

# 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](mailto:piracy@datacumulus.com). Thanks!
- Best of luck for the exam and happy learning!

# AWS Certified Developer Associate Course

## DVA-C01

# AWS Certified Developer Certification

- We're going to prepare the AWS Certified Developer Certification
- It's the most difficult associate-level certification
- It's the most personally rewarding
- We'll need to cover over 25 AWS Services
- You'll be a great developer after this course
- Let's have some fun!

# 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

# My certification: 98.4%

## AWS Certified Developer - Associate (Released June 2018)

### Notice of Exam Results

Candidate: Stephane Maarek	Exam Date: September 18, 2018
Candidate ID: AWS00614912	Registration Number: 328291
Candidate Score: 984	Pass/Fail: PASS

# About me

- I'm Stephane!
- Working as an IT consultant and AWS Developer & Solution's Architect
- Worked with AWS many years: built websites, apps, streaming platforms
- Veteran Udemy instructor on AWS (CloudFormation, Lambda, EC2...)
- You can find me on
  - GitHub: <https://github.com/simplesteph>
  - LinkedIn: <https://www.linkedin.com/in/stephanemaarek>
  - Medium: <https://medium.com/@stephane.maarek>
  - Twitter: <https://twitter.com/stephanemaarek>



4.6 Instructor Rating  
 10,917 Reviews  
 36,154 Students  
 14 Courses

# Advice for this course

- Take notes!
  - Practice what I teach at your work if possible
  - Set a target date for when you want to pass the exam
- 

- If you feel overwhelmed, take breaks
- Re-watch lectures if I went too quickly
- You can speed up / slow down the videos on the Udemy Video Player
- Enable the subtitles if you can't understand my English
- Take as much time as you need

# Format of this course

- Theory lectures with slides and diagrams
- Hands-on lectures to solidify learnings
- Quizzes at the end of every section
- Practice exam at the end of the course
- Professional subtitles for every lecture

# What we'll learn in this course



Amazon EC2



Amazon ECR



Amazon ECS



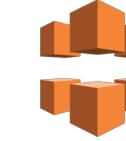
AWS Elastic Beanstalk



AWS Lambda



Elastic Load Balancing



Amazon CloudFront



Amazon Kinesis



Amazon Route 53



Amazon S3



Amazon RDS



Amazon DynamoDB



Amazon DynamoDB Accelerator



Amazon ElastiCache



Amazon SQS



Amazon SNS



AWS Step Functions



Amazon SWF



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



# The AWS developer journey

## Fundamentals for 3-tier web apps



IAM



Amazon EC2



Elastic Load  
Balancing



Amazon  
Route 53



Amazon  
Route 53



Amazon  
RDS



Amazon  
ElastiCache



Amazon  
S3

4 HOURS

# The AWS developer journey

## Developer Tools

1 HOUR



AWS CLI



Python (boto)



Node.js



IAM

# The AWS developer journey

## Deploying Properly & Continuously with Monitoring and Infrastructure as Code

4.5 HOURS



AWS Elastic  
Beanstalk



AWS  
CodeCommit



AWS  
CodeBuild



AWS  
CodeDeploy



AWS  
CodePipeline



AWS  
CloudFormation



Amazon  
CloudWatch



AWS  
CloudTrail



AWS  
X-Ray

# The AWS developer journey

## Application decoupling & integration

1.5 HOURS



# The AWS developer journey

## Serverless paradigm

5 HOURS



AWS  
Lambda



Amazon  
DynamoDB



Amazon DynamoDB  
Accelerator



Amazon API  
Gateway



Amazon  
Cognito



SAM

# The AWS developer journey

## Always Secure

1 HOUR



IAM



Amazon EC2  
Systems Manager



AWS KMS



Encryption SDK

# The AWS developer journey

## Overview of other services

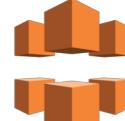
0.5 HOURS



Amazon ECR



Amazon ECS



Amazon  
CloudFront



AWS Step Functions



Amazon  
SWF



Amazon  
SES

# The AWS Certified Developer Exam (DVA-C01)

- Will test your AWS knowledge from a developer perspective
- Two question types:
  - **Multiple-choice:** Has one correct response and three incorrect responses
  - **Multiple-response:** Has two or more correct responses out of five+ options
- There is no penalty for guessing. Answer all questions!
- Pass or fail exam: score at least 720 out of 1000
- Read the exam guide!  
<https://aws.amazon.com/certification/certified-developer-associate/>

# Exam Details

Domain	% of Examination
Domain 1: Deployment	22%
Domain 2: Security	26%
Domain 3: Development with AWS Services	30%
Domain 4: Refactoring	10%
Domain 5: Monitoring and Troubleshooting	12%
<b>TOTAL</b>	<b>100%</b>

- **Deployment:** CI/CD, Beanstalk, Serverless
- **Security:** each service deep-dive + dedicated section
- **Development with AWS Services:** Serverless, API, SDK & CLI
- **Refactoring:** Understand all the AWS services for the best migration
- **Monitoring and Troubleshooting:** CloudWatch, CloudTrail, X-Ray

# AWS Fundamentals – Part I

Regions, IAM & EC2

# AWS Regions

- AWS has regions all around the world (us-east-1)
- Each region has availability zones (us-east-1a, us-east-1b...)
- Each availability zone is a physical data center in the region, but separate from the other ones (so that they're isolated from disasters)
- AWS Consoles are region scoped (except IAM, S3 & Route53)

AWS Region  
(Sydney: ap-southeast-2)

ap-southeast-2a

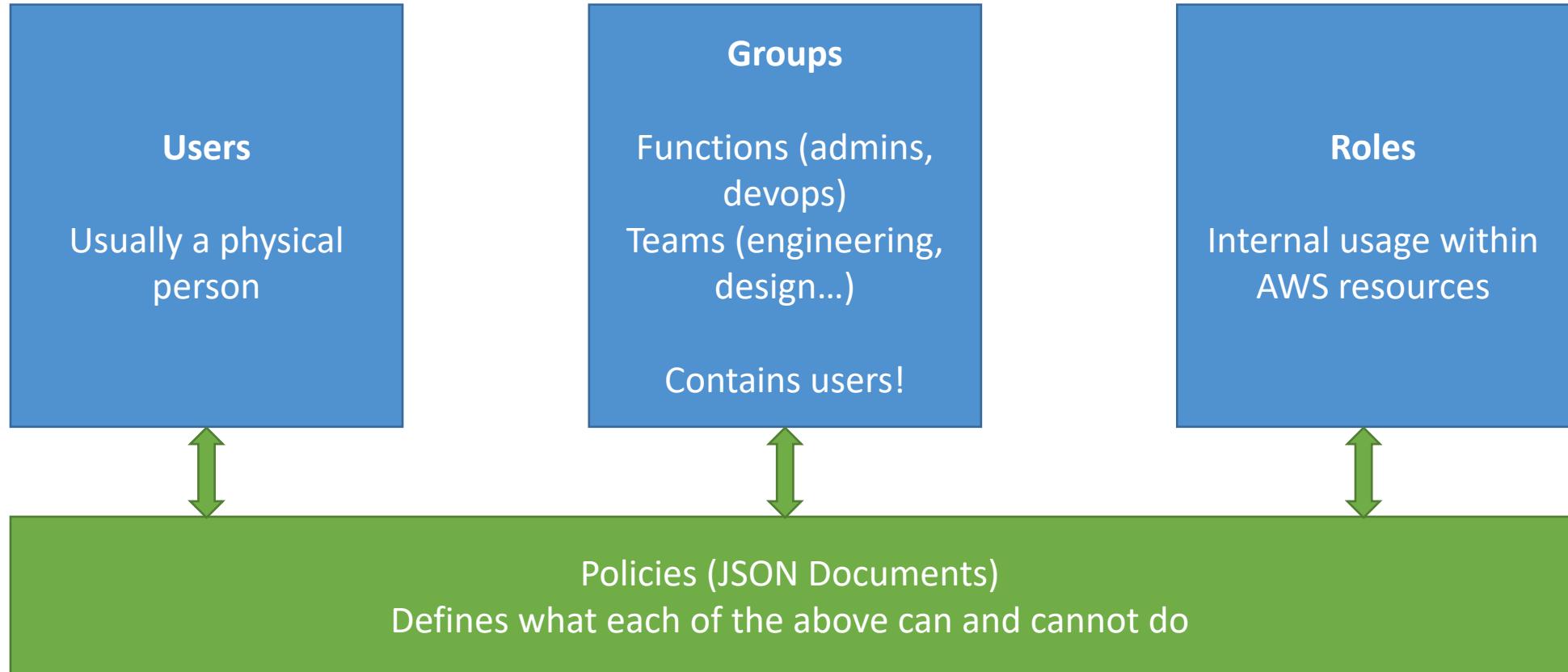
ap-southeast-2b

ap-southeast-2c

# IAM Introduction

- IAM (Identity and Access Management)
- Your whole AWS security is there:
  - Users
  - Groups
  - Roles
- Root account should never be used (and shared)
- Users must be created with proper permissions
- IAM is at the center of AWS
- Policies are written in JSON (JavaScript Object Notation)

# IAM Introduction



# IAM Introduction

- IAM has a **global** view
- Permissions are governed by Policies (JSON)
- MFA (Multi Factor Authentication) can be setup
- IAM has predefined “managed policies”
- We’ll see IAM policies in details in the future
- It’s best to give users the minimal amount of permissions they need to perform their job (least privilege principles)

# IAM Federation

- Big enterprises usually integrate their own repository of users with IAM
- This way, one can login into AWS using their company credentials
- Identity Federation uses the SAML standard (Active Directory)

# IAM 101 Brain Dump

- One IAM User per PHYSICAL PERSON
- One IAM Role per Application
- IAM credentials should NEVER BE SHARED
- Never, ever, ever, ever, write IAM credentials in code. EVER.
- And even less, NEVER EVER EVER COMMIT YOUR IAM credentials
- Never use the ROOT account except for initial setup.
- Never use ROOT IAM Credentials

# What is EC2?

- EC2 is one of most popular of AWS offering
- 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



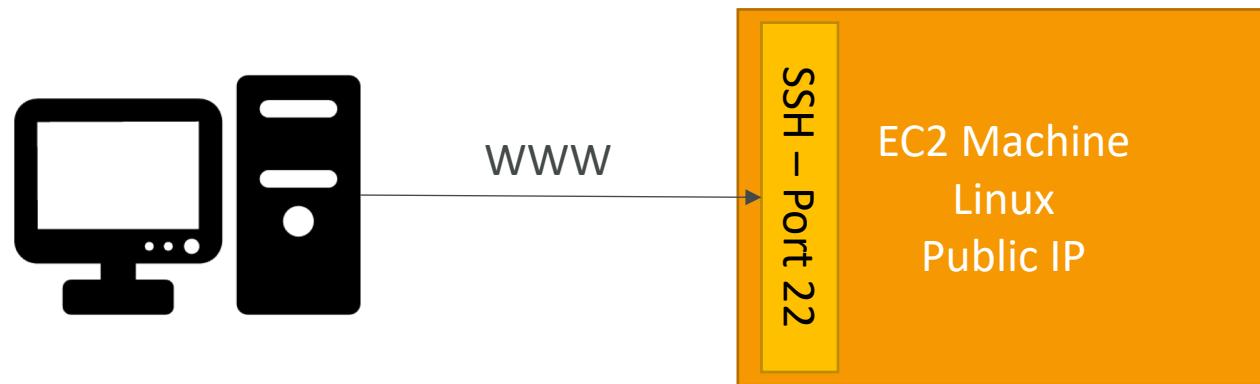
# Hands-On: Launching an EC2 Instance running Linux

- 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 learn how to start / stop / terminate our instance.

# 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.

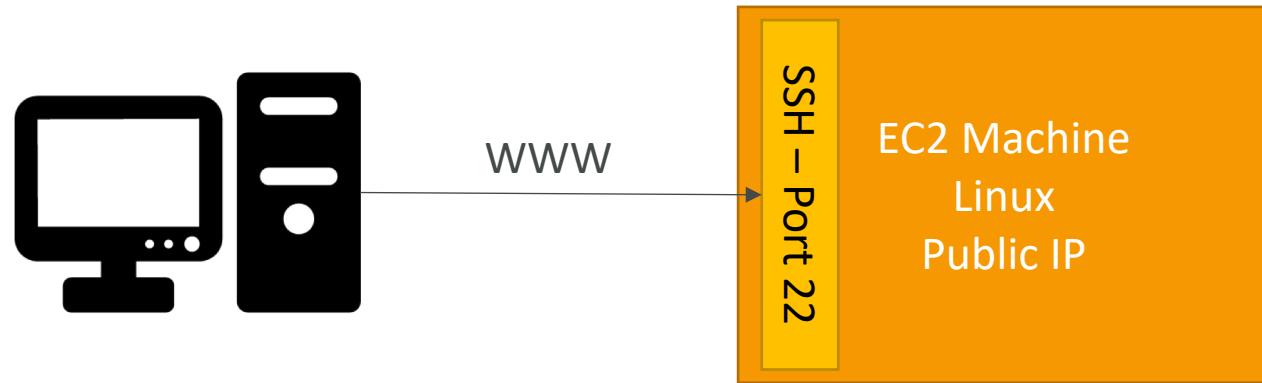


- We will see how we can configure OpenSSH `~/.ssh/config` to facilitate the SSH into our EC2 instances

# How to SSH into your EC2 Instance

## Windows

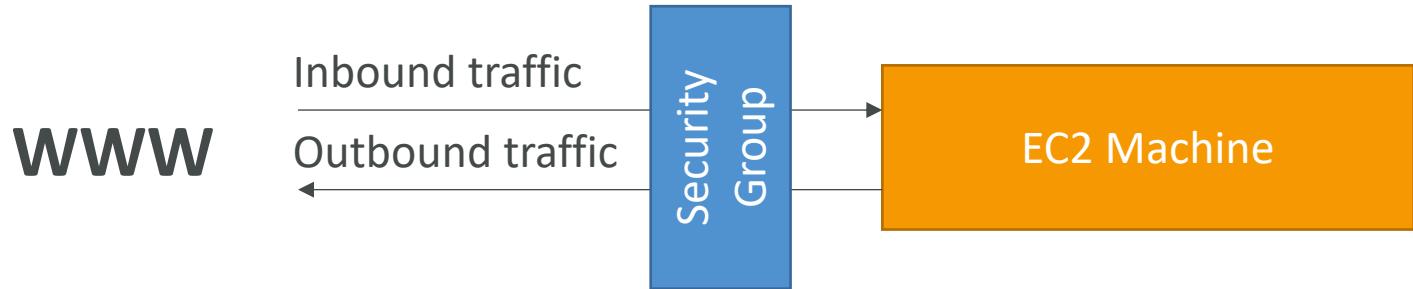
- We'll learn how to SSH into your EC2 instance using [Windows](#)
- SSH is one of the most important function. It allows you to control a remote machine, all using the command line.



- We will configure all the required parameters necessary for doing SSH on Windows using the free tool [Putty](#).

# 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 Machines.



- It is the most fundamental skill to learn to troubleshoot networking issues
- In this lecture, we'll learn how to use them to **allow**, **inbound** and **outbound** ports

