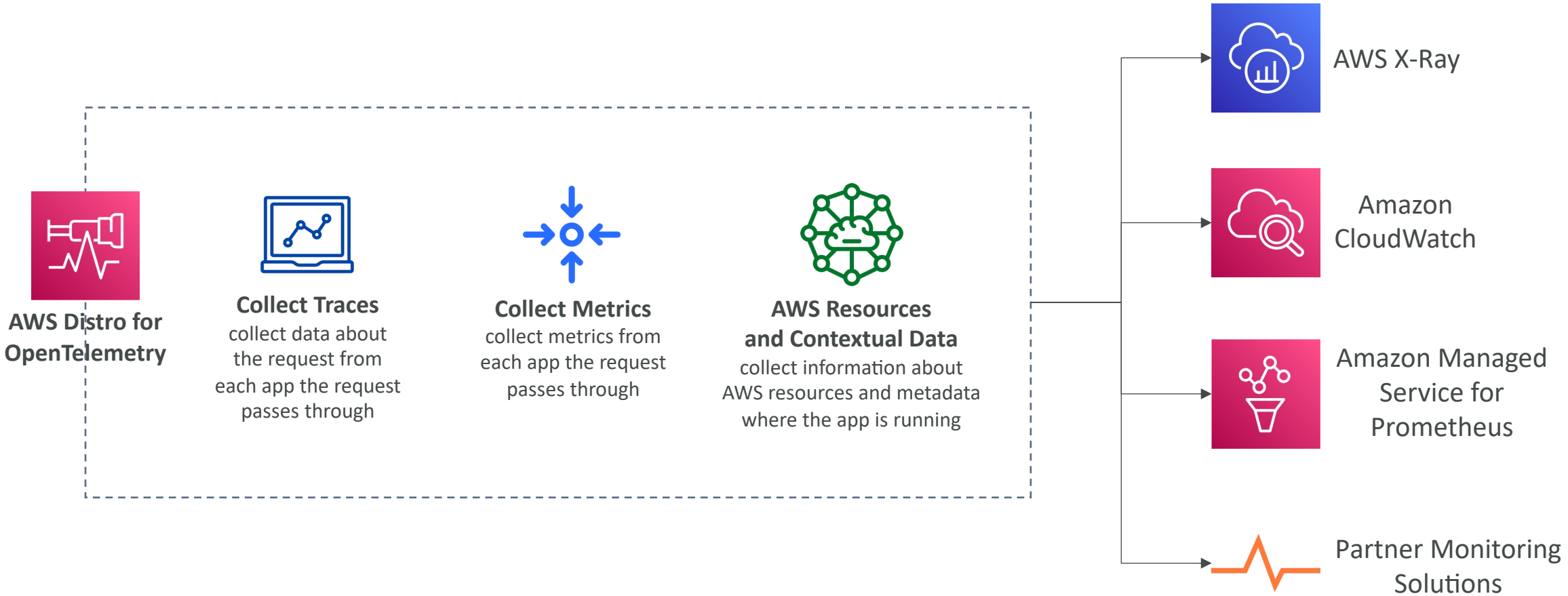
```
{
  "name": "xray-daemon",
  "image": "123456789012.dkr.ecr.us-east-2.amazonaws.com/xray-daemon",
  "cpu": 32,
  "memoryReservation": 256,
  "portMappings" : [
      {
          "hostPort": 0,
          "containerPort": 2000,
          "protocol": "udp"
      }
   ],
},
{
  "name": "scorekeep-api",
  "image": "123456789012.dkr.ecr.us-east-2.amazonaws.com/scorekeep-api",
  "cpu": 192,
  "memoryReservation": 512,
  "environment": [
      { "name" : "AWS_REGION", "value" : "us-east-2" },
      { "name" : "NOTIFICATION_TOPIC", "value" : "arn:aws:sns:us-east-2:123456789012:scorekeep-notifications" },
      { "name" : "AWS_XRAY_DAEMON_ADDRESS", "value" : "xray-daemon:2000" }
  ],
  "portMappings" : [
      {
          "hostPort": 5000,
          "containerPort": 5000
      }
  ],
  "links": [
    "xray-daemon"
  ]
}
```

https://docs.aws.amazon.com/xray/latest/devguide/xray-daemon-ecs.html#xray-daemon-ecs-build

AWS Distro for OpenTelemetry

- Secure, production-ready AWS-supported distribution of the open-source project OpenTelemetry project
- Provides a single set of APIs, libraries, agents, and collector services
- Collects distributed traces and metrics from your apps
- Collects metadata from your AWS resources and services
- **Auto-instrumentation Agents** to collect traces without changing your code
- Send traces and metrics to multiple AWS services and partner solutions
  - X-Ray, CloudWatch, Prometheus…
- Instrument your apps running on AWS (e.g., EC2, ECS, EKS, Fargate, Lambda) as well as on-premises
- **Migrate from X-Ray to AWS Distro for Temeletry if you want to standardize with open-source APIs from Telemetry or send traces to multiple destinations simultaneously**

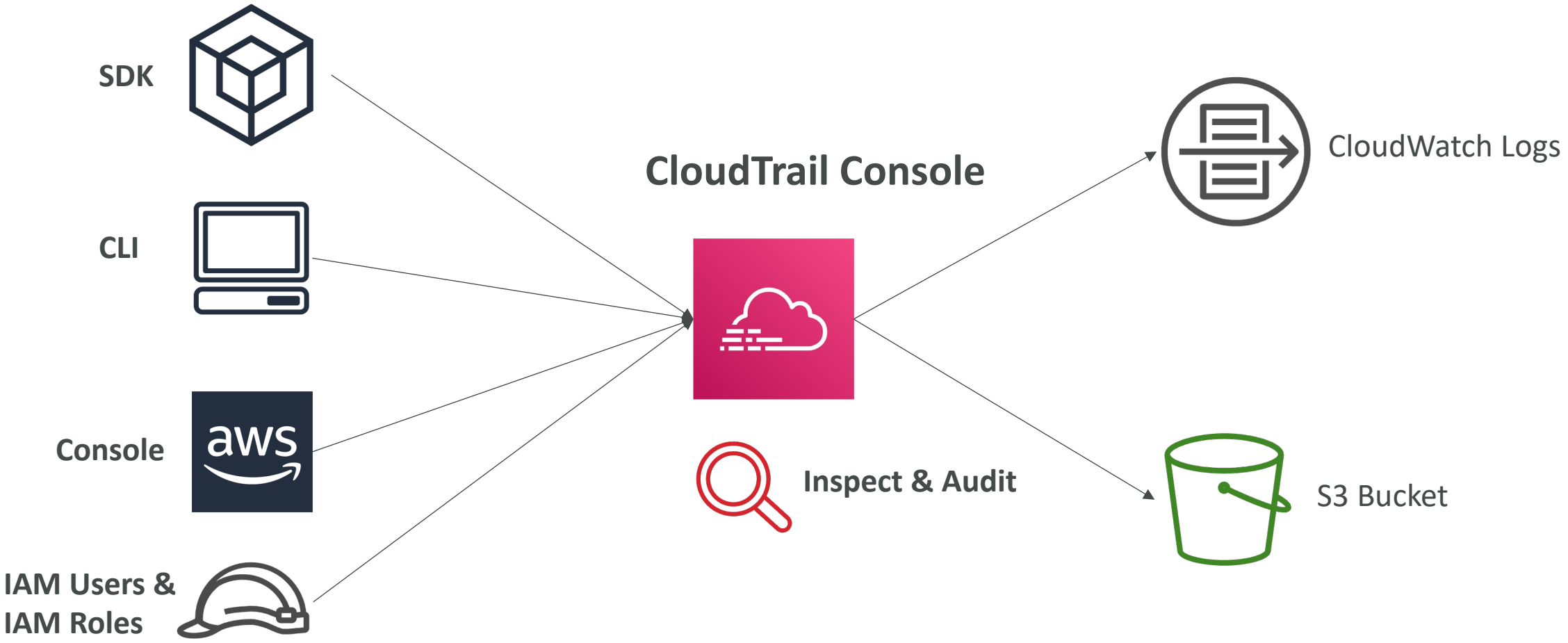
AWS Distro for OpenTelemetry



AWS CloudTrail

- Provides governance, compliance and audit for your AWS Account
- CloudTrail is enabled by default!
- Get an history of events / API calls made within your AWS Account by:
  - Console
  - SDK
  - CLI
  - AWS Services
- Can put logs from CloudTrail into CloudWatch Logs or S3
- A trail can be applied to All Regions (default) or a single Region.
- If a resource is deleted in AWS, investigate CloudTrail first!

CloudTrail Diagram



CloudTrail Events

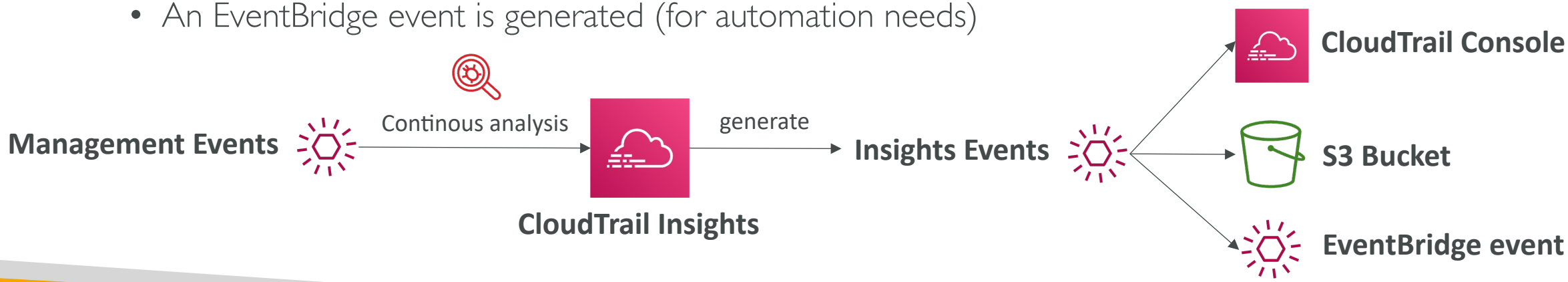
- Management Events:
  - Operations that are performed on resources in your AWS account
  - Examples:
    - Configuring security (IAM `AttachRolePolicy`)
    - Configuring rules for routing data (Amazon EC2 `CreateSubnet`)
    - Setting up logging (AWS CloudTrail `CreateTrail`)
  - By default, trails are configured to log management events.
  - Can separate **Read Events** (that don't modify resources) from **Write Events** (that may modify resources)

- Data Events:
  - By default, data events are not logged (because high volume operations)
  - Amazon S3 object-level activity (ex: `GetObject`, `DeleteObject`, `PutObject`): can separate Read and Write Events
  - AWS Lambda function execution activity (the `Invoke` API)

- CloudTrail Insights Events:
  - See next slide ☺

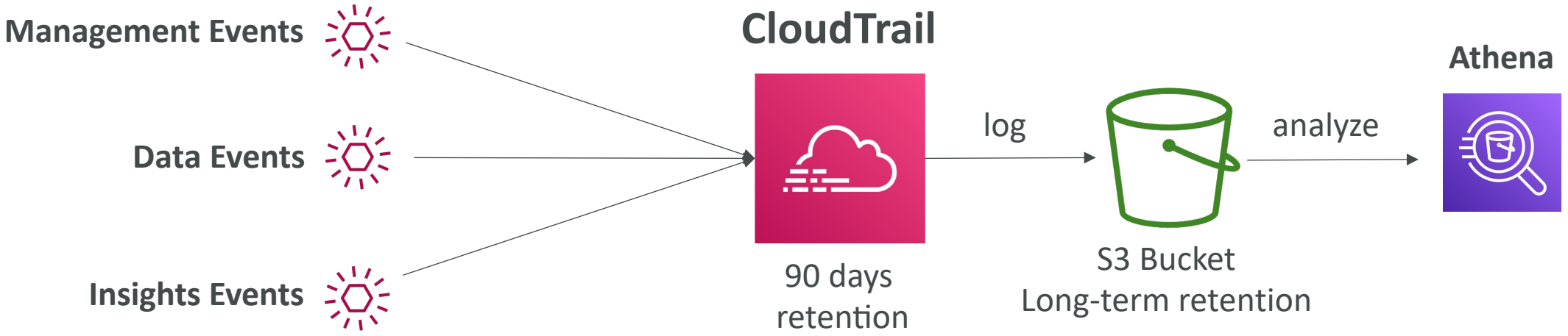
CloudTrail Insights

- Enable **CloudTrail Insights to detect unusual activity** in your account:
  - inaccurate resource provisioning
  - hitting service limits
  - Bursts of AWS IAM actions
  - Gaps in periodic maintenance activity
- CloudTrail Insights analyzes normal management events to create a baseline
- And then **continuously analyzes write events to detect unusual patterns**
  - Anomalies appear in the CloudTrail console
  - Event is sent to Amazon S3
  - An EventBridge event is generated (for automation needs)

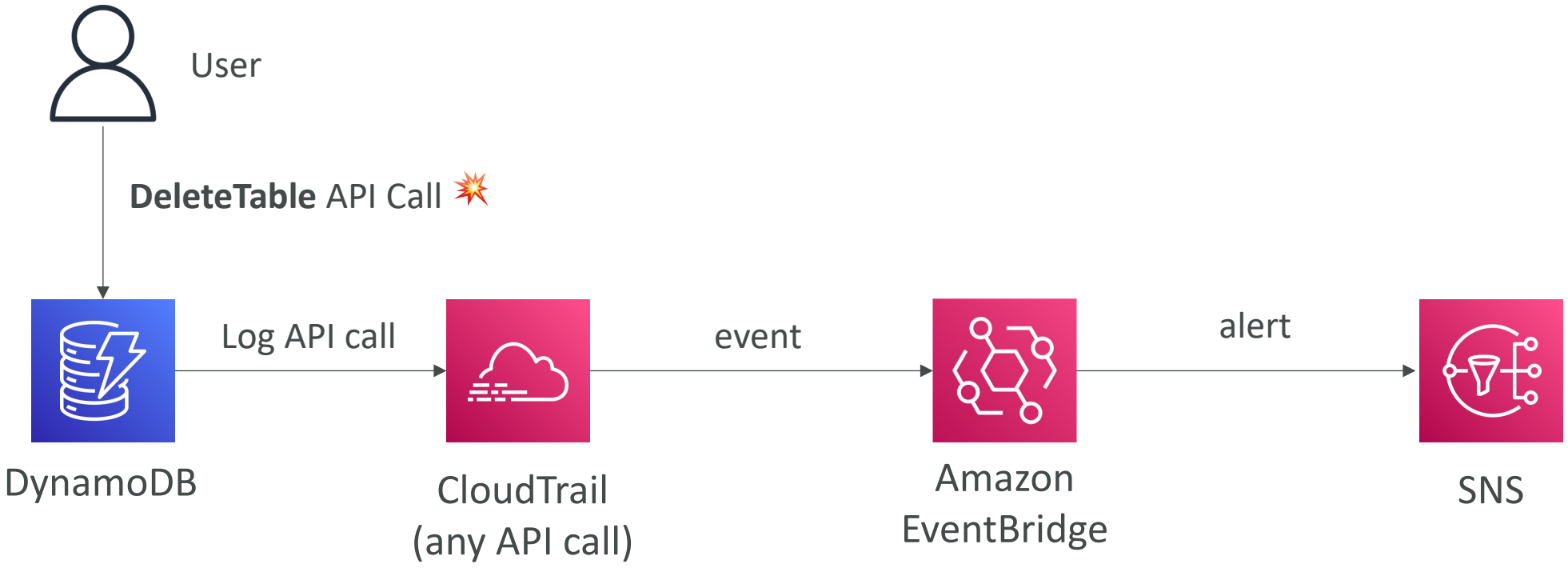


CloudTrail Events Retention

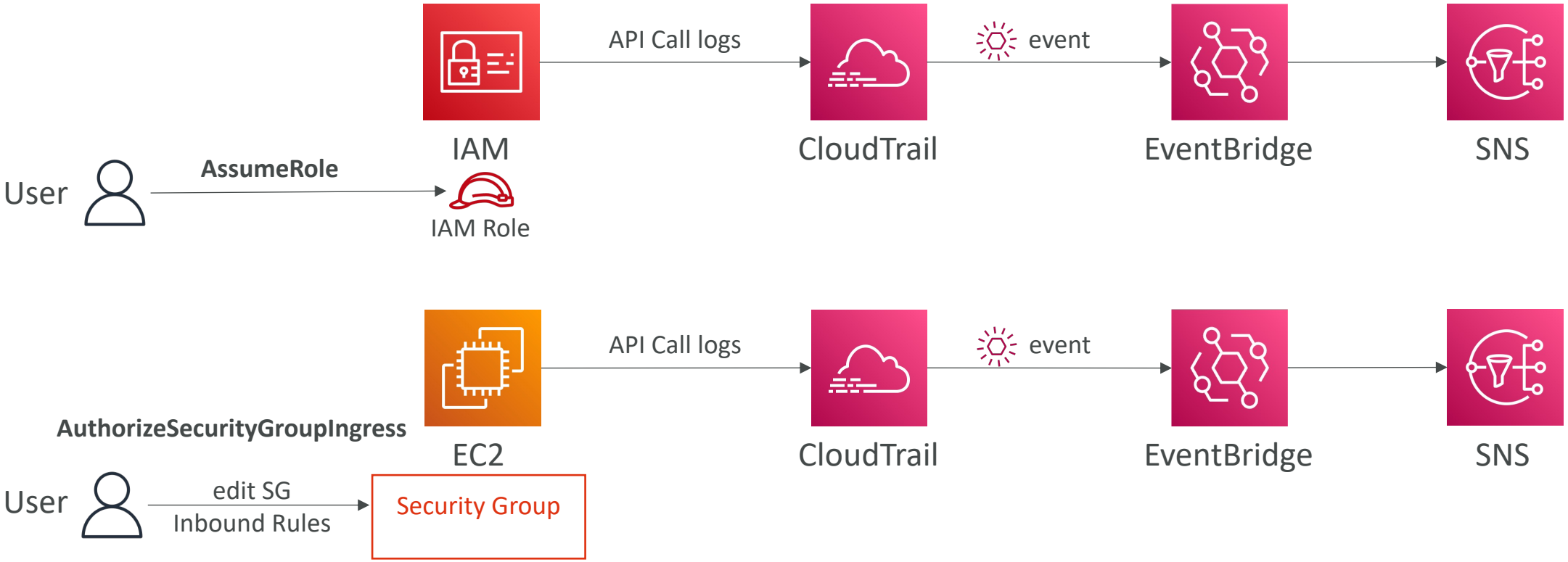
- Events are stored for 90 days in CloudTrail
- To keep events beyond this period, log them to S3 and use Athena



Amazon EventBridge – Intercept API Calls



Amazon EventBridge + CloudTrail



CloudTrail vs CloudWatch vs X-Ray

- CloudTrail:
  - Audit API calls made by users / services / AWS console
  - Useful to detect unauthorized calls or root cause of changes
- CloudWatch:
  - CloudWatch Metrics over time for monitoring
  - CloudWatch Logs for storing application log
  - CloudWatch Alarms to send notifications in case of unexpected metrics
- X-Ray:
  - Automated Trace Analysis & Central Service Map Visualization
  - Latency, Errors and Fault analysis
  - Request tracking across distributed systems

AWS Lambda

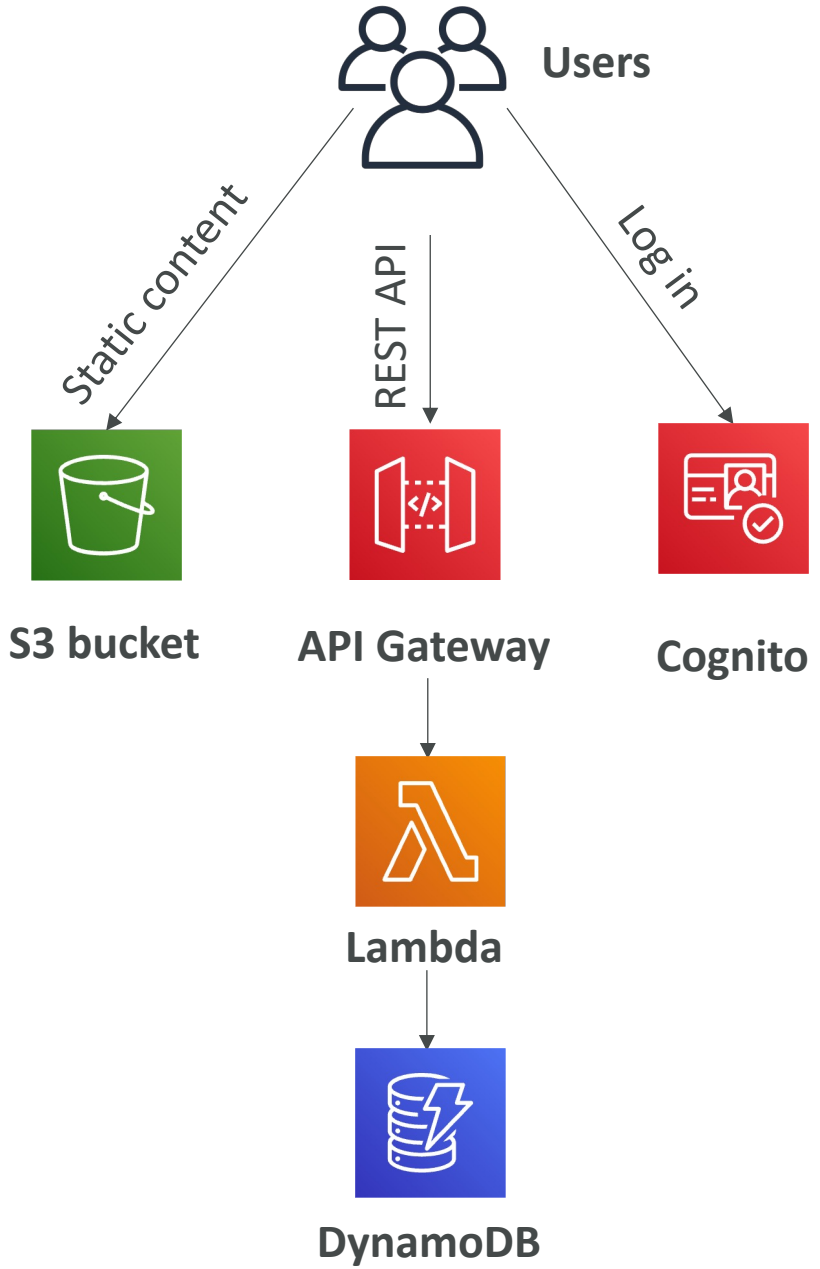
It's a serverless world

What's serverless?

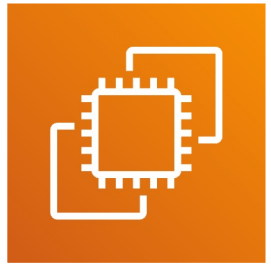
- Serverless is a new paradigm in which the developers don't have to manage servers anymore…
- They just deploy code
- They just deploy… functions !
- Initially... Serverless == FaaS (Function as a Service)
- Serverless was pioneered by AWS Lambda but now also includes anything that's managed: "databases, messaging, storage, etc."
- **Serverless does not mean there are no servers…**
  it means you just don't manage / provision / see them

Serverless in AWS

- AWS Lambda
- DynamoDB
- AWS Cognito
- AWS API Gateway
- Amazon S3
- AWS SNS & SQS
- AWS Kinesis Data Firehose
- Aurora Serverless
- Step Functions
- Fargate



Why AWS Lambda



Amazon EC2

- Virtual Servers in the Cloud
- Limited by RAM and CPU
- Continuously running
- Scaling means intervention to add / remove servers

Amazon Lambda

- Virtual **functions** – no servers to manage!
- Limited by time - **short executions**
- Run **on-demand**
- **Scaling is automated**!

Benefits of AWS Lambda

- Easy Pricing:
  - Pay per request and compute time
  - Free tier of 1,000,000 AWS Lambda requests and 400,000 GBs of compute time

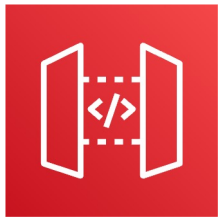
- Integrated with the whole AWS suite of services
- Integrated with many programming languages
- Easy monitoring through AWS CloudWatch
- Easy to get more resources per functions (up to 10GB of RAM!)
- Increasing RAM will also improve CPU and network!

AWS Lambda language support

- Node.js (JavaScript)
- Python
- Java
- C# (.NET Core) / Powershell
- Ruby
- Custom Runtime API (community supported, example Rust or Golang)

- Lambda Container Image
  - The container image must implement the Lambda Runtime API
  - ECS / Fargate is preferred for running arbitrary Docker images

AWS Lambda Integrations
Main ones



API Gateway

Kinesis

DynamoDB

S3

CloudFront

CloudWatch Events
EventBridge

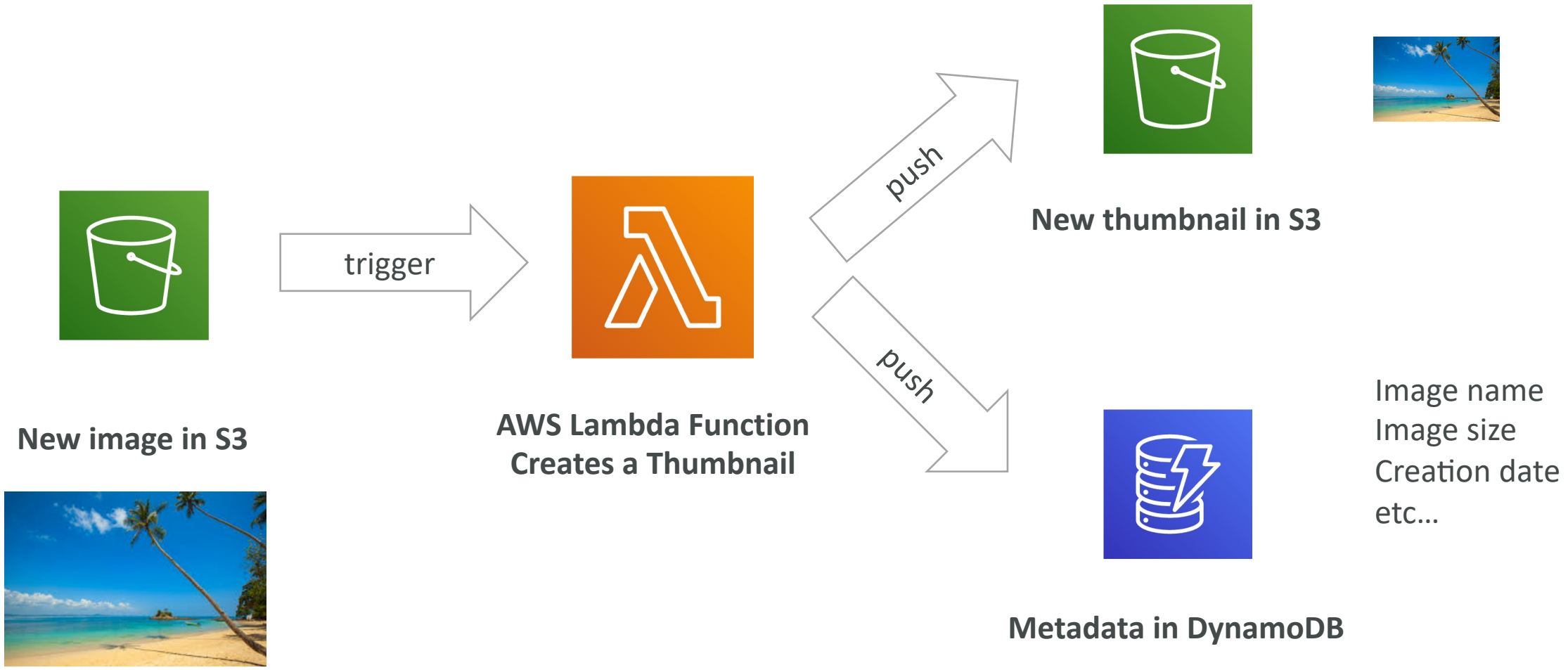
CloudWatch Logs

SNS

SQS

Cognito

Example: Serverless Thumbnail creation



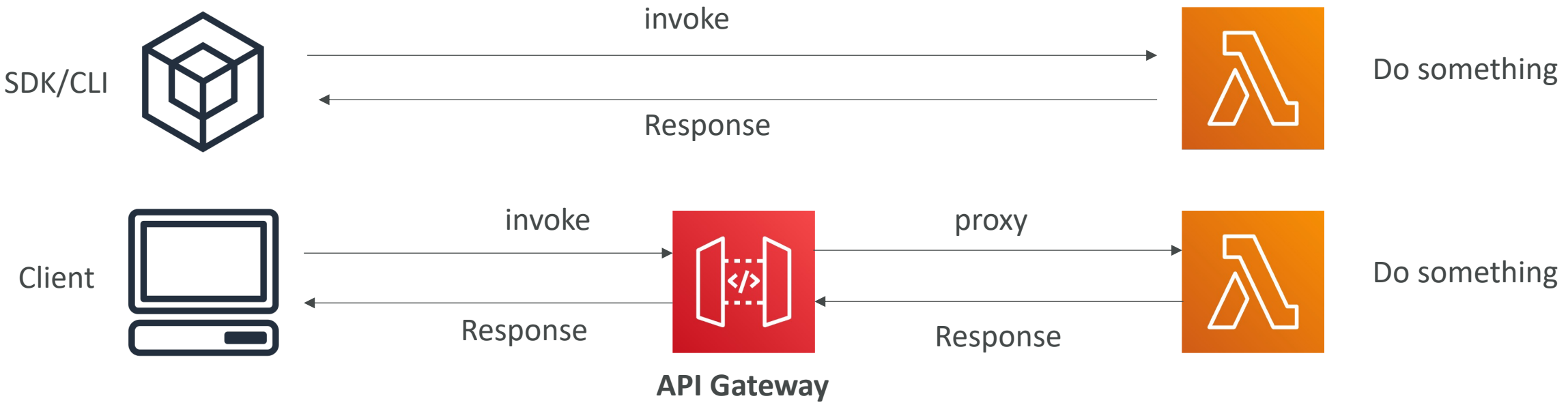
Example: Serverless CRON Job

AWS Lambda Pricing: example

- You can find overall pricing information here: https://aws.amazon.com/lambda/pricing/
- Pay per **calls**:
  - First 1,000,000 requests are free
  - $0.20 per 1 million requests thereafter ($0.0000002 per request)
- Pay per **duration**: (in increment of 1 ms)
  - 400,000 GB-seconds of compute time per month for FREE
  - == 400,000 seconds if function is 1GB RAM
  - == 3,200,000 seconds if function is 128 MB RAM
  - After that $1.00 for 600,000 GB-seconds
- It is usually very cheap to run AWS Lambda so it's very popular

Lambda – Synchronous Invocations

- Synchronous: CLI, SDK, API Gateway, Application Load Balancer
  - Results is returned right away
  - Error handling must happen client side (retries, exponential backoff, etc…)

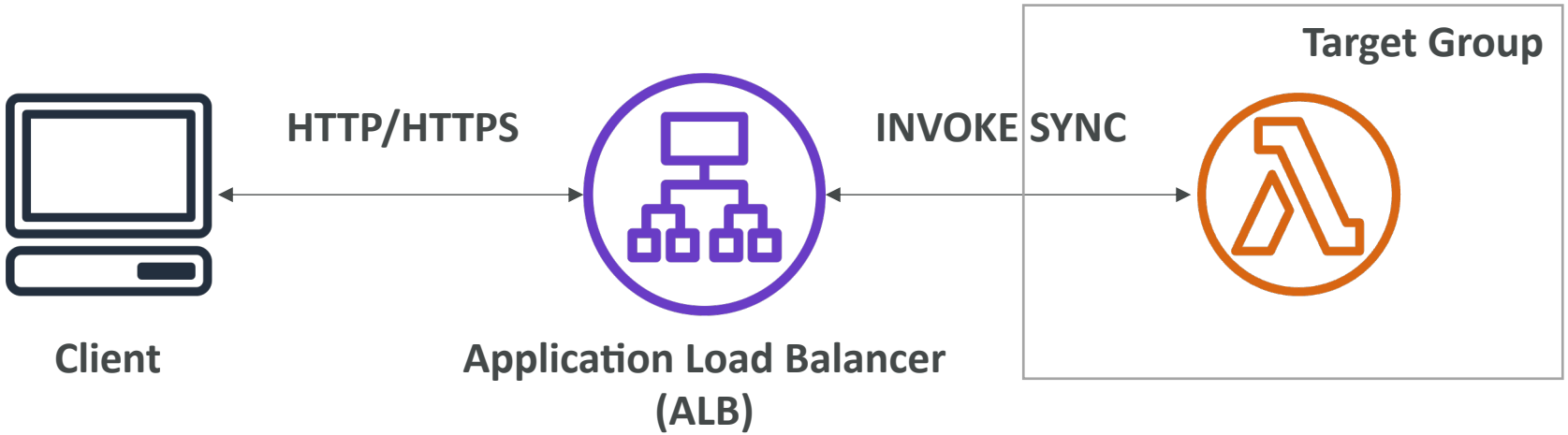


Lambda - Synchronous Invocations - Services

- User Invoked:
  - Elastic Load Balancing (Application Load Balancer)
  - Amazon API Gateway
  - Amazon CloudFront (Lambda@Edge)
  - Amazon S3 Batch
- Service Invoked:
  - Amazon Cognito
  - AWS Step Functions
- Other Services:
  - Amazon Lex
  - Amazon Alexa
  - Amazon Kinesis Data Firehose

Lambda Integration with ALB

- To expose a Lambda function as an HTTP(S) endpoint…
- You can use the **Application Load Balancer** (or an API Gateway)
- The Lambda function must be registered in a **target group**



ALB to Lambda: HTTP to JSON

Request Payload for Lambda Function

```
{
    "requestContext": {
        "elb": {
            "targetGroupArn": "arn:aws:elasticloadbalancing:us-east-2:12
            49e9d65c45c6791a"
        }
    },
    "httpMethod": "GET",
    "path": "/lambda",
    "queryStringParameters": {
        "query": "1234ABCD"
    },
    "headers": {
        "connection": "keep-alive",
        "host": "lambda-alb-123578498.us-east-2.elb.amazonaws.com",
        "user-agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWe
        Safari/537.36",
        "x-amzn-trace-id": "Root=1-5c536348-3d683b8b04734faae651f476",
        "x-forwarded-for": "72.12.164.125",
        "x-forwarded-port": "80",
        "x-forwarded-proto": "http",
    },
    "body": "",
    "isBase64Encoded": false
}
```

ELB information

HTTP Method & Path

Query String Parameters as Key/Value pairs

Headers as Key/Value pairs

Body (for POST, PUT...) & isBase64Encoded

Lambda to ALB conversions: JSON to HTTP

Response from the Lambda Function

```
{
    "statusCode": 200,
    "statusDescription": "200 OK",
    "headers": {
        "Content-Type": "text/html; charset=utf-8"
    },
    "body": "<h1>Hello world!</h1>",
    "isBase64Encoded": false
}
```

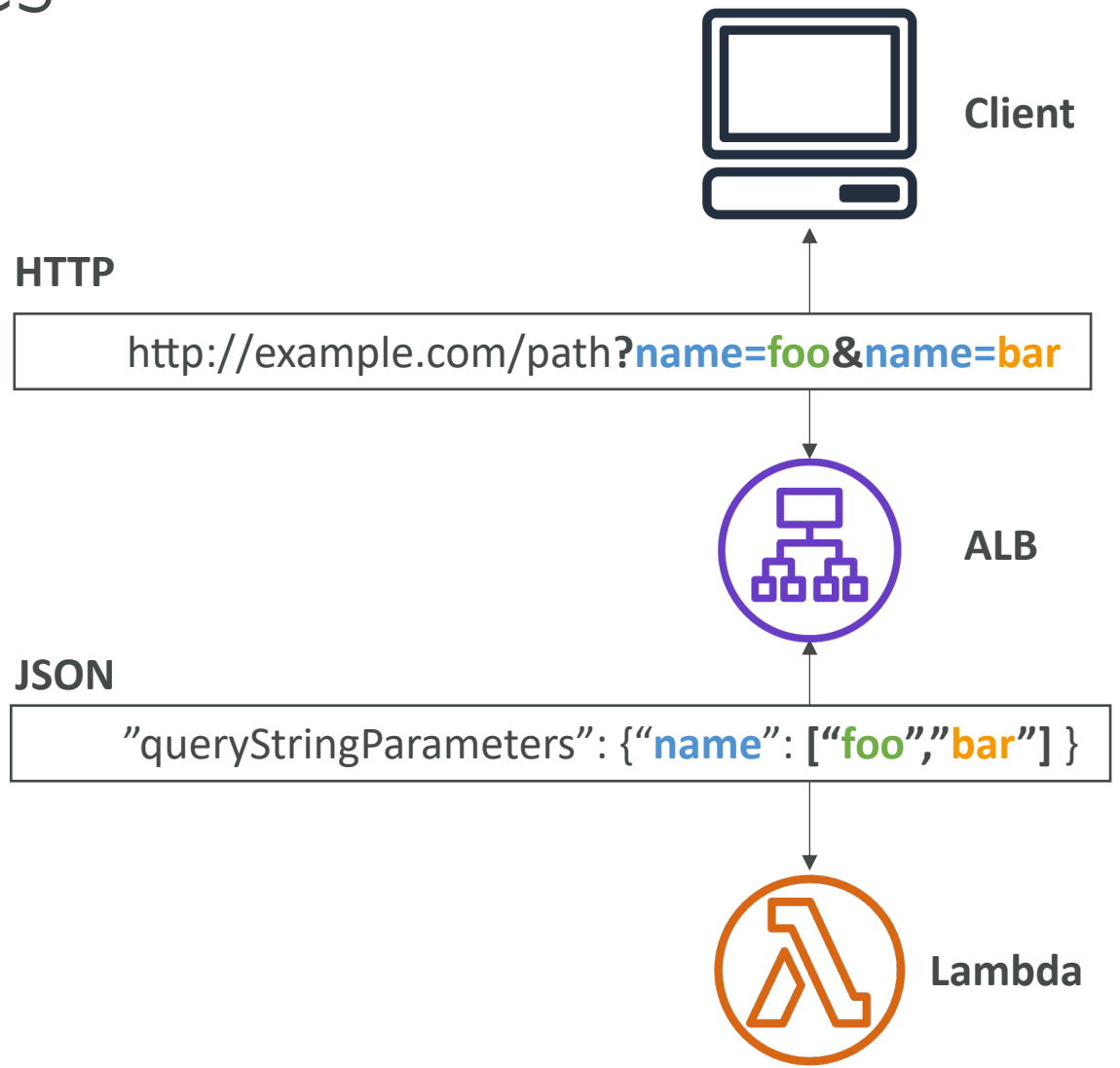
Status Code & Description

Headers as Key/Value pairs

Body & isBase64Encoded

ALB Multi-Header Values

- ALB can support multi header values (ALB setting)
- When you enable multi-value headers, **HTTP headers and query string parameters** that are sent with **multiple values** are shown **as arrays** within the AWS Lambda event and response objects.



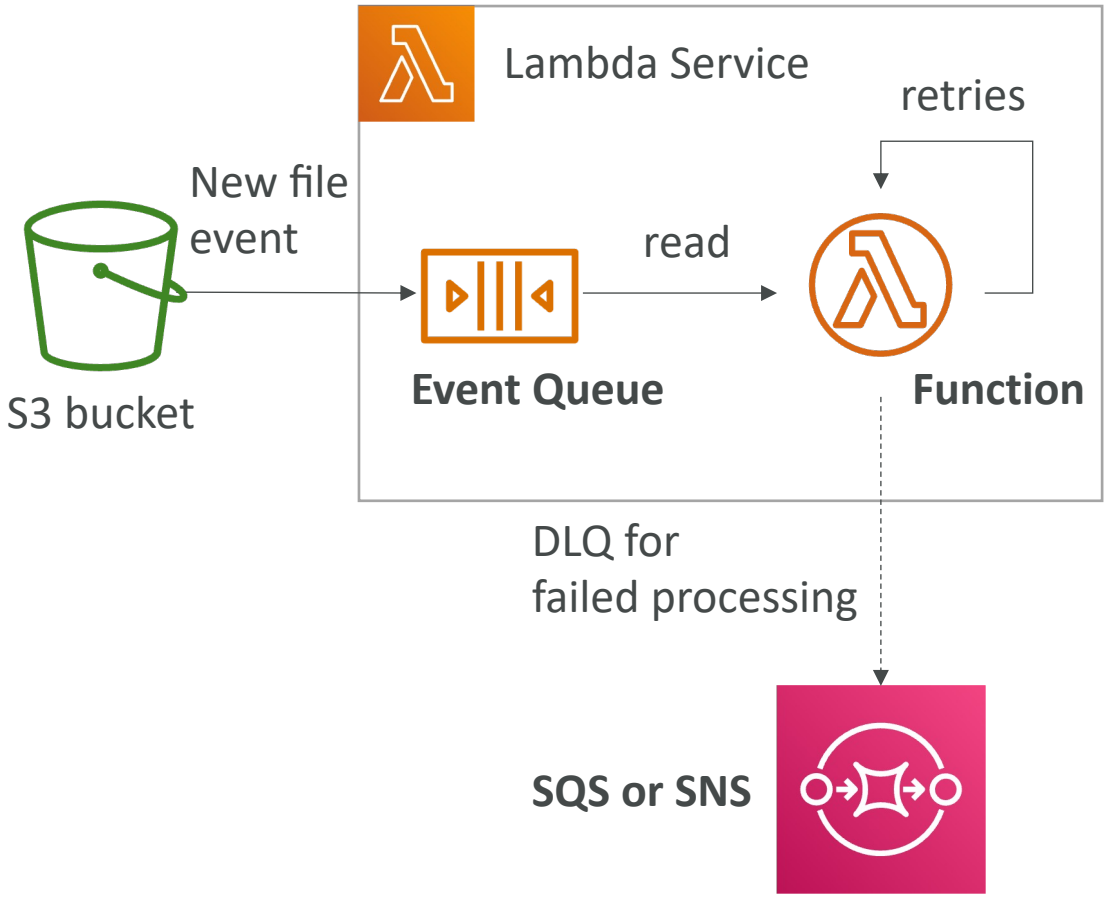
ALB + Lambda – Permissios

```
{
    "Statement": {
        "Effect": "Allow",
        "Principal": {
            "Service": "elasticloadbalancing.amazonaws.com"
        },
        "Action": "lambda:InvokeFunction",
        "Resource": "arn:aws:lambda:us-west-2:123456789012:function:alb-function"
    }
}
```

Lambda Resource Policy

Lambda – Asynchronous Invocations

- S3, SNS, CloudWatch Events…
- The events are placed in an **Event Queue**
- Lambda attempts to retry on errors
  - 3 tries total
  - 1 minute wait after 1st , then 2 minutes wait
- Make sure the processing is **idempotent** (in case of retries)
- If the function is retried, you will see **duplicate logs entries in CloudWatch Logs**
- Can define a DLQ (dead-letter queue) – **SNS or SQS** – for failed processing (need correct IAM permissions)
- Asynchronous invocations allow you to speed up the processing if you don't need to wait for the result (ex: you need 1000 files processed)



Lambda - Asynchronous Invocations - Services

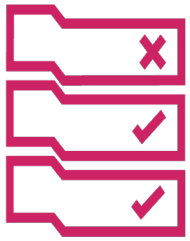
- Amazon Simple Storage Service (S3)
- Amazon Simple Notification Service (SNS)
- Amazon CloudWatch Events / EventBridge
- AWS CodeCommit (CodeCommit Trigger: new branch, new tag, new push)
- AWS CodePipeline (invoke a Lambda function during the pipeline, Lambda must callback)

----- other -----

- Amazon CloudWatch Logs (log processing)
- Amazon Simple Email Service
- AWS CloudFormation
- AWS Config
- AWS IoT
- AWS IoT Events

CloudWatch Events / EventBridge

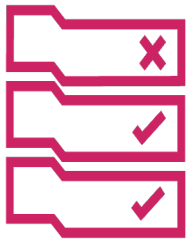
CRON or Rate EventBridge Rule



Trigger
Every 1 hour

AWS Lambda Function
Perform a task

CodePipeline EventBridge Rule



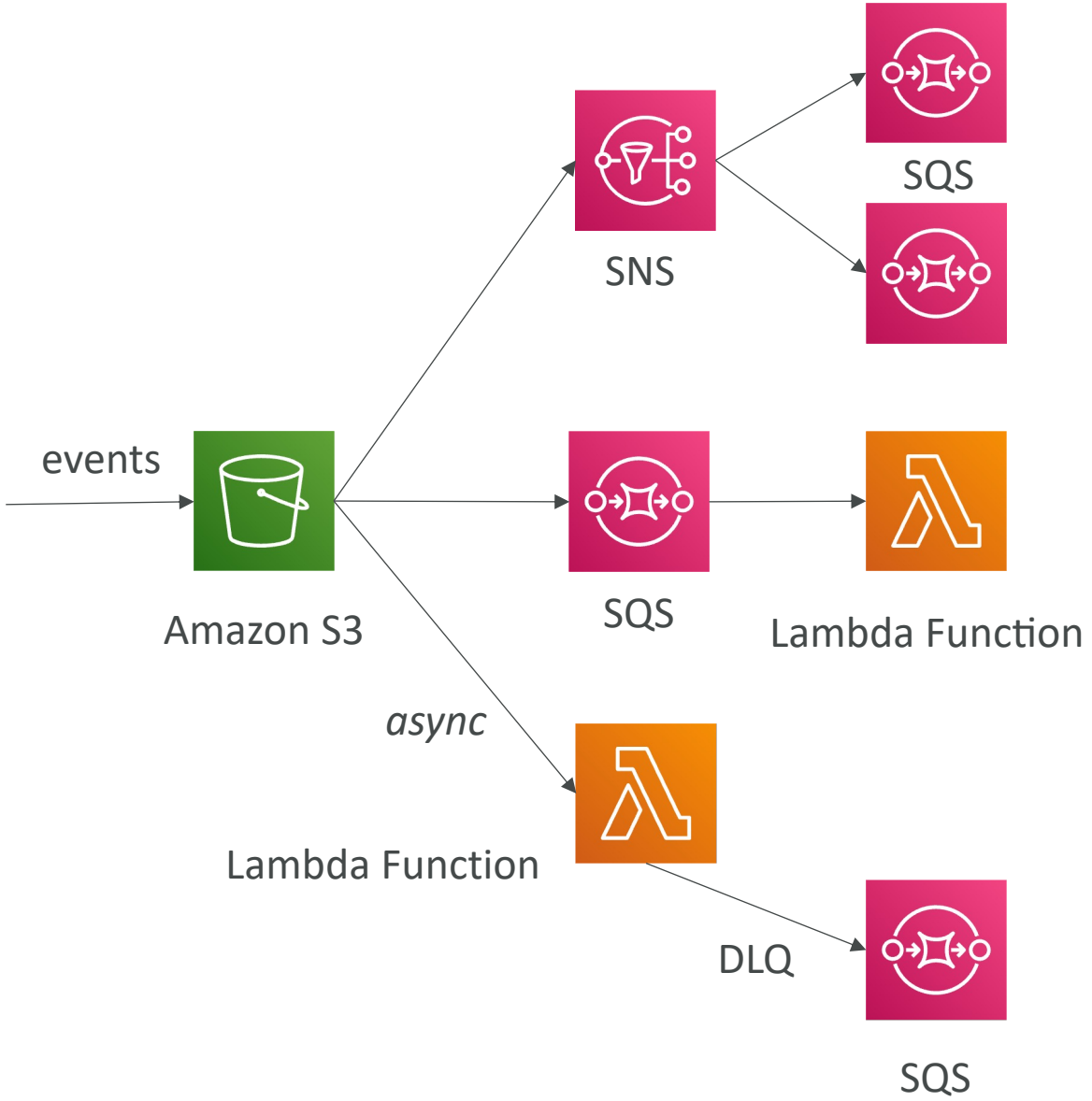
Trigger on
State Changes

AWS Lambda Function
Perform a task

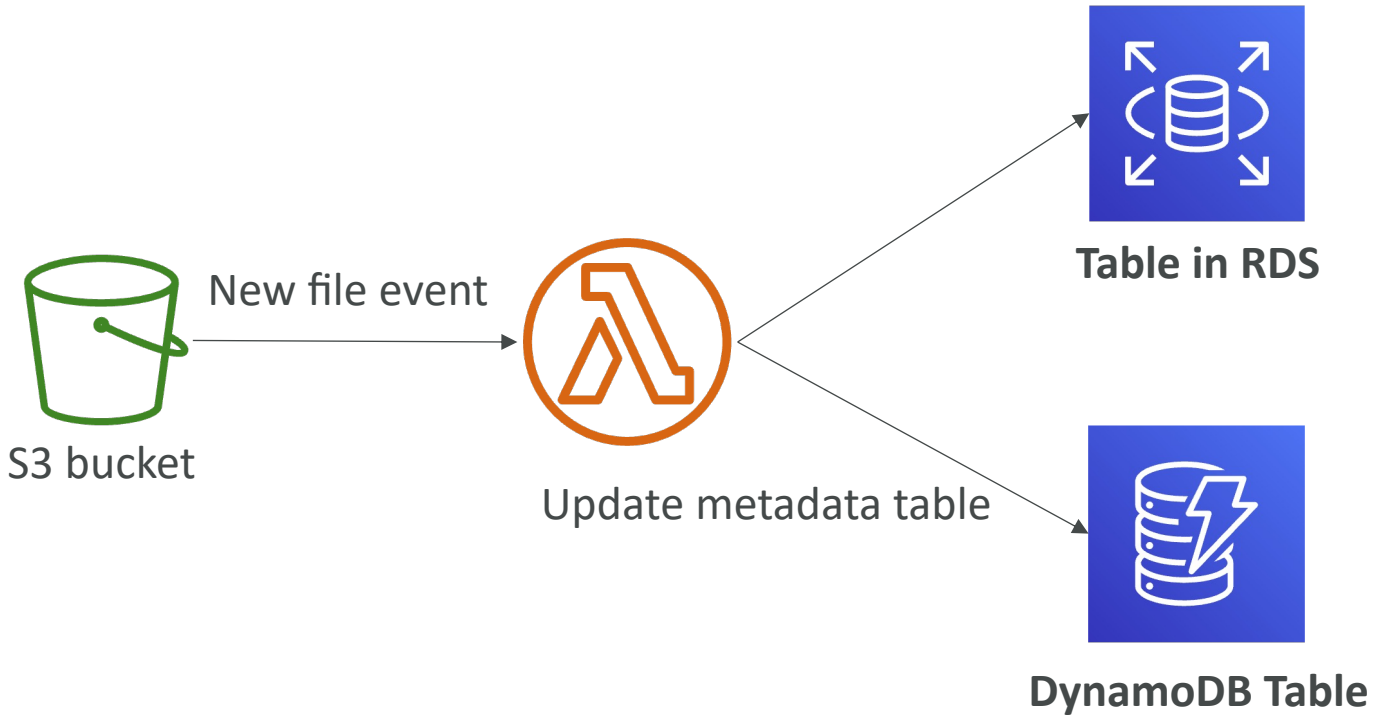
S3 Events Notifications

- S3:ObjectCreated, S3:ObjectRemoved, S3:ObjectRestore, S3:Replication…
- Object name filtering possible (*.jpg)
- Use case: generate thumbnails of images uploaded to S3

- S3 event notifications typically deliver events in seconds but can sometimes take a minute or longer
- If two writes are made to a single non-versioned object at the same time, it is possible that only a single event notification will be sent
- If you want to ensure that an event notification is sent for every successful write, you can enable versioning on your bucket.



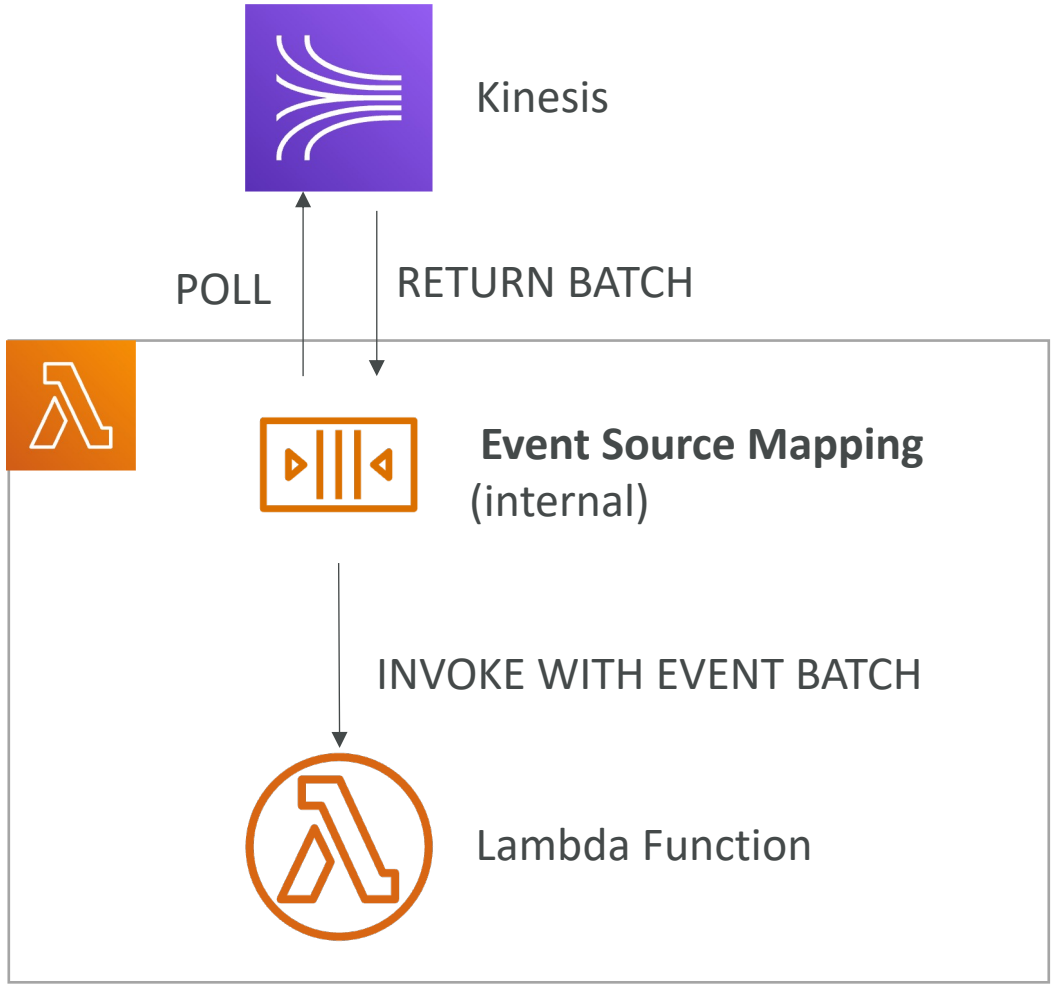
Simple S3 Event Pattern – Metadata Sync



Lambda – Event Source Mapping

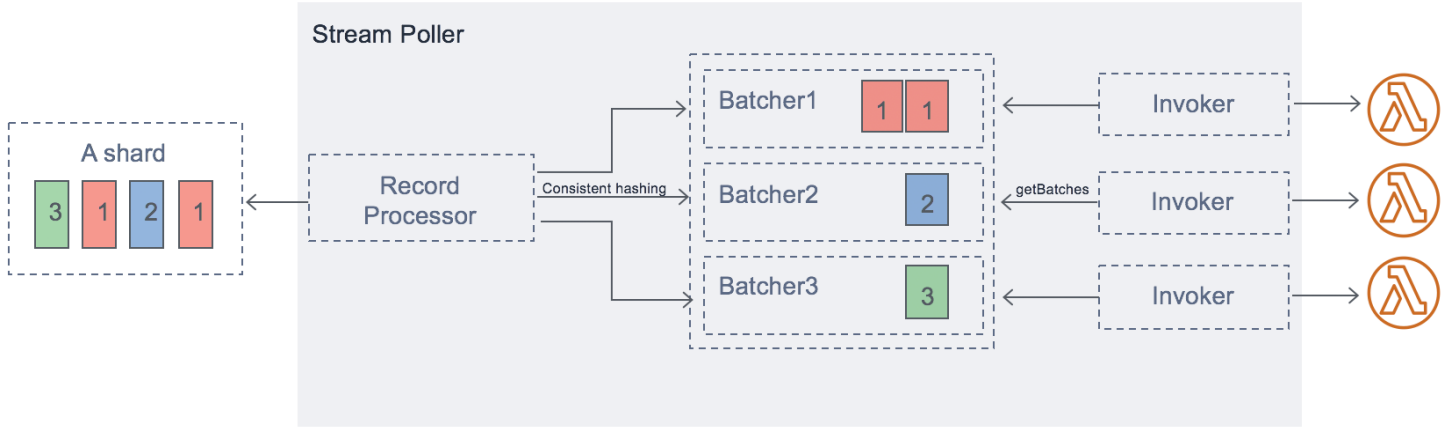
- Kinesis Data Streams
- SQS & SQS FIFO queue
- DynamoDB Streams

- Common denominator: records need to be polled from the source
- Your Lambda function is invoked synchronously



Streams & Lambda (Kinesis & DynamoDB)

- An event source mapping creates an iterator for each shard, processes items in order
- Start with new items, from the beginning or from timestamp
- Processed items aren't removed from the stream (other consumers can read them)
- Low traffic: use batch window to accumulate records before processing
- You can process multiple batches in parallel
  - up to 10 batches per shard
  - in-order processing is still guaranteed for each partition key,



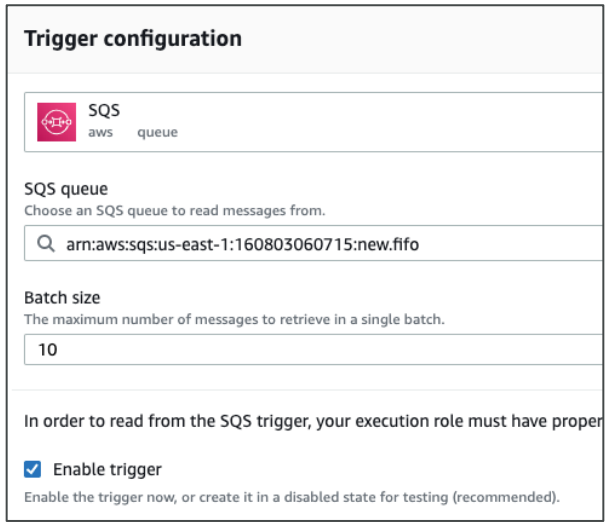
https://aws.amazon.com/blogs/compute/new-aws-lambda-scaling-controls-for-kinesis-and-dynamodb-event-sources/

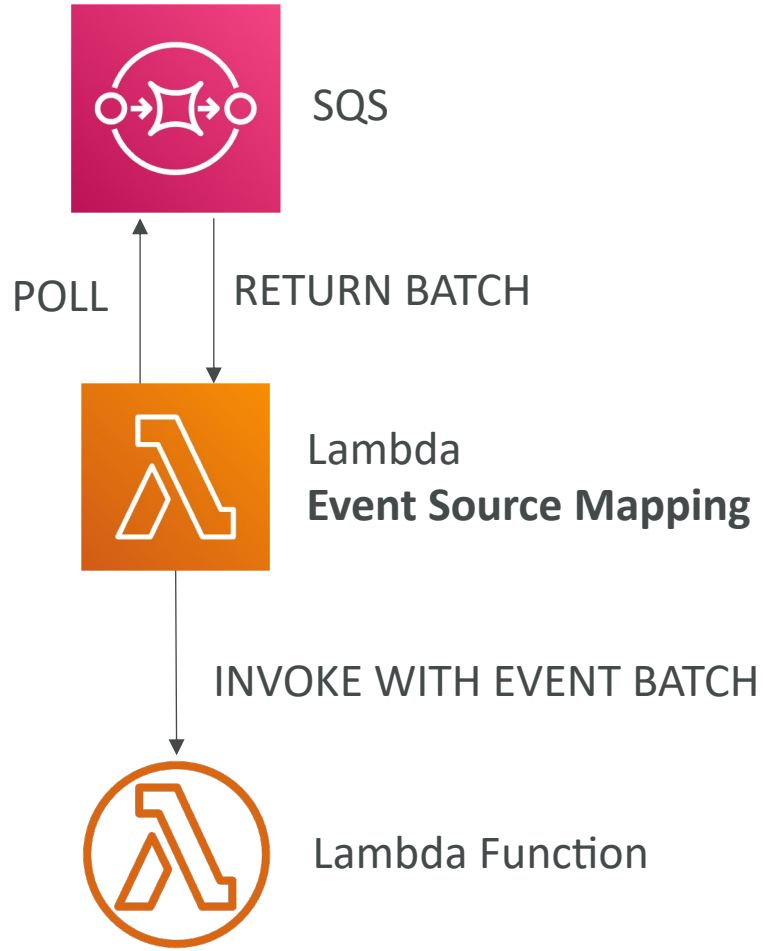
Streams & Lambda – Error Handling

- By default, if your function returns an error, the entire batch is reprocessed until the function succeeds, or the items in the batch expire.
- To ensure in-order processing, processing for the affected shard is paused until the error is resolved
- You can configure the event source mapping to:
  - discard old events
  - restrict the number of retries
  - split the batch on error (to work around Lambda timeout issues)
- Discarded events can go to a Destination

Lambda – Event Source Mapping
SQS & SQS FIFO

- Event Source Mapping will poll SQS (**Long Polling**)
- Specify **batch size** (1-10 messages)
- Recommended: Set the queue visibility timeout to 6x the timeout of your Lambda function
- To use a DLQ
  - **set-up on the SQS queue, not Lambda** (DLQ for Lambda is only for async invocations)
  - Or use a Lambda destination for failures





Queues & Lambda

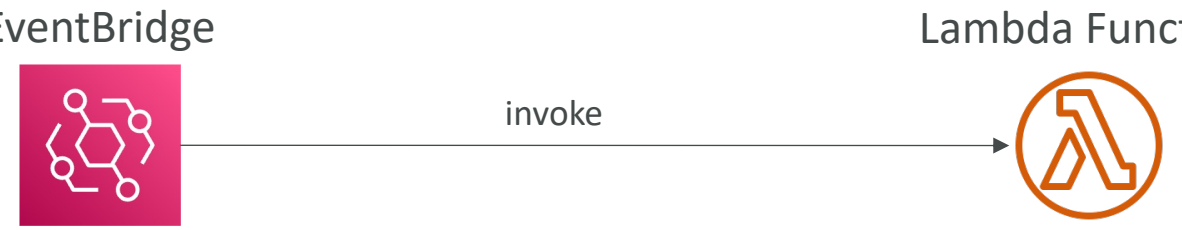
- Lambda also supports in-order processing for FIFO (first-in, first-out) queues, **scaling up to the number of active message groups.**
- For standard queues, items aren't necessarily processed in order.
- Lambda scales up to process a standard queue as quickly as possible.

- When an error occurs, batches are returned to the queue as individual items and might be processed in a different grouping than the original batch.
- Occasionally, the event source mapping might receive the same item from the queue twice, even if no function error occurred.
- Lambda deletes items from the queue after they're processed successfully.
- You can configure the source queue to send items to a dead-letter queue if they can't be processed.

Lambda Event Mapper Scaling

- Kinesis Data Streams & DynamoDB Streams:
  - One Lambda invocation per stream shard
  - If you use parallelization, up to 10 batches processed per shard simultaneously
- SQS Standard:
  - Lambda adds 60 more instances per minute to scale up
  - Up to 1000 batches of messages processed simultaneously
- SQS FIFO:
  - Messages with the same GroupID will be processed in order
  - The Lambda function scales to the number of active message groups

Lambda – Event and Context Objects




```
{
    "version": "0",
    "id": "fe8d3c65-xmpl-c5c3-2c87-81584709a377",
    "detail-type": "RDS DB Instance Event",
    "source": "aws.rds",
    "account": "123456789012",
    "time": "2020-04-28T07:20:20Z",
    "region": "us-east-2",
    "resources": [
        "arn:aws:rds:us-east-2:123456789012:db:rdz6xmpliljlb1"
    ],
    "detail": {
        "EventCategories": [ "backup" ],
        "SourceType": "DB_INSTANCE",
        "SourceArn": "arn:aws:rds:us-east-2:123456789012:db:rdz6xmpliljlb1",
        "Date": "2020-04-28T07:20:20.112Z",
        "Message": "Finished DB Instance backup",
        "SourceIdentifier": "rdz6xmpliljlb1"
    }
}
```

Event Object

```
{
    "aws_request_id": "a3de505e-f16b-42f4-b3e6-bcd2e4a73903",
    "function_name": "MyFunction-AULC3LB8Q02F",
    "invoked_function_arn": "arn:aws:lambda:us-west-2:123456789012:function:
MyFunction-AULC3LB8Q02F",
    "function_version": "$LATEST",
    "memory_limit_in_mb": "128",
    "log_group_name": "/aws/lambda/MyFunction-AULC3LB8Q02F",
    "log_stream_name": "2015/10/26/[$LATEST]c71058d852474b980f221f73402ad",
    "client_context": "None",
    ...
}
```

Context Object

Lambda – Event and Context Objects

- Event Object
  - JSON-formatted document contains data for the function to process
  - Contains information from the invoking service (e.g., EventBridge, custom, …)
  - Lambda runtime converts the event to an object (e.g., *dict* type in Python)
  - Example: input arguments, invoking service arguments, …

- Context Object
  - Provides methods and properties that provide information about the invocation, function, and runtime environment
  - Passed to your function by Lambda at runtime
  - Example: aws_request_id, function_name, memory_limit_in_mb, …

Lambda – Event and Context Objects

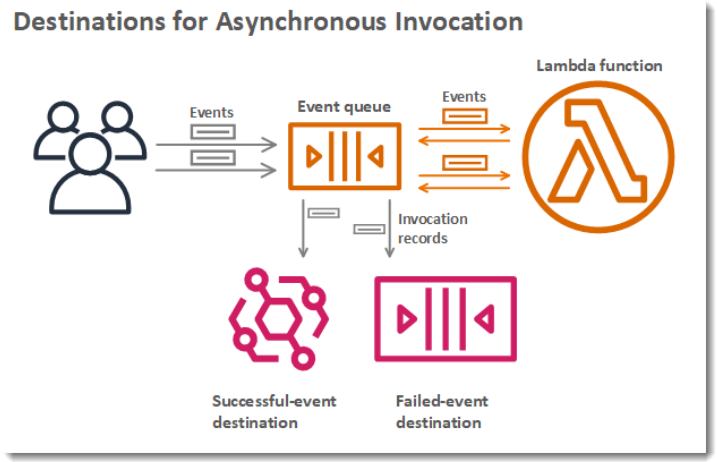
Access Event & Context Objects using Python

```
def lambda_handler(event, context):
    print("Event Source:", event.source)
    print("Event Region:", event.region)


    print("Lambda Request ID:", context.aws_request_id)
    print("Lambda function ARN:", context.function_name)
    print("Lambda function ARN:", context.invoked_function_arn)
    print("Lambda function memory limits in MB:", context.memory_limit_in_mb)
    print("CloudWatch log stream name:", context.log_stream_name)
    print("CloudWatch log group name:",  context.log_group_name)
```

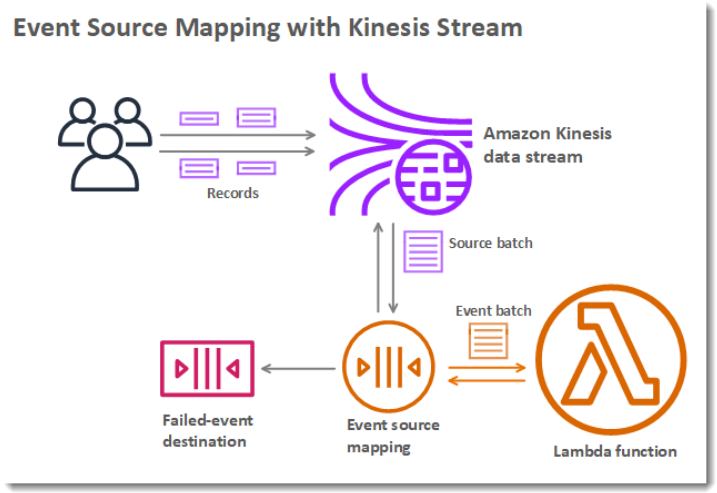
Lambda – Destinations

- Nov 2019: Can configure to send result to a **destination**
- **Asynchronous invocations** - can define destinations for successful and failed event:
  - Amazon SQS
  - Amazon SNS
  - AWS Lambda
  - Amazon EventBridge bus
- *Note: AWS recommends you use destinations instead of DLQ now (but both can be used at the same time)*

- **Event Source mapping:** for discarded event batches
  - Amazon SQS
  - Amazon SNS
- *Note: you can send events to a DLQ directly from SQS*



https://docs.aws.amazon.com/lambda/latest/dg/invocation-async.html



https://docs.aws.amazon.com/lambda/latest/dg/invocation-eventsourcemapping.html

Lambda Execution Role (IAM Role)

- Grants the Lambda function permissions to AWS services / resources
- Sample managed policies for Lambda:
  - AWSLambdaBasicExecutionRole – Upload logs to CloudWatch.
  - AWSLambdaKinesisExecutionRole – Read from Kinesis
  - AWSLambdaDynamoDBExecutionRole – Read from DynamoDB Streams
  - AWSLambdaSQSQueueExecutionRole – Read from SQS
  - AWSLambdaVPCAccessExecutionRole – Deploy Lambda function in VPC
  - AWSXRayDaemonWriteAccess – Upload trace data to X-Ray.

- When you use an event source mapping to invoke your function, Lambda uses the execution role to read event data.
- Best practice: create one Lambda Execution Role per function

Lambda Resource Based Policies

- Use resource-based policies to give other accounts and AWS services permission to use your Lambda resources
- Similar to S3 bucket policies for S3 bucket
- An IAM principal can access Lambda:
  - if the IAM policy attached to the principal authorizes it (e.g. user access)
  - OR if the resource-based policy authorizes (e.g. service access)

- When an AWS service like Amazon S3 calls your Lambda function, the resource-based policy gives it access.