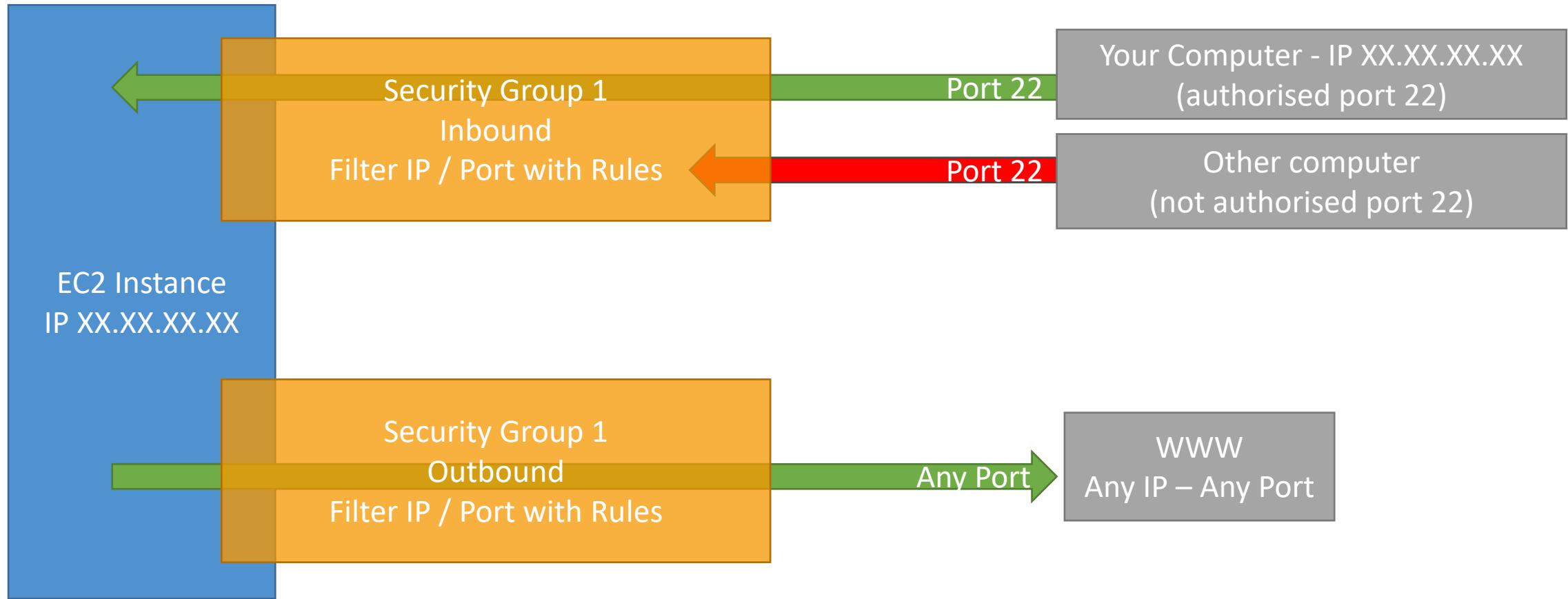
# Security Groups

## 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



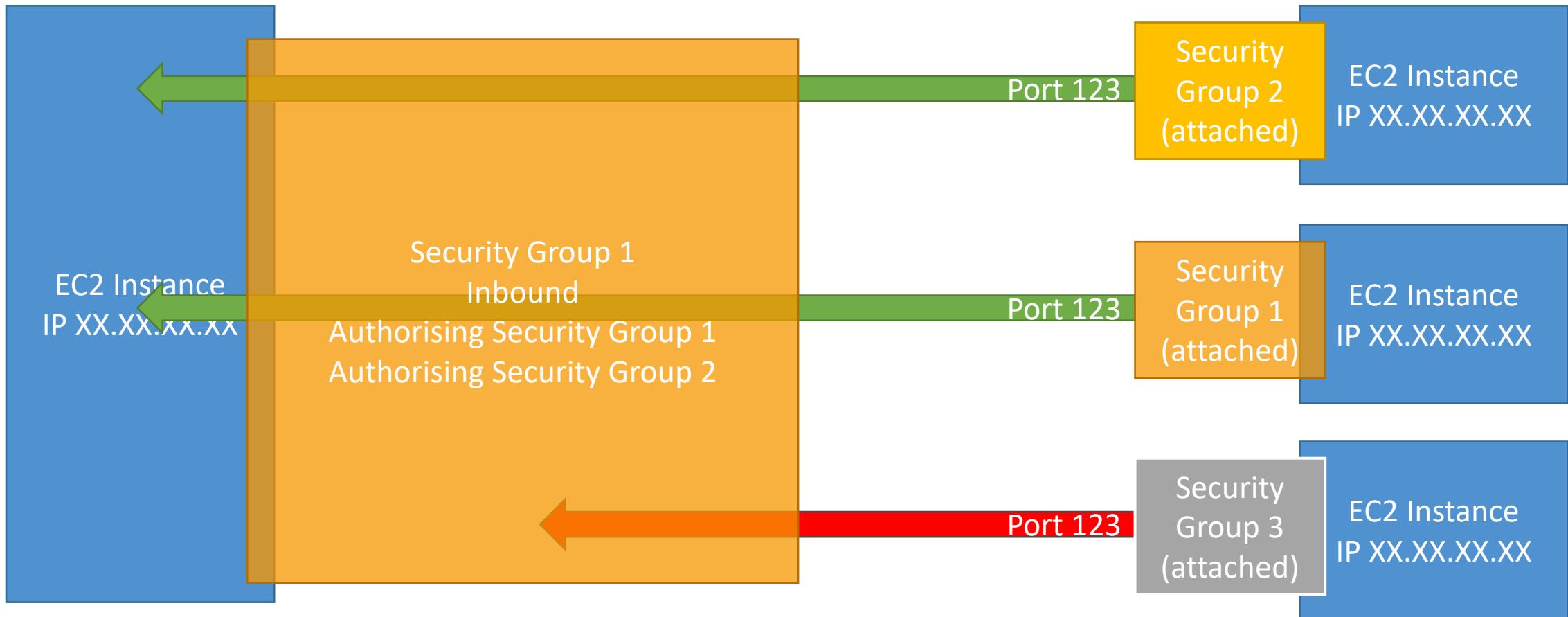
# Security Groups

## 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

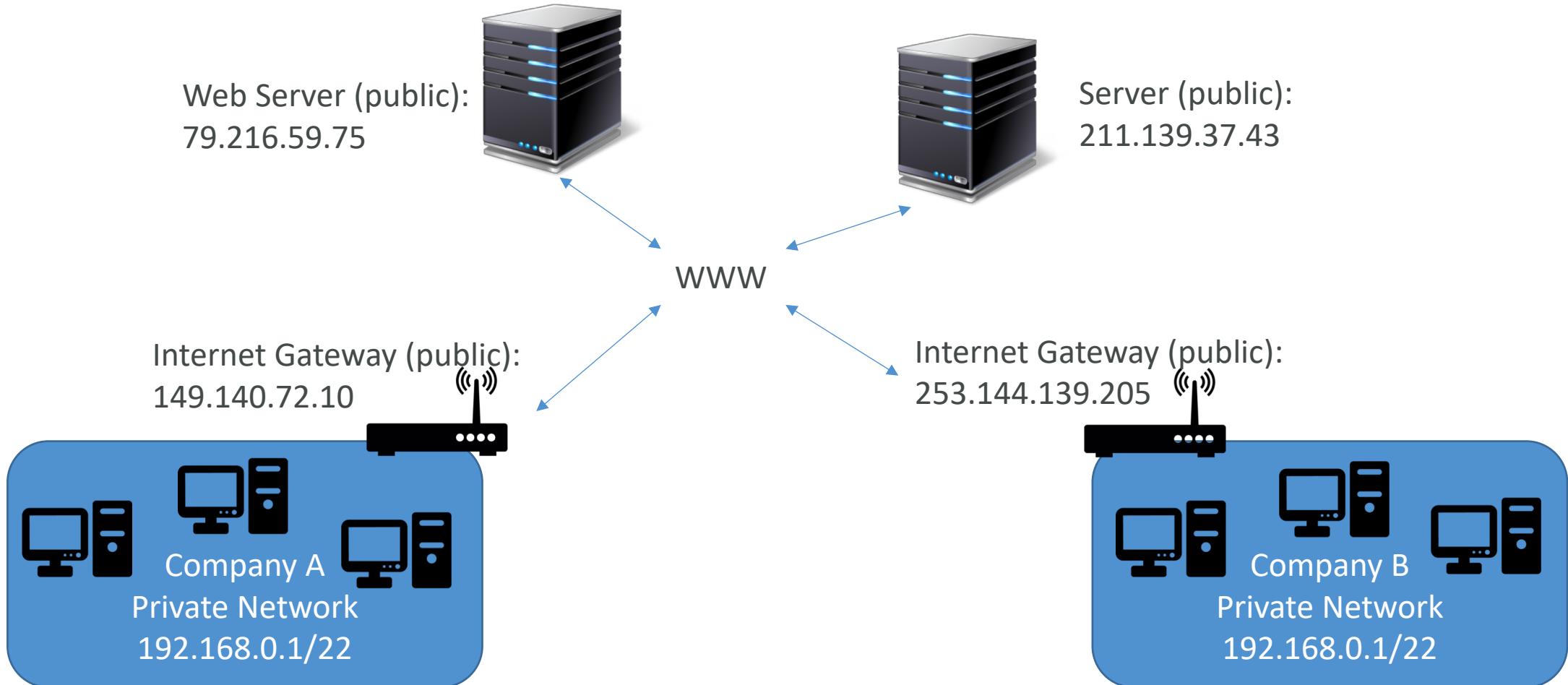
## Diagram



# Private vs Public IP (IPv4)

- Networking has two sorts of IPs. IPv4 and IPv6:
  - IPv4: **1.160.10.240**
  - IPv6: 3ffe: **1900:4545:3:200:f8ff:fe21:67cf**
- In this course, we will only be using IPv4.
- IPv4 is still the most common format used online.
- IPv6 is newer and solves problems for the Internet of Things (IoT).
- IPv4 allows for **3.7 billion** different addresses in the public space
- IPv4: [0-255].[0-255].[0-255].[0-255].

# Private vs Public IP (IPv4) Example



# Private vs Public IP (IPv4)

## Fundamental Differences

- Public IP:
  - Public IP means the machine can be identified on the internet (WWW)
  - Must be unique across the whole web (not two machines can have the same public IP).
  - Can be geo-located easily
- Private IP:
  - Private IP means the machine can only be identified on a private network only
  - The IP must be unique across the private network
  - BUT two different private networks (two companies) can have the same IPs.
  - Machines connect to WWW using an internet gateway (a proxy)
  - Only a specified range of IPs can be used as private IP

# Elastic IPs

- When you stop and then start an EC2 instance, it can change its public IP.
- If you need to have a fixed public IP for your instance, you need an Elastic IP
- An Elastic IP is a public IPv4 IP you own as long as you don't delete it
- You can attach it to one instance at a time

# Elastic IP

- With an Elastic IP address, you can mask the failure of an instance or software by rapidly remapping the address to another instance in your account.
- You can only have 5 Elastic IP in your account (you can ask AWS to increase that).
- Overall, [try to avoid using Elastic IP](#):
  - They often reflect poor architectural decisions
  - Instead, use a random public IP and register a DNS name to it
  - Or, as we'll see later, use a Load Balancer and don't use a public IP

# Private vs Public IP (IPv4)

## In AWS EC2 – Hands On

- By default, your EC2 machine comes with:
  - A private IP for the internal AWS Network
  - A public IP for the WWW.
- When we are doing SSH into our EC2 machines:
  - We can't use a private IP, because we are not in the same network
  - We can only use the public IP.
- If your machine is stopped and then started,  
**the public IP can change**

# Launching an Apache Server on EC2

- Let's leverage our EC2 instance
- We'll install an Apache Web Server to display a web page
- We'll create an index.html that shows the hostname of our machine

# 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

# EC2 User Data Hands-On

- We want to make sure that this EC2 instance has an Apache HTTP server installed on it – to display a simple web page
- For it, we are going to write a user-data script.
- This script will be executed at the first boot of the instance.
  
- Let's get hands on!

# EC2 Instance Launch Types

- On Demand Instances: short workload, predictable pricing
- Reserved Instances: long workloads ( $\geq 1$  year)
- Convertible Reserved Instances: long workloads with flexible instances
- Scheduled Reserved Instances: launch within time window you reserve
- Spot Instances: short workloads, for cheap, can lose instances
- Dedicated Instances: no other customers will share your hardware
- Dedicated Hosts: book an entire physical server, control instance placement

# EC2 On Demand

- Pay for what you use (billing per second, after the first minute)
  - 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 75% discount compared to On-demand
- Pay upfront for what you use with long term commitment
- Reservation period can be 1 or 3 years
- Reserve a specific instance type
- Recommended for steady state usage applications (think database)
- **Convertible Reserved Instance**
  - can change the EC2 instance type
  - Up to 54% discount
- **Scheduled Reserved Instances**
  - launch within time window you reserve
  - When you require a fraction of day / week / month

# EC2 Spot Instances

- Can get a discount of up to 90% compared to On-demand
- You bid a price and get the instance as long as its under the price
- Price varies based on offer and demand
- Spot instances are reclaimed with a 2 minute notification warning when the spot price goes above your bid
- Used for batch jobs, Big Data analysis, or workloads that are resilient to failures.
- Not great for critical jobs or databases

# EC2 Dedicated Hosts

- Physical dedicated EC2 server for your use
  - Full control of EC2 Instance placement
  - Visibility into the underlying sockets / physical cores of the hardware
  - Allocated for your account for a 3 year period reservation
  - More expensive
- 
- 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 running 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

# Which host 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.
- **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

# EC2 Pricing

- EC2 instances prices (per hour) varies based on these parameters:
  - Region you're in
  - Instance Type you're using
  - On-Demand vs Spot vs Reserved vs Dedicated Host
  - Linux vs Windows vs Private OS (RHEL, SLES, Windows SQL)
- You are billed by the second, with a minimum of 60 seconds.
- You also pay for other factors such as storage, data transfer, fixed IP public addresses, load balancing
- **You do not pay for the instance if the instance is stopped**

# EC2 Pricing Example

- t2.small in US-EAST-1 (VIRGINIA), cost \$0.023 per Hour
- If used for:
  - 6 seconds, it costs  $\$0.023/60 = \$0.000383$  (minimum of 60 seconds)
  - 60 seconds, it costs  $\$0.023/60 = \$0.000383$  (minimum of 60 seconds)
  - 30 minutes, it costs  $\$0.023/2 = \$0.0115$
  - 1 month, it costs  $\$0.023 * 24 * 30 = \$16.56$  (assuming a month is 30 days)
  - X seconds ( $X > 60$ ), it costs  $\$0.023 * X / 3600$
- The best way to know the pricing is to consult the pricing page:  
<https://aws.amazon.com/ec2/pricing/on-demand/>

# What's an AMI?

- As we saw, AWS comes with base images such as:
  - Ubuntu
  - Fedora
  - RedHat
  - Windows
  - Etc...
- These images can be customised at runtime using EC2 User data
- But what if we could create our own image, ready to go?
- That's an AMI – an image to use to create our instances
- AMIs can be built for Linux or Windows machines

# Why would you use a custom AMI?

- Using a custom built AMI can provide the following advantages:
  - Pre-installed packages needed
  - Faster boot time (no need for long ec2 user data at boot time)
  - Machine comes configured with monitoring / enterprise software
  - Security concerns – control over the machines in the network
  - Control of maintenance and updates of AMIs over time
  - Active Directory Integration out of the box
  - Installing your app ahead of time (for faster deploys when auto-scaling)
  - Using someone else's AMI that is optimised for running an app, DB, etc...
- AMI are built for a specific AWS region (!)

# EC2 Instances Overview

- Instances have 5 distinct characteristics advertised on the website:
  - The RAM (type, amount, generation)
  - The CPU (type, make, frequency, generation, number of cores)
  - The I/O (disk performance, EBS optimisations)
  - The Network (network bandwidth, network latency)
  - The Graphical Processing Unit (GPU)
- It may be daunting to choose the right instance type (there are over 50 of them) -  
<https://aws.amazon.com/ec2/instance-types/>
- <https://ec2instances.info/> can help with summarizing the types of instances
- R/C/P/G/H/X/I/F/Z/CR are specialised in RAM, CPU, I/O, Network, GPU
- M instance types are balanced
- T2/T3 instance types are “burstable”

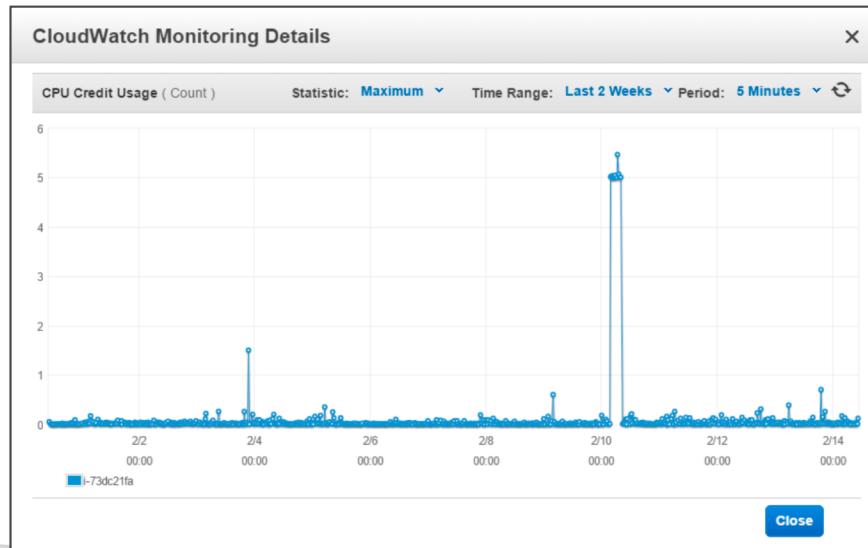
# Burstable Instances (T2)

- AWS has the concept of burstable instances (T2 machines)
- Burst means that overall, the instance has OK CPU performance.
- When the machine needs to process something unexpected (a spike in load for example), it can burst, and CPU can be VERY good.
- If the machine bursts, it utilizes “burst credits”
- If all the credits are gone, the CPU becomes BAD
- If the machine stops bursting, credits are accumulated over time

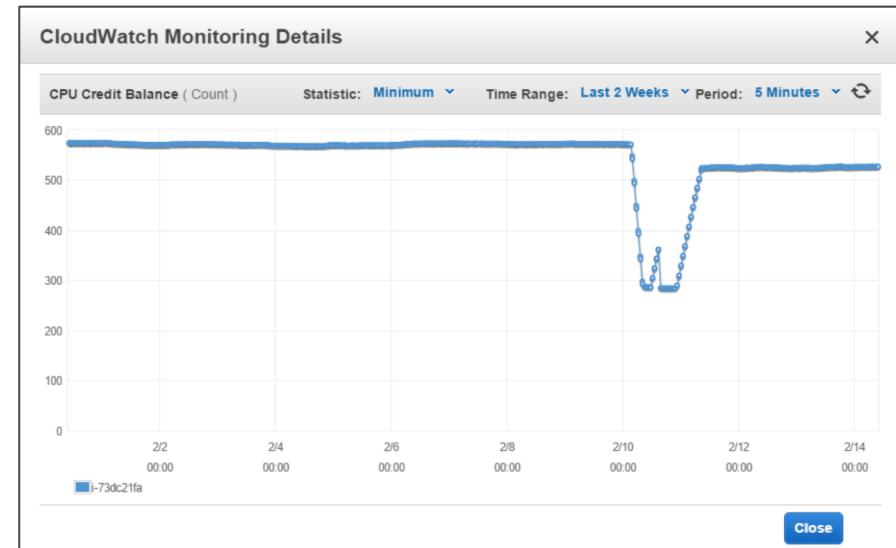
# Burstable Instances (T2)

- Burstable instances can be amazing to handle unexpected traffic and getting the insurance that it will be handled correctly
- If your instance consistently runs low on credit, you need to move to a different kind of non-burstable instance (all the ones described before).

Credit usage



Credit balance



# CPU Credits

Instance type	Launch credits	vCPUs	CPU credits earned per hour	Maximum earned CPU credit balance	vCPUs	Baseline performance (% CPU utilization)
t2.nano	30	1	3	72	1	5%
t2.micro	30	1	6	144	1	10%
t2.small	30	1	12	288	1	20%
t2.medium	60	2	24	576	2	40% (of 200% max)*
t2.large	60	2	36	864	2	60% (of 200% max)*
t2.xlarge	120	4	54	1296	4	90% (of 400% max)*
t2.2xlarge	240	8	81	1944	8	135% (of 800% max)*

# T2 Unlimited

- Nov 2017: It is possible to have an “unlimited burst credit balance”
- You pay extra money if you go over your credit balance, but you don’t lose in performance
- Overall, it is a new offering, so be careful, costs could go high if you’re not monitoring the health of your instances
- Read more here: <https://aws.amazon.com/blogs/aws/new-t2-unlimited-going-beyond-the-burst-with-high-performance/>

# EC2 – Checklist

- Know how to SSH into EC2 (and change .pem file permissions)
- Know how to properly use security groups
- Know the fundamental differences between private vs public vs elastic IP
- Know how to use User Data to customize your instance at boot time
- Know that you can build custom AMI to enhance your OS
- EC2 instances are billed by the second and can be easily created and thrown away, welcome to the cloud!

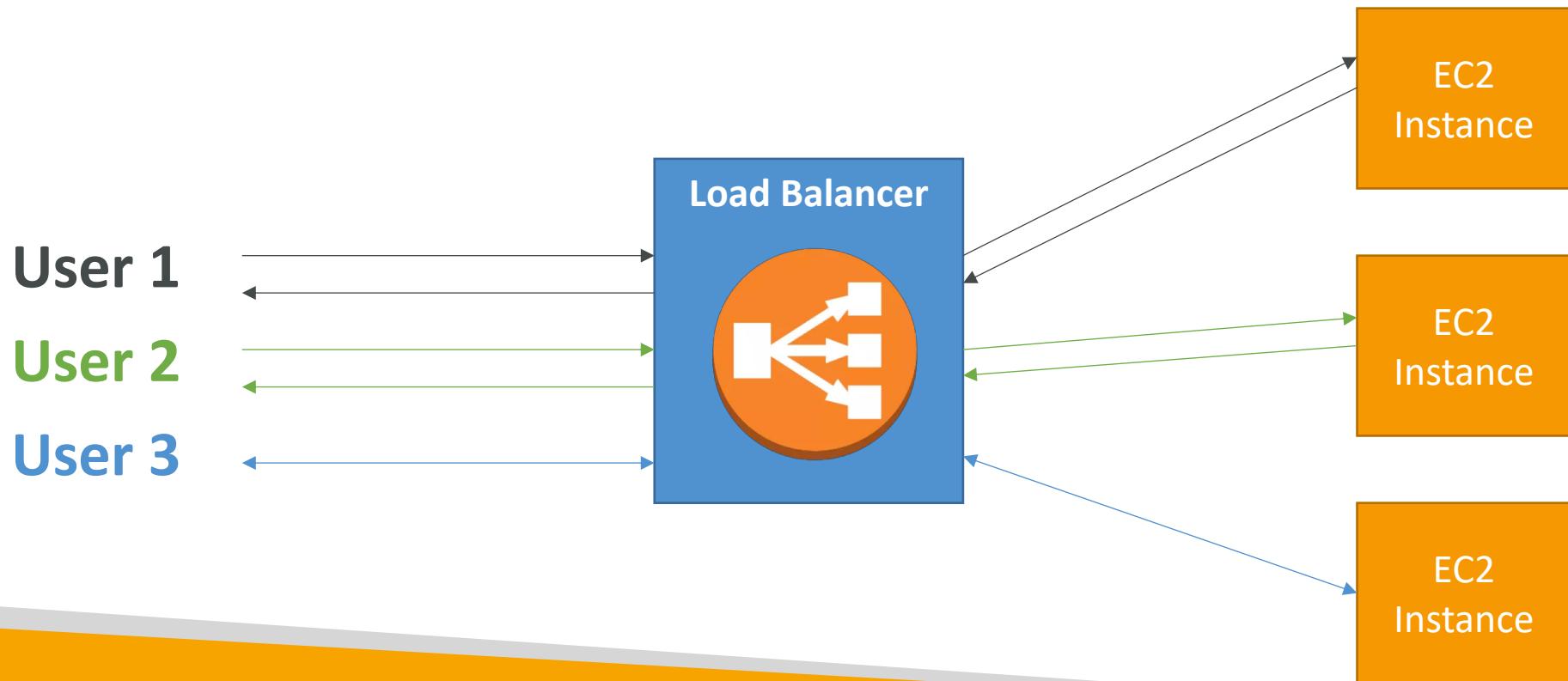
# AWS Fundamentals – Part II

Load Balancing, Auto Scaling Groups and EBS Volumes



# What is load balancing?

- Load balancers are servers that forward internet traffic to multiple servers (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 EC2 Load Balancer?

- An ELB (EC2 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

# Types of load balancer on AWS

- AWS has **3 kinds of Load Balancers**
- Classic Load Balancer (v1 - old generation) - 2009
- Application Load Balancer (v2 - new generation) - 2016
- Network Load Balancer (v2 - new generation) - 2017
- Overall, it is recommended to use the newer / v2 generation load balancers as they provide more features
- You can setup **internal** (private) or **external** (public) ELBs

# 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

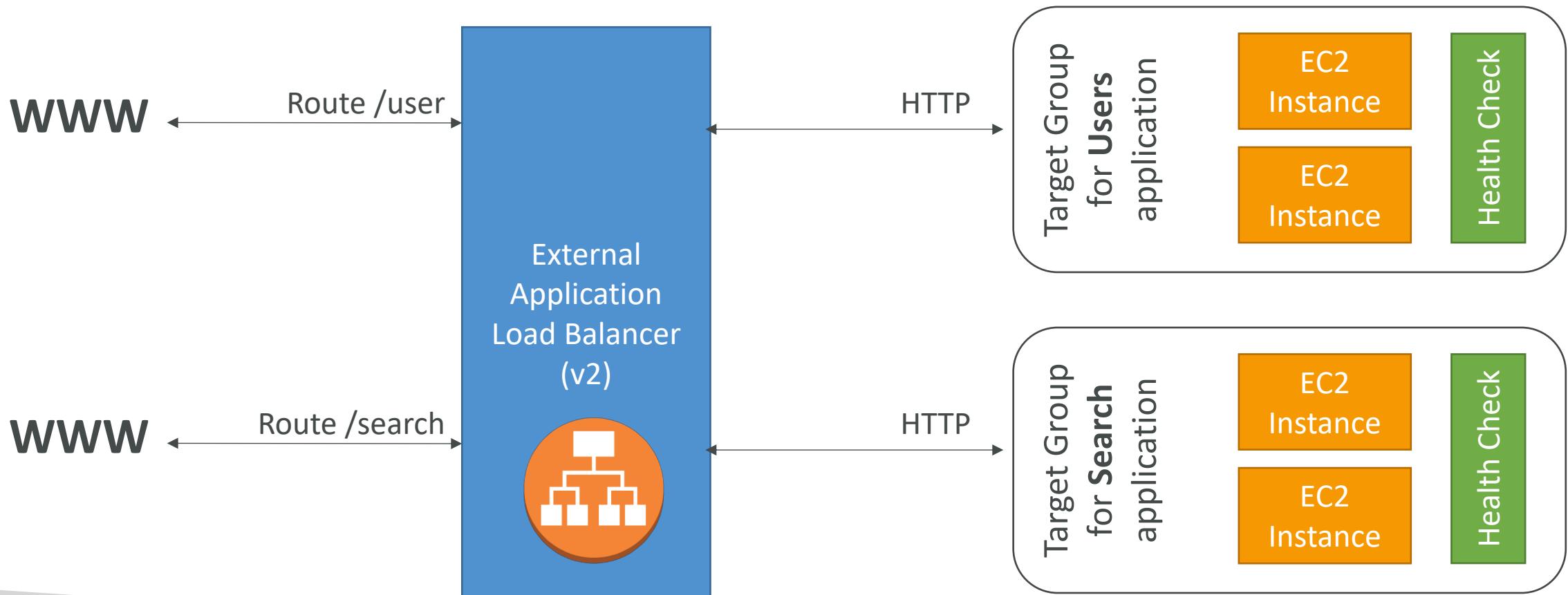


# Application Load Balancer (v2)

- Application load balancers (Layer 7) allow to do:
  - Load balancing to multiple HTTP applications across machines (target groups)
  - Load balancing to multiple applications on the same machine (ex: containers)
  - Load balancing based on route in URL
  - Load balancing based on hostname in URL
- Basically, they're awesome for micro services & container-based application (example: Docker & Amazon ECS)
- Has a port mapping feature to redirect to a dynamic port
- In comparison, we would need to create one Classic Load Balancer per application before. That was very expensive and inefficient!

# Application Load Balancer (v2)

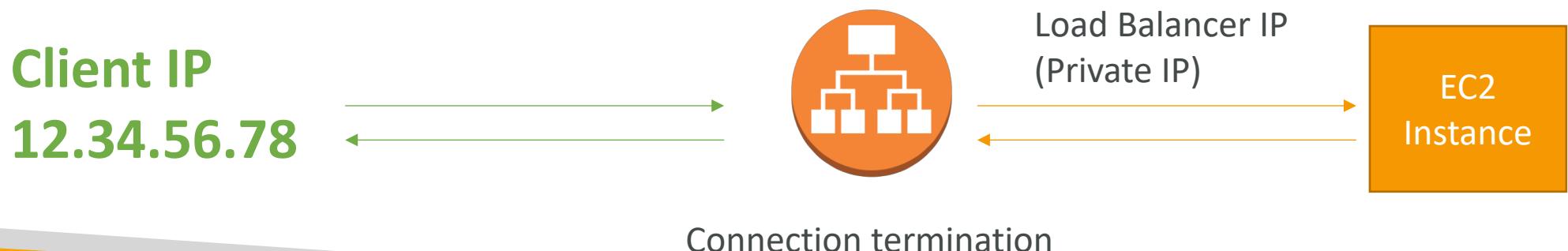
## HTTP Based Traffic



# Application Load Balancer v2

## Good to Know

- Stickiness can be enabled at the target group level
  - Same request goes to the same instance
  - Stickiness is directly generated by the ALB (not the application)
- ALB support HTTP/HTTPS & Websockets protocols
- 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**)

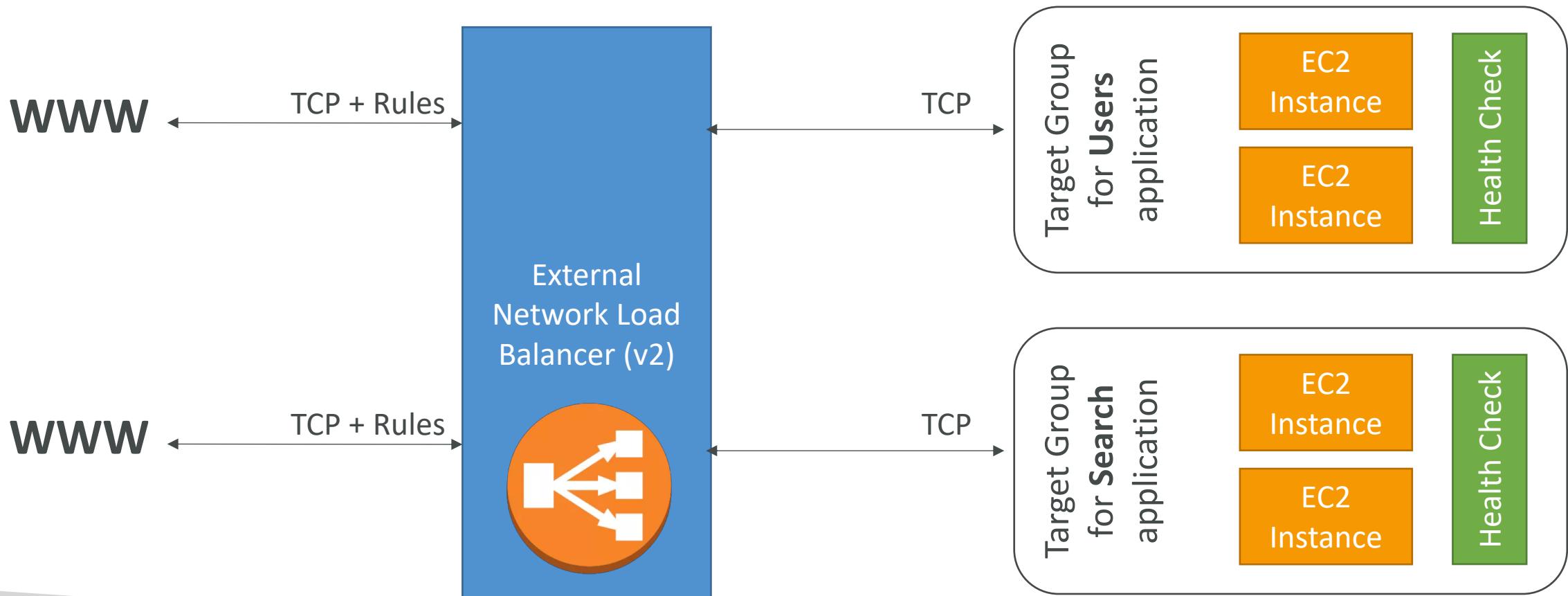


# Network Load Balancer (v2)

- Network load balancers (Layer 4) allow to do:
  - Forward TCP traffic to your instances
  - Handle millions of requests per second
  - Support for static IP or elastic IP
  - Less latency ~100 ms (vs 400 ms for ALB)
- Network Load Balancers are mostly used for extreme performance and should not be the default load balancer you choose
- Overall, the creation process is the same as Application Load Balancers

# Network Load Balancer (v2)

## TCP Based Traffic

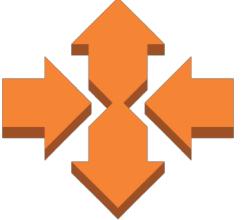


# Load Balancer Good to Know

- Classic Load Balancers are Deprecated
  - Application Load Balancers for HTTP / HTTPS & Websocket
  - Network Load Balancer for TCP
- CLB, ALB, NLB support SSL certificates and provide SSL termination
- All Load Balancers have health check capability
- ALB can route on based on hostname / path
- ALB is a great fit with ECS (Docker)

# Load Balancer Good to Know

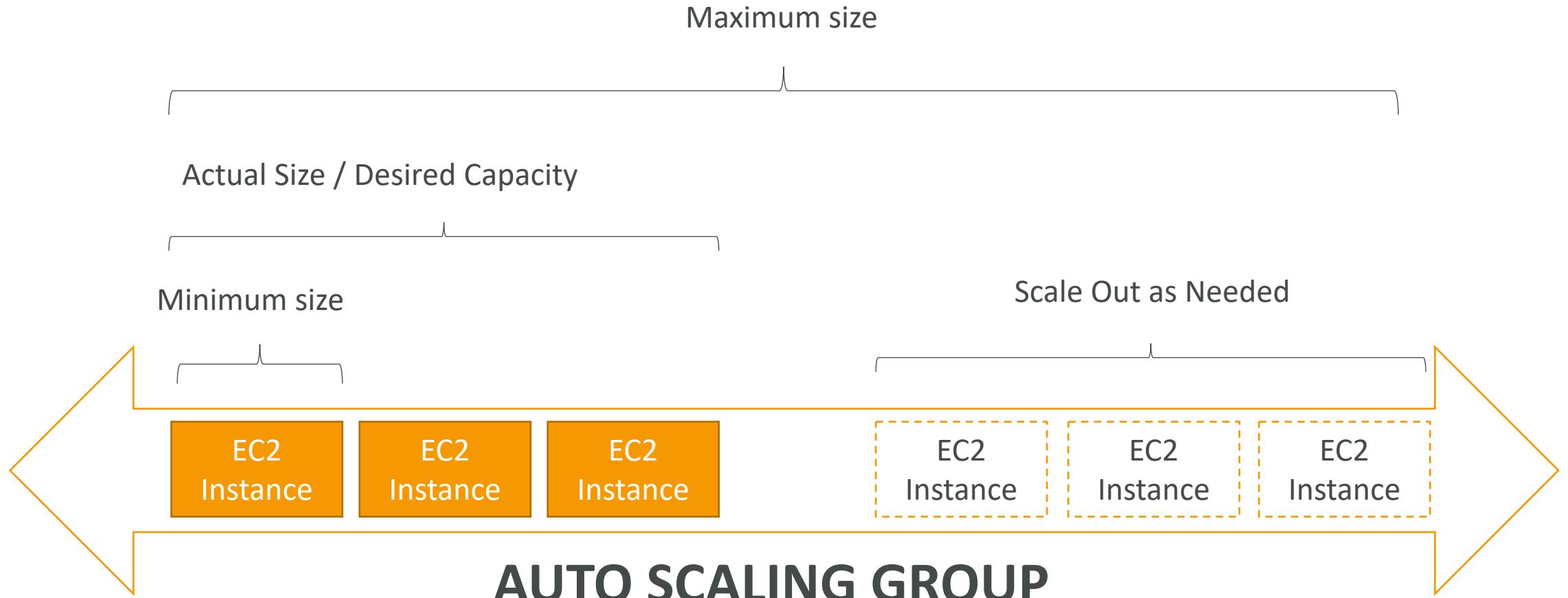
- Any Load Balancer (CLB, ALB, NLB) has a static host name. Do not resolve and use underlying IP
- LBs can scale but not instantaneously – contact AWS for a “warm-up”
- NLB directly see the client IP
- 4xx errors are client induced errors
- 5xx errors are application induced errors
  - Load Balancer Errors 503 means at capacity or no registered target
- If the LB can't connect to your application, check your security groups!



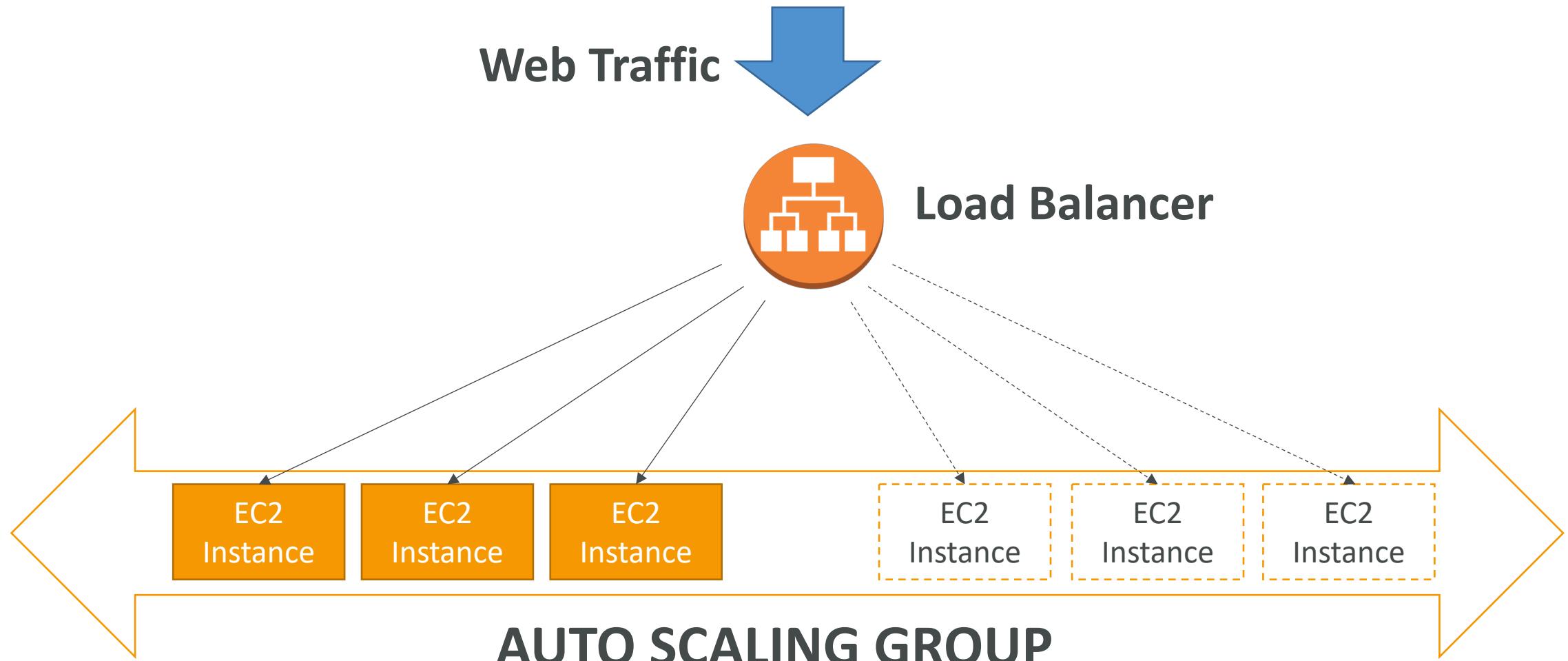
# What's an Auto Scaling Group?

- 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 machines running
  - Automatically Register new instances to a load balancer

# Auto Scaling Group in AWS



# Auto Scaling Group in AWS With Load Balancer

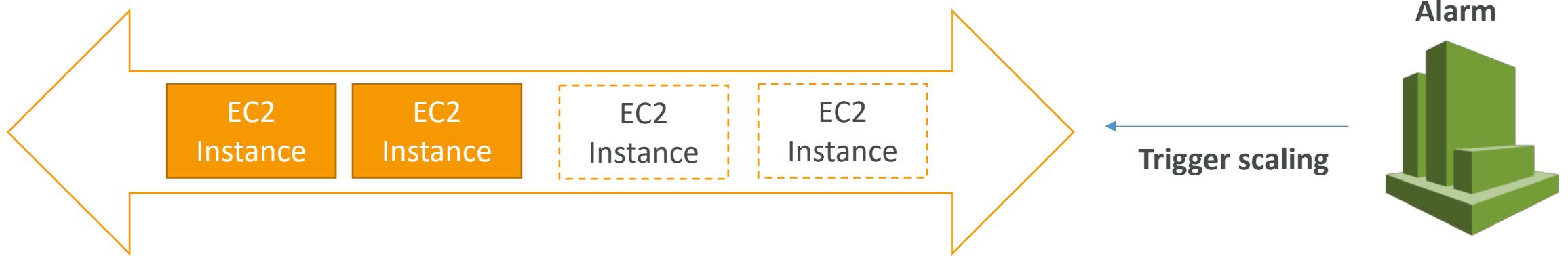


# ASGs have the following attributes

- A launch configuration
  - AMI + Instance Type
  - EC2 User Data
  - EBS Volumes
  - Security Groups
  - SSH Key Pair
- Min Size / Max Size / Initial Capacity
- Network + Subnets Information
- Load Balancer Information
- Scaling Policies

# Auto Scaling Alarms

- It is possible to scale an ASG based on CloudWatch alarms
- An Alarm monitors a metric (such as Average CPU)
- Metrics 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 New Rules

- It is now possible to define "better" auto scaling rules that are directly managed by EC2
  - Target Average CPU Usage
  - Number of requests on the ELB per instance
  - Average Network In
  - Average Network Out
- These rules are easier to set up and can make more sense

# Auto Scaling Custom Metric

- We can auto scale based on a custom metric (ex: number of connected users)
- 1. Send custom metric from application on EC2 to CloudWatch (PutMetric API)
- 2. Create CloudWatch alarm to react to low / high values
- 3. Use the CloudWatch alarm as the scaling policy for ASG

# ASG Brain dump

- Scaling policies can be on CPU, Network... and can even be on custom metrics or based on a schedule (if you know your visitors patterns)
- ASGs use Launch configurations and you update an ASG by providing a new launch configuration
- IAM roles attached to an ASG will get assigned to EC2 instances
- ASG are free. You pay for the underlying resources being launched
- Having instances under an ASG means that if they get terminated for whatever reason, the ASG will restart them. Extra safety!
- ASG can terminate instances marked as unhealthy by an LB (and hence replace them)

# What's an EBS Volume?

- An EC2 machine loses its root volume (main drive) when it is manually terminated.
- Unexpected terminations might happen from time to time (AWS would email you)
- Sometimes, you need a way to store your instance data somewhere
- 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

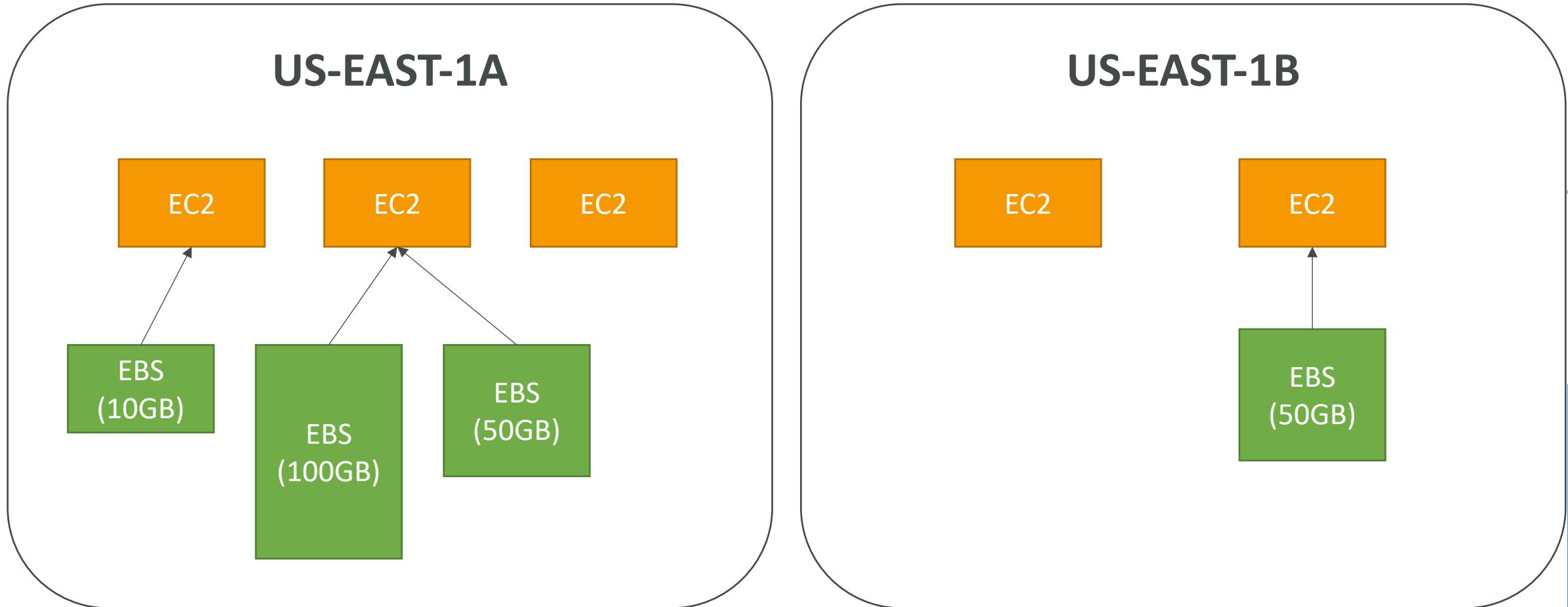


Amazon EBS

# 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 Volume Types

- EBS Volumes come in 4 types
  - **GP2 (SSD)**: General purpose SSD volume that balances price and performance for a wide variety of workloads
  - **IO1 (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
  - **SCI (HDD)**: Lowest cost HDD volume designed for less frequently accessed workloads
- EBS Volumes are characterized in Size | Throughput | IOPS
- When in doubt always consult the AWS documentation – it's good!
- We've been only using GP2 volumes so far