Lambda Environment Variables

- Environment variable = key / value pair in "String" form
- Adjust the function behavior without updating code
- The environment variables are available to your code
- Lambda Service adds its own system environment variables as well

- Helpful to store secrets (encrypted by KMS)
- Secrets can be encrypted by the Lambda service key, or your own CMK

Lambda Logging & Monitoring

- CloudWatch Logs:
  - AWS Lambda execution logs are stored in AWS CloudWatch Logs
  - Make sure your AWS Lambda function has an execution role with an IAM policy that authorizes writes to CloudWatch Logs
- CloudWatch Metrics:
  - AWS Lambda metrics are displayed in AWS CloudWatch Metrics
  - Invocations, Durations, Concurrent Executions
  - Error count, Success Rates, Throttles
  - Async Delivery Failures
  - Iterator Age (Kinesis & DynamoDB Streams)

Lambda Tracing with X-Ray

- Enable in Lambda configuration (**Active Tracing**)
- Runs the X-Ray daemon for you
- Use AWS X-Ray SDK in Code
- Ensure Lambda Function has a correct IAM Execution Role
  - The managed policy is called AWSXRayDaemonWriteAccess
- Environment variables to communicate with X-Ray
  - **_X_AMZN_TRACE_ID**: contains the tracing header
  - **AWS_XRAY_CONTEXT_MISSING**: by default, LOG_ERROR
  - **AWS_XRAY_DAEMON_ADDRESS**: the X-Ray Daemon IP_ADDRESS:PORT

Customization At The Edge

- Many modern applications execute some form of the logic at the edge
- **Edge Function:**
  - A code that you write and attach to CloudFront distributions
  - Runs close to your users to minimize latency
- CloudFront provides two types: **CloudFront Functions** & **Lambda@Edge**
- You don't have to manage any servers, deployed globally

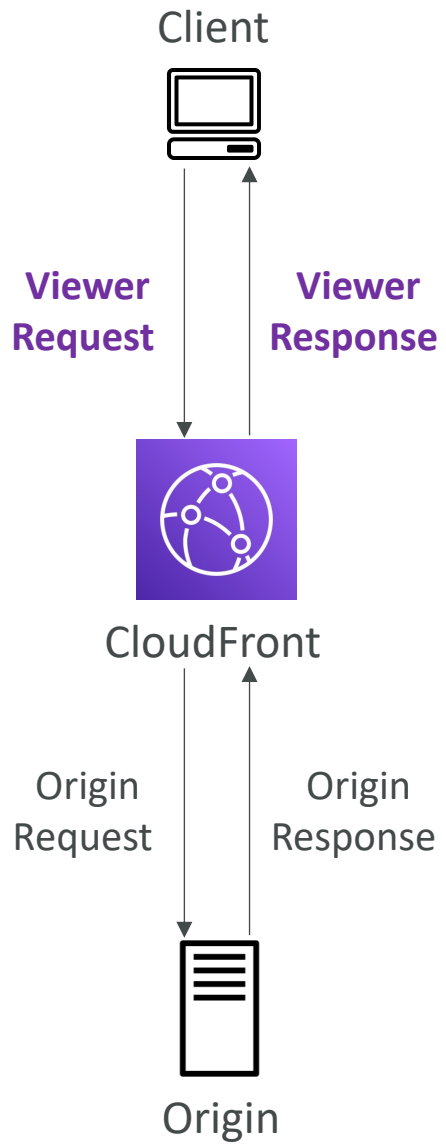
- Use case: customize the CDN content
- Pay only for what you use
- Fully serverless

CloudFront Functions & Lambda@Edge Use Cases

- Website Security and Privacy
- Dynamic Web Application at the Edge
- Search Engine Optimization (SEO)
- Intelligently Route Across Origins and Data Centers
- Bot Mitigation at the Edge
- Real-time Image Transformation
- A/B Testing
- User Authentication and Authorization
- User Prioritization
- User Tracking and Analytics

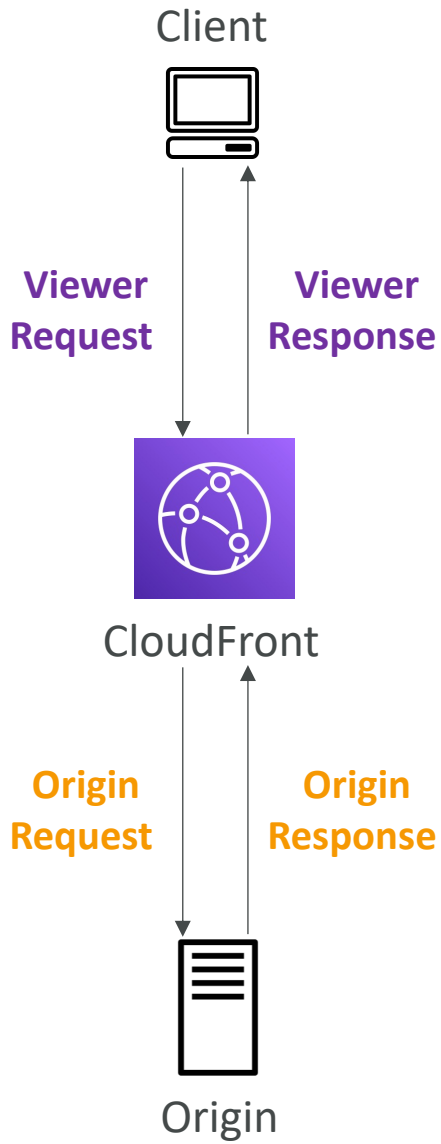
CloudFront Functions

- Lightweight functions written in JavaScript
- For high-scale, latency-sensitive CDN customizations
- Sub-ms startup times, **millions of requests/second**
- Used to change Viewer requests and responses:
  - **Viewer Request:** after CloudFront receives a request from a viewer
  - **Viewer Response:** before CloudFront forwards the response to the viewer
- Native feature of CloudFront (manage code entirely within CloudFront)



Lambda@Edge

- Lambda functions written in NodeJS or Python
- Scales to **1000s of requests/second**
- Used to change CloudFront requests and responses:
  - **Viewer Request** – after CloudFront receives a request from a viewer
  - **Origin Request** – before CloudFront forwards the request to the origin
  - **Origin Response** – after CloudFront receives the response from the origin
  - **Viewer Response** – before CloudFront forwards the response to the viewer
- Author your functions in one AWS Region (us-east-1), then CloudFront replicates to its locations



CloudFront Functions vs. Lambda@Edge

	CloudFront Functions	Lambda@Edge
Runtime Support	JavaScript	Node.js, Python
# of Requests	**Millions** of requests per second	**Thousands** of requests per second
CloudFront Triggers	- Viewer Request/Response	- Viewer Request/Response - Origin Request/Response
Max. Execution Time	< 1 ms	5 – 10 seconds
Max. Memory	2 MB	128 MB up to 10 GB
Total Package Size	10 KB	1 MB – 50 MB
Network Access, File System Access	No	Yes
Access to the Request Body	No	Yes
Pricing	Free tier available, 1/6th price of @Edge	No free tier, charged per request & duration

CloudFront Functions vs. Lambda@Edge - Use Cases

CloudFront Functions

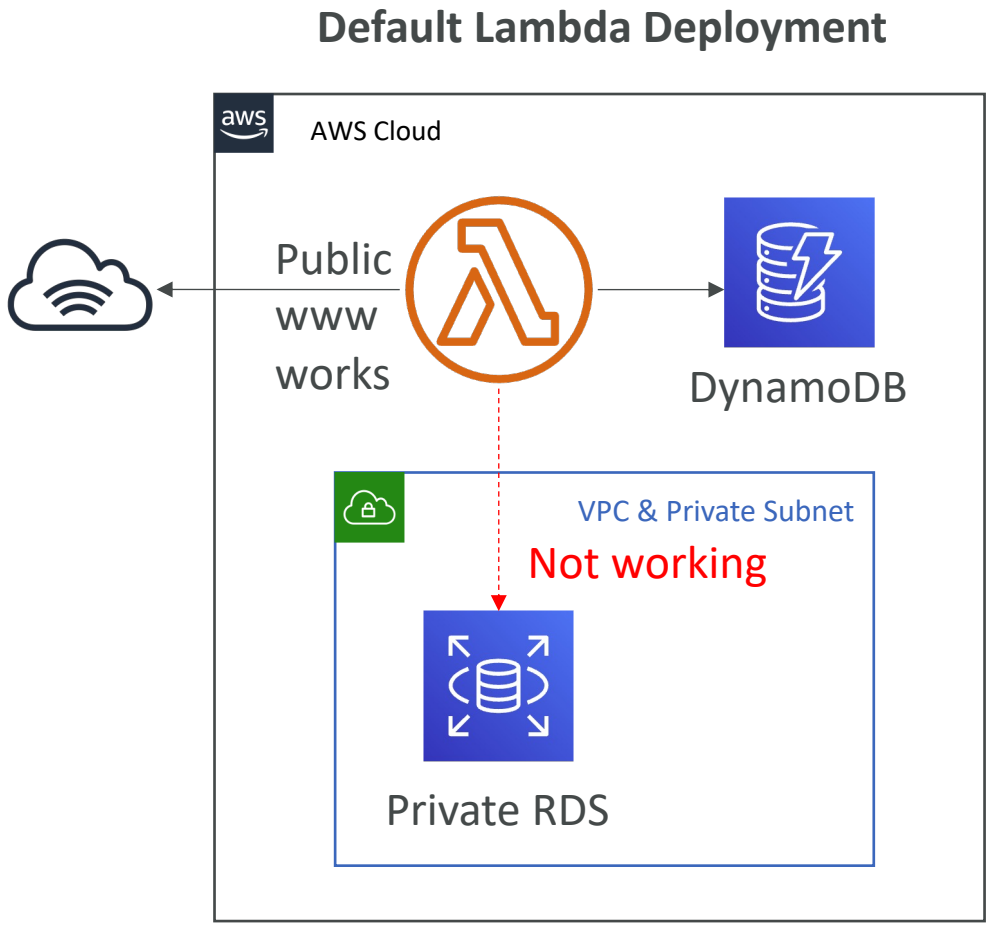
- Cache key normalization
  - Transform request attributes (headers, cookies, query strings, URL) to create an optimal Cache Key
- Header manipulation
  - Insert/modify/delete HTTP headers in the request or response
- URL rewrites or redirects
- Request authentication & authorization
  - Create and validate user-generated tokens (e.g., JWT) to allow/deny requests

Lambda@Edge

- Longer execution time (several ms)
- Adjustable CPU or memory
- Your code depends on a 3rd libraries (e.g., AWS SDK to access other AWS services)
- Network access to use external services for processing
- File system access or access to the body of HTTP requests

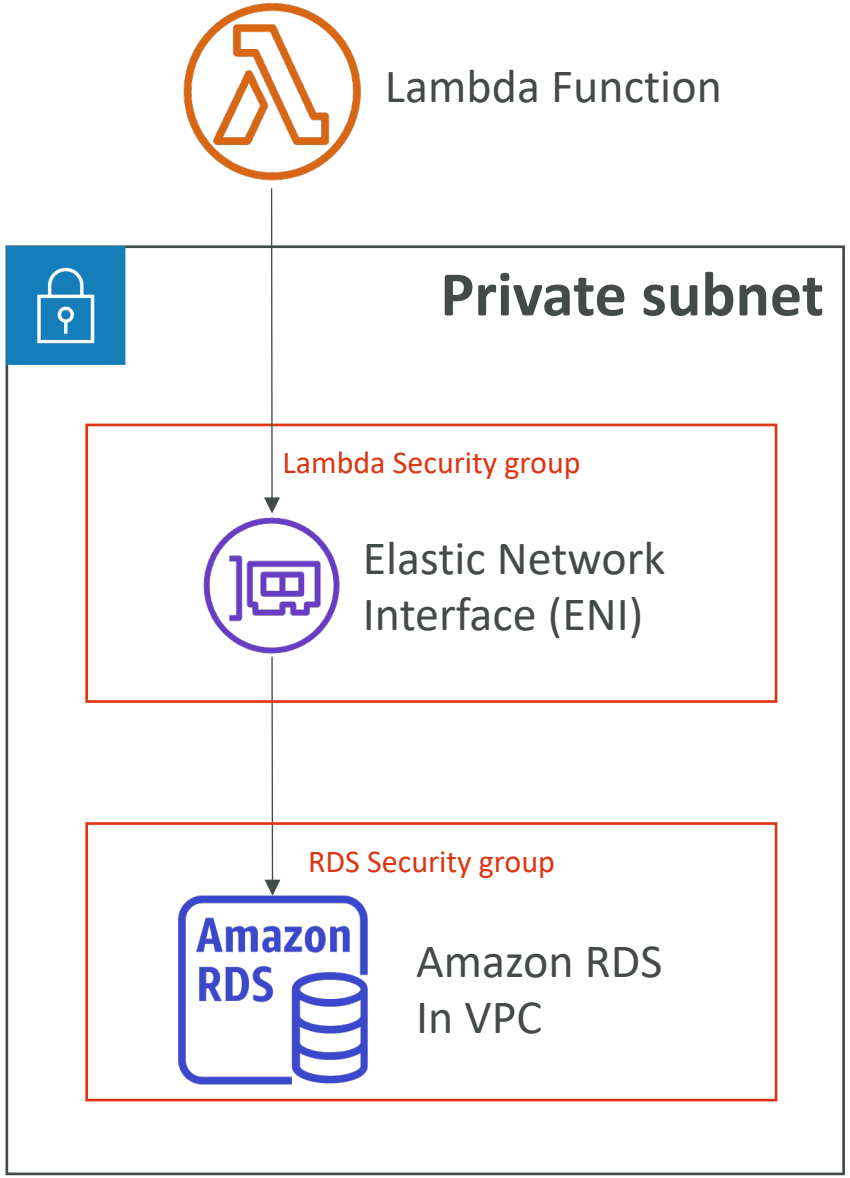
Lambda by default

- By default, your Lambda function is launched outside your own VPC (in an AWS-owned VPC)
- Therefore it cannot access resources in your VPC (RDS, ElastiCache, internal ELB…)



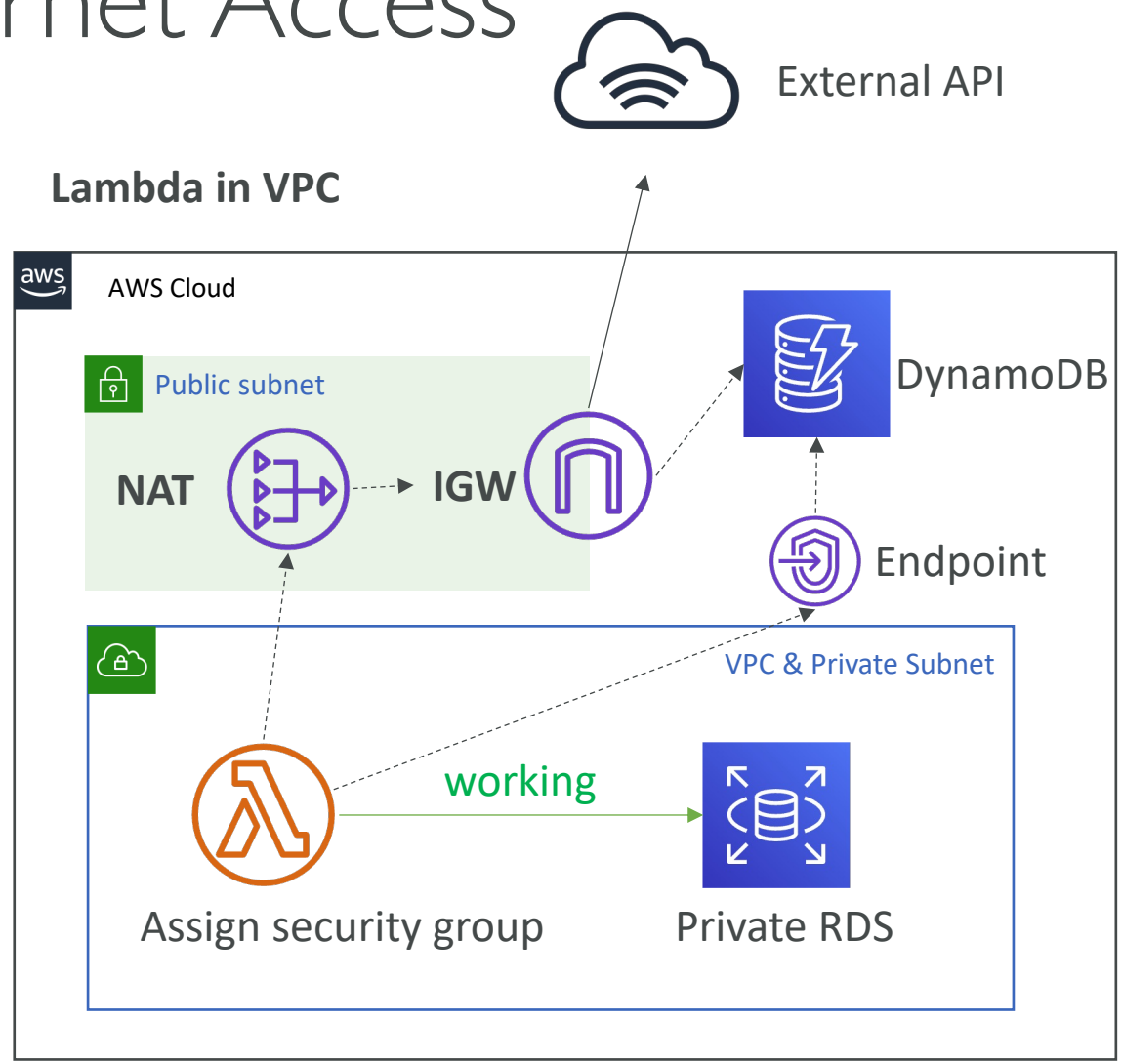
Lambda in VPC

- You must define the VPC ID, the Subnets and the Security Groups
- Lambda will create an ENI (Elastic Network Interface) in your subnets
- AWSLambdaVPCAccessExecutionRole



Lambda in VPC – Internet Access

- A Lambda function in your VPC does not have internet access
- Deploying a Lambda function in a public subnet does not give it internet access or a public IP
- Deploying a Lambda function in a private subnet gives it internet access if you have a NAT Gateway / Instance
- You can use VPC endpoints to privately access AWS services without a NAT



Note: Lambda - CloudWatch Logs works even without endpoint or NAT Gateway

Lambda Function Configuration

- RAM:
  - From 128MB to 10GB in 1MB increments
  - The more RAM you add, the more vCPU credits you get
  - At 1,792 MB, a function has the equivalent of one full vCPU
  - After 1,792 MB, you get more than one CPU, and need to use multi-threading in your code to benefit from it (up to 6 vCPU)
- If your application is CPU-bound (computation heavy), increase RAM

- Timeout: default 3 seconds, maximum is 900 seconds (15 minutes)

Lambda Execution Context

- The execution context is a temporary runtime environment that initializes any external dependencies of your lambda code
- Great for database connections, HTTP clients, SDK clients…
- The execution context is maintained for some time in anticipation of another Lambda function invocation
- The next function invocation can "re-use" the context to execution time and save time in initializing connections objects
- The execution context includes the */tmp* directory

Initialize outside the handler

BAD!

```
import os

def get_user_handler(event, context):

    DB_URL = os.getenv("DB_URL")
    db_client = db.connect(DB_URL)
    user = db_client.get(user_id = event["user_id"])

    return user
```

The DB connection is established
At every function invocation

GOOD!

```
import os

DB_URL = os.getenv("DB_URL")
db_client = db.connect(DB_URL)

def get_user_handler(event, context):

    user = db_client.get(user_id = event["user_id"])

    return user
```

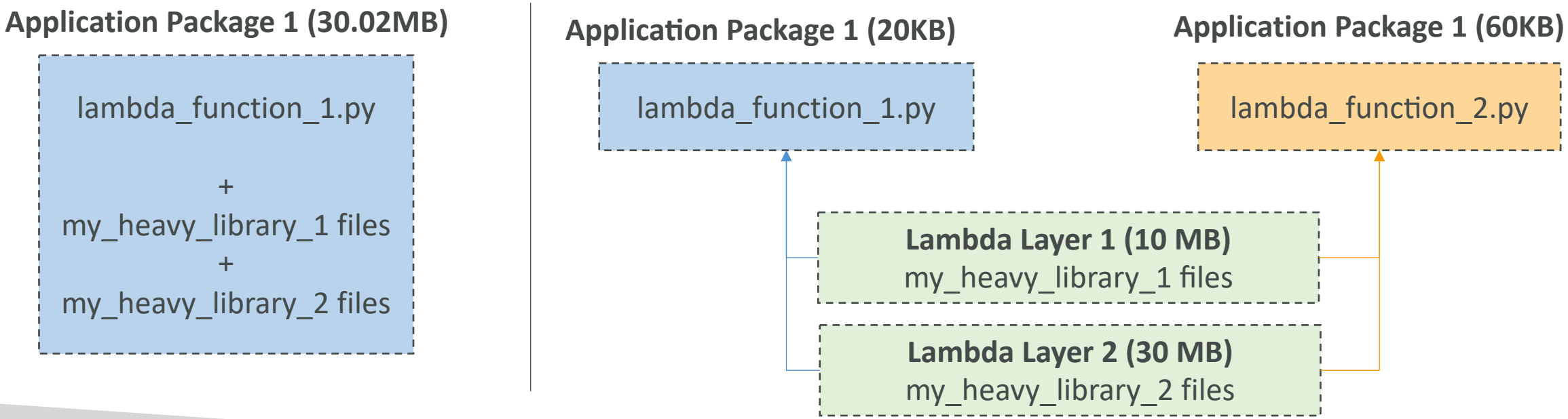
The DB connection is established once
And re-used across invocations

Lambda Functions /tmp space

- If your Lambda function needs to download a big file to work…
- If your Lambda function needs disk space to perform operations…
- You can use the /tmp directory
- Max size is 10GB
- The directory content remains when the execution context is frozen, providing transient cache that can be used for multiple invocations (helpful to checkpoint your work)
- For permanent persistence of object (non temporary), use S3
- To encrypt content on /tmp, you must generate KMS Data Keys

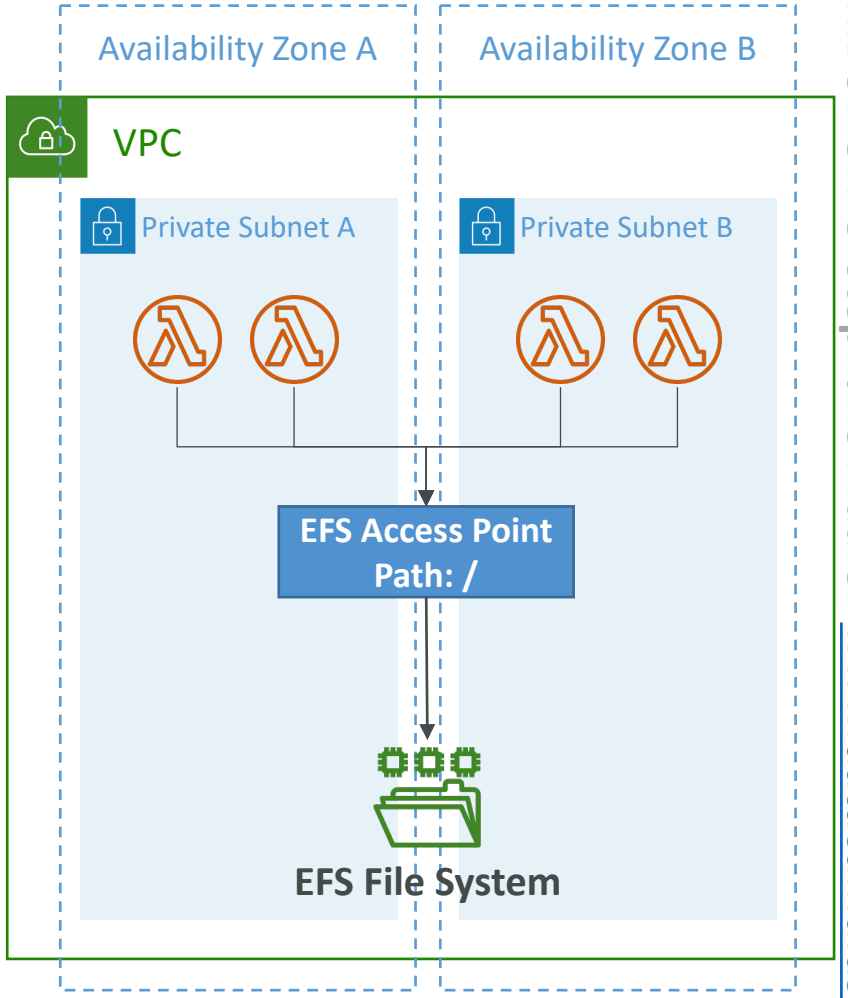
Lambda Layers

- Custom Runtimes
  - Ex: C++ https://github.com/awslabs/aws-lambda-cpp
  - Ex: Rust https://github.com/awslabs/aws-lambda-rust-runtime
- Externalize Dependencies to re-use them:



Lambda – File Systems Mounting

- Lambda functions can access EFS file systems if they are running in a VPC
- Configure Lambda to mount EFS file systems to local directory during initialization
- Must leverage EFS Access Points
- Limitations: watch out for the EFS connection limits (one function instance = one connection) and connection burst limits

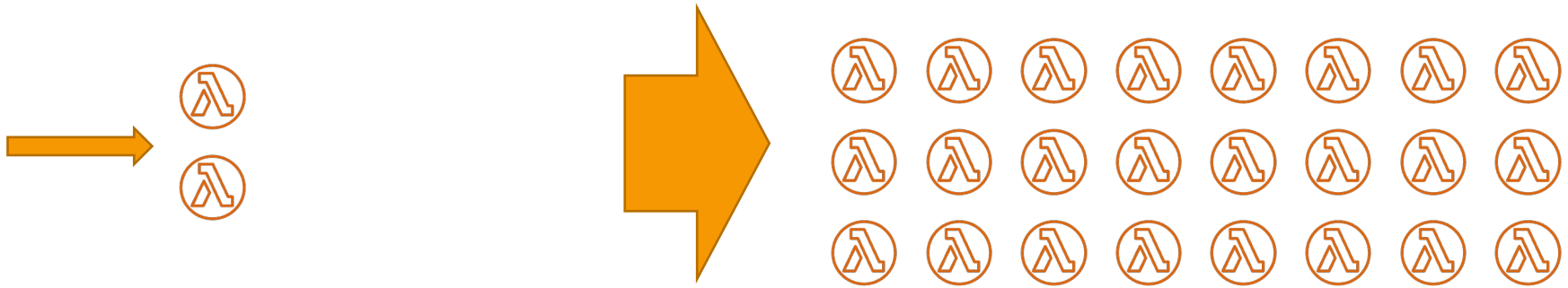


Lambda – Storage Options

	Ephemeral Storage /tmp	Lambda Layers	Amazon S3	Amazon EFS
Max. Size	10,240 MB	5 layers per function up to 250MB total	Elastic	Elastic
Persistence	Ephemeral	Durable	Durable	Durable
Content	Dynamic	Static	Dynamic	Dynamic
Storage Type	File System	Archive	Object	File System
Operations supported	any File System operation	Immutable	Atomic with Versioning	any File System operation
Pricing	Included in Lambda	Included in Lambda	Storage + Requests + Data Transfer	Storage + Data Transfer + Throughput
Sharing/Permissions	Function Only	IAM	IAM	IAM + NFS
Relative Data Access Speed from Lambda	Fastest	Fastest	Fast	Very Fast
Shared Across All Invocations	No	Yes	Yes	Yes

Lambda Concurrency and Throttling

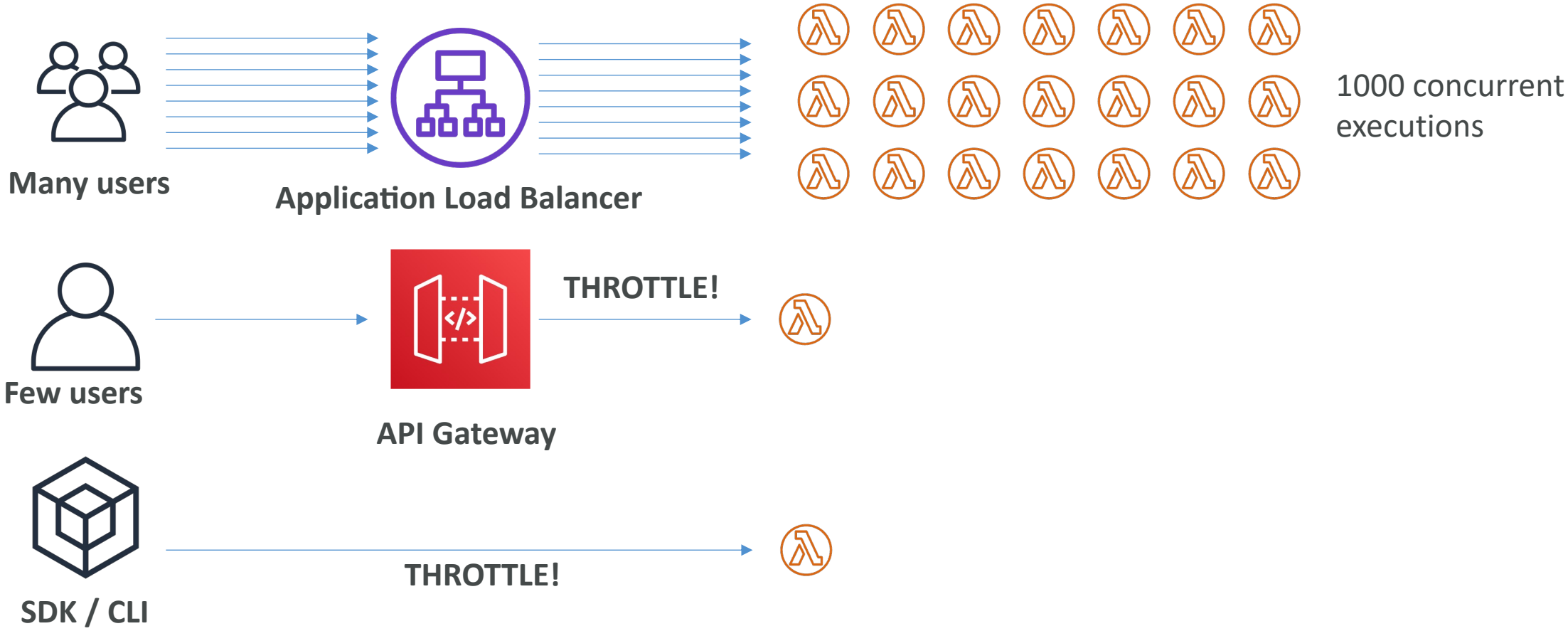
- Concurrency limit: up to 1000 concurrent executions



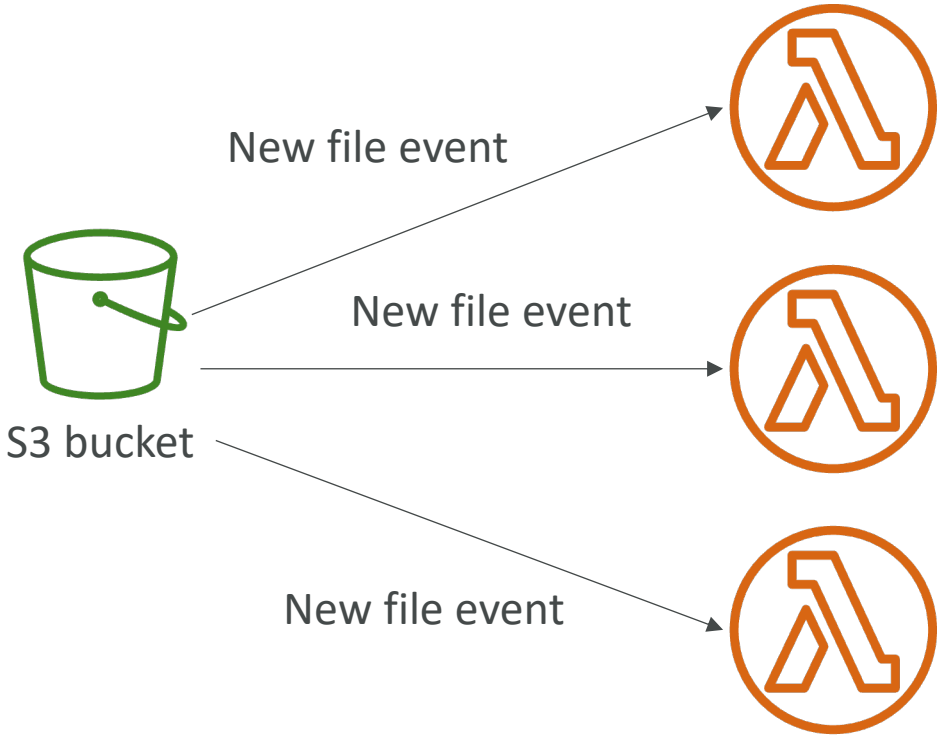
- Can set a "**reserved concurrency**" at the function level (=limit)
- Each invocation over the concurrency limit will trigger a "Throttle"
- Throttle behavior:
  - If synchronous invocation => return ThrottleError - 429
  - If asynchronous invocation => retry automatically and then go to DLQ
- If you need a higher limit, open a support ticket

Lambda Concurrency Issue

- If you don't reserve (=limit) concurrency, the following can happen:



Concurrency and Asynchronous Invocations

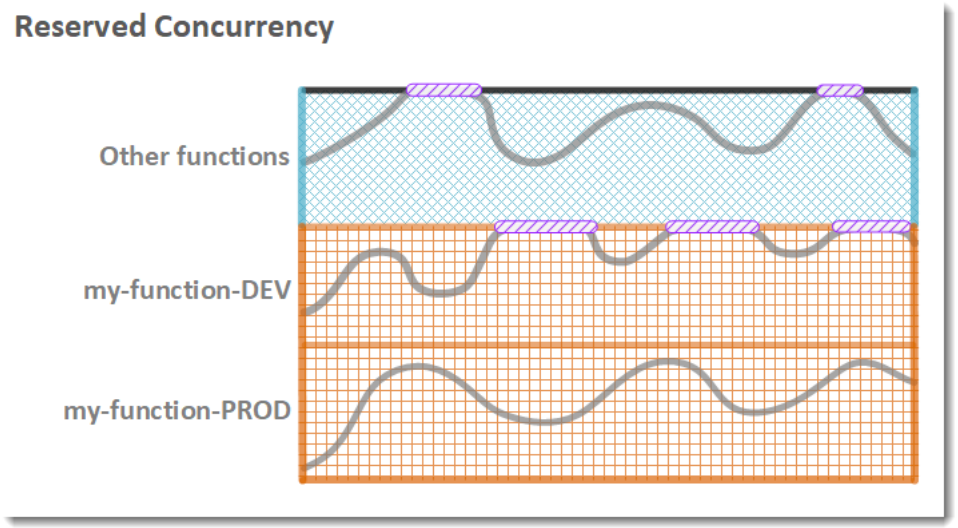


- If the function doesn't have enough concurrency available to process all events, additional requests are throttled.
- For throttling errors (429) and system errors (500-series), Lambda returns the event to the queue and attempts to run the function again for up to 6 hours.
- The retry interval increases exponentially from 1 second after the first attempt to a maximum of 5 minutes.

Cold Starts & Provisioned Concurrency

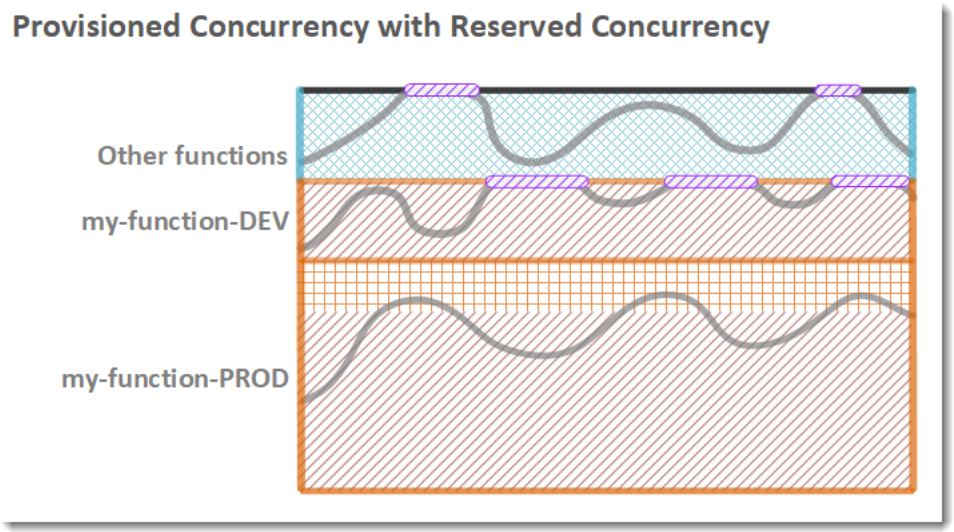
- Cold Start:
  - New instance => code is loaded and code outside the handler run (init)
  - If the init is large (code, dependencies, SDK…) this process can take some time.
  - First request served by new instances has higher latency than the rest
- Provisioned Concurrency:
  - Concurrency is allocated before the function is invoked (in advance)
  - So the cold start never happens and all invocations have low latency
  - Application Auto Scaling can manage concurrency (schedule or target utilization)
- Note:
  - Note: cold starts in VPC have been dramatically reduced in Oct & Nov 2019
  - https://aws.amazon.com/blogs/compute/announcing-improved-vpc-networking-for-aws-lambda-functions/

Reserved and Provisioned Concurrency



Legend

- Function concurrency
- Reserved concurrency
- Unreserved concurrency
- Throttling



Legend

- Function concurrency
- Reserved concurrency
- Provisioned concurrency
- Unreserved concurrency
- Throttling

https://docs.aws.amazon.com/lambda/latest/dg/configuration-concurrency.html

Lambda Function Dependencies

- If your Lambda function depends on external libraries: for example AWS X-Ray SDK, Database Clients, etc…
- You need to install the packages alongside your code and zip it together
  - For Node.js, use npm & "node_modules" directory
  - For Python, use pip --target options
  - For Java, include the relevant .jar files
- Upload the zip straight to Lambda if less than 50MB, else to S3 first
- Native libraries work: they need to be compiled on Amazon Linux
- AWS SDK comes by default with every Lambda function

Lambda and CloudFormation – inline

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Description: Lambda function inline
Resources:
  primer:
    Type: AWS::Lambda::Function
    Properties:
      Runtime: python3.x
      Role: arn:aws:iam::123456789012:role/lambda-role
      Handler: index.handler
      Code:
        ZipFile: |
          import os

          DB_URL = os.getenv("DB_URL")
          db_client = db.connect(DB_URL)
          def handler(event, context):
              user = db_client.get(user_id = event["user_id"])
              return user
```

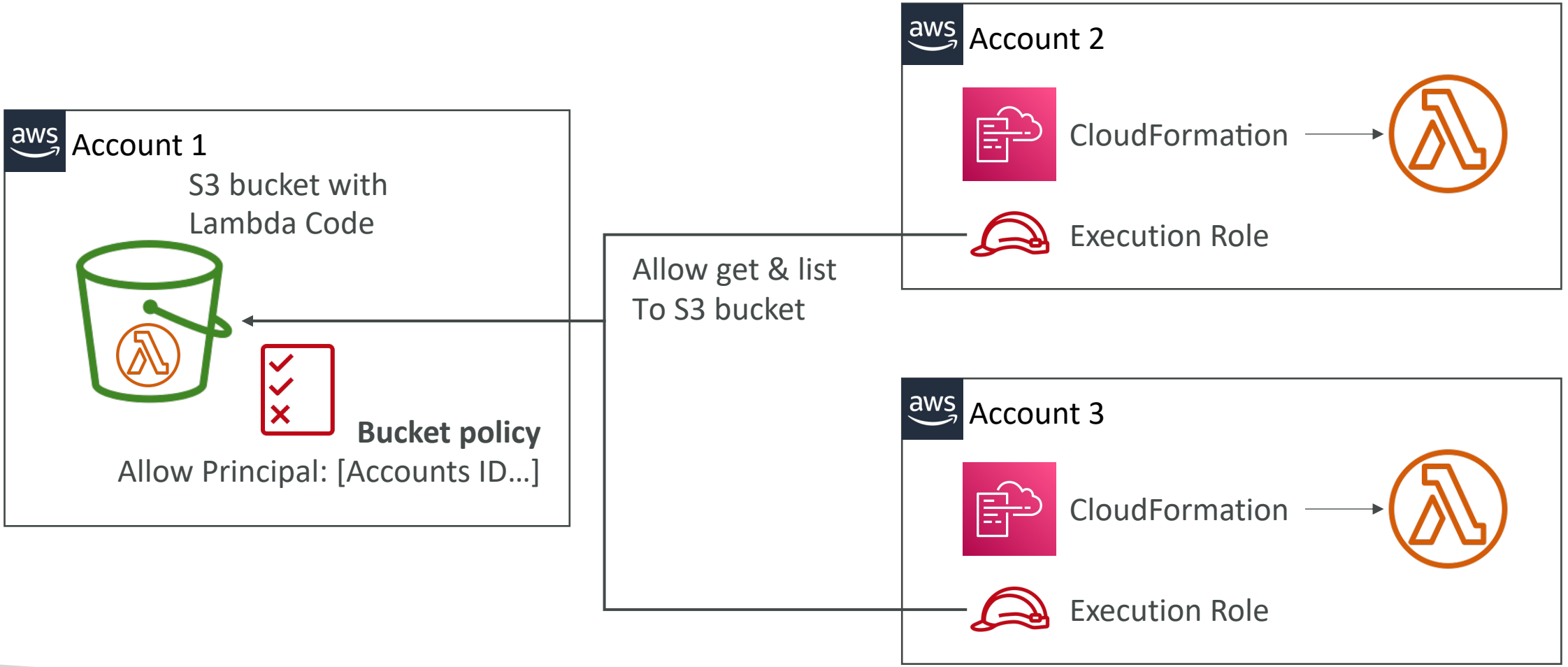
- Inline functions are very simple
- Use the **Code.ZipFile** property
- You cannot include function dependencies with inline functions

Lambda and CloudFormation – through S3

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Description: Lambda from S3
Resources:
  Function:
    Type: AWS::Lambda::Function
    Properties:
      Handler: index.handler
      Role: arn:aws:iam::123456789012:role/lambda-role
      Code:
        S3Bucket: my-bucket
        S3Key: function.zip
        S3ObjectVersion: String
      Runtime: nodejs12.x
```

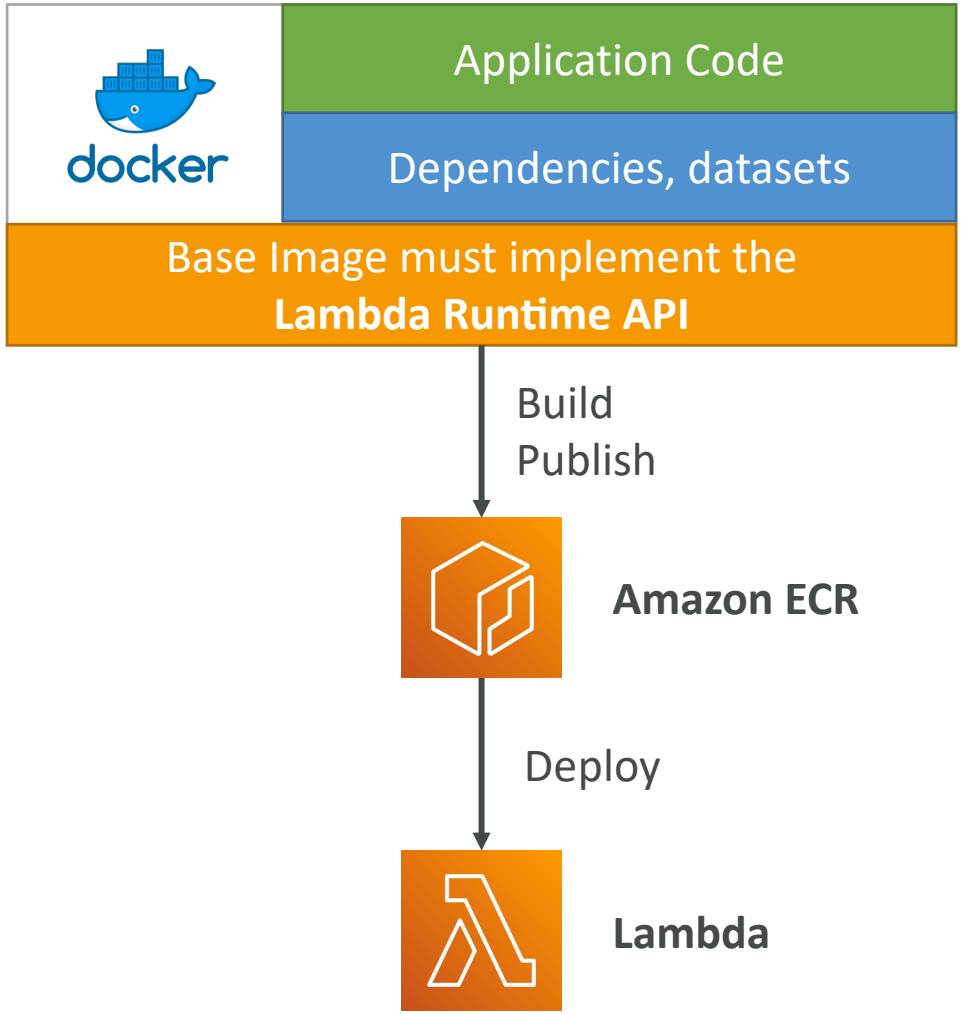
- You must store the Lambda zip in S3
- You must refer the S3 zip location in the CloudFormation code
  - S3Bucket
  - S3Key: full path to zip
  - S3ObjectVersion: if versioned bucket
- If you update the code in S3, but don't update S3Bucket, S3Key or S3ObjectVersion, CloudFormation won't update your function

Lambda and CloudFormation – through S3 Multiple accounts



Lambda Container Images

- Deploy Lambda function as container images of up to 10GB from ECR
- Pack complex dependencies, large dependencies in a container
- Base images are available for Python, Node.js, Java, .NET, Go, Ruby
- Can create your own image as long as it implements **the Lambda Runtime API**
- Test the containers locally using the Lambda Runtime Interface Emulator
- Unified workflow to build apps



Lambda Container Images

- Example: build from the base images provided by AWS

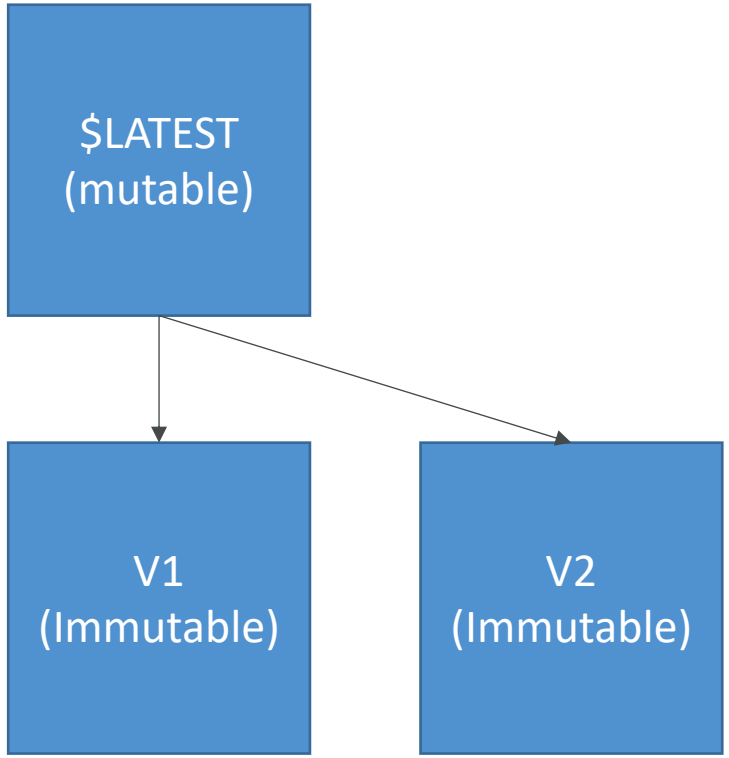
```
# Use an image that implements the Lambda Runtime API
FROM amazon/aws-lambda-nodejs:12

# Copy your application code and files
COPY app.js package*.json ./

# Install the dependencies in the container
RUN npm install

# Function to run when the Lambda function is invoked
CMD [ "app.lambdaHandler" ]
```

Lambda Container Images – Best Practices

- Strategies for optimizing container images:
  - Use AWS-provided Base Images
    - Stable, Built on Amazon Linux 2, cached by Lambda service
  - Use Multi-Stage Builds
    - Build your code in larger preliminary images, copy only the artifacts you need in your final container image, discard the preliminary steps
  - Build from Stable to Frequently Changing
    - Make your most frequently occurring changes as late in your *Dockerfile* as possible
  - Use a Single Repository for Functions with Large Layers
    - ECR compares each layer of a container image when it is pushed to avoid uploading and storing duplicates
- Use them to upload large Lambda Functions (up to 10 GB)

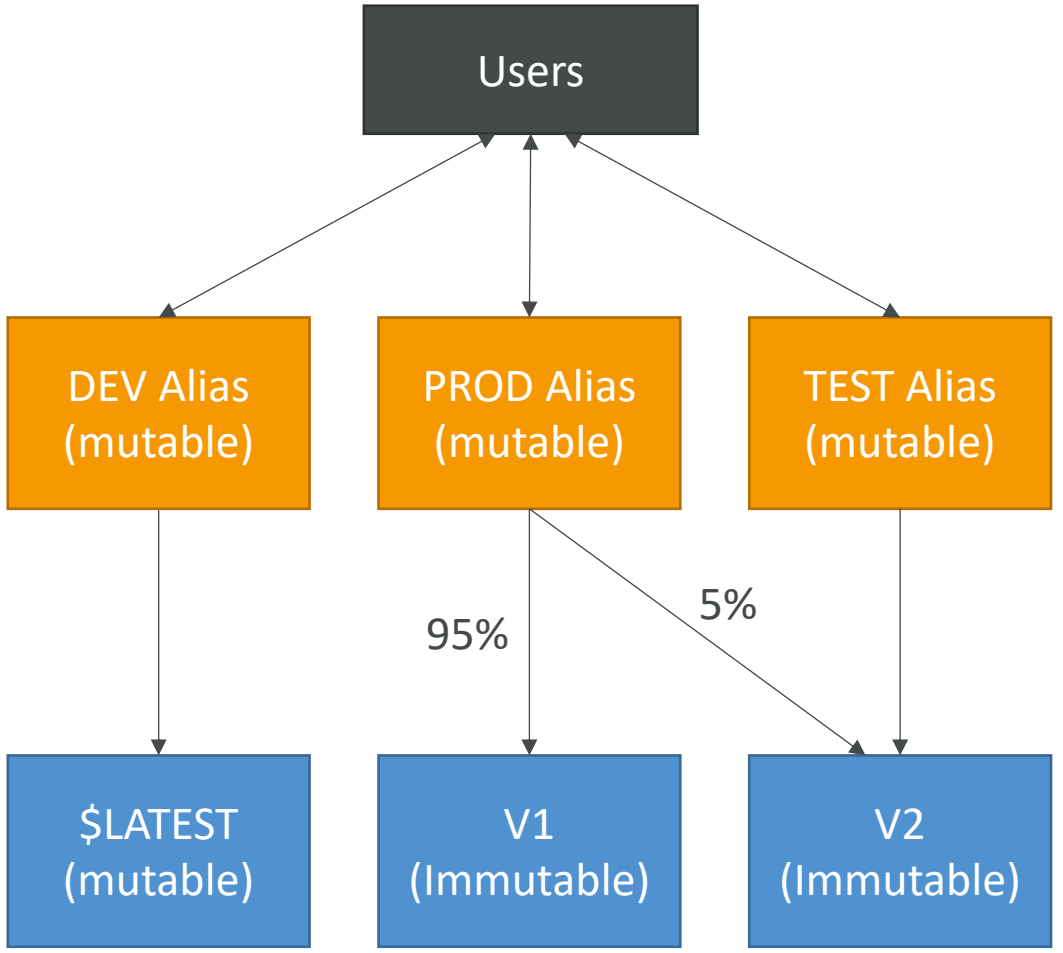
AWS Lambda Versions

- When you work on a Lambda function, we work on **$LATEST**
- When we're ready to publish a Lambda function, we create a version
- Versions are immutable
- Versions have increasing version numbers
- Versions get their own ARN (Amazon Resource Name)
- Version = code + configuration (nothing can be changed - immutable)
- Each version of the lambda function can be accessed



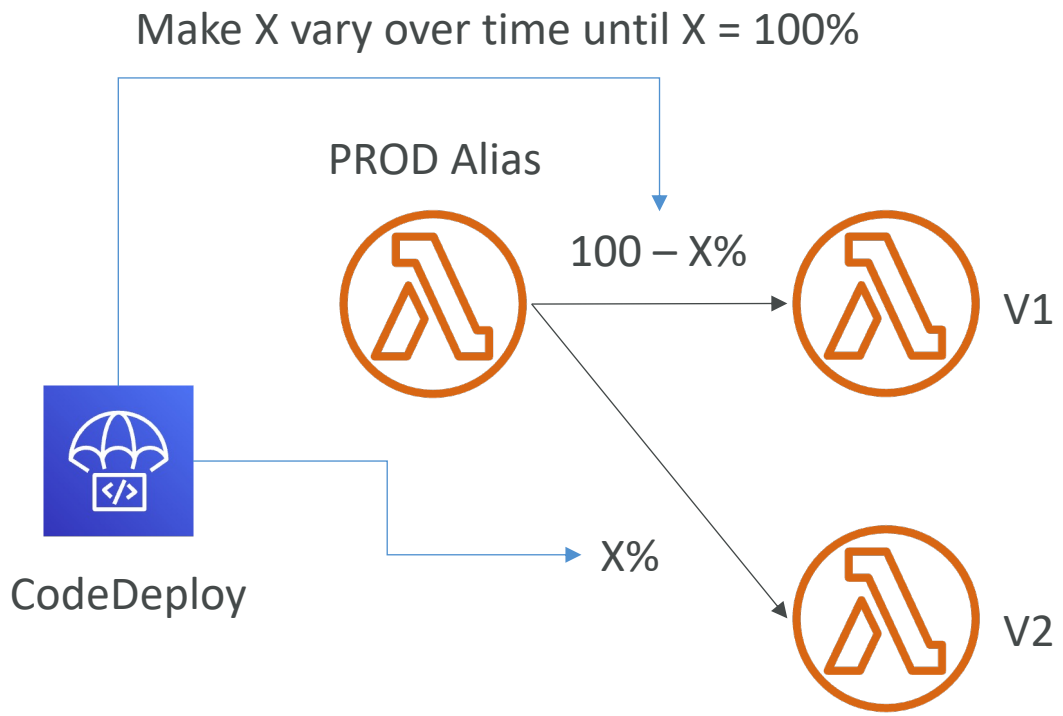
AWS Lambda Aliases

- Aliases are "pointers" to Lambda function versions
- We can define a "dev", "test", "prod" aliases and have them point at different lambda versions
- Aliases are mutable
- Aliases enable Canary deployment by assigning weights to lambda functions
- Aliases enable stable configuration of our event triggers / destinations
- Aliases have their own ARNs
- Aliases cannot reference aliases



Lambda & CodeDeploy

- **CodeDeploy** can help you automate traffic shift for Lambda aliases
- Feature is integrated within the SAM framework
- **Linear:** grow traffic every N minutes until 100%
  - Linear10PercentEvery3Minutes
  - Linear10PercentEvery10Minutes
- **Canary:** try X percent then 100%
  - Canary10Percent5Minutes
  - Canary10Percent30Minutes
- **AllAtOnce:** immediate
- Can create Pre & Post Traffic hooks to check the health of the Lambda function



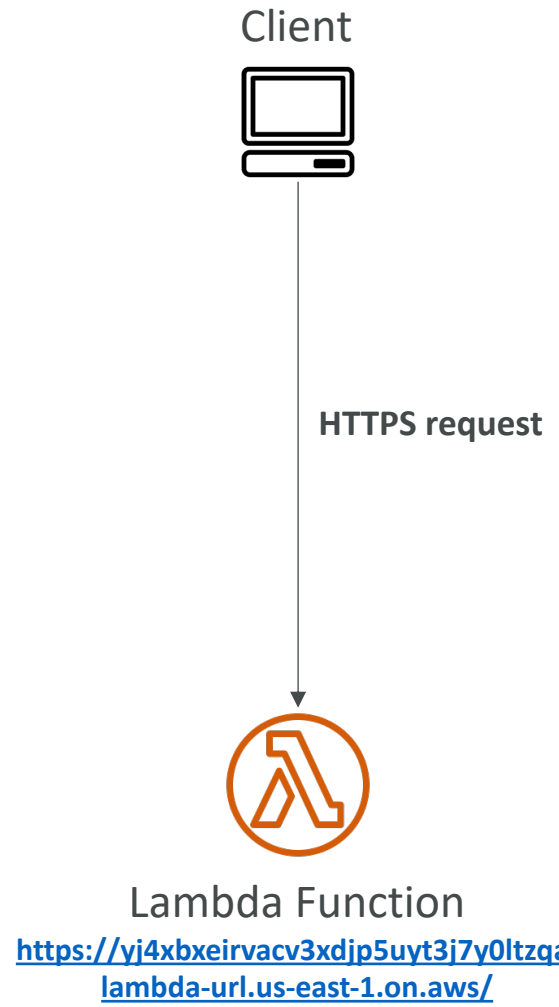
Lambda & CodeDeploy – AppSpec.yml

```yaml
version: 0.0

Resources:
  - myLambdaFunction:
      Type: AWS::Lambda::Function
      Properties:
        Name: myLambdaFunction
        Alias: myLambdaFunctionAlias
        CurrentVersion: 1
        TargetVersion: 2
```

- **Name (required)** – the name of the Lambda function to deploy
- **Alias (required)** – the name of the alias to the Lambda function
- **CurrentVersion (required)** – the version of the Lambda function traffic currently points to
- **TargetVersion (required)** – the version of the Lambda function traffic is shifted to

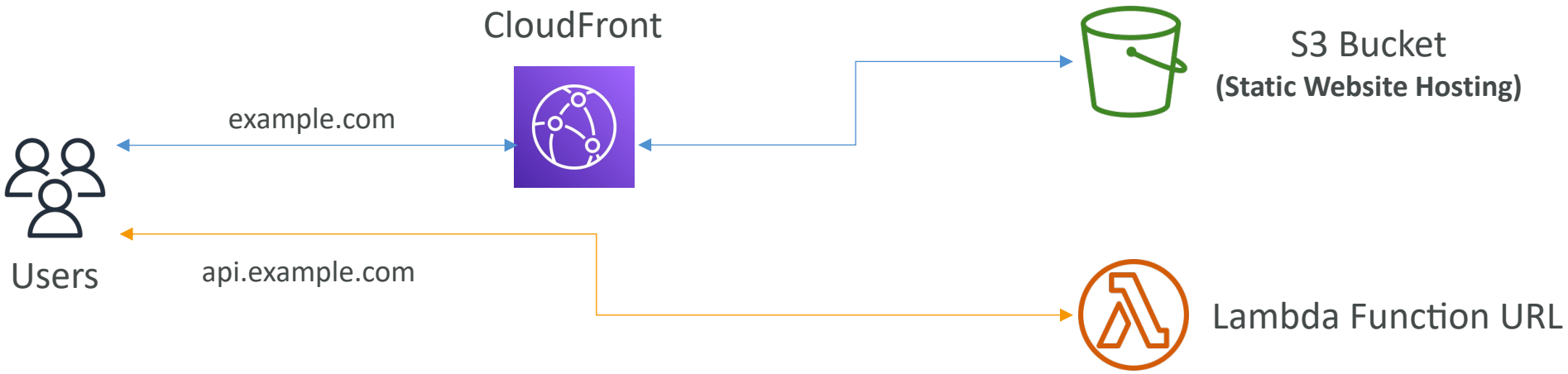
Lambda – Function URL

- Dedicated HTTP(S) endpoint for your Lambda function
- A unique URL endpoint is generated for you (never changes)
  - https://<*url-id*>.lambda-url.<*region*>.on.aws (dual-stack IPv4 & IPv6)
- Invoke via a web browser, curl, Postman, or any HTTP client
- Access your function URL through the public Internet only
  - Doesn't support PrivateLink (Lambda functions do support)
- Supports Resource-based Policies & CORS configurations
- Can be applied to any function alias or to $LATEST (can't be applied to other function versions)
- Create and configure using AWS Console or AWS API
- Throttle your function by using Reserved Concurrency



Lambda – Function URL Security

- Resource-based Policy
  - Authorize other accounts / specific CIDR / IAM principals
- Cross-Origin Resource Sharing (CORS)
  - If you call your Lambda function URL from a different domain

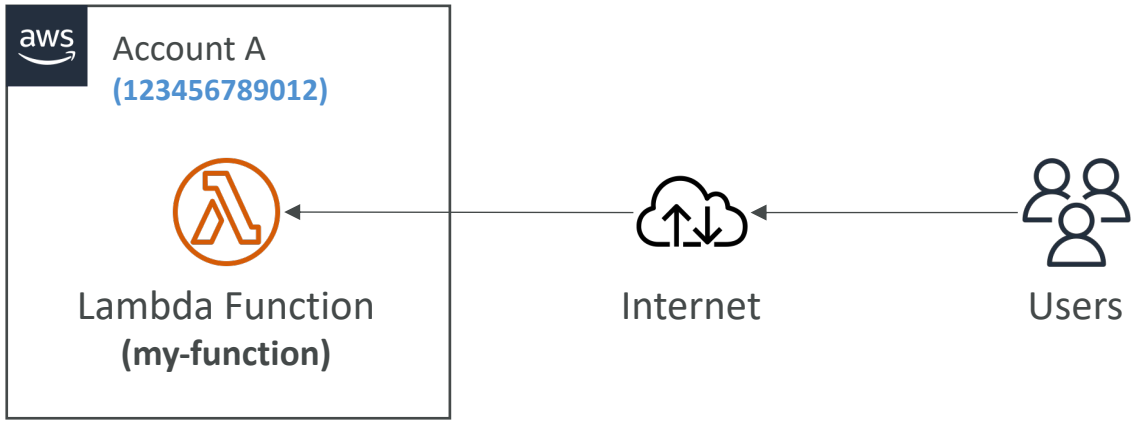


Lambda – Function URL Security

- AuthType NONE – allow public and unauthenticated access
  - Resource-based Policy is always in effect (must grant public access)

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "lambda:InvokeFunctionUrl",
            "Resource": "arn:aws:lambda:us-east-1:123456789012:
function:my-function",
            "Condition": {
                "StringEquals": {
                    "lambda:FunctionUrlAuthType": "NONE"
                }
            }
        }
    ]
}
```

Resource-based Policy

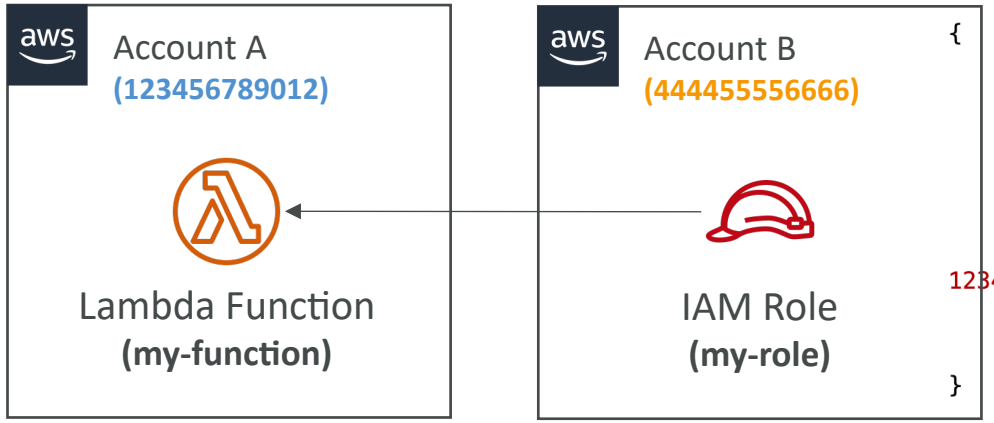


Lambda – Function URL Security

- AuthType AWS_IAM – IAM is used to authenticate and authorize requests
  - Both Principal's Identity-based Policy & Resource-based Policy are evaluated
  - Principal must have **lambda:InvokeFunctionUrl** permissions
  - **Same account** – Identity-based Policy **OR** Resource-based Policy as ALLOW
  - **Cross account** – Identity-based Policy **AND** Resource Based Policy as ALLOW

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::444455556666:role/my-role"
            },
            "Action": "lambda:InvokeFunctionUrl",
            "Resource": "arn:aws:lambda:us-east-1:123456789012:
function:my-function",
            "Condition": {
                "StringEquals": {
                    "lambda:FunctionUrlAuthType": "AWS_IAM"
                }
            }
        }
    ]
}
```

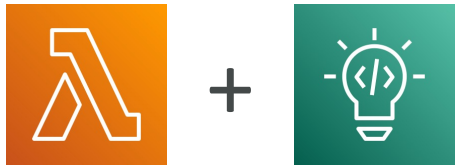
Resource-based Policy



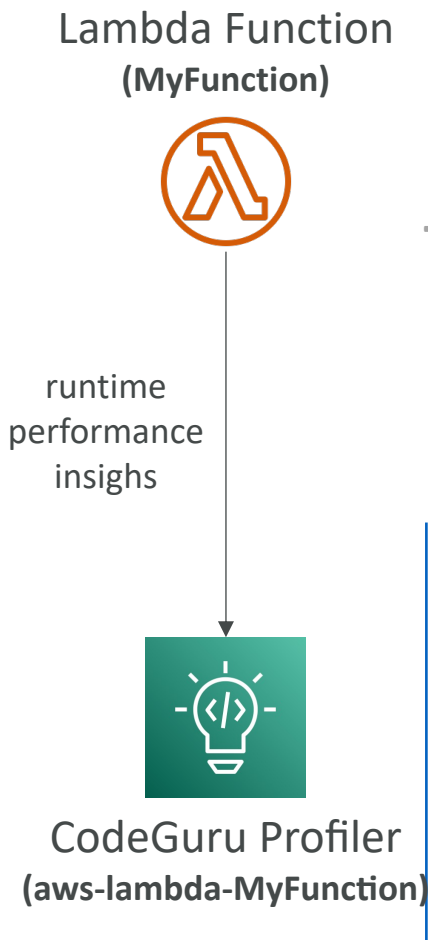

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "lambda:InvokeFunctionUrl",
            "Resource": "arn:aws:lambda:us-east-1:
123456789012:function:my-function"
        }
    ]
}
```

Identity-based Policy

Lambda and CodeGuru Profiling



- Gain insights into runtime performance of your Lambda functions using CodeGuru Profiler
- CodeGuru creates a Profiler Group for your Lambda function
- Supported for Java and Python runtimes
- Activate from AWS Lambda Console
- When activated, Lambda adds:
  - CodeGuru Profiler layer to your function
  - Environment variables to your function
  - **AmazonCodeGuruProfilerAgentAccess** policy to your function



AWS Lambda Limits to Know - per region

- Execution:
  - Memory allocation: 128 MB – 10GB (1 MB increments)
  - Maximum execution time: 900 seconds (15 minutes)
  - Environment variables (4 KB)
  - Disk capacity in the "function container" (in /tmp): 512 MB to 10GB
  - Concurrency executions: 1000 (can be increased)
- Deployment:
  - Lambda function deployment size (compressed .zip): 50 MB
  - Size of uncompressed deployment (code + dependencies): 250 MB
  - Can use the /tmp directory to load other files at startup
  - Size of environment variables: 4 KB

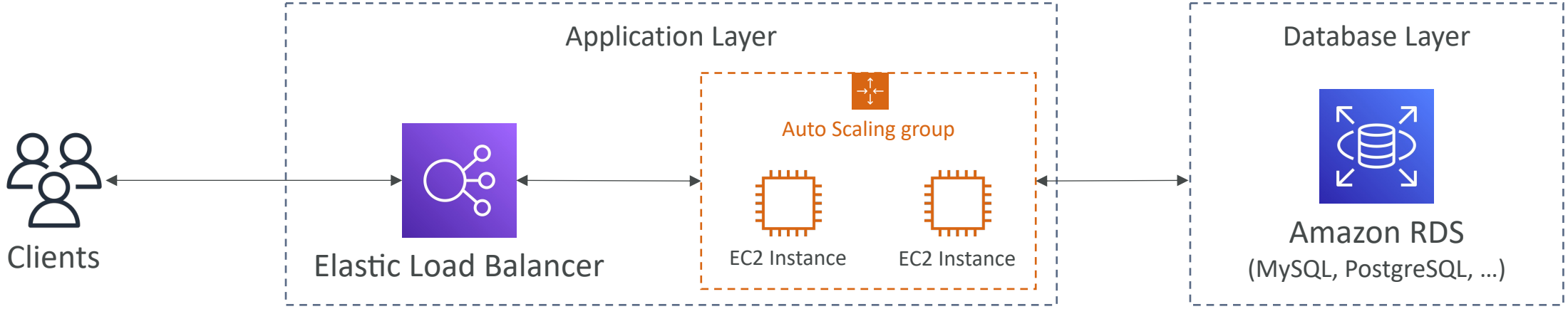
AWS Lambda Best Practices

- Perform heavy-duty work outside of your function handler
  - Connect to databases outside of your function handler
  - Initialize the AWS SDK outside of your function handler
  - Pull in dependencies or datasets outside of your function handler
- Use environment variables for:
  - Database Connection Strings, S3 bucket, etc… don't put these values in your code
  - Passwords, sensitive values… they can be encrypted using KMS
- Minimize your deployment package size to its runtime necessities.
  - Break down the function if need be
  - Remember the AWS Lambda limits
  - Use Layers where necessary
- Avoid using recursive code, never have a Lambda function call itself

Amazon DynamoDB

NoSQL Serverless Database

Traditional Architecture



- Traditional applications leverage RDBMS databases
- These databases have the SQL query language
- Strong requirements about how the data should be modeled
- Ability to do query joins, aggregations, complex computations
- Vertical scaling (getting a more powerful CPU / RAM / IO)
- Horizontal scaling (increasing reading capability by adding EC2 / RDS Read Replicas)

NoSQL databases

- NoSQL databases are non-relational databases and are **distributed**
- NoSQL databases include MongoDB, DynamoDB, …
- NoSQL databases do not support query joins (or just limited support)
- All the data that is needed for a query is present in one row
- NoSQL databases don't perform aggregations such as "SUM", "AVG", …
- **NoSQL databases scale horizontally**

- There's no "right or wrong" for NoSQL vs SQL, they just require to model the data differently and think about user queries differently

Amazon DynamoDB

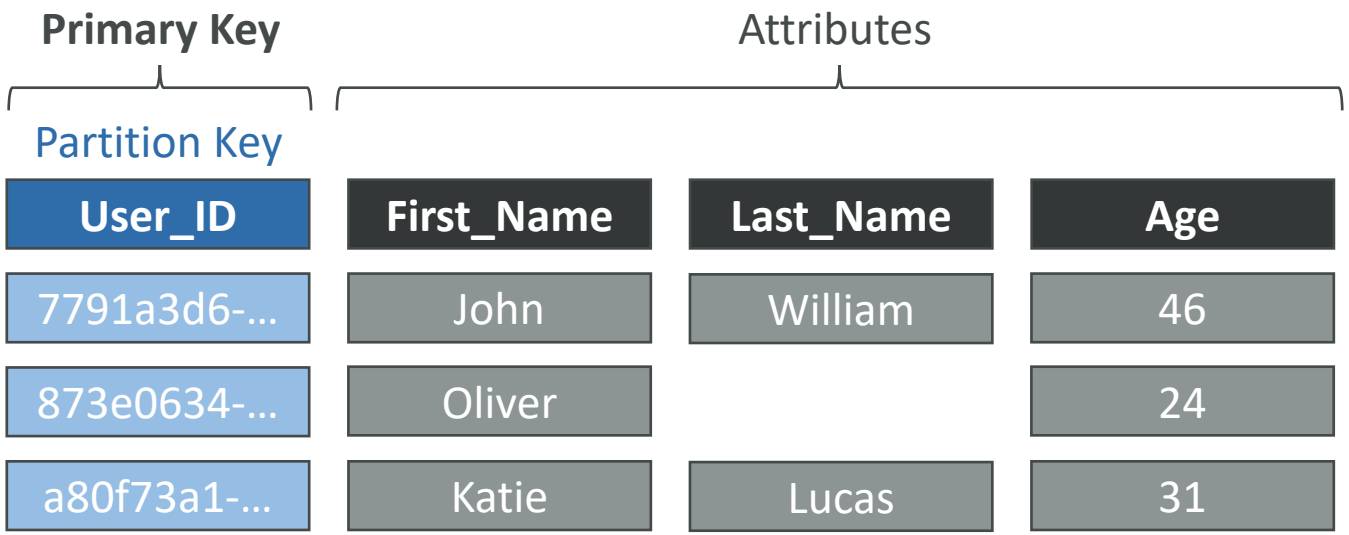
- Fully managed, highly available with replication across multiple AZs
- NoSQL database - not a relational database
- Scales to massive workloads, distributed database
- Millions of requests per seconds, trillions of row, 100s of TB of storage
- Fast and consistent in performance (low latency on retrieval)
- Integrated with IAM for security, authorization and administration
- Enables event driven programming with DynamoDB Streams
- Low cost and auto-scaling capabilities
- Standard & Infrequent Access (IA) Table Class

DynamoDB - Basics

- DynamoDB is made of **Tables**
- Each table has a **Primary Key** (must be decided at creation time)
- Each table can have an infinite number of items (= rows)
- Each item has **attributes** (can be added over time – can be null)
- Maximum size of an item is **400KB**
- Data types supported are:
  - **Scalar Types** – String, Number, Binary, Boolean, Null
  - **Document Types** – List, Map
  - **Set Types** – String Set, Number Set, Binary Set

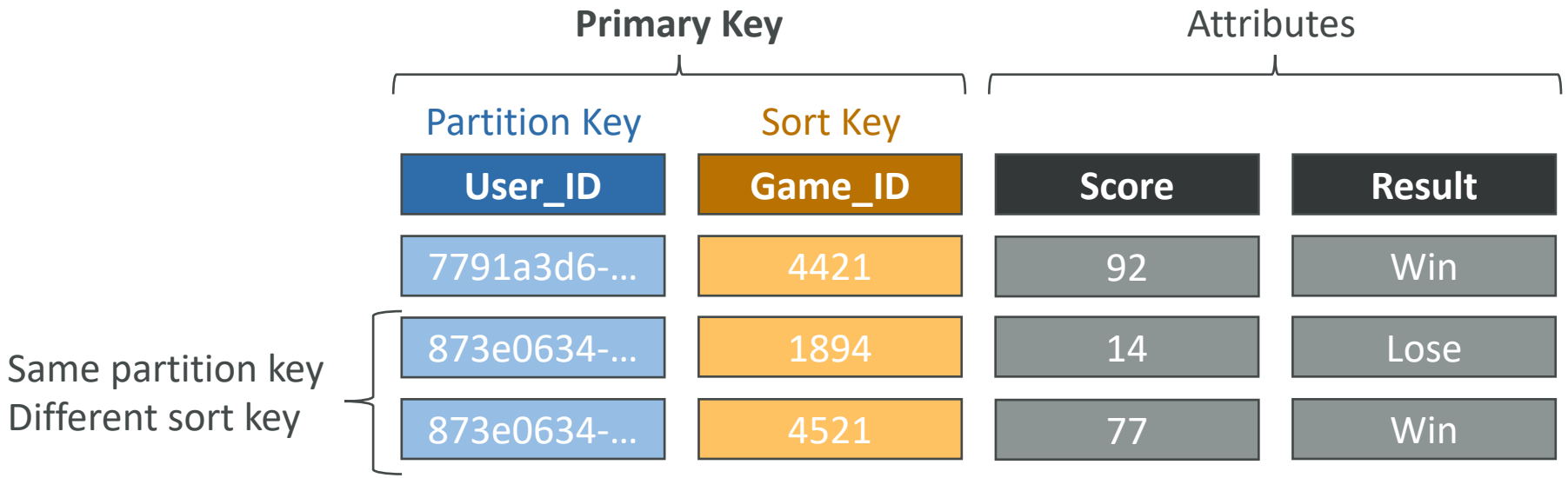
DynamoDB – Primary Keys

- Option 1: Partition Key (HASH)
  - Partition key must be unique for each item
  - Partition key must be "diverse" so that the data is distributed
  - Example: "User_ID" for a users table



DynamoDB – Primary Keys

- Option 2: Partition Key + Sort Key (HASH + RANGE)
  - The combination must be unique for each item
  - Data is grouped by partition key
  - Example: users-games table, "User_ID" for Partition Key and "Game_ID" for Sort Key



DynamoDB – Partition Keys (Exercise)

- We're building a movie database
- What is the best Partition Key to maximize data distribution?
  - movie_id
  - producer_name
  - leader_actor_name
  - movie_language

- "movie_id" has the highest cardinality so it's a good candidate
- "movie_language" doesn't take many values and may be skewed towards English so it's not a great choice for the Partition Key

DynamoDB – Read/Write Capacity Modes

- Control how you manage your table's capacity (read/write throughput)

- Provisioned Mode (default)
  - You specify the number of reads/writes per second
  - You need to plan capacity beforehand
  - Pay for provisioned read & write capacity units

- On-Demand Mode
  - Read/writes automatically scale up/down with your workloads
  - No capacity planning needed
  - Pay for what you use, more expensive ($$$)

- You can switch between different modes once every 24 hours

R/W Capacity Modes – Provisioned

- Table must have provisioned read and write capacity units
- **Read Capacity Units (RCU)** – throughput for reads
- **Write Capacity Units (WCU)** – throughput for writes
- Option to setup auto-scaling of throughput to meet demand
- Throughput can be exceeded temporarily using **"Burst Capacity"**
- If Burst Capacity has been consumed, you'll get a **"ProvisionedThroughputExceededException"**
- It's then advised to do an **exponential backoff** retry

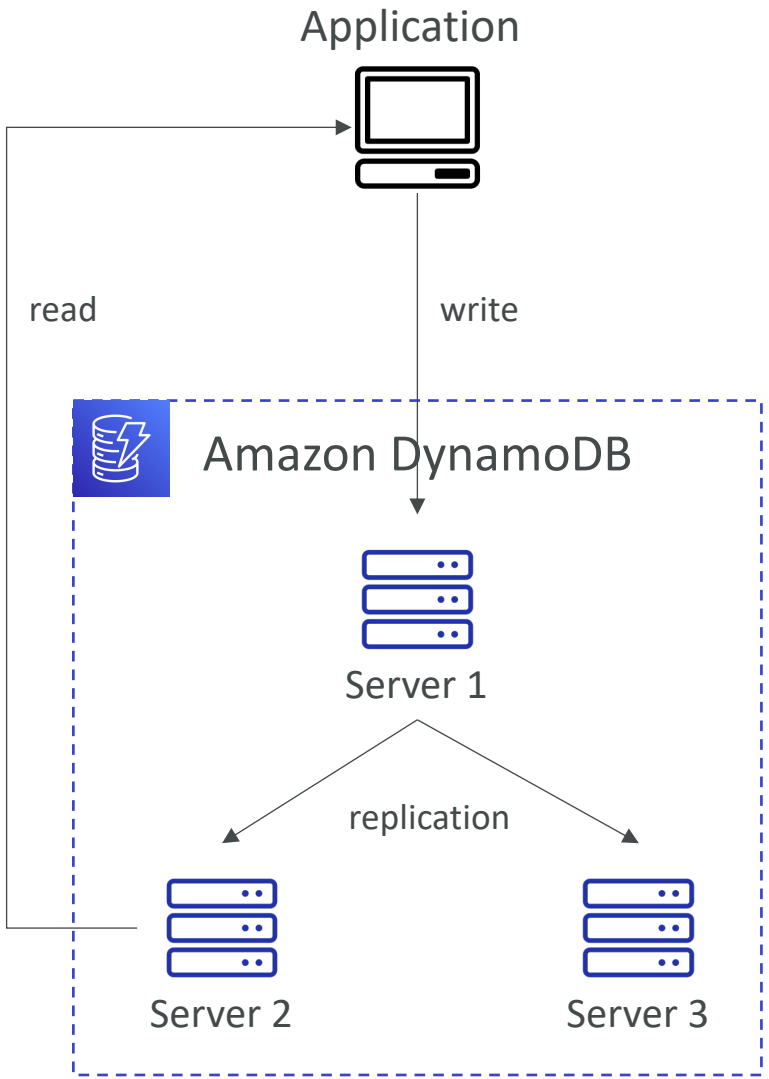
DynamoDB – Write Capacity Units (WCU)

- One *Write Capacity Unit (WCU)* represents one write per second for an item up to **1 KB** in size
- If the items are larger than 1 KB, more WCUs are consumed

- **Example 1:** we write 10 items per second, with item size 2 KB
  - We need $10 * (\frac{2\ KB}{1\ KB}) = 20\ WCUs$
- **Example 2:** we write 6 items per second, with item size 4.5 KB
  - We need $6 * (\frac{5\ KB}{1\ KB}) = 30\ WCUs$ (4.5 gets rounded to the upper KB)
- **Example 3:** we write 120 items per minute, with item size 2 KB
  - We need $\left(\frac{120}{60}\right) * (\frac{2\ KB}{1\ KB}) = 4\ WCUs$

Strongly Consistent Read vs. Eventually Consistent Read

- Eventually Consistent Read (default)
  - If we read just after a write, it's possible we'll get some stale data because of replication
- Strongly Consistent Read
  - If we read just after a write, we will get the correct data
  - Set "**ConsistentRead**" parameter to **True** in API calls (GetItem, BatchGetItem, Query, Scan)
  - Consumes twice the RCU



DynamoDB – Read Capacity Units (RCU)

- One *Read Capacity Unit (RCU)* represents **one Strongly Consistent Read** per second, or **two Eventually Consistent Reads** per second, for an item up to **4 KB** in size
- If the items are larger than 4 KB, more RCUs are consumed

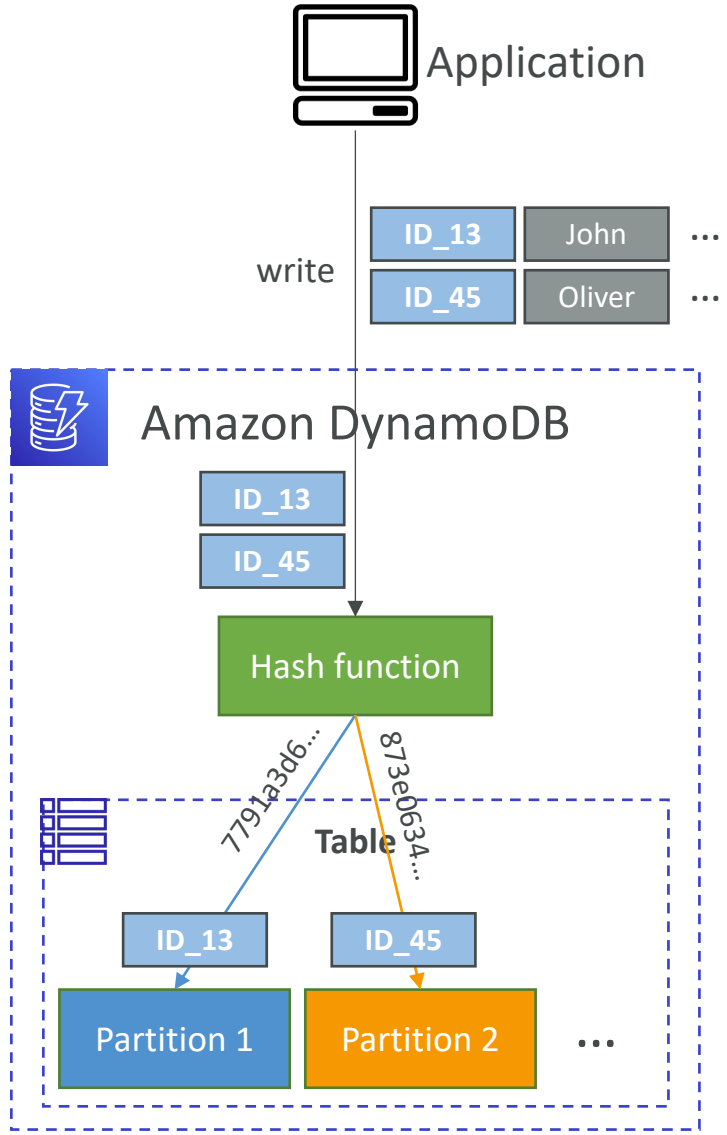
- **Example 1:** 10 Strongly Consistent Reads per second, with item size 4 KB
  - We need $10 * \left(\frac{4\ KB}{4\ KB}\right) = 10\ RCUs$
- **Example 2:** 16 Eventually Consistent Reads per second, with item size 12 KB
  - We need $\left(\frac{16}{2}\right) * \left(\frac{12\ KB}{4\ KB}\right) = 24\ RCUs$
- **Example 3:** 10 Strongly Consistent Reads per second, with item size 6 KB
  - We need $10 * \left(\frac{8\ KB}{4\ KB}\right) = 20\ RCUs$ (we must round up 6 KB to 8 KB)

DynamoDB – Partitions Internal

- Data is stored in partitions
- Partition Keys go through a hashing algorithm to know to which partition they go to

- To compute the number of partitions:
  - $\#\ of\ partitions_{by\ capacity} = \left(\frac{RCUs_{Total}}{3000}\right) + \left(\frac{WCUs_{Total}}{1000}\right)$
  - $\#\ of\ partitions_{by\ size} = \frac{Total\ Size}{10\ GB}$
  - $\#\ of\ partitions = \text{ceil}(\max(\#\ of\ partitions_{by\ capacity}, \#\ of\ partitions_{by\ size}))$

- WCUs and RCUs are spread evenly across partitions



DynamoDB – Throttling

- If we exceed provisioned RCUs or WCUs, we get "**ProvisionedThroughputExceededException**"
- Reasons:
  - **Hot Keys** – one partition key is being read too many times (e.g., popular item)
  - **Hot Partitions**
  - **Very large items,** remember RCU and WCU depends on size of items
- Solutions:
  - **Exponential backoff** when exception is encountered (already in SDK)
  - **Distribute partition keys** as much as possible
  - If RCU issue, we can **use DynamoDB Accelerator (DAX)**

R/W Capacity Modes – On-Demand

- Read/writes automatically scale up/down with your workloads
- No capacity planning needed (WCU / RCU)
- Unlimited WCU & RCU, no throttle, more expensive
- You're charged for reads/writes that you use in terms of **RRU** and **WRU**
- **Read Request Units (RRU)** – throughput for reads (same as RCU)
- **Write Request Units (WRU)** – throughput for writes (same as WCU)
- 2.5x more expensive than provisioned capacity (use with care)
- Use cases: unknown workloads, unpredictable application traffic, …

DynamoDB – Writing Data

- PutItem
  - Creates a new item or fully replace an old item (same Primary Key)
  - Consumes WCUs
- UpdateItem
  - Edits an existing item's attributes or adds a new item if it doesn't exist
  - Can be used to implement **Atomic Counters** – a numeric attribute that's unconditionally incremented
- Conditional Writes
  - Accept a write/update/delete only if conditions are met, otherwise returns an error
  - Helps with concurrent access to items
  - No performance impact

DynamoDB – Reading Data

- GetItem
  - Read based on Primary key
  - Primary Key can be HASH or HASH+RANGE
  - Eventually Consistent Read (default)
  - Option to use Strongly Consistent Reads (more RCU - might take longer)
  - ProjectionExpression can be specified to retrieve only certain attributes

DynamoDB – Reading Data (Query)

- **Query** returns items based on:
  - **KeyConditionExpression**
    - Partition Key value (**must be = operator**) – required
    - Sort Key value (=, <, <=, >, >=, Between, Begins with) – optional
  - **FilterExpression**
    - Additional filtering after the Query operation (before data returned to you)
    - Use only with non-key attributes (does not allow HASH or RANGE attributes)
- Returns:
  - The number of items specified in **Limit**
  - Or up to 1 MB of data
- Ability to do pagination on the results
- Can query table, a Local Secondary Index, or a Global Secondary Index

DynamoDB – Reading Data (Scan)

- **Scan** the entire table and then filter out data (inefficient)
- Returns up to 1 MB of data – use pagination to keep on reading
- Consumes a lot of RCU
- Limit impact using **Limit** or reduce the size of the result and pause
- For faster performance, use **Parallel Scan**
  - Multiple workers scan multiple data segments at the same time
  - Increases the throughput and RCU consumed
  - Limit the impact of parallel scans just like you would for Scans
- Can use **ProjectionExpression** & **FilterExpression** (no changes to RCU)

DynamoDB – Deleting Data

- DeleteItem
  - Delete an individual item
  - Ability to perform a conditional delete

- DeleteTable
  - Delete a whole table and all its items
  - Much quicker deletion than calling **DeleteItem** on all items

DynamoDB – Batch Operations

- Allows you to save in latency by reducing the number of API calls
- Operations are done in parallel for better efficiency
- Part of a batch can fail; in which case we need to try again for the failed items

- BatchWriteItem
  - Up to 25 **PutItem** and/or **DeleteItem** in one call
  - Up to 16 MB of data written, up to 400 KB of data per item
  - Can't update items (use **UpdateItem**)
  - UnprocessedItems for failed write operations (exponential backoff or add WCU)

- BatchGetItem
  - Return items from one or more tables
  - Up to 100 items, up to 16 MB of data
  - Items are retrieved in parallel to minimize latency
  - UnprocessedKeys for failed read operations (exponential backoff or add RCU)

DynamoDB – PartiQL

- SQL-compatible query language for DynamoDB
- Allows you to select, insert, update, and delete data in DynamoDB using SQL
- Run queries across multiple DynamoDB tables
- Run PartiQL queries from:
  - AWS Management Console
  - NoSQL Workbench for DynamoDB
  - DynamoDB APIs
  - AWS CLI
  - AWS SDK

```
SELECT OrderID, Total
FROM Orders
WHERE OrderID IN [1, 2, 3]
ORDER BY OrderID DESC
```

DynamoDB – Conditional Writes

- For **PutItem**, **UpdateItem**, **DeleteItem**, and **BatchWriteItem**
- You can specify a Condition expression to determine which items should be modified:
  - **attribute_exists**
  - **attribute_not_exists**
  - **attribute_type**
  - **contains** (for string)
  - **begins_with** (for string)
  - ProductCategory **IN** (:cat1, :cat2) **and** Price **between** :low **and** :high
  - **size** (string length)

- Note: Filter Expression filters the results of read queries, while Condition Expressions are for write operations

Conditional Writes – Example on Update Item

```
aws dynamodb update-item \
    --table-name ProductCatalog \
    --key '{ "Id": { "N": "456" } }' \
    --update-expression "SET Price = Price - :discount" \
    --condition-expression "Price > :limit" \
    --expression-attribute-values file://values.json
```

```
{
    ":discount": {
        "N": "150"
    },
    ":limit": {
        "N": "500"
    }
}
```

values.json

```
{
    "Id": {
        "N": "456"
    },
    "Price": {
        "N": "650"
    },
    "ProductCategory": {
        "S": "Sporting Goods"
    }
}
```

```
{
    "Id": {
        "N": "456"
    },
    "Price": {
        "N": "500"
    },
    "ProductCategory": {
        "S": "Sporting Goods"
    }
}
```

Conditional Writes – Example on Delete Item

- attribute_not_exists
  - Only succeeds if the attribute doesn't exist yet (no value)

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{ "Id": { "N": "456" } }' \
    --condition-expression "attribute_not_exists(Price)"
```

- attribute_exists
  - Opposite of attribute_not_exists

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{ "Id": { "N": "456" } }' \
    --condition-expression "attribute_exists(ProductReviews.OneStar)"
```

Conditional Writes – Do Not Overwrite Elements

- attribute_not_exists(partition_key)
  - Make sure the item isn't overwritten

- attribute_not_exists(partition_key) and attribute_not_exists(sort_key)
  - Make sure the partition / sort key combination is not overwritten

Conditional Writes – Example Complex Condition

```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{ "Id": { "N": "456" } }' \
    --condition-expression "(ProductCategory IN (:cat1, :cat2)) and (Price between :lo and :hi)" \
    --expression-attribute-values file://values.json
```

```
{
    ":cat1": {
        "S": "Sporting Goods"
    },
    ":cat2": {
        "S": "Gardening Supplies"
    },
    ":lo": {
        "N": "500"
    },
    ":hi": {
        "N": "600"
    }
}
```

values.json

```
{
    "Id": {
        "N": "456"
    },
    "Price": {
        "N": "650"
    },
    "ProductCategory": {
        "S": "Sporting Goods"
    }
}
```

Conditional Writes – Example of String Comparisons

- **begins_with** – check if prefix matches
- **contains** – check if string is contained in another string

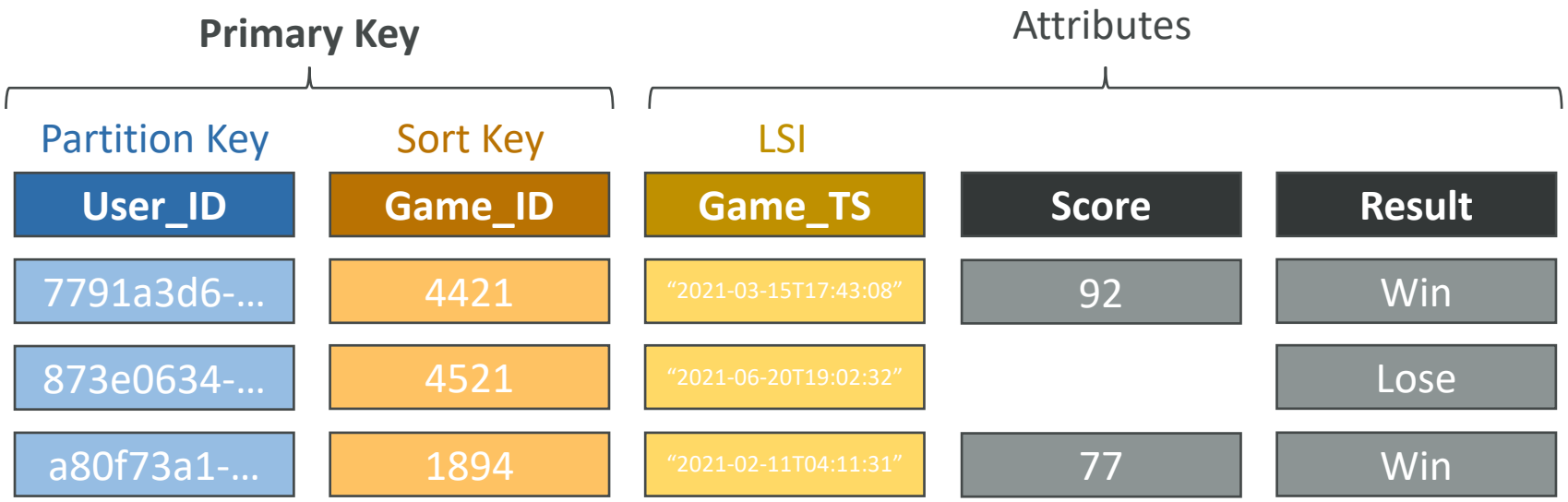
```
aws dynamodb delete-item \
    --table-name ProductCatalog \
    --key '{ "Id": { "N": "456" } }' \
    --condition-expression "begins_with(Pictures.FrontView, :v_sub)" \
    --expression-attribute-values file://values.json
```

```
{
    ":v_sub": {
        "S": "http://"
    }
}
```

values.json

DynamoDB – Local Secondary Index (LSI)

- **Alternative Sort Key** for your table (same **Partition Key** as that of base table)
- The Sort Key consists of one scalar attribute (String, Number, or Binary)
- Up to 5 Local Secondary Indexes per table
- **Must be defined at table creation time**
- **Attribute Projections** – can contain some or all the attributes of the base table (KEYS_ONLY, INCLUDE, ALL)



DynamoDB – Global Secondary Index (GSI)

- **Alternative Primary Key (HASH or HASH+RANGE)** from the base table
- Speed up queries on non-key attributes
- The Index Key consists of scalar attributes (String, Number, or Binary)
- **Attribute Projections** – some or all the attributes of the base table (**KEYS_ONLY**, **INCLUDE**, **ALL**)
- Must provision RCUs & WCUs for the index
- **Can be added/modified after table creation**

Partition Key	Sort Key	Attributes
User_ID	**Game_ID**	**Game_TS**
7791a3d6-...	4421	"2021-03-15T17:43:08"
873e0634-...	4521	"2021-06-20T19:02:32"
a80f73a1-...	1894	"2021-02-11T04:11:31"

TABLE (query by "User_ID")

Partition Key	Sort Key	Attributes
Game_ID	**Game_TS**	**User_ID**
4421	"2021-03-15T17:43:08"	7791a3d6-...
4521	"2021-06-20T19:02:32"	873e0634-...
1894	"2021-02-11T04:11:31"	a80f73a1-...

INDEX GSI (query by "Game_ID")

DynamoDB – Indexes and Throttling

- Global Secondary Index (GSI):
  - If the writes are throttled on the GSI, then the main table will be throttled!
  - Even if the WCU on the main tables are fine
  - Choose your GSI partition key carefully!
  - Assign your WCU capacity carefully!

- Local Secondary Index (LSI):
  - Uses the WCUs and RCUs of the main table
  - No special throttling considerations

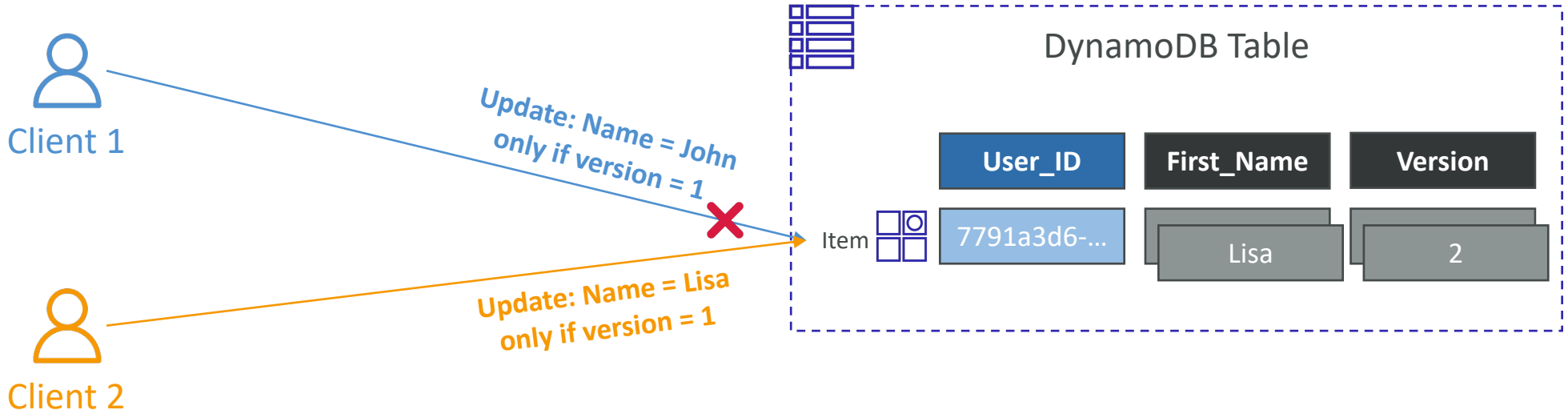
DynamoDB - PartiQL

- Use a SQL-like syntax to manipulate DynamoDB tables

- Supports some (but not all) statements:
  - INSERT
  - UPDATE
  - SELECT
  - DELETE
- It supports Batch operations

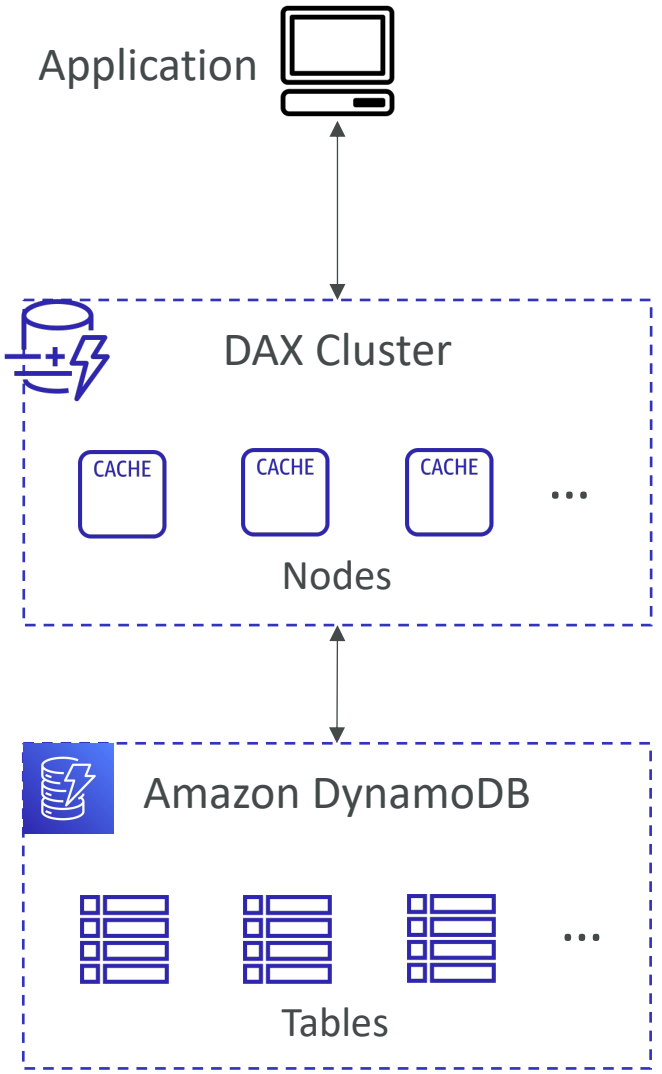
DynamoDB – Optimistic Locking

- DynamoDB has a feature called "**Conditional Writes**"
- A strategy to ensure an item hasn't changed before you update/delete it
- Each item has an attribute that acts as a *version number*

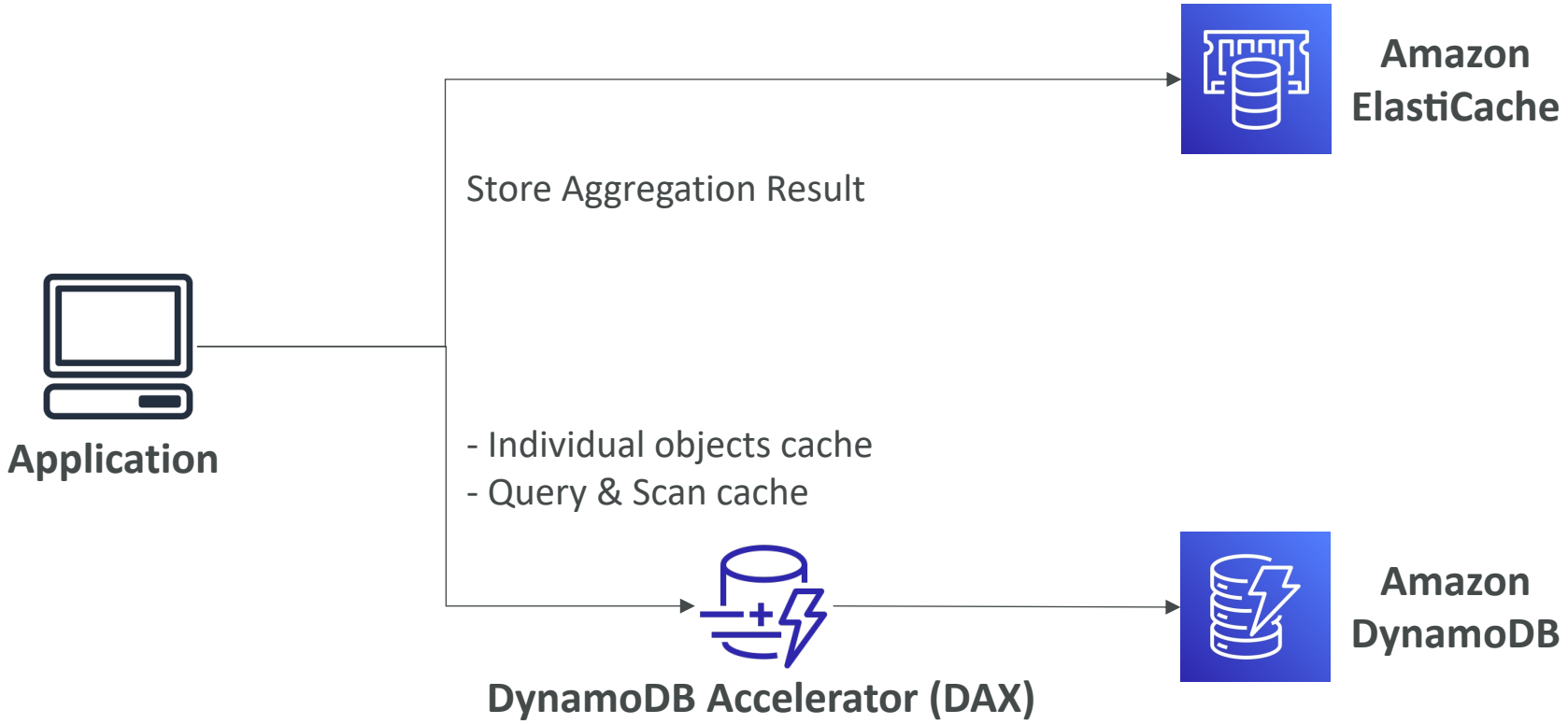


DynamoDB Accelerator (DAX)

- Fully-managed, highly available, seamless in-memory cache for DynamoDB
- Microseconds latency for cached reads & queries
- Doesn't require application logic modification (compatible with existing DynamoDB APIs)
- Solves the "Hot Key" problem (too many reads)
- 5 minutes TTL for cache (default)
- Up to 10 nodes in the cluster
- Multi-AZ (3 nodes minimum recommended for production)
- Secure (Encryption at rest with KMS, VPC, IAM, CloudTrail, …)



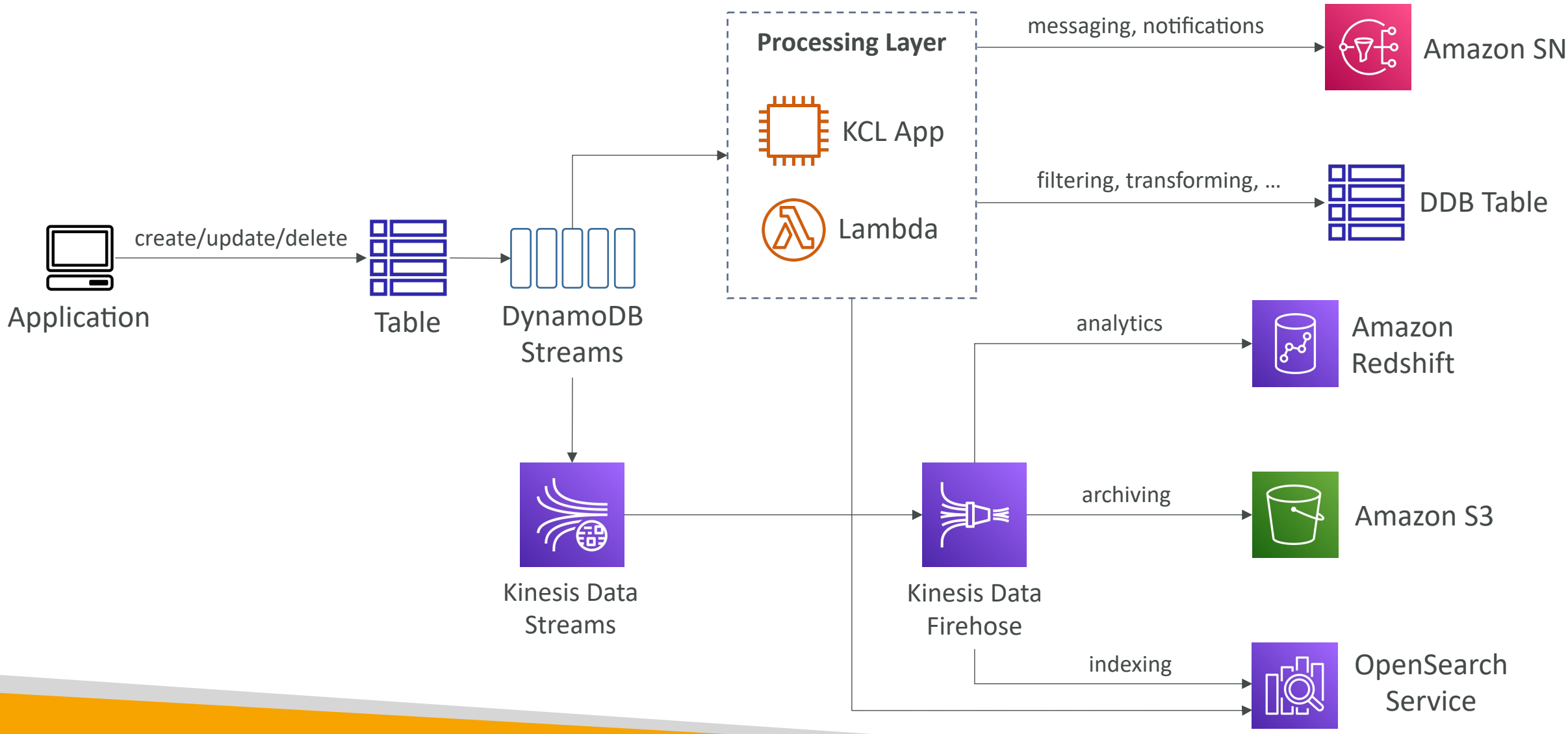
DynamoDB Accelerator (DAX) vs. ElastiCache



DynamoDB Streams

- Ordered stream of item-level modifications (create/update/delete) in a table
- Stream records can be:
  - Sent to **Kinesis Data Streams**
  - Read by **AWS Lambda**
  - Read by **Kinesis Client Library applications**
- Data Retention for up to 24 hours
- Use cases:
  - react to changes in real-time (welcome email to users)
  - Analytics
  - Insert into derivative tables
  - Insert into OpenSearch Service
  - Implement cross-region replication

DynamoDB Streams



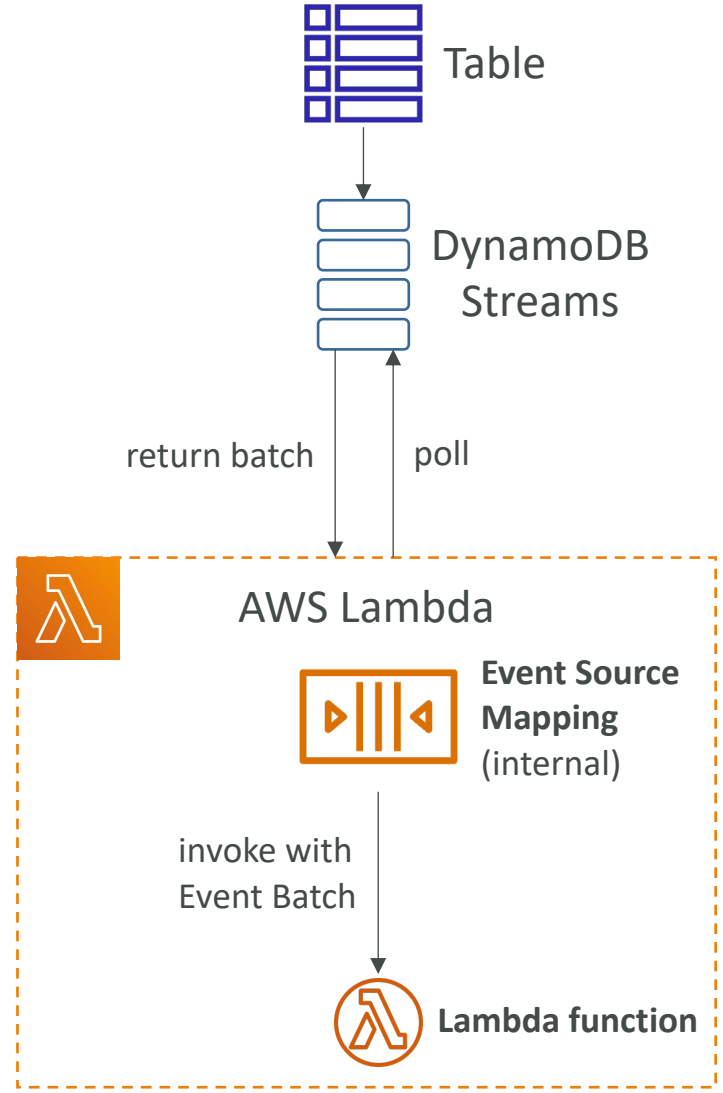
DynamoDB Streams

- Ability to choose the information that will be written to the stream:
  - KEYS_ONLY – only the key attributes of the modified item
  - NEW_IMAGE – the entire item, as it appears after it was modified
  - OLD_IMAGE – the entire item, as it appeared before it was modified
  - NEW_AND_OLD_IMAGES – both the new and the old images of the item
- DynamoDB Streams are made of shards, just like Kinesis Data Streams
- You don't provision shards, this is automated by AWS

- Records are not retroactively populated in a stream after enabling it

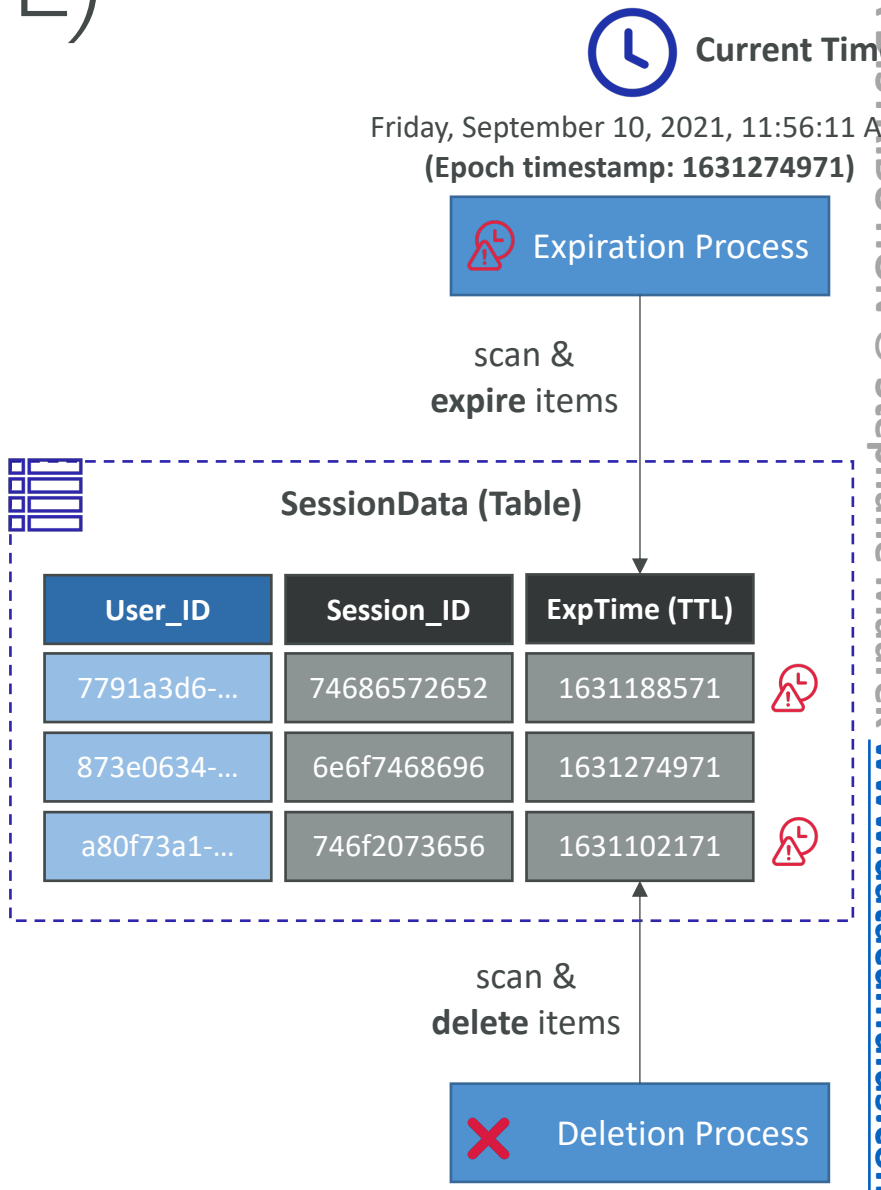
DynamoDB Streams & AWS Lambda

- You need to define an **Event Source Mapping** to read from a DynamoDB Streams
- You need to ensure the Lambda function has the **appropriate permissions**
- Your Lambda function is invoked synchronously



DynamoDB – Time To Live (TTL)

- Automatically delete items after an expiry timestamp
- Doesn't consume any WCUs (i.e., no extra cost)
- The TTL attribute must be a **"Number"** data type with **"Unix Epoch timestamp"** value
- Expired items deleted within 48 hours of expiration
- Expired items, that haven't been deleted, appears in reads/queries/scans (if you don't want them, filter them out)
- Expired items are deleted from both LSIs and GSIs
- A delete operation for each expired item enters the DynamoDB Streams (can help recover expired items)
- Use cases: reduce stored data by keeping only current items, adhere to regulatory obligations, …



DynamoDB CLI – Good to Know

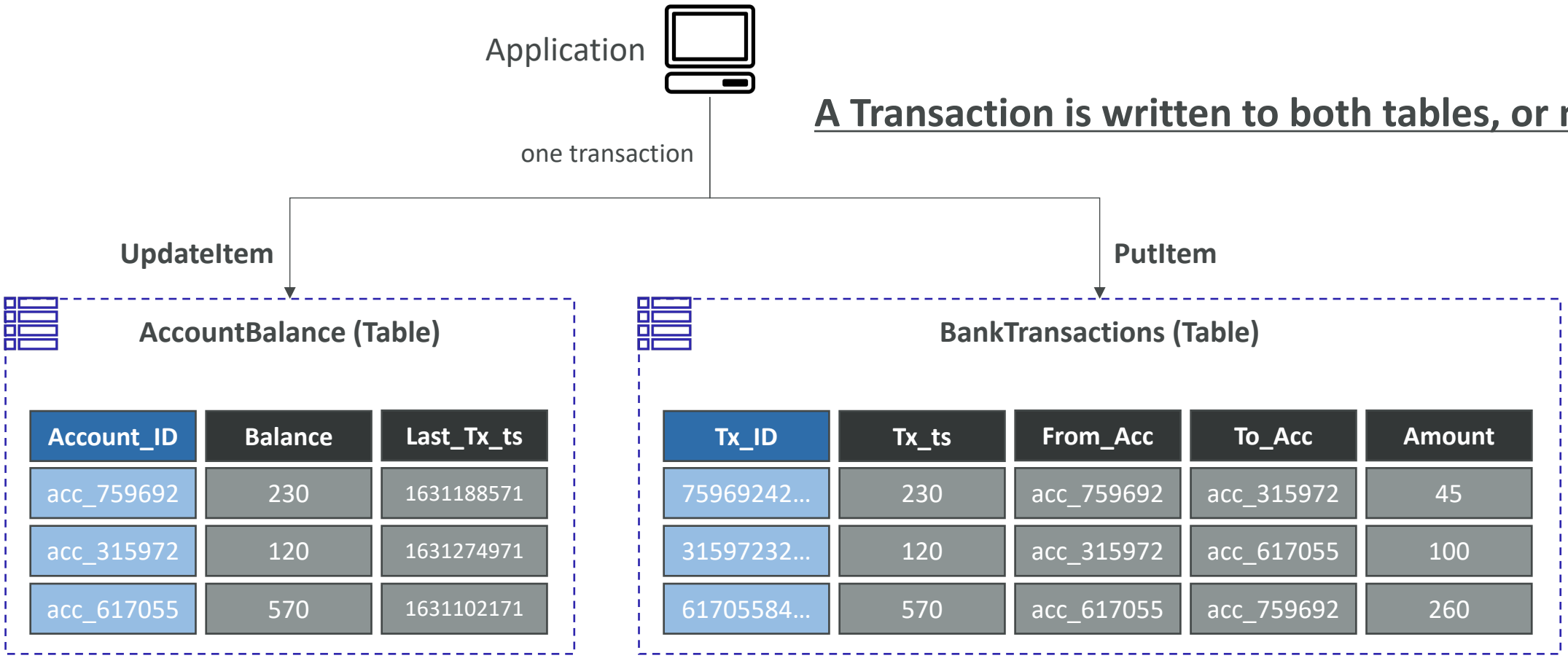
- **--projection-expression:** one or more attributes to retrieve
- **--filter-expression:** filter items before returned to you

- General AWS CLI Pagination options (e.g., DynamoDB, S3, …)
  - **--page-size:** specify that AWS CLI retrieves the full list of items but with a larger number of API calls instead of one API call (default: 1000 items)
  - **--max-items:** max. number of items to show in the CLI (returns **NextToken**)
  - **--starting-token:** specify the last **NextToken** to retrieve the next set of items

DynamoDB Transactions

- Coordinated, all-or-nothing operations (add/update/delete) to multiple items across one or more tables
- Provides Atomicity, Consistency, Isolation, and Durability (ACID)
- **Read Modes** – Eventual Consistency, Strong Consistency, Transactional
- **Write Modes** – Standard, Transactional
- **Consumes 2x WCUs & RCUs**
  - DynamoDB performs 2 operations for every item (prepare & commit)
- Two operations:
  - **TransactGetItems** – one or more **GetItem** operations
  - **TransactWriteItems** – one or more **PutItem**, **UpdateItem**, and **DeleteItem** operations
- Use cases: financial transactions, managing orders, multiplayer games, …

DynamoDB Transactions



DynamoDB Transactions – Capacity Computations

- Important for the exam!

- **Example 1:** 3 Transactional writes per second, with item size 5 KB
  - We need $3 * \left(\frac{5\ KB}{1\ KB}\right) * 2\ (transactional\ cost) = 30\ WCUs$

- **Example 2:** 5 Transaction reads per second , with item size 5 KB
  - We need $5 * \left(\frac{8\ KB}{4\ KB}\right) * 2\ (transactional\ cost) = 20\ RCUs$
  - (5 gets rounded to the upper 4 KB)

DynamoDB as Session State Cache

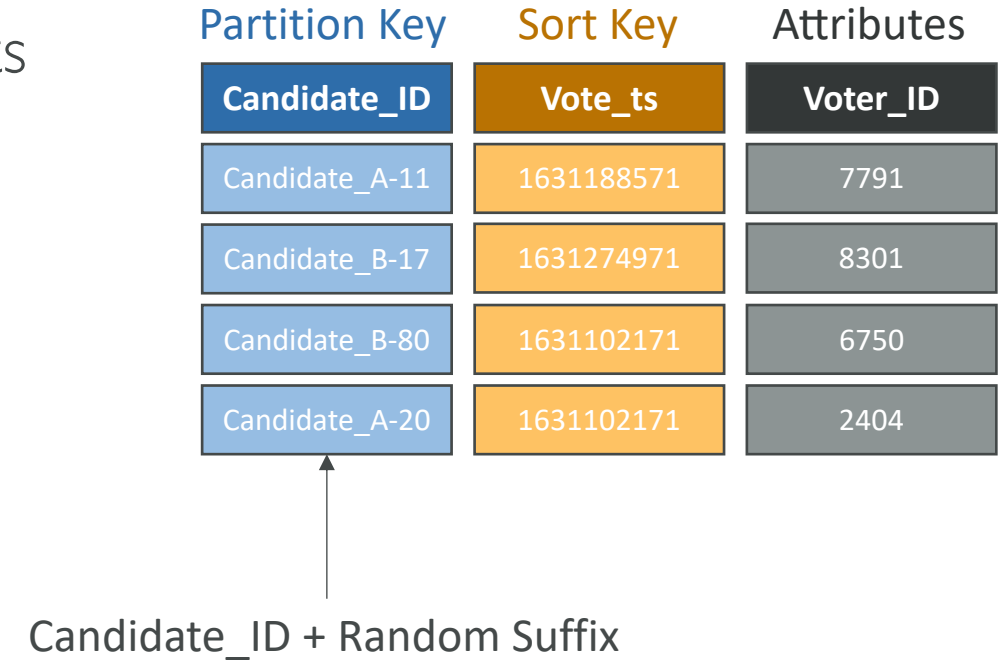
- It's common to use DynamoDB to store session states

- vs. ElastiCache
  - ElastiCache is in-memory, but DynamoDB is serverless
  - Both are key/value stores
- vs. EFS
  - EFS must be attached to EC2 instances as a network drive
- vs. EBS & Instance Store
  - EBS & Instance Store can only be used for local caching, not shared caching
- vs. S3
  - S3 is higher latency, and not meant for small objects

DynamoDB Write Sharding

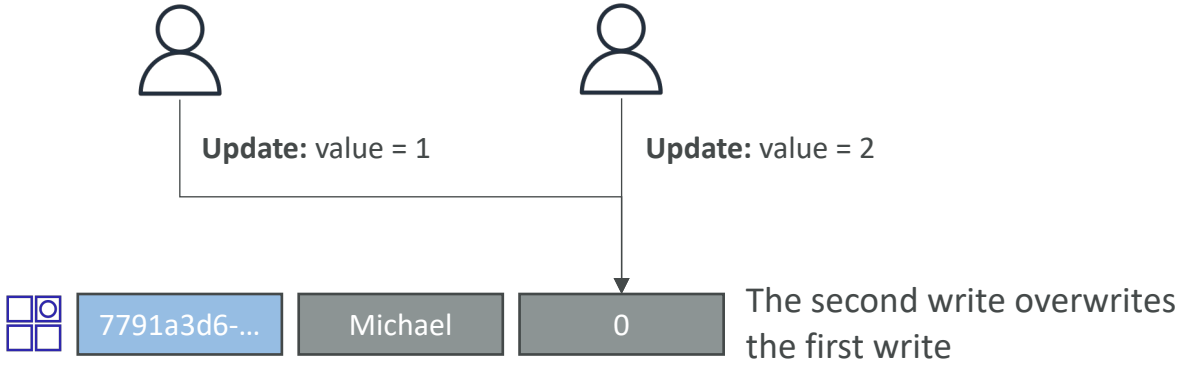
- Imagine we have a voting application with two candidates, **candidate A** and **candidate B**
- If **Partition Key** is **"Candidate_ID"**, this results into two partitions, which will generate issues (e.g., Hot Partition)

- A strategy that allows better distribution of items evenly across partitions
- **Add a suffix to Partition Key value**
- Two methods:
  - Sharding Using Random Suffix
  - Sharding Using Calculated Suffix



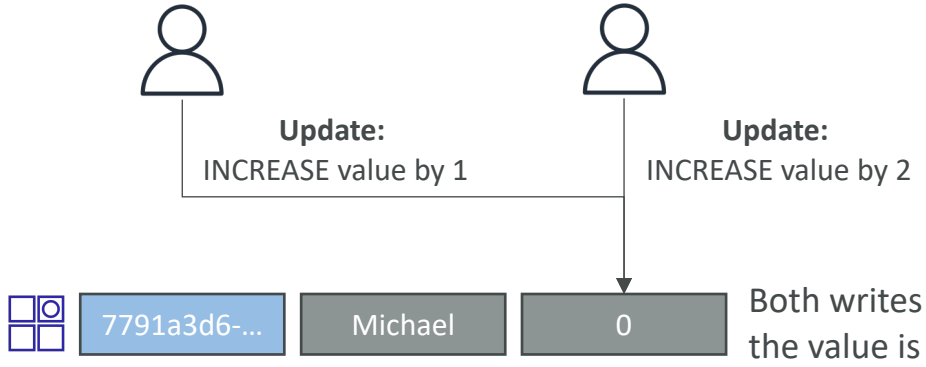
DynamoDB – Write Types

Concurrent Writes



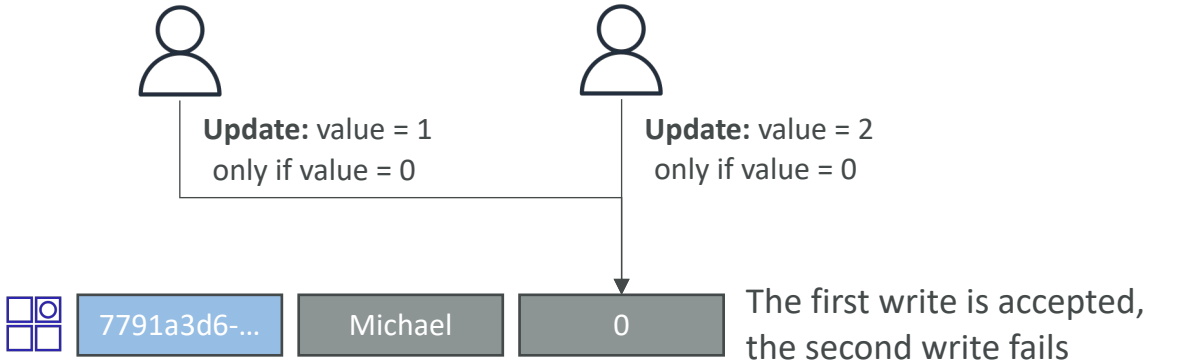
The second write overwrites the first write

Atomic Writes



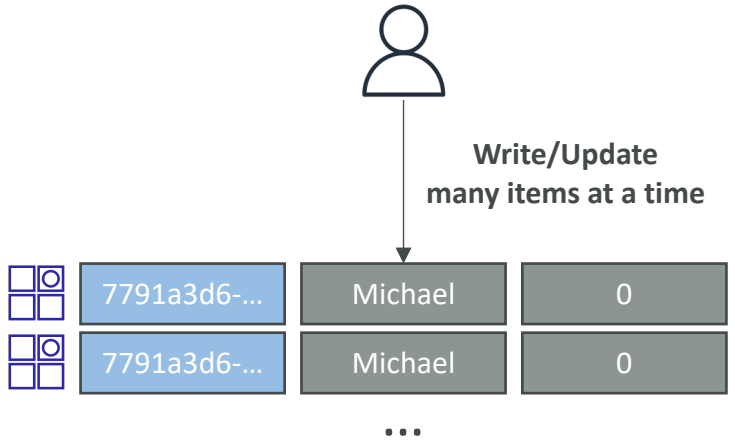
Both writes succeed, the value is increased by 3

Conditional Writes

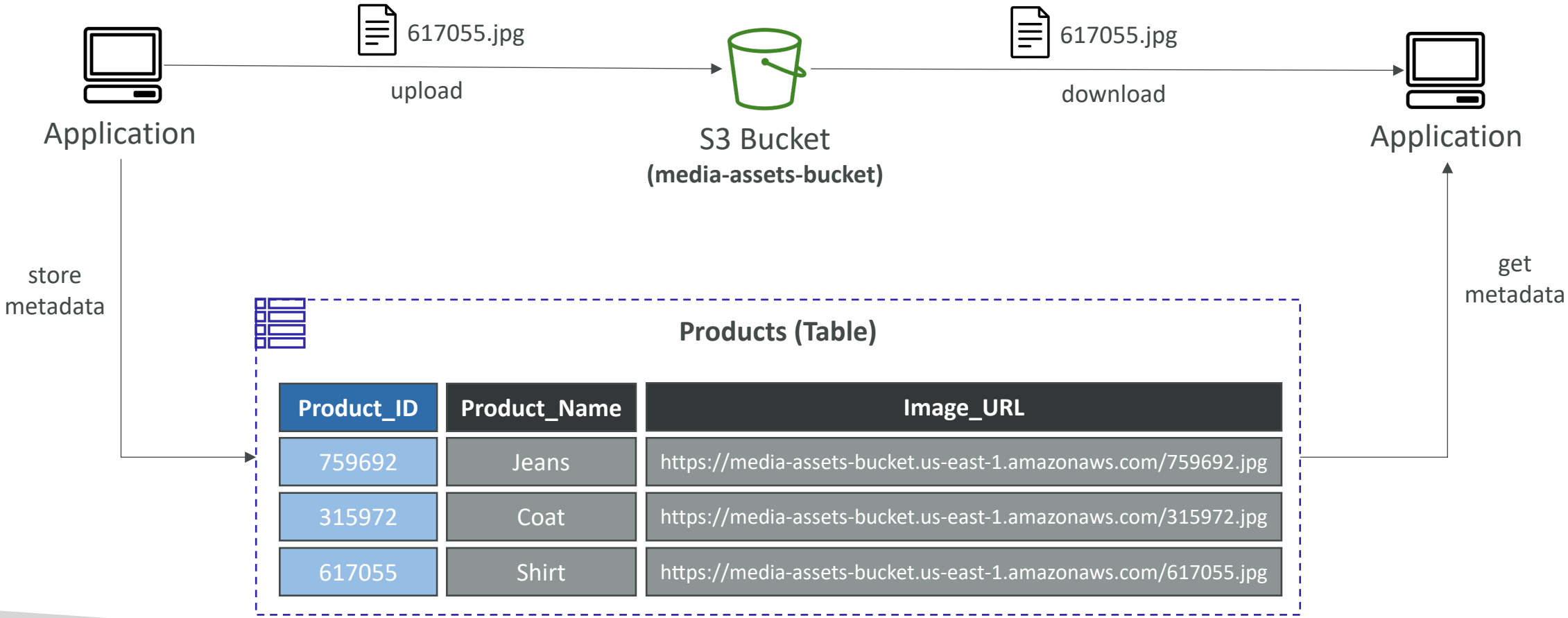


The first write is accepted, the second write fails

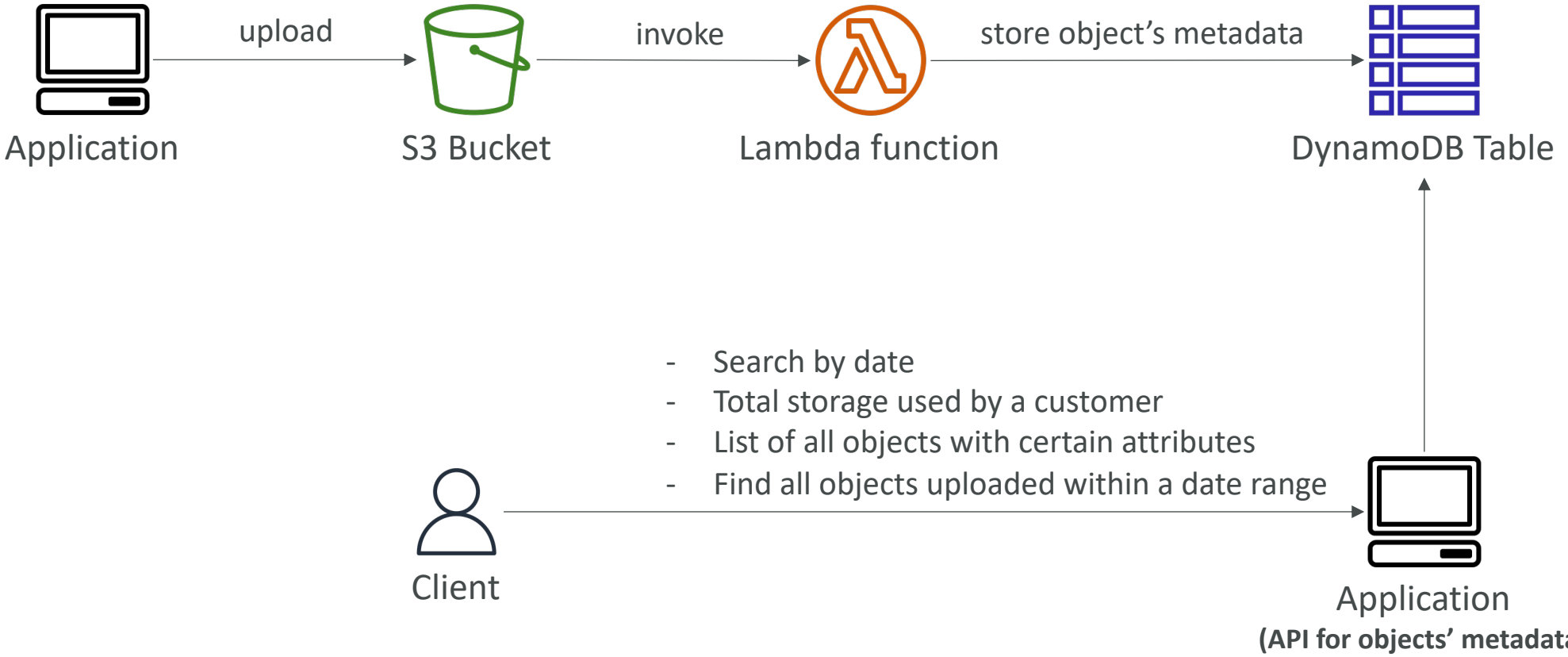
Batch Writes



DynamoDB – Large Objects Pattern

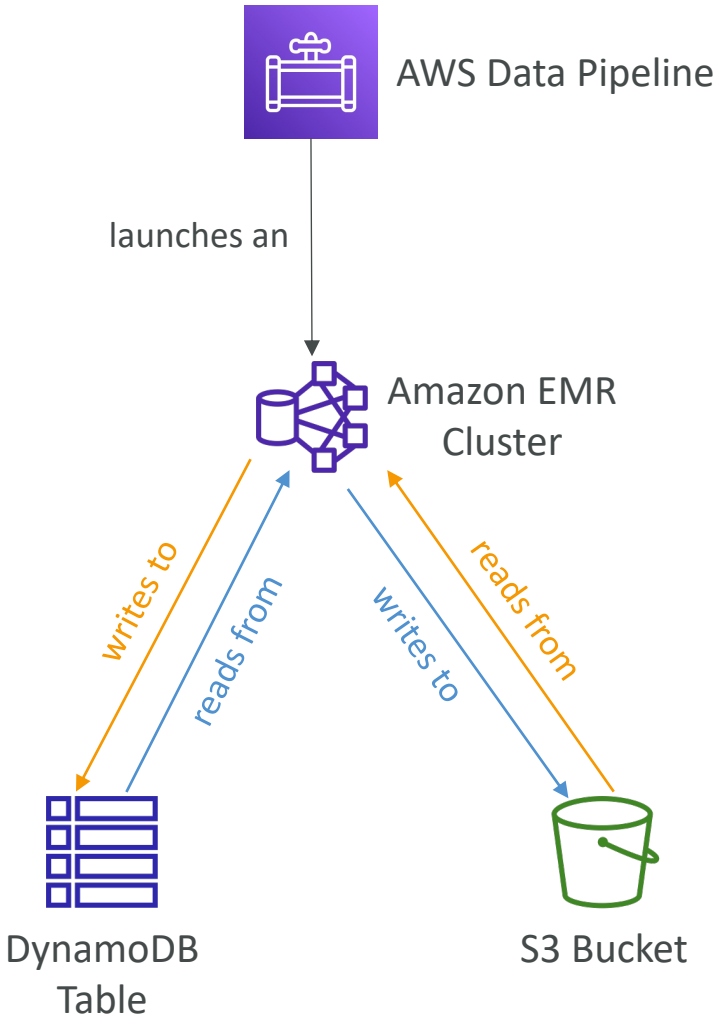


DynamoDB – Indexing S3 Objects Metadata



DynamoDB Operations

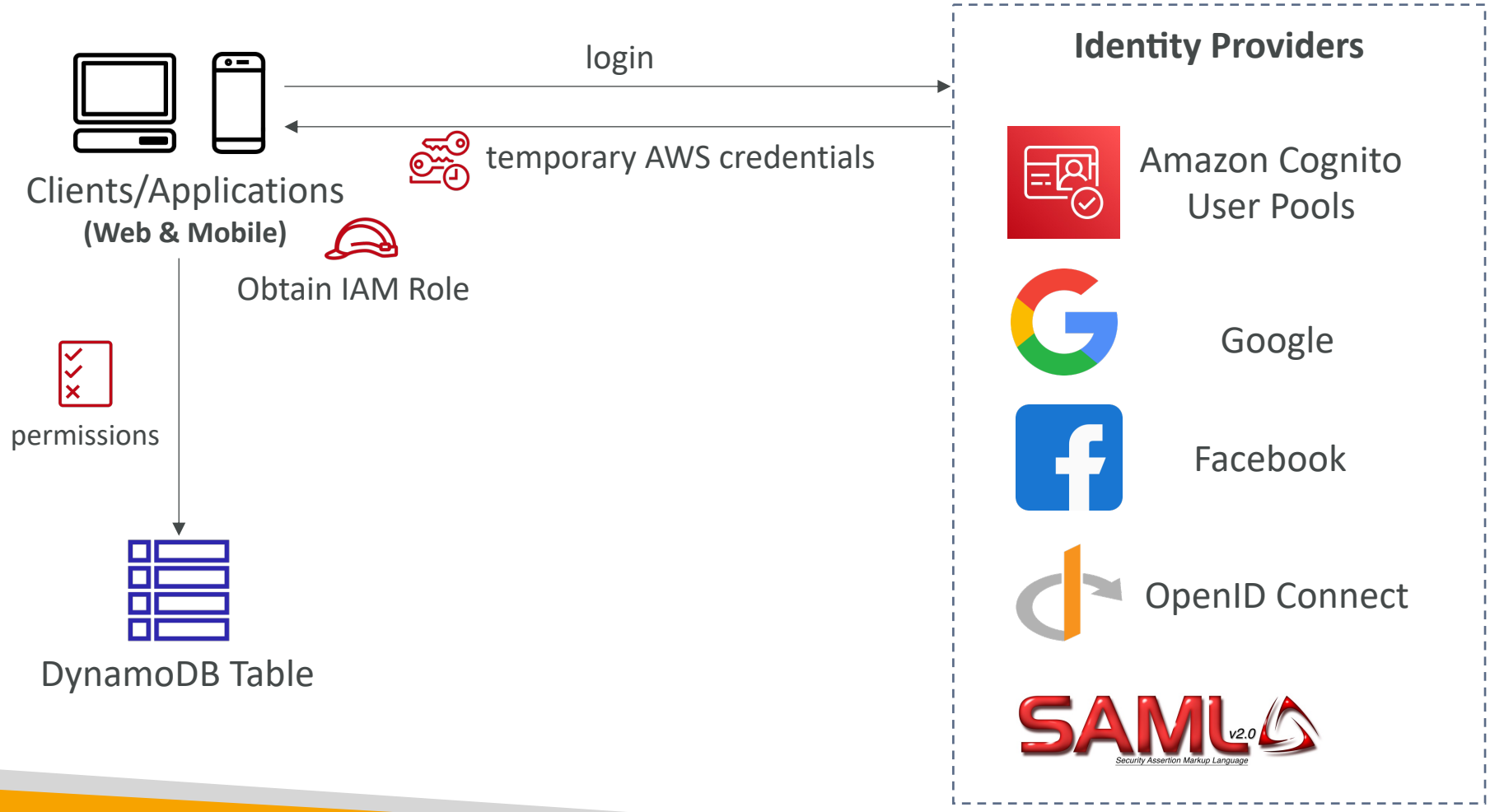
- Table Cleanup
  - Option 1: Scan + DeleteItem
    - Very slow, consumes RCU & WCU, expensive
  - Option 2: Drop Table + Recreate table
    - Fast, efficient, cheap
- Copying a DynamoDB Table
  - Option 1: Using AWS Data Pipeline
  - Option 2: Backup and restore into a new table
    - Takes some time
  - Option 3: Scan + PutItem or BatchWriteItem
    - Write your own code



DynamoDB – Security & Other Features

- Security
  - VPC Endpoints available to access DynamoDB without using the Internet
  - Access fully controlled by IAM
  - Encryption at rest using AWS KMS and in-transit using SSL/TLS
- Backup and Restore feature available
  - Point-in-time Recovery (PITR) like RDS
  - No performance impact
- Global Tables
  - Multi-region, multi-active, fully replicated, high performance
- DynamoDB Local
  - Develop and test apps locally without accessing the DynamoDB web service (without Internet)
- AWS Database Migration Service (AWS DMS) can be used to migrate to DynamoDB (from MongoDB, Oracle, MySQL, S3, …)

DynamoDB – Users Interact with DynamoDB Directly



DynamoDB – Fine-Grained Access Control

- Using **Web Identity Federation** or **Cognito Identity Pools**, each user gets AWS credentials
- You can assign an IAM Role to these users with a **Condition** to limit their API access to DynamoDB
- **LeadingKeys** – limit row-level access for users on the Primary Key
- **Attributes** – limit specific attributes the user can see

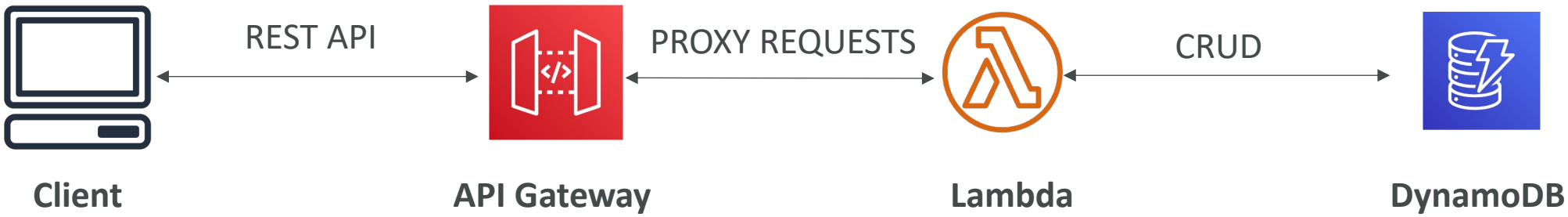
```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetIem", "dynamodb:BatchGetItem", "dynamodb:Query",
                "dynamodb:PutItem", "dynamodb:UpdateItem", "dynamodb:DeleteItem",
                "dynamodb:BatchWriteItem"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/MyTable",
            "Condition": {
                "ForAllValues:StringEquals": {
                    "dynamodb:LeadingKeys": ["${cognito-identity.amazonaws.com:sub}"]
                }
            }
        }
    ]
}
```

More at: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/specifying-conditions.html

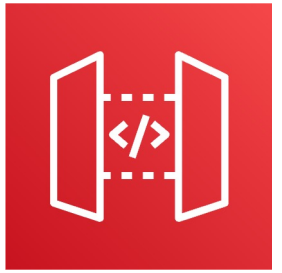
Amazon API Gateway

Build, Deploy and Manage APIs

Example: Building a Serverless API



AWS API Gateway

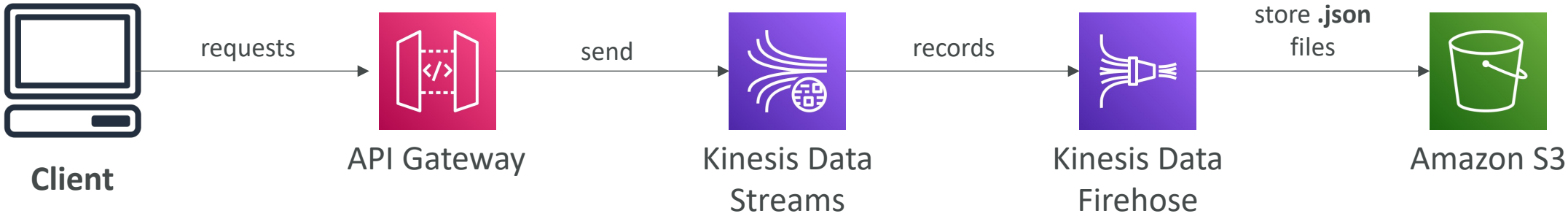


- AWS Lambda + API Gateway: No infrastructure to manage
- Support for the WebSocket Protocol
- Handle API versioning (v1, v2…)
- Handle different environments (dev, test, prod…)
- Handle security (Authentication and Authorization)
- Create API keys, handle request throttling
- Swagger / Open API import to quickly define APIs
- Transform and validate requests and responses
- Generate SDK and API specifications
- Cache API responses

API Gateway – Integrations High Level

- Lambda Function
  - Invoke Lambda function
  - Easy way to expose REST API backed by AWS Lambda
- HTTP
  - Expose HTTP endpoints in the backend
  - Example: internal HTTP API on premise, Application Load Balancer…
  - Why? Add rate limiting, caching, user authentications, API keys, etc…
- AWS Service
  - Expose any AWS API through the API Gateway
  - Example: start an AWS Step Function workflow, post a message to SQS
  - Why? Add authentication, deploy publicly, rate control…

API Gateway – AWS Service Integration Kinesis Data Streams example



API Gateway - Endpoint Types

- **Edge-Optimized (default):** For global clients
  - Requests are routed through the CloudFront Edge locations (improves latency)
  - The API Gateway still lives in only one region
- **Regional:**
  - For clients within the same region
  - Could manually combine with CloudFront (more control over the caching strategies and the distribution)
- **Private:**
  - Can only be accessed from your VPC using an interface VPC endpoint (ENI)
  - Use a resource policy to define access

API Gateway – Security

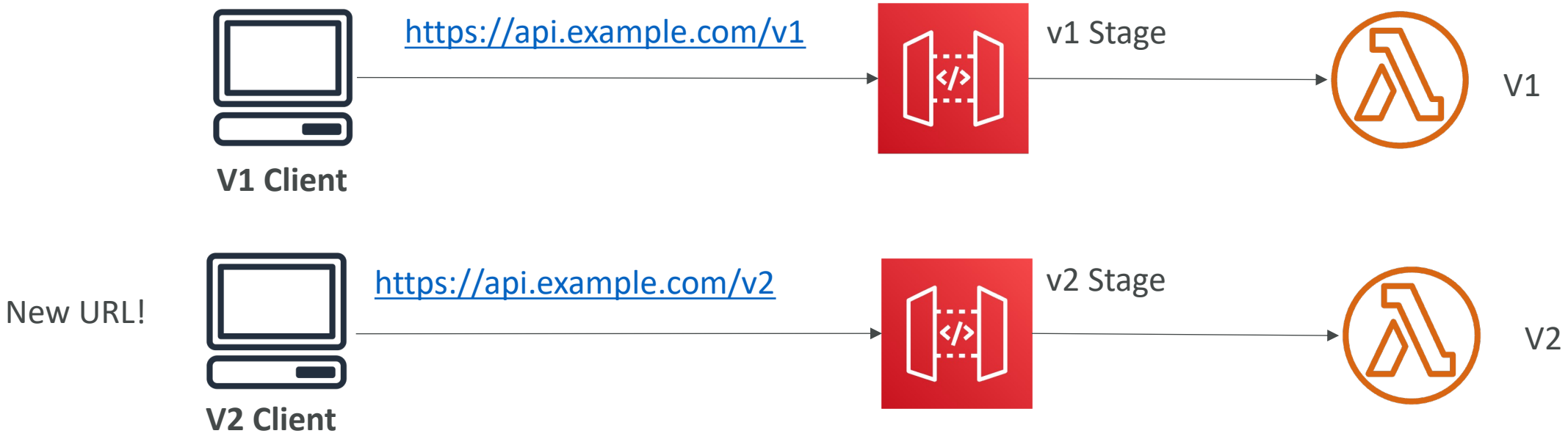
- User Authentication through
  - IAM Roles (useful for internal applications)
  - Cognito (identity for external users – example mobile users)
  - Custom Authorizer (your own logic)

- Custom Domain Name HTTPS security through integration with AWS Certificate Manager (ACM)
  - If using Edge-Optimized endpoint, then the certificate must be in **us-east-1**
  - If using Regional endpoint, the certificate must be in the API Gateway region
  - Must setup CNAME or A-alias record in Route 53

API Gateway – Deployment Stages

- Making changes in the API Gateway does not mean they're effective
- You need to make a "deployment" for them to be in effect
- It's a common source of confusion
- Changes are deployed to "Stages" (as many as you want)
- Use the naming you like for stages (dev, test, prod)
- Each stage has its own configuration parameters
- Stages can be rolled back as a history of deployments is kept

API Gateway – Stages v1 and v2
API breaking change

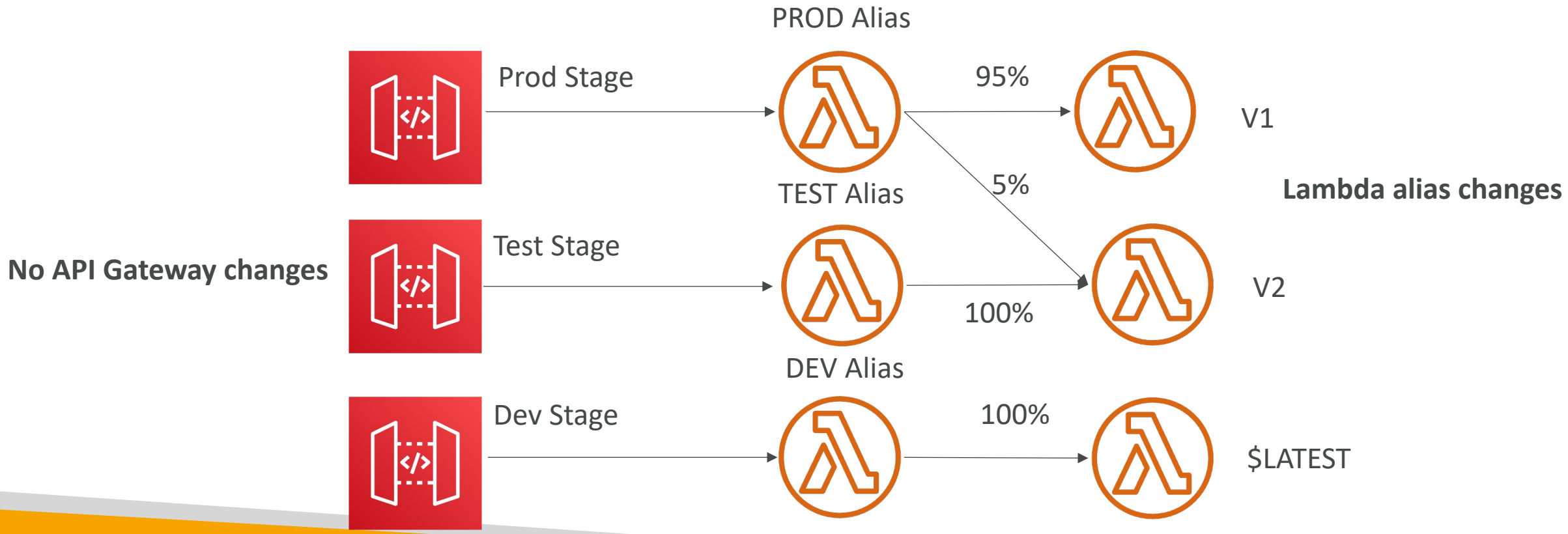


API Gateway – Stage Variables

- Stage variables are like environment variables for API Gateway
- Use them to change often changing configuration values
- They can be used in:
  - Lambda function ARN
  - HTTP Endpoint
  - Parameter mapping templates
- Use cases:
  - Configure HTTP endpoints your stages talk to (dev, test, prod…)
  - Pass configuration parameters to AWS Lambda through mapping templates
- Stage variables are passed to the "context" object in AWS Lambda
- Format: ${stageVariables.variableName}

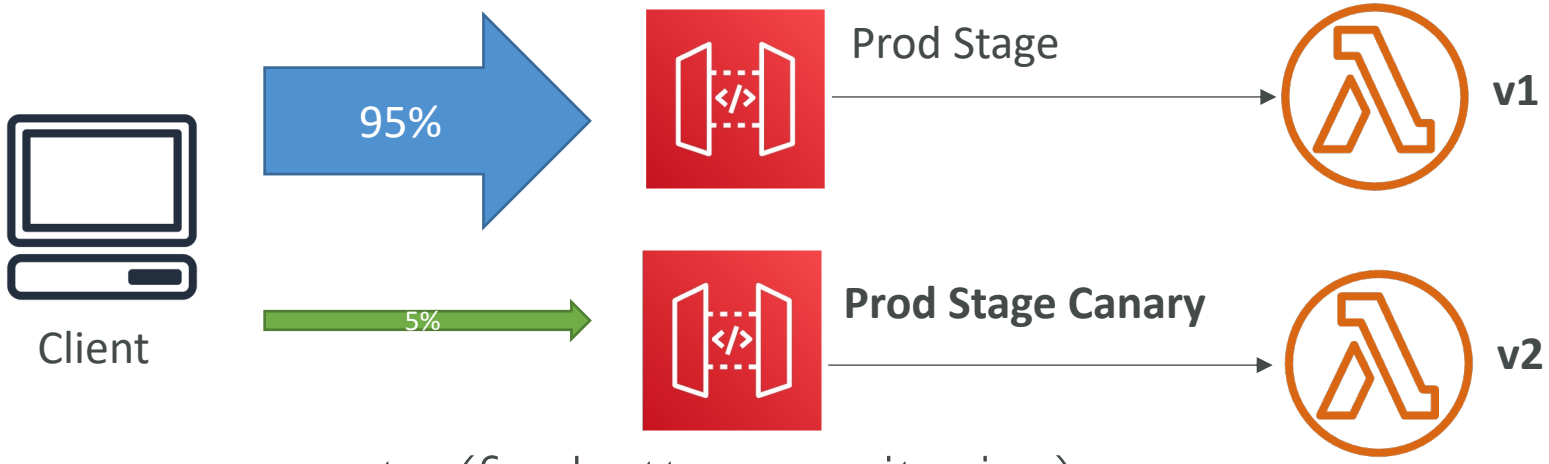
API Gateway Stage Variables & Lambda Aliases

- We create a **stage variable** to indicate the corresponding Lambda alias
- Our API gateway will automatically invoke the right Lambda function!



API Gateway – Canary Deployment

- Possibility to enable canary deployments for any stage (usually prod)
- Choose the % of traffic the canary channel receives

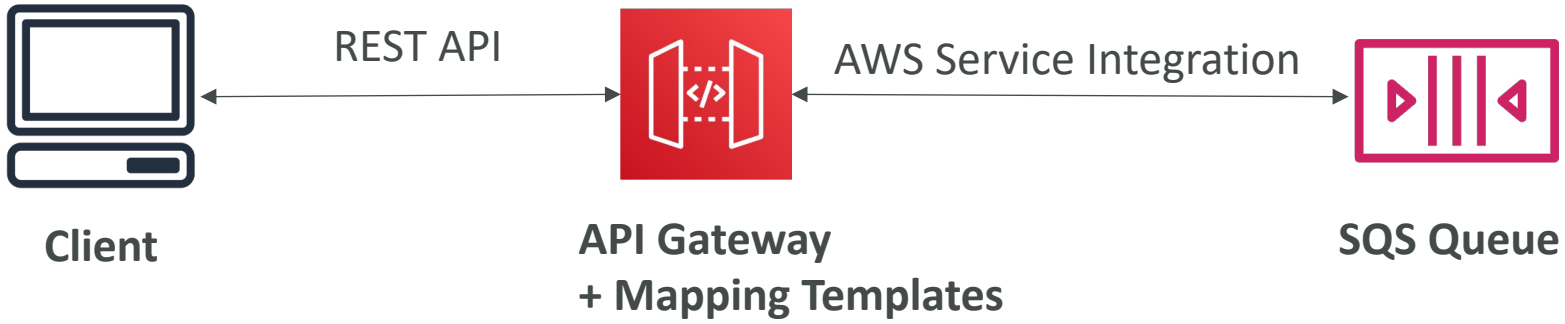


- Metrics & Logs are separate (for better monitoring)
- Possibility to override stage variables for canary
- This is blue / green deployment with AWS Lambda & API Gateway

API Gateway - Integration Types

- Integration Type **MOCK**
  - API Gateway returns a response without sending the request to the backend

- Integration Type **HTTP / AWS (Lambda & AWS Services)**
  - you must configure both the integration request and integration response
  - Setup data mapping using **mapping templates** for the request & response



API Gateway - Integration Types

- Integration Type **AWS_PROXY (Lambda Proxy)**:
  - incoming request from the client is the input to Lambda
  - The function is responsible for the logic of request / response
  - No mapping template, headers, query string parameters… are passed as arguments

```
{
    "resource": "Resource path",
    "path": "Path parameter",
    "httpMethod": "Incoming request's method name",
    "headers": "String containing incoming request headers",
    "multiValueHeaders": "List of strings containing incomin
    "queryStringParameters": "query string parameters ",
    "multiValueQueryStringParameters": "List of query string
    "pathParameters":  "path parameters",
    "stageVariables": "Applicable stage variables",
    "requestContext": "Request context, including authorizer
    "body": "A JSON string of the request payload.",
    "isBase64Encoded": "A boolean flag"
}
```

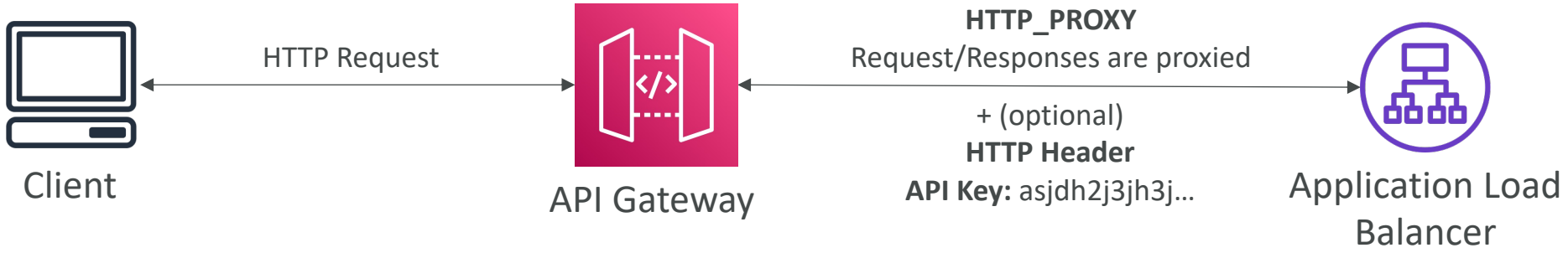
Lambda function invocation payload

```
{
    "isBase64Encoded": "true|false",
    "statusCode": "httpStatusCode",
    "headers": { "headerName": "headerValue", ... },
    "multiValueHeaders": { "headerName": ["headerValue", "heade
    "body": "..."
}
```

Lambda function expected response

API Gateway - Integration Types

- Integration Type **HTTP_PROXY**
  - No mapping template
  - The HTTP request is passed to the backend
  - The HTTP response from the backend is forwarded by API Gateway
  - Possibility to add HTTP Headers if need be (ex: API key)

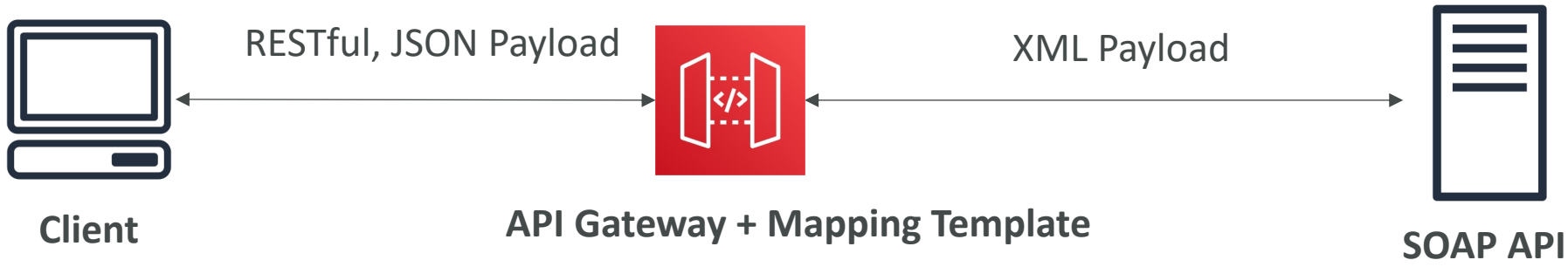


Mapping Templates (AWS & HTTP Integration)

- Mapping templates can be used to modify request / responses
- Rename / Modify **query string parameters**
- Modify **body content**
- **Add headers**
- Uses Velocity Template Language (VTL): for loop, if etc…
- Filter output results (remove unnecessary data)
- Content-Type can be set to **application/json** or **application/xml**

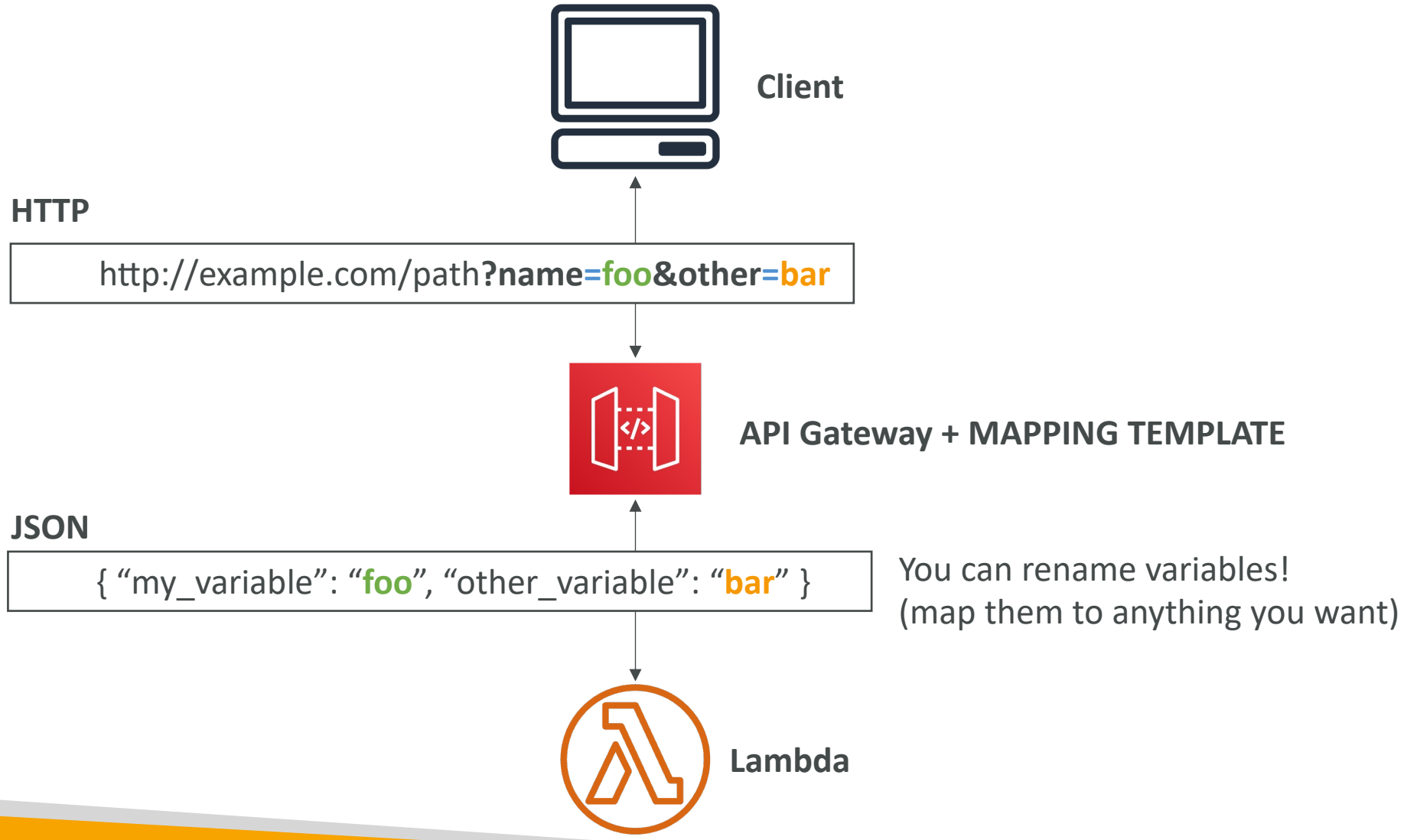
Mapping Example: JSON to XML with SOAP

- SOAP API are XML based, whereas REST API are JSON based



- In this case, API Gateway should:
  - Extract data from the request: either path, payload or header
  - Build SOAP message based on request data (mapping template)
  - Call SOAP service and receive XML response
  - Transform XML response to desired format (like JSON), and respond to the user

Mapping Example: Query String parameters



API Gateway - Open API spec

- Common way of defining REST APIs, using API definition as code
- Import existing OpenAPI 3.0 spec to API Gateway
  - Method
  - Method Request
  - Integration Request
  - Method Response
  - + AWS extensions for API gateway and setup every single option
- Can export current API as OpenAPI spec
- OpenAPI specs can be written in YAML or JSON
- Using OpenAPI we can generate SDK for our applications

REST API – Request Validation

- You can configure API Gateway to perform basic validation of an API request before proceeding with the integration request
- When the validation fails, API Gateway immediately fails the request
  - Returns a 400-error response to the caller
- This reduces unnecessary calls to the backend
- Checks:
  - The required request parameters in the URI, query string, and headers of an incoming request are included and non-blank
  - The applicable request payload adheres to the configured JSON Schema request model of the method

REST API – RequestValidation – OpenAPI

- Setup request validation by importing OpenAPI definitions file

```
{
    "openapi": "3.0.0",
    "info": {
        "title": "ReqValidation Sample",
        "version": "1.0.0"
    },
    "servers": [ ... ],
    "x-amazon-apigateway-request-validators": {
        "all": {
            "validateRequestBody": true,
            "validateRequestParameters": true
        },
        "params-only": {
            "validateRequestBody": false,
            "validateRequestParameters": true
        }
    }
}
```

Defines the Validators

```
{
    "openapi": "3.0.0",
    "info": {
        "title": "ReqValidation Sample",
        "version": "1.0.0"
    },
    "servers": [ ... ],
    "x-amazon-apigateway-request-validator": "params-only",
    ...
}
```

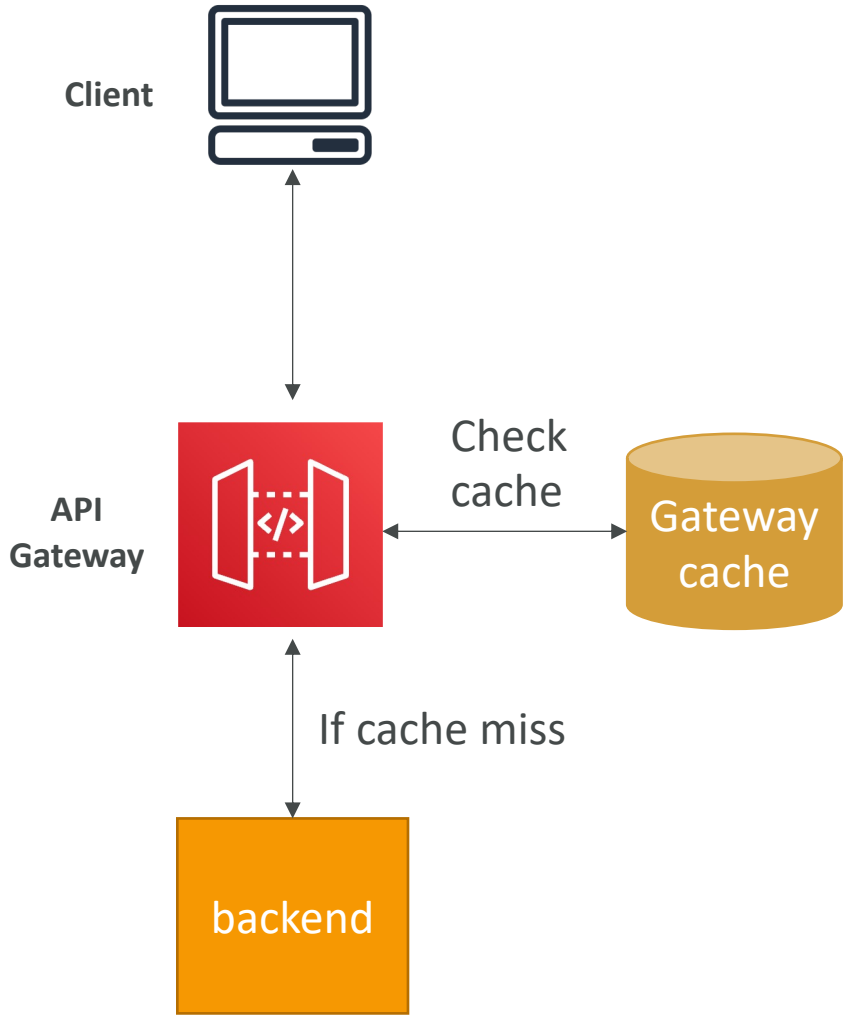
Enable **params-only** Validator on all API methods

```
{
    "openapi": "3.0.0",
    "info": {
        "title": "ReqValidation Sample",
        "version": "1.0.0"
    },
    "servers": [ ... ],
    "paths": {
        "/validation": {
            "post": {
                "x-amazon-apigateway-request-validator": "all"
            }
        }
    },
    ...
}
```

Enable **all** Validator on the **POST /validation** method (overrides the **params-only** validator inherited from the API)

Caching API responses

- Caching reduces the number of calls made to the backend
- Default TTL (time to live) is 300 seconds (min: 0s, max: 3600s)
- **Caches are defined per stage**
- Possible to override cache settings **per method**
- Cache encryption option
- Cache capacity between 0.5GB to 237GB
- Cache is expensive, makes sense in production, may not make sense in dev / test



API Gateway Cache Invalidation

- Able to flush the entire cache (invalidate it) immediately
- Clients can invalidate the cache with **header: Cache-Control: max-age=0** (with proper IAM authorization)
- If you don't impose an InvalidateCache policy (or choose the Require authorization check box in the console), any client can invalidate the API cache

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "execute-api:InvalidateCache"
            ],
            "Resource": [
                "arn:...:api-id/stage-name/GET/resource-path-specifier"
            ]
        }
    ]
}
```

API Gateway – Usage Plans & API Keys

- If you want to make an API available as an offering ($) to your customers
- Usage Plan:
  - who can access one or more deployed API stages and methods
  - how much and how fast they can access them
  - uses API keys to identify API clients and meter access
  - configure throttling limits and quota limits that are enforced on individual client
- API Keys:
  - alphanumeric string values to distribute to your customers
  - Ex: WBjHxNtoAb4WPKBC7cGm64CBibIb24b4jt8jJHo9
  - Can use with usage plans to control access
  - Throttling limits are applied to the API keys
  - Quotas limits is the overall number of maximum requests

API Gateway – Correct Order for API keys

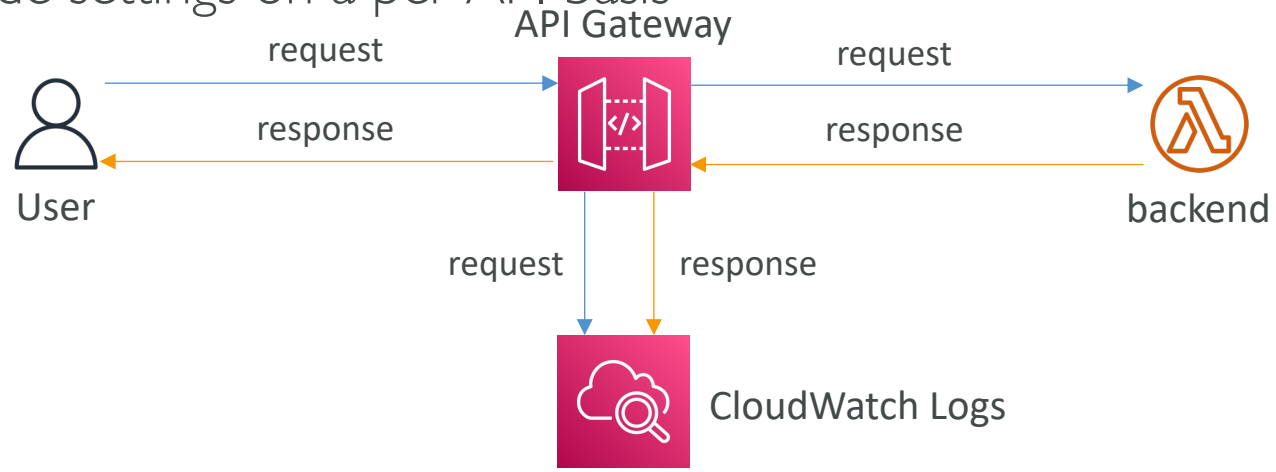
- To configure a usage plan

1. Create one or more APIs, configure the methods to require an API key, and deploy the APIs to stages.
2. Generate or import API keys to distribute to application developers (your customers) who will be using your API.
3. Create the usage plan with the desired throttle and quota limits.
4. Associate API stages and API keys with the usage plan.