# EBS Volume Resizing

- Feb 2017: You can **resize** the EBS volumes
- You can only increase the EBS volumes:
  - Size (any volume type)
  - IOPS (only in IO1)
- After resizing an EBS volume, you need to repartition your drive

# EBS Snapshots

- EBS Volumes can be backed up using “snapshots”
- Snapshots only take the actual space of the blocks on the volume
- If you snapshot a 100GB drive that only has 5 GB of data, then your EBS snapshot will only be 5GB
- Snapshots are used for:
  - Backups: ensuring you can save your data in case of catastrophe
  - Volume migration:
    - Resizing a volume down
    - Changing the volume type
    - Encrypt a volume

# EBS Encryption

- When you create an encrypted EBS volume, you get the following:
  - Data at rest is encrypted inside the volume
  - All the data in flight moving between the instance and the volume is encrypted
  - All snapshots are encrypted
  - All volumes created from the snapshot
- Encryption and decryption are handled transparently (you have nothing to do)
- Encryption has a minimal impact on latency
- EBS Encryption leverages keys from KMS (AES-256)
- Copying an unencrypted snapshot allows encryption

# EBS vs Instance Store

- Some instances do not come with Root EBS volumes
- Instead, they come with “Instance Store”.
- Instance store is physically attached to the machine
- Pros:
  - Better I/O performance
- Cons:
  - On termination, the instance store is lost
  - You can't resize the instance store
  - Backups must be operated by the user
- Overall, EBS-backed instances should fit most applications workloads

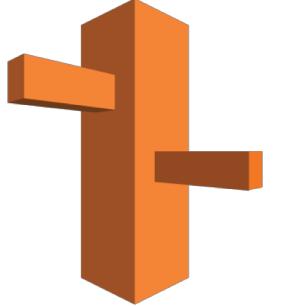
# EBS Brain Dump

- EBS can be attached to only one instance at a time
- EBS are locked at the AZ level
- Migrating an EBS volume across AZ means first backing it up (snapshot), then recreating it in the other 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)

# AWS Fundamentals – Part III

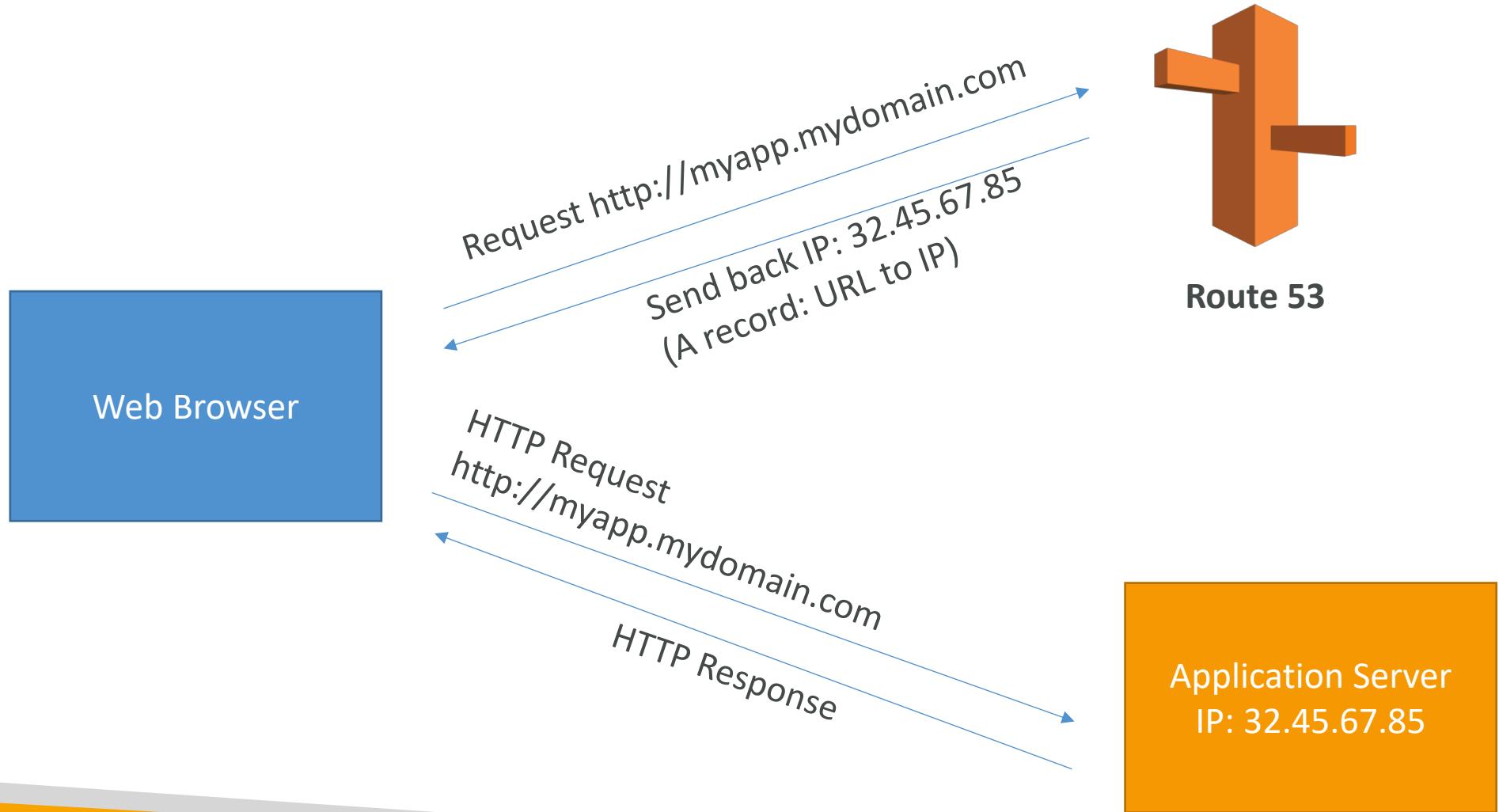
Route 53, RDS, ElastiCache and VPC

# AWS Route 53 Overview



- Route53 is a Managed DNS (Domain Name System)
- DNS is a collection of rules and records which helps clients understand how to reach a server through URLs.
- In AWS, the most common records are:
  - A: URL to IPv4
  - AAAA: URL to IPv6
  - CNAME: URL to URL
  - Alias: URL to AWS resource.

# Route 53 – Diagram for A Record

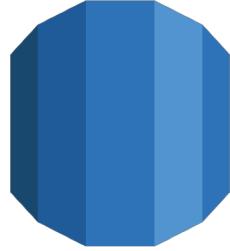


# AWS Route 53 Overview

- Route53 can use:
  - public domain names you own (or buy)  
[application1.mypublicdomain.com](http://application1.mypublicdomain.com)
  - private domain names that can be resolved by your instances in your VPCs.  
[application1.company.internal](http://application1.company.internal)
- Route53 has advanced features such as:
  - Load balancing (through DNS – also called client load balancing)
  - Health checks (although limited...)
  - Routing policy: simple, failover, geolocation, geoproximity, latency, weighted
- Prefer Alias over CNAME for AWS resources (for performance reasons)

# AWS Route 53 Overview

- Overall Route53 is not much used in the AWS Certified Developer Exam
- You should know all the record types:
  - A: URL to IPv4
  - AAAA: URL to IPv6
  - CNAME: URL to URL
  - Alias: URL to AWS resource
- You should know to use Alias records over CNAME for AWS resources



# AWS RDS Overview

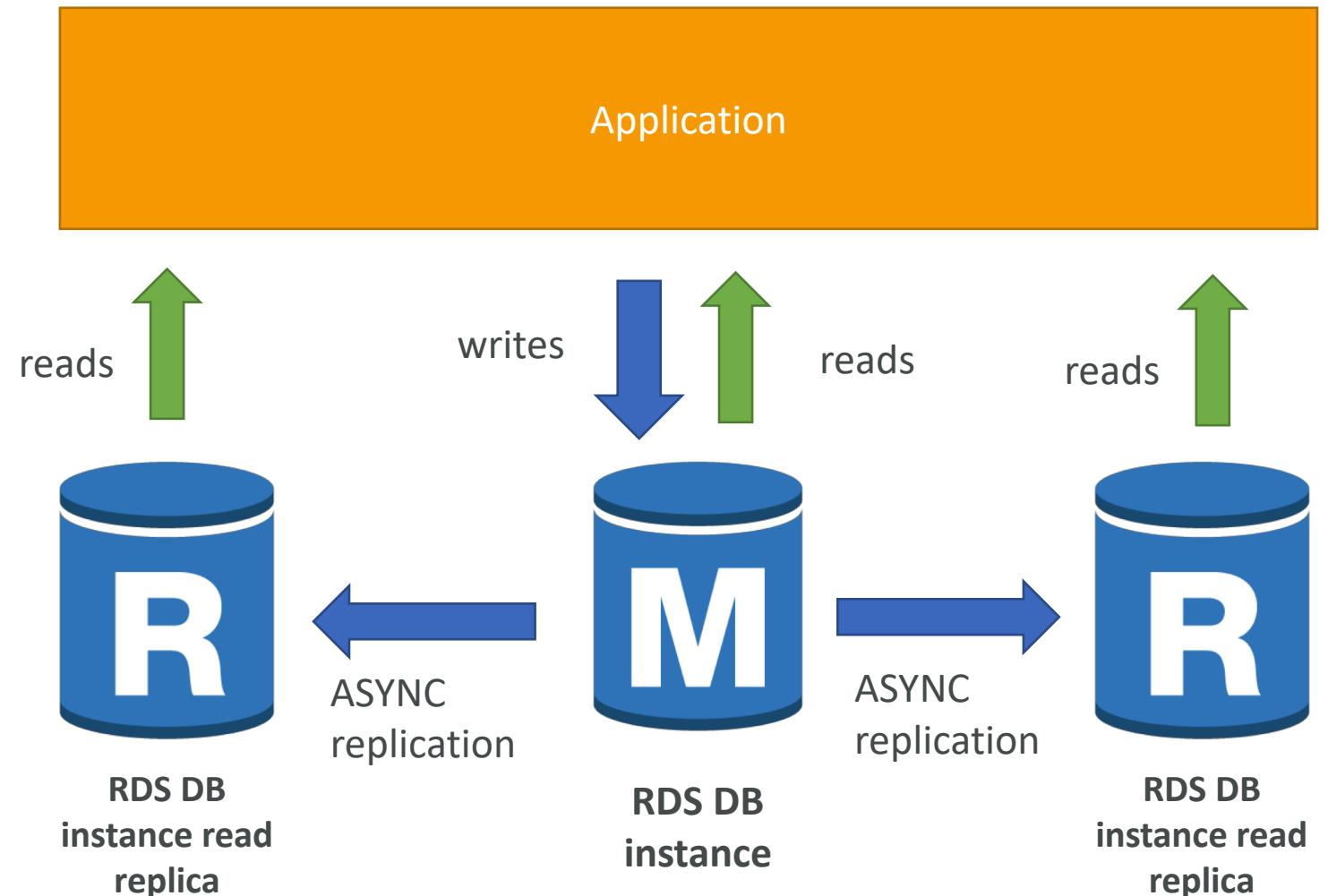
- RDS stands for Relational Database Service
- It's a managed DB service for DB use SQL as a query language.
- It allows you to create databases in the cloud that are managed by AWS
  - Postgres
  - Oracle
  - MySQL
  - MariaDB
  - Oracle
  - Microsoft SQL Server
  - Aurora (AWS Proprietary database)

# Advantage over using RDS versus deploying DB on EC2

- Managed service:
- OS patching level
- 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)
- BUT you can't SSH into your instances

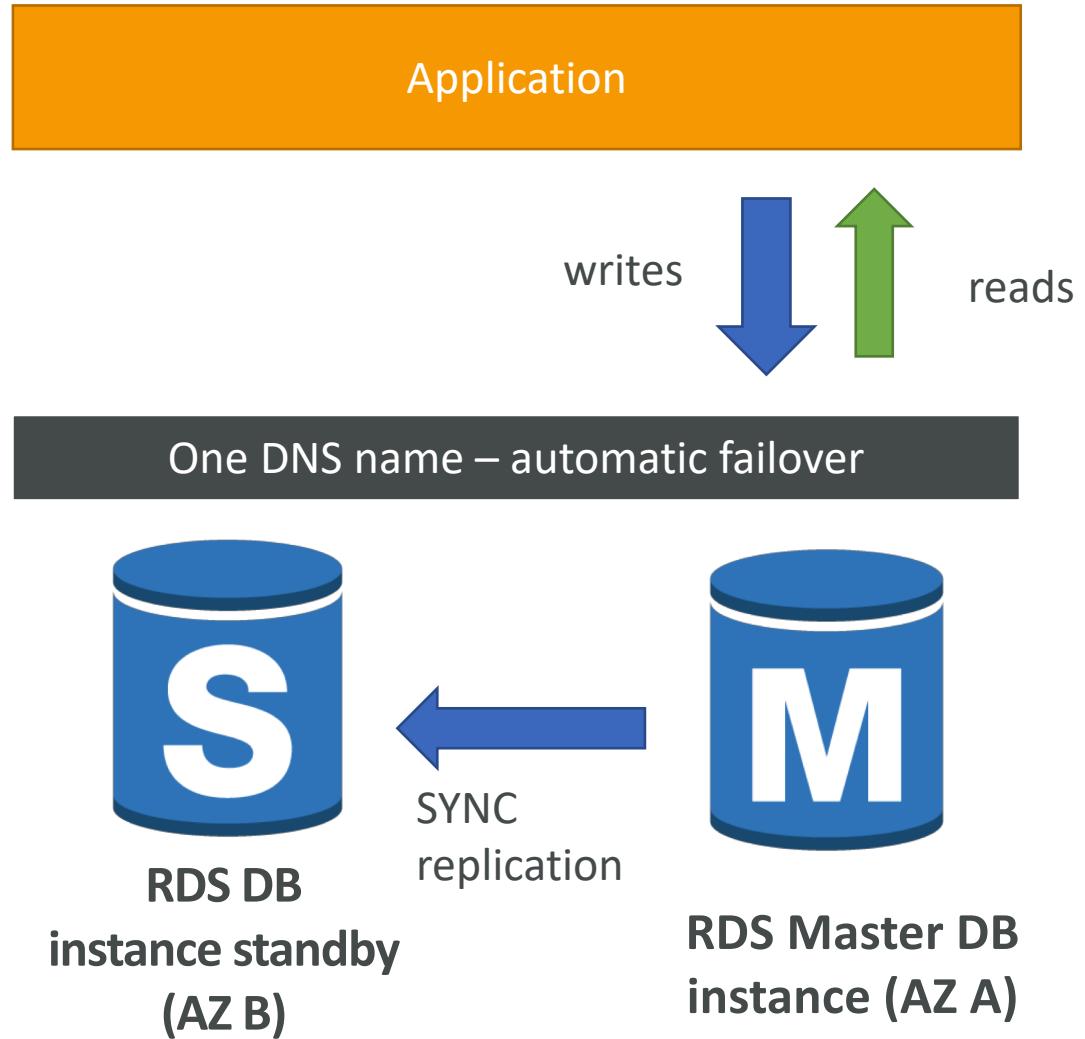
# RDS Read Replicas for read scalability

- Up to 5 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 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



# RDS Backups

- Backups are automatically enabled in RDS
- Automated backups:
  - Daily full snapshot of the database
  - Capture transaction logs in real time
  - => ability to restore to any point in time
  - 7 days retention (can be increased to 35 days)
- DB Snapshots:
  - Manually triggered by the user
  - Retention of backup for as long as you want

# RDS Encryption

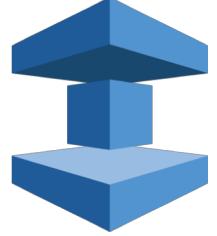
- Encryption at rest capability with AWS KMS - AES-256 encryption
- SSL certificates to encrypt data to RDS in flight
- To enforce SSL:
  - PostgreSQL: rds.force\_ssl=1 in the AWS RDS Console (Parameter Groups)
  - MySQL: Within the DB:  
GRANT USAGE ON \*.\* TO 'mysqluser'@'%' REQUIRE SSL;
- To connect using SSL:
  - Provide the SSL Trust certificate (can be download from AWS)
  - Provide SSL options when connecting to database

# RDS Security

- RDS databases are usually deployed within a private subnet, not in a public one
- RDS Security works by leveraging security groups (the same concept as for EC2 instances) – it controls who can **communicate** with RDS
- IAM policies help control who can **manage** AWS RDS
- Traditional Username and Password can be used to **login** to the database
- IAM users can now be used too (for MySQL / Aurora – **NEW!**)

# RDS vs 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 64 TB.
- Aurora can have 15 replicas while MySQL has 5, and the replication process is faster (sub 10 ms replica lag)
- Failover in Aurora is instantaneous. It’s HA native.
- Aurora costs more than RDS (20% more) – but is more efficient



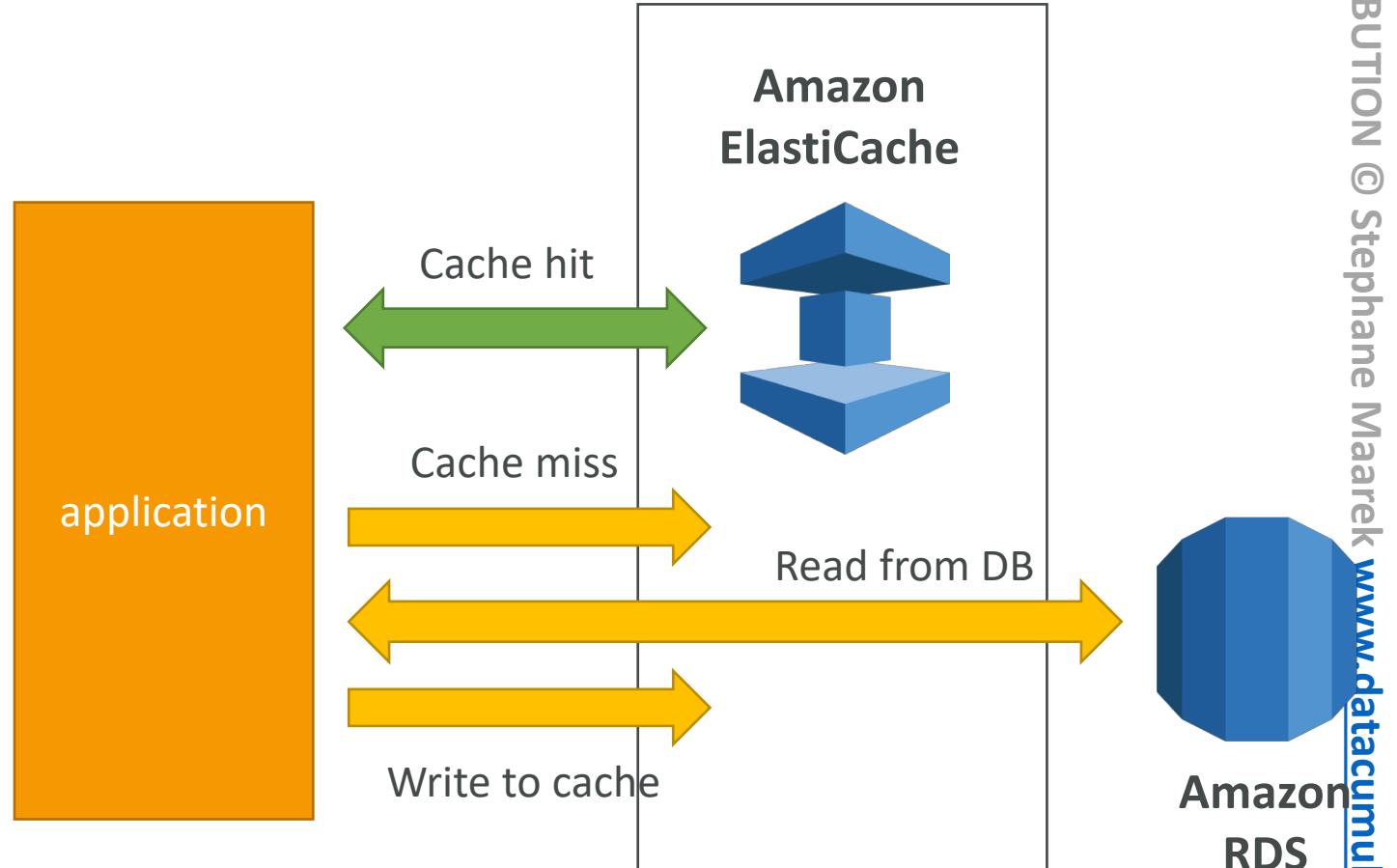
# AWS ElastiCache Overview

- 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
- Write Scaling using sharding
- Read Scaling using Read Replicas
- Multi AZ with Failover Capability
- AWS takes care of OS maintenance / patching, optimizations, setup, configuration, monitoring, failure recovery and backups