- Callers of the API must supply an assigned API key in the x-api-key header in requests to the API.

API Gateway – Logging & Tracing

- CloudWatch Logs
  - Log contains information about request/response body
  - Enable CloudWatch logging at the Stage level (with Log Level - ERROR, DEBUG, INFO)
  - Can override settings on a per API basis



- X-Ray
  - Enable tracing to get extra information about requests in API Gateway
  - X-Ray API Gateway + AWS Lambda gives you the full picture

API Gateway – CloudWatch Metrics

- Metrics are by stage, Possibility to enable detailed metrics
- **CacheHitCount** & **CacheMissCount**: efficiency of the cache
- **Count**: The total number API requests in a given period.
- **IntegrationLatency**: The time between when API Gateway relays a request to the backend and when it receives a response from the backend.
- **Latency**: The time between when API Gateway receives a request from a client and when it returns a response to the client. The latency includes the integration latency and other API Gateway overhead.
- **4XXError** (client-side) & **5XXError** (server-side)

API Gateway Throttling

- Account Limit
  - API Gateway throttles requests at 10000 rps across all API
  - Soft limit that can be increased upon request
- In case of throttling => **429 Too Many Requests** (retriable error)
- Can set **Stage limit & Method limits** to improve performance
- Or you can define **Usage Plans** to throttle per customer

- Just like Lambda Concurrency, one API that is overloaded, if not limited, can cause the other APIs to be throttled

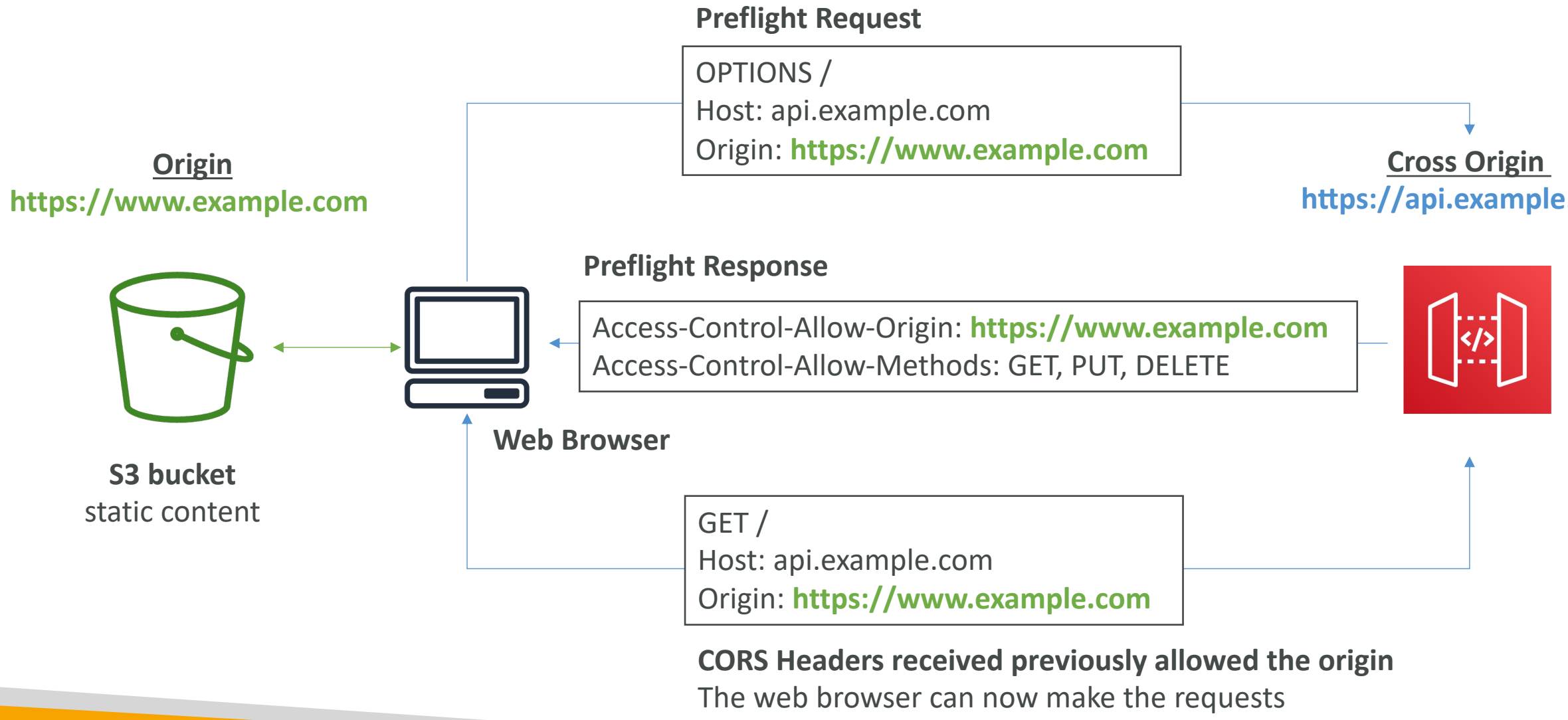
API Gateway - Errors

- 4xx means Client errors
  - 400: Bad Request
  - 403: Access Denied, WAF filtered
  - 429: Quota exceeded, Throttle
- 5xx means Server errors
  - 502: Bad Gateway Exception, usually for an incompatible output returned from a Lambda proxy integration backend and occasionally for out-of-order invocations due to heavy loads.
  - 503: Service Unavailable Exception
  - 504: Integration Failure – ex Endpoint Request Timed-out Exception
    API Gateway requests time out after 29 second maximum

AWS API Gateway - CORS

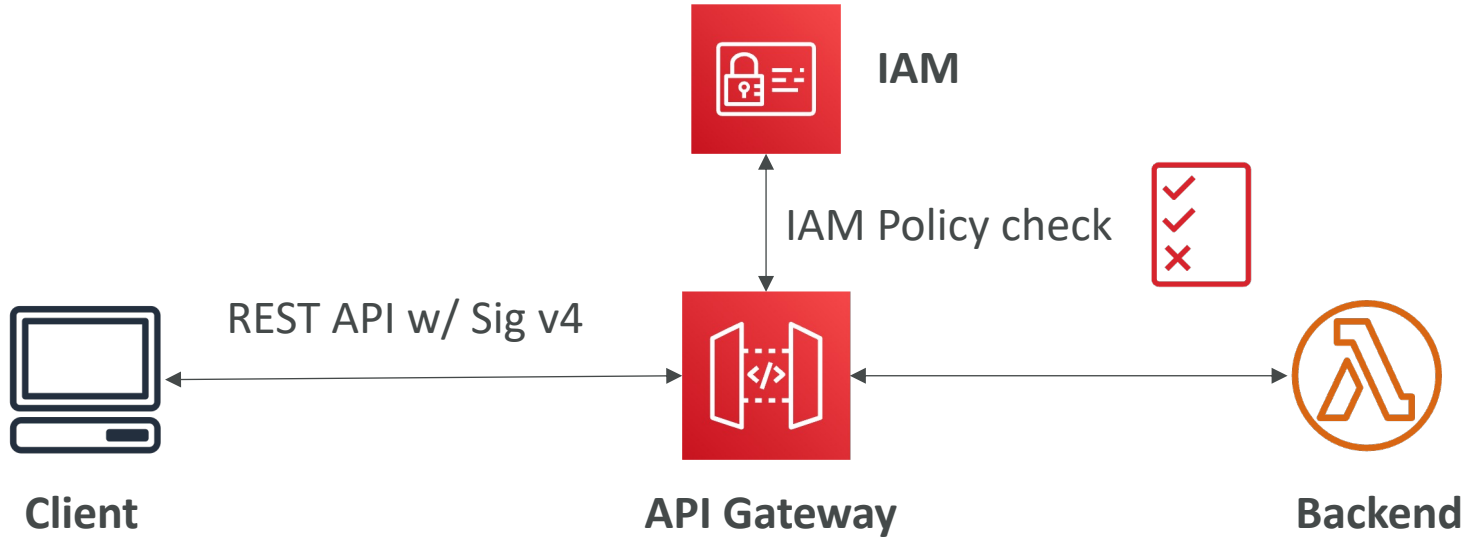
- CORS must be enabled when you receive API calls from another domain.
- The OPTIONS pre-flight request must contain the following headers:
  - Access-Control-Allow-Methods
  - Access-Control-Allow-Headers
  - Access-Control-Allow-Origin
- CORS can be enabled through the console

CORS – Enabled on the API Gateway



API Gateway – Security
IAM Permissions

- Create an IAM policy authorization and attach to User / Role
- Authentication = IAM | Authorization = IAM Policy
- Good to provide access within AWS (EC2, Lambda, IAM users…)
- Leverages "Sig v4" capability where IAM credential are in headers



API Gateway – Resource Policies

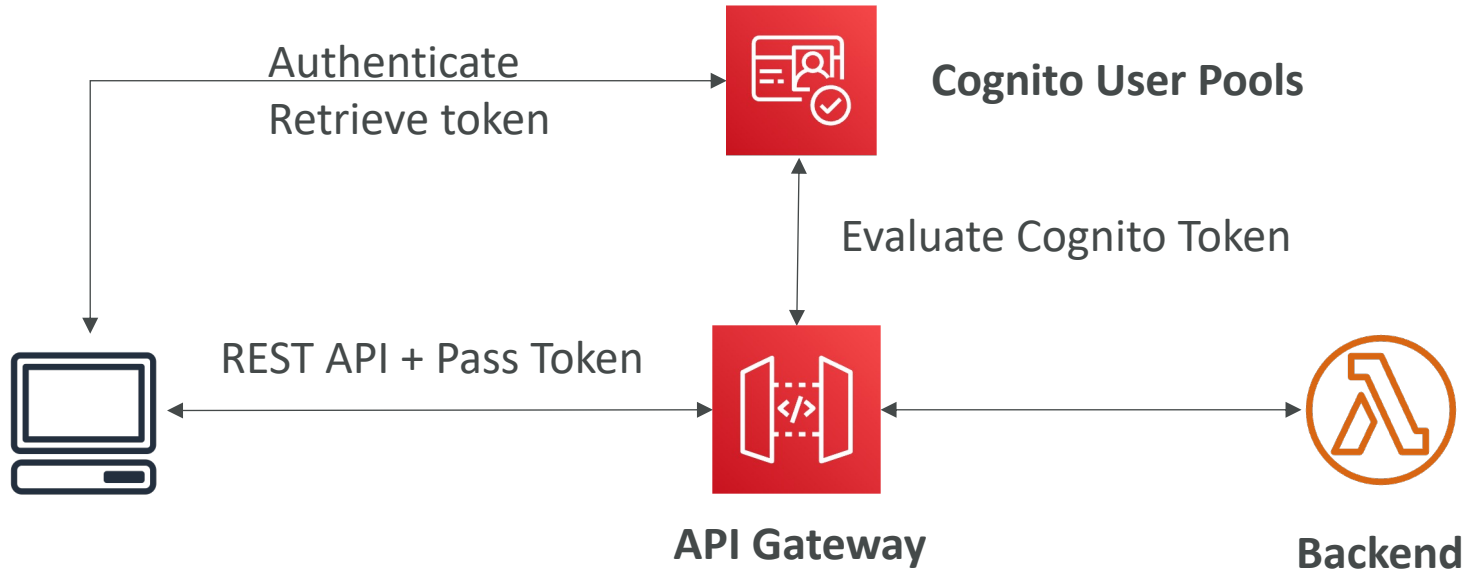
- **Resource policies** (similar to Lambda Resource Policy)

- **Allow for Cross Account Access (combined with IAM Security)**
- Allow for a specific source IP address
- Allow for a VPC Endpoint

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::account-id-2:user/Alice",
                    "account-id-2"
                ]
            },
            "Action": "execute-api:Invoke",
            "Resource": [
                "arn:aws:execute-api:region:account-id-1:api-id/stage/GET/pets"
            ]
        }
    ]
}
```

API Gateway – Security
Cognito User Pools

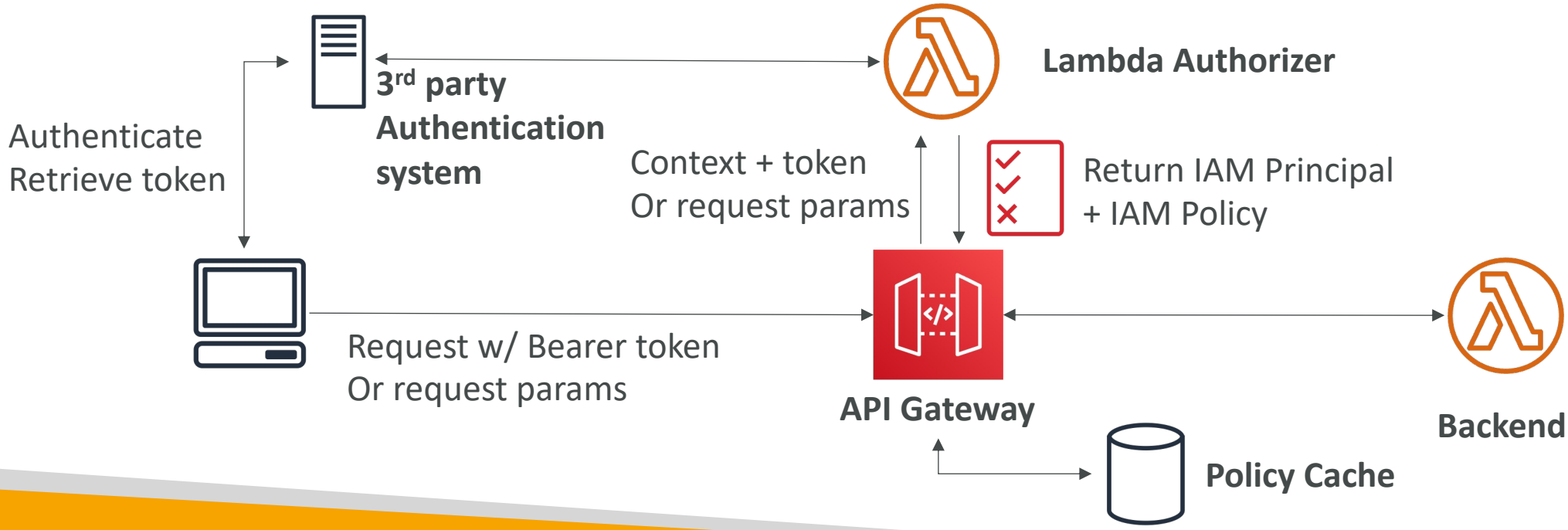
- Cognito fully manages user lifecycle, token expires automatically
- API gateway verifies identity automatically from AWS Cognito
- No custom implementation required
- Authentication = Cognito User Pools | Authorization = API Gateway Methods



API Gateway – Security

Lambda Authorizer (formerly Custom Authorizers)

- Token-based authorizer (bearer token) – ex JWT (JSON Web Token) or Oauth
- A request parameter-based Lambda authorizer (headers, query string, stage var)
- Lambda must return an IAM policy for the user, result policy is cached
- Authentication = External | Authorization = Lambda function



API Gateway – Security – Summary

- IAM:
  - Great for users / roles already within your AWS account, + resource policy for cross account
  - Handle authentication + authorization
  - Leverages Signature v4
- Custom Authorizer:
  - Great for 3rd party tokens
  - Very flexible in terms of what IAM policy is returned
  - Handle Authentication verification + Authorization in the Lambda function
  - Pay per Lambda invocation, results are cached
- Cognito User Pool:
  - You manage your own user pool (can be backed by Facebook, Google login etc…)
  - No need to write any custom code
  - Must implement authorization in the backend

API Gateway – HTTP API vs REST API

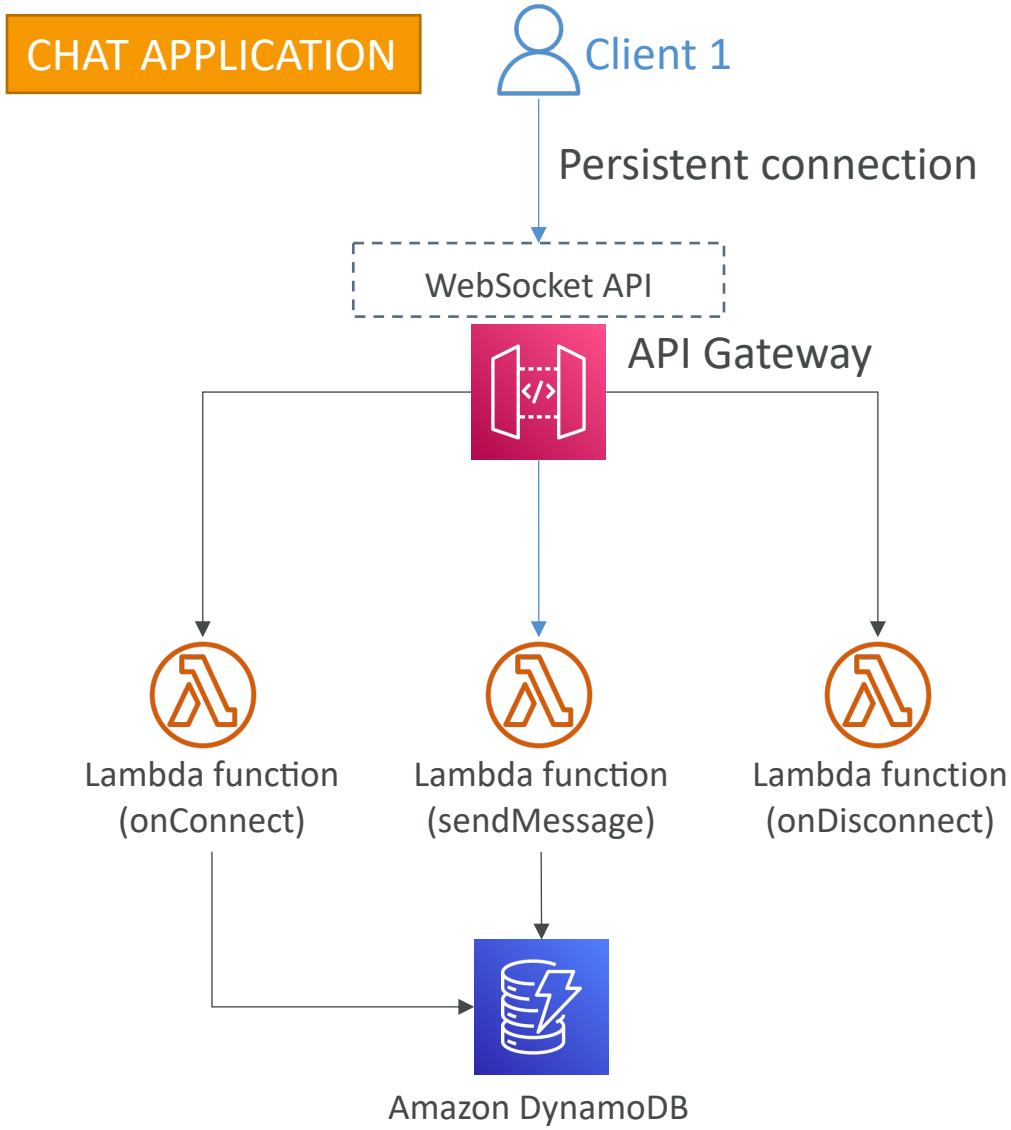
- HTTP APIs
  - low-latency, cost-effective AWS Lambda proxy, HTTP proxy APIs and private integration (no data mapping)
  - support OIDC and OAuth 2.0 authorization, and built-in support for CORS
  - No usage plans and API keys
- REST APIs
  - All features (except Native OpenID Connect / OAuth 2.0)

Authorizers	HTTP API	REST API
AWS Lambda	✓	✓
IAM	✓	✓
Resource Policies		✓
Amazon Cognito	✓ *	✓
Native OpenID Connect / OAuth 2.0 / JWT	✓	

Full list here: https://docs.aws.amazon.com/apigateway/latest/developerguide/http-api-vs-rest.html

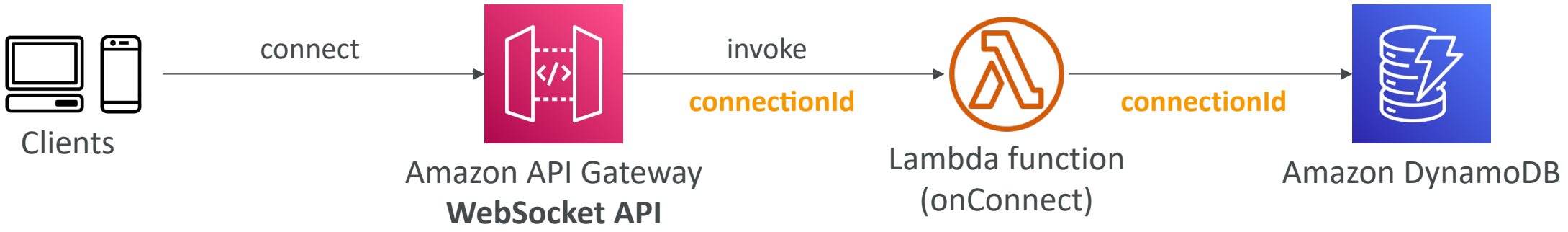
API Gateway – WebSocket API – Overview

- What's WebSocket?
  - Two-way interactive communication between a user's browser and a server
  - Server can push information to the client
  - This enables **stateful** application use cases
- WebSocket APIs are often used in **real-time applications** such as chat applications, collaboration platforms, multiplayer games, and financial trading platforms.
- Works with AWS Services (Lambda, DynamoDB) or HTTP endpoints



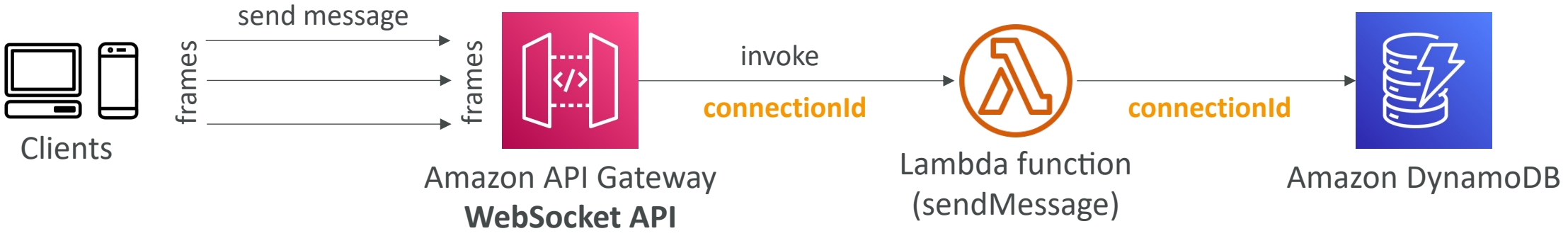
Connecting to the API

WebSocket URL
wss://[some-uniqueid].execute-api.[region].amazonaws.com/[stage-name]



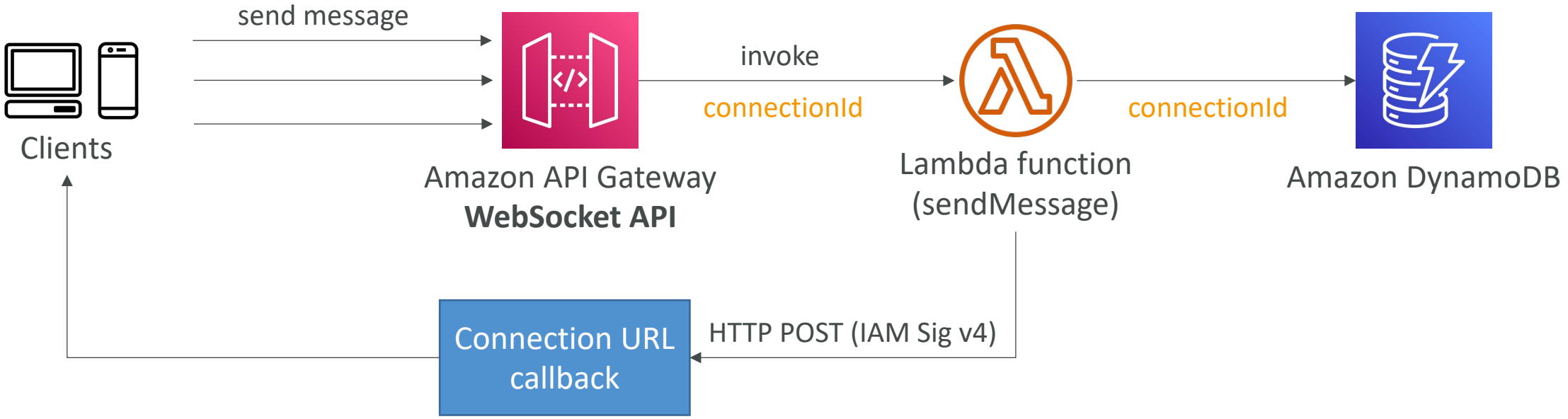
Client to Server Messaging
ConnectionID is re-used

WebSocket URL
wss://abcdef.execute-api.us-west-1.amazonaws.com/dev



Server to Client Messaging

WebSocket URL
wss://abcdef.execute-api.us-west-1.amazonaws.com/dev



Connection URL
wss://abcdef.execute-api.us-west-1.amazonaws.com/dev/@connections/connectionId

Connection URL Operations

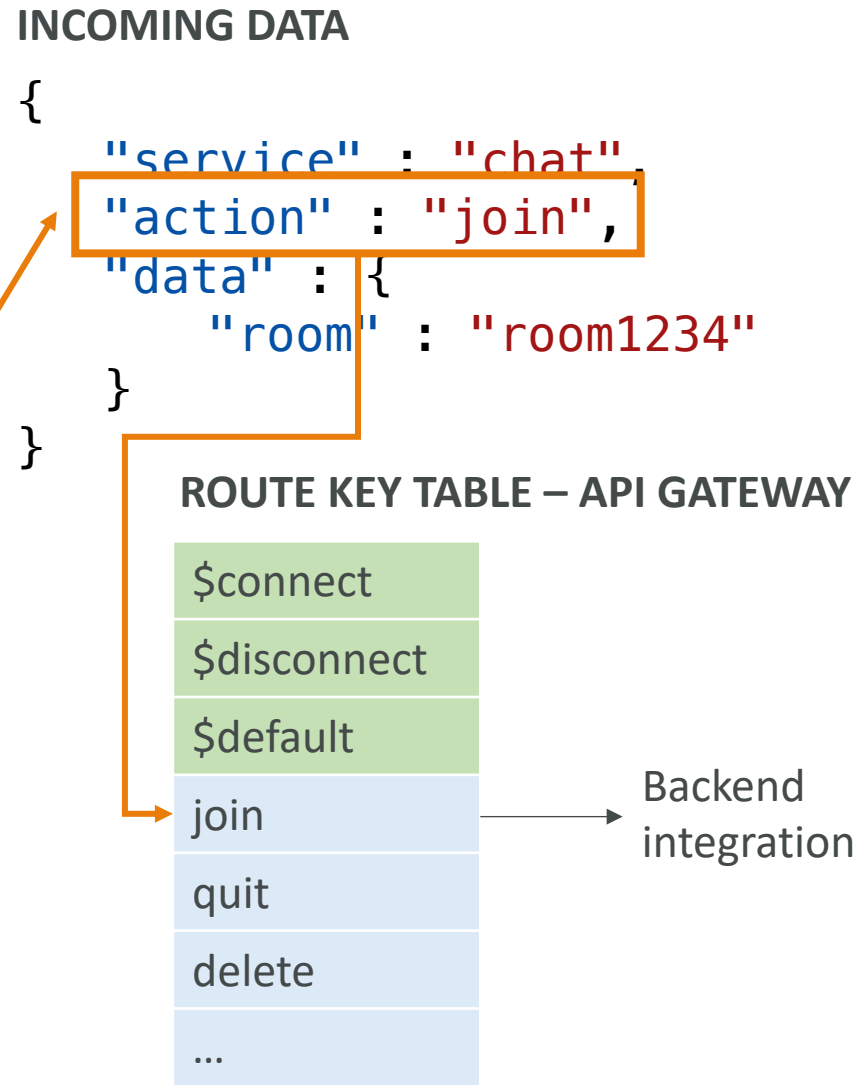
Connection URL

wss://abcdef.execute-api.us-west-1.amazonaws.com/dev/@connections/connectionId

Operation	Action
POST	Sends a message from the Server to the connected WS Client
GET	Gets the latest connection status of the connected WS Client
DELETE	Disconnect the connected Client from the WS connection

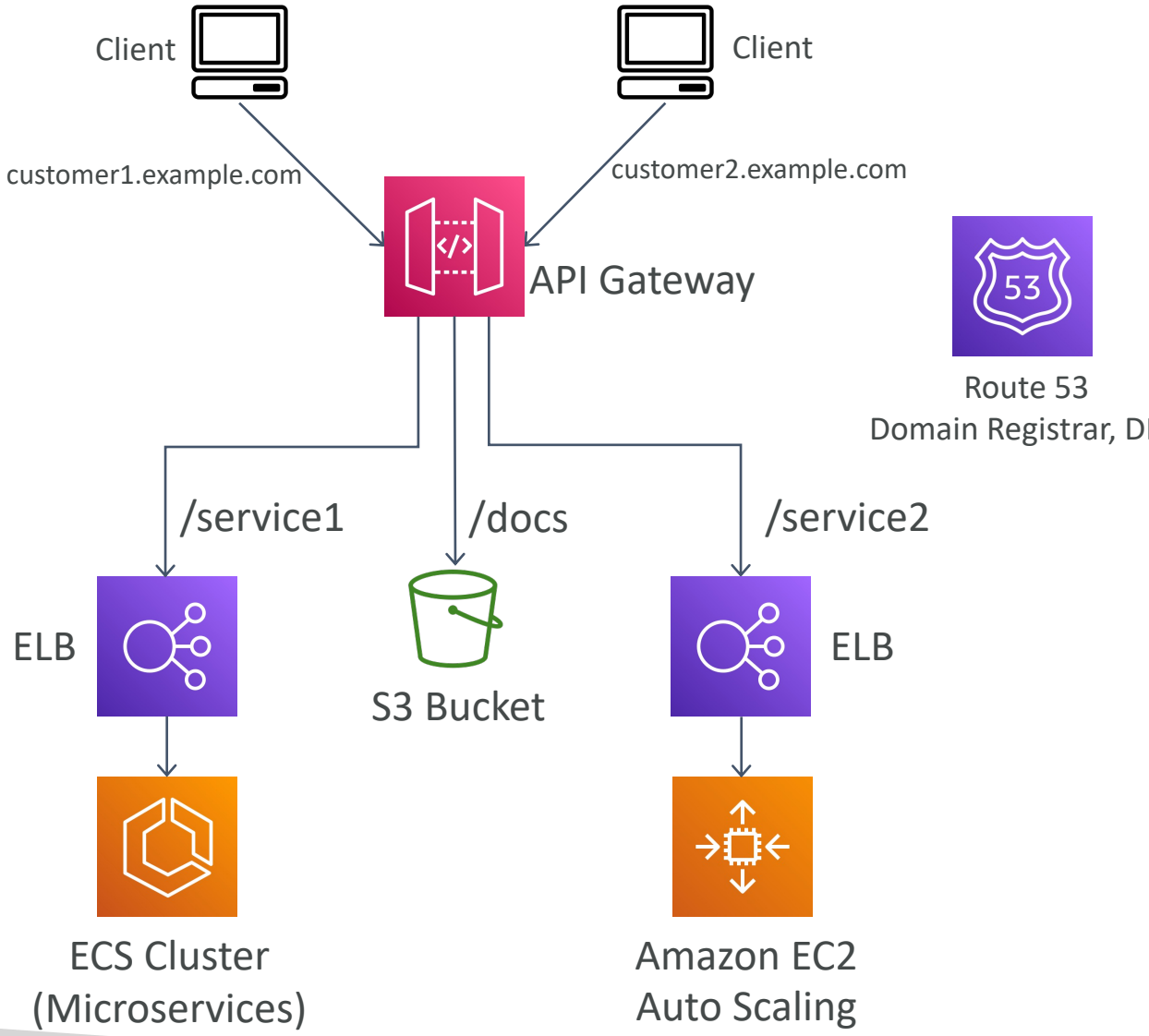
API Gateway – WebSocket API – Routing

- Incoming JSON messages are routed to different backend
- If no routes => sent to $default
- You request a *route selection expression* to select the field on JSON to route from
- Sample expression: **$request.body.action**
- The result is evaluated against the route keys available in your API Gateway
- The route is then connected to the backend you've setup through API Gateway



API Gateway - Architecture

- Create a single interface for all the microservices in your company
- Use API endpoints with various resources
- Apply a simple domain name and SSL certificates
- Can apply forwarding and transformation rules at the API Gateway level



AWS CICD

CodeCommit, CodePipeline, CodeBuild, CodeDeploy, …

CICD – Introduction

- We have learned how to:
  - Create AWS resources, manually (fundamentals)
  - Interact with AWS programmatically (AWS CLI)
  - Deploy code to AWS using Elastic Beanstalk
- All these manual steps make it very likely for us to do mistakes!

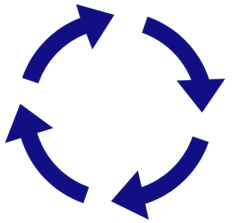
- We would like our code "in a repository" and have it deployed onto AWS
  - Automatically
  - The right way
  - Making sure it's tested before being deployed
  - With possibility to go into different stages (dev, test, staging, prod)
  - With manual approval where needed
- To be a proper AWS developer… we need to learn AWS CICD

CICD – Introduction

- This section is all about automating the deployment we've done so far while adding increased safety

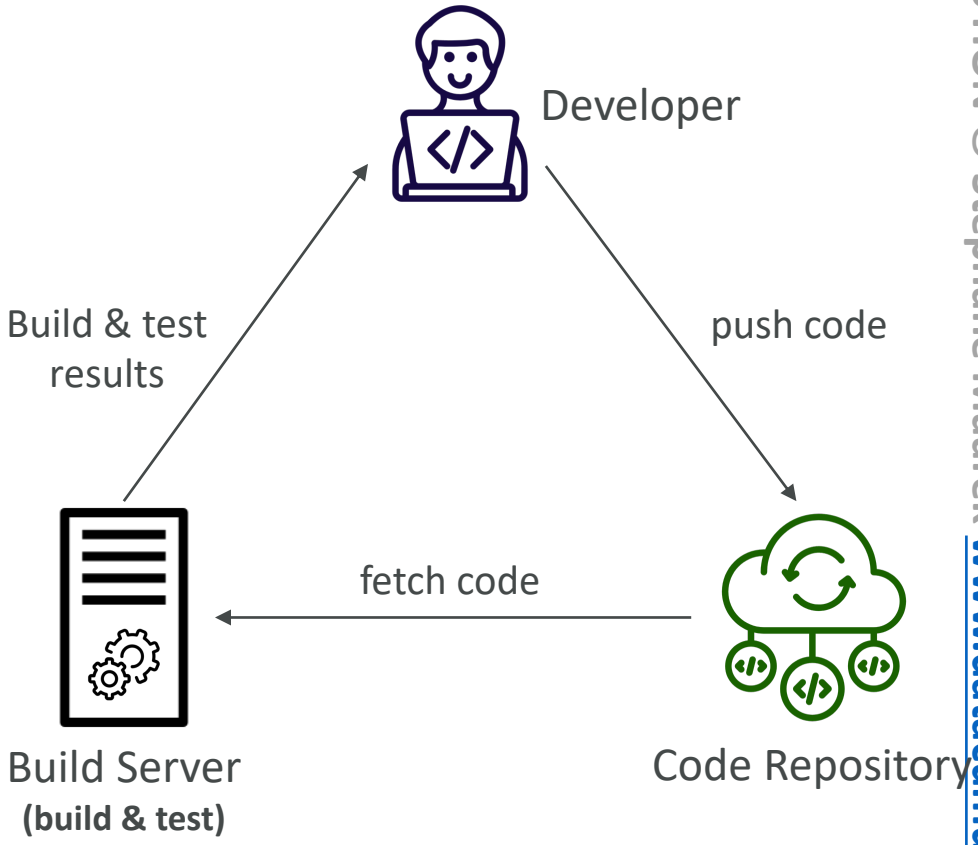
- We'll learn about:
  - **AWS CodeCommit** – storing our code
  - **AWS CodePipeline** – automating our pipeline from code to Elastic Beanstalk
  - **AWS CodeBuild** – building and testing our code
  - **AWS CodeDeploy** – deploying the code to EC2 instances (not Elastic Beanstalk)
  - **AWS CodeStar** – manage software development activities in one place
  - **AWS CodeArtifact** – store, publish, and share software packages
  - **AWS CodeGuru** – automated code reviews using Machine Learning

Continuous Integration (CI)



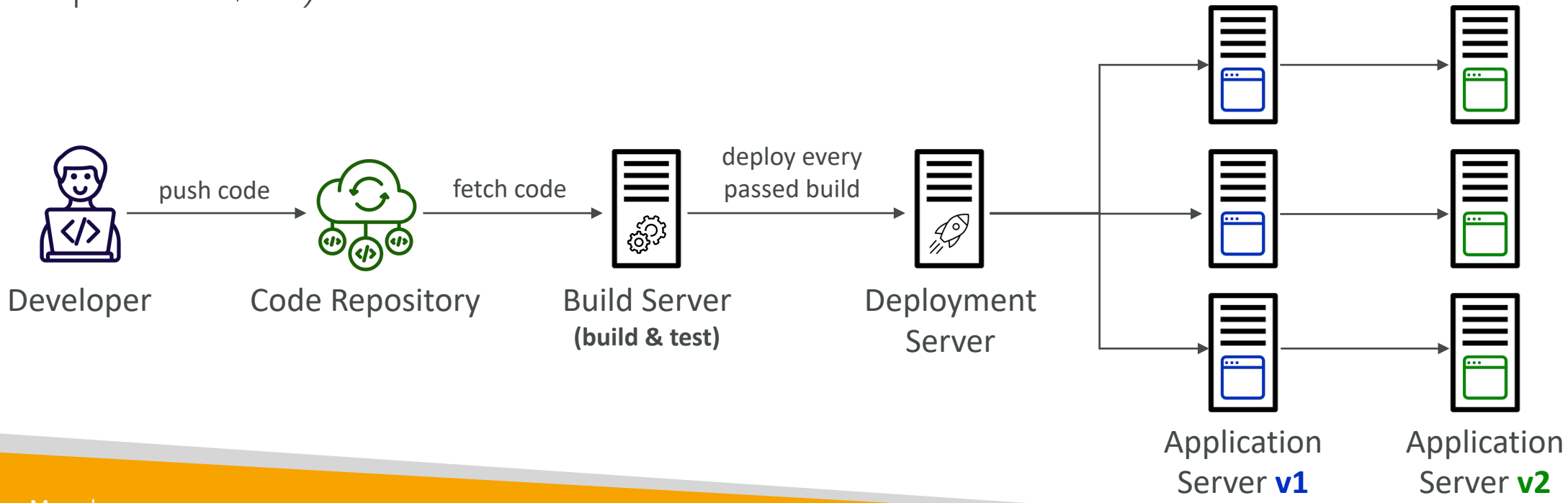
- Developers push the code to a code repository often (e.g., GitHub, CodeCommit, Bitbucket…)
- A testing / build server checks the code as soon as it's pushed (CodeBuild, Jenkins CI, …)
- The developer gets feedback about the tests and checks that have passed / failed

- Find bugs early, then fix bugs
- Deliver faster as the code is tested
- Deploy often
- Happier developers, as they're unblocked

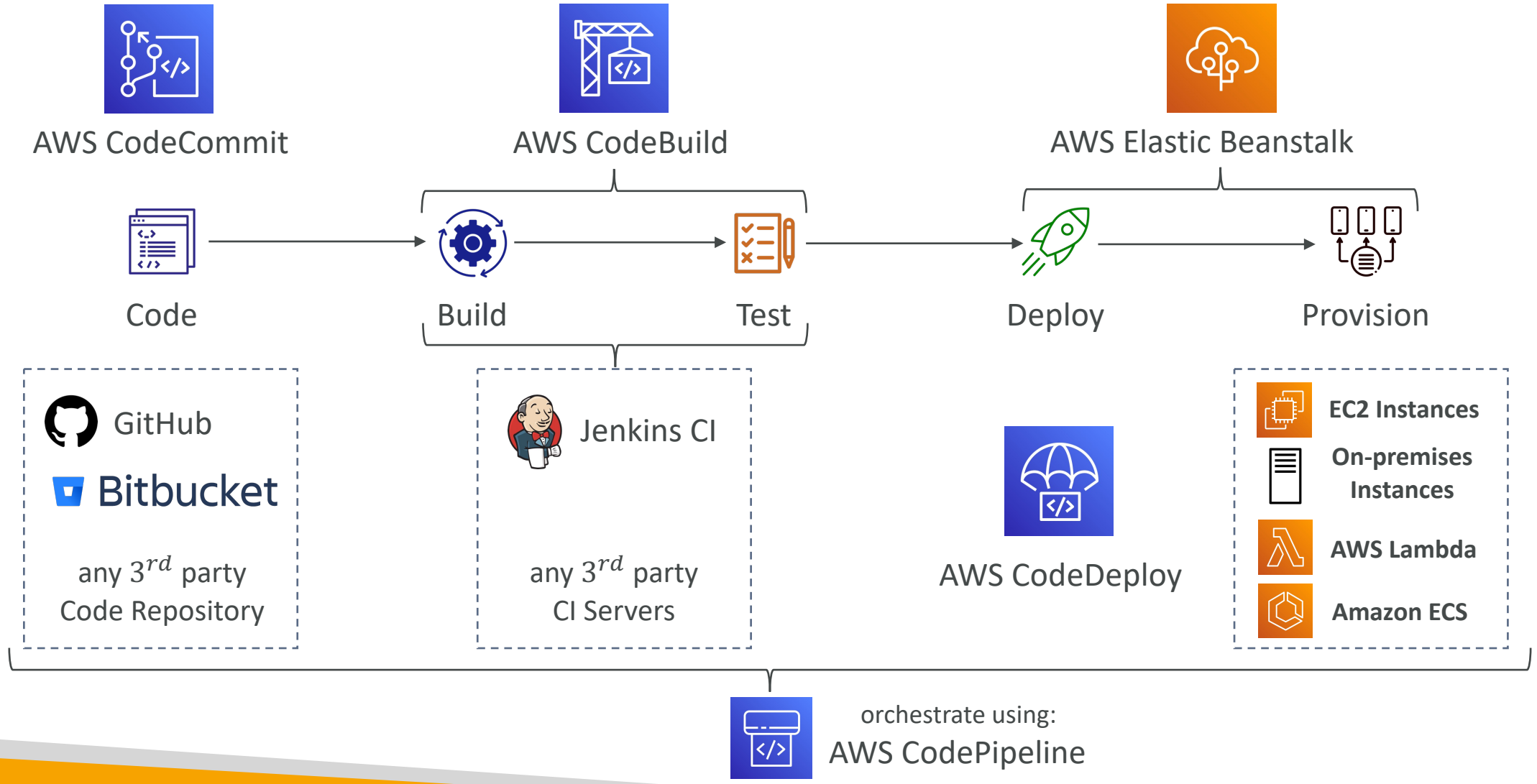


Continuous Delivery (CD)

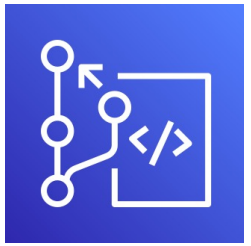
- Ensures that the software can be released reliably whenever needed
- Ensures deployments happen often and are quick
- Shift away from "one release every 3 months" to "5 releases a day"
- That usually means automated deployment (e.g., CodeDeploy, Jenkins CD, Spinnaker, …)



Technology Stack for CICD



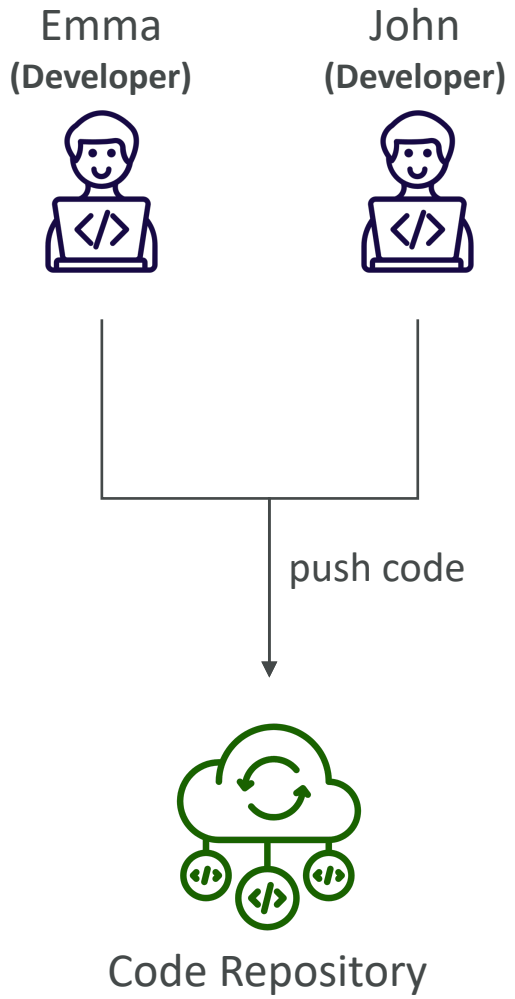
AWS CodeCommit



- **Version control** is the ability to understand the various changes that happened to the code over time (and possibly roll back)
- All these are enabled by using a version control system such as **Git**
- A Git repository can be synchronized on your computer, but it usually is uploaded on a central online repository
- Benefits are:
  - Collaborate with other developers
  - Make sure the code is backed-up somewhere
  - Make sure it's fully viewable and auditable

AWS CodeCommit

- Git repositories can be expensive
- The industry includes GitHub, GitLab, Bitbucket, …
- And **AWS CodeCommit:**
  - Private Git repositories
  - No size limit on repositories (scale seamlessly)
  - Fully managed, highly available
  - Code only in AWS Cloud account => increased security and compliance
  - Security (encrypted, access control, …)
  - Integrated with Jenkins, AWS CodeBuild, and other CI tools



CodeCommit – Security

- Interactions are done using Git (standard)
- Authentication
  - SSH Keys – AWS Users can configure SSH keys in their IAM Console
  - HTTPS – with AWS CLI Credential helper or Git Credentials for IAM user
- Authorization
  - IAM policies to manage users/roles permissions to repositories
- Encryption
  - Repositories are automatically encrypted at rest using AWS KMS
  - Encrypted in transit (can only use HTTPS or SSH – both secure)
- Cross-account Access
  - Do NOT share your SSH keys or your AWS credentials
  - Use an IAM Role in your AWS account and use AWS STS (**AssumeRole** API)

CodeCommit vs. GitHub

	CodeCommit	GitHub
Support Code Review (Pull Requests)	✔	✔
Integration with AWS CodeBuild	✔	✔
Authentication (SSH & HTTPS)	✔	✔
Security	IAM Users & Roles	GitHub Users
Hosting	Managed & hosted by AWS	- Hosted by GitHub - GitHub Enterprise: self hosted on your servers
UI	Minimal	Fully Featured

CodeCommit – Important – Deprecation

- On July 25th 2024, AWS abruptly discontinued CodeCommit
- New customers cannot use the service
- AWS recommends to migrate to an external Git solution

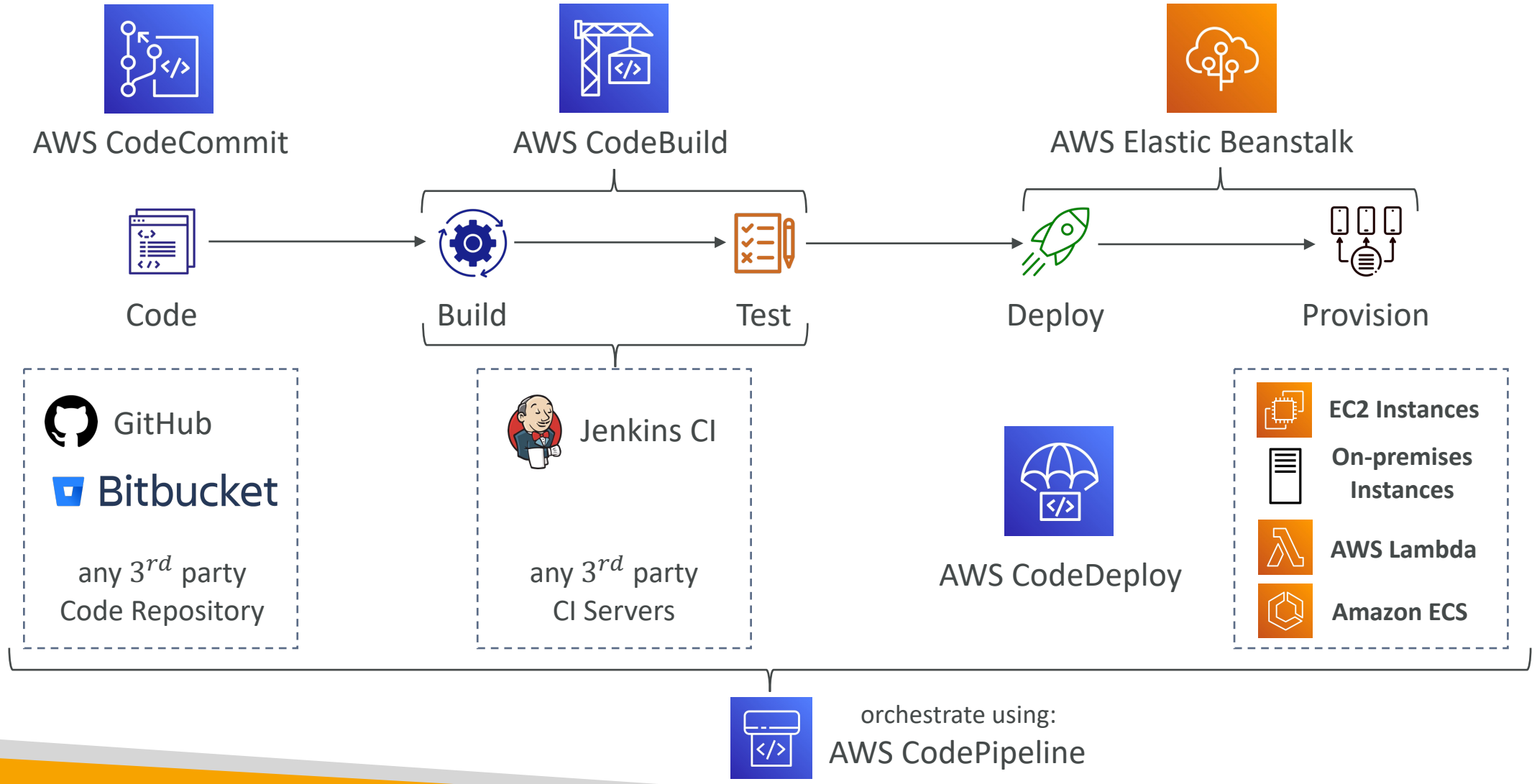
3rd party

- For this course:
  - CodeCommit might still appear at the exam (for now)
  - Anytime I use CodeCommit, please use GitHub instead (we set it up once together)
  - Every time I mention CodeCommit, assume there's a GitHub integration

AWS CodePipeline

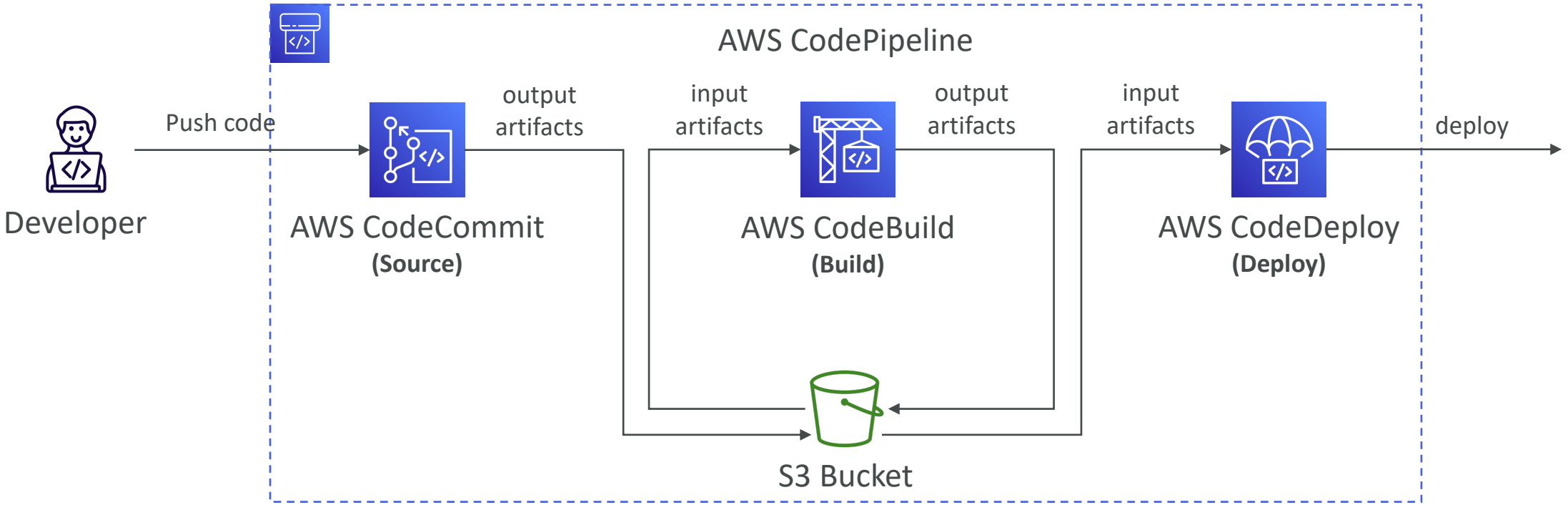
- Visual Workflow to orchestrate your CICD
- **Source** – CodeCommit, ECR, S3, Bitbucket, GitHub
- **Build** – CodeBuild, Jenkins, CloudBees, TeamCity
- **Test** – CodeBuild, AWS Device Farm, 3rd party tools, …
- **Deploy** – CodeDeploy, Elastic Beanstalk, CloudFormation, ECS, S3, …
- **Invoke** – Lambda, Step Functions
- Consists of stages:
  - Each stage can have sequential actions and/or parallel actions
  - Example: Build ➔ Test ➔ Deploy ➔ Load Testing ➔ …
  - Manual approval can be defined at any stage

Technology Stack for CICD



CodePipeline – Artifacts

- Each pipeline stage can create **artifacts**
- Artifacts stored in an S3 bucket and passed on to the next stage



CodePipeline – Troubleshooting

- For CodePipeline Pipeline/Action/Stage Execution State Changes
- Use **CloudWatch Events (Amazon EventBridge).** Example:
  - You can create events for failed pipelines
  - You can create events for cancelled stages

- If CodePipeline fails a stage, your pipeline stops, and you can get information in the console
- If pipeline can't perform an action, make sure the "IAM Service Role" attached does have enough IAM permissions (IAM Policy)
- AWS CloudTrail can be used to audit AWS API calls

AWS CodeBuild

- A fully managed continuous integration (CI) service
- Continuous scaling (no servers to manage or provision – no build queue)
- Compile source code, run tests, produce software packages, …
- Alternative to other build tools (e.g., Jenkins)
- Charged per minute for compute resources (time it takes to complete the builds)
- Leverages Docker under the hood for reproducible builds
- Use prepackaged Docker images or create your own custom Docker image
- Security:
  - Integration with KMS for encryption of build artifacts
  - IAM for CodeBuild permissions, and VPC for network security
  - AWS CloudTrail for API calls logging

AWS CodeBuild

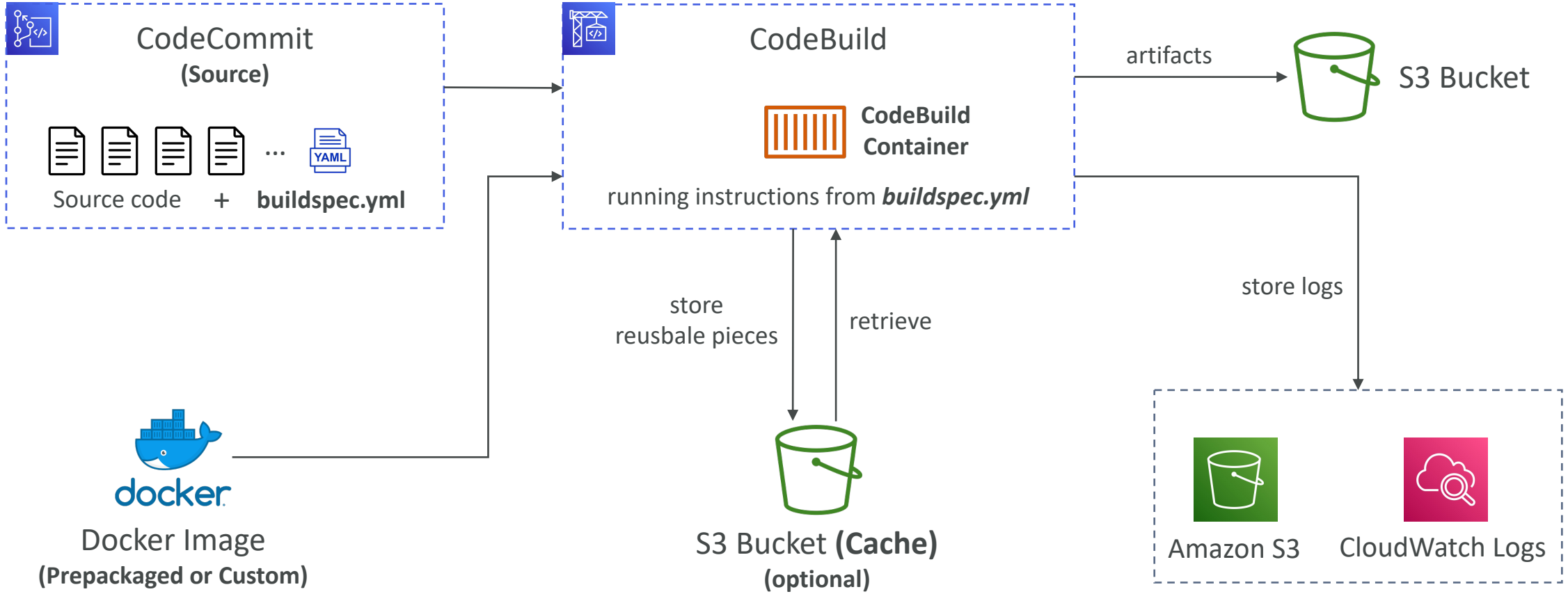
- **Source** – CodeCommit, S3, Bitbucket, GitHub
- **Build instructions**: Code file **buildspec.yml** or insert manually in Console
- **Output logs** can be stored in Amazon S3 & CloudWatch Logs
- Use CloudWatch Metrics to monitor build statistics
- Use EventBridge to detect failed builds and trigger notifications
- Use CloudWatch Alarms to notify if you need "thresholds" for failures

- **Build Projects can be defined within CodePipeline or CodeBuild**

CodeBuild – Supported Environments

- Java
- Ruby
- Python
- Go
- Node.js
- Android
- .NET Core
- PHP
- Docker – extend any environment you like

CodeBuild – How it Works



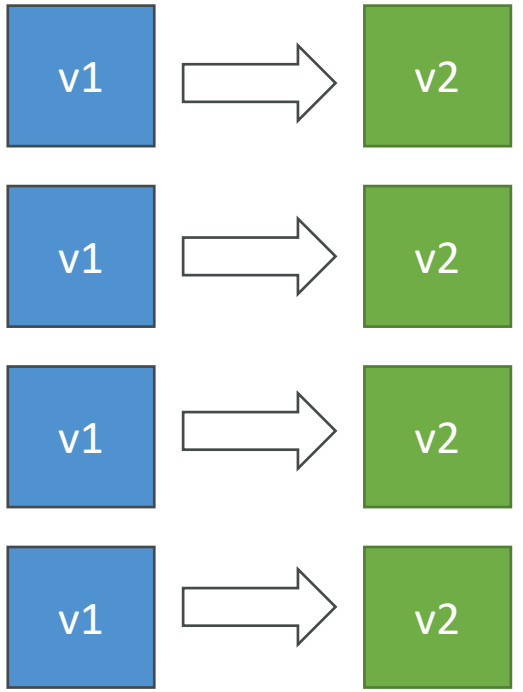
CodeBuild – buildspec.yml

- **buildspec.yml** file must be at the **root** of your code
- **env** – define environment variables
  - **variables** – plaintext variables
  - **parameter-store** – variables stored in SSM Parameter Store
  - **secrets-manager** – variables stored in AWS Secrets Manager
- **phases** – specify commands to run:
  - **install** – install dependencies you may need for your build
  - **pre_build** – final commands to execute before build
  - **Build** – **actual build commands**
  - **post_build** – finishing touches (e.g., zip output)
- **artifacts** – what to upload to S3 (encrypted with KMS)
- **cache** – files to cache (usually dependencies) to S3 for future build speedup