# ElastiCache

## Solution Architecture - DB Cache

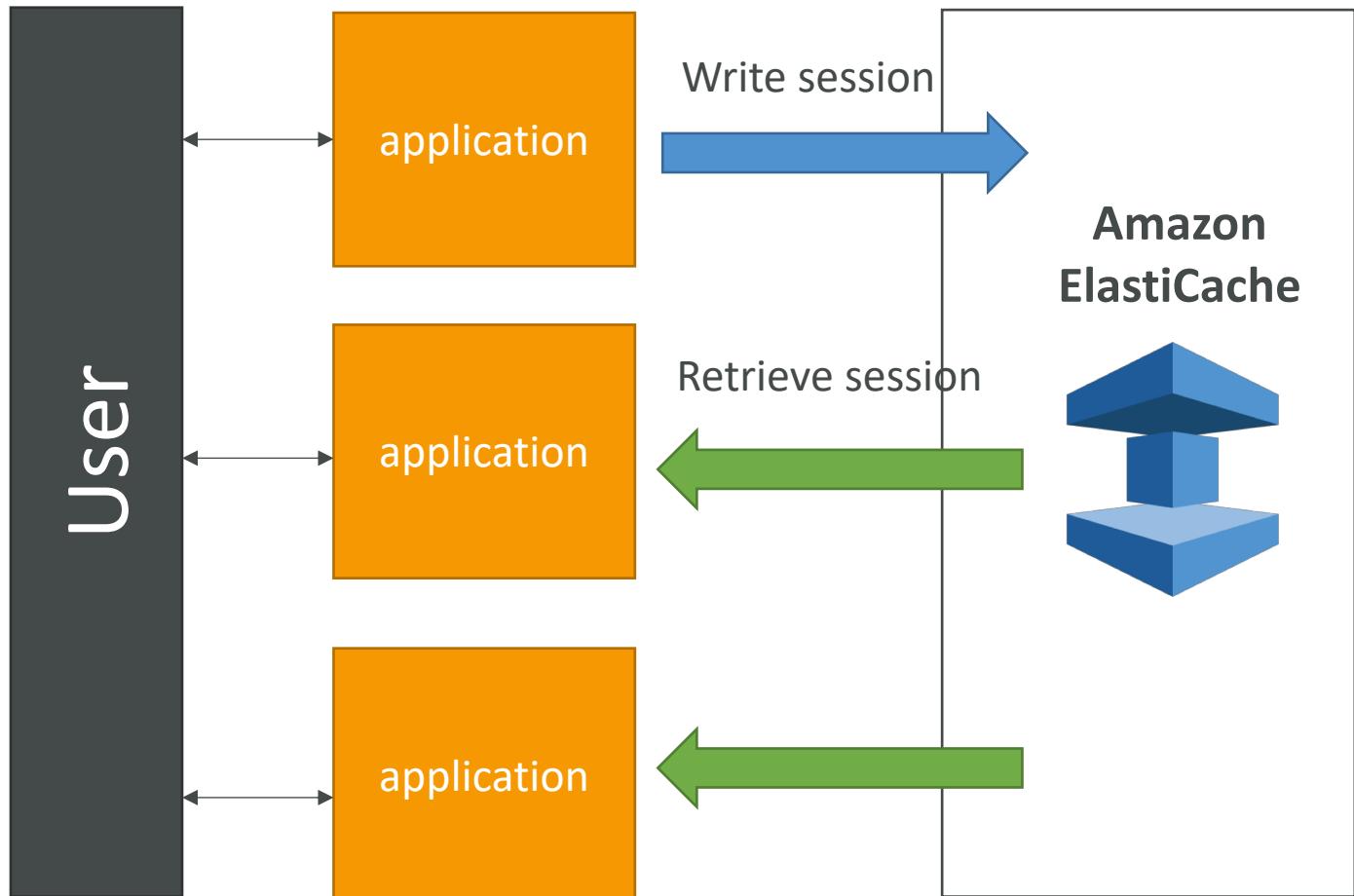
- Applications queries ElastiCache, if not available, get from RDS and store in ElastiCache.
- Helps relieve load in RDS
- Cache must have an invalidation strategy to make sure only the most current data is used in there.



# 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



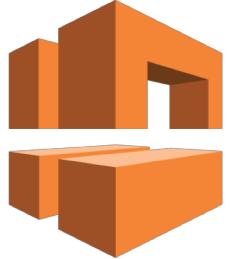
# Redis Overview

- Redis is an in-memory key-value store
- Super low latency (sub ms)
- Cache survive reboots by default (it's called persistence)
- Great to host
  - User sessions
  - Leaderboard (for gaming)
  - Distributed states
  - Relieve pressure on databases (such as RDS)
  - Pub / Sub capability for messaging
- Multi AZ with Automatic Failover for disaster recovery if you don't want to lose your cache data
- Support for Read Replicas

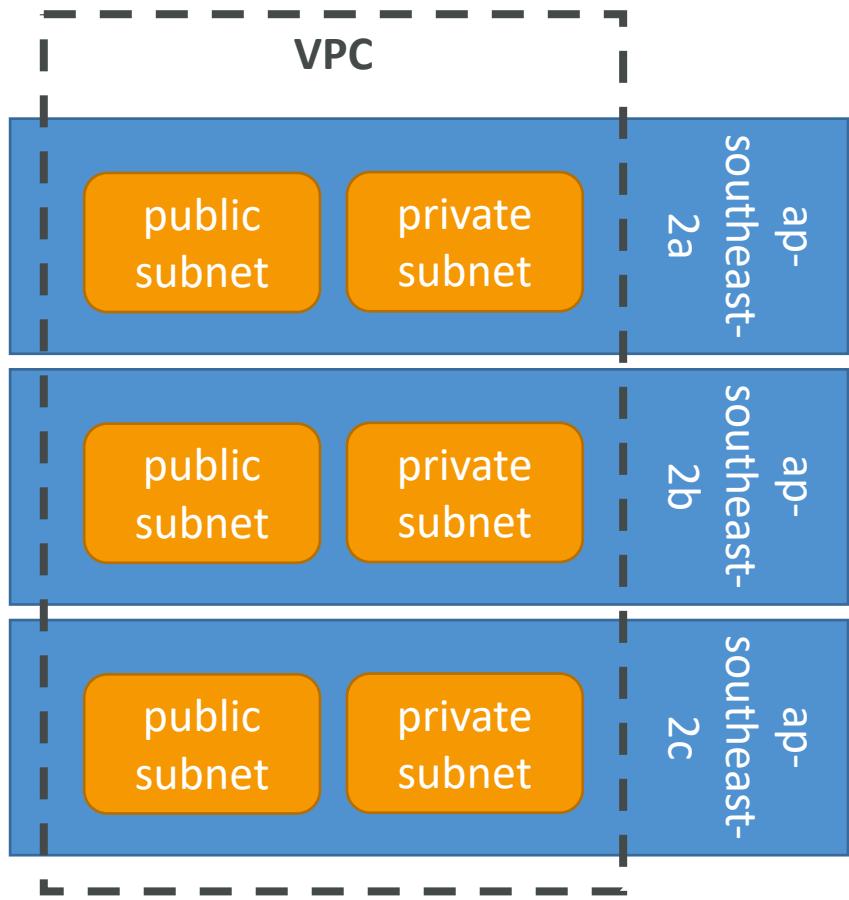
# Memcached Overview

- Memcached is an in-memory object store
- Cache doesn't survive reboots
- Use cases:
  - Quick retrieval of objects from memory
  - Cache often accessed objects
- Overall, Redis has largely grown in popularity and has better feature sets than Memcached.
- I would personally only use Redis for caching needs.
- AWS exam wouldn't ask if Redis or Memcached is better. Just "ElastiCache" in general

# AWS VPC

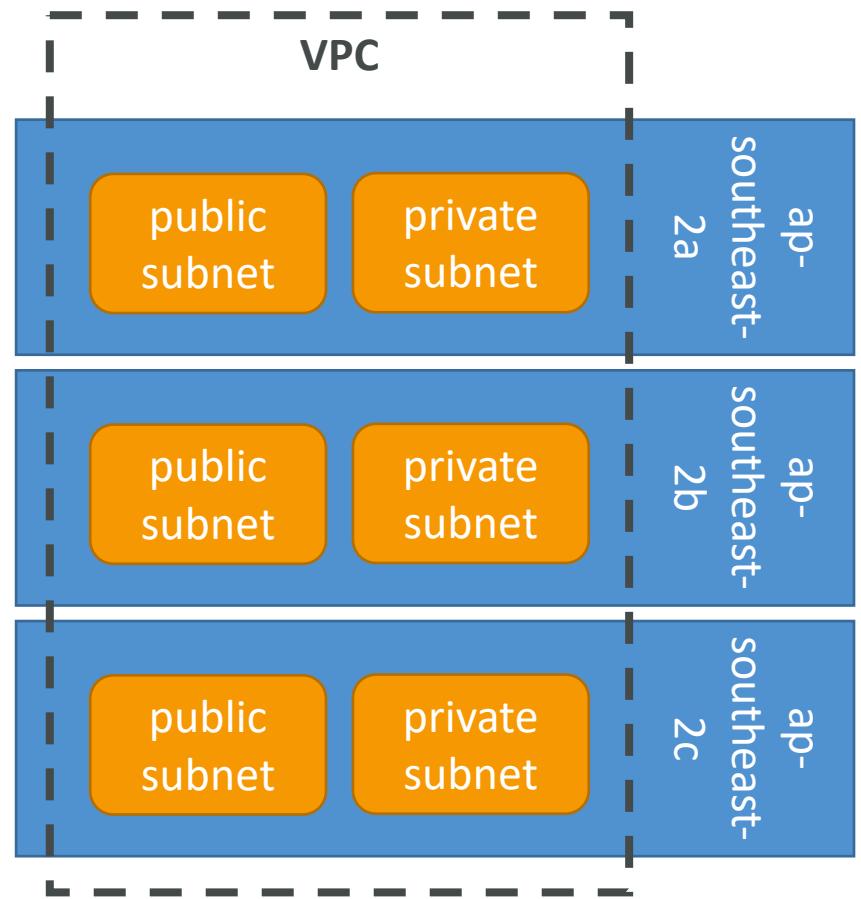


- Within a Region, you're able to create VPCs (Virtual Private Cloud)
- Each VPC contains subnets (networks)
- Each subnet must be mapped to an AZ
- It's common to have a public subnet (public IP)
- It's common to have a private subnet (private IP)
- It's common to have many subnets per AZ



# AWS VPC

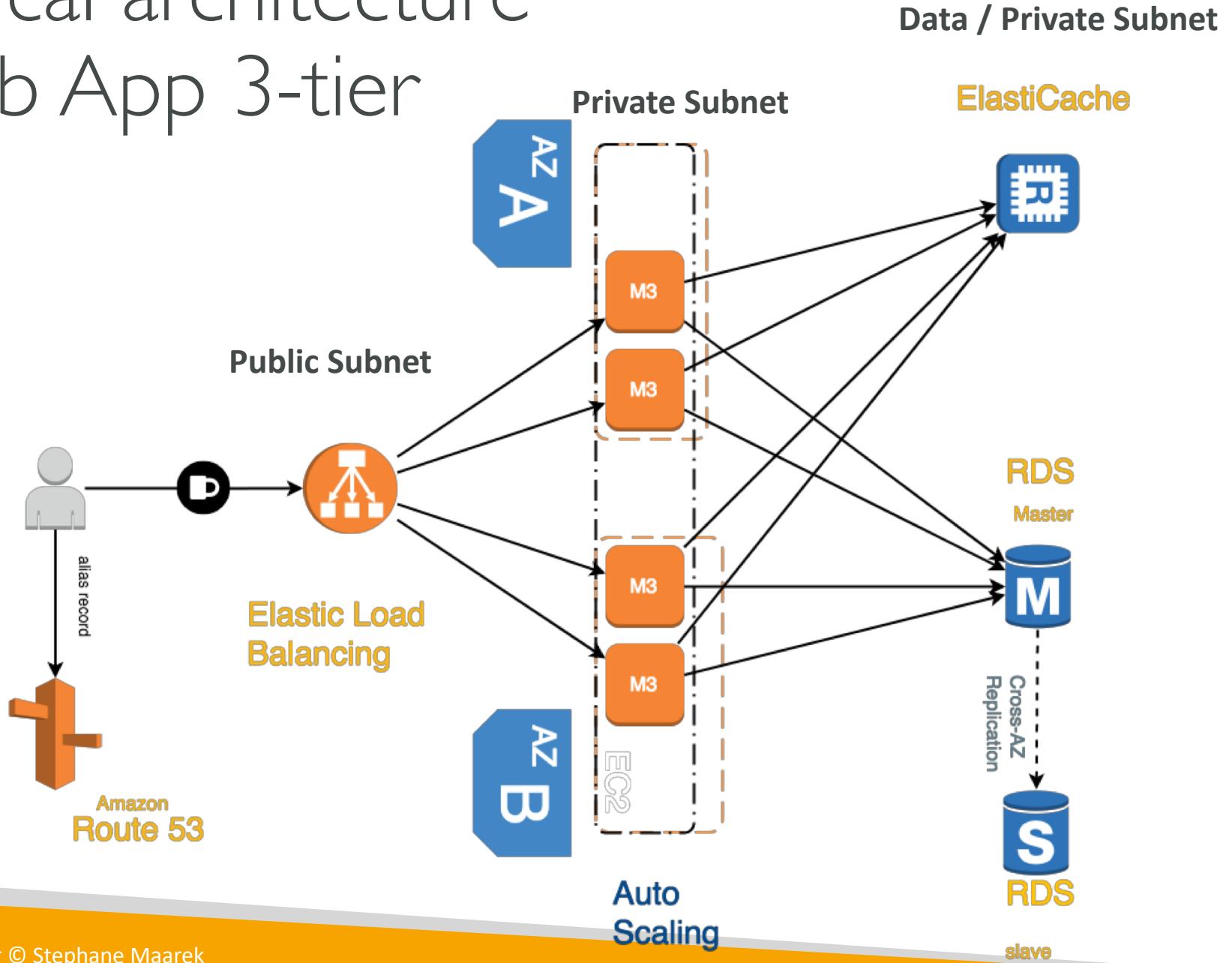
- Public Subnets usually contains:
  - Load Balancers
  - Static Websites
  - Files
  - Public Authentication Layers
- Private Subnets usually contains:
  - Web application servers
  - Databases
- Public and private subnets can communicate if they're in the same VPC!



# AWS VPC Brain Dump

- VPC & Regions aren't much asked at the developer associate exam
- All new accounts come with a default VPC
- It's possible to use a VPN to connect to a VPC  
(and access all the private IP straight from your laptop)
- VPC Flow Logs allow you to monitor the traffic within, in and out of your VPC (useful for security, performance, audit)
- VPC are per Account per Region
- Subnets are per VPC per AZ
- Some AWS resources can be deployed in VPC while others can't
- You can peer VPC (within or across accounts) to make it look like they're part of the same network

# Typical architecture Web App 3-tier

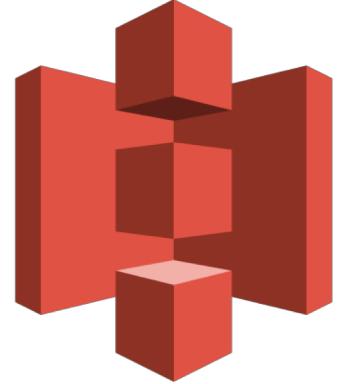


# Amazon S3

Another base block of AWS

# Section introduction

- Amazon S3 is one of the main building blocks of AWS
  - It's advertised as "infinitely scaling" storage
  - It's widely popular and deserves its own section
- 
- Many websites use AWS S3 as a backbone
  - Many AWS services uses AWS S3 as an integration as well
- 
- We'll have a step-by-step approach to S3



# AWS S3 Overview - Buckets

- Amazon S3 allows people to store objects (files) in “buckets” (directories)
- Buckets must have a **globally unique name**
- Buckets are defined at the region level
- Naming convention
  - No uppercase
  - No underscore
  - 3-63 characters long
  - Not an IP
  - Must start with lowercase letter or number



# AWS S3 Overview - Objects

- Objects (files) have a Key. The key is the **FULL** path:
  - <my\_bucket>/[my\\_file.txt](#)
  - <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 Values are the content of the body:
  - Max Size is 5TB
  - 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)



# AWS S3 - Versioning



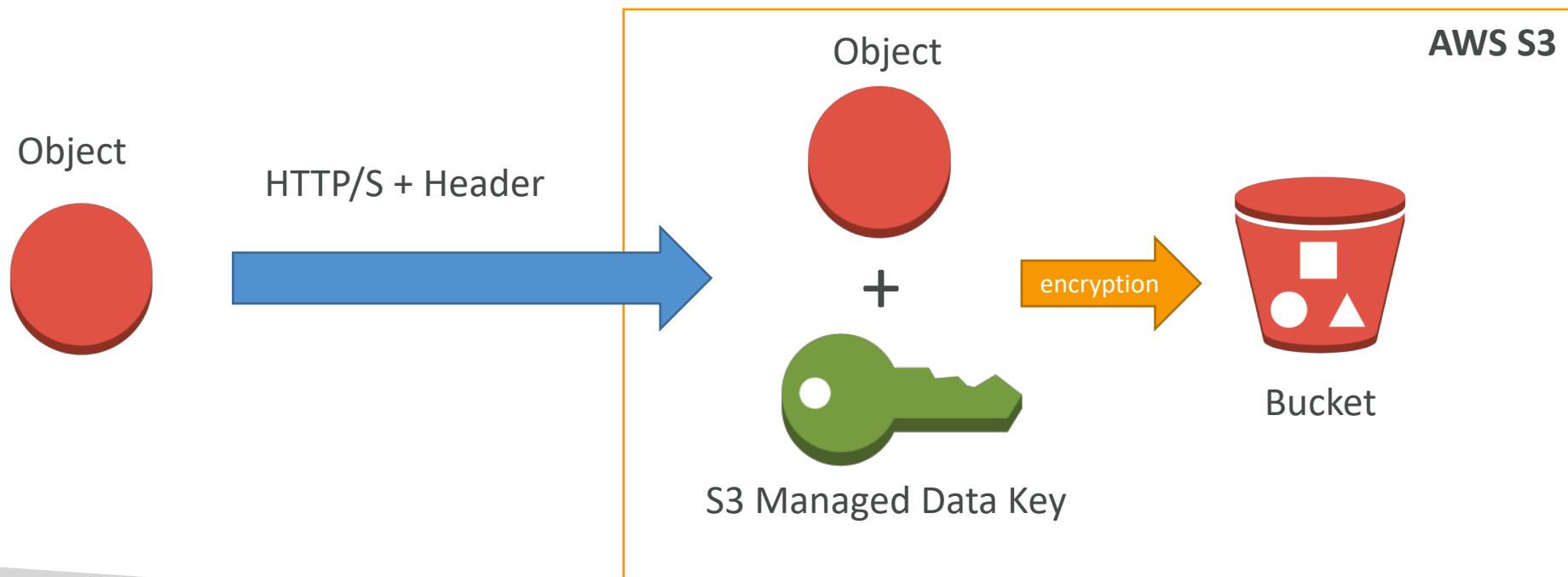
- You can version your files in AWS S3
- It is enabled at the **bucket level**
- Same key overwrite will increment the “version”: 1, 2, 3....
- It is best practice to version your buckets
  - Protect against unintended deletes (ability to restore a version)
  - Easy roll back to previous version
- Any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

# S3 Encryption for Objects

- There are 4 methods of encrypting objects in S3
  - SSE-S3: encrypts S3 objects using keys handled & managed by AWS
  - SSE-KMS: leverage AWS Key Management Service to manage encryption keys
  - SSE-C: when you want to manage your own encryption keys
  - Client Side Encryption
- It's important to understand which ones are adapted to which situation for the exam

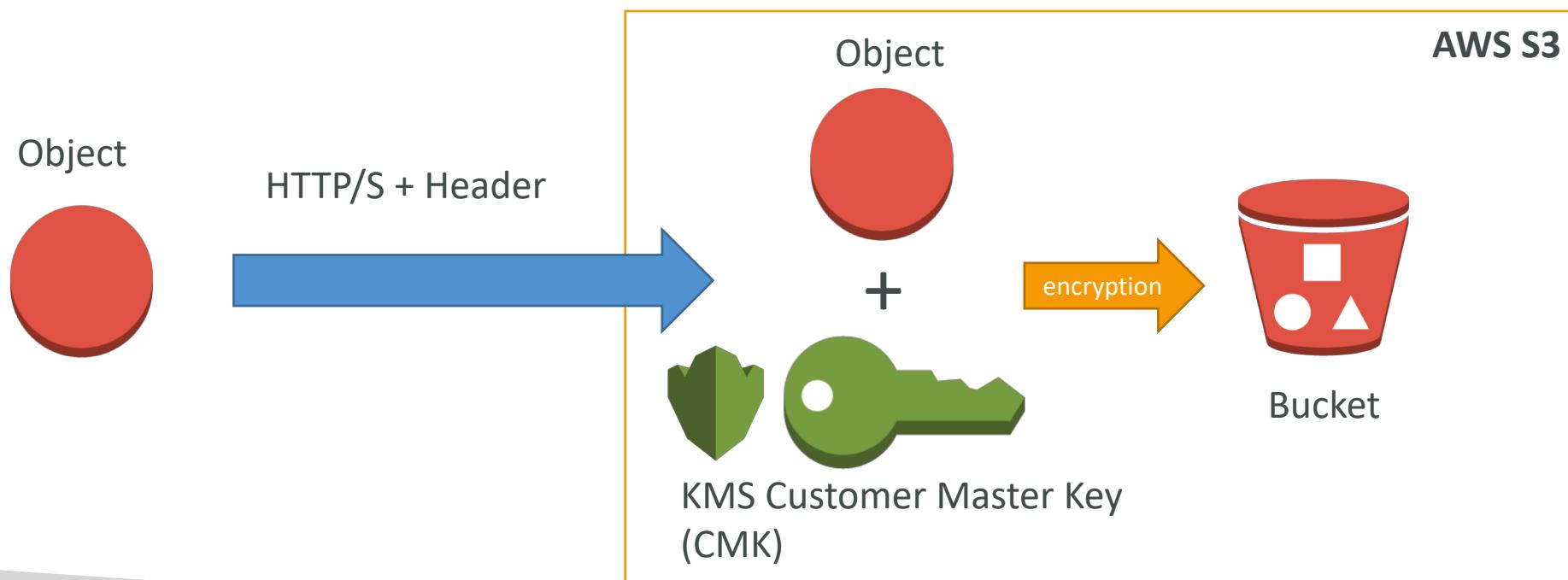
# SSE-S3

- SSE-S3: encryption using keys handled & managed by AWS S3
- Object is encrypted server side
- AES-256 encryption type
- Must set header: "x-amz-server-side-encryption": "AES256"



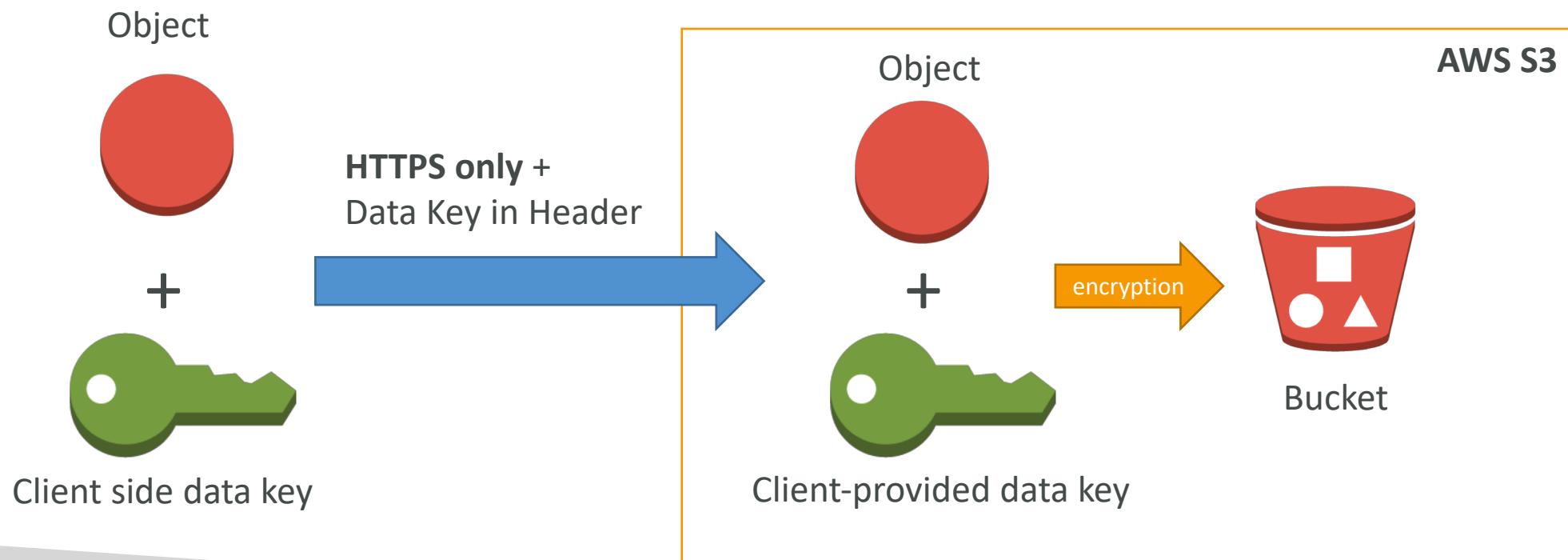
# SSE-KMS

- SSE-KMS: encryption using keys handled & managed by KMS
- KMS Advantages: user control + audit trail
- Object is encrypted server side
- Must set header: "x-amz-server-side-encryption": "aws:kms"



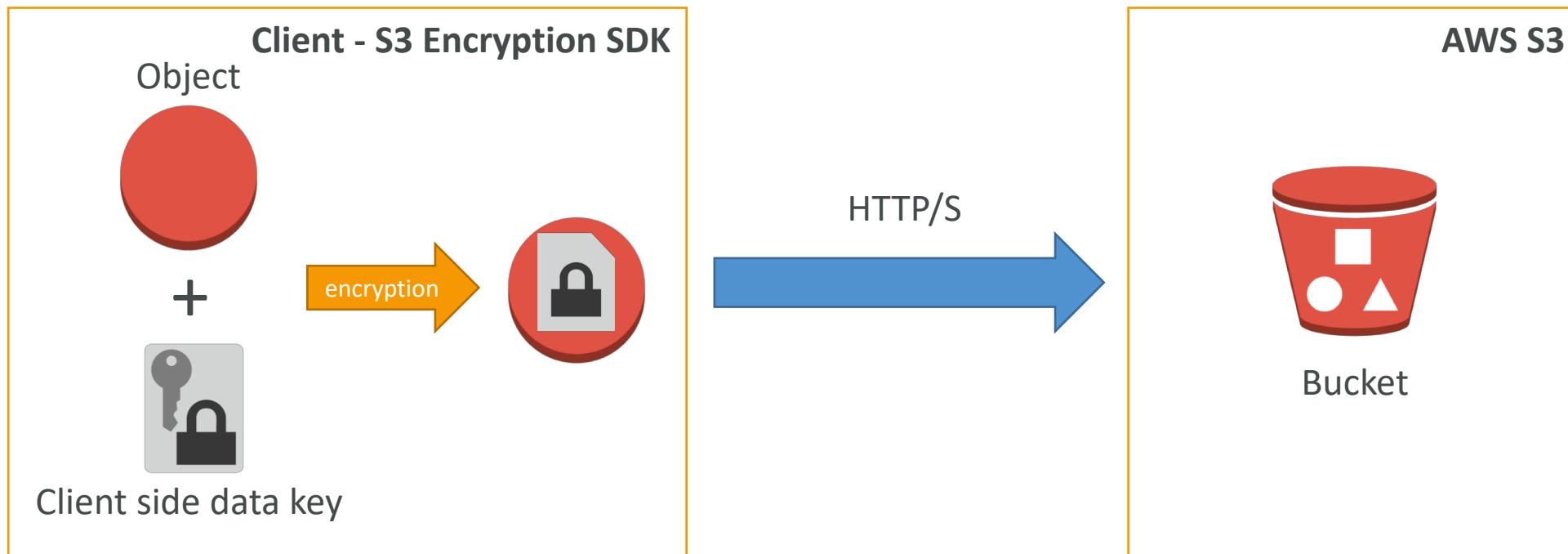
# SSE-C

- SSE-C: server-side encryption using data 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



# Client Side Encryption

- Client library such as the Amazon S3 Encryption Client
- Clients must encrypt data themselves before sending to S3
- Clients must decrypt data themselves when retrieving from S3
- Customer fully manages the keys and encryption cycle



# Encryption in transit (SSL)

- AWS S3 exposes:
  - HTTP endpoint: non encrypted
  - HTTPS endpoint: encryption in flight
- You're free to use the endpoint you want, but HTTPS is recommended
- HTTPS is mandatory for SSE-C
- Encryption in flight is also called SSL / TLS

# S3 Security

- User based
  - IAM policies - which API calls should be allowed for a specific user from IAM console
- Resource Based
  - Bucket Policies - bucket wide rules from the S3 console - allows cross account
  - Object Access Control List (ACL) – finer grain
  - Bucket Access Control List (ACL) – less common

# S3 Bucket Policies

- JSON based policies
  - Resources: buckets and objects
  - Actions: Set of API to Allow or Deny
  - Effect: Allow / 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)

# S3 Security - Other

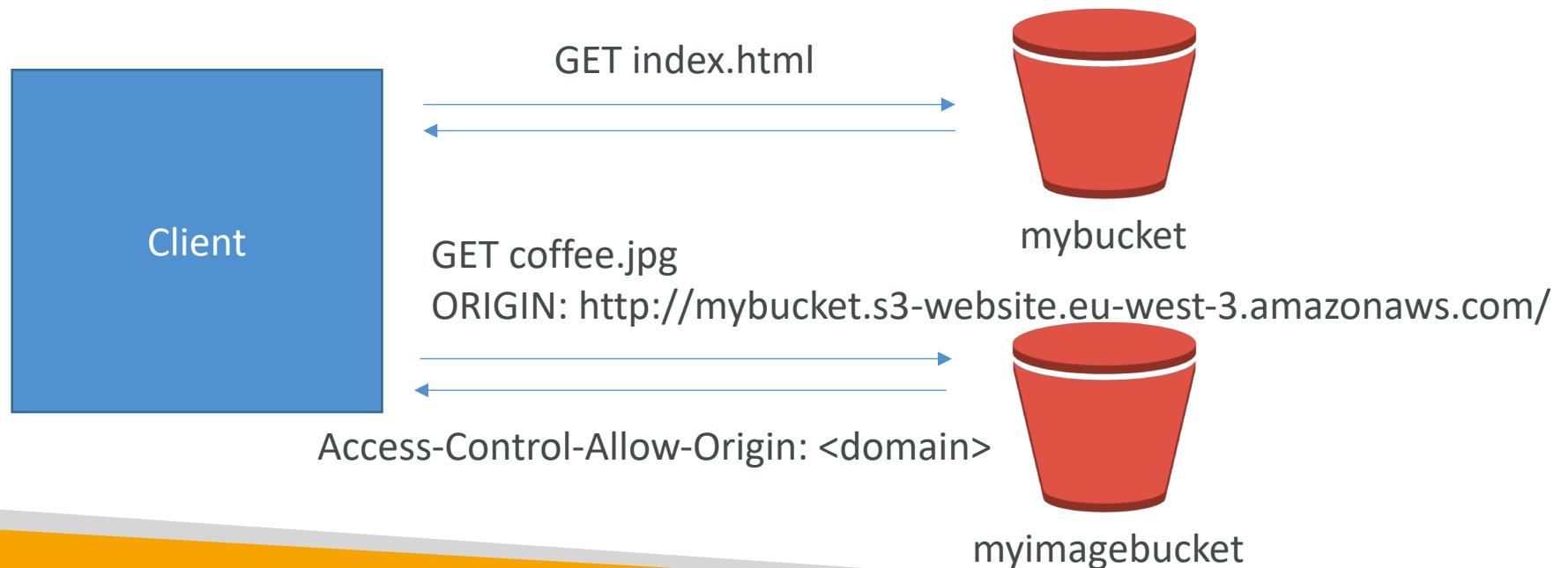
- Networking:
  - Supports VPC Endpoints (for instances in VPC without www internet)
- Logging and Audit:
  - S3 access logs can be stored in other S3 bucket
  - API calls can be logged in AWS CloudTrail
- User Security:
  - MFA (multi factor authentication) can be required in versioned buckets to delete objects
  - Signed URLs: URLs that are valid only for a limited time (ex: premium video service for logged in users)

# S3 Websites

- S3 can host static websites and have them accessible on the www
- The website URL will be:
  - <bucket-name>.s3-website-<AWS-region>.amazonaws.com
  - OR
  - <bucket-name>.s3-website.<AWS-region>.amazonaws.com
- If you get a 403 (Forbidden) error, make sure the bucket policy allows public reads!

# S3 CORS

- If you request data from another S3 bucket, you need to enable CORS
- Cross Origin Resource Sharing allows you to limit the number of websites that can request your files in S3 (and limit your costs)
- It's a popular exam question



# AWS S3 - Consistency Model

- Read after write consistency for PUTS of new objects
  - As soon as an object is written, we can retrieve it  
ex: (PUT 200 -> GET 200)
  - This is true, **except** if we did a GET before to see if the object existed  
ex: (GET 404 -> PUT 200 -> GET 404) – eventually consistent
- Eventual Consistency for DELETES and PUTS of existing objects
  - If we read an object after updating, we might get the older version  
ex: (PUT 200 -> PUT 200 -> GET 200 (might be older version))
  - If we delete an object, we might still be able to retrieve it for a short time  
ex: (DELETE 200 -> GET 200)

# AWS S3 - Other

- S3 can send notifications on changes to
  - AWS SQS: queue service
  - AWS SNS: notification service
  - AWS Lambda: serverless service
- S3 has a cross region replication feature (managed)

# AWS S3 Performance – Key Names

## Historic fact and current exam

- When you had > 100 TPS (transaction per second), S3 performance could degrade
- Behind the scene, each object goes to an S3 partition and for the best performance, we want the highest partition distribution
- In the exam, and historically, it was recommended to have random characters in front of your key name to optimise performance:
  - <my\_bucket>/5r4d\_my\_folder/my\_file1.txt
  - <my\_bucket>/a91e\_my\_folder/my\_file2.txt
  - ...
- It was recommended **never to use dates to prefix keys**:
  - <my\_bucket>/2018\_09\_09\_my\_folder/my\_file1.txt
  - <my\_bucket>/2018\_09\_10\_my\_folder/my\_file2.txt

# AWS S3 Performance – Key Names

## Current performance (not yet exam)

- <https://aws.amazon.com/about-aws/whats-new/2018/07/amazon-s3-announces-increased-request-rate-performance/>
- As of July 17<sup>th</sup> 2018, we can scale up to 3500 RPS for PUT and 5500 RPS for GET for EACH PREFIX
- “This S3 request rate performance increase removes any previous guidance to randomize object prefixes to achieve faster performance”
- It’s a “good to know”, until the exam gets updated ☺

# AWS S3 Performance

- Faster upload of large objects (>5GB), use multipart upload:
  - parallelizes PUTs for greater throughput
  - maximize your network bandwidth
  - decrease time to retry in case a part fails
- Use CloudFront to cache S3 objects around the world (improves reads)
- S3 Transfer Acceleration (uses edge locations) – just need to change the endpoint you write to, not the code.
- If using SSE-KMS encryption, you may be limited to your AWS limits for KMS usage (~100s – 1000s downloads / uploads per second)

# Developing on AWS

CLI, SDK and IAM Policies

# Section Introduction

- So far, we've interacted with services manually and they exposed standard information for clients:
  - EC2 exposes a standard Linux machine we can use any way we want
  - RDS exposes a standard database we can connect to using a URL
  - ElastiCache exposes a cache URL we can connect to using a URL
  - ASG / ELB are automated and we don't have to program against them
  - Route53 was setup manual
- Developing against AWS has two components:
  - How to perform interactions with AWS without using the Online Console?
  - How to interact with AWS Proprietary services? (S3, DynamoDB, etc...)

# Section Introduction

- Developing and performing AWS tasks against AWS can be done in several ways
  - Using the AWS CLI on our local computer
  - Using the AWS CLI on our EC2 machines
  - Using the AWS SDK on our local computer
  - Using the AWS SDK on our EC2 machines
  - Using the AWS Instance Metadata Service for EC2
- In this section, we'll learn:
  - How to do all of those
  - In the right & most secure way, adhering to best practices

# AWS CLI Setup Windows

- We'll setup the CLI properly on Windows

# AWS CLI Setup Mac OS X

- We'll setup the CLI properly on Mac OS X

# AWS CLI Setup Linux

- We'll setup the CLI properly on Linux

# AWS CLI Configuration

- Let's learn how to properly configure the CLI



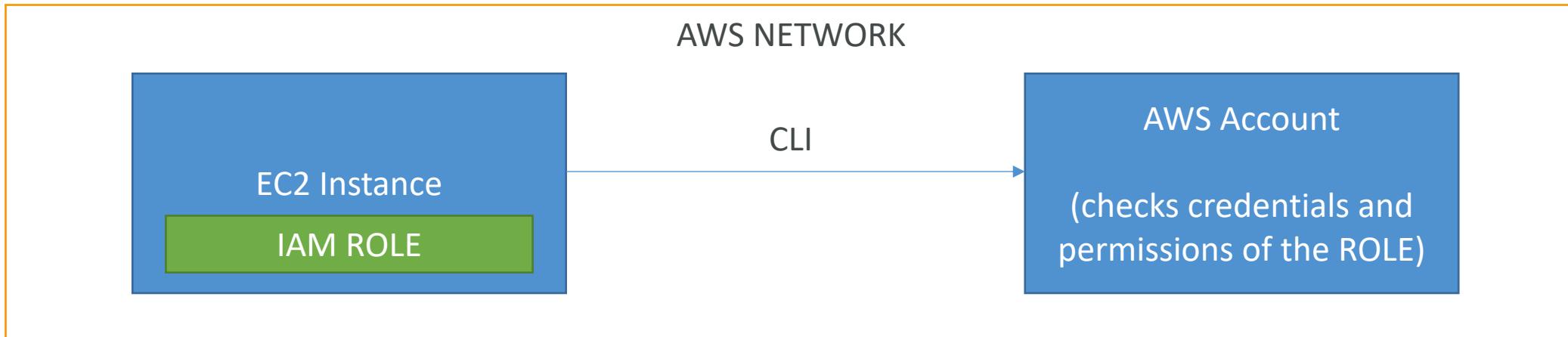
- We'll learn how to get our access credentials and protect them
- Do not share your AWS Access Key and Secret key with anyone!

# AWS CLI ON EC2... THE BAD WAY

- We could run `aws configure` on EC2 just like we did (and it'll work)
- But... it's SUPER INSECURE
- NEVER EVER EVER PUT YOUR PERSONAL CREDENTIALS ON AN EC2
- Your PERSONAL credentials are PERSONAL and only belong on your PERSONAL computer
- If the EC2 is compromised, so is your personal account
- If the EC2 is shared, other people may perform AWS actions while impersonating you
- For EC2, there's a better way... it's called AWS IAM Roles

# AWS CLI ON EC2... THE RIGHT WAY

- IAM Roles can be attached to EC2 instances
- IAM Roles can come with a policy authorizing exactly what the EC2 instance should be able to do



- EC2 Instances can then use these profiles automatically without any additional configurations
- This is the best practice on AWS and you should 100% do this.

# AWS CLI Dry Runs

- Sometimes, we'd just like to make sure we have the permissions...
- But not actually run the commands!
- Some AWS CLI commands (such as EC2) can become expensive if they succeed, say if we wanted to try to create an EC2 Instance
- Some AWS CLI commands (not all) contain a `--dry-run` option to simulate API calls
- Let's practice!

# AWS CLI STS Decode Errors

- When you run API calls and they fail, you can get a long error message
- This error message can be decoded using the **STS** command line:
- `sts decode-authorization-message`
  
- Let's practice!

# AWS EC2 Instance Metadata

- AWS EC2 Instance Metadata 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!