```yaml
version: 0.2

env:
  variables:
    JAVA_HOME: "/usr/lib/jvm/java-8-openjdk-amd64"
  parameter-store:
    LOGIN_PASSWORD: /CodeBuild/dockerLoginPassword

phases:
  install:
    commands:
      - echo "Entered the install phase..."
      - apt-get update -y
      - apt-get install -y maven
  pre_build:
    commands:
      - echo "Entered the pre_build phase..."
      - docker login -u User -p $LOGIN_PASSWORD
  build:
    commands:
      - echo "Entered the build phase..."
      - echo "Build started on `date`"
      - mvn install
  post_build:
    commands:
      - echo "Entered the post_build phase..."
      - echo "Build completed on `date`"

artifacts:
  files:
    - target/messageUtil-1.0.jar

cache:
  paths:
    - "/root/.m2/**/*"
```

AWS CodeDeploy

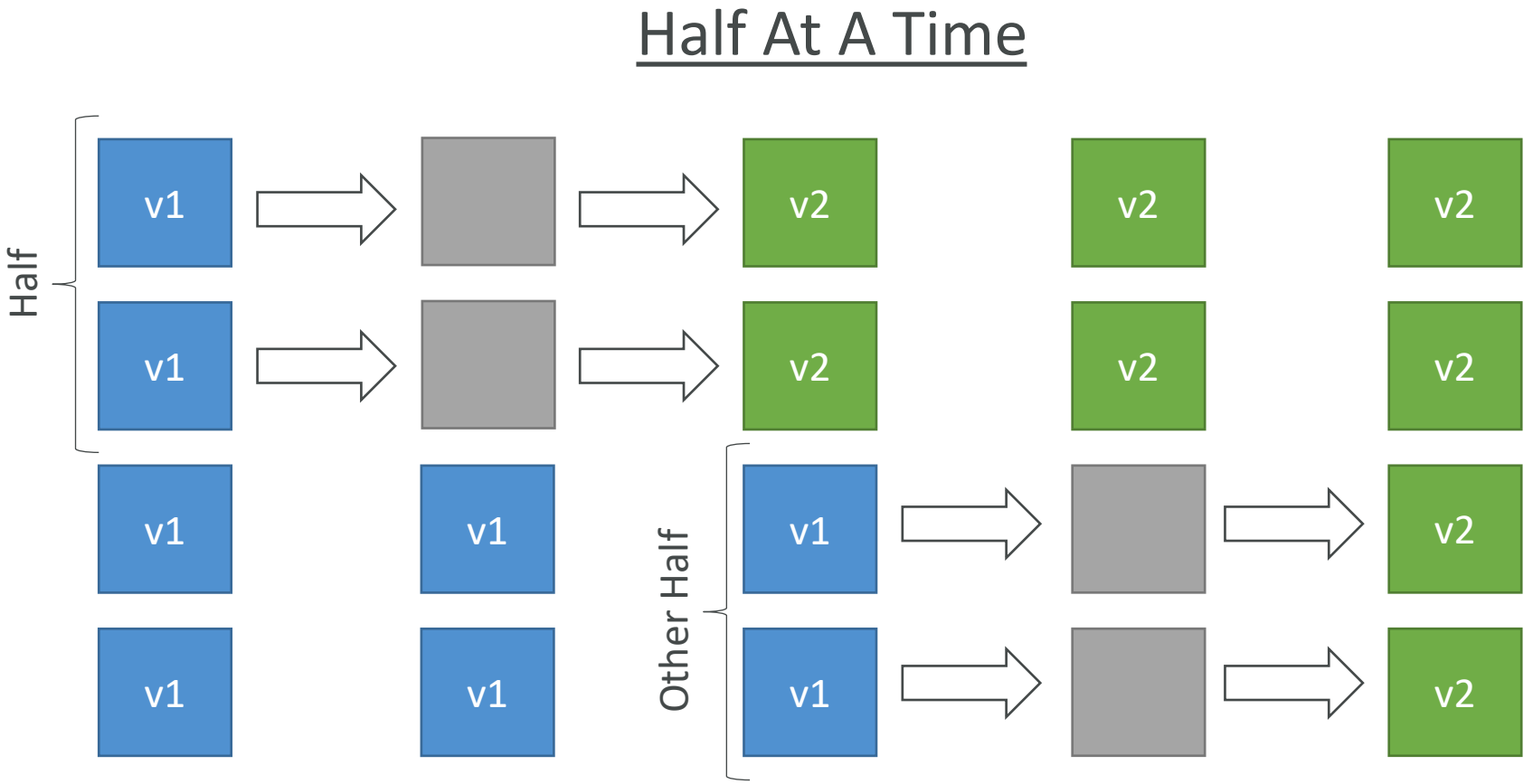
- Deployment service that automates application deployment
- Deploy new applications versions to EC2 Instances, On-premises servers, Lambda functions, ECS Services
- Automated Rollback capability in case of failed deployments, or trigger CloudWatch Alarm
- Gradual deployment control
- A file named appspec.yml defines how the deployment happens



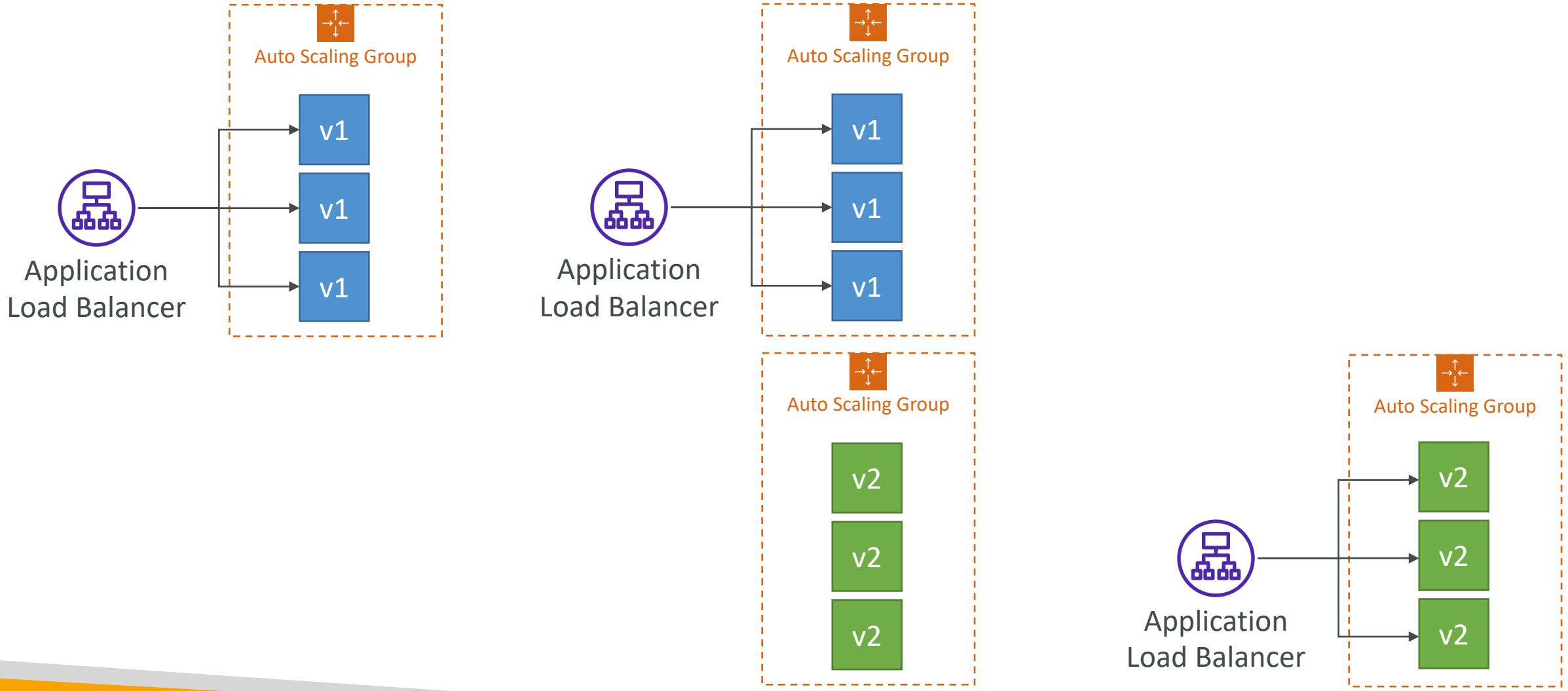
CodeDeploy – EC2/On-premises Platform

- Can deploy to EC2 Instances & on-premises servers
- Perform in-place deployments or blue/green deployments
- Must run the **CodeDeploy Agent** on the target instances
- Define deployment speed
  - AllAtOnce: most downtime
  - HalfAtATime: reduced capacity by 50%
  - OneAtATime: slowest, lowest availability impact
  - Custom: define your %

CodeDeploy – In-Place Deployment

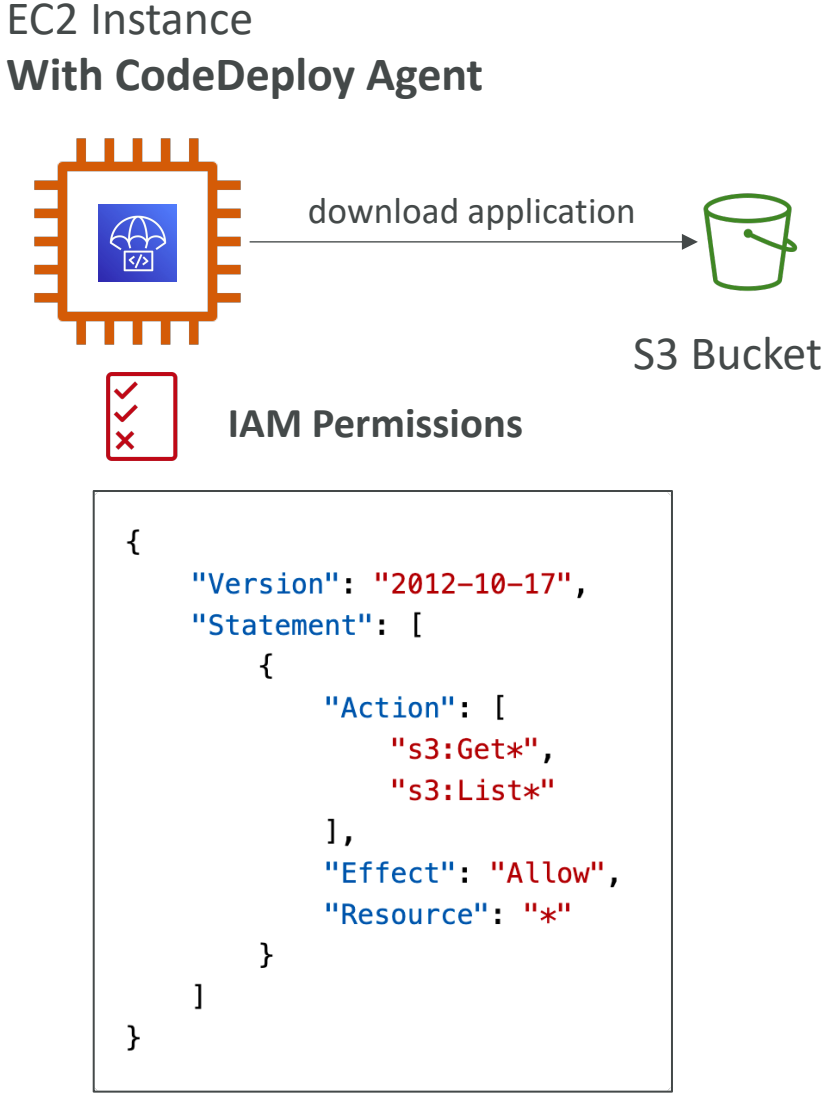


CodeDeploy – Blue-Green Deployment



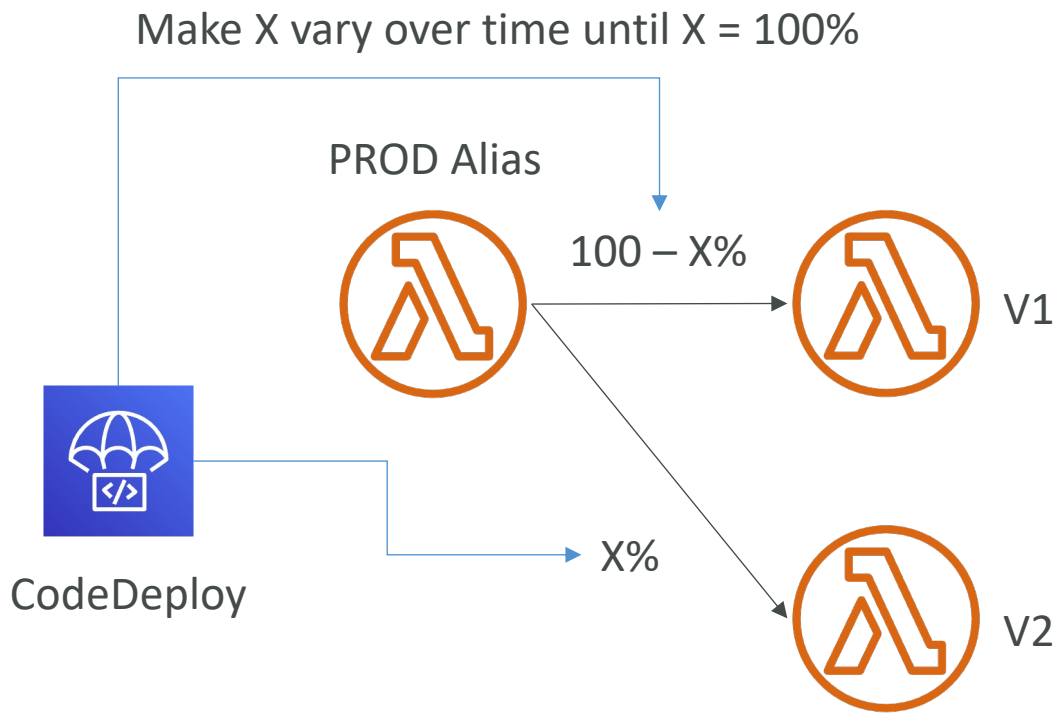
CodeDeploy Agent

- The CodeDeploy Agent must be running on the EC2 instances as a pre-requisites
- It can be installed and updated automatically if you're using Systems Manager
- The EC2 Instances must have sufficient permissions to access Amazon S3 to get deployment bundles



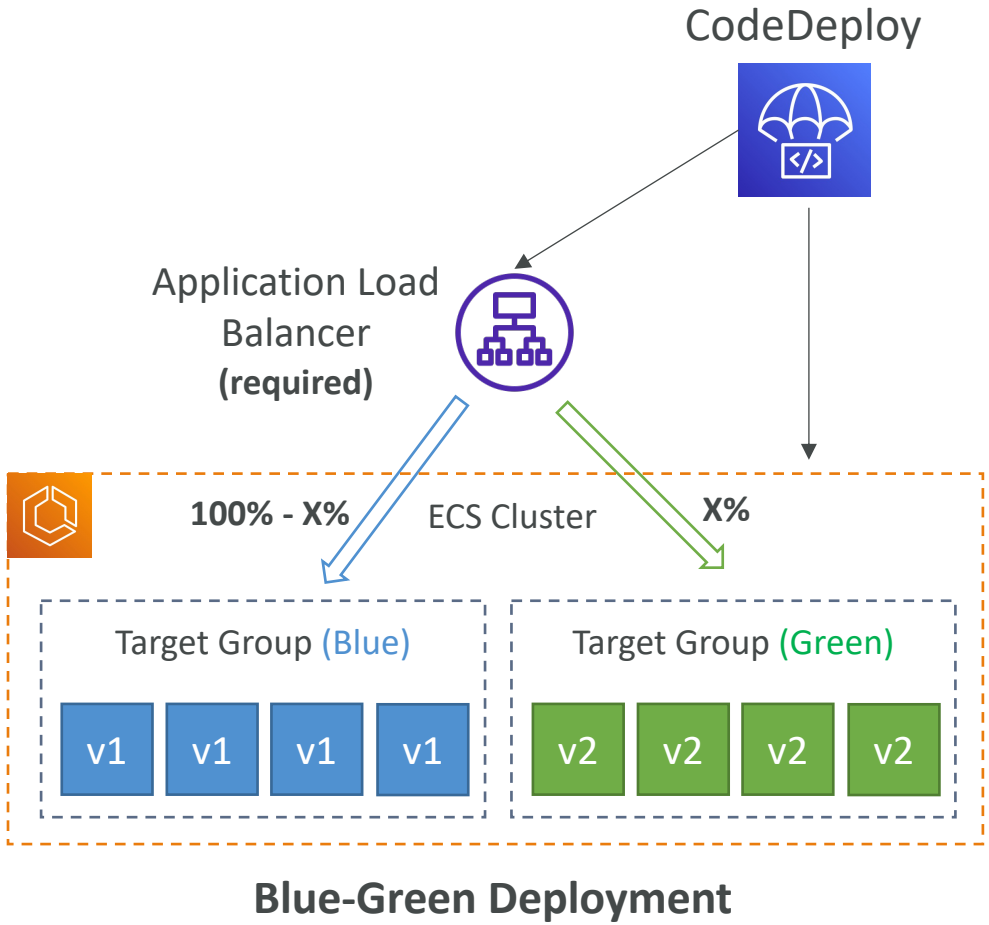
CodeDeploy – Lambda Platform

- **CodeDeploy** can help you automate traffic shift for Lambda aliases
- Feature is integrated within the SAM framework
- **Linear:** grow traffic every N minutes until 100%
  - LambdaLinear10PercentEvery3Minutes
  - LambdaLinear10PercentEvery10Minutes
- **Canary:** try X percent then 100%
  - LambdaCanary10Percent5Minutes
  - LambdaCanary10Percent30Minutes
- **AllAtOnce:** immediate



CodeDeploy – ECS Platform

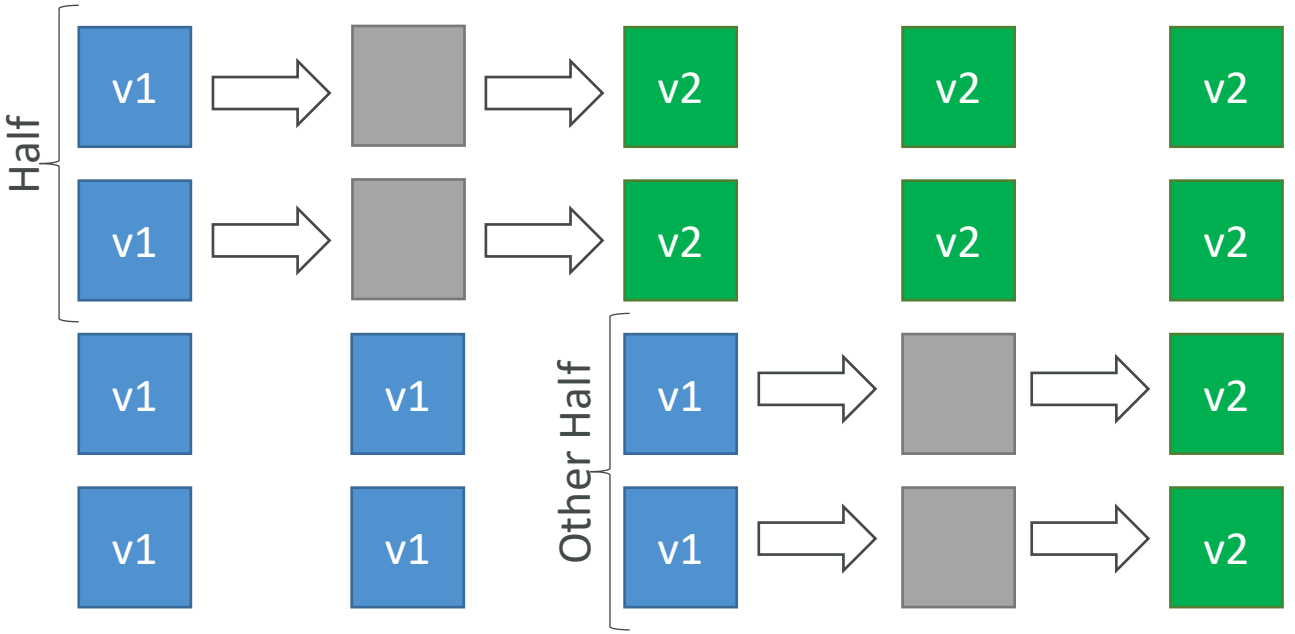
- **CodeDeploy** can help you automate the deployment of a new ECS Task Definition
- Only Blue/Green Deployments
- **Linear:** grow traffic every N minutes until 100%
  - ECSLinear10PercentEvery3Minutes
  - ECSLinear10PercentEvery10Minutes
- **Canary:** try X percent then 100%
  - ECSCanary10Percent5Minutes
  - ECSCanary10Percent30Minutes
- **AllAtOnce:** immediate



Blue-Green Deployment

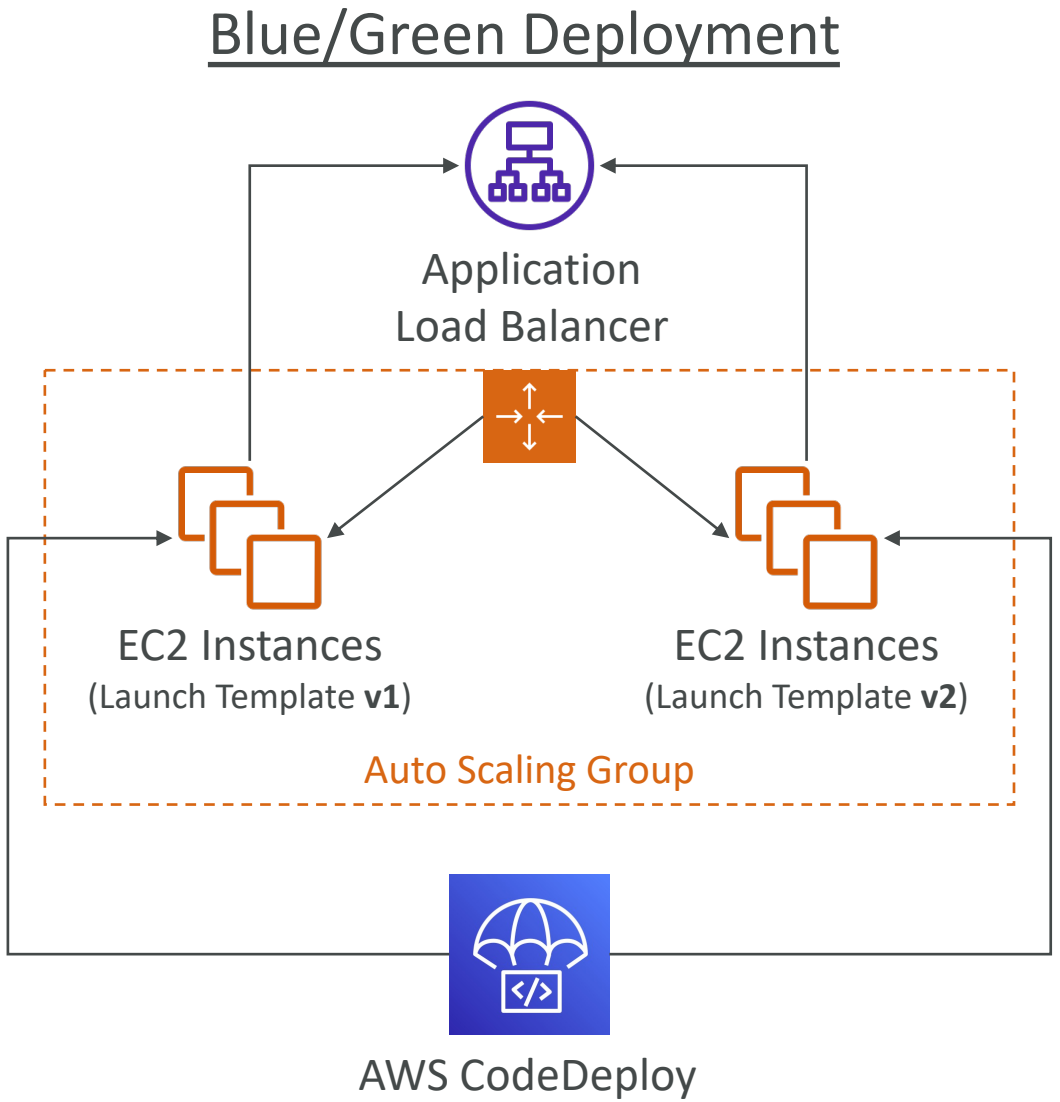
CodeDeploy – Deployment to EC2

- Define how to deploy the application using **appspec.yml** + Deployment Strategy
- Will do In-place update to your fleet of EC2 instances
- Can use hooks to verify the deployment after each deployment phase



CodeDeploy – Deploy to an ASG

- In-place Deployment
  - Updates existing EC2 instances
  - Newly created EC2 instances by an ASG will also get automated deployments
- Blue/Green Deployment
  - A new Auto-Scaling Group is created (settings are copied)
  - Choose how long to keep the old EC2 instances (old ASG)
  - Must be using an ELB



CodeDeploy – Redeploy & Rollbacks

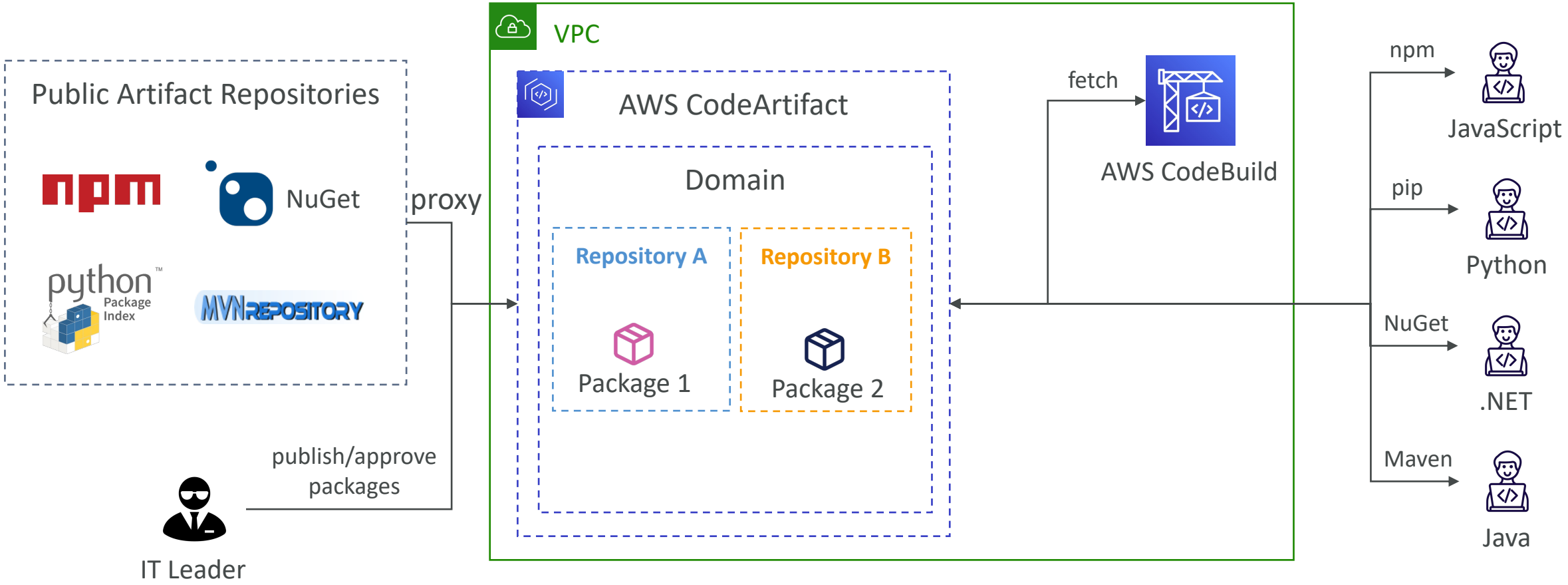
- Rollback = redeploy a previously deployed revision of your application
- Deployments can be rolled back:
  - **Automatically** – rollback when a deployment fails or rollback when a CloudWatch Alarm thresholds are met
  - **Manually**
- Disable Rollbacks — do not perform rollbacks for this deployment

- If a roll back happens, CodeDeploy redeploys the last known good revision as a new deployment (not a restored version)

AWS CodeArtifact

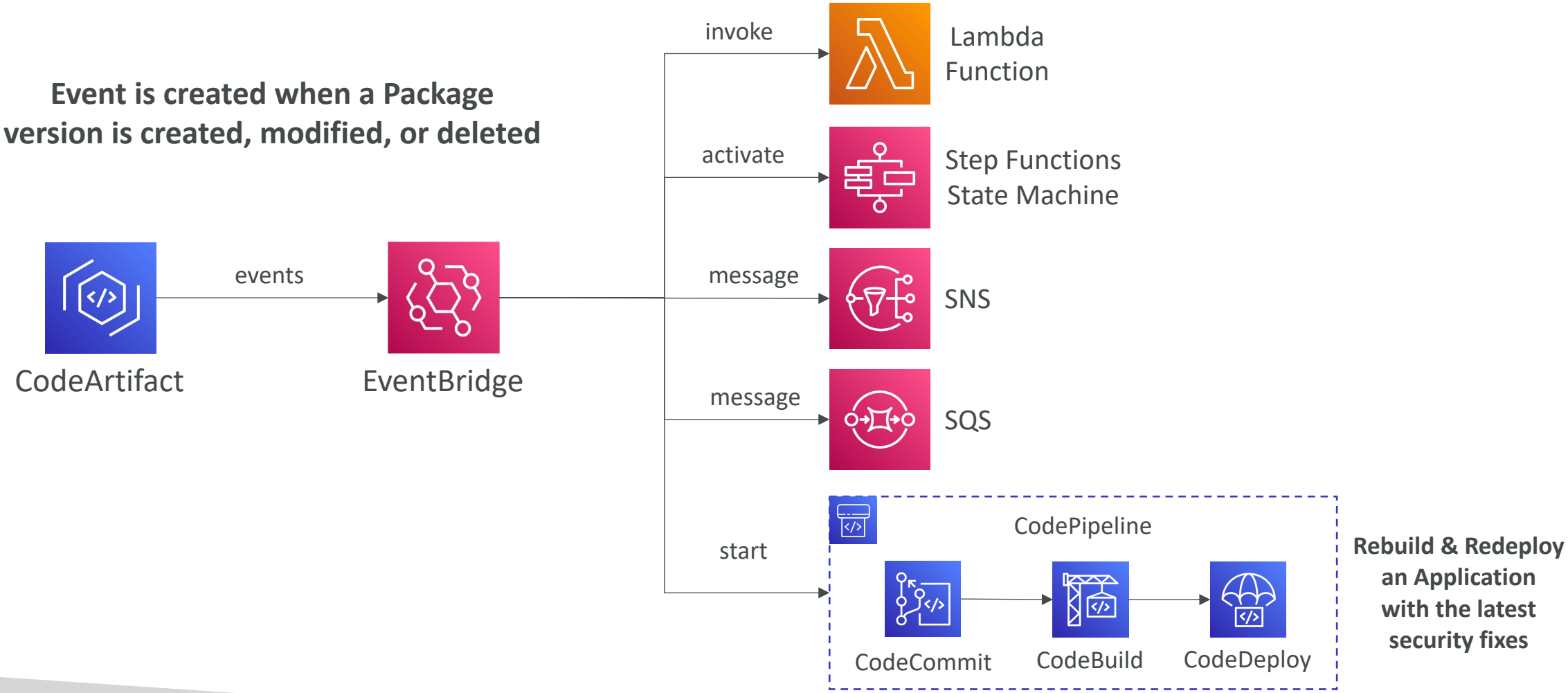
- Software packages depend on each other to be built (also called code dependencies), and new ones are created
- Storing and retrieving these dependencies is called **artifact management**
- Traditionally you need to setup your own artifact management system
- **CodeArtifact** is a secure, scalable, and cost-effective **artifact management** for software development
- Works with common dependency management tools such as Maven, Gradle, npm, yarn, twine, pip, and NuGet
- **Developers and CodeBuild can then retrieve dependencies straight from CodeArtifact**

AWS CodeArtifact



CodeArtifact – EventBridge Integration

Event is created when a Package version is created, modified, or deleted



Rebuild & Redeploy an Application with the latest security fixes

CodeArtifact – Resource Policy

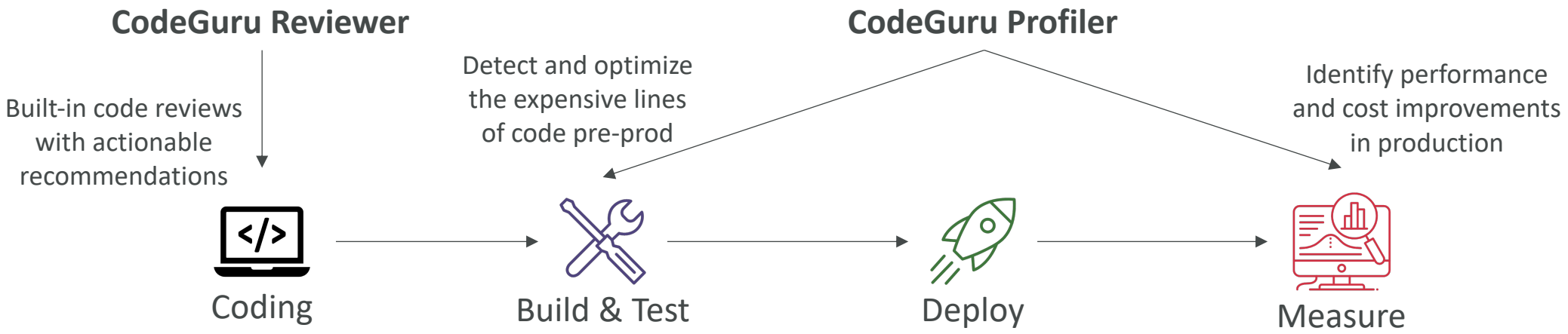
- Can be used to authorize another account to access CodeArtifact
- A given principal can either read all the packages in a repository or none of them


```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codeartifact:DescribePackageVersion",
                "codeartifact:DescribeRepository",
                "codeartifact:GetPackageVersionReadme",
                "codeartifact:GetRepositoryEndpoint",
                "codeartifact:ListPackages",
                "codeartifact:ListPackageVersions",
                "codeartifact:ListPackageVersionAssets",
                "codeartifact:ListPackageVersionDependencies",
                "codeartifact:ReadFromRepository"
            ],
            "Principal": {
                "AWS": [
                    "arn:aws:iam::123456789012:root",
                    "arn:aws:iam::222333344555:user/bob"
                ]
            },
            "Resource": "*"
        }
    ]
}
```

Repository Resource Policy

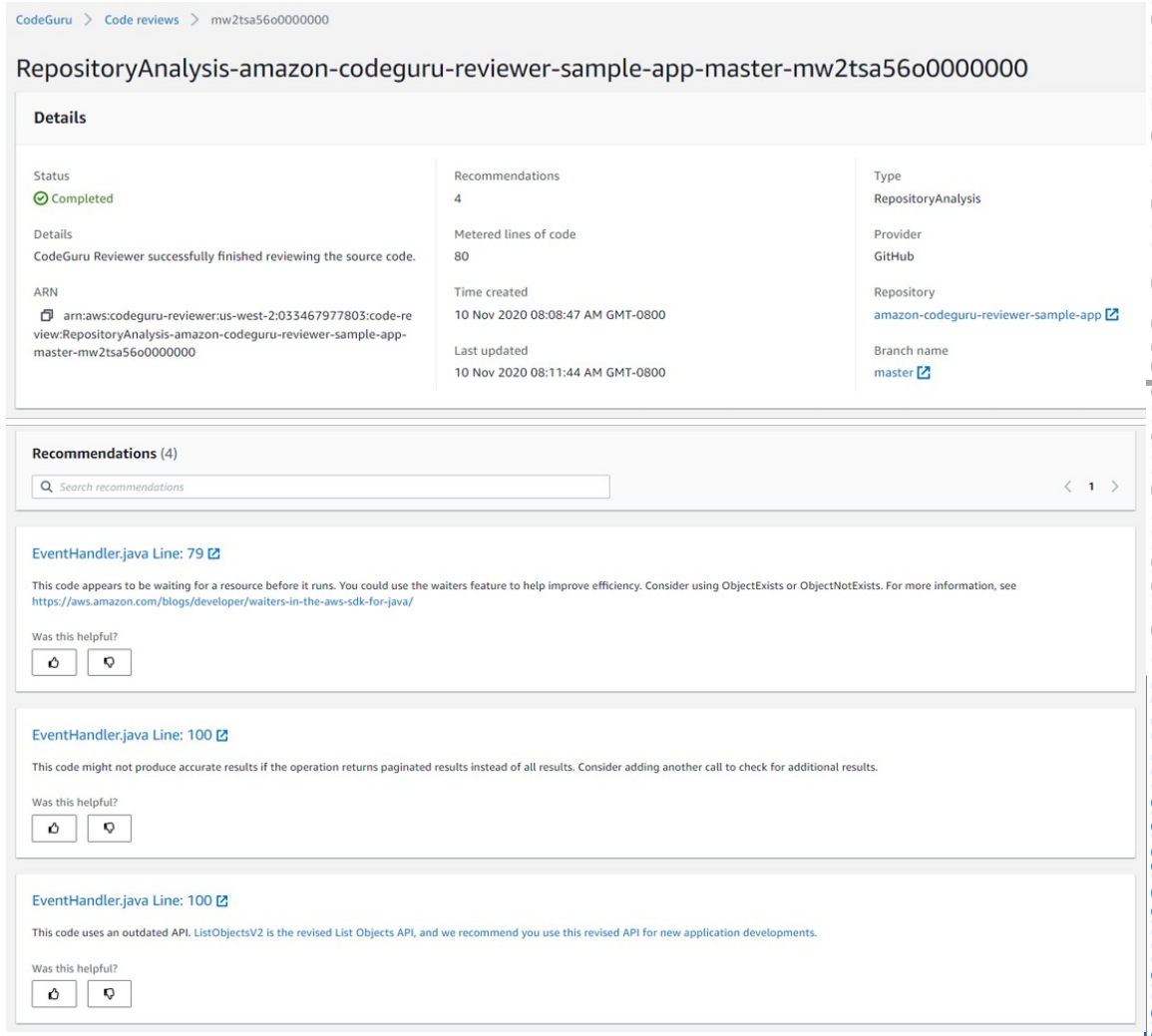
Amazon CodeGuru

- An ML-powered service for **automated code reviews** and **application performance recommendations**
- Provides two functionalities
  - **CodeGuru Reviewer:** automated code reviews for static code analysis (development)
  - **CodeGuru Profiler:** visibility/recommendations about application performance during runtime (production)



Amazon CodeGuru Reviewer

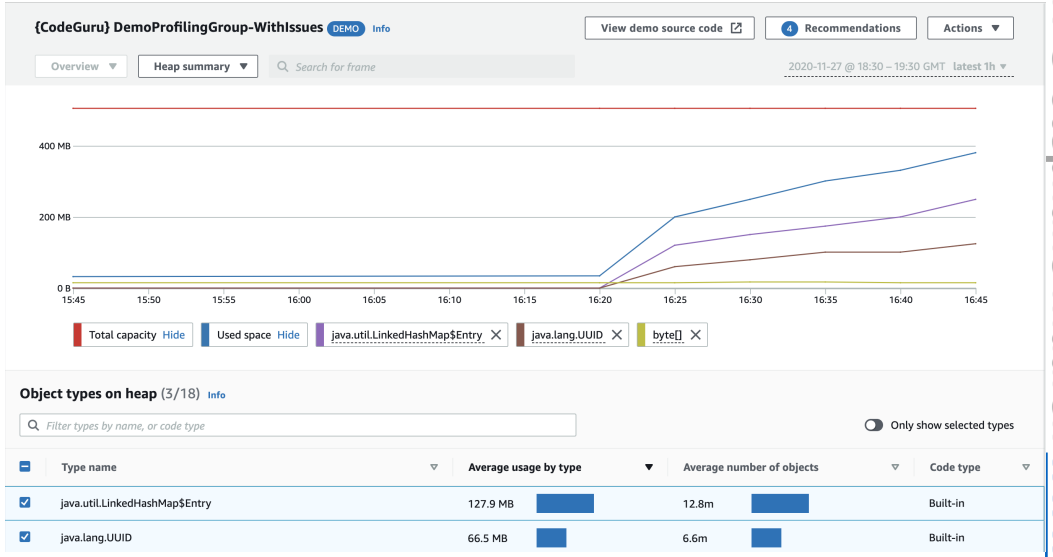
- Identify critical issues, security vulnerabilities, and hard-to-find bugs
- Example: common coding best practices, resource leaks, security detection, input validation
- Uses Machine Learning and automated reasoning
- Hard-learned lessons across millions of code reviews on 1000s of open-source and Amazon repositories
- Supports Java and Python
- Integrates with GitHub, Bitbucket, and AWS CodeCommit



https://aws.amazon.com/codeguru/features/

Amazon CodeGuru Profiler

- Helps understand the runtime behavior of your application
- Example: identify if your application is consuming excessive CPU capacity on a logging routine
- Features:
  - Identify and remove code inefficiencies
  - Improve application performance (e.g., reduce CPU utilization)
  - Decrease compute costs
  - Provides heap summary (identify which objects using up memory)
  - Anomaly Detection
- Support applications running on AWS or on-premise
- Minimal overhead on application



https://aws.amazon.com/codeguru/features/

Amazon CodeGuru – Agent Configuration

- **MaxStackDepth** – the maximum depth of the stacks in the code that is represented in the profile
  - Example: if CodeGuru Profiler finds a method A, which calls method B, which calls method C, which calls method D, then the depth is 4
  - If the MaxStackDepth is set to 2, then the profiler evaluates A and B
- **MemoryUsageLimitPercent** – the memory percentage used by the profiler
- **MinimumTimeForReportingInMilliseconds** – the minimum time between sending reports (milliseconds)
- **ReportingIntervalInMilliseconds** – the reporting interval used to report profiles (milliseconds)
- **SamplingIntervalInMilliseconds** – the sampling interval that is used to profile samples (milliseconds)
  - Reduce to have a higher sampling rate

AWS Serverless Application Model (SAM)

Taking your Serverless Development to the next level

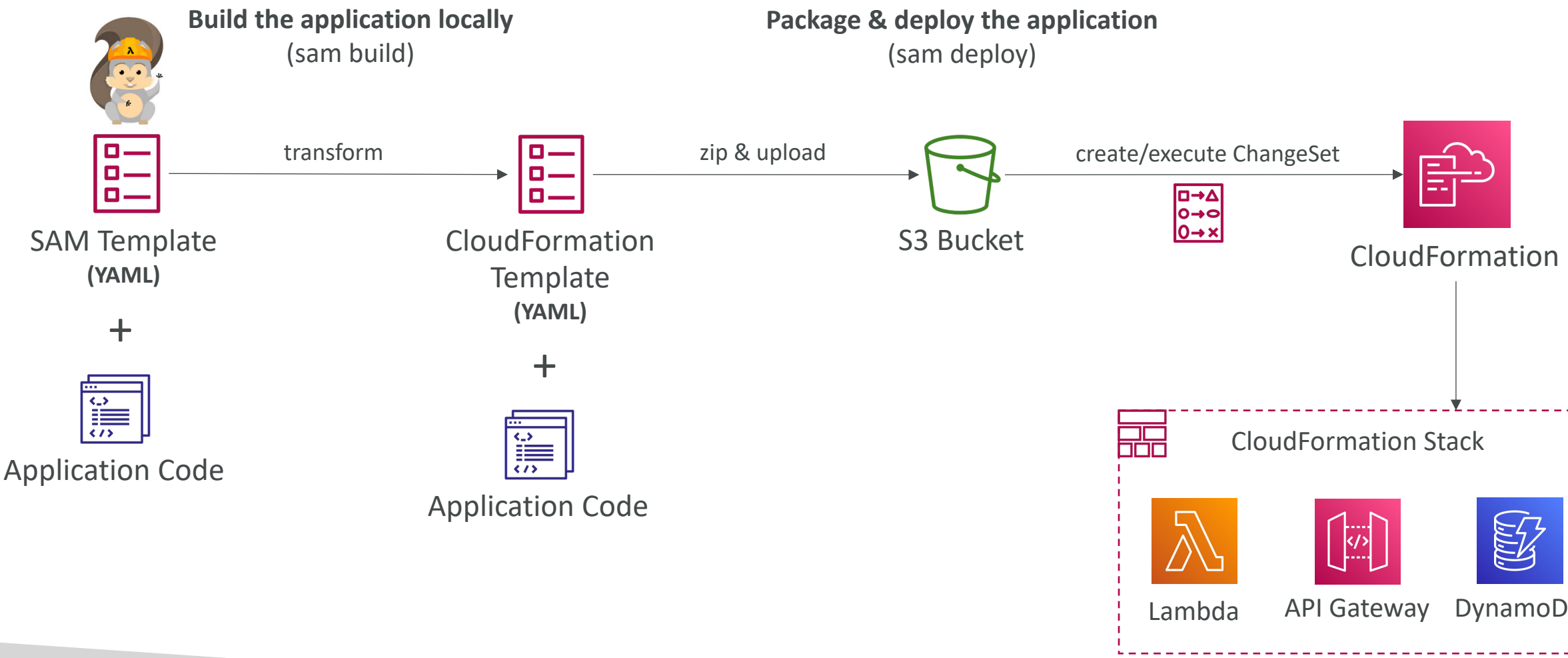
AWS SAM

- SAM = Serverless Application Model
- Framework for developing and deploying serverless applications
- All the configuration is YAML code
- Generate complex CloudFormation from simple SAM YAML file
- Supports anything from CloudFormation: Outputs, Mappings, Parameters, Resources…
- SAM can use CodeDeploy to deploy Lambda functions
- SAM can help you to run Lambda, API Gateway, DynamoDB locally

AWS SAM – Recipe

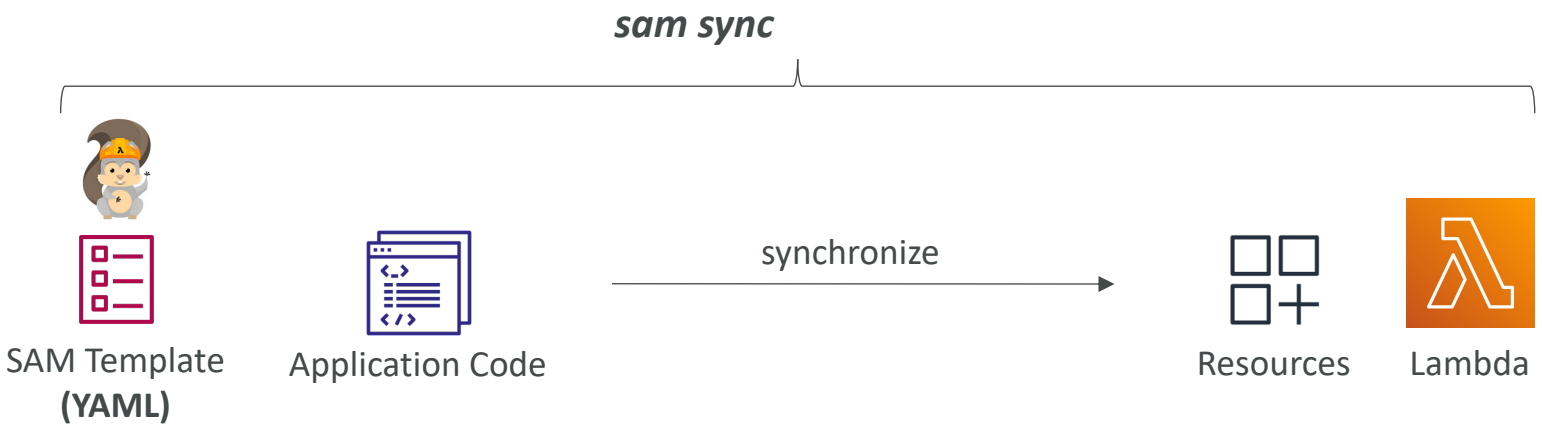
- Transform Header indicates it's SAM template:
  - `Transform: 'AWS::Serverless-2016-10-31'`
- Write Code
  - `AWS::Serverless::Function`
  - `AWS::Serverless::Api`
  - `AWS::Serverless::SimpleTable`
- Package & Deploy: `sam deploy` (optionally preceded by "`sam package`")
- Quickly sync local changes to AWS Lambda (SAM Accelerate): `sam sync --watch`

Deep Dive into SAM Deployment



SAM Accelerate (sam sync)

- SAM Accelerate is a set of features to reduce latency while deploying resources to AWS
- *sam sync*
  - Synchronizes your project declared in SAM templates to AWS
  - Synchronizes code changes to AWS without updating infrastructure (uses service APIs & **bypass CloudFormation**)

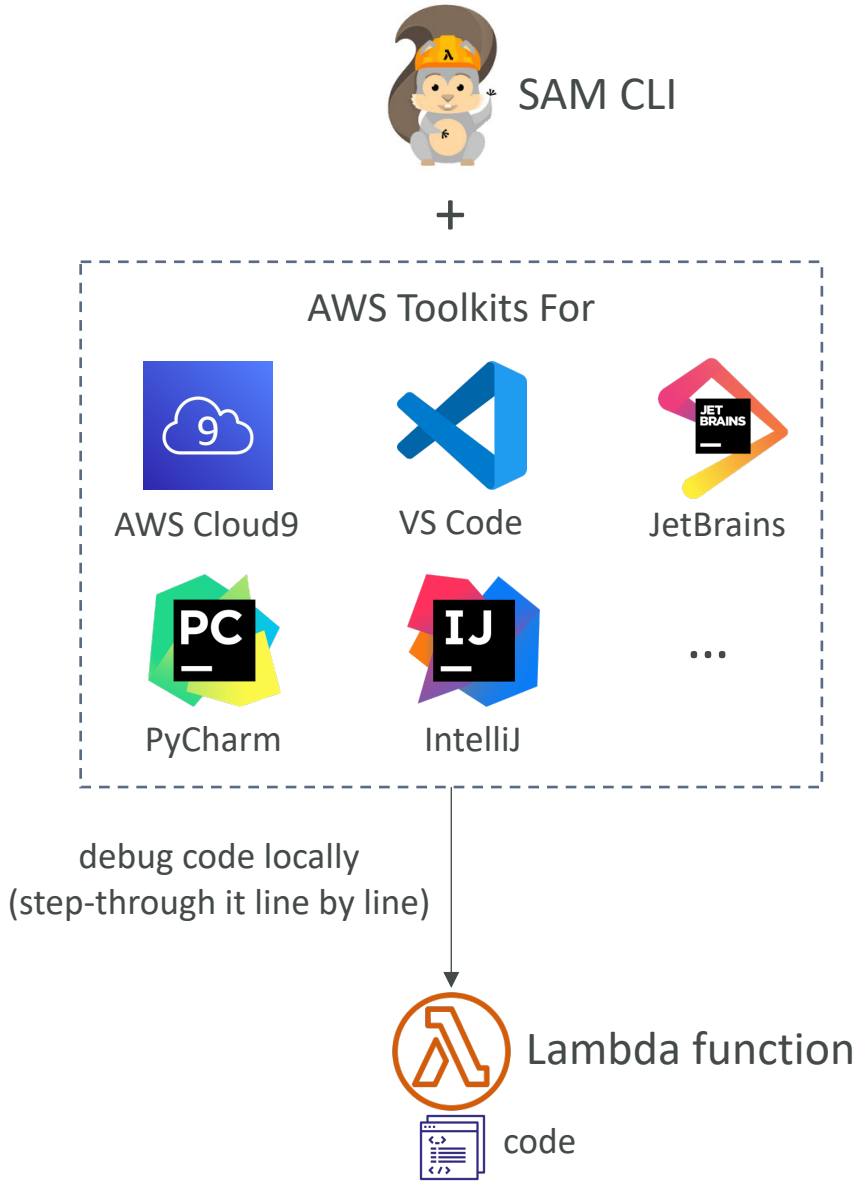


SAM Accelerate (sam sync) – Examples

- sam sync (no options)
  - Synchronize code and infrastructure
- sam sync --code
  - Synchronize code changes without updating infrastructure (bypass CloudFormation, update in seconds)
- sam sync --code --resource AWS::Serverless::Function
  - Synchronize only all Lambda functions and their dependencies
- sam sync --code --resource-id HelloWorldLambdaFunction
  - Synchronize only a specific resource by its ID
- sam sync --watch
  - Monitor for file changes and automatically synchronize when changes are detected
  - If changes include configuration, it uses **sam sync**
  - If changes are code only, it uses **sam sync --code**

SAM – CLI Debugging

- Locally build, test, and debug your serverless applications that are defined using AWS SAM templates
- Provides a lambda-like execution environment locally
- SAM CLI + AWS Toolkits => step-through and debug your code
- Supported IDEs: AWS Cloud9, Visual Studio Code, JetBrains, PyCharm, IntelliJ, …
- **AWS Toolkits:** IDE plugins which allows you to build, test, debug, deploy, and invoke Lambda functions built using AWS SAM



SAM Policy Templates

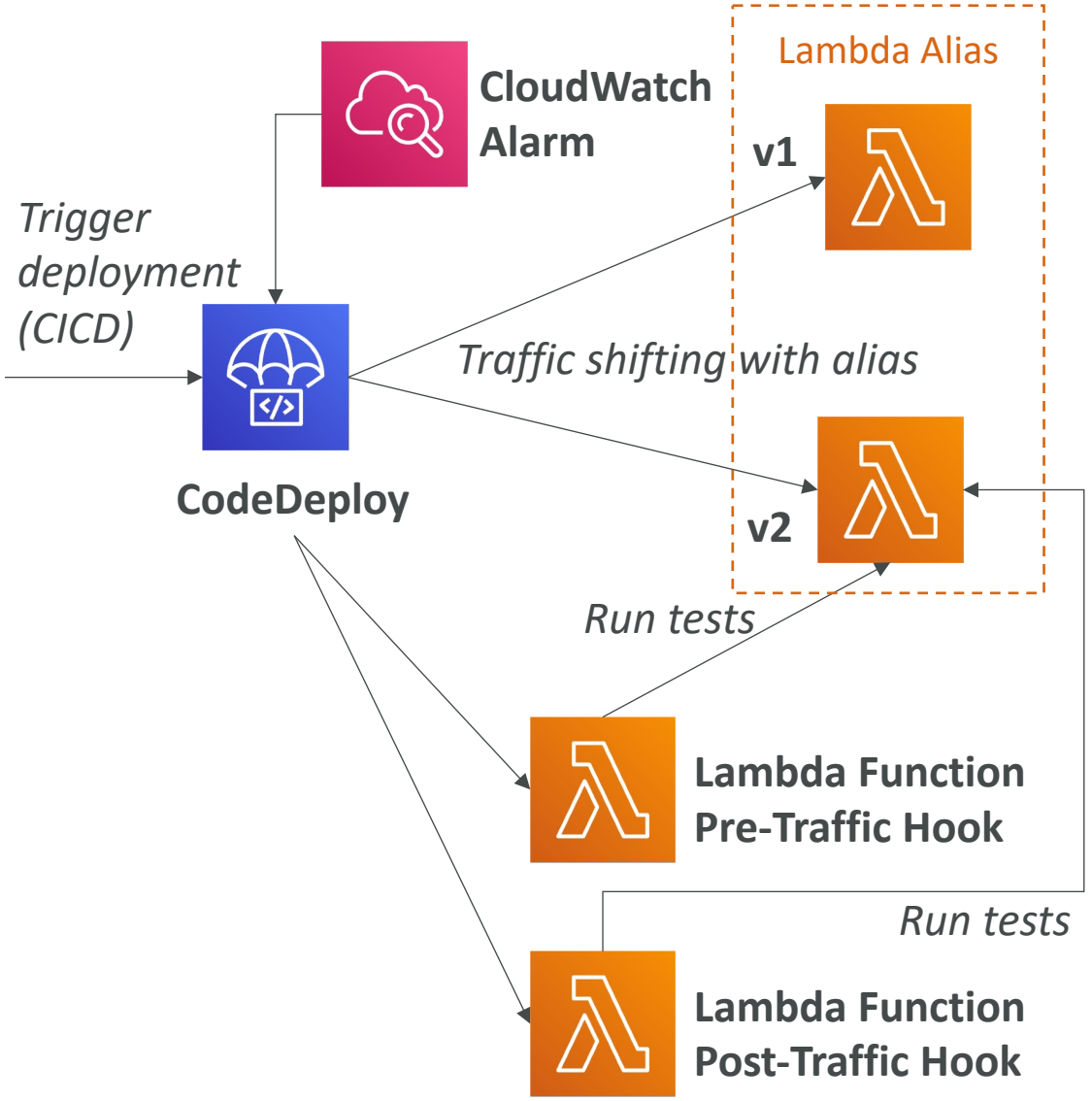
- List of templates to apply permissions to your Lambda Functions
- Full list available here: https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/serverless-policy-templates.html#serverless-policy-template-table
- Important examples:
  - **S3ReadPolicy:** Gives read only permissions to objects in S3
  - **SQSPollerPolicy:** Allows to poll an SQS queue
  - **DynamoDBCrudPolicy:** CRUD = create read update delete

```
MyFunction:
  Type: 'AWS::Serverless::Function'
  Properties:
    CodeUri: ${codeuri}
    Handler: hello.handler
    Runtime: python2.7
    Policies:
      - SQSPollerPolicy:
          QueueName:
            !GetAtt MyQueue.QueueName
```

SAM and CodeDeploy

- SAM framework natively uses CodeDeploy to update Lambda functions

- Traffic Shifting feature
- Pre and Post traffic hooks features to validate deployment (before the traffic shift starts and after it ends)
- Easy & automated rollback using CloudWatch Alarms



SAM and CodeDeploy

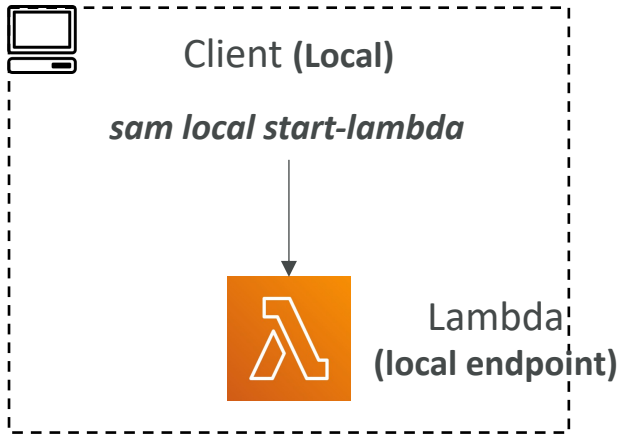
- AutoPublishAlias
  - Detects when new code is being deployed
  - Creates and publishes an updated version of that function with the latest code
  - Points the alias to the updated version of the Lambda function
- DeploymentPreference
  - Canary, Linear, AllAtOnce
- Alarms
  - Alarms that can trigger a rollback
- Hooks
  - Pre and post traffic shifting Lambda functions to test your deployment

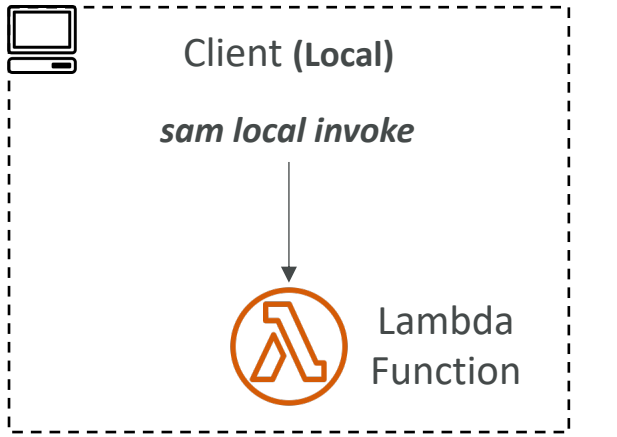
```yaml
Resources:
MyLambdaFunction:
  Type: AWS::Serverless::Function
  Properties:
    Handler: index.handler
    Runtime: nodejs12.x
    CodeUri: s3://bucket/code.zip

    AutoPublishAlias: live

    DeploymentPreference:
      Type: Canary10Percent10Minutes
      Alarms:
        # A list of alarms that you want to monitor
        - !Ref AliasErrorMetricGreaterThanZeroAlarm
        - !Ref LatestVersionErrorMetricGreaterThanZeroAlarm
      Hooks:
        # Validation Lambda functions that are run before & after traffic shifting
        PreTraffic: !Ref PreTrafficLambdaFunction
        PostTraffic: !Ref PostTrafficLambdaFunction
```

SAM – Local Capabilities

- Locally start AWS Lambda
  - *sam local start-lambda*
  - Starts a local endpoint that emulates AWS Lambda
  - Can run automated tests against this local endpoint

- Locally Invoke Lambda Function
  - *sam local invoke*
  - Invoke Lambda function with payload once and quit after invocation completes
  - Helpful for generating test cases
  - If the function make API calls to AWS, make sure you are using the correct *--profile* option

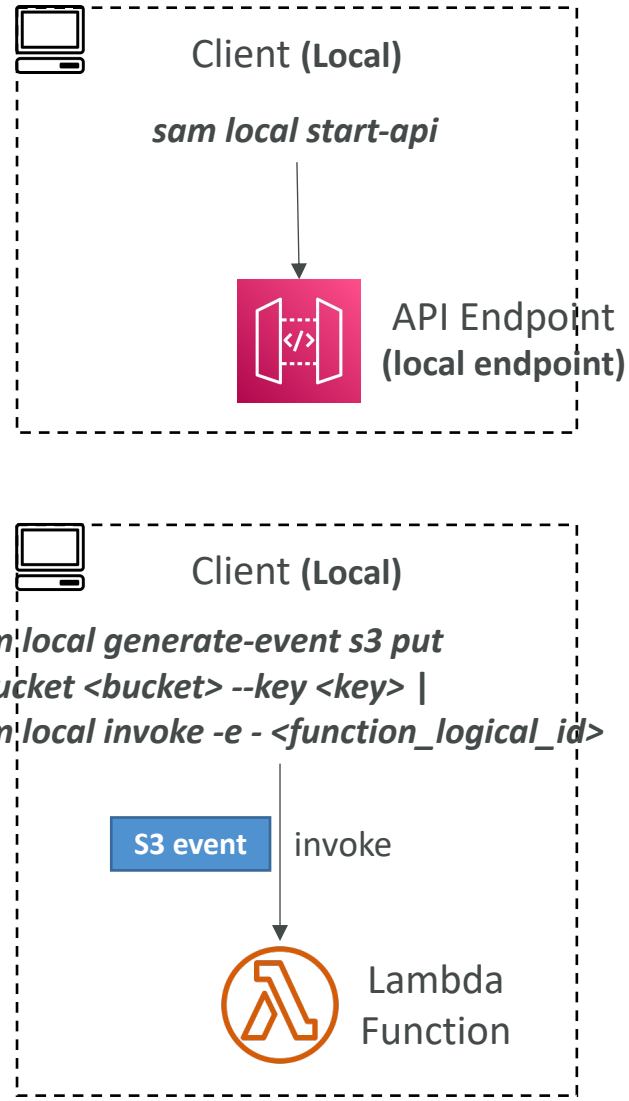




SAM – Local Capabilities

- Locally Start an API Gateway Endpoint
  - *sam local start-api*
  - Starts a local HTTP server that hosts all your functions
  - Changes to functions are automatically reloaded

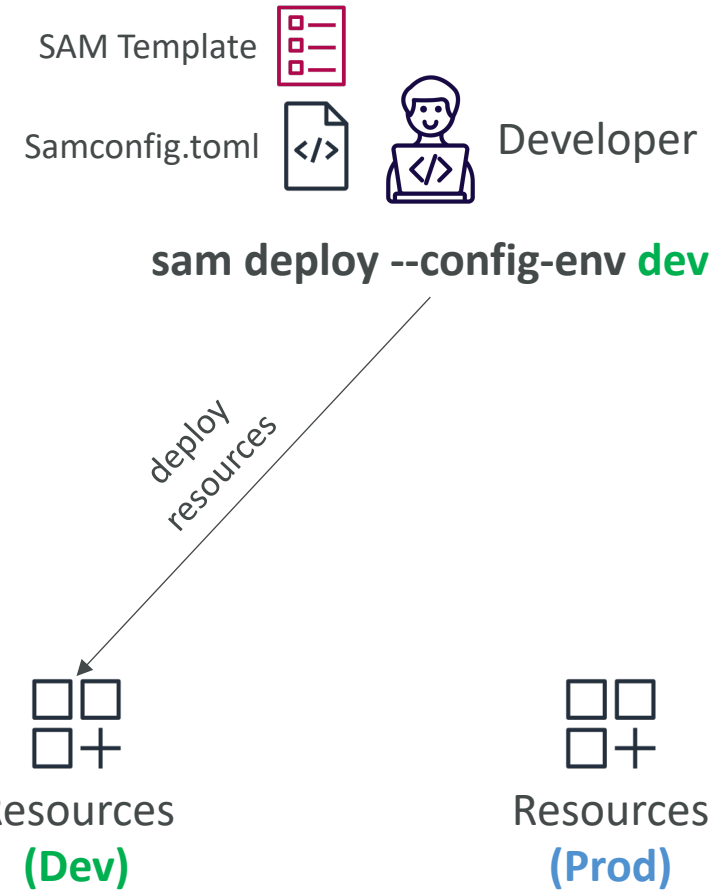
- Generate AWS Events for Lambda Functions
  - *sam local generate-event*
  - Generate sample payloads for event sources
  - S3, API Gateway, SNS, Kinesis, DynamoDB…



SAM – Multiple Environments

samconfig.toml

```
version = 0.1

[dev.deploy.parameters]
stack_name = "my-dev-stack"
s3_bucket = "XXXXX-dev"
s3_prefix = "XXXXX/dev"
region = "us-east-1"
capabilities = "CAPABILITY_IAM"
parameter_overrides = "Environment=Development"

[prod.deploy.parameters]
stack_name = "my-prod-stack"
s3_bucket = "XXXXX-prod"
s3_prefix = "XXXXX/prod"
region = "us-east-1"
capabilities = "CAPABILITY_IAM"
parameter_overrides = "Environment=Production"

[dev.sync.parameters]
watch = true

[prod.sync.parameters]
watch = false
```



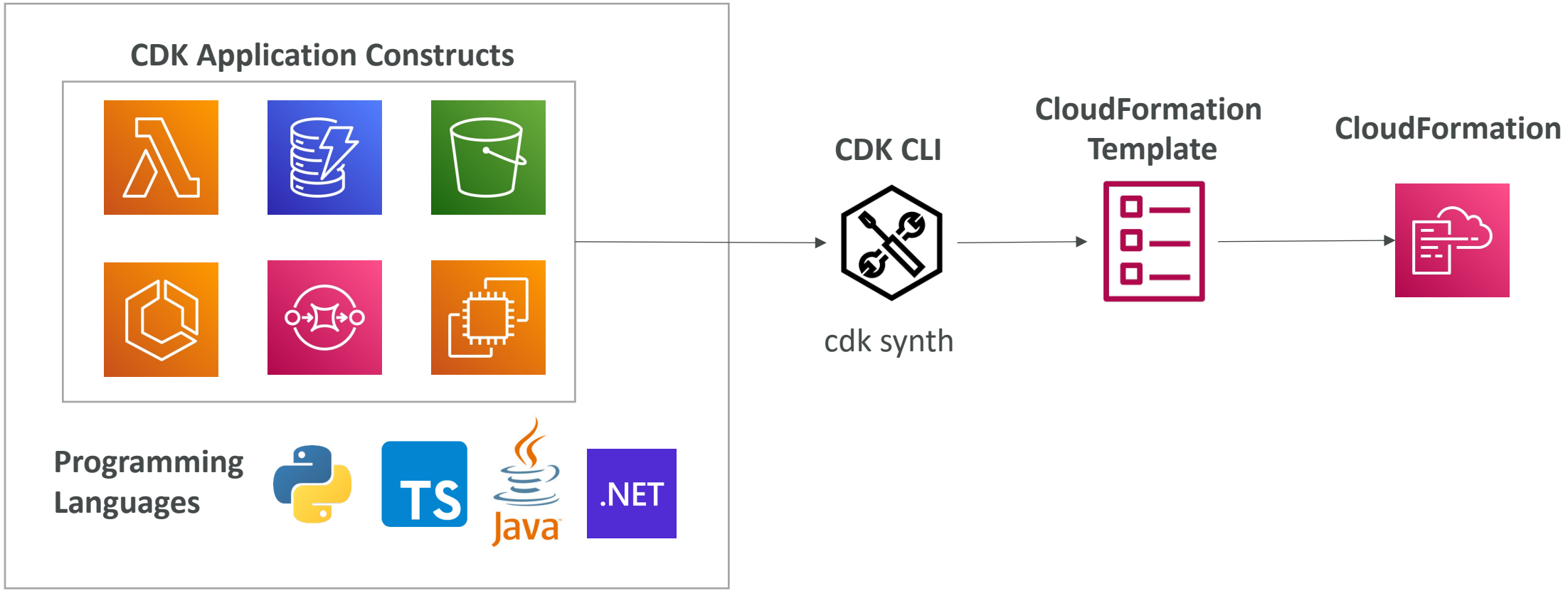
AWS Cloud Development Kit

AWS Cloud Development Kit (CDK)

- Define your cloud infrastructure using a familiar language:
  - JavaScript/TypeScript, Python, Java, and .NET
- Contains high level components called **constructs**
- The code is "compiled" into a CloudFormation template (JSON/YAML)
- **You can therefore deploy infrastructure and application runtime code together**
  - Great for Lambda functions
  - Great for Docker containers in ECS / EKS

```
export class MyEcsConstructStack extends core.Stack {
  constructor(scope: core.App, id: string, props?: core.StackProps)
    super(scope, id, props);

    const vpc = new ec2.Vpc(this, "MyVpc", {
      maxAzs: 3 // Default is all AZs in region
    });

    const cluster = new ecs.Cluster(this, "MyCluster", {
      vpc: vpc
    });

    // Create a load-balanced Fargate service and make it public
    new ecs_patterns.ApplicationLoadBalancedFargateService(this, "My
      cluster: cluster, // Required
      cpu: 512, // Default is 256
      desiredCount: 6, // Default is 1
      taskImageOptions: { image: ecs.ContainerImage.fromRegistry("am
      memoryLimitMiB: 2048, // Default is 512
      publicLoadBalancer: true // Default is false
    });
  }
}
```

CDK in a diagram

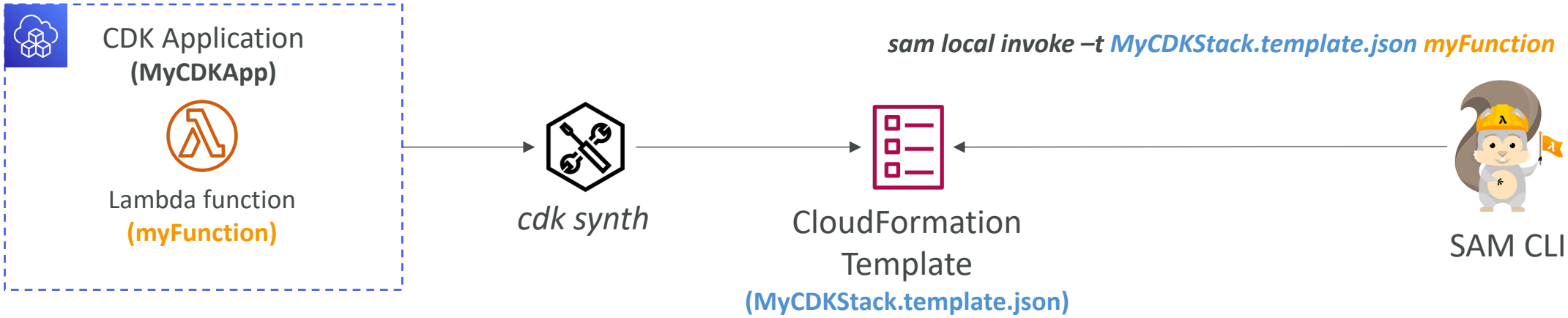


CDK vs SAM

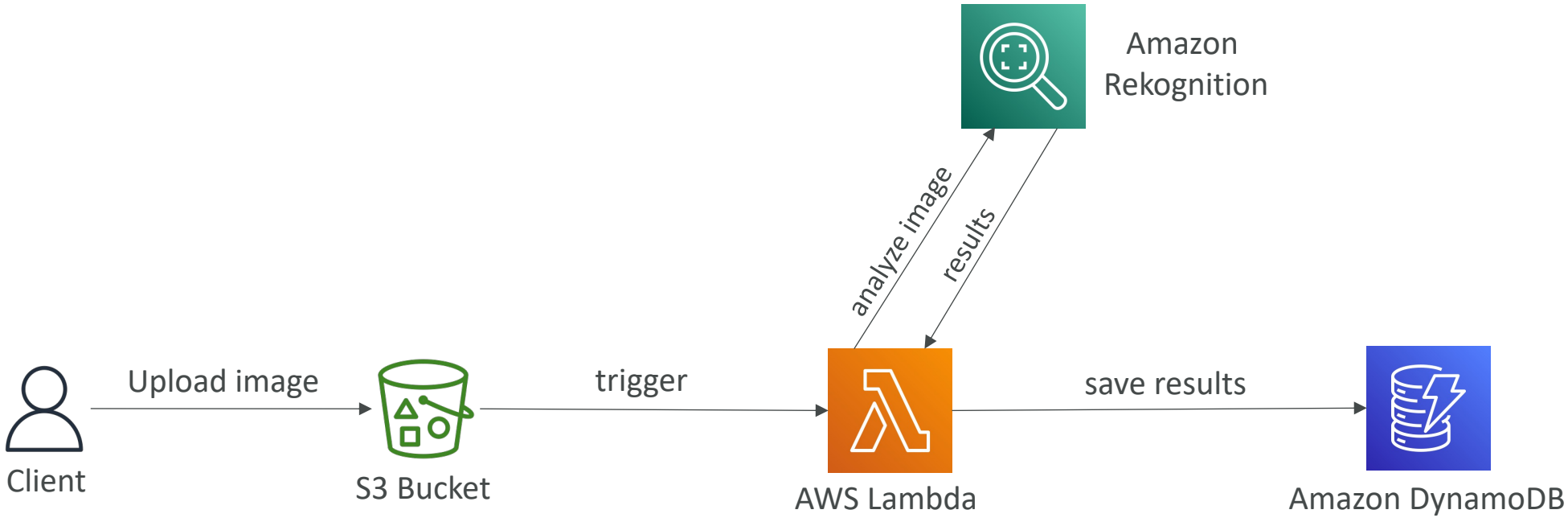
- SAM:
  - Serverless focused
  - Write your template declaratively in JSON or YAML
  - Great for quickly getting started with Lambda
  - Leverages CloudFormation
- CDK:
  - All AWS services
  - Write infra in a programming language JavaScript/TypeScript, Python, Java, and .NET
  - Leverages CloudFormation

CDK + SAM

- You can use SAM CLI to locally test your CDK apps
- You must first run *cdk synth*



CDK Hands-On



CDK Constructs

- CDK Construct is a component that encapsulates everything CDK needs to create the final CloudFormation stack
- Can represent a single AWS resource (e.g., S3 bucket) or multiple related resources (e.g., worker queue with compute)
- AWS Construct Library
  - A collection of Constructs included in AWS CDK which contains Constructs for every AWS resource
  - Contains 3 different levels of Constructs available (L1, L2, L3)
- Construct Hub – contains additional Constructs from AWS, 3rd parties, and open-source CDK community

CDK Constructs – Layer 1 Constructs (L1)

- Can be called *CFN Resources* which represents all resources directly available in CloudFormation
- Constructs are periodically generated from **CloudFormation Resource Specification**
- Construct names start with **Cfn** (e.g., **CfnBucket**)
- **You must explicitly configure all resource properties**

```
const bucket = new s3.CfnBucket(this, "MyBucket", {
    bucketName: "MyBucket"
});
```

CDK Constructs – Layer 2 Constructs (L2)

- Represents AWS resources but with a higher level (intent-based API)
- Similar functionality as L1 but with convenient defaults and boilerplate
  - You don't need to know all the details about the resource properties
- Provide methods that make it simpler to work with the resource (e.g., **bucket.addLifeCycleRule()**)

```
const s3 = require('aws-cdk-lib/aws-s3');

const bucket = new s3.Bucket(this, 'MyBucket', {
  versioned: true,
  encryption: s3.BucketEncryption.KMS
});

// Returns the HTTPS URL of an S3 Object
const objectUrl = bucket.urlForObject('MyBucket/MyObject');
```

CDK Constructs – Layer 3 Constructs (L3)

- Can be called *Patterns*, which represents multiple related resources
- Helps you complete common tasks in AWS
- Examples:
  - *aws-apigateway.LambdaRestApi* represents an API Gateway backed by a Lambda function
  - *aws-ecs-patterns.ApplicationLoadBalancerFargateService* which represents an architecture that includes a Fargate cluster with Application Load Balancer

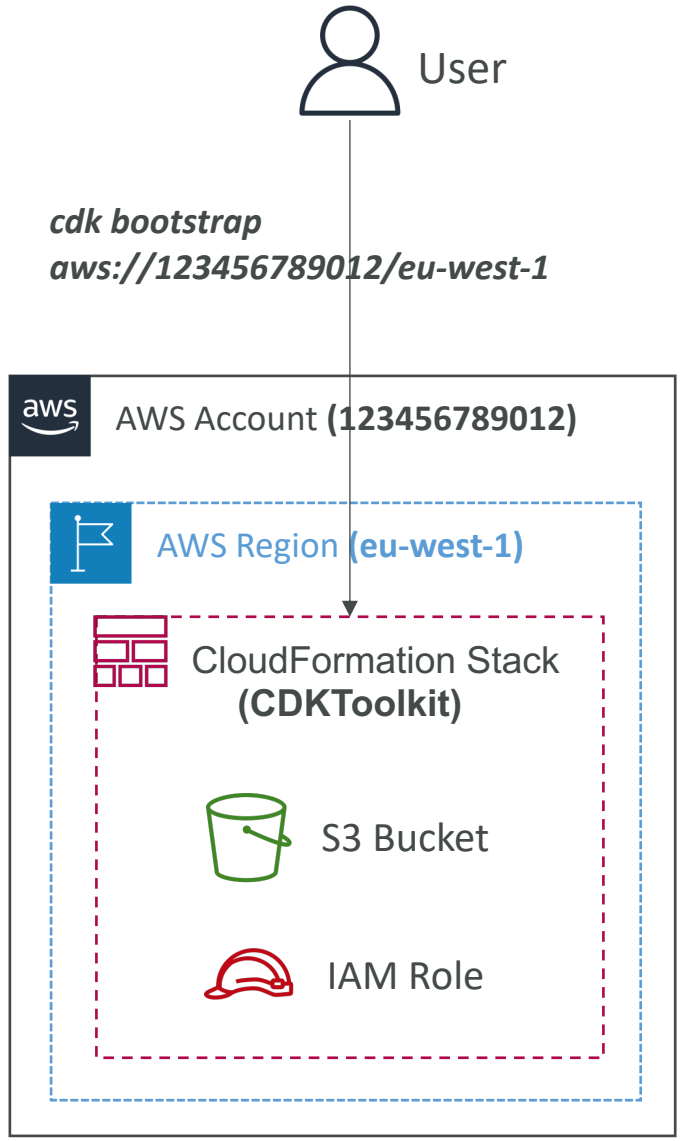
```
const api = new apigateway.LambdaRestApi(this, 'myapi', {
  handler: backend,
  proxy: false
});

const items = api.root.addResource('items');
items.addMethod('GET');  // GET /items
items.addMethod('POST'); // POST /items

const item = items.addResource('{item}');
item.addMethod('GET');   // GET /items/{item}

item.addMethod('DELETE', new apigateway.HttpIntegration('http://amazon.com'));
```

CDK – Important Commands to know

Command	Description
npm install -g aws-cdk-lib	Install the CDK CLI and libraries
cdk init app	Create a new CDK project from a specified template
cdk synth	Synthesizes and prints the CloudFormation template
cdk bootstrap	Deploys the CDK Toolkit staging Stack
cdk deploy	Deploy the Stack(s)
cdk diff	View differences of local CDK and deployed Stack
cdk destroy	Destroy the Stack(s)

CDK – Bootstrapping

- The process of provisioning resources for CDK before you can deploy CDK apps into an AWS environment
- **AWS Environment = account & region**
- CloudFormation Stack called **CDKToolkit** is created and contains:
  - **S3 Bucket** – to store files
  - **IAM Roles** – to grant permissions to perform deployments
- You must run the following command for each new environment:
  - *cdk bootstrap aws://<aws_account>/<aws_region>*
- Otherwise, you will get an **error "Policy contains a statement with one or more invalid principal"**



CDK – Testing

- To test CDK apps, use **CDK Assertions Module** combined with popular test frameworks such as Jest (JavaScript) or Pytest (Python)
- Verify we have specific resources, rules, conditions, parameters…
- Two types of tests:
  - **Fine-grained Assertions (common)** – test specific aspects of the CloudFormation template (e.g., check if a resource has this property with this value)
  - **Snapshot Tests** – test the synthesized CloudFormation template against a previously stored baseline template
- To import a template
  - Template.fromStack(MyStack) : stack built in CDK
  - Template.fromString(mystring) : stack build outside CDK

```
describe("StateMachineStack", () => {
    test("synthesizes the way we expect", () => {
        ...