# 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

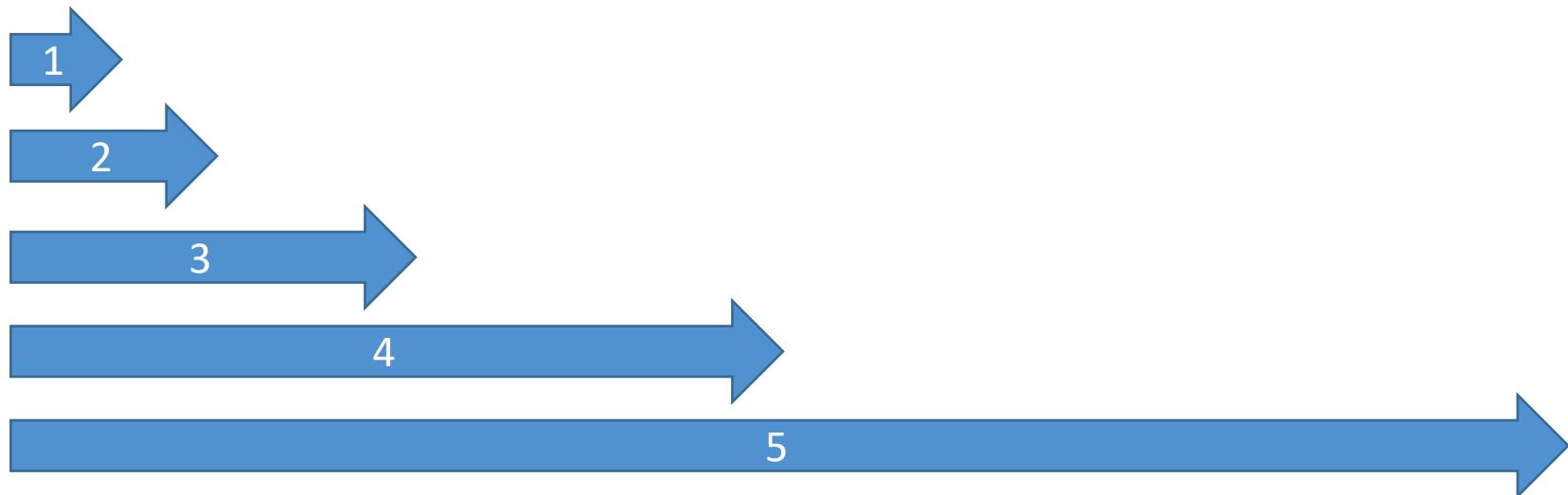
- We have to use the AWS SDK when coding against AWS Services such as DynamoDB
- Fun fact... the AWS CLI uses the Python SDK (boto3)
- The exam expects you to know when you should use an SDK
- We'll practice the AWS SDK when we get to the Lambda functions
- Good to know: if you don't specify or configure a default region, then us-east-1 will be chosen by default

# AWS SDK Credentials Security

- It's recommend to use the **default credential provider chain**
- The **default credential provider chain** works seamlessly with:
  - AWS credentials at `~/.aws/credentials` (only on our computers or on premise)
  - Instance Profile Credentials using IAM Roles (for EC2 machines, etc...)
  - Environment variables (`AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`)
- Overall, **NEVER EVER STORE AWS CREDENTIALS IN YOUR CODE.**
- Best practice is for credentials to be inherited from mechanisms above, and 100% IAM Roles if working from within AWS Services

# Exponential Backoff

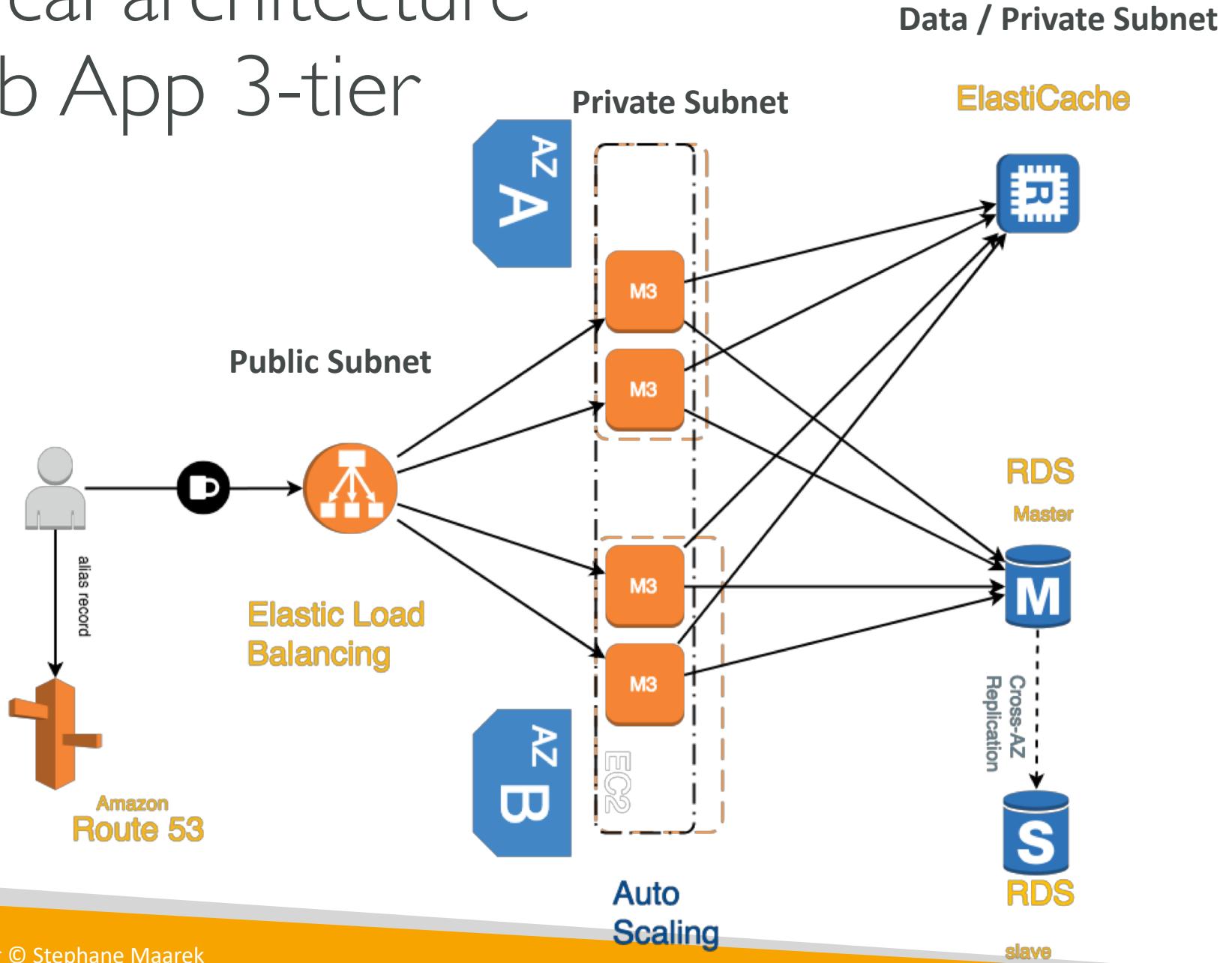
- Any API that fails because of too many calls needs to be retried with Exponential Backoff
- These apply to rate limited API
- Retry mechanism included in SDK API calls



# AWS Elastic Beanstalk

Deploying applications in AWS safely and predictably

# Typical architecture Web App 3-tier

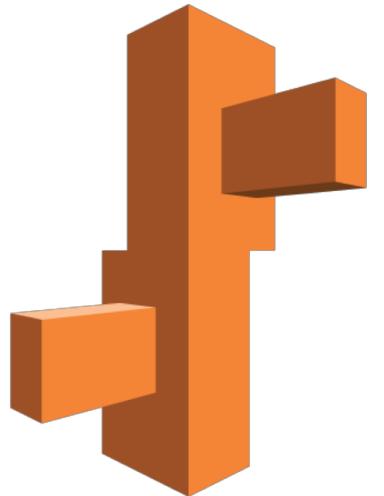


# Developer problems on AWS

- 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

# AWS 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, etc...
- But it's all in one view that's easy to make sense of!
- We still have full control over the configuration
- Beanstalk is free but you pay for the underlying instances

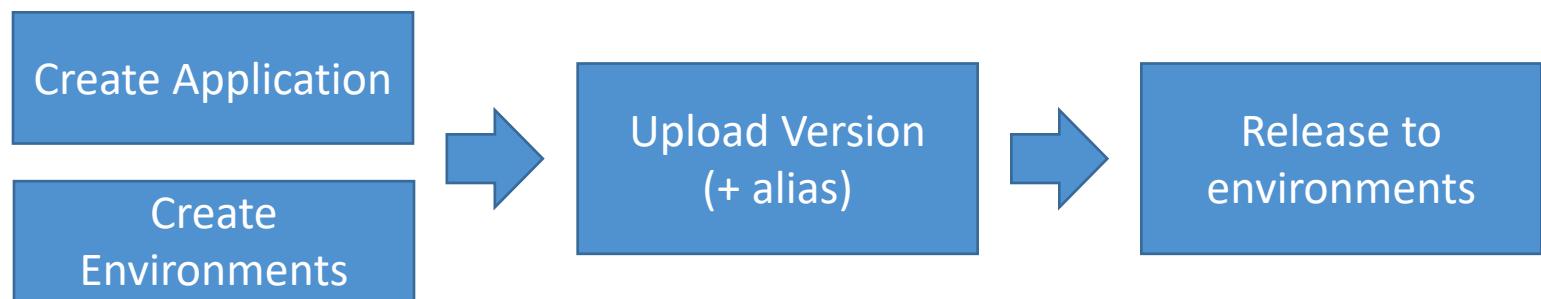


# Elastic Beanstalk

- Managed service
  - Instance configuration / OS is handled by Beanstalk
  - Deployment strategy is configurable but performed by Elastic Beanstalk
- Just the application code is the responsibility of the developer
- Three architecture models:
  - Single Instance deployment: good for dev
  - LB + ASG: great for production or pre-production web applications
  - ASG only: great for non-web apps in production (workers, etc..)

# Elastic Beanstalk

- Elastic Beanstalk has three components
  - Application
  - Application version: each deployment gets assigned a version
  - Environment name (dev, test, prod...): free naming
- You deploy application versions to environments and can promote application versions to the next environment
- Rollback feature to previous application version
- Full control over lifecycle of environments

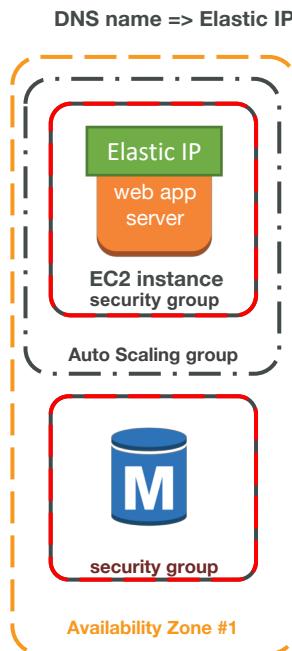


# Elastic Beanstalk

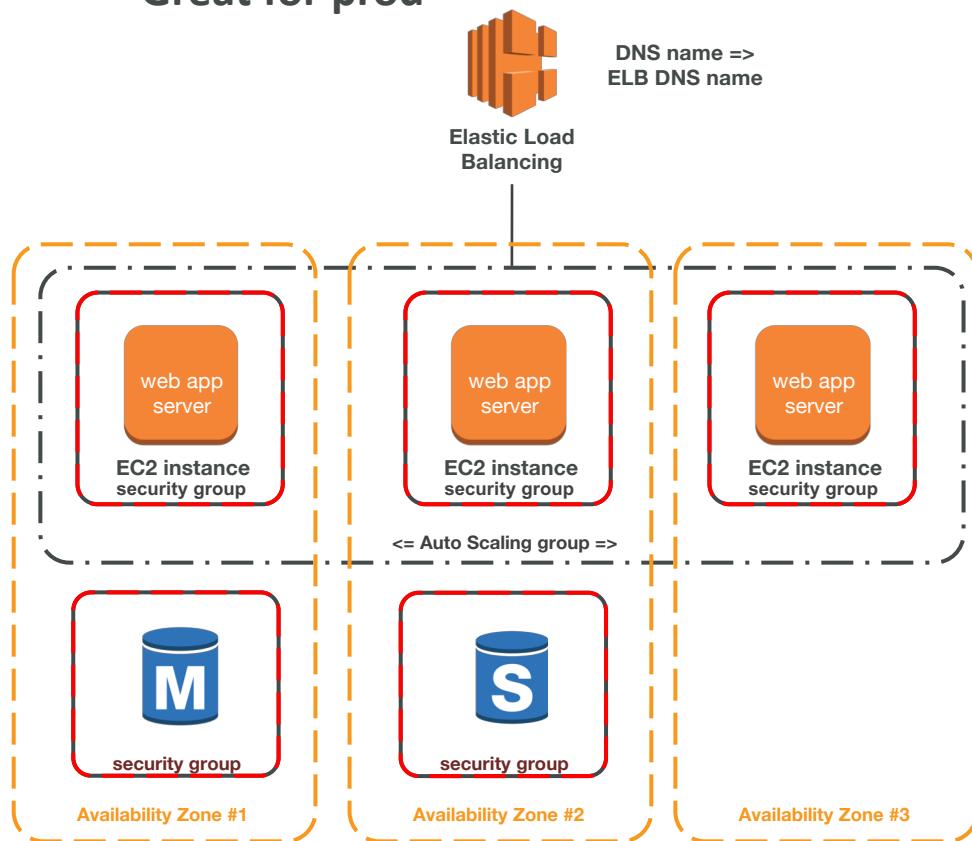
- Support for many platforms:
  - Go
  - Java SE
  - Java with Tomcat
  - .NET on Windows Server with IIS
  - Node.js
  - PHP
  - Python
  - Ruby
  - Packer Builder
- Single Container Docker
- Multicontainer Docker
- Preconfigured Docker
- If not supported, you can write your custom platform (advanced)

# Elastic Beanstalk Deployment Modes

## Single Instance Great for dev



## High Availability with Load Balancer Great for prod



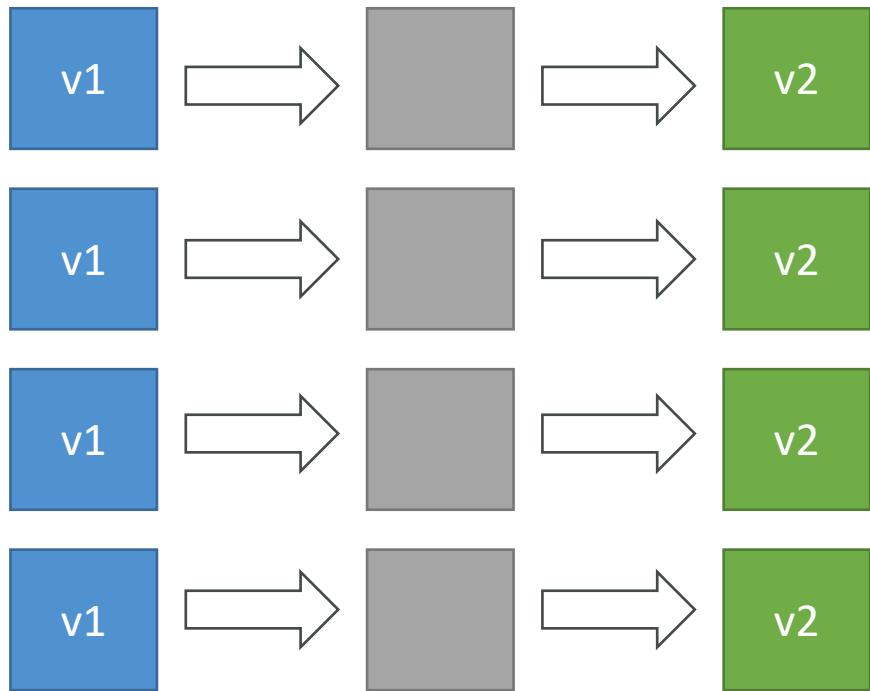
# Beanstalk Deployment Options for Updates

- **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

# Elastic Beanstalk Deployment

## All at once

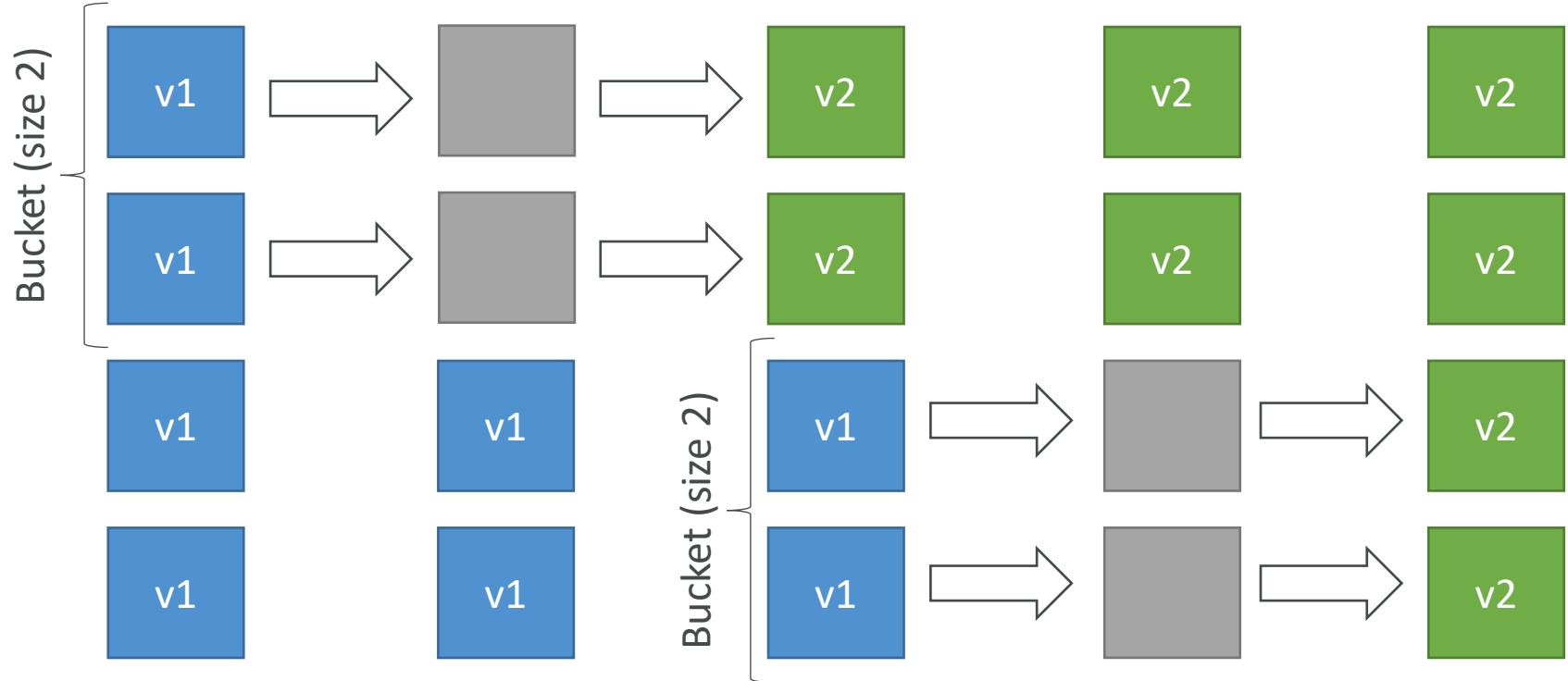
- Fastest deployment
- Application has downtime
- Great for quick iterations in development environment
- No additional cost