        // Prepare the stack for assertions
        const template = Template.fromStack(MyStack);

        // Assert it creates Lambda with correct properties...
        template.hasResourceProperties("AWS::Lambda::Function", {
            Handler: "handler",
            Runtime: "nodejs14.x",
        });

        // Assert it creates the SNS subscription...
        template.resourceCountIs("AWS::SNS::Subscription", 1);

        // Assert the synthesized CloudFormation template
        // against a previously stored baseline template
        expect(template.toJSON()).toMatchSnapshot();
    }
});
```

Fine-grained Assertions

Snapshot Test

Amazon Cognito

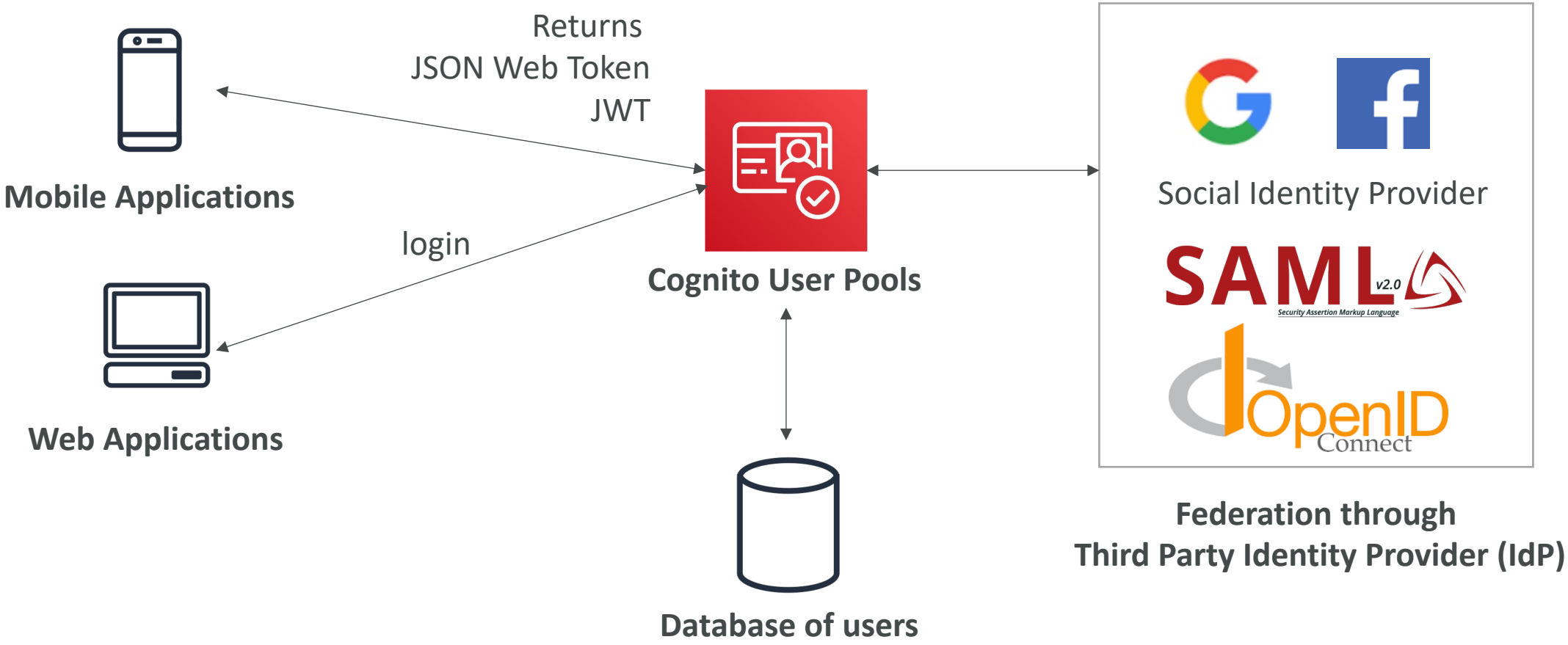
Amazon Cognito

- Give users an identity to interact with our web or mobile application
- Cognito User Pools:
  - Sign in functionality for app users
  - Integrate with API Gateway & Application Load Balancer
- Cognito Identity Pools (Federated Identity):
  - Provide AWS credentials to users so they can access AWS resources directly
  - Integrate with Cognito User Pools as an identity provider
- Cognito vs IAM: "hundreds of users", "mobile users", "authenticate with SAML"

Cognito User Pools (CUP) – User Features

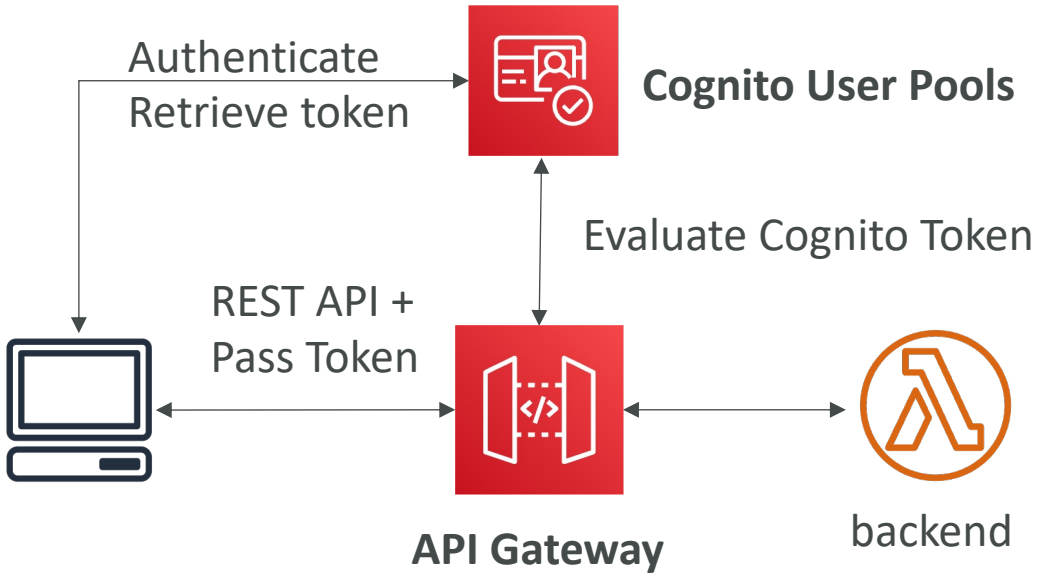
- Create a serverless database of user for your web & mobile apps
- Simple login: Username (or email) / password combination
- Password reset
- Email & Phone Number Verification
- Multi-factor authentication (MFA)
- Federated Identities: users from Facebook, Google, SAML…
- Feature: block users if their credentials are compromised elsewhere
- Login sends back a JSON Web Token (JWT)

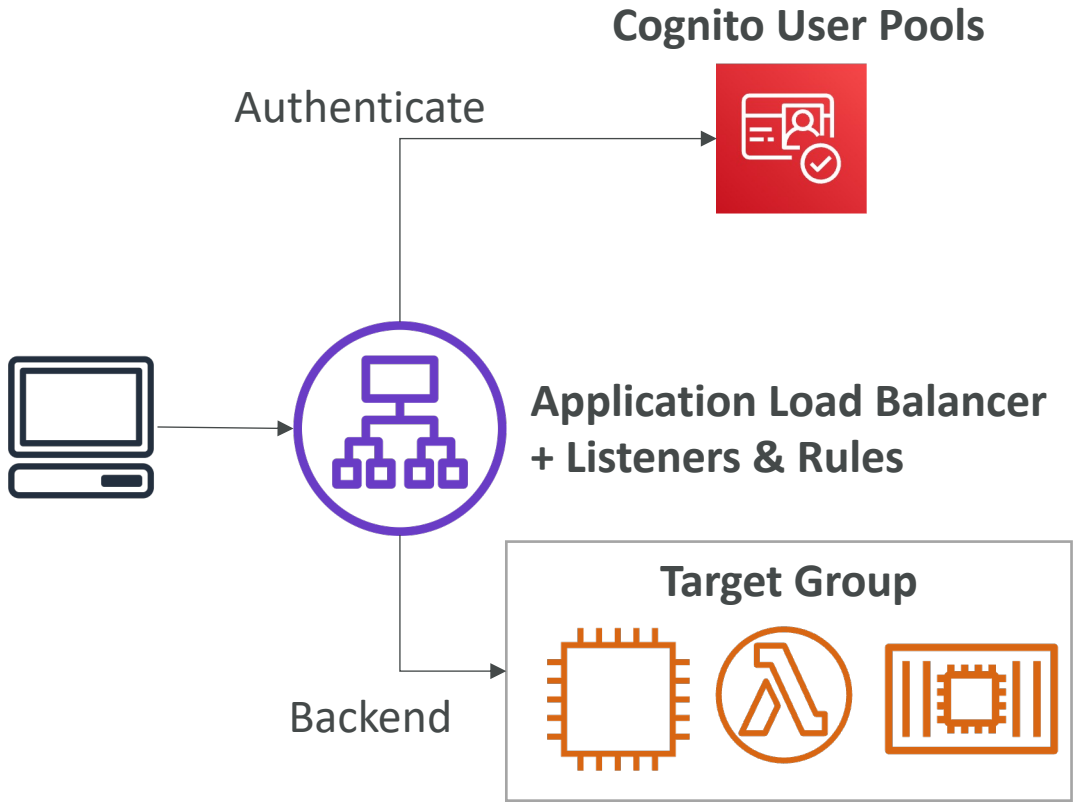
Cognito User Pools (CUP) – Diagram



Cognito User Pools (CUP) - Integrations

- CUP integrates with **API Gateway** and **Application Load Balancer**





Cognito User Pools – Lambda Triggers

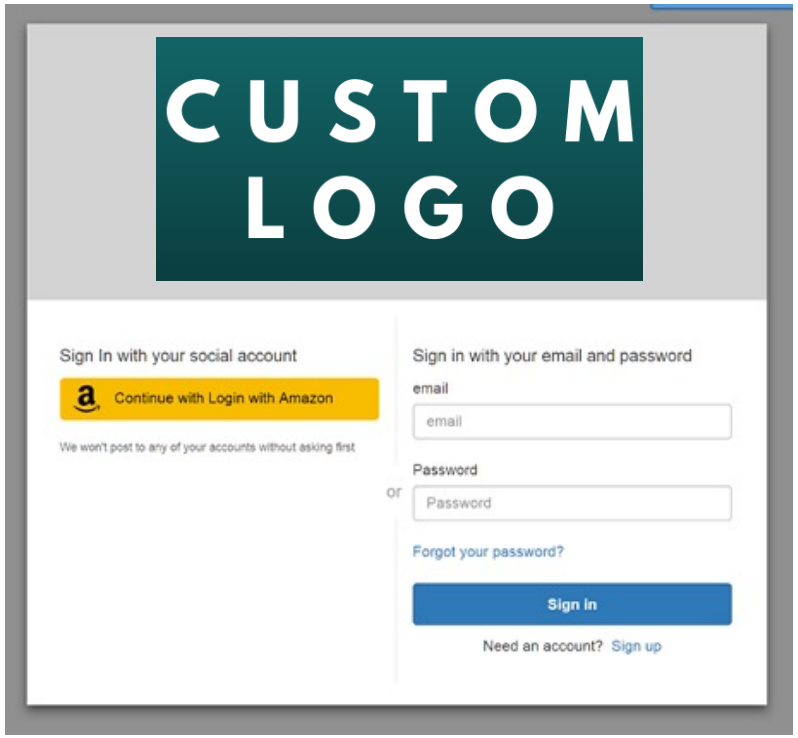
- CUP can invoke a Lambda function synchronously on these triggers:

User Pool Flow	Operation	Description
Authentication Events	Pre Authentication Lambda Trigger	Custom validation to accept or deny the sign-in request
	Post Authentication Lambda Trigger	Event logging for custom analytics
	Pre Token Generation Lambda Trigger	Augment or suppress token claims
Sign-Up	Pre Sign-up Lambda Trigger	Custom validation to accept or deny the sign-up request
	Post Confirmation Lambda Trigger	Custom welcome messages or event logging for custom analytics
	Migrate User Lambda Trigger	Migrate a user from an existing user directory to user pools
Messages	Custom Message Lambda Trigger	Advanced customization and localization of messages
Token Creation	Pre Token Generation Lambda Trigger	Add or remove attributes in Id tokens

https://docs.aws.amazon.com/cognito/latest/developerguide/cognito-user-identity-pools-working-with-aws-lambda-triggers.html

Cognito User Pools – Hosted Authentication UI

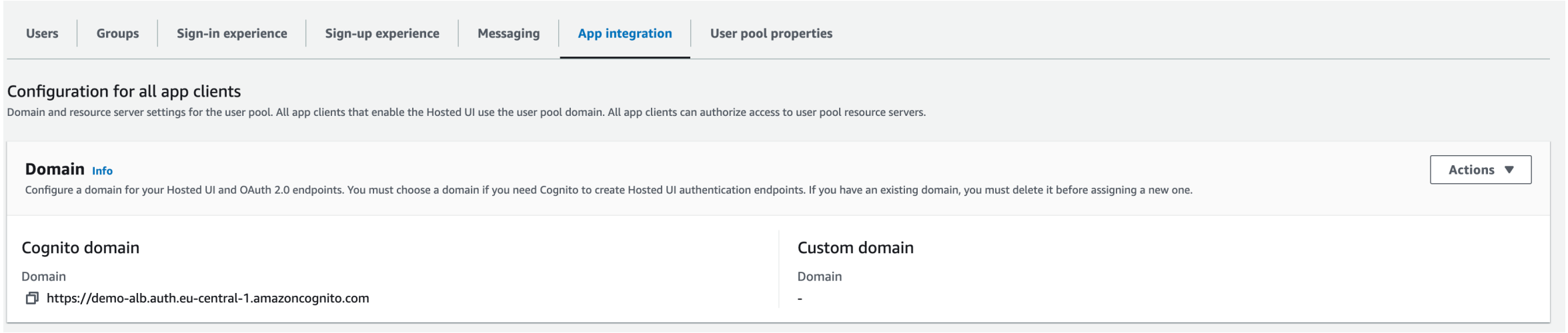
- Cognito has a **hosted authentication UI** that you can add to your app to handle sign-up and sign-in workflows
- Using the hosted UI, you have a foundation for integration with social logins, OIDC or SAML
- Can customize with a **custom logo** and **custom CSS**



https://aws.amazon.com/blogs/aws/launch-amazon-cognito-user-pools-general-availability-app-integration-and-federation/

CUP – Hosted UI Custom Domain

- For custom domains, you must create an ACM certificate in us-east-1
- The custom domain must be defined in the "App Integration" section



CUP – Adaptive Authentication

- Block sign-ins or require MFA if the login appears suspicious
- Cognito examines each sign-in attempt and generates a risk score (low, medium, high) for how likely the sign-in request is to be from a malicious attacker
- Users are prompted for a second MFA only when risk is detected
- Risk score is based on different factors such as if the user has used the same device, location, or IP address
- Checks for compromised credentials, account takeover protection, and phone and email verification
- Integration with CloudWatch Logs (sign-in attempts, risk score, failed challenges…)

Decoding a ID Token; JWT – JSON Web Token

- CUP issues JWT tokens (Base64 encoded):
  - Header
  - Payload
  - Signature
- The signature must be verified to ensure the JWT can be trusted
- Libraries can help you verify the validity of JWT tokens issued by Cognito User Pools
- The Payload will contain the user information (sub UUID, given_name, email, phone_number, attributes…)
- From the sub UUID, you can retrieve all users details from Cognito / OIDC

```
<header>.
{
  "sub":"aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee",
  "email":"my-test-user@example.com",
  "email_verified":true,
  "middle_name":"Jane",
  "cognito:username":"my-test-user",
  "cognito:groups":[
     "my-test-group"
  ],
  "cognito:roles":[
     "arn:aws:iam::111122223333:role/my-test-role"
  ],
  "cognito:preferred_role":"arn:aws:iam::111122223333:role/my-test-role",
  "iss":"https://cognito-idp.us-west-2.amazonaws.com/us-west-2_example",
  "nonce":"abcdefg",
  "origin_jti":"aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee",
  "aud":"xxxxxxxxxxxxexample",
  "event_id":"64f513be-32db-42b0-b78e-b02127b4f463",
  "token_use":"id",
  "auth_time":1676312777,
  "exp":1676316377,
  "iat":1676312777,
  "jti":"aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee",
}
.<token signature>
```

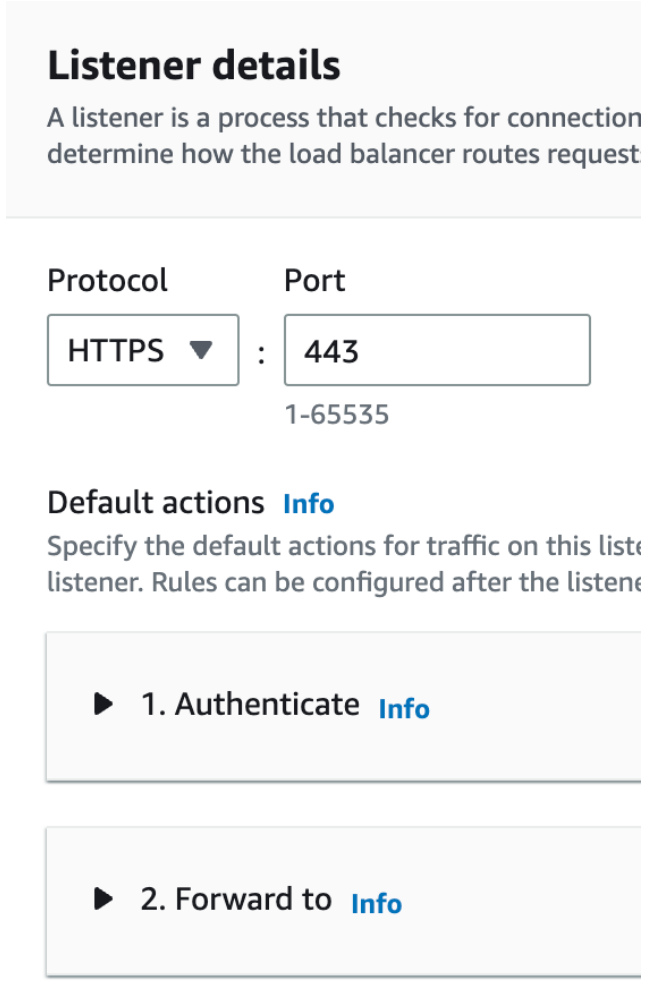
User ID in Cognito DB

Expiry & Issued At

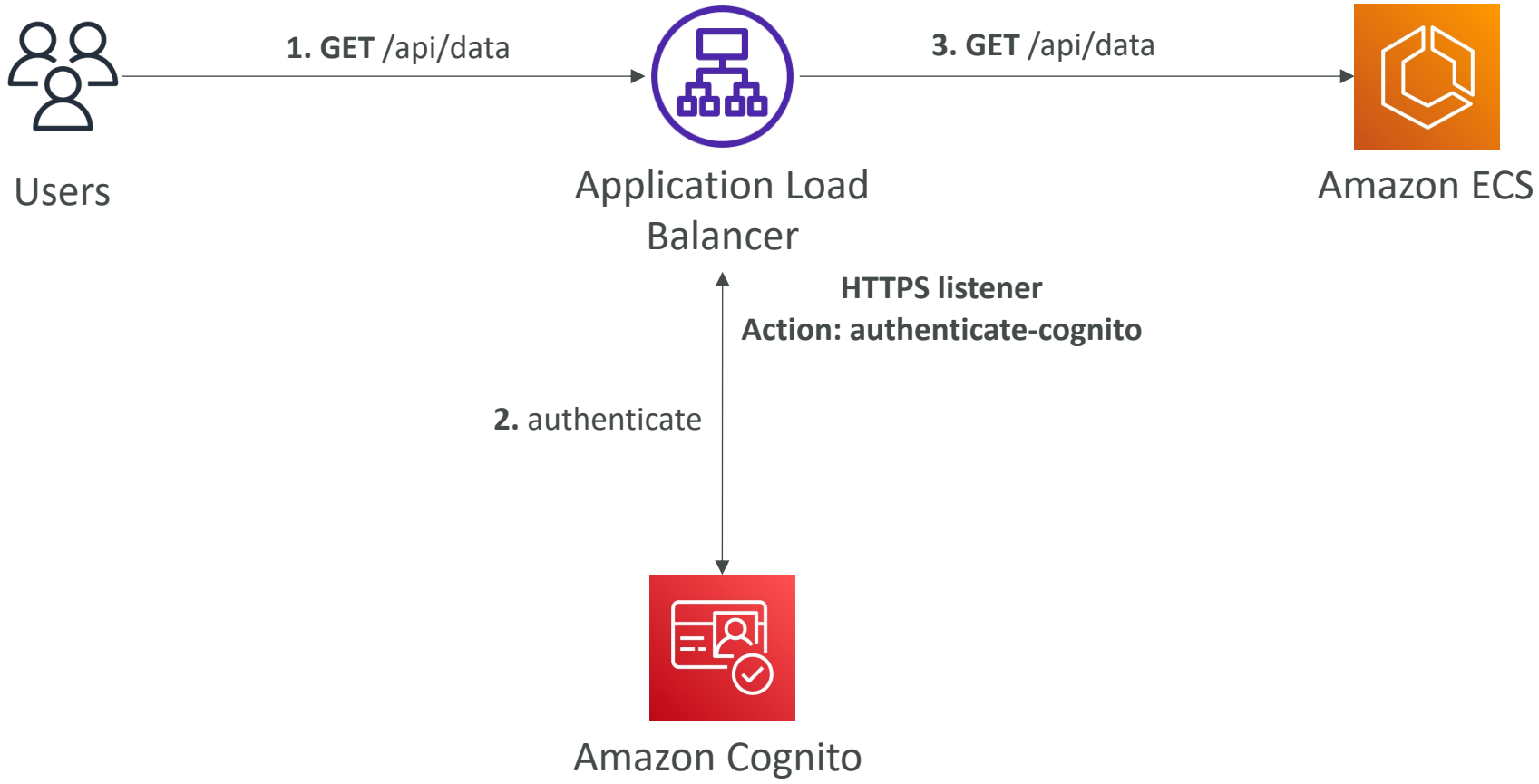
ID JWT Token Payload

Application Load Balancer – Authenticate Users

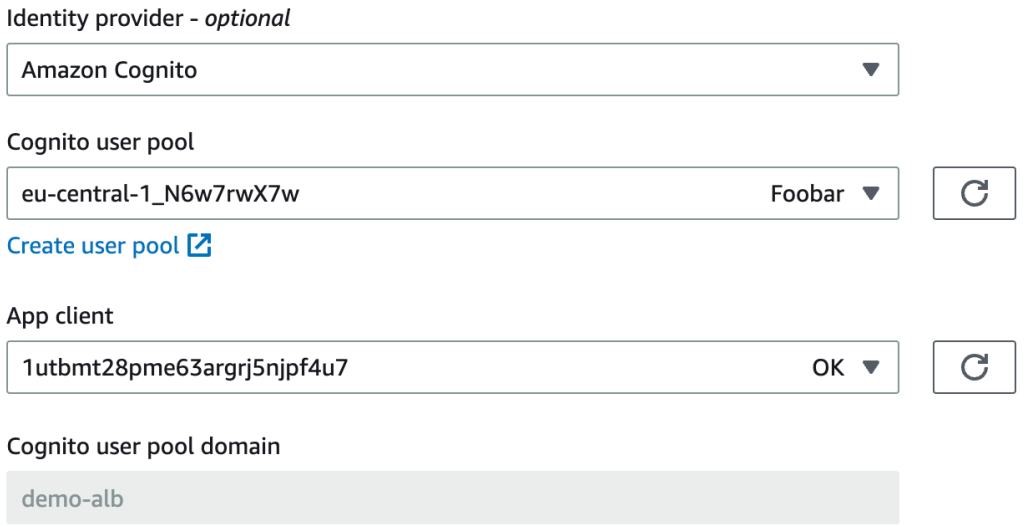
- Your Application Load Balancer can securely authenticate users
  - Offload the work of authenticating users to your load balancer
  - Your applications can focus on their business logic
- Authenticate users through:
  - Identity Provider (IdP): OpenID Connect (OIDC) compliant
  - Cognito User Pools:
    - Social IdPs, such as Amazon, Facebook, or Google
    - Corporate identities using SAML, LDAP, or Microsoft AD
- Must use an HTTPS listener to set authenticate-oidc & authenticate-cognito rules
- OnUnauthenticatedRequest – authenticate (default), deny, allow



Application Load Balancer – Cognito Auth.

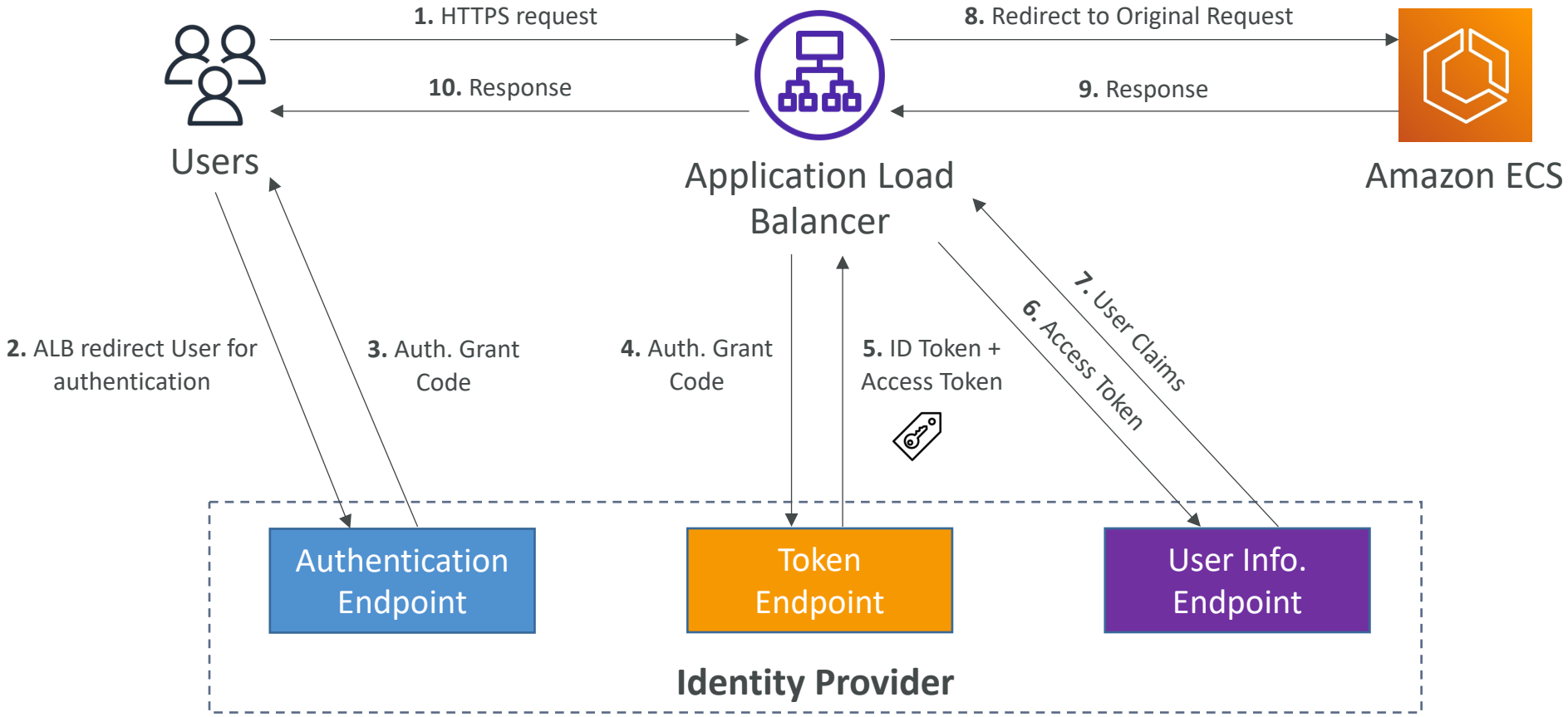


ALB – Auth through Cognito User Pools

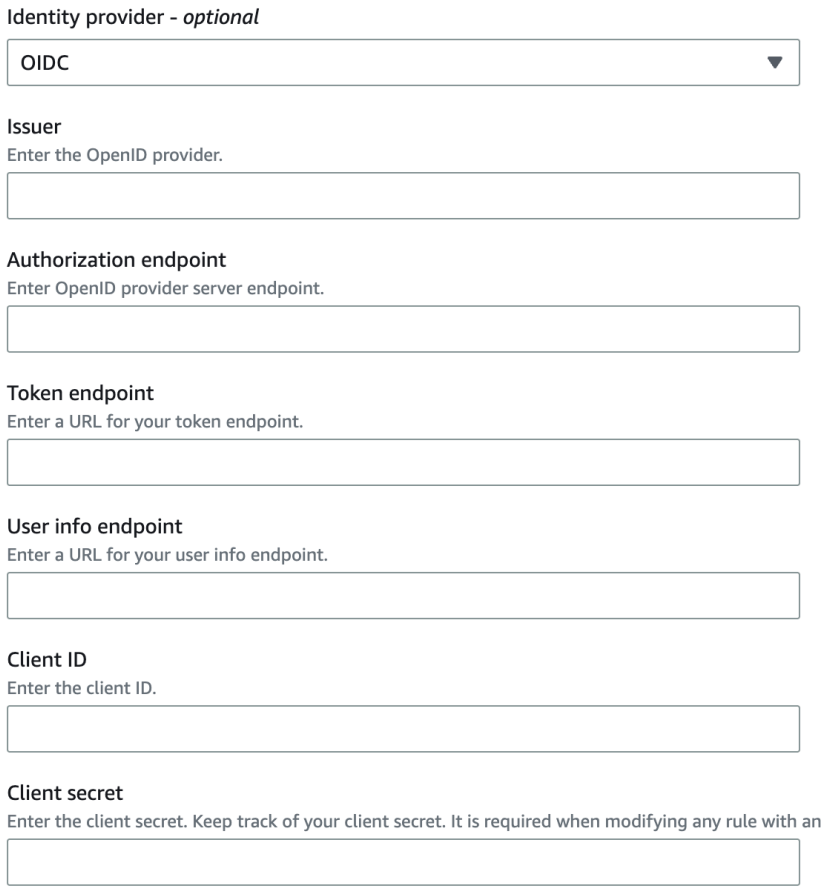


- Create Cognito User Pool, Client and Domain
- Make sure an ID token is returned
- Add the social or Corporate IdP if needed
- Several URL redirections are necessary
- Allow your Cognito User Pool Domain on your IdP app's callback URL. For example:
  - https://domain-prefix.auth.*region*.amazoncognito.com/saml2/idpresponse
  - https://user-pool-domain/oauth2/idpresponse

Application Load Balancer – OIDC Auth.



ALB – Auth. Through an Identity Provider (IdP) That is OpenID Connect (OIDC) Compliant

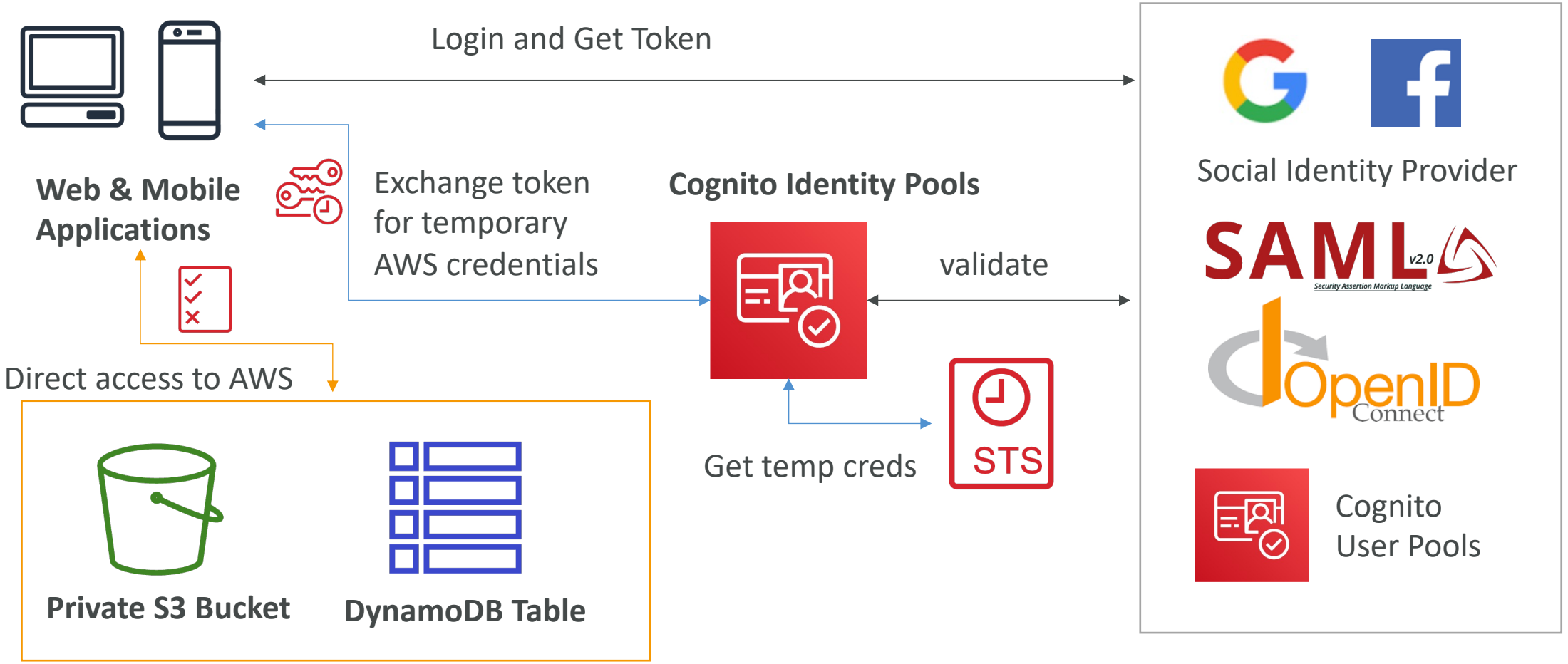


- Configure a Client ID & Client Secret
- Allow redirect from OIDC to your Application Load Balancer DNS name (AWS provided) and CNAME (DNS Alias of your app)
  - https://DNS/oauth2/idpresponse
  - https://CNAME/oauth2/idpresponse

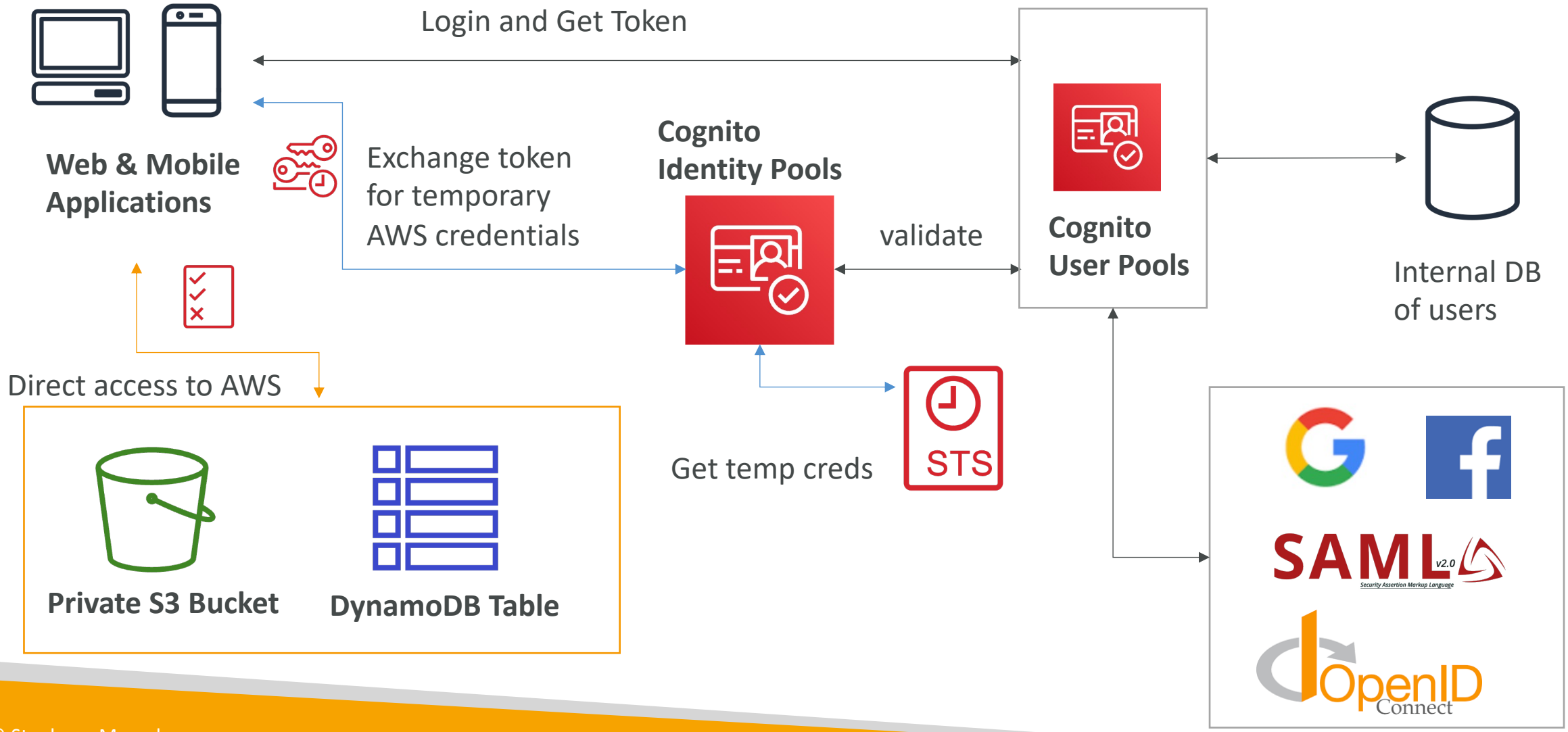
Cognito Identity Pools (Federated Identities)

- Get identities for "users" so they obtain temporary AWS credentials
- Your identity pool (e.g identity source) can include:
  - Public Providers (Login with Amazon, Facebook, Google, Apple)
  - Users in an Amazon Cognito user pool
  - OpenID Connect Providers & SAML Identity Providers
  - Developer Authenticated Identities (custom login server)
  - Cognito Identity Pools allow for unauthenticated (guest) access
- Users can then access AWS services directly or through API Gateway
  - The IAM policies applied to the credentials are defined in Cognito
  - They can be customized based on the user_id for fine grained control

Cognito Identity Pools – Diagram



Cognito Identity Pools – Diagram with CUP



Cognito Identity Pools – IAM Roles

- Default IAM roles for authenticated and guest users
- Define rules to choose the role for each user based on the user's ID
- You can partition your users' access using **policy variables**

- IAM credentials are obtained by Cognito Identity Pools through STS
- The roles must have a "trust" policy of Cognito Identity Pools

Cognito Identity Pools – Guest User example

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "s3:GetObject"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:s3:::mybucket/assets/my_picture.jpg"
            ]
        }
    ]
}
```

Cognito Identity Pools – Policy variable on S3

```
{
    "Version": "2012-10-17",
    "Statement": [
      {
        "Action": ["s3:ListBucket"],
        "Effect": "Allow",
        "Resource": ["arn:aws:s3:::mybucket"],
        "Condition": {"StringLike": {"s3:prefix": ["${cognito-identity.amazonaws.com:sub}/*"]}}
      },
      {
        "Action": [
          "s3:GetObject",
          "s3:PutObject"
        ],
        "Effect": "Allow",
        "Resource": ["arn:aws:s3:::mybucket/${cognito-identity.amazonaws.com:sub}/*"]
      }
    ]
}
```

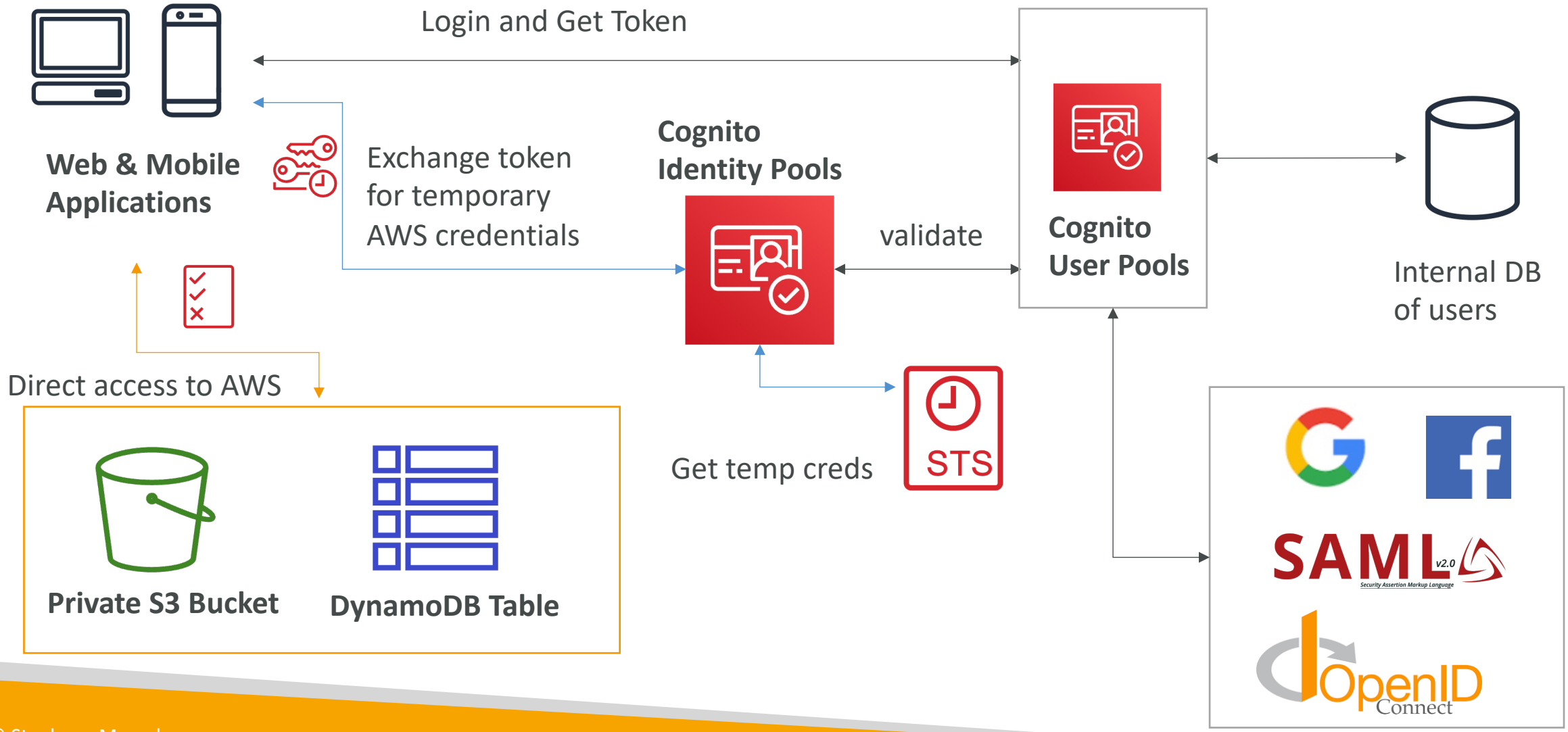
Cognito Identity Pools – DynamoDB

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem", "dynamodb:BatchGetItem", "dynamodb:Query",
                "dynamodb:PutItem", "dynamodb:UpdateItem", "dynamodb:DeleteItem",
                "dynamodb:BatchWriteItem"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-west-2:123456789012:table/MyTable"
            ],
            "Condition": {
                "ForAllValues:StringEquals": {
                    "dynamodb:LeadingKeys": [
                        "${cognito-identity.amazonaws.com:sub}"
                    ]
                }
            }
        }
    ]
}
```

Cognito User Pools vs Identity Pools

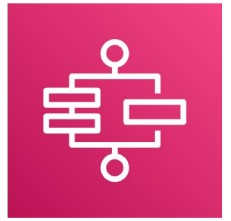
- Cognito User Pools (for authentication = identity verification)
  - Database of users for your web and mobile application
  - Allows to federate logins through Public Social, OIDC, SAML…
  - Can customize the hosted UI for authentication (including the logo)
  - Has triggers with AWS Lambda during the authentication flow
  - Adapt the sign-in experience to different risk levels (MFA, adaptive authentication, etc…)
- Cognito Identity Pools (for authorization = access control)
  - Obtain AWS credentials for your users
  - Users can login through Public Social, OIDC, SAML & Cognito User Pools
  - Users can be unauthenticated (guests)
  - Users are mapped to IAM roles & policies, can leverage policy variables
- CUP + CIP = authentication + authorization

Cognito Identity Pools – Diagram with CUP

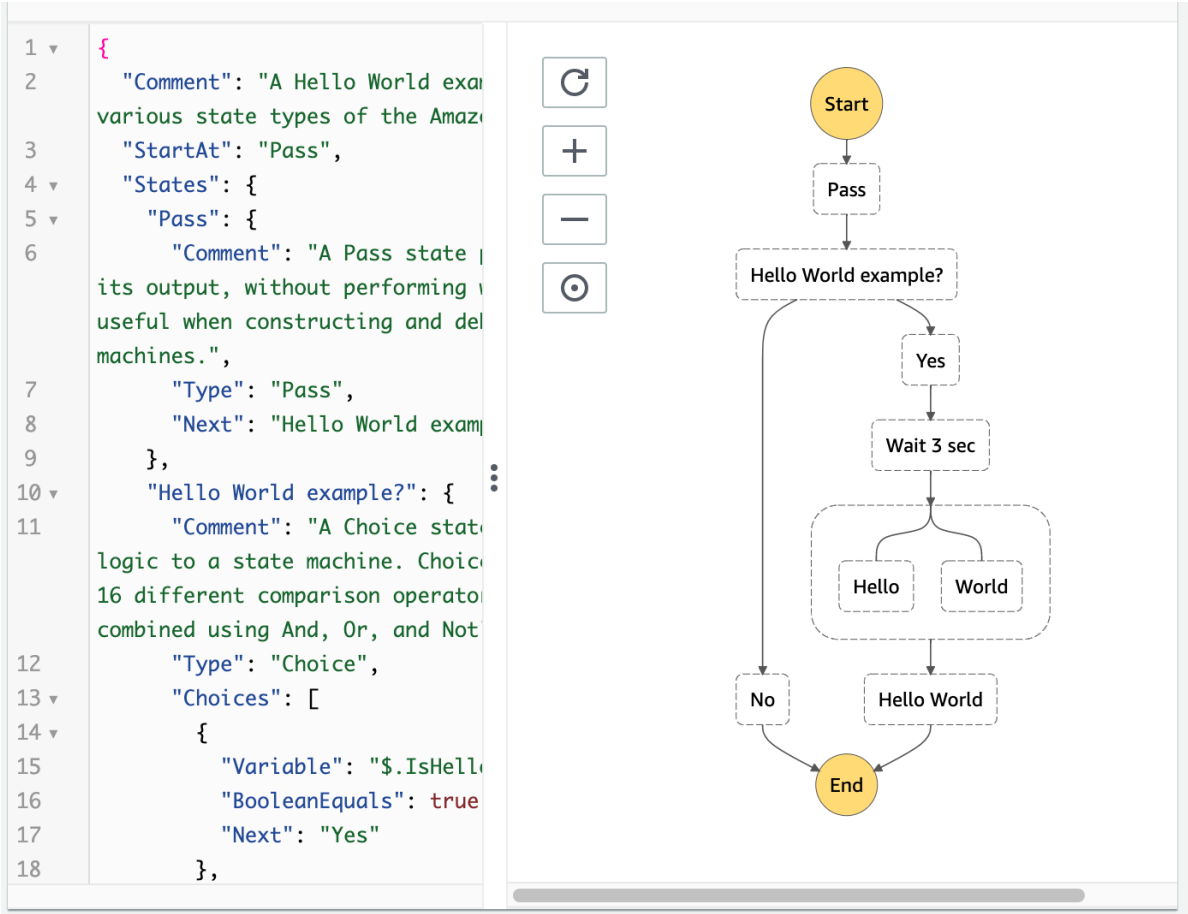


Other Serverless

AWS Step Functions

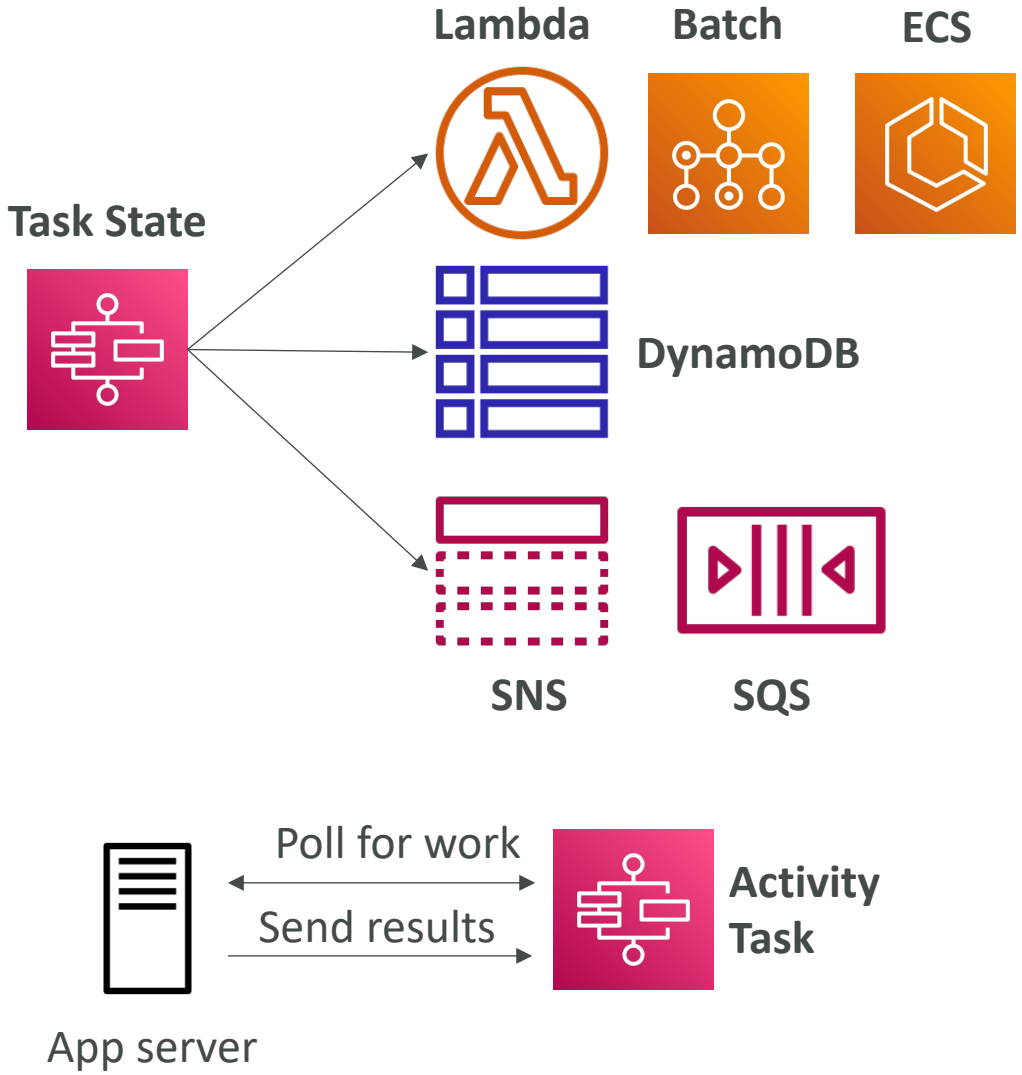


- Model your **workflows** as **state machines (one per workflow)**
  - Order fulfillment, Data processing
  - Web applications, Any workflow
- Written in JSON
- Visualization of the workflow and the execution of the workflow, as well as history
- Start workflow with SDK call, API Gateway, Event Bridge (CloudWatch Event)



Step Function – Task States

- Do some work in your state machine
- Invoke one AWS service
  - Can invoke a **Lambda function**
  - Run an AWS Batch job
  - Run an ECS task and wait for it to complete
  - Insert an item from DynamoDB
  - Publish message to SNS, SQS
  - Launch another Step Function workflow…
- Run an one Activity
  - EC2, Amazon ECS, on-premises
  - Activities poll the Step functions for work
  - Activities send results back to Step Functions



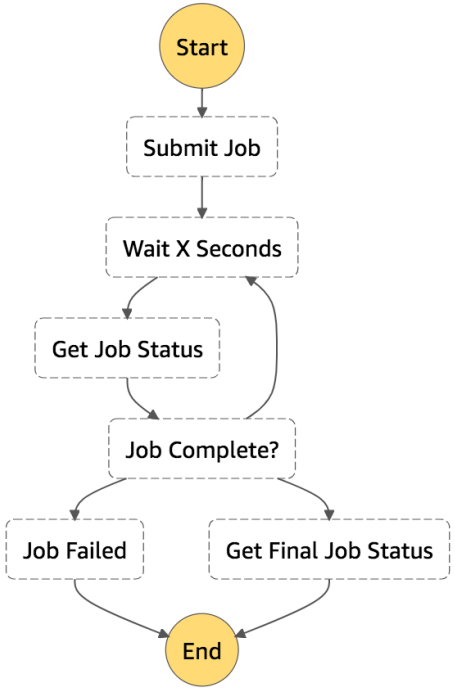
Example – Invoke Lambda Function

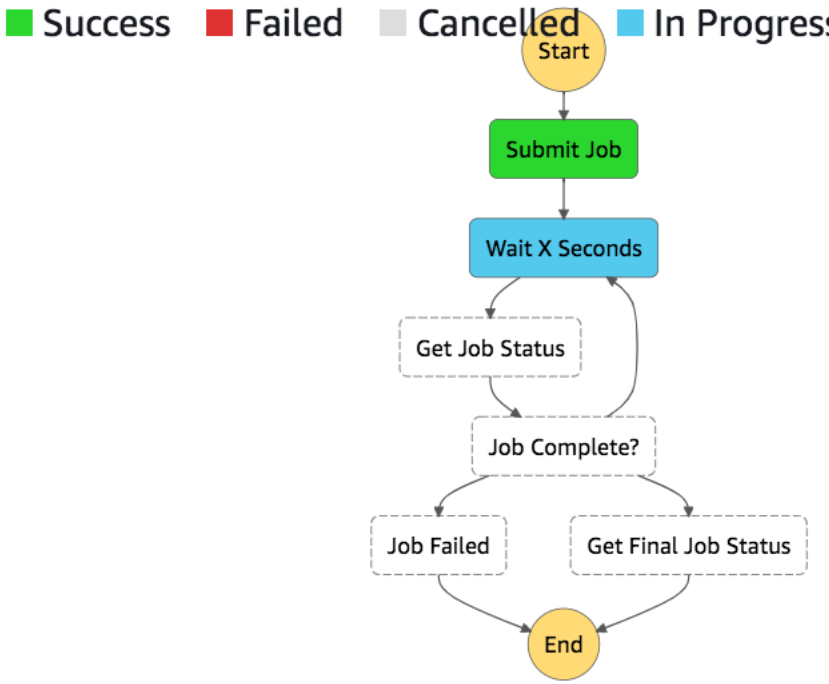
```
"Invoke Lambda function": {
    "Type": "Task",
    "Resource": "arn:aws:states:::lambda:invoke",
    "Parameters": {
        "FunctionName":
        "arn:aws:lambda:REGION:ACCOUNT_ID:function:FUNCTION_NAME",
        "Payload": {
            "Input.$": "$"
        }
    },
    "Next": "NEXT_STATE",
    "TimeoutSeconds": 300
}
```

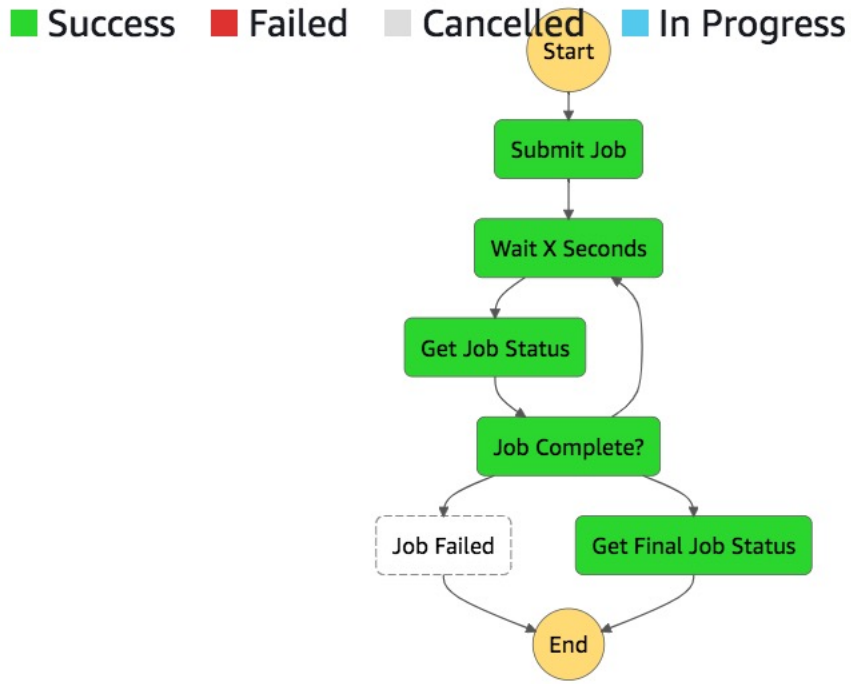
Step Function - States

- **Choice State** - Test for a condition to send to a branch (or default branch)
- **Fail or Succeed State** - Stop execution with failure or success
- **Pass State** - Simply pass its input to its output or inject some fixed data, without performing work.
- **Wait State** - Provide a delay for a certain amount of time or until a specified time/date.
- **Map State** - Dynamically iterate steps.'
- ***Parallel State*** *- Begin parallel branches of execution.*

Visual workflow in Step Functions







Error Handling in Step Functions

- Any state can encounter runtime errors for various reasons:
  - State machine definition issues (for example, no matching rule in a Choice state)
  - Task failures (for example, an exception in a Lambda function)
  - Transient issues (for example, network partition events)
- Use **Retry** (to retry failed state) and **Catch** (transition to failure path) in the State Machine to handle the errors instead of inside the Application Code
- Predefined error codes:
  - States.ALL : matches any error name
  - States.Timeout: Task ran longer than TimeoutSeconds or no heartbeat received
  - States.TaskFailed: execution failure
  - States.Permissions: insufficient privileges to execute code
- The state may report is own errors

Step Functions – Retry (Task or Parallel State)

```
"HelloWorld": {
  "Type": "Task",
  "Resource": "arn:aws:lambda:REGION:ACCOUNT_ID:function:FUNCTION_NAME",
  "Retry": [
    {
      "ErrorEquals": ["CustomError"],
      "IntervalSeconds": 1,
      "MaxAttempts": 2,
      "BackoffRate": 2.0
    },
    {
      "ErrorEquals": ["States.TaskFailed"],
      "IntervalSeconds": 30,
      "MaxAttempts": 2,
      "BackoffRate": 2.0
    },
    {
      "ErrorEquals": ["States.ALL"],
      "IntervalSeconds": 5,
      "MaxAttempts": 5,
      "BackoffRate": 2.0
    }
  ],
  "End": true
}
```

- Evaluated from top to bottom
- **ErrorEquals**: match a specific kind of error
- **IntervalSeconds**: initial delay before retrying
- **BackoffRate**: multiple the delay after each retry
- **MaxAttempts**: default to 3, set to 0 for never retried
- When max attempts are reached, the **Catch** kicks in

Step Functions – Catch (Task or Parallel State)

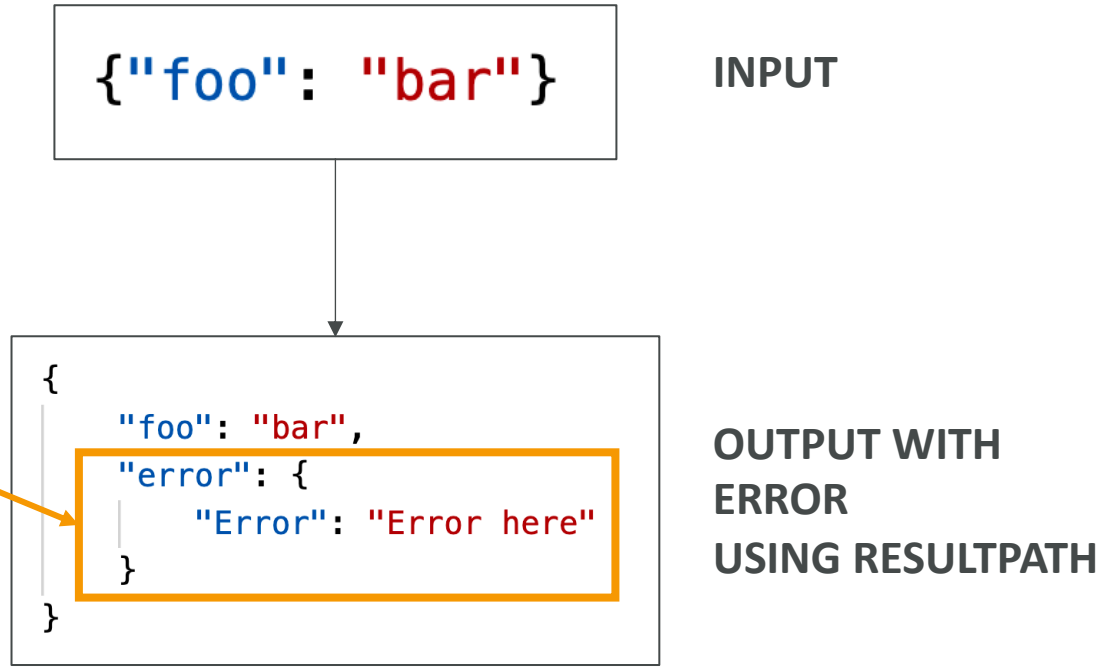
```
"HelloWorld": {
  "Type": "Task",
  "Resource": "arn:aws:lambda:....",
  "Catch": [
    {
      "ErrorEquals": ["CustomError"],
      "Next": "CustomErrorFallback"
    },
    {
      "ErrorEquals": ["States.TaskFailed"],
      "Next": "ReservedTypeFallback"
    },
    {
        "ErrorEquals": ["States.ALL"],
        "Next": "NextTask",
        "ResultPath": "$.error"
    }
  ],
  "End": true
},
"CustomErrorFallback": {
  "Type": "Pass",
  "Result": "This is a fallback from a custom lambda function exception"
  "End": true
},
```

- Evaluated from top to bottom
- **ErrorEquals**: match a specific kind of error
- **Next**: State to send to
- **ResultPath** - A path that determines what input is sent to the state specified in the Next field.

Step Function – ResultPath

```
"HelloWorld": {
  "Type": "Task",
  "Resource": "arn:aws:lambda:....",
  "Catch": [{
      "ErrorEquals": ["States.ALL"],
      "Next": "NextTask",
      "ResultPath": "$.error"
    }],
  "End": true
},
"NextTask": {
  "Type": "Pass",
  "Result": "This is a fallback from a reserved error code",
  "End": true
}
```

- Include the error in the input



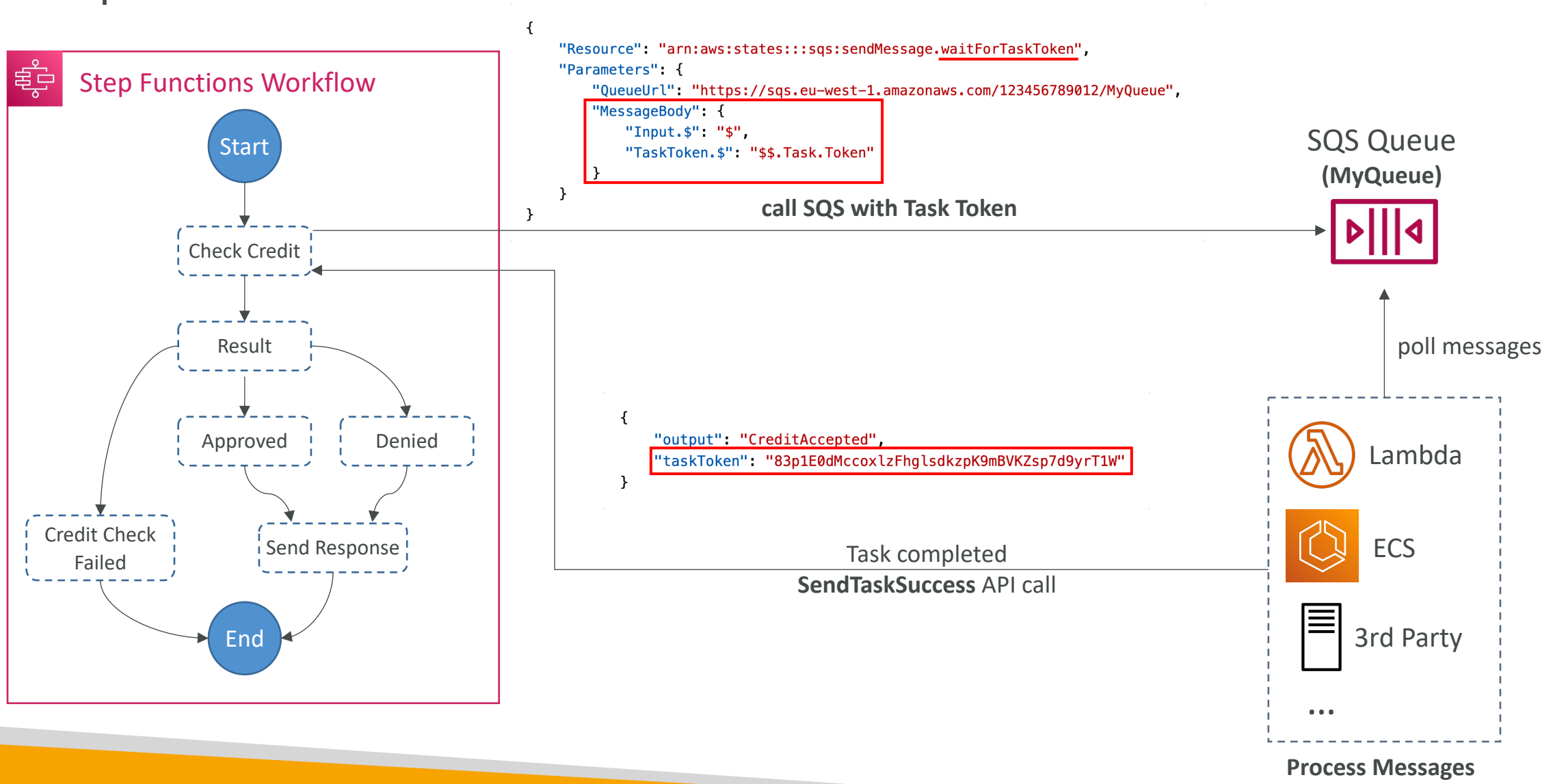
Step Functions – Wait for Task Token

- Allows you to pause Step Functions during a Task until a Task Token is returned
- Task might wait for other AWS services, human approval, 3rd party integration, call legacy systems…
- Append **.waitForTaskToken** to the **Resource** field to tell Step Functions to wait for the Task Token to be returned

```
"Resource": "arn:aws:states:::sqs:sendMessage.waitForTaskToken"
```

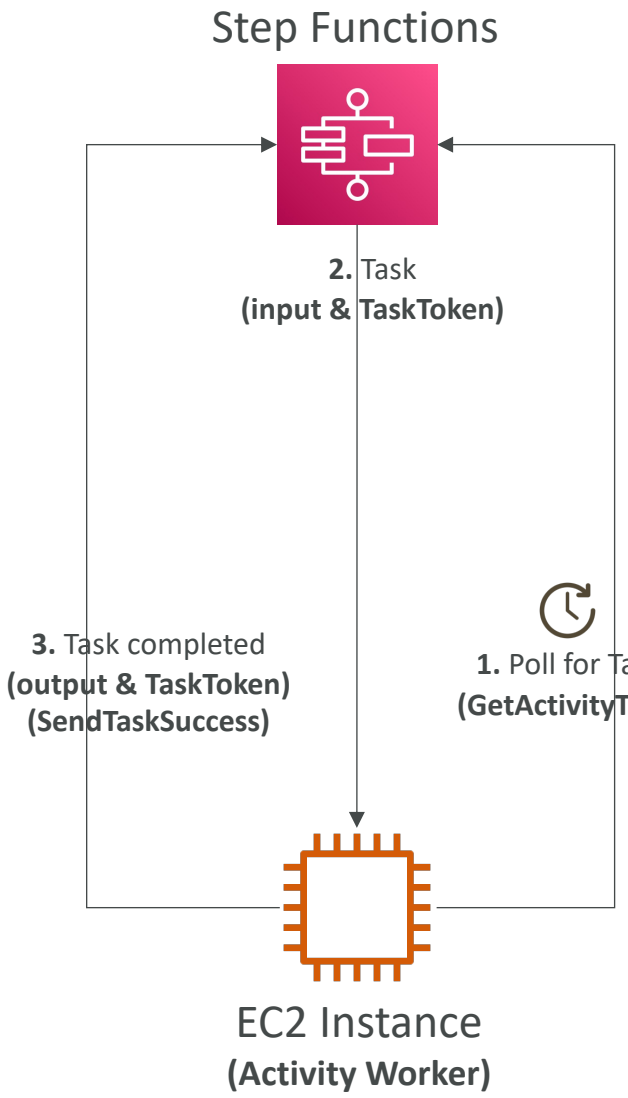
- Task will pause until it receives that Task Token back with a **SendTaskSuccess** or **SendTaskFailure** API call

Step Functions – Wait for Task Token

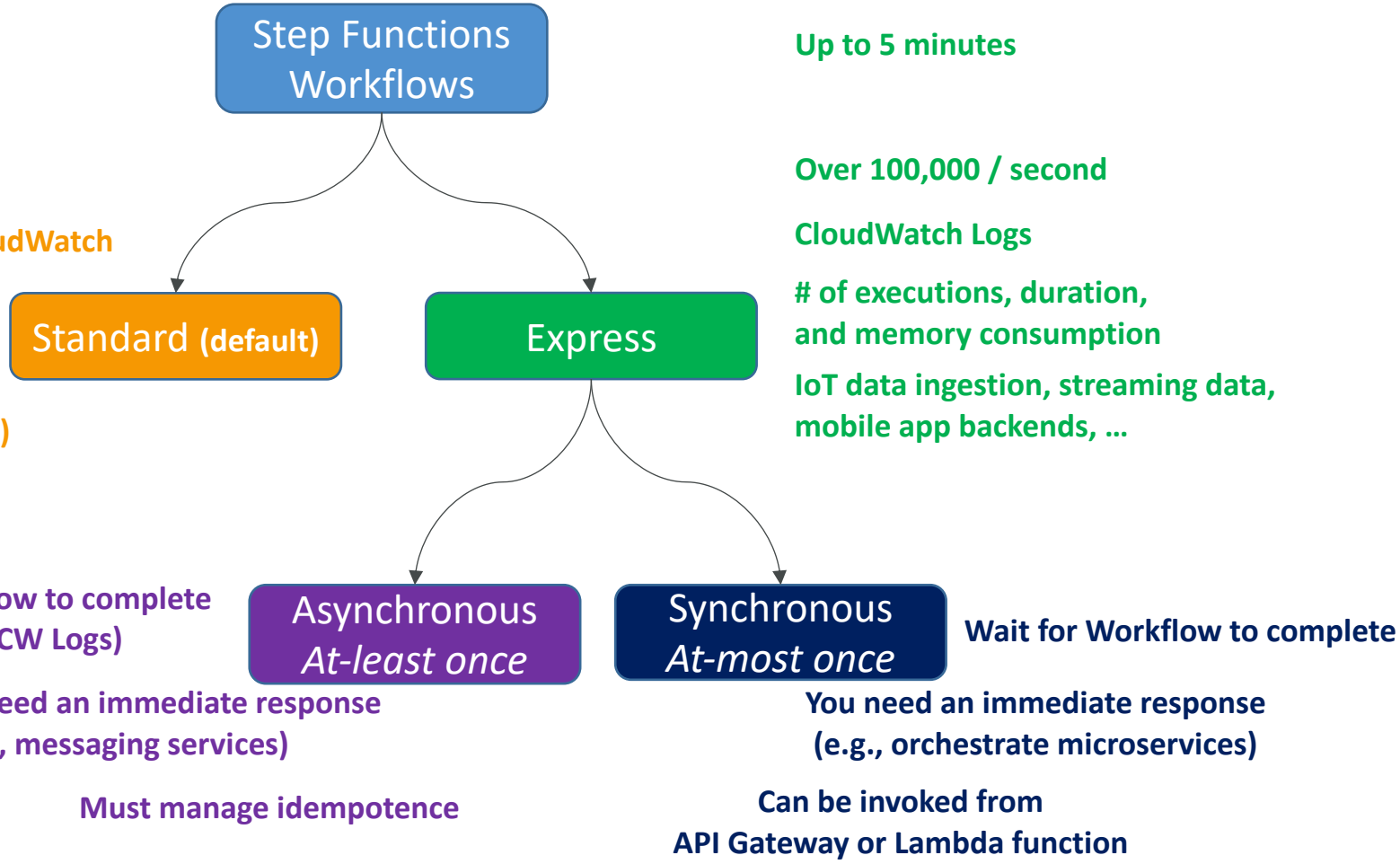


Step Functions – Activity Tasks

- Enables you to have the Task work performed by an *Activity Worker*
- **Activity Worker** apps can be running on EC2, Lambda, mobile device…
- Activity Worker poll for a Task using **GetActivityTask** API
- After Activity Worker completes its work, it sends a response of its success/failure using **SendTaskSuccess** or **SendTaskFailure**
- To keep the Task active:
  - Configure how long a task can wait by setting **TimeoutSeconds**
  - Periodically send a heartbeat from your Activity Worker using **SendTaskHeartBeat** within the time you set in **HeartBeatSeconds**
- By configuring a long **TimeoutSeconds** and actively sending a heartbeat, Activity Task can wait up to 1 year



Step Functions – Standard vs. Express



	Standard (default)	Express
Max. Duration	Up to 1 year	Up to 5 minutes
Execution Model	Exactly-once Execution	
Execution Rate	Over 2000 / second	Over 100,000 / second
Execution History	Up to 90 days or using CloudWatch	CloudWatch Logs
Pricing	# of State Transitions	# of executions, duration, and memory consumption
Use cases	Non-idempotent actions (e.g., processing Payments)	IoT data ingestion, streaming data, mobile app backends, …

	Asynchronous *At-least once*	Synchronous *At-most once*
	Doesn't wait for Workflow to complete (get results from CW Logs)	Wait for Workflow to complete
	You don't need an immediate response (e.g., messaging services)	You need an immediate response (e.g., orchestrate microservices)
	Must manage idempotence	Can be invoked from API Gateway or Lambda function

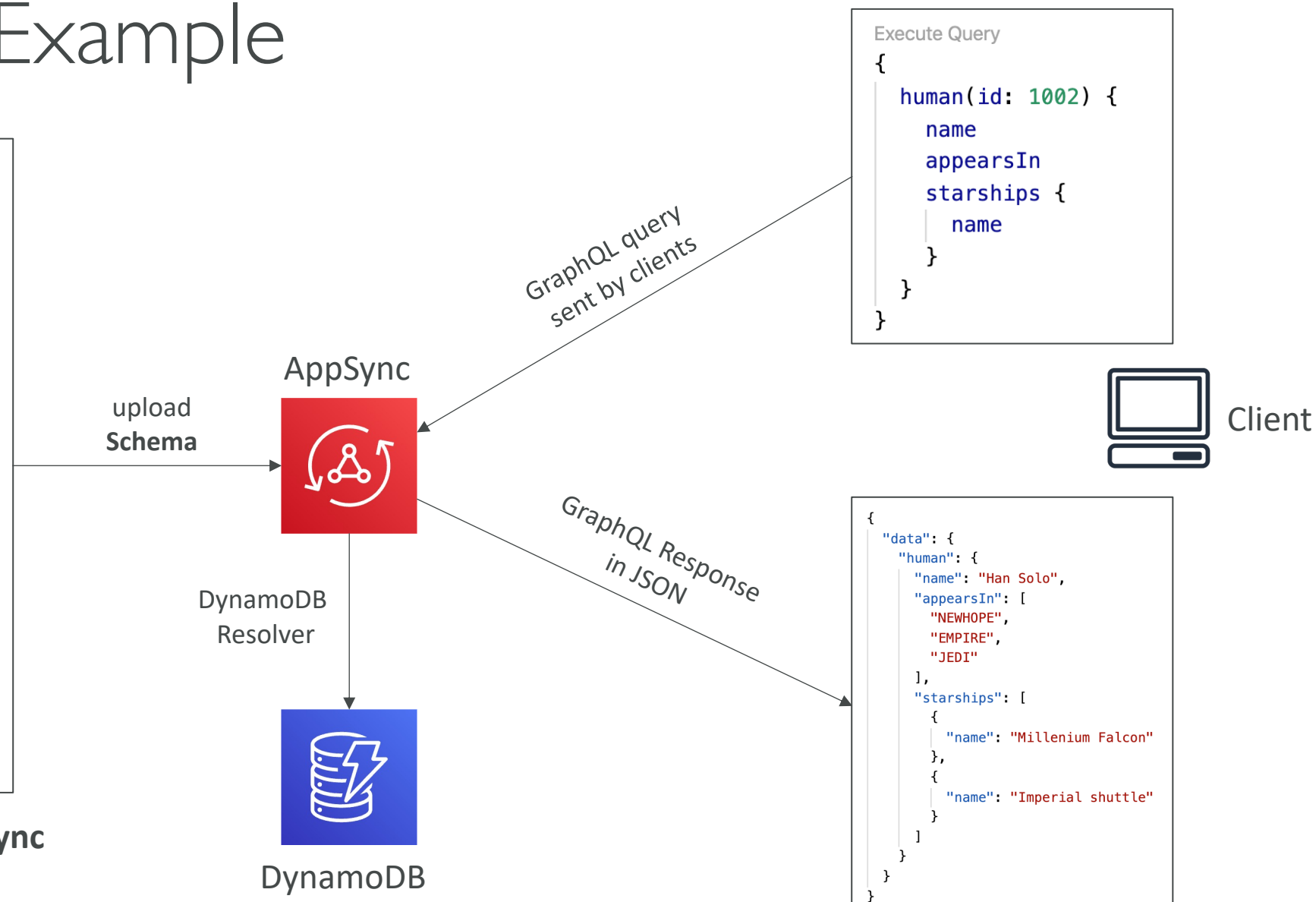
AWS AppSync - Overview

- **AppSync** is a managed service that uses **GraphQL**
- **GraphQL** makes it easy for applications to get exactly the data they need.
- This includes combining data from **one or more sources**
  - NoSQL data stores, Relational databases, HTTP APIs…
  - Integrates with DynamoDB, Aurora, OpenSearch & others
  - Custom sources with AWS Lambda
- Retrieve data in **real-time with WebSocket or MQTT on WebSocket**
- For mobile apps: local data access & data synchronization
- It all starts with uploading one **GraphQL schema**

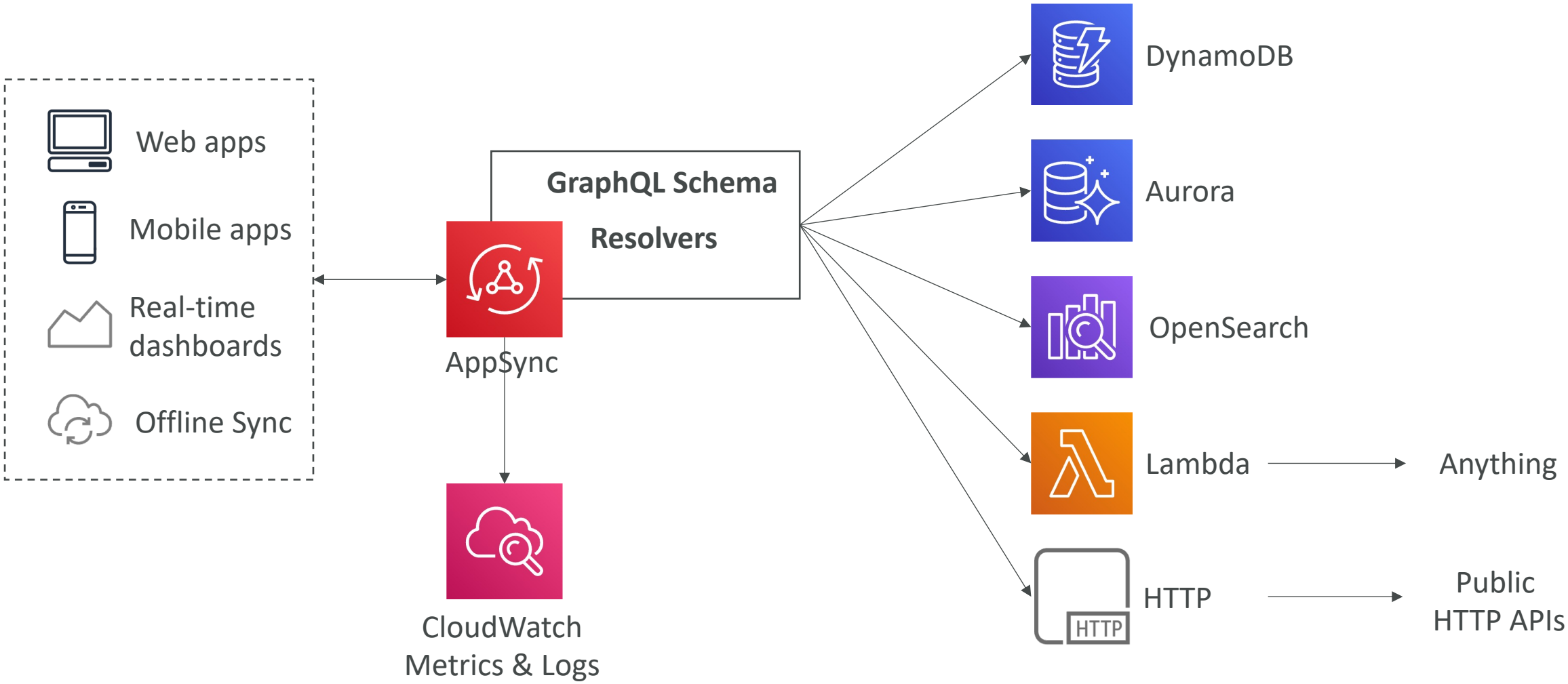
GraphQL Example

```
type Query {
  human(id: ID!): Human
}

type Human {
  name: String
  appearsIn: [Episode]
  starships: [Starship]
}

enum Episode {
  NEWHOPE
  EMPIRE
  JEDI
}

type Starship {
  name: String
}
```

GraphQL Schema on AppSync



AppSync Diagram



AppSync – Security

- There are four ways you can authorize applications to interact with your AWS AppSync GraphQL API:
- **API_KEY**
- **AWS_IAM:** IAM users / roles / cross-account access
- **OPENID_CONNECT:** OpenID Connect provider / JSON Web Token
- **AMAZON_COGNITO_USER_POOLS**

- For custom domain & HTTPS, use CloudFront in front of AppSync

AWS Amplify
Create mobile and web applications

Amplify Studio

Visually build a full-stack app,
both front-end UI and a backend.

Amplify CLI

Configure an Amplify backend
With a guided CLI workflow

Amplify Libraries

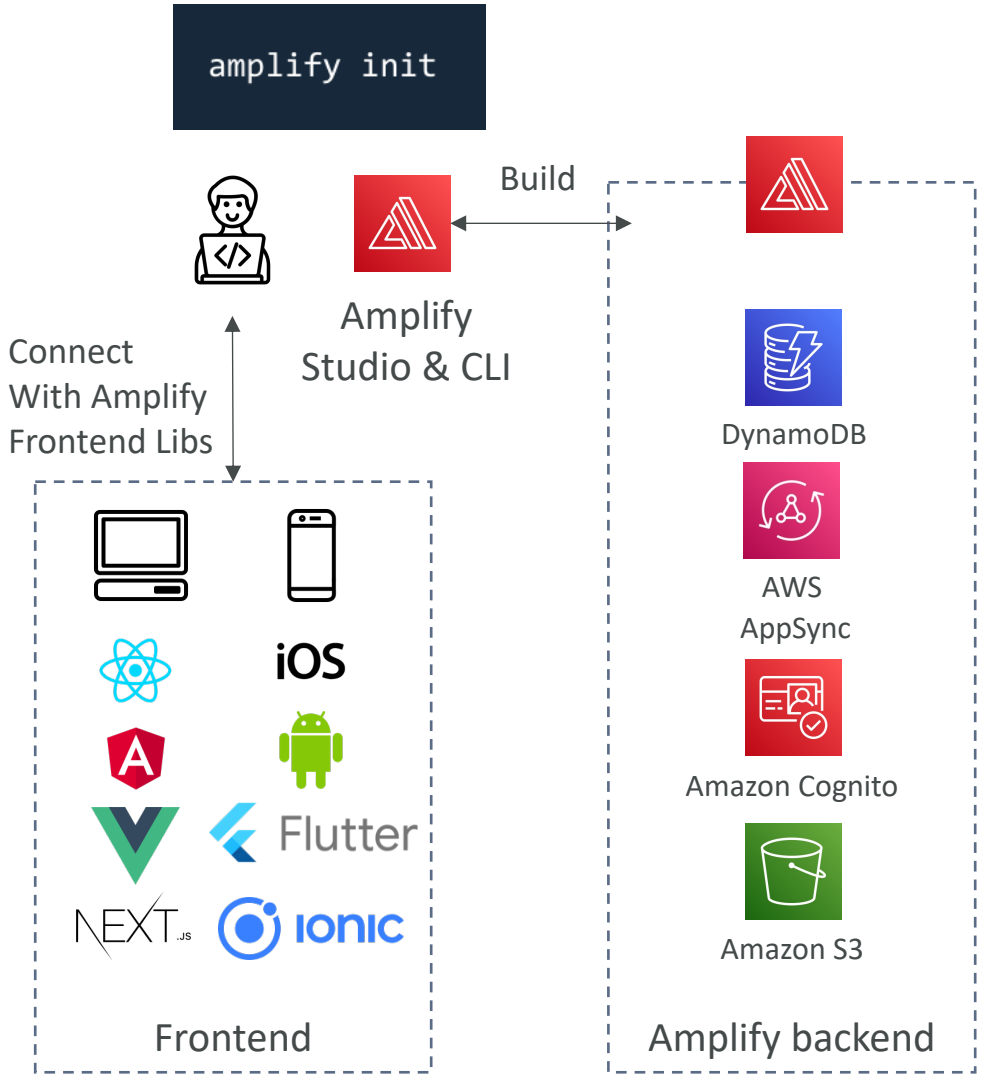
Connect your app to existing AWS
Services (Cognito, S3 and more)

Amplify Hosting

Host secure, reliable, fast web apps or websites
via the AWS content delivery network.

AWS Amplify

- Set of tools to get started with creating **mobile and web applications**
- "Elastic Beanstalk for mobile and web applications"
- Must-have features such as **data storage, authentication, storage, and machine-learning**, all powered by AWS services
- **Front-end libraries** with ready-to-use components for React.js, Vue, Javascript, iOS, Android, Flutter, etc…
- Incorporates AWS best practices to for reliability, security, scalability
- Build and deploy with the **Amplify CLI** or **Amplify Studio**



AWS Amplify – Important Features

```
amplify add auth
```

AUTHENTICATION

- Leverages Amazon Cognito
- User registration, authentication, account recovery & other operations
- Support MFA, Social Sign-in, etc…
- Pre-built UI components
- Fine-grained authorization

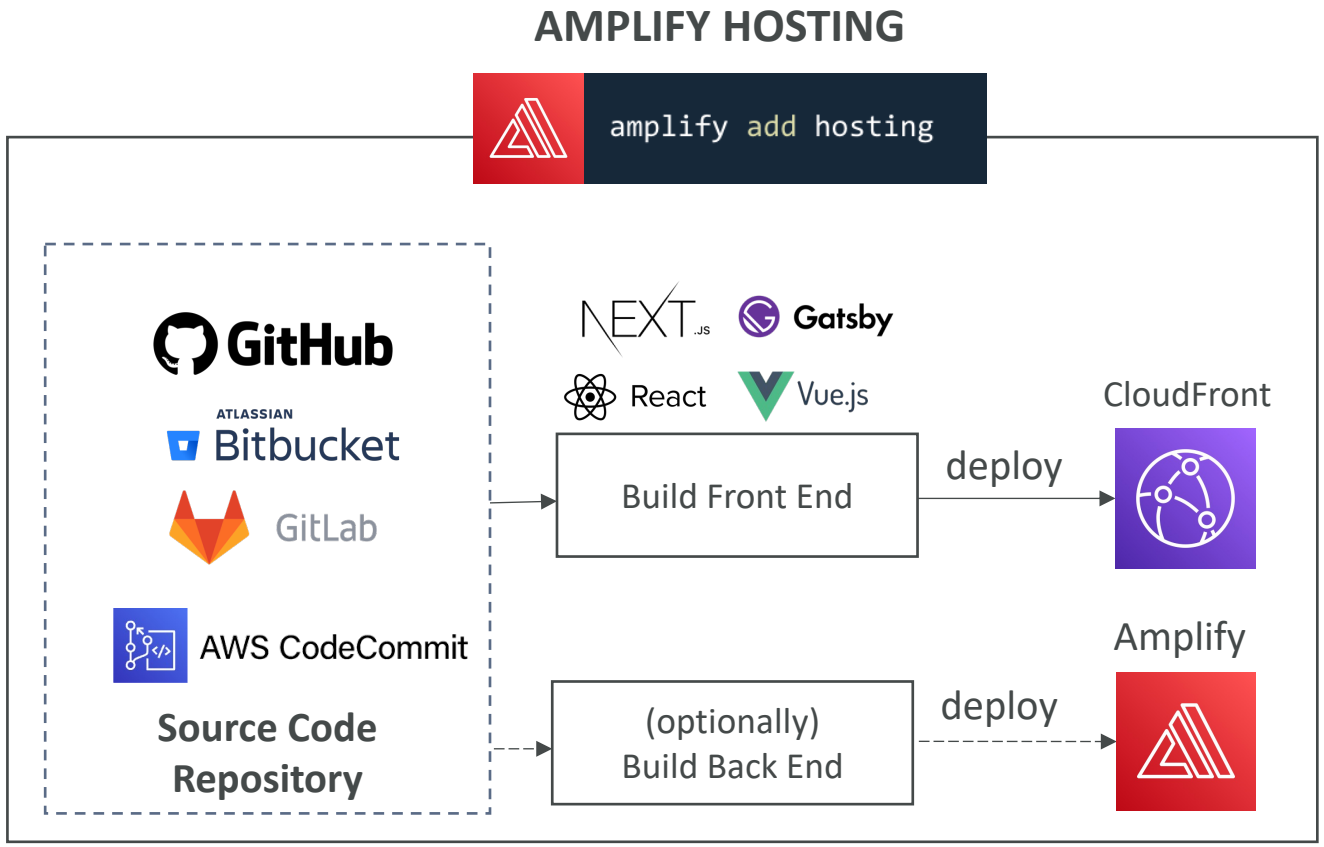
```
amplify add api
```

DATASTORE

- Leverages Amazon AppSync and Amazon DynamoDB
- Work with local data and have **automatic synchronization to the cloud** without complex code
- Powered by GraphQL
- Offline and real-time capabilities
- Visual data modeling w/ Amplify Studio

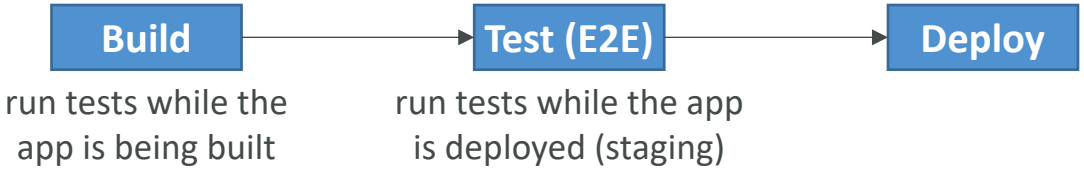
AWS Amplify Hosting

- Build and Host Modern Web Apps
- CICD (build, test, deploy)
- Pull Request Previews
- Custom Domains
- Monitoring
- Redirect and Custom Headers
- Password protection



AWS Amplify – End-to-End (E2E) Testing

- Run end-to-end (E2E) tests in the **test phase** in Amplify
- Catch regressions before pushing code to production
- Use the test step to run any test commands at build time **(amplify.yml)**
- **Integrated with Cypress testing framework**
  - Allows you to generate UI report for your tests




```yaml
test:
  phases:
    preTest:
      commands:
        - npm ci
        - npm install -g pm2
        - npm install -g wait-on
        - npm install mocha mochawesome ...
        - pm2 start npm -- start
        - wait-on http://localhost:3000
    test:
      commands:
        - 'npx cypress run --reporter ...'
    postTest:
      commands:
        - npx mochawesome-merge cypress/...
        - pm2 kill

  artifacts:
    baseDirectory: cypress
    configFilePath: "**/mochawesome.json"
    files:
      - "**/*.png"
      - "**/*.mp4"
```

amplify.yml

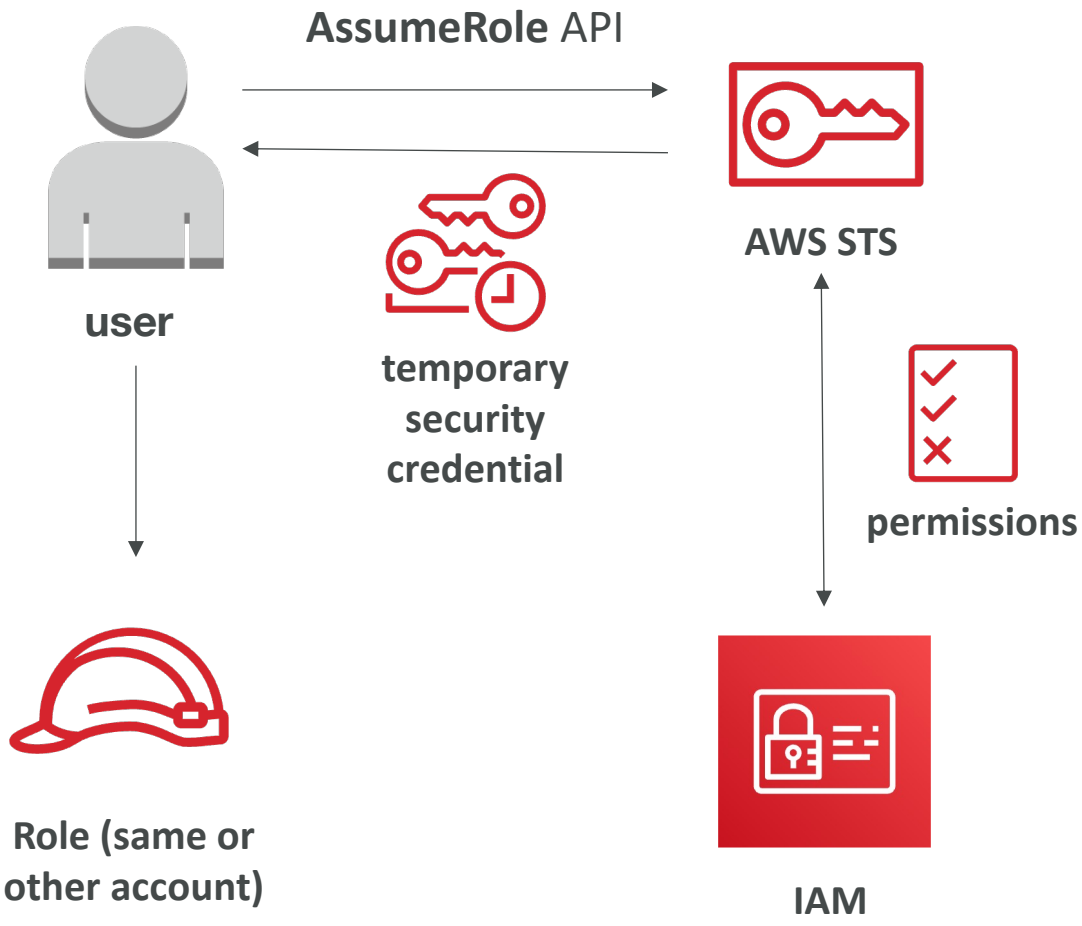
Advanced Identity in AWS

AWS STS – Security Token Service

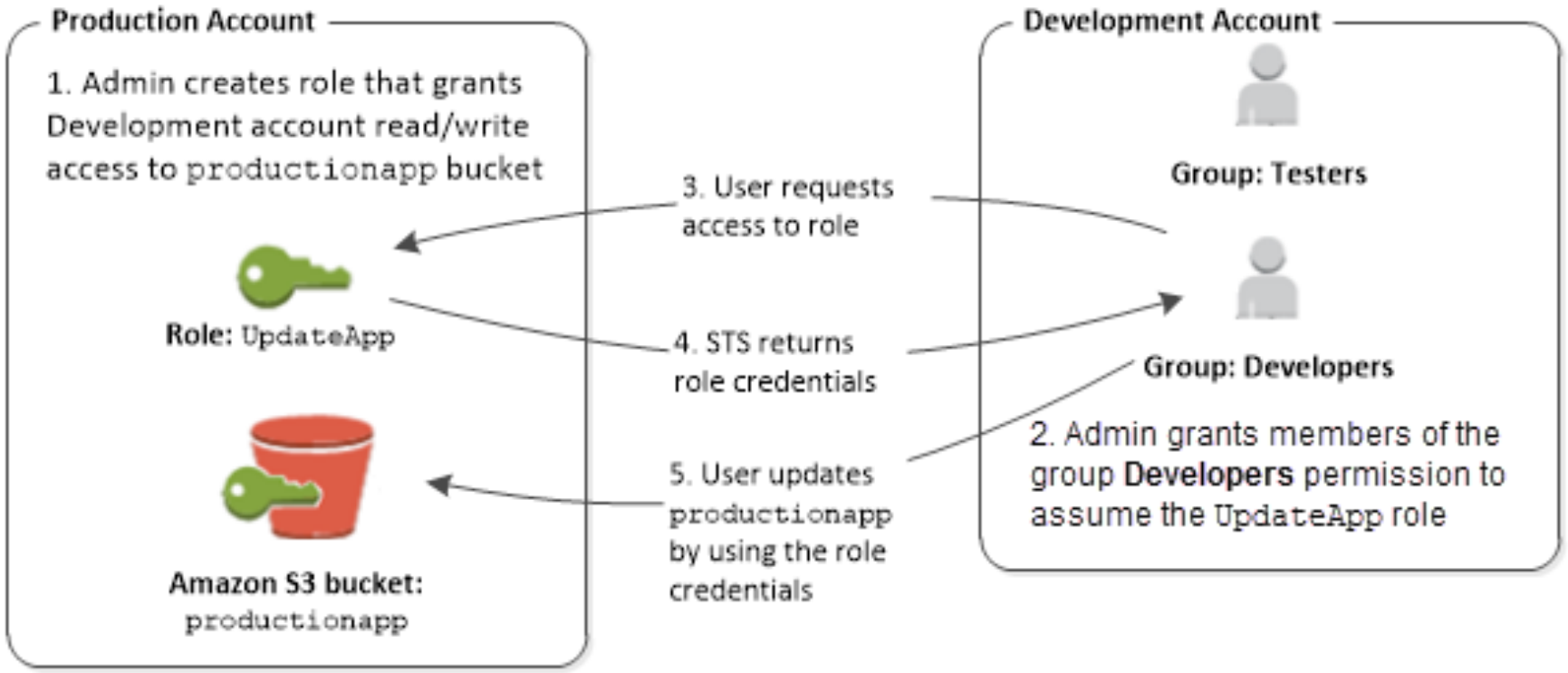
- Allows to grant limited and temporary access to AWS resources (up to 1 hour).
- **AssumeRole:** Assume roles within your account or cross account
- **AssumeRoleWithSAML:** return credentials for users logged with SAML
- **AssumeRoleWithWebIdentity**
  - return creds for users logged with an IdP (Facebook Login, Google Login, OIDC compatible…)
  - AWS recommends against using this, and using **Cognito Identity Pools** instead
- **GetSessionToken:** for MFA, from a user or AWS account root user
- **GetFederationToken**: obtain temporary creds for a federated user
- **GetCallerIdentity**: return details about the IAM user or role used in the API call
- **DecodeAuthorizationMessage**: decode error message when an AWS API is denied

Using STS to Assume a Role

- Define an IAM Role within your account or cross-account
- Define which principals can access this IAM Role
- Use AWS STS (Security Token Service) to retrieve credentials and impersonate the IAM Role you have access to (**AssumeRole API**)
- Temporary credentials can be valid between 15 minutes to 1 hour



Cross account access with STS



https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_common-scenarios_aws-accounts.html

STS with MFA

- Use **GetSessionToken** from STS
- Appropriate IAM policy using IAM Conditions
- **aws:MultiFactorAuthPresent:true**
- Reminder, GetSessionToken returns:
  - Access ID
  - Secret Key
  - Session Token
  - Expiration date

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:StopInstances",
                "ec2:TerminateInstances"
            ],
            "Resource": [
                "*"
            ],
            "Condition": {
                "Bool": {
                    "aws:MultiFactorAuthPresent": "true"
                }
            }
        }
    ]
}
```

IAM Best Practices – General

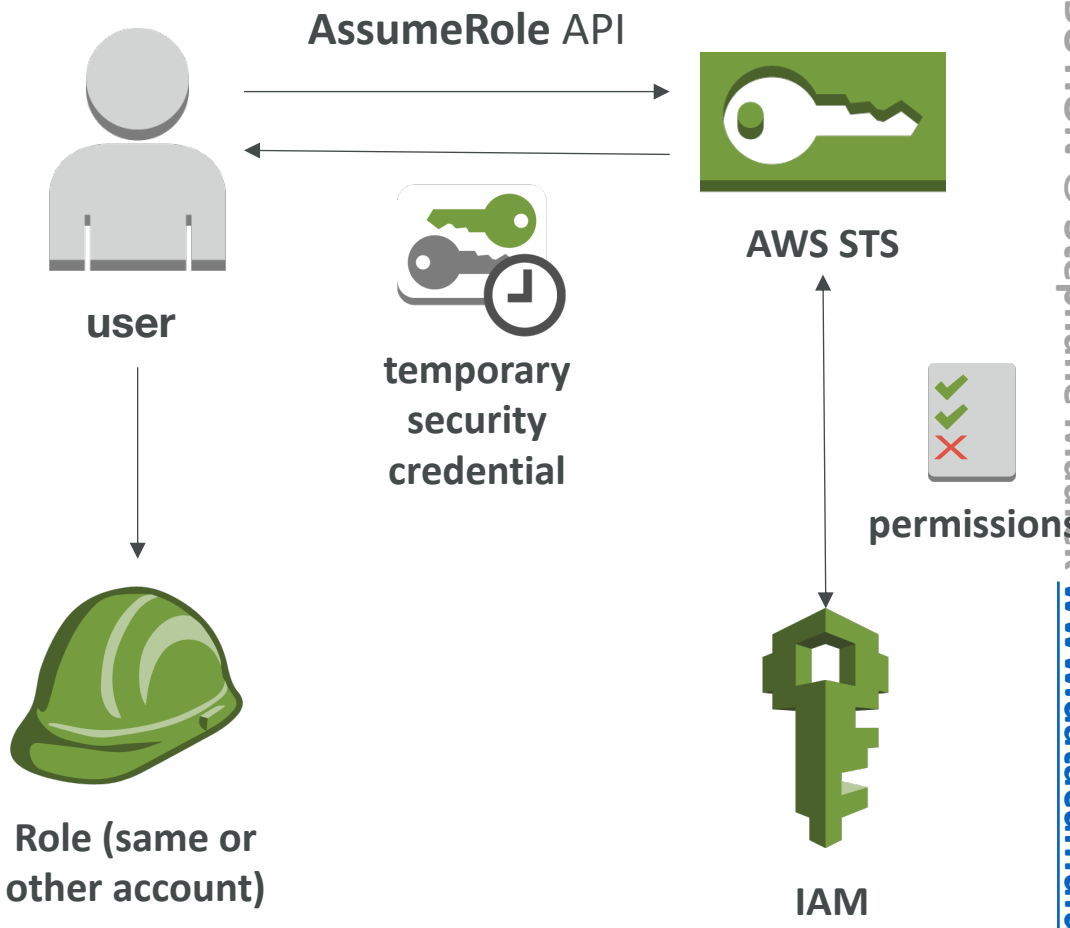
- Never use Root Credentials, enable MFA for Root Account
- Grant Least Privilege
  - Each Group / User / Role should only have the minimum level of permission it needs
  - Never grant a policy with "*" access to a service
  - Monitor API calls made by a user in CloudTrail (especially Denied ones)
- Never ever ever store IAM key credentials on any machine but a personal computer or on-premise server
- On premise server best practice is to call STS to obtain temporary security credentials

IAM Best Practices – IAM Roles

- EC2 machines should have their own roles
- Lambda functions should have their own roles
- ECS Tasks should have their own roles (ECS_ENABLE_TASK_IAM_ROLE=true)
- CodeBuild should have its own service role
- Create a least-privileged role for any service that requires it
- Create a role per application / lambda function (do not reuse roles)

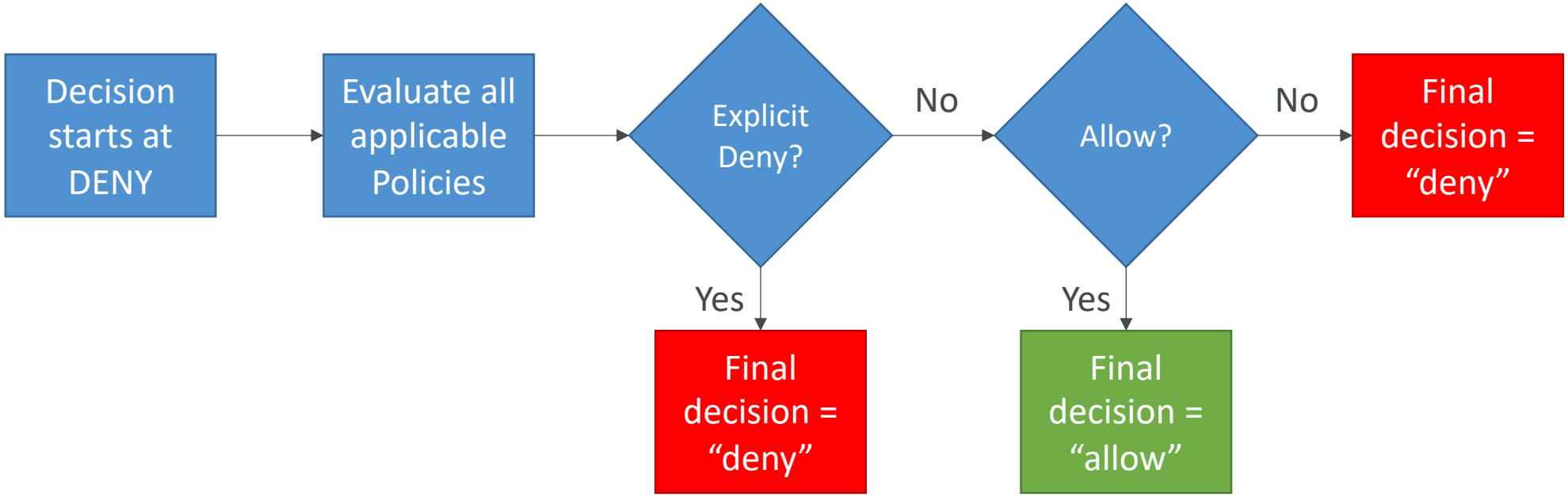
IAM Best Practices – Cross Account Access

- Define an IAM Role for another account to access
- Define which accounts can access this IAM Role
- Use AWS STS (Security Token Service) to retrieve credentials and impersonate the IAM Role you have access to (**AssumeRole API**)
- Temporary credentials can be valid between 15 minutes to 1 hour



Advanced IAM - Authorization Model Evaluation of Policies, simplified

1. If there's an explicit DENY, end decision and DENY
2. If there's an ALLOW, end decision with ALLOW
3. Else DENY



IAM Policies & S3 Bucket Policies

- IAM Policies are attached to users, roles, groups
- S3 Bucket Policies are attached to buckets
- When evaluating if an IAM Principal can perform an operation X on a bucket, the union of its assigned IAM Policies and S3 Bucket Policies will be evaluated.

Example 1

- IAM Role attached to EC2 instance, authorizes RW to "my_bucket"
- No S3 Bucket Policy attached

- => EC2 instance can read and write to "my_bucket"

Example 2

- IAM Role attached to EC2 instance, authorizes RW to "my_bucket"
- S3 Bucket Policy attached, explicit deny to the IAM Role

- => EC2 instance **cannot** read and write to "my_bucket"

Example 3

- IAM Role attached to EC2 instance, no S3 bucket permissions
- S3 Bucket Policy attached, explicit RW allow to the IAM Role

- => EC2 instance **can** read and write to "my_bucket"

Example 4

- IAM Role attached to EC2 instance, explicit deny S3 bucket permissions
- S3 Bucket Policy attached, explicit RW allow to the IAM Role

- => EC2 instance **cannot** read and write to "my_bucket"

Dynamic Policies with IAM

- How do you assign each user a /home/<user> folder in an S3 bucket?
- Option 1:
  - Create an IAM policy allowing georges to have access to /home/georges
  - Create an IAM policy allowing sarah to have access to /home/sarah
  - Create an IAM policy allowing matt to have access to /home/matt
  - … One policy per user!
  - This doesn't scale
- Option 2:
  - Create one dynamic policy with IAM
  - Leverage the special policy variable ${aws:username}

Dynamic Policy example

```
{
    "Sid": "AllowAllS3ActionsInUserFolder",
    "Action":["s3:*"],
    "Effect":"Allow",
    "Resource": ["arn:aws:s3:::my-company/home/${aws:username}/*"]
}
```

Inline vs Managed Policies

- AWS Managed Policy
  - Maintained by AWS
  - Good for power users and administrators
  - Updated in case of new services / new APIs
- Customer Managed Policy
  - Best Practice, re-usable, can be applied to many principals
  - Version Controlled + rollback, central change management
- Inline
  - Strict one-to-one relationship between policy and principal
  - Policy is deleted if you delete the IAM principal

Granting a User Permissions to Pass a Role to an AWS Service

- To configure many AWS services, you must **pass** an IAM role to the service (this happens only once during setup)
- The service will later assume the role and perform actions
- Example of passing a role:
  - To an EC2 instance
  - To a Lambda function
  - To an ECS task
  - To CodePipeline to allow it to invoke other services

- For this, you need the IAM permission **iam:PassRole**
- It often comes with iam:GetRole to view the role being passed

IAM PassRole example

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::123456789012:role/S3Access"
        }
    ]
}
```

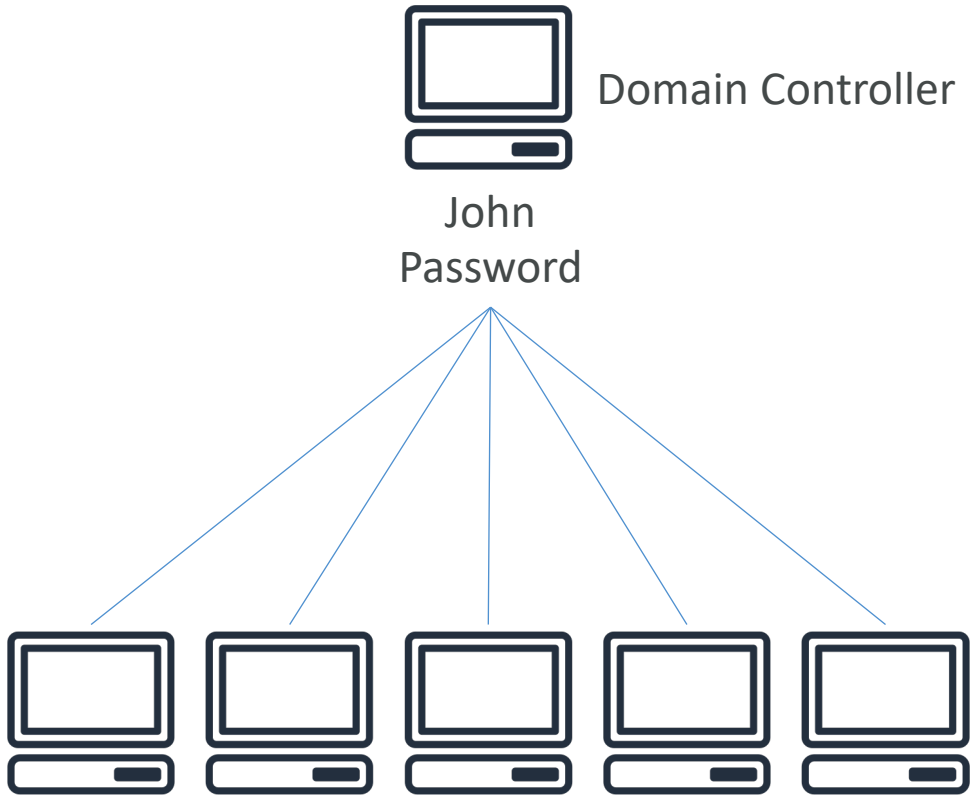
Can a role be passed to any service?

- No: Roles can only be passed to what their trust allows
- A *trust policy* for the role that allows the service to assume the role

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Sid": "TrustPolicyStatementThatAllowsEC2ServiceToAssumeTheAttachedRole",
        "Effect": "Allow",
        "Principal": {
            "Service": "ec2.amazonaws.com"
        },
        "Action": "sts:AssumeRole"
    }
}
```

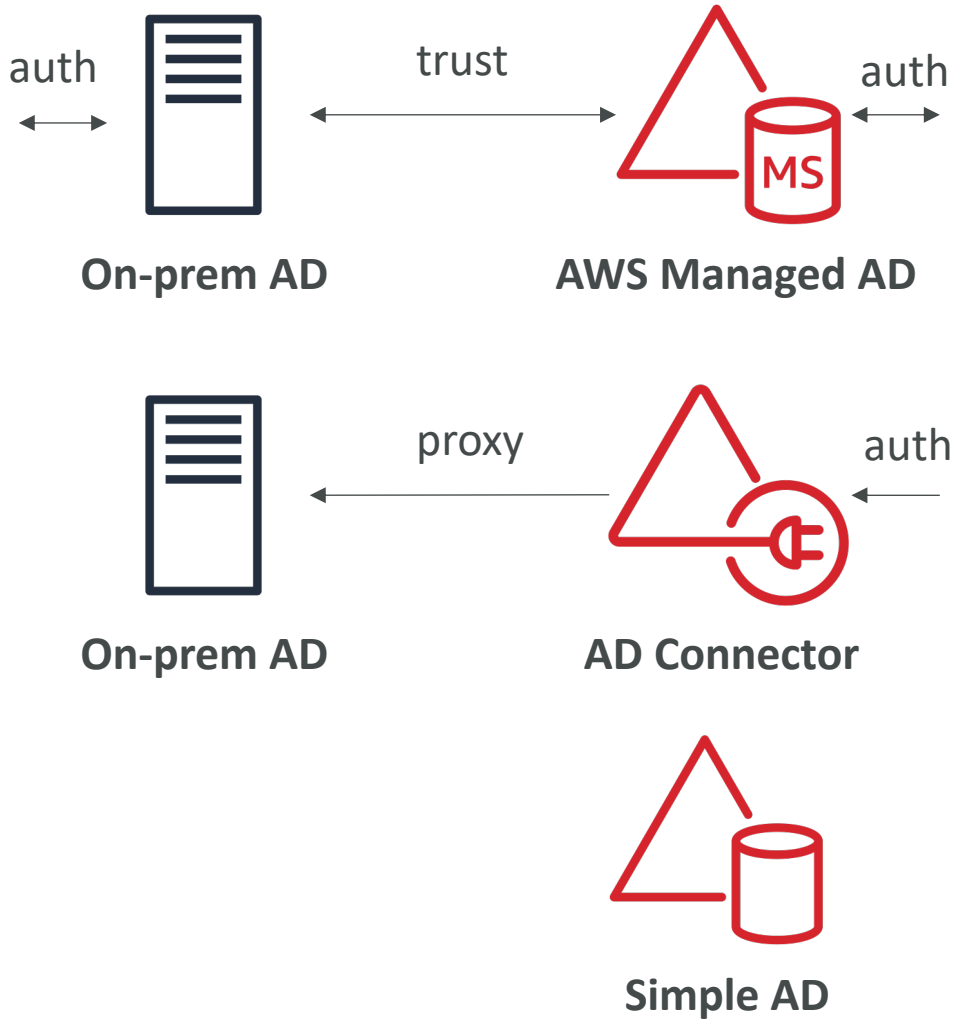
What is Microsoft Active Directory (AD)?

- Found on any Windows Server with AD Domain Services
- Database of **objects**: User Accounts, Computers, Printers, File Shares, Security Groups
- Centralized security management, create account, assign permissions
- Objects are organized in **trees**
- A group of trees is a **forest**



AWS Directory Services

- AWS Managed Microsoft AD
  - Create your own AD in AWS, manage users locally, supports MFA
  - Establish "trust" connections with your on-premise AD
- AD Connector
  - Directory Gateway (proxy) to redirect to on-premise AD, supports MFA
  - Users are managed on the on-premise AD
- Simple AD
  - AD-compatible managed directory on AWS
  - Cannot be joined with on-premise AD

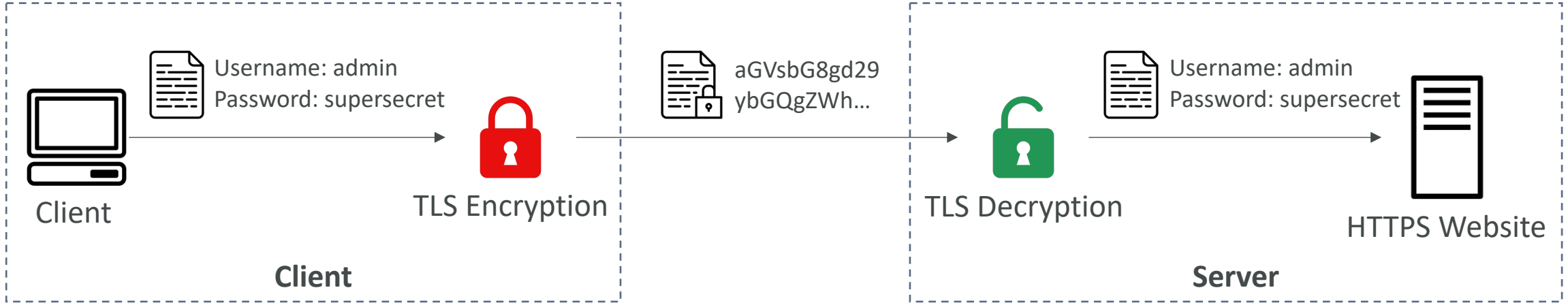


AWS Security & Encryption

KMS, Encryption SDK, SSM Parameter Store

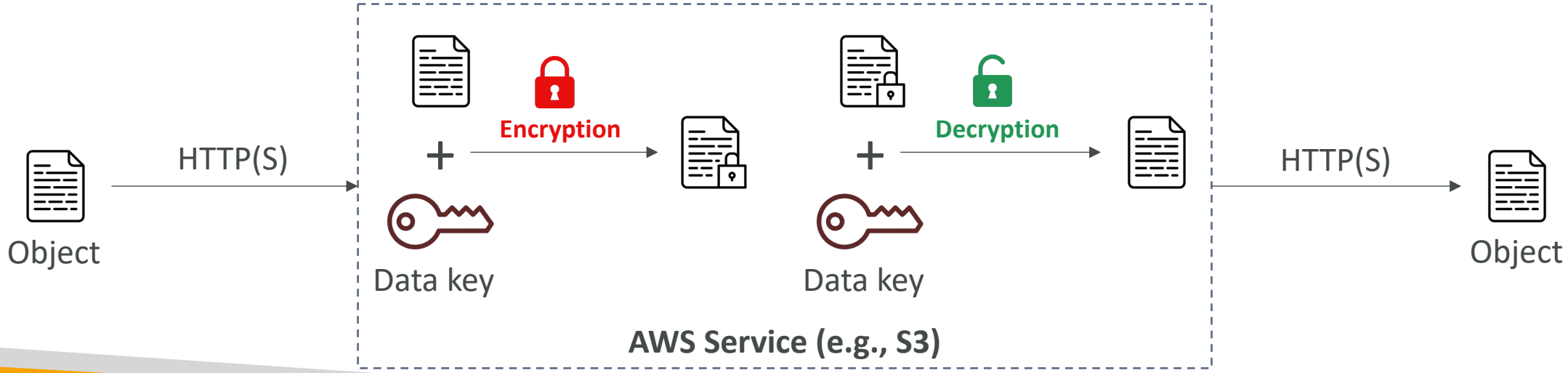
Why encryption?
Encryption in flight (TLS / SSL)

- Data is encrypted before sending and decrypted after receiving
- TLS certificates help with encryption (HTTPS)
- Encryption in flight ensures no MITM (man in the middle attack) can happen



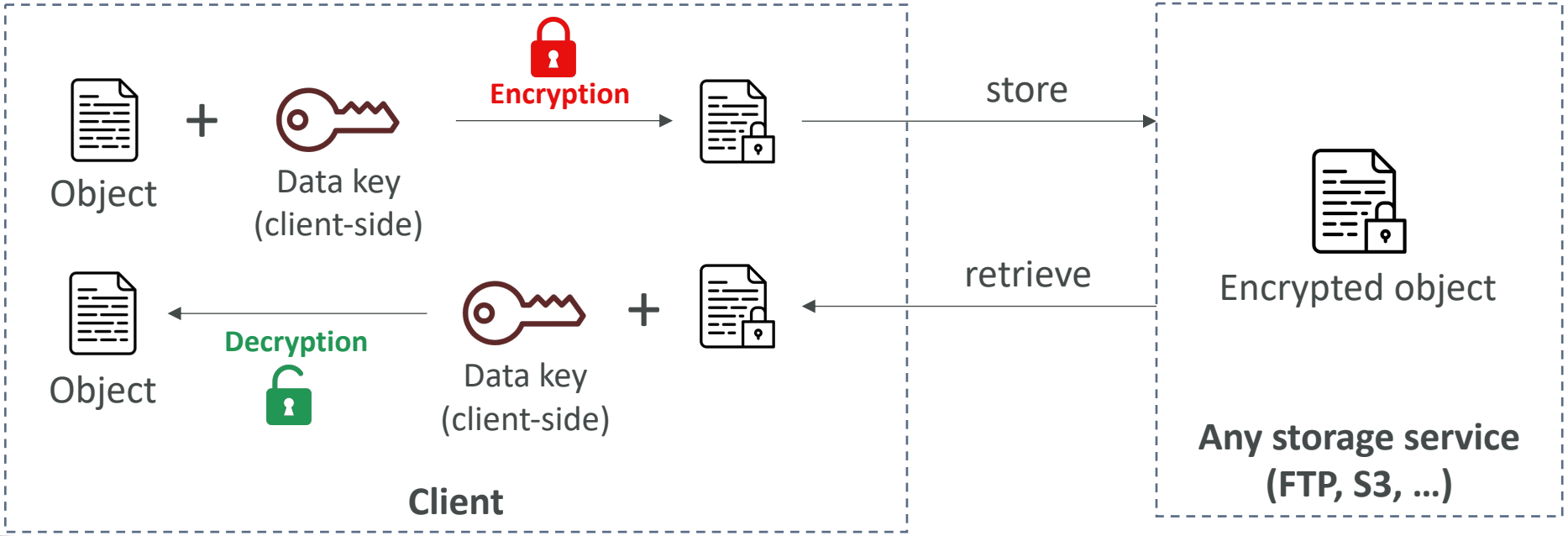
Why encryption?
Server-side encryption at rest

- Data is encrypted after being received by the server
- Data is decrypted before being sent
- It is stored in an encrypted form thanks to a key (usually a data key)
- The encryption / decryption keys must be managed somewhere, and the server must have access to it



Why encryption?
Client-side encryption

- Data is encrypted by the client and never decrypted by the server
- Data will be decrypted by a receiving client
- The server should not be able to decrypt the data
- Could leverage Envelope Encryption



AWS KMS (Key Management Service)

- Anytime you hear "encryption" for an AWS service, it's most likely KMS
- AWS manages encryption keys for us
- Fully integrated with IAM for authorization
- Easy way to control access to your data
- Able to audit KMS Key usage using CloudTrail
- Seamlessly integrated into most AWS services (EBS, S3, RDS, SSM…)
- **Never ever store your secrets in plaintext, especially in your code!**
  - KMS Key Encryption also available through API calls (SDK, CLI)
  - Encrypted secrets can be stored in the code / environment variables

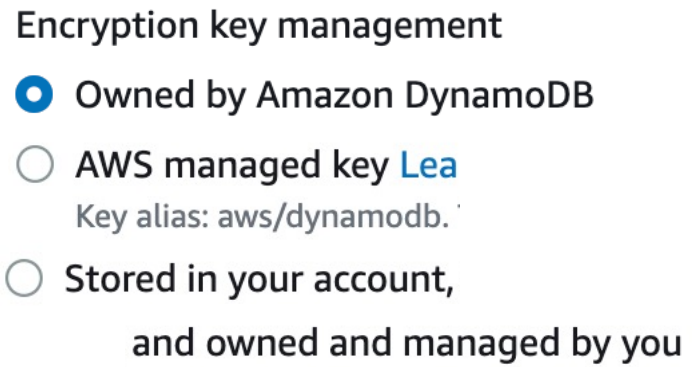
KMS Keys Types

- KMS Keys is the new name of KMS Customer *Master* Key
- Symmetric (AES-256 keys)
  - Single encryption key that is used to Encrypt and Decrypt
  - AWS services that are integrated with KMS use Symmetric CMKs
  - You never get access to the KMS Key unencrypted (must call KMS API to use)
- Asymmetric (RSA & ECC key pairs)
  - Public (Encrypt) and Private Key (Decrypt) pair
  - Used for Encrypt/Decrypt, or Sign/Verify operations
  - The public key is downloadable, but you can't access the Private Key unencrypted
  - Use case: encryption outside of AWS by users who can't call the KMS API

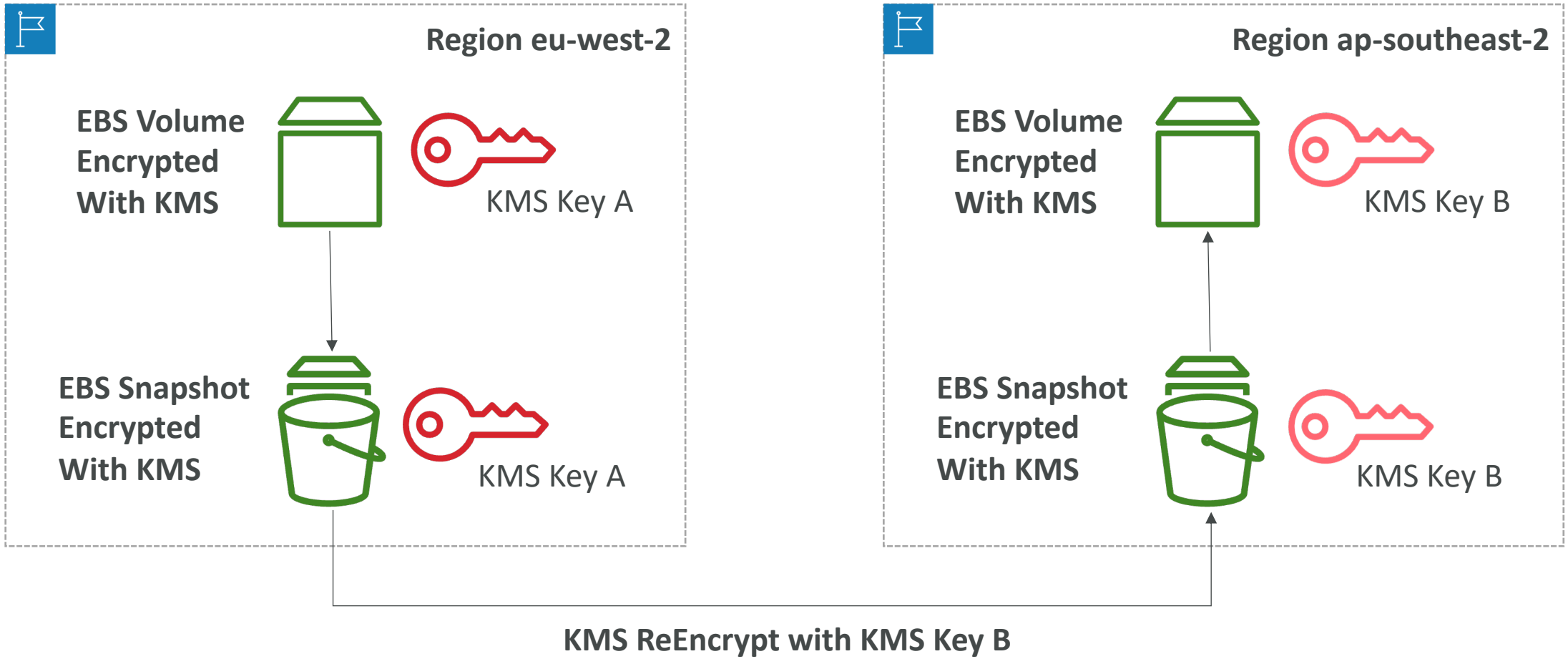
AWS KMS (Key Management Service)

- Types of KMS Keys:
  - AWS Owned Keys (free): SSE-S3, SSE-SQS, SSE-DDB (default key)
  - AWS Managed Key: **free** (aws/*service-name*, example: aws/rds or aws/ebs)
  - Customer managed keys created in KMS: **$1 / month**
  - Customer managed keys imported: **$1 / month**
  - + pay for API call to KMS ($0.03 / 10000 calls)

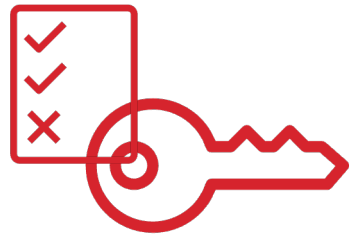
- Automatic Key rotation:
  - AWS-managed KMS Key: automatic every 1 year
  - Customer-managed KMS Key: (must be enabled) automatic & on-demand
  - Imported KMS Key: only manual rotation possible using alias



Copying Snapshots across regions



KMS Key Policies



- Control access to KMS keys, "similar" to S3 bucket policies
- Difference: you cannot control access without them

- Default KMS Key Policy:
  - Created if you don't provide a specific KMS Key Policy
  - Complete access to the key to the root user = entire AWS account
- Custom KMS Key Policy:
  - Define users, roles that can access the KMS key
  - Define who can administer the key
  - Useful for cross-account access of your KMS key

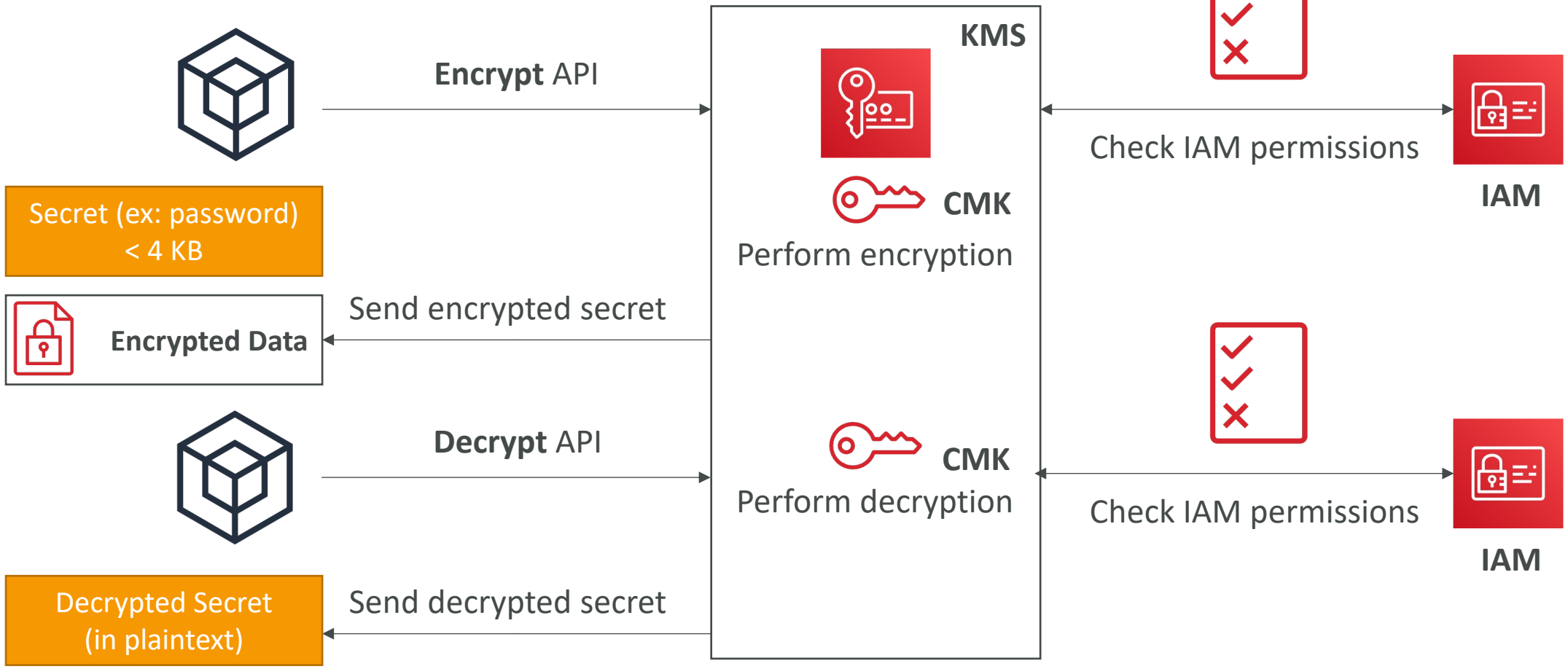
Copying Snapshots across accounts

1. Create a Snapshot, encrypted with your own KMS Key (Customer Managed Key)
2. **Attach a KMS Key Policy to authorize cross-account access**
3. Share the encrypted snapshot
4. (in target) Create a copy of the Snapshot, encrypt it with a CMK in your account
5. Create a volume from the snapshot

```
{
    "Sid": "Allow use of the key with destination account",
    "Effect": "Allow",
    "Principal": {
      "AWS": "arn:aws:iam::TARGET-ACCOUNT-ID:role/ROLENAME"
    },
    "Action": [
      "kms:Decrypt",
      "kms:CreateGrant"
    ],
    "Resource": "*",
    "Condition": {
      "StringEquals": {
        "kms:ViaService": "ec2.REGION.amazonaws.com",
        "kms:CallerAccount": "TARGET-ACCOUNT-ID"
      }
    }
}
```

KMS Key Policy

How does KMS work?
API – Encrypt and Decrypt

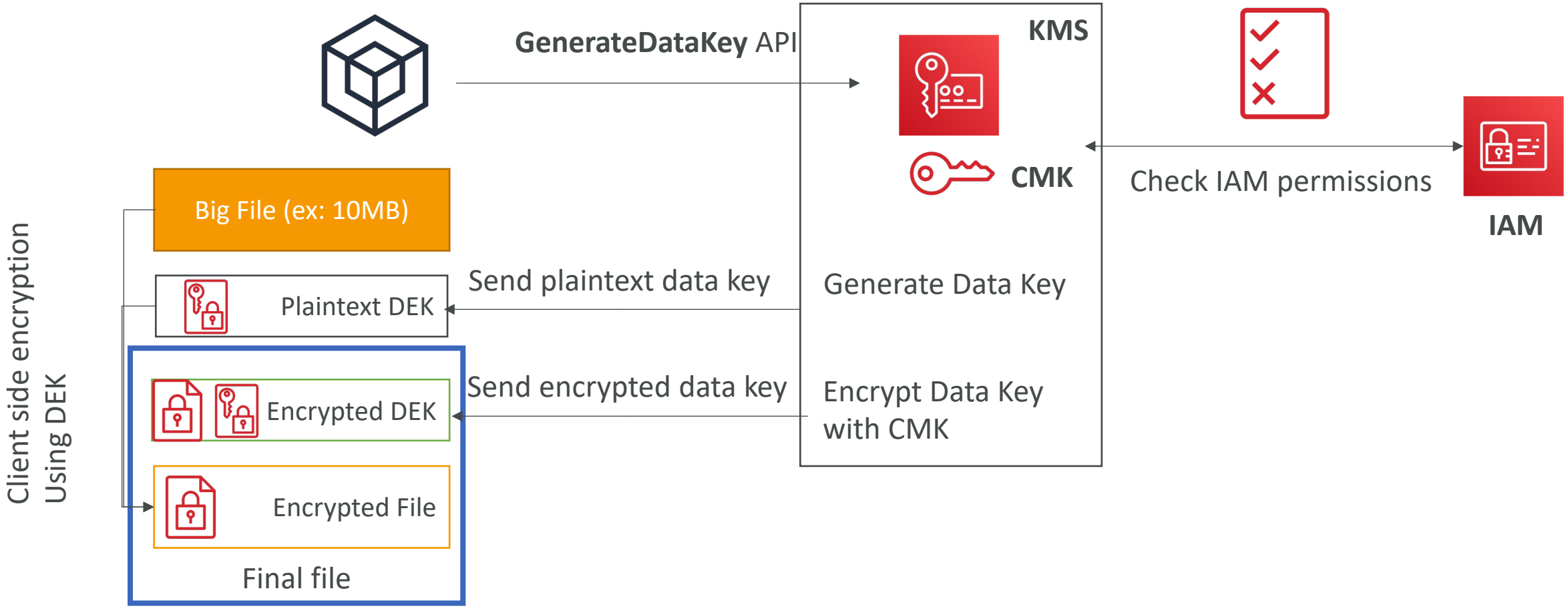


Envelope Encryption

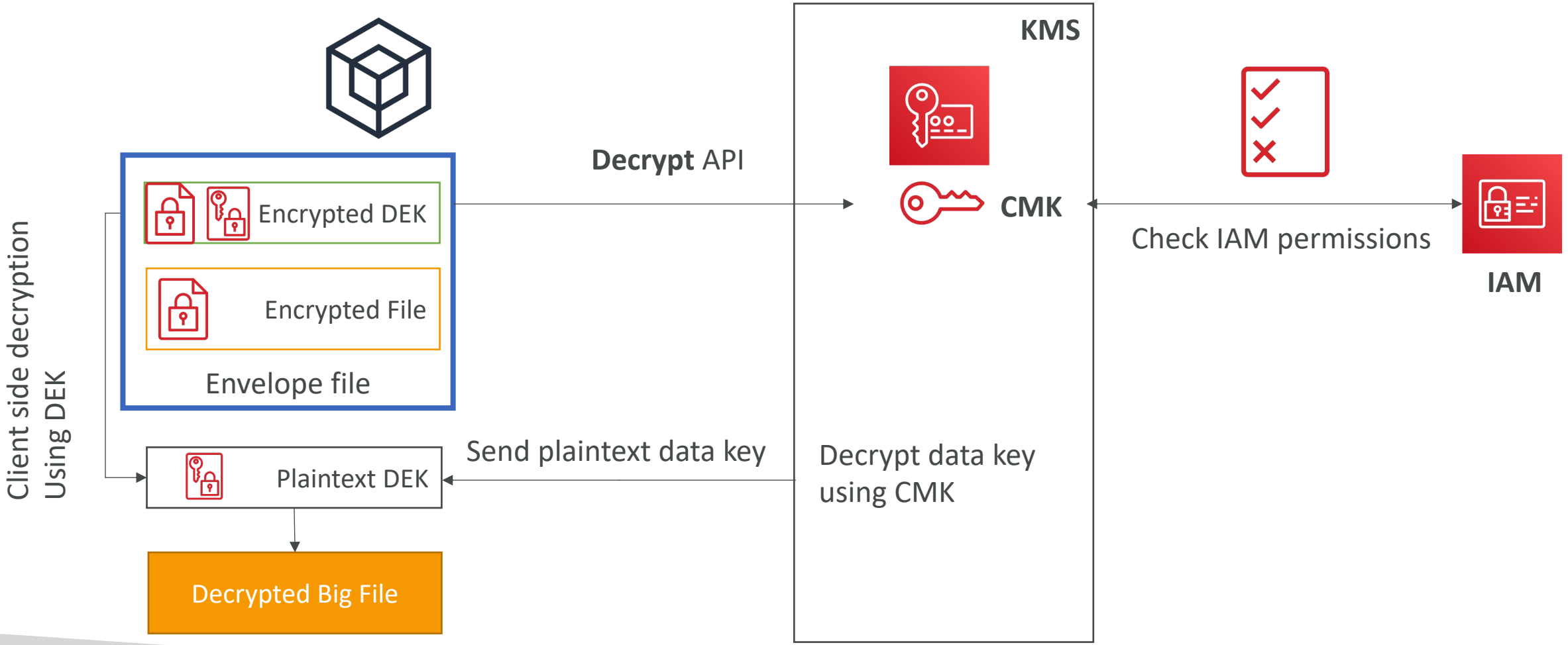
- KMS Encrypt API call has a limit of 4 KB
- If you want to encrypt >4 KB, we need to use **Envelope Encryption**
- The main API that will help us is the **GenerateDataKey** API

- For the exam: anything over 4 KB of data that needs to be encrypted must use the Envelope Encryption == GenerateDataKey API

Deep dive into Envelope Encryption
GenerateDataKey API



Deep dive into Envelope Encryption
Decrypt envelope data



Encryption SDK

- The AWS Encryption SDK implemented Envelope Encryption for us
- The Encryption SDK also exists as a CLI tool we can install
- Implementations for Java, Python, C, JavaScript

- **Feature - Data Key Caching:**
  - re-use data keys instead of creating new ones for each encryption
  - Helps with reducing the number of calls to KMS with a security trade-off
  - Use LocalCryptoMaterialsCache (max age, max bytes, max number of messages)

Encryption SDK – diagram

- The SDK encrypts the data encryption key and stores it (encrypted) as part of the returned ciphertext.

KMS Symmetric – API Summary

- **Encrypt**: encrypt up to 4 KB of data through KMS
- **GenerateDataKey**: generates a unique symmetric data key (DEK)
  - returns a plaintext copy of the data key
  - AND a copy that is encrypted under the CMK that you specify
- **GenerateDataKeyWithoutPlaintext**:
  - Generate a DEK to use at some point (not immediately)
  - DEK that is encrypted under the CMK that you specify (must use Decrypt later)
- **Decrypt**: decrypt up to 4 KB of data (including Data Encryption Keys)
- **GenerateRandom**: Returns a random byte string

KMS Request Quotas

- When you exceed a request quota, you get a ThrottlingException:

```
You have exceeded the rate at which you may call KMS. Reduce the frequency of your calls.
(Service: AWSKMS; Status Code: 400; Error Code: ThrottlingException; Request ID: <ID>
```

- To respond, use **exponential backoff** (backoff and retry)
- For cryptographic operations, they share a quota
- This includes requests made by AWS on your behalf (ex: SSE-KMS)
- For GenerateDataKey, consider using DEK caching from the Encryption SDK
- You can request a Request Quotas increase through API or AWS support

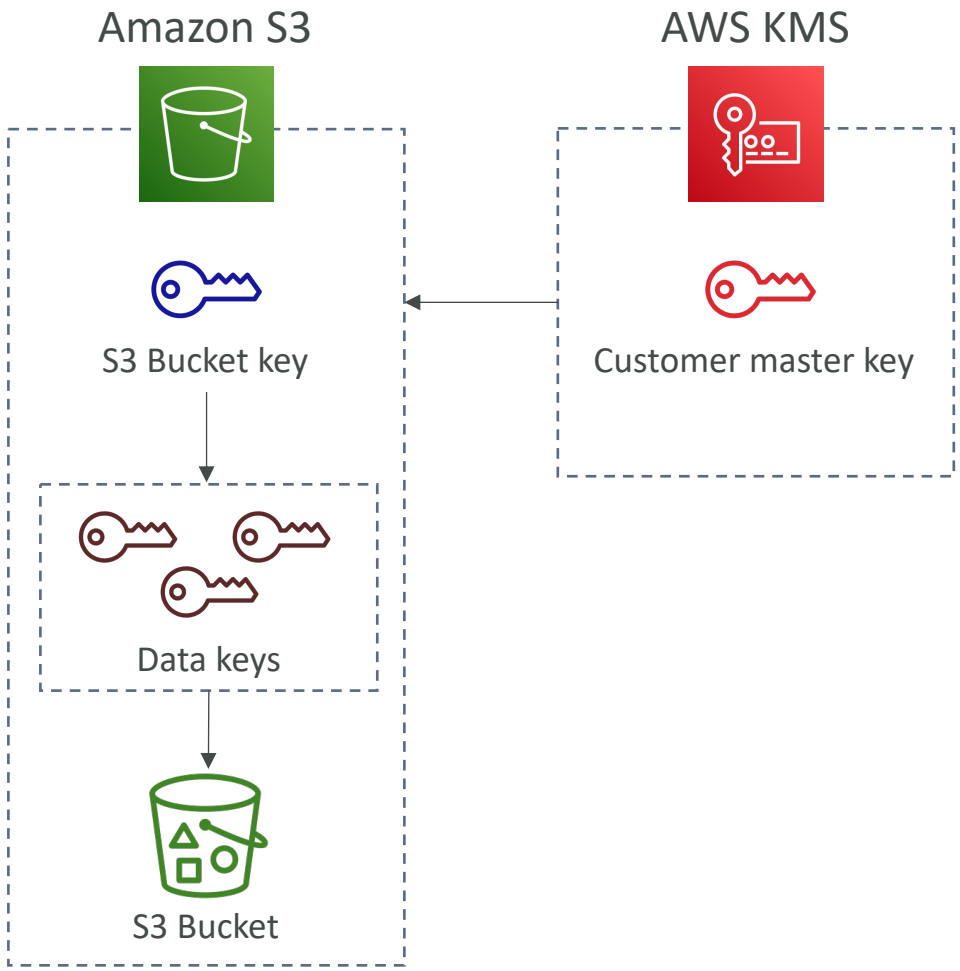
KMS Request Quotas

API operation	Request quotas (per second)

Decrypt
Encrypt
GenerateDataKey (symmetric)
GenerateDataKeyWithoutPlaintext (symmetric)
GenerateRandom
ReEncrypt
Sign (asymmetric)
Verify (asymmetric)
	These shared quotas vary with the AWS Region and the type of CMK used in the request. Each quota is calculated separately.
	
	Symmetric CMK quota:
	• 5,500 (shared)
	• 10,000 (shared) in the following Regions:
	  • us-east-2, ap-southeast-1, ap-southeast-2, ap-northeast-1, eu-central-1, eu-west-2
	• 30,000 (shared) in the following Regions:
	  • us-east-1, us-west-2, eu-west-1
	
	Asymmetric CMK quota:
	• 500 (shared) for RSA CMKs
	• 300 (shared) for Elliptic curve (ECC) CMKs

S3 Bucket Key for SSE-KMS encryption

- New setting to decrease…
  - Number of API calls made to KMS from S3 by 99%
  - Costs of overall KMS encryption with Amazon S3 by 99%
- This leverages data keys
  - A "S3 bucket key" is generated
  - That key is used to encrypt KMS objects with new data keys
- You will see **less KMS CloudTrail events in CloudTrail**



Key Policy – Examples

Default KMS Key Policy

```
{
    "Effect": "Allow",
    "Action": "kms:*",
    "Principal": {
        "AWS": "arn:aws:iam::123456789012:root"
    },
    "Resource": "*"
}
```

Allow Federated User

```
{
    "Effect": "Allow",
    "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:DescribeKey"
    ],
    "Principal": {
        "AWS": "arn:aws:sts::123456789012:federated-user/user-name"
    },
    "Resource": "*"
}
```

Principal Options in IAM Policies

- AWS Account and Root User

```
"Principal": { "AWS": "123456789012" }
"Principal": { "AWS": "arn:aws:iam::123456789012:root" }
```

- IAM Roles

```
"Principal": { "AWS": "arn:aws:iam::123456789012:role/role-name" }
```

- IAM Role Sessions

```
"Principal": { "AWS": "arn:aws:sts::123456789012:assumed-role/role-name/role-session-name" }
"Principal": { "Federated": "cognito-identity.amazonaws.com" }
"Principal": { "Federated": "arn:aws:iam::123456789012:saml-provider/provider-name" }
```

Principal Options in IAM Policies

- IAM Users

```
"Principal": { "AWS": "arn:aws:iam::123456789012:user/user-name" }
```

- Federated User Sessions

```
"Principal": { "AWS": "arn:aws:sts::123456789012:federated-user/user-name" }
```

- AWS Services

```
"Principal": {
    "Service": [
        "ecs.amazonaws.com",
        "elasticloadbalancing.amazonaws.com"
    ]
}
```

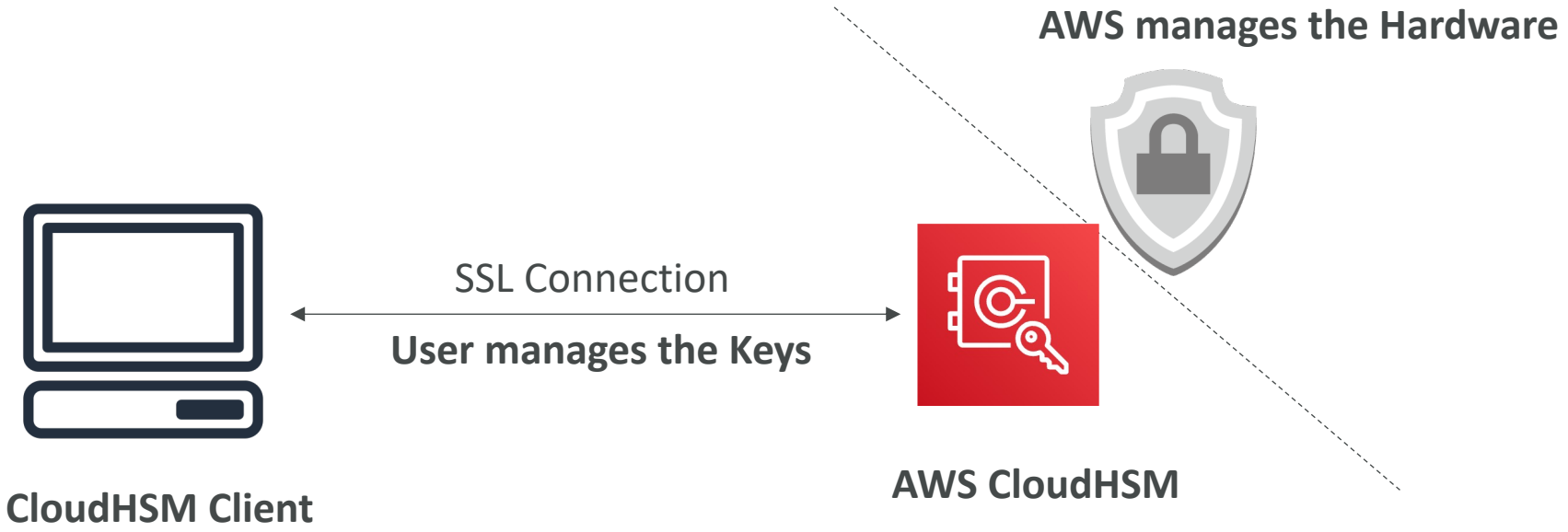
- All Principals

```
"Principal": "*"
"Principal": { "AWS": "*" }
```

CloudHSM

- KMS => AWS manages the software for encryption
- CloudHSM => AWS provisions encryption **hardware**
- Dedicated Hardware (HSM = Hardware Security Module)
- You manage your own encryption keys entirely (not AWS)
- HSM device is tamper resistant, FIPS 140-2 Level 3 compliance
- Supports both symmetric and **asymmetric** encryption (SSL/TLS keys)
- No free tier available
- Must use the CloudHSM Client Software
- Redshift supports CloudHSM for database encryption and key management
- **Good option to use with SSE-C encryption**

CloudHSM Diagram



IAM permissions:

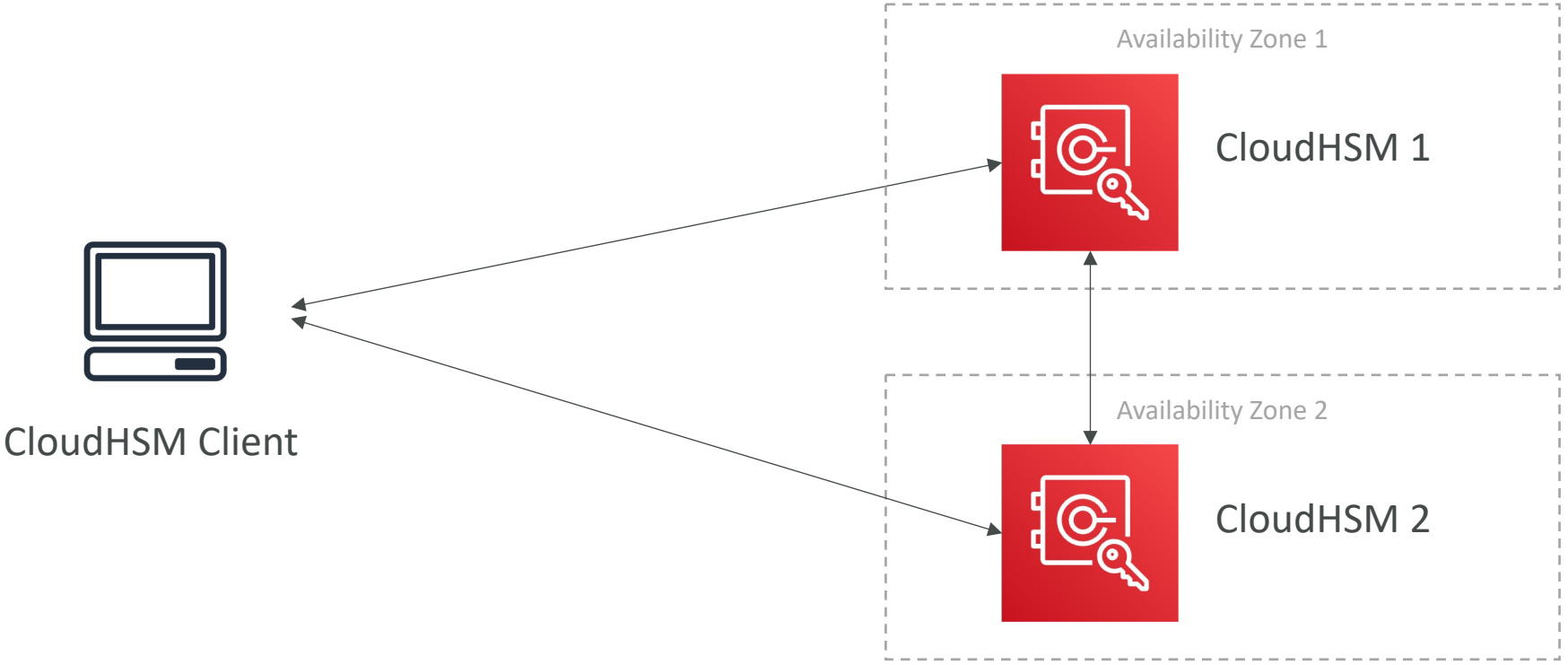
- CRUD an HSM Cluster

CloudHSM Software:

- Manage the Keys
- Manage the Users

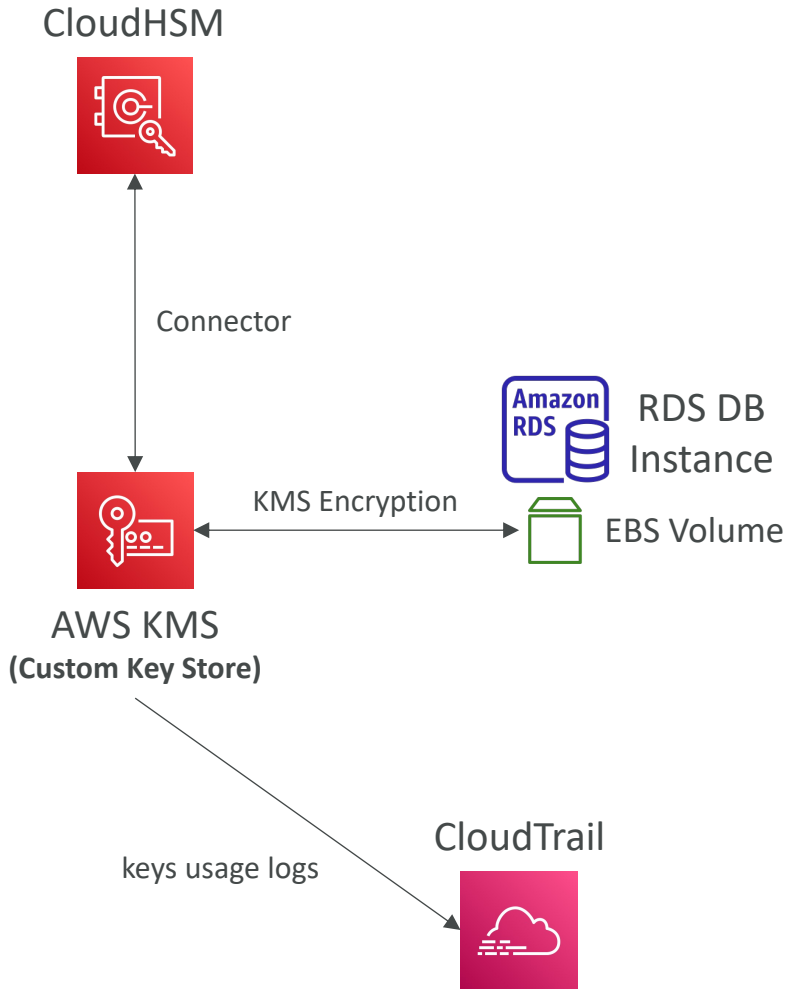
CloudHSM – High Availability

- CloudHSM clusters are spread across Multi AZ (HA)
- Great for availability and durability



CloudHSM – Integration with AWS Services

- Through integration with AWS KMS
- Configure KMS Custom Key Store with CloudHSM
- Example: EBS, S3, RDS …



CloudHSM vs. KMS

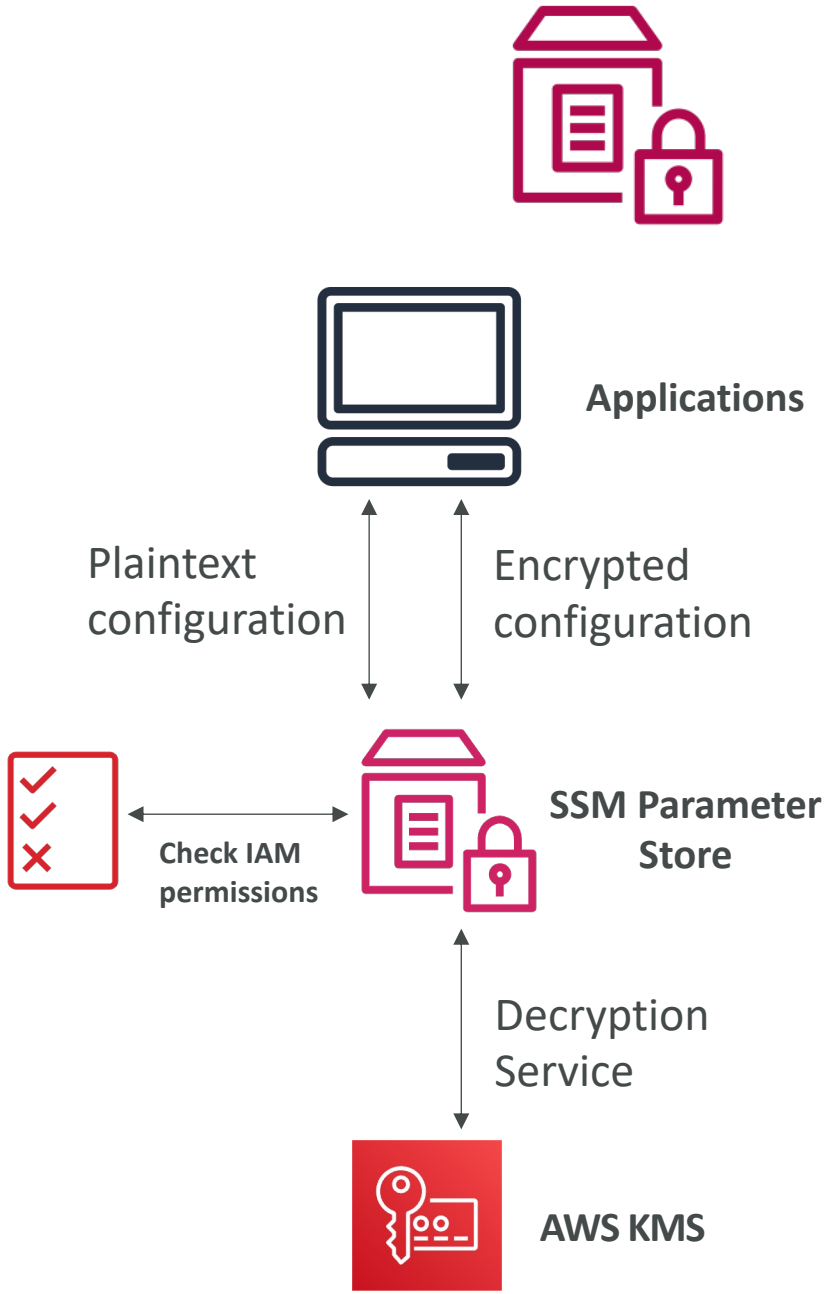
Feature	AWS KMS	AWS CloudHSM
Tenancy	Multi-Tenant	Single-Tenant
Standard	FIPS 140-2 Level 3	FIPS 140-2 Level 3
Master Keys	• AWS Owned CMK • AWS Managed CMK • Customer Managed CMK	Customer Managed CMK
Key Types	• Symmetric • Asymmetric • Digital Signing	• Symmetric • Asymmetric • Digital Signing & Hashing
Key Accessibility	Accessible in multiple AWS regions (can't access keys outside the region it's created in)	• Deployed and managed in a VPC • Can be shared across VPCs (VPC Peering)
Cryptographic Acceleration	None	• SSL/TLS Acceleration • Oracle TDE Acceleration
Access & Authentication	AWS IAM	You create users and manage their permissions

CloudHSM vs. KMS

Feature	AWS KMS	AWS CloudHSM
High Availability	AWS Managed Service	Add multiple HSMs over different AZs
Audit Capability	• CloudTrail • CloudWatch	• CloudTrail • CloudWatch • MFA support
Free Tier	Yes	No

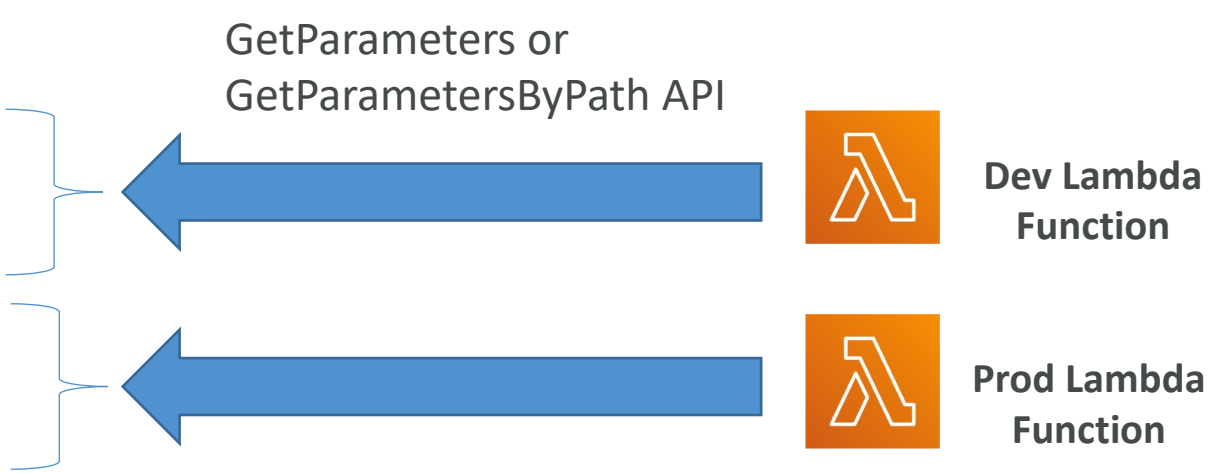
SSM Parameter Store

- Secure storage for configuration and secrets
- Optional Seamless Encryption using KMS
- Serverless, scalable, durable, easy SDK
- Version tracking of configurations / secrets
- Security through IAM
- Notifications with Amazon EventBridge
- Integration with CloudFormation



SSM Parameter Store Hierarchy

- /my-department/
  - my-app/
    - dev/
      - db-url
      - db-password
    - prod/
      - db-url
      - db-password
  - other-app/
- /other-department/
- */aws/reference/secretsmanager/secret_ID_in_Secrets_Manager*
- */aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2 (public)*



Standard and advanced parameter tiers

	Standard	Advanced
Total number of parameters allowed (per AWS account and Region)	10,000	100,000
Maximum size of a parameter value	4 KB	8 KB
Parameter policies available	No	Yes
Cost	No additional charge	Charges apply
Storage Pricing	Free	$0.05 per advanced parameter per month

Parameters Policies (for advanced parameters)

- Allow to assign a TTL to a parameter (expiration date) to force updating or deleting sensitive data such as passwords
- Can assign multiple policies at a time

Expiration (to delete a parameter)

```
{
    "Type":"Expiration",
    "Version":"1.0",
    "Attributes":{
       "Timestamp":"2020-12-02T21:34:33.000Z"
    }
}
```

ExpirationNotification (EventBridge)

```
{
    "Type":"ExpirationNotification",
    "Version":"1.0",
    "Attributes":{
       "Before":"15",
       "Unit":"Days"
    }
}
```

NoChangeNotification (EventBridge)

```
{
    "Type":"NoChangeNotification",
    "Version":"1.0",
    "Attributes":{
       "After":"20",
       "Unit":"Days"
    }
}
```

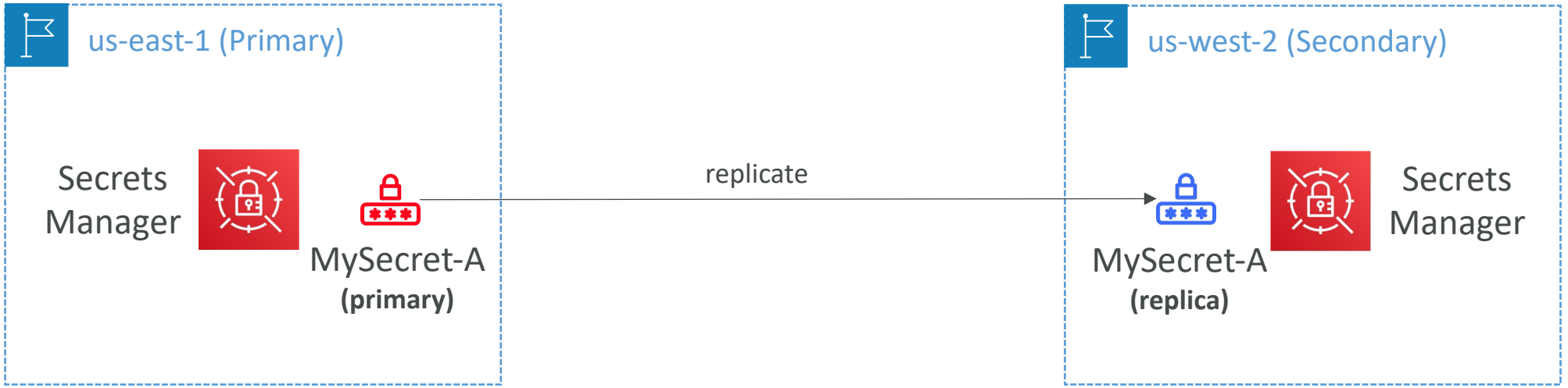
AWS Secrets Manager

- Newer service, meant for storing secrets
- Capability to force **rotation of secrets** every X days
- Automate generation of secrets on rotation (uses Lambda)
- Integration with **Amazon RDS** (MySQL, PostgreSQL, Aurora)
- Secrets are encrypted using KMS

- Mostly meant for RDS integration

AWS Secrets Manager – Multi-Region Secrets

- Replicate Secrets across multiple AWS Regions
- Secrets Manager keeps read replicas in sync with the primary Secret
- Ability to promote a read replica Secret to a standalone Secret
- Use cases: multi-region apps, disaster recovery strategies, multi-region DB…

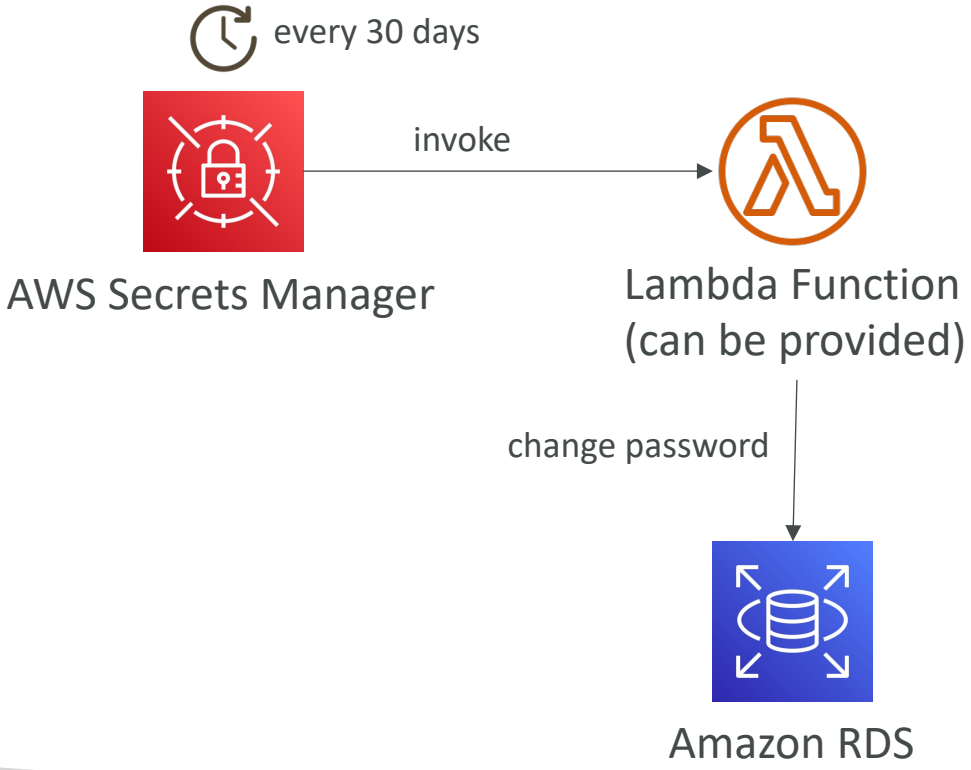


SSM Parameter Store vs Secrets Manager

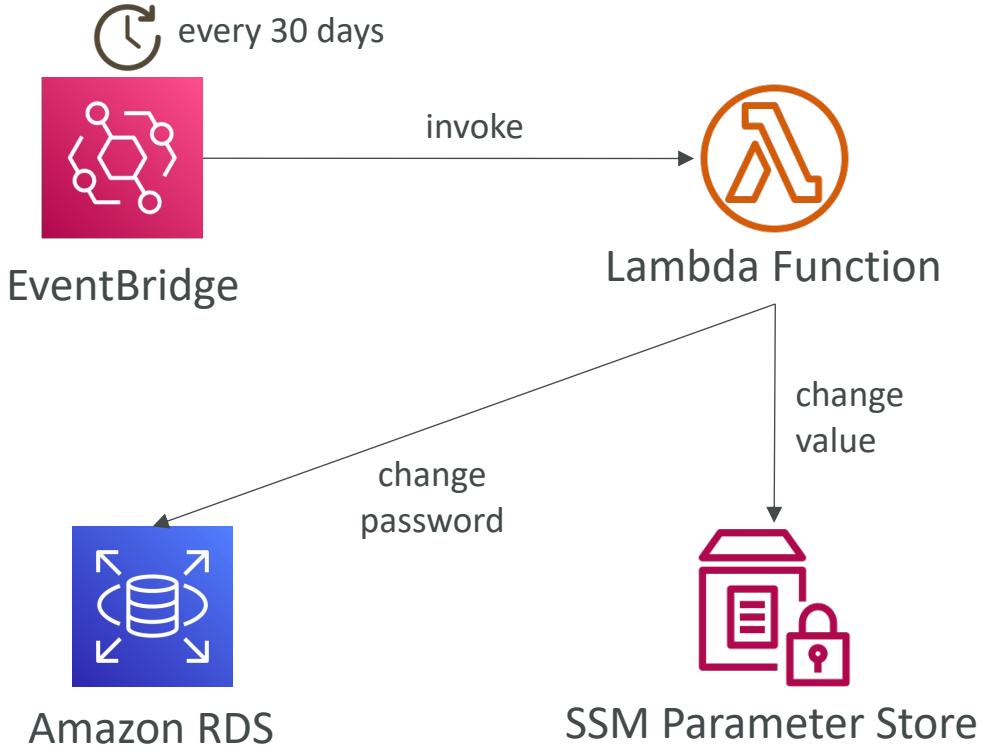
- Secrets Manager ($$$):
  - Automatic rotation of secrets with AWS Lambda
  - Lambda function is provided for RDS, Redshift, DocumentDB
  - KMS encryption is mandatory
  - Can integration with CloudFormation
- SSM Parameter Store ($):
  - Simple API
  - No secret rotation (can enable rotation using Lambda triggered by EventBridge)
  - KMS encryption is optional
  - Can integration with CloudFormation
  - Can pull a Secrets Manager secret using the SSM Parameter Store API

SSM Parameter Store vs. Secrets Manager Rotation

AWS Secrets Manager



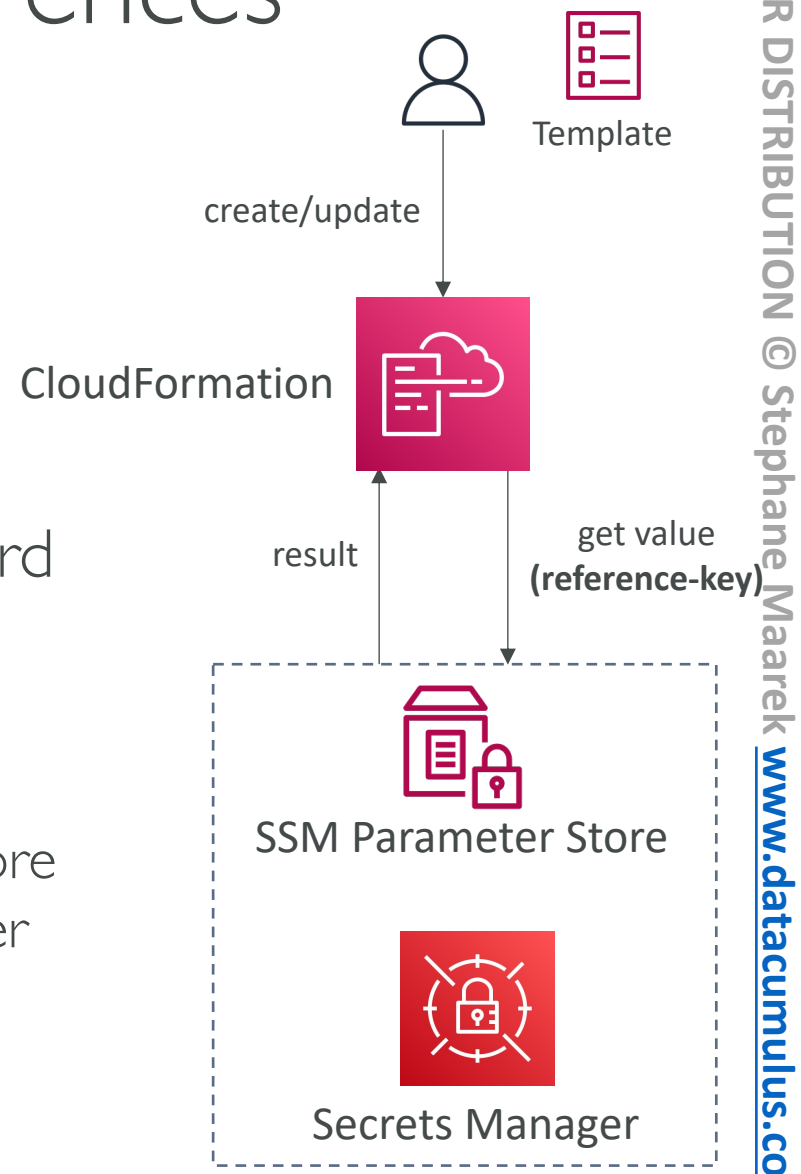
SSM Parameter Store



CloudFormation – Dynamic References

- Reference external values stored in **Systems Manager Parameter Store** and **Secrets Manager** within CloudFormation templates
- CloudFormation retrieves the value of the specified reference during **create/update/delete operations**
- For example: retrieve RDS DB Instance master password from Secrets Manager
- Supports
  - **ssm** – for plaintext values stored in SSM Parameter Store
  - **ssm-secure** – for secure strings stored in SSM Parameter Store
  - **secretsmanager** – for secret values stored in Secrets Manager

'{{resolve:*service-name*:*reference-key*}}'



CloudFormation – Dynamic References

SSM

{{resolve:**ssm**:*parameter-name*:*version*}}

```
Resources:
  S3Bucket:
    Type: AWS::S3::Bucket
    Properties:
      AccessControl: '{{resolve:ssm:S3AccessControl:2}}'
```

SSM Secure

{{resolve:**ssm-secure**:*parameter-name*:*version*}}

```
Resources:
  IAMUser:
    Type: AWS::IAM::User
    Properties:
      UserName: john
      LoginProfile:
        Password: '{{resolve:ssm-secure:IAMUserPassword:10}}'
```

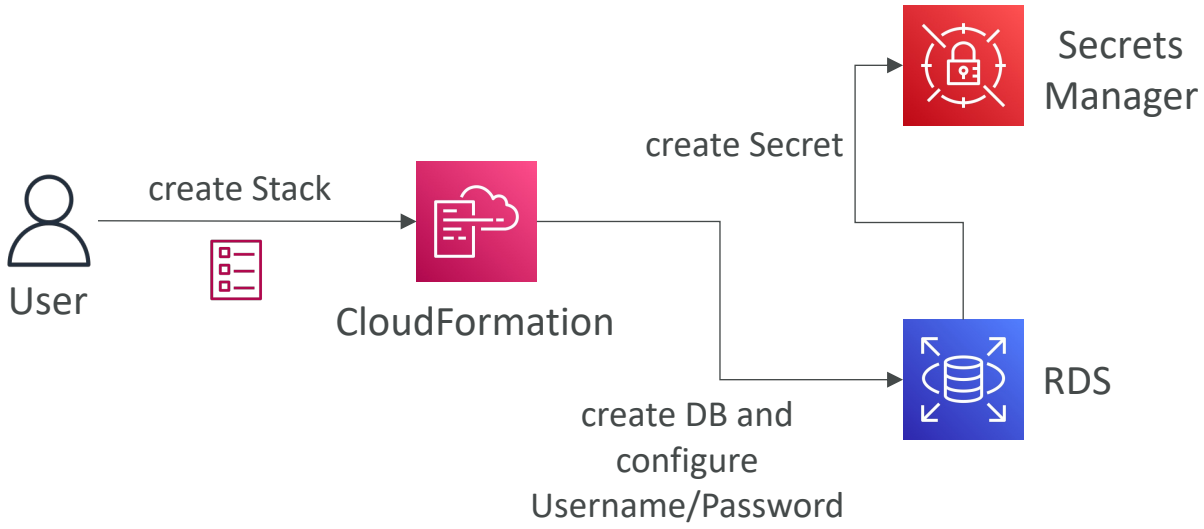
Secrets Manager

{{resolve:**secretsmanager**:*secret-id*:*secret-string*:*json-key*:*version-stage*:*version-id*}}

```
Resources:
  DBInstance:
    Type: AWS::RDS::DBInstance
    Properties:
      DBName: MyRDSInstance
      MasterUsername: '{{resolve:secretsmanager:MyRDSSecret:SecretString:username}}'
      MasterUserPassword: '{{resolve:secretsmanager:MyRDSSecret:SecretString:password}}'
```

CloudFormation, Secrets Manager & RDS Option 1 – ManageMasterUserPassword

- **ManageMasterUserPassword** – creates admin secret implicitly
- RDS, Aurora will manage the secret in Secrets Manager and its rotation




```yaml
Resources:
  MyCluster:
    Type: AWS::RDS::DBCluster
    Properties:
      Engine: aurora-mysql
      MasterUsername: masteruser
      ManageMasterUserPassword: true

Outputs:
  Secret:
    Value: !GetAtt MyCluster.MasterUserSecret.SecretArn
```

CloudFormation, Secrets Manager & RDS Option 2 – Dynamic Reference

1. secret is generated

```yaml
Resources:
  MyDatabaseSecret:
    Type: AWS::SecretsManager::Secret
    Properties:
      Name: MyDatabaseSecret
      GenerateSecretString:
        SecretStringTemplate: '{"username": "admin"}'
        GenerateStringKey: "password"
        PasswordLength: 16
        ExcludeCharacters: '"@/\'
```

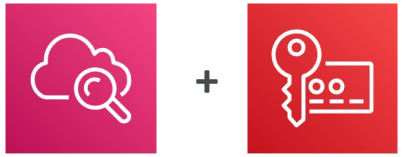
3. link the secret to RDS DB instance (for rotation)

```yaml
  SecretRDSAttachment:
    Type: AWS::SecretsManager::SecretTargetAttachment
    Properties:
      SecretId: !Ref MyDatabaseSecret
      TargetId: !Ref MyDBInstance
      TargetType: AWS::RDS::DBInstance
```

```yaml
  MyDBInstance:
    Type: AWS::RDS::DBInstance
    Properties:
      DBName: mydatabase
      AllocatedStorage: 20
      DBInstanceClass: db.t2.micro
      Engine: mysql
      MasterUsername: '{{resolve:secretsmanager:MyDatabaseSecret:SecretString:username}}'
      MasterUserPassword: '{{resolve:secretsmanager:MyDatabaseSecret:SecretString:password}}'
```

2. Reference secret in RDS DB instance

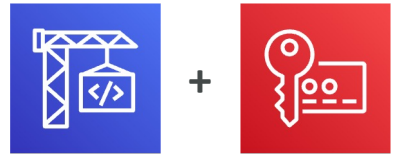
CloudWatch Logs - Encryption



- You can encrypt CloudWatch logs with KMS keys
- Encryption is enabled at the log group level, by associating a CMK with a log group, either when you create the log group or after it exists.
- You cannot associate a CMK with a log group using the CloudWatch console.

- You must use the CloudWatch Logs API:
  - **associate-kms-key** : if the log group already exists
  - **create-log-group:** if the log group doesn't exist yet

CodeBuild Security



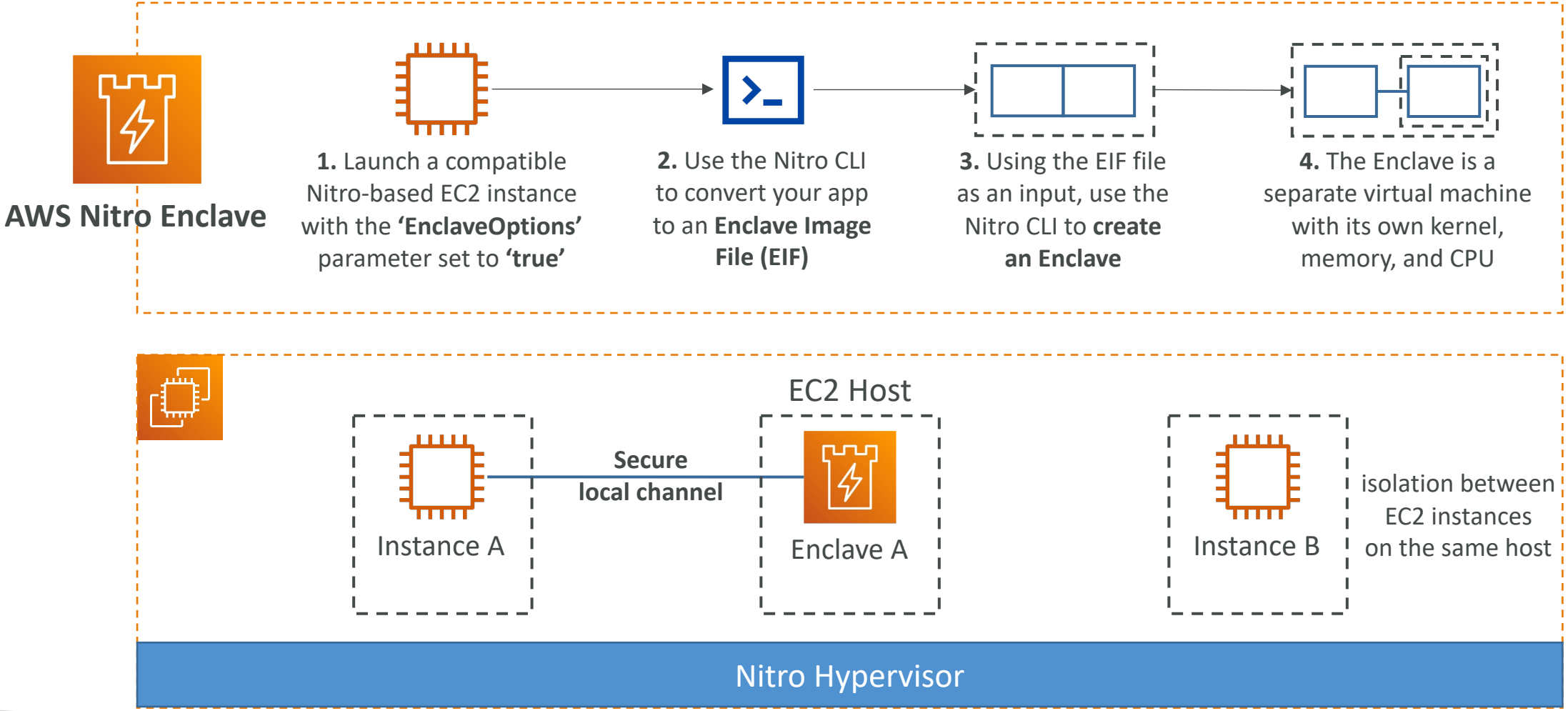
- To access resources in your VPC, make sure you specify a VPC configuration for your CodeBuild

- Secrets in CodeBuild:
- Don't store them as plaintext in environment variables
- Instead…
  - Environment variables can reference parameter store parameters
  - Environment variables can reference secrets manager secrets

AWS Nitro Enclaves

- Process highly sensitive data in an isolated compute environment
  - Personally Identifiable Information (PII), healthcare, financial, …
- Fully isolated virtual machines, hardened, and highly constrained
  - Not a container, not persistent storage, no interactive access, no external networking
- Helps reduce the attack surface for sensitive data processing apps
  - **Cryptographic Attestation** – only authorized code can be running in your Enclave
  - Only Enclaves can access sensitive data (integration with KMS)
- Use cases: securing private keys, processing credit cards, secure multi-party computation…

AWS Nitro Enclaves



Other Services

Quick overview of other services that might have questions on at the exam

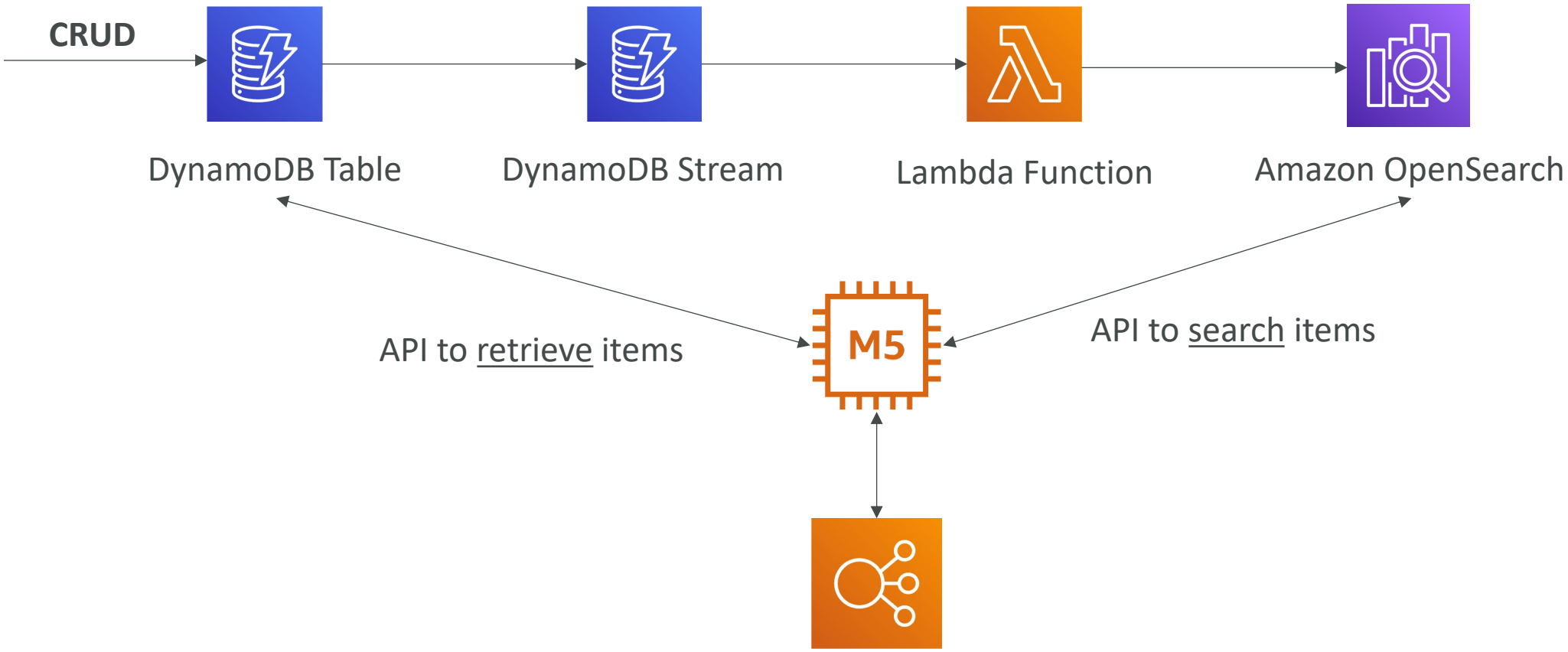
Amazon Simple Email Service (Amazon SES)

- **Fully managed service to send emails securely, globally and at scale**
- Allows inbound/outbound emails
- Reputation dashboard, performance insights, anti-spam feedback
- Provides statistics such as email deliveries, bounces, feedback loop results, email open
- Supports DomainKeys Identified Mail (DKIM) and Sender Policy Framework (SPF)
- Flexible IP deployment: shared, dedicated, and customer-owned IPs
- Send emails using your application using AWS Console, APIs, or SMTP
- Use cases: transactional, marketing and bulk email communications

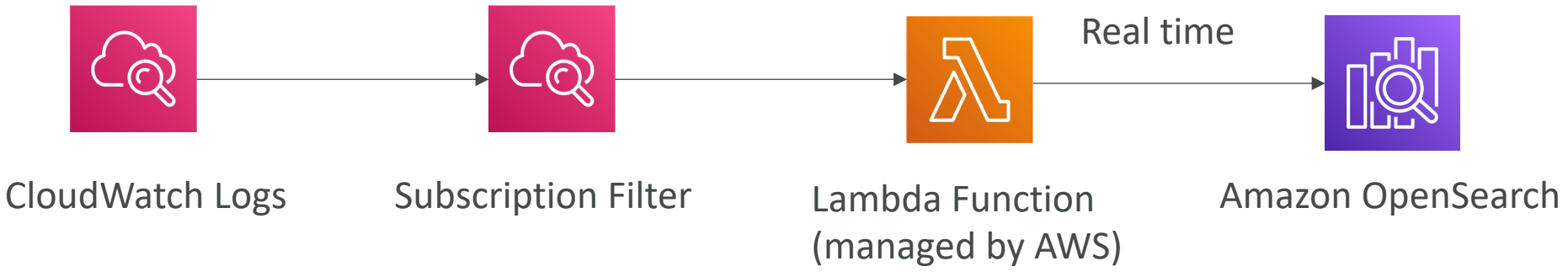
Amazon OpenSearch Service

- *Amazon OpenSearch is successor to Amazon ElasticSearch*
- In DynamoDB, queries only exist by primary key or indexes…
- **With OpenSearch, you can search any field, even partially matches**
- It's common to use OpenSearch as a complement to another database
- Two modes: managed cluster or serverless cluster
- Does *not* natively support SQL (can be enabled via a plugin)
- Ingestion from Kinesis Data Firehose, AWS IoT, and CloudWatch Logs
- Security through Cognito & IAM, KMS encryption, TLS
- Comes with OpenSearch Dashboards (visualization)

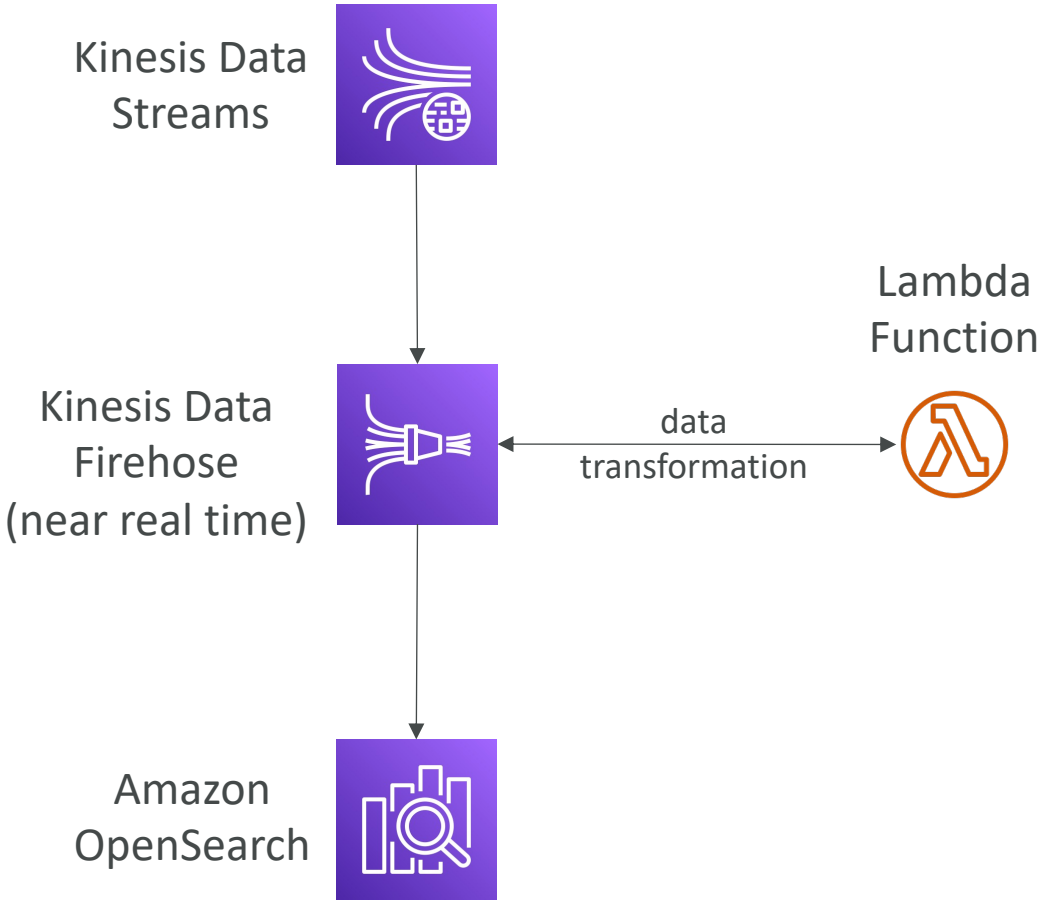
OpenSearch patterns DynamoDB

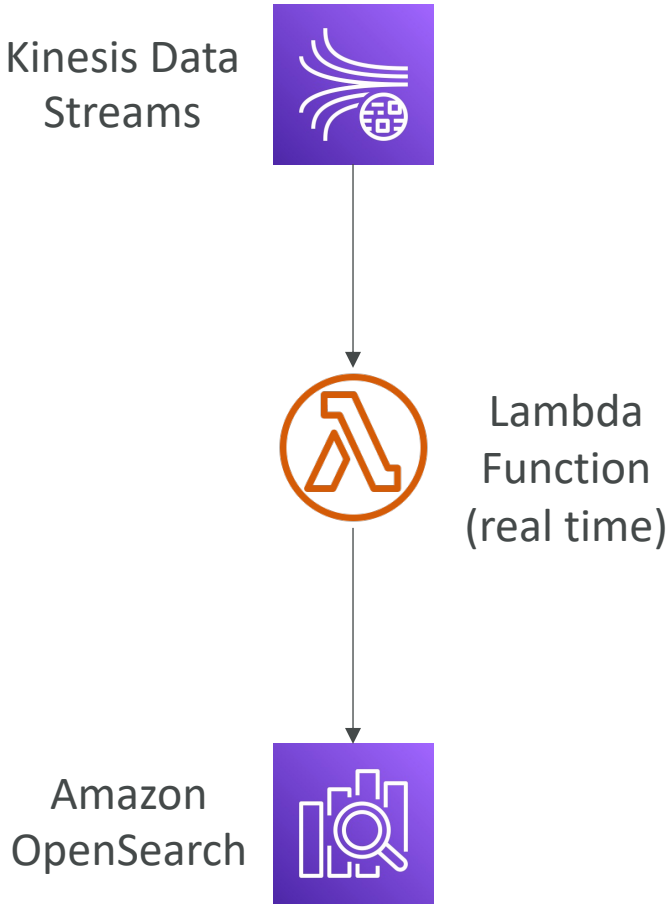


OpenSearch patterns CloudWatch Logs



OpenSearch patterns
Kinesis Data Streams & Kinesis Data Firehose

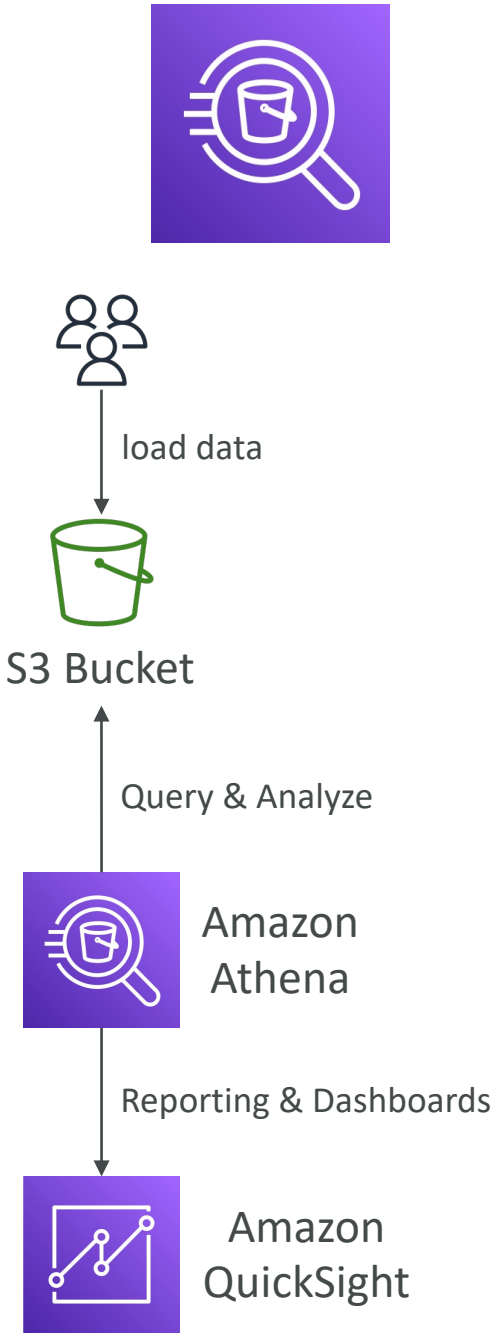




Amazon Athena

- **Serverless** query service to analyze data stored in Amazon S3
- Uses standard SQL language to query the files (built on Presto)
- Supports CSV, JSON, ORC, Avro, and Parquet
- Pricing: $5.00 per TB of data scanned
- Commonly used with Amazon Quicksight for reporting/dashboards

- **Use cases:** Business intelligence / analytics / reporting, analyze & query VPC Flow Logs, ELB Logs, **CloudTrail trails**, etc...
- Exam Tip: analyze data in S3 using serverless SQL, use Athena

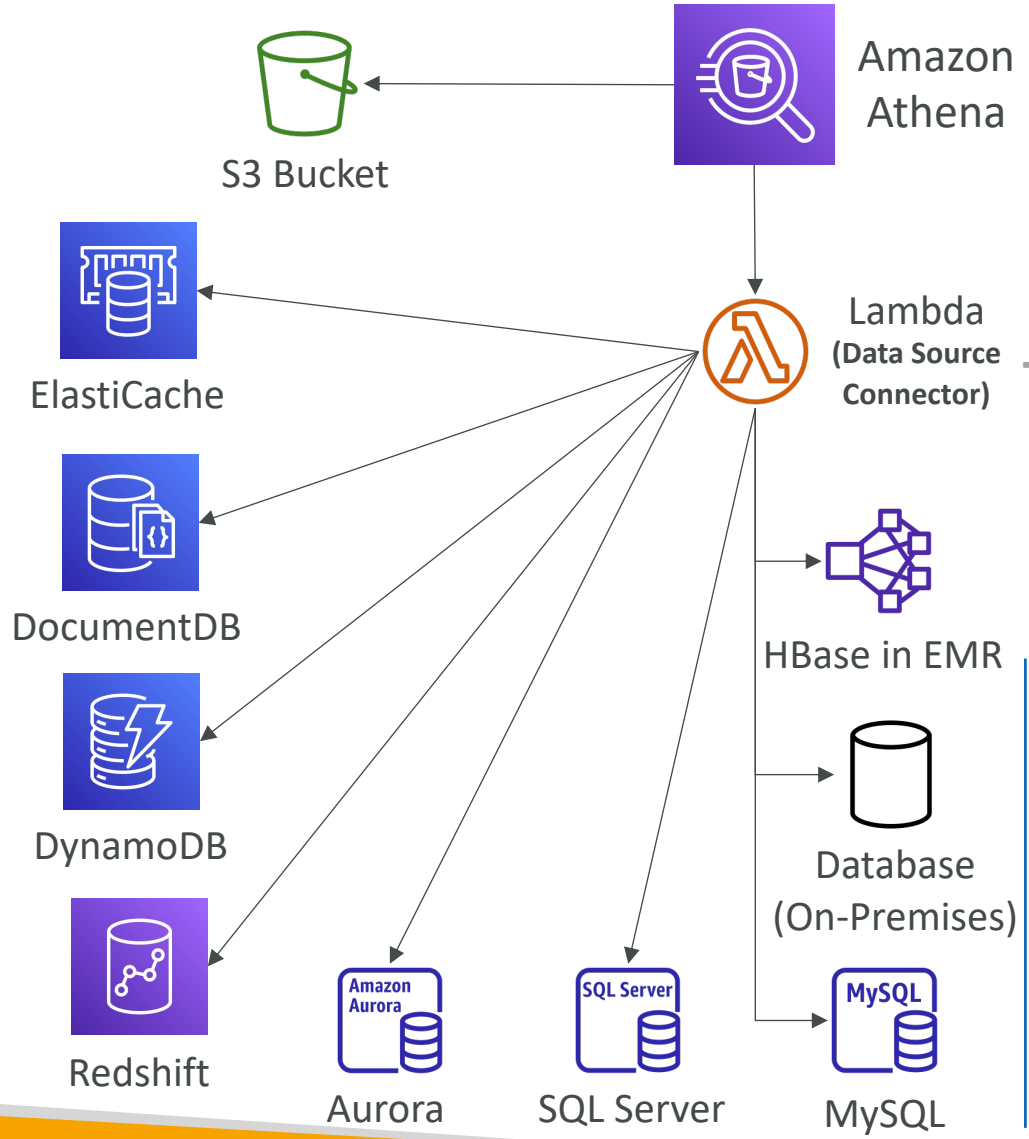


Amazon Athena – Performance Improvement

- Use **columnar data** for cost-savings (less scan)
  - Apache Parquet or ORC is recommended
  - Huge performance improvement
  - Use Glue to convert your data to Parquet or ORC
- **Compress data** for smaller retrievals (bzip2, gzip, lz4, snappy, zlip, zstd…)
- **Partition** datasets in S3 for easy querying on virtual columns
  - s3://yourBucket/pathToTable
    /<PARTITION_COLUMN_NAME>=<VALUE>
    /<PARTITION_COLUMN_NAME>=<VALUE>
    /<PARTITION_COLUMN_NAME>=<VALUE>
    /etc…
  - Example: s3://athena-examples/flight/parquet/year=1991/month=1/day=1/
- **Use larger files** (> 128 MB) to minimize overhead

Amazon Athena – Federated Query

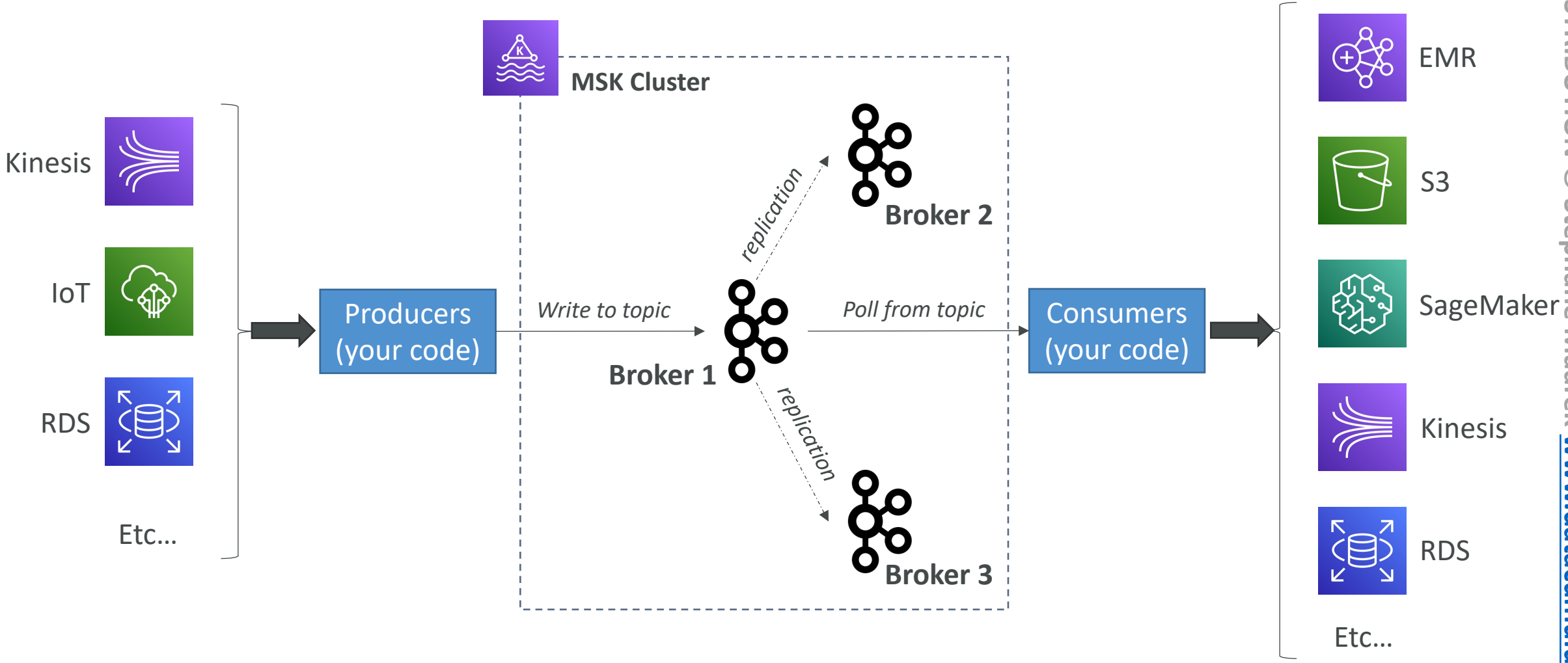
- Allows you to run SQL queries across data stored in relational, non-relational, object, and custom data sources (AWS or on-premises)
- Uses Data Source Connectors that run on AWS Lambda to run Federated Queries (e.g., CloudWatch Logs, DynamoDB, RDS, …)
- Store the results back in Amazon S3



Amazon Managed Streaming for Apache K (Amazon MSK)

- Alternative to Amazon Kinesis
- Fully managed Apache Kafka on AWS
  - Allow you to create, update, delete clusters
  - MSK creates & manages Kafka brokers nodes & Zookeeper nodes for you
  - Deploy the MSK cluster in your VPC, multi-AZ (up to 3 for HA)
  - Automatic recovery from common Apache Kafka failures
  - Data is stored on EBS volumes **for as long as you want**
- **MSK Serverless**
  - Run Apache Kafka on MSK without managing the capacity
  - MSK automatically provisions resources and scales compute & storage

Apache Kafka at a high level



Kinesis Data Streams vs. Amazon MSK

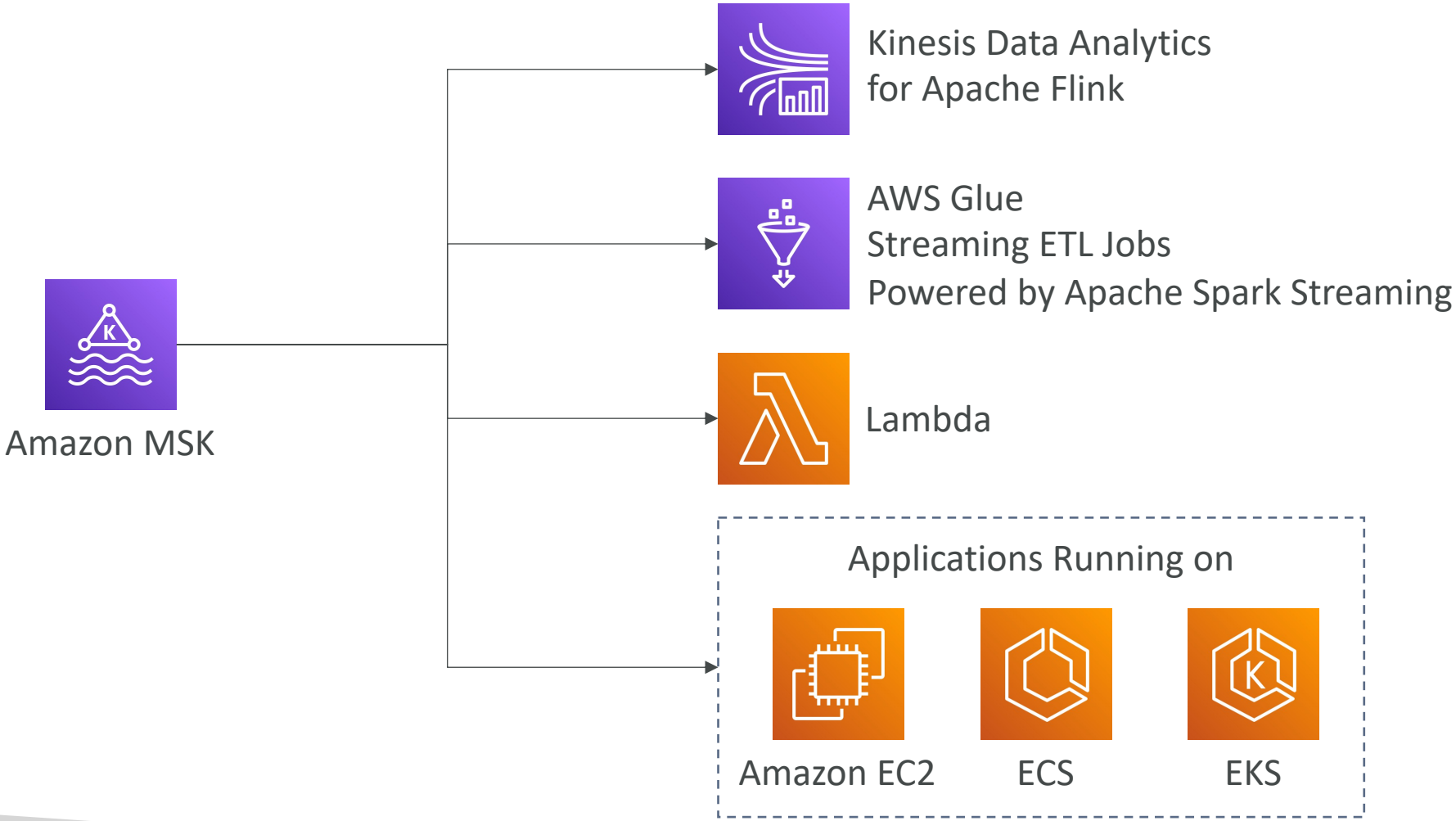
Kinesis Data Streams

- 1 MB message size limit
- Data Streams with Shards
- Shard Splitting & Merging
- TLS In-flight encryption
- KMS at-rest encryption

Amazon MSK

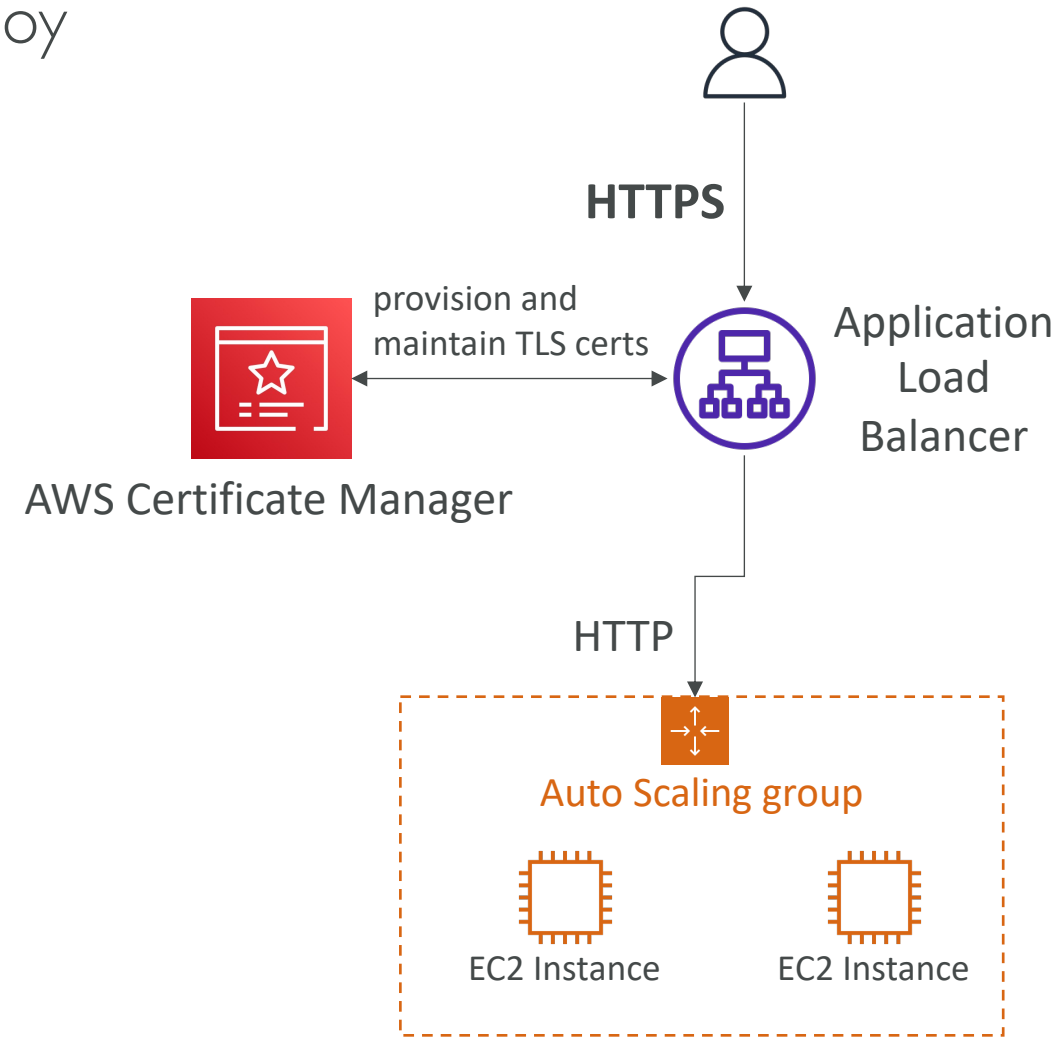
- 1MB default, configure for higher (ex: 10MB)
- Kafka Topics with Partitions
- Can only add partitions to a topic
- PLAINTEXT or TLS In-flight Encryption
- KMS at-rest encryption

Amazon MSK Consumers



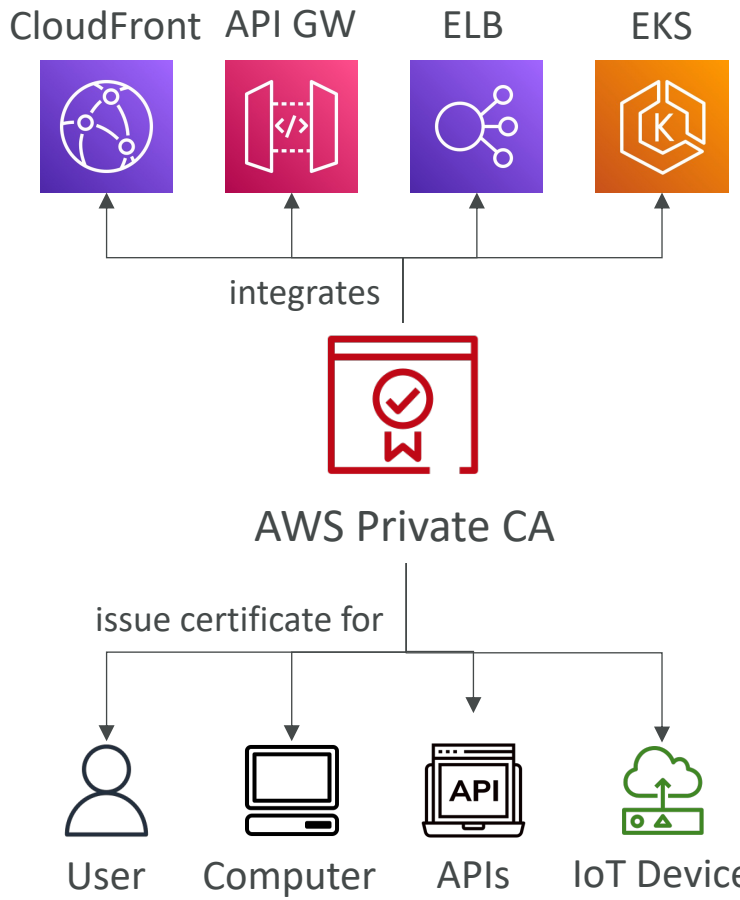
AWS Certificate Manager (ACM)

- Let's you easily provision, manage, and deploy **SSL/TLS Certificates**
- Used to provide in-flight encryption for websites (HTTPS)
- Supports both public and private TLS certificates
- Free of charge for public TLS certificates
- Automatic TLS certificate renewal
- Integrations with (load TLS certificates on)
  - Elastic Load Balancers
  - CloudFront Distributions
  - APIs on API Gateway



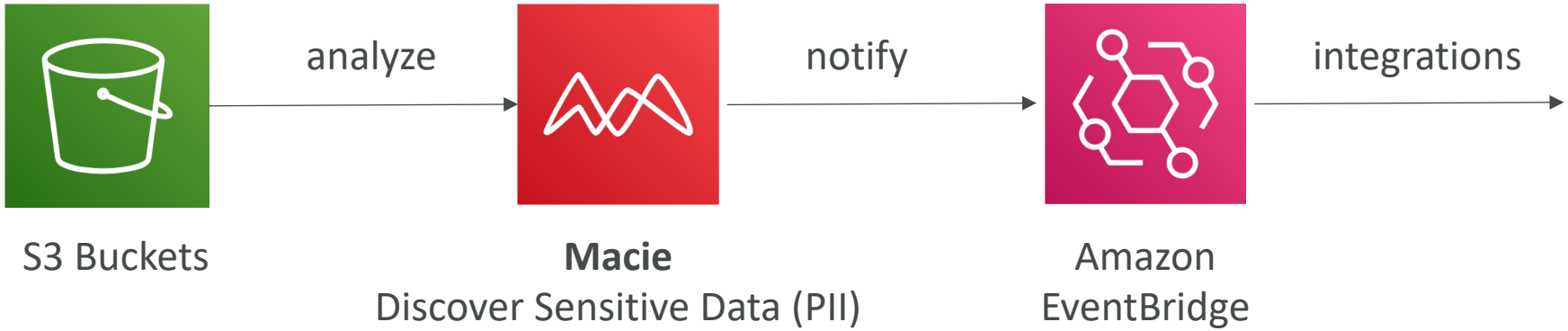
AWS Private Certificate Authority (CA)

- Managed service allows you to create private Certificate Authorities (CA), including root and subordinaries CAs
- Can issue and deploy end-entity X.509 certificates
- Certificates are trusted only by your Organization (not the public Internet)
- Works for AWS services that are integrated with ACM
- Use cases:
  - Encrypted TLS communication, Cryptographically signing code
  - Authenticate users, computers, API endpoints, and IoT devices
  - Enterprise customers building a Public Key Infrastructure (PKI)



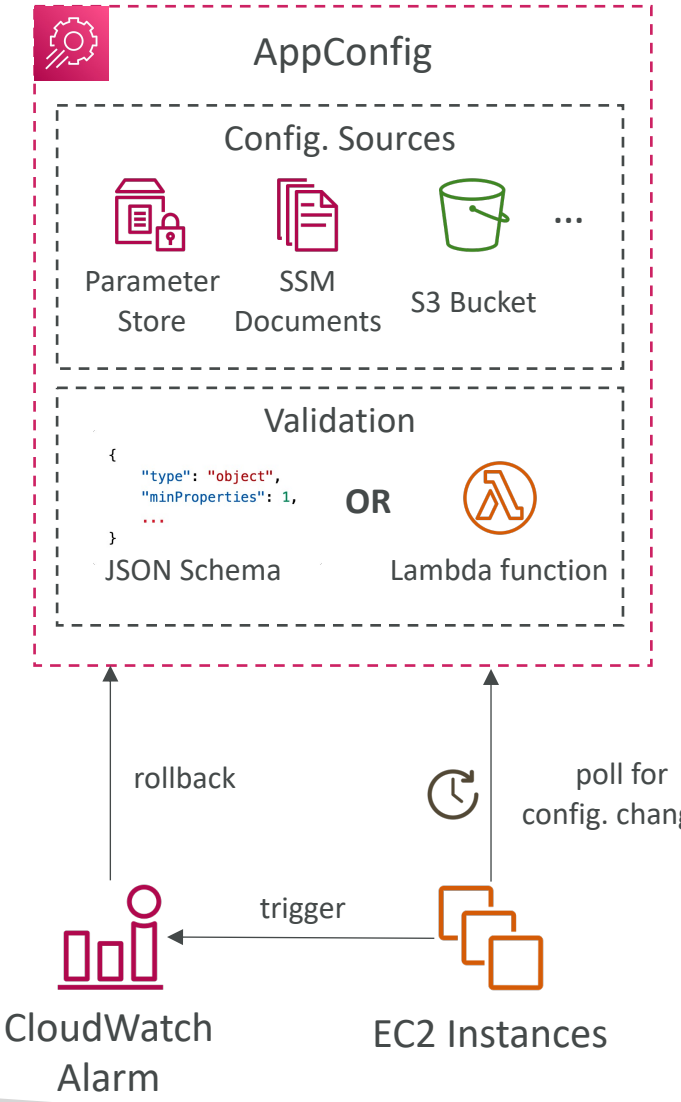
AWS Macie

- Amazon Macie is a fully managed data security and data privacy service that uses **machine learning and pattern matching to discover and protect your sensitive data in AWS.**
- Macie helps identify and alert you to **sensitive data, such as personally identifiable information (PII)**



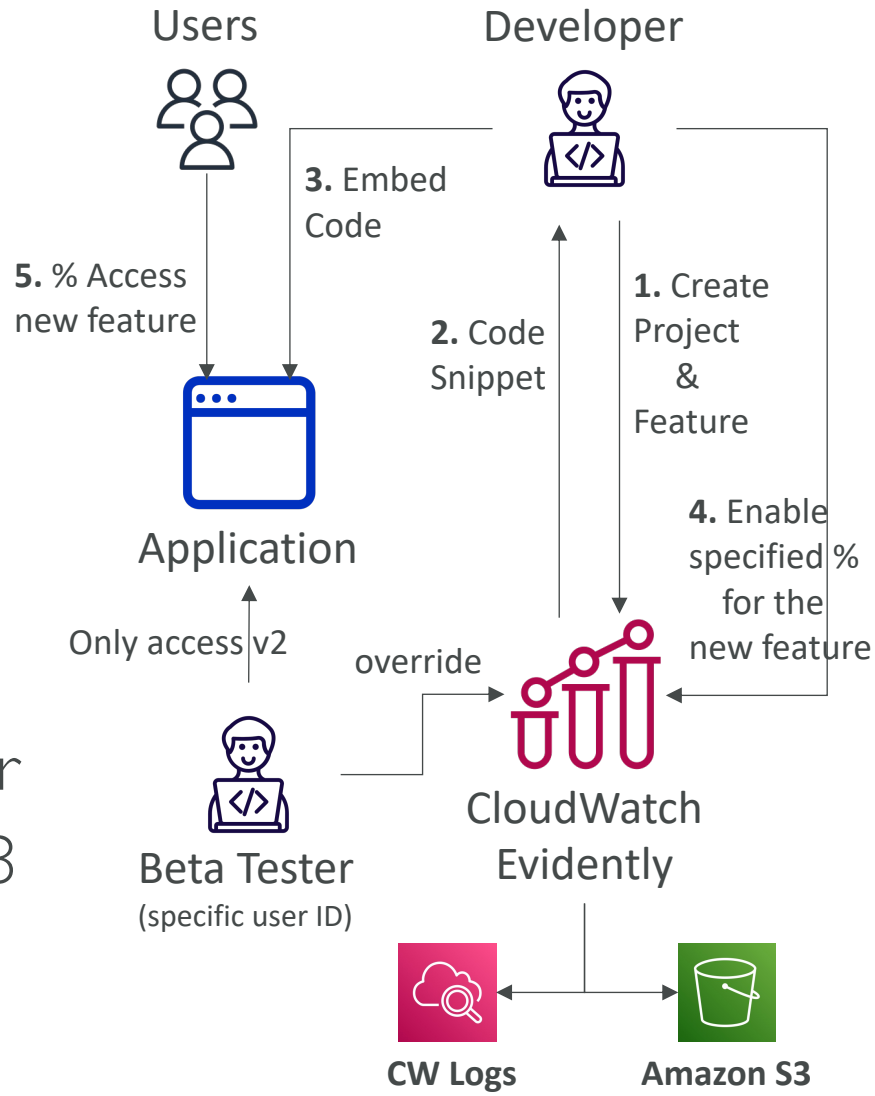
AWS AppConfig

- Configure, validate, and deploy dynamic configurations
- Deploy dynamic configuration changes to your applications independently of any code deployments
  - You don't need to restart the application
- Feature flags, application tuning, allow/block listing…
- Use with apps on EC2 instances, Lambda, ECS, EKS…
- Gradually deploy the configuration changes and rollback if issues occur
- Validate configuration changes before deployment using:
  - **JSON Schema** (syntactic check) or
  - **Lambda Function** – run code to perform validation (semantic check)



CloudWatch Evidently

- Safely validate new features by serving them to a specified % of your users
  - Reduce risk and identify unintended consequences
  - Collect experiment data, analyze using stats, monitor performance
- **Launches** (= feature flags): enable and disable features for a subset of users
- **Experiments** (= A/B testing): compare multiple versions of the same feature
- **Overrides**: pre-define a variation for a specific user
- Store evaluation events in CloudWatch Logs or S3



Exam Review & Tips

State of learning checkpoint

- Let's look how far we've gone on our learning journey

- https://aws.amazon.com/certification/certified-developer-associate/

Practice makes perfect

- If you're new to AWS, take a bit of AWS practice thanks to this course before rushing to the exam
- The exam recommends you to have one or more years of hands-on developing and maintaining an AWS based applications
- Practice makes perfect!

- If you feel overwhelmed by the amount of knowledge you just learned, just go through it one more time

Ideas for practicing…!

- Take one of your existing applications
- Try deploying it manually on EC2
- Try deploying it on Elastic Beanstalk and have it scale
- Try creating a CICD pipeline for it
- Try decoupling components using SQS / SNS
- If possible, try running it on AWS Lambda & friends
- Write automation scripts using the CLI / SDK
  - Idea 1: Shut down EC2 instances at night / start in the morning
  - Idea 2: Automate snapshots of EBS volumes at night
  - Idea 3: List all under-utilized EC2 instances (CPU Utilization < 10%)

Proceed by elimination

- Most questions are going to be scenario based
- For all the questions, rule out answers that you know for sure are wrong
- For the remaining answers, understand which one makes the most sense

- There are very few trick questions
- Don't over-think it
- If a solution seems feasible but highly complicated, it's probably wrong

Skim the AWS Whitepapers

- You can read about some AWS White Papers here:
  - AWS Security Best Practices
  - AWS Well-Architected Framework
  - Architecting for the Cloud AWS Best Practices
  - Practicing Continuous Integration and Continuous Delivery on AWS Accelerating Software Delivery with DevOps
  - Microservices on AWS
  - Serverless Architectures with AWS Lambda
  - Optimizing Enterprise Economics with Serverless Architectures
  - Running Containerized Microservices on AWS
  - Blue/Green Deployments on AWS
- Overall we've explored all the most important concepts in the course
- It's never bad to have a look at the whitepapers you think are interesting!

Read each service's FAQ

- FAQ = Frequently asked questions
- Example: https://aws.amazon.com/lambda/faqs/

- FAQ cover a lot of the questions asked at the exam
- They help confirm your understanding of a service

Get into the AWS Community

- Help out and discuss with other people in the course Q&A
- Review questions asked by other people in the Q&A
- Do the practice test in this section

- Read forums online
- Read online blogs
- Attend local meetups and discuss with other AWS engineers
- Watch re-invent videos on Youtube (AWS Conference)

How will the exam work?

- You'll have to register online at https://www.aws.training/
- Fee for the exam is 150 USD
- Provide one identity documents (ID, Passport, details are in emails sent to you…)
- No notes are allowed, no pen is allowed, no speaking
- 65 questions will be asked in 130 minutes
- Use the "Flag" feature to mark questions you want to re-visit
- At the end you can optionally review all the questions / answers

- To pass you need a score of a least 720 out of 1000
- You will know within 5 days if you passed / failed the exams (most of the time less)
- You will know the overall score a few days later (email notification)
- You will not know which answers were right / wrong
- If you fail, you can retake the exam again 14 days later

Your AWS Certification journey

Foundational

Knowledge-based certification for
foundational understanding of AWS Cloud.
No prior experience needed.

Associate

Role-based certifications that showcase your knowledge
and skills on AWS and build your credibility as an AWS Cloud professional.
Prior cloud and/or strong on-premises IT experience recommended.

Professional

Role-based certifications that validate advanced skills
and knowledge required to design secure, optimized,
and modernized applications and to automate processes on AWS.
2 years of prior AWS Cloud experience recommended.

Specialty

Dive deeper and position yourself as a trusted advisor to your
stakeholders and/or customers in these strategic areas.
Refer to the exam guides on the exam pages for recommended experience.

AWS Certification Paths – Architecture

Architecture

Solutions Architect

Design, develop, and manage cloud infrastructure and assets, work with DevOps to migrate applications to the cloud

optional for IT/ cloud professionals

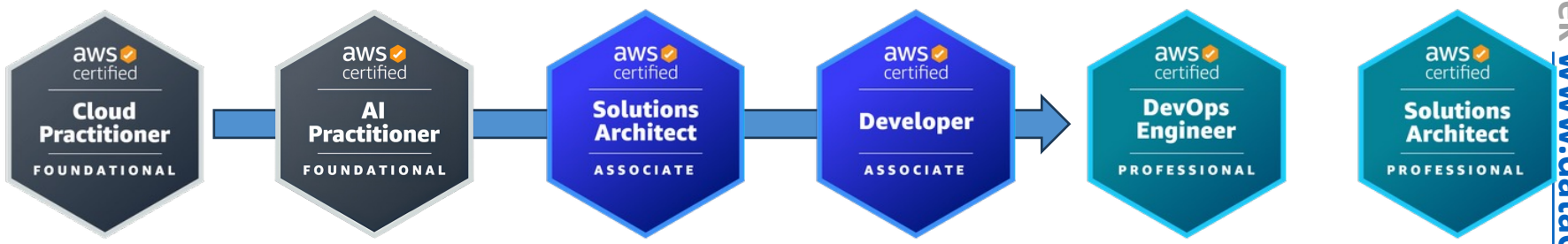
recommended for IT/cloud professionals to leverage AI

Dive Deep

Architecture

Application Architect

Design significant aspects of application architecture including user interface, middleware, and infrastructure, and ensure enterprise-wide scalable, reliable, and manageable systems



optional for IT/ cloud professionals

recommended for IT/cloud professionals to leverage AI

Dive Deep

https://d1.awsstatic.com/training-and-certification/docs/AWS_certification_paths.pdf

AWS Certification Paths – Operations

Operations
Systems Administrator
Install, upgrade, and maintain computer components and software, and integrate automation processes

optional for IT/ cloud professionals

Dive Deep

Operations
Cloud Engineer
Implement and operate an organization's networked computing infrastructure and Implement security systems to maintain data safety

optional for IT/ cloud professionals

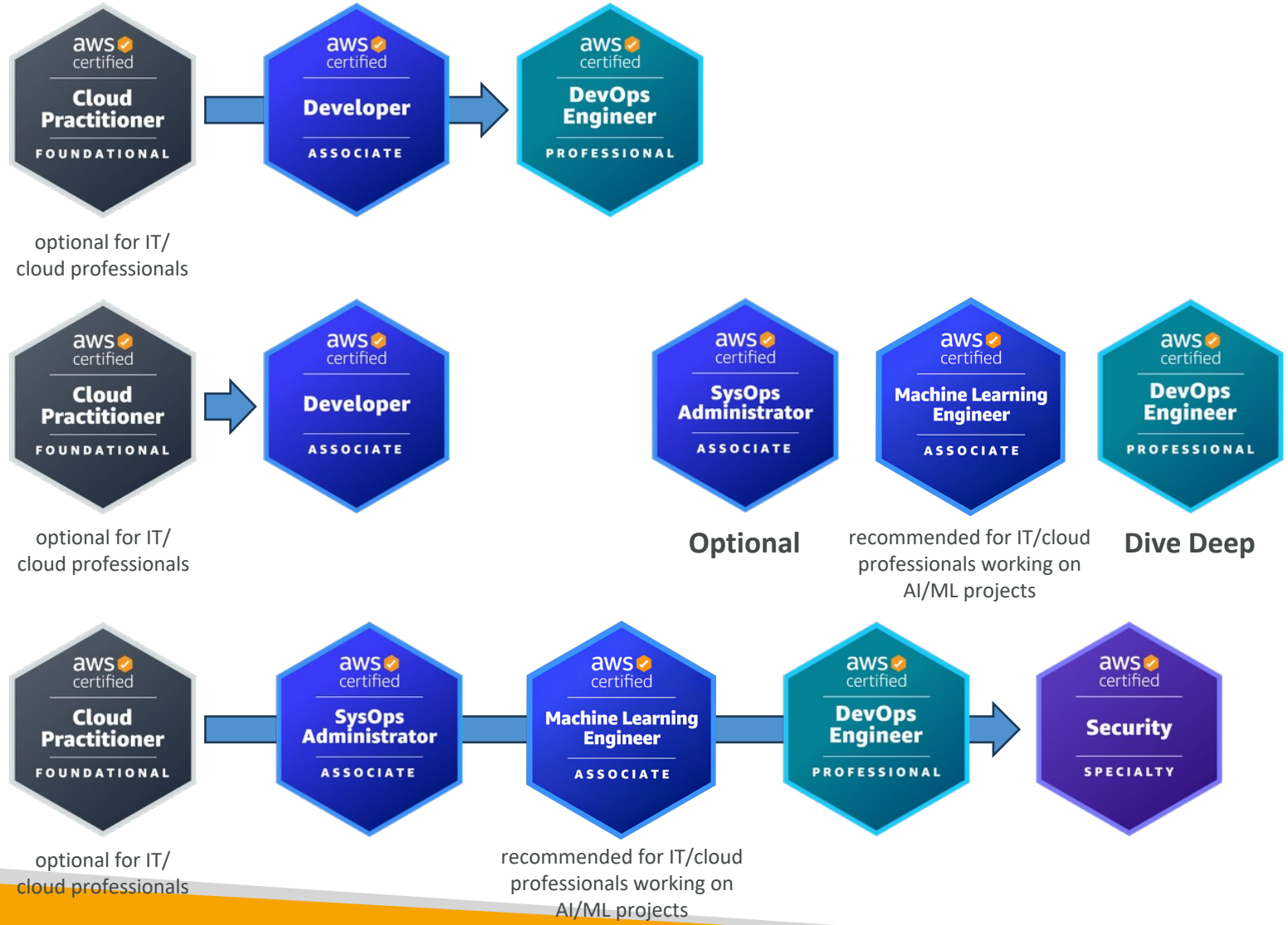
Dive Deep

AWS Certification Paths – DevOps

DevOps
Test Engineer
Embed testing and quality best practices for software development from design to release, throughout the product life cycle

DevOps
Cloud DevOps Engineer
Design, deployment, and operations of large-scale global hybrid cloud computing environment, advocating for end-to-end automated CI/CD DevOps pipelines

DevOps
DevSecOps Engineer
Accelerate enterprise cloud adoption while enabling rapid and stable delivery of capabilities using CI/CD principles, methodologies, and technologies



AWS Certification Paths – Security

Security

Cloud Security Engineer

Design computer security architecture and develop detailed cyber security designs. Develop, execute, and track performance of security measures to protect information

optional for IT/ cloud professionals

recommended for IT/cloud professionals to secure AI/ML systems

Dive Deep

Security

Cloud Security Architect

Design and implement enterprise cloud solutions applying governance to identify, communicate, and minimize business and technical risks

optional for IT/ cloud professionals

recommended for IT/cloud professionals to secure AI/ML systems

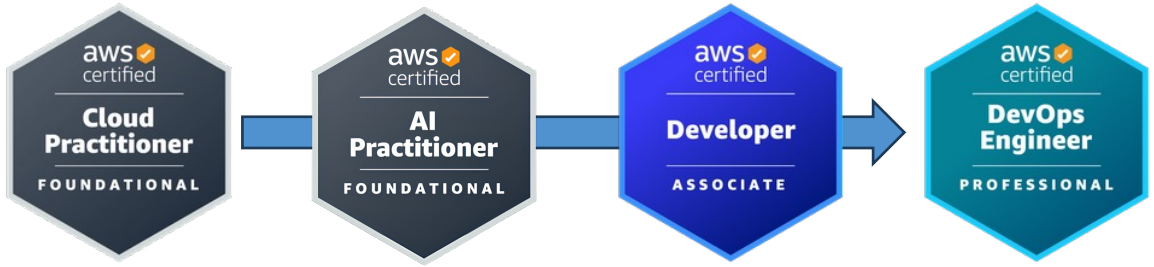
Dive Deep

AWS Certification Paths – Development & Networking

Development

Software Development Engineer

Develop, construct, and maintain software across platforms and devices



optional for IT/ cloud professionals

recommended for IT/cloud professionals to leverage AI

Networking

Network Engineer

Design and implement computer and information networks, such as local area networks (LAN), wide area networks (WAN), intranets, extranets, etc.

optional for IT/ cloud professionals

Dive Deep

AWS Certification Paths – Data Analytics & AI/ML

Data Analytics

Cloud Data Engineer

Automate collection and processing of structured/semi-structured data and monitor data pipeline performance

optional for IT/ cloud professionals

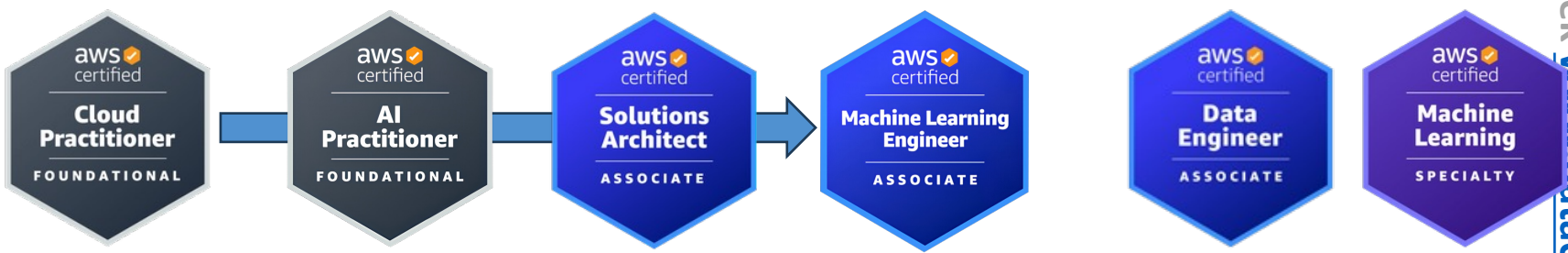
recommended for IT/cloud professionals working on AI/ML projects

Dive Deep

AI/ML

Machine Learning Engineer

Research, build, and design artificial intelligence (AI) systems to automate predictive models, and design machine learning systems, models, and schemes



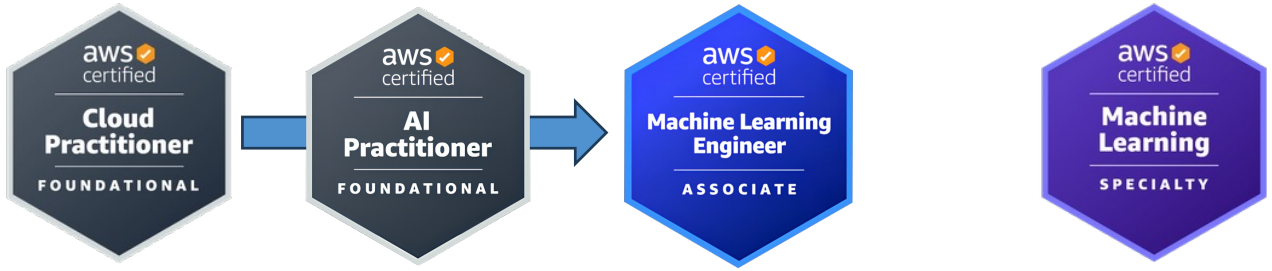
optional for IT/ cloud professionals

optional for AI/ML professionals

Dive Deep

AWS Certification Paths – AI/ML

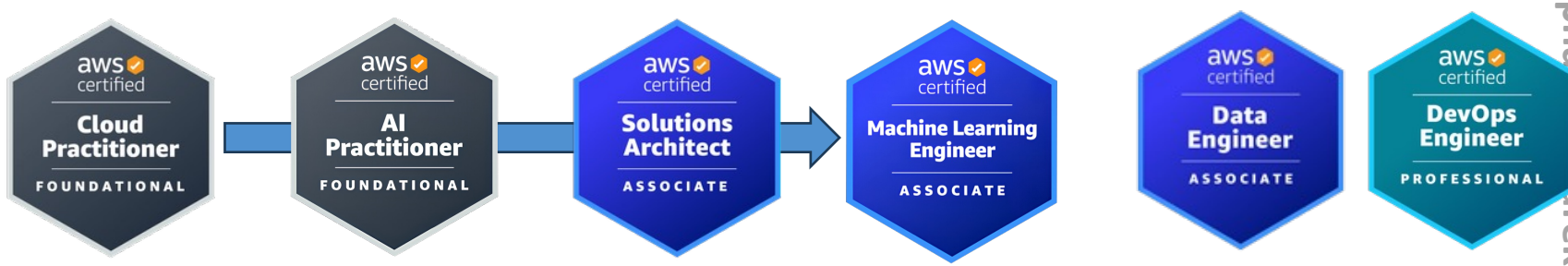
AI/ML
Prompt Engineer
Design, test, and refine text prompts to optimize the performance of AI language models



optional for IT/ cloud professionals

Dive Deep

AI/ML
Machine Learning Ops Engineer
Build and maintain AI and ML platforms and infrastructure. Design, implement, and operationally support AI/ML model activity and deployment infrastructure



optional for IT/ cloud professionals

optional for AI/ML professionals

AI/ML
Data Scientist
Develop and maintain AI/ML models to solve business problems. Train and fine tune models and evaluate their performance

optional for IT/ cloud professionals

optional for AI/ML professionals

Congratulations & Next Steps!

Congratulations!

- Congrats on finishing the course!
- I hope you will pass the exam without a hitch ☺

- If you passed, I'll be more than happy to know I've helped
  - Post it in the Q&A to help & motivate other students. Share your tips!
  - Post it on LinkedIn and tag me!

- Overall, I hope you learned how to use AWS and that you will be a tremendously good AWS Developer

Next Steps

- We've spent a lot of time getting an overview of each service
- Each service on its own deserves its own course and study time
- Find out what services you liked and get specialized in them!
- My personal favorites: AWS Lambda, CloudFormation, EC2 & ECS
- Happy learning!