# 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

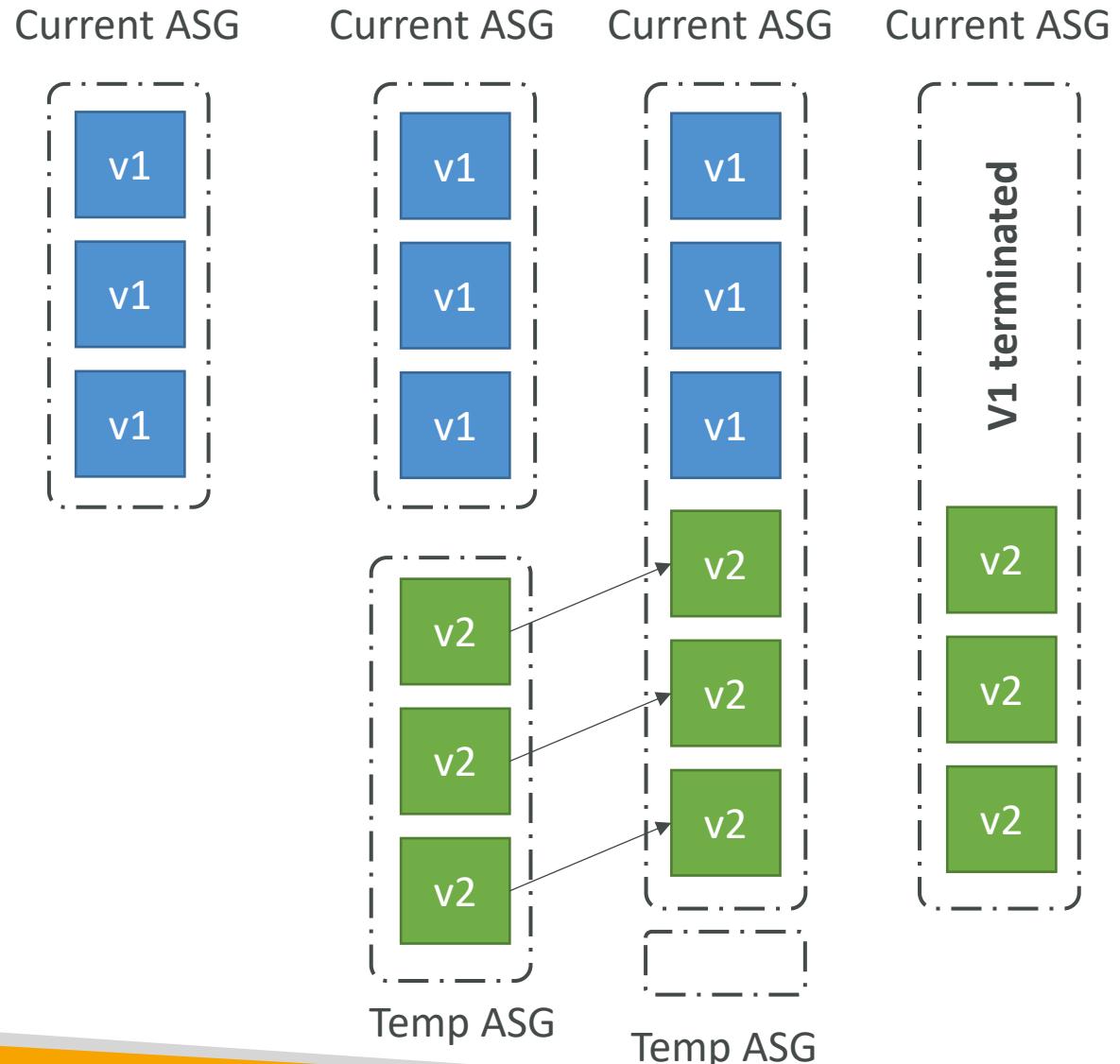
- Application is running at capacity
- Can set the bucket size
- Application is running both versions simultaneously
- Small additional cost
- Additional batch is removed at the end of the deployment
- Longer deployment
- Good for prod



# Elastic Beanstalk Deployment

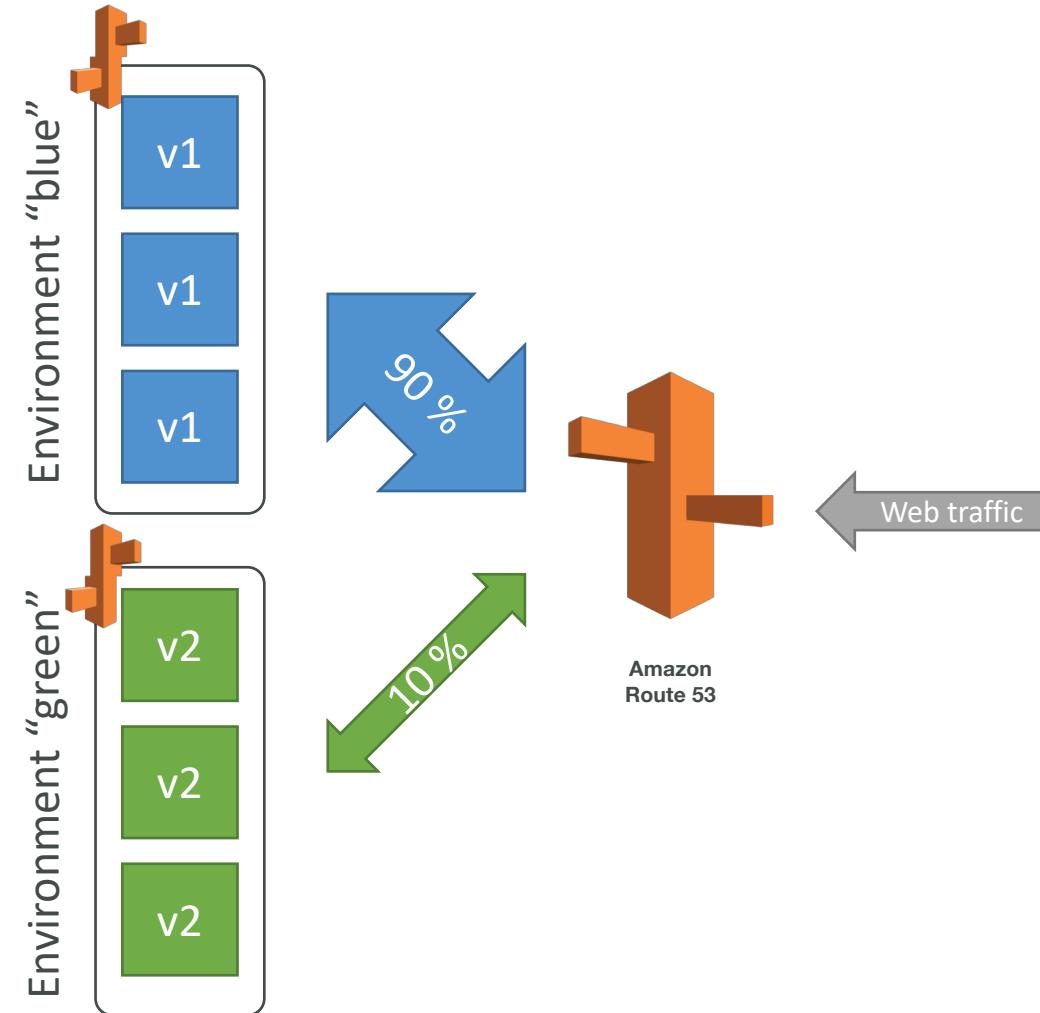
## Immutable

- 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 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 prior to failure running new application version	⊕ ⊕ †	✓	✓	Manual Redeploy	Existing instances	
Rolling with additional batch	Minimal if first batch fails, otherwise, similar to Rolling	⊕ ⊕ ⊕ †	✓	✓	Manual Redeploy	New and existing instances	
Immutable	Minimal	⊕ ⊕ ⊕ ⊕	✓	✓	Terminate New Instances	New instances	
Blue/green	Minimal	⊕ ⊕ ⊕ ⊕	✓	X	Swap URL	New instances	

# 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 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 Under the Hood

- Under the hood, Elastic Beanstalk relies on CloudFormation
- CloudFormation is an AWS service we'll see later
- Let's have a sneak peak into it!

# ElasticBeanstalk Deployment Mechanism

- Describe dependencies  
(requirements.txt for Python, package.json for Node.js)
- Package code as zip
- Zip file is uploaded to each EC2 machine
- Each EC2 machine resolves dependencies (SLOW)
- Optimization in case of long deployments: Package dependencies with source code to improve deployment performance and speed

# AWS CICD

CodeCommit, CodePipeline, CodeBuild, CodeDeploy

# CICD Section Introduction

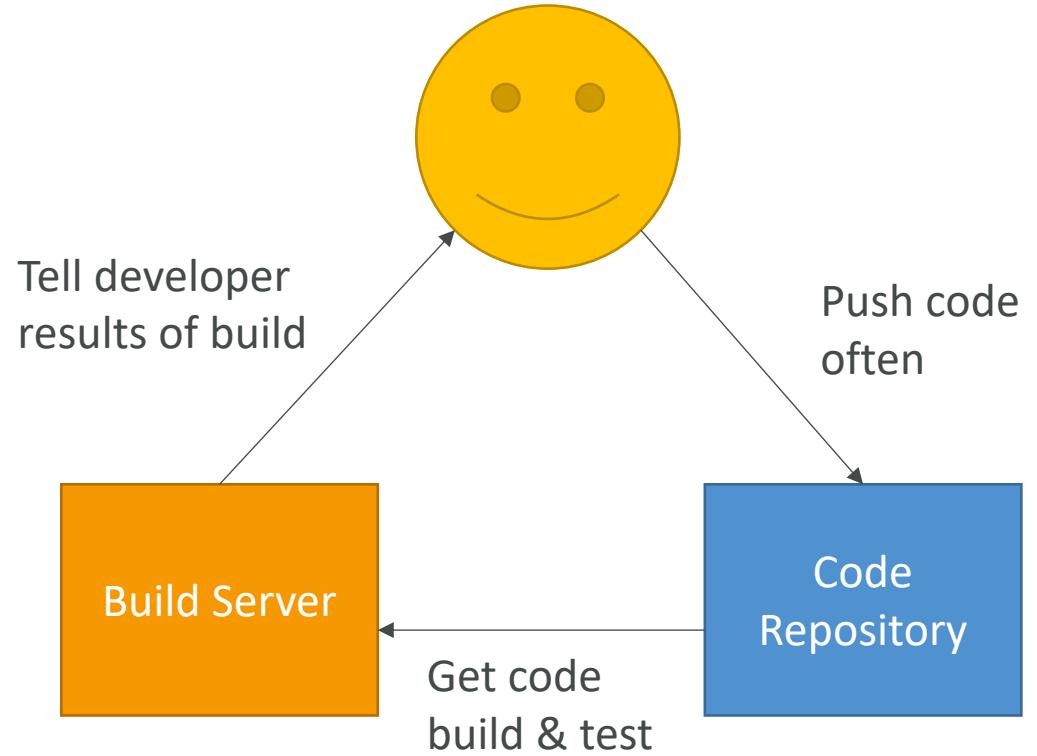
- We now know how to create resources in AWS manually (Fundamentals)
- We know how to interact with AWS programmatically (CLI)
- We've seen how to deploy code to AWS using Elastic Beanstalk
- All these manual steps make it very likely for us to do mistakes!
- What we'd like is to push our code “in a repository” and have it deployed onto the AWS
  - Automatically
  - The right way
  - Making sure it's tested before deploying
  - With possibility to go into different stages (dev, test, pre-prod, prod)
  - With manual approval where needed
- To be a proper AWS developer... we need to learn AWS CICD

# CICD Section Introduction

- This section is all about automating the deployment we've done so far while adding increased safety.
- It correspond to a whole part of the AWS Certification
- We'll learn about
  - AWS CodeCommit: storing our code
  - AWS CodePipeline: automating our pipeline from code to ElasticBeanstalk
  - AWS CodeBuild: building and testing our code
  - AWS CodeDeploy: deploying the code to EC2 fleets (not Beanstalk)

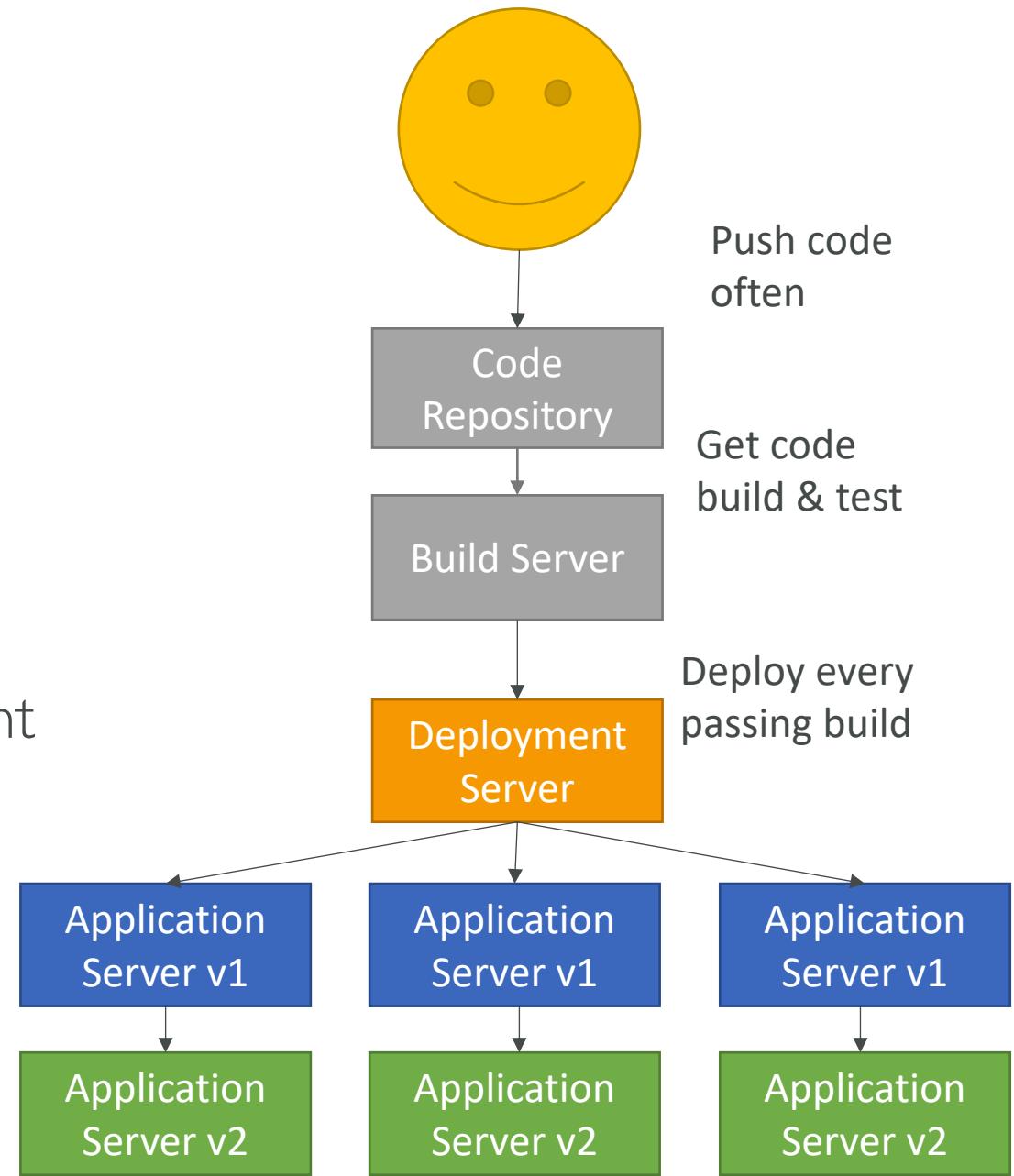
# Continuous Integration

- Developers push the code to a code repository often (GitHub / CodeCommit / Bitbucket / etc...)
- A testing / build server checks the code as soon as it's pushed (CodeBuild / Jenkins CI / etc...)
- The developer gets feedback about the tests and checks that have passed / failed
- Find bugs early, fix bugs
- Deliver faster as the code is tested
- Deploy often
- Happier developers, as they're unblocked

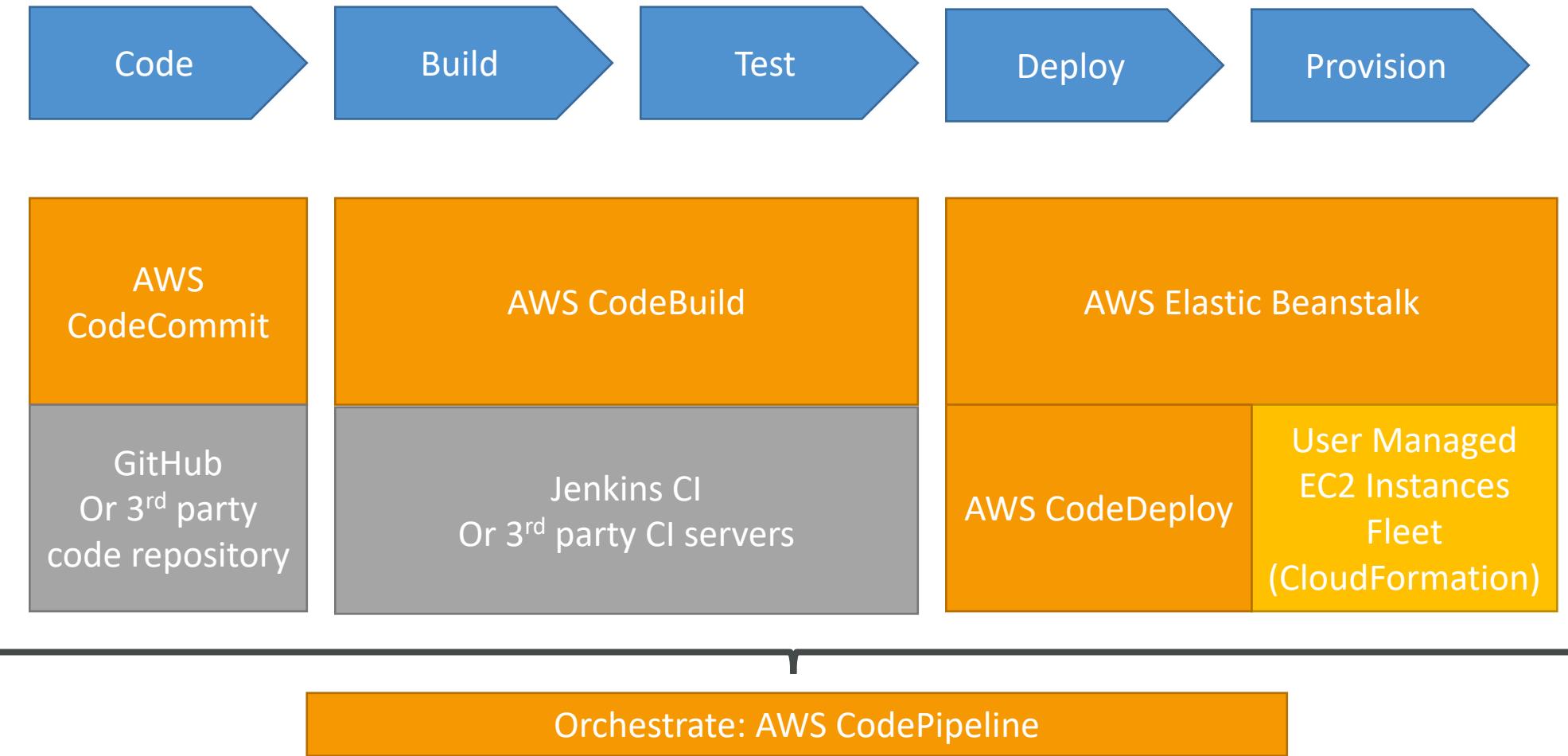


# Continuous Delivery

- Ensure 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
  - CodeDeploy
  - Jenkins CD
  - Spinnaker
  - Etc...



# Technology Stack for CICD



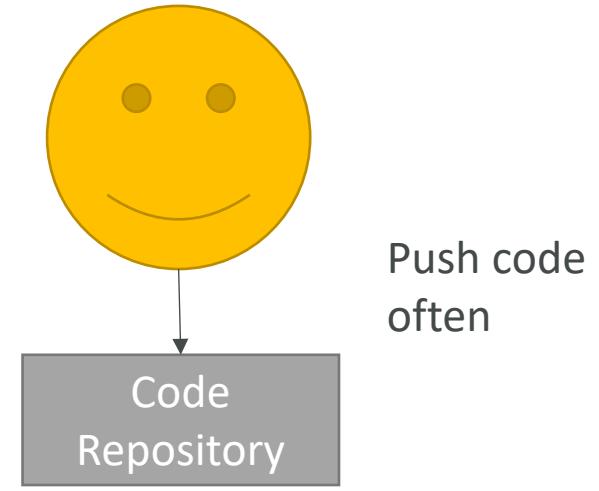


# CodeCommit

- **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 live on one's machine, but it usually lives 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

# CodeCommit

- Git repositories can be expensive.
- The industry includes:
  - GitHub: free public repositories, paid private ones
  - BitBucket
  - Etc...
- 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
  - Secure (encrypted, access control, etc...)
  - Integrated with Jenkins / CodeBuild / other CI tools



Push code  
often

# CodeCommit Security

- Interactions are done using Git (standard)
- Authentication in Git:
  - SSH Keys: AWS Users can configure SSH keys in their IAM Console
  - HTTPS: Done through the AWS CLI Authentication helper or Generating HTTPS credentials
  - MFA (multi factor authentication) can be enabled for extra safety
- Authorization in Git:
  - IAM Policies manage user / roles rights to repositories
- Encryption:
  - Repositories are automatically encrypted at rest using KMS
  - Encrypted in transit (can only use HTTPS or SSH – both secure)
- Cross Account access:
  - Do not share your SSH keys
  - Do not share your AWS credentials
  - Use IAM Role in your AWS Account and use AWS STS (with AssumeRole API)

# CodeCommit vs GitHub

## Similarities:

- Both are git repositories
- Both support code review (pull requests)
- GitHub and CodeCommit can be integrated with AWS CodeBuild
- Both support HTTPS and SSH method of authentication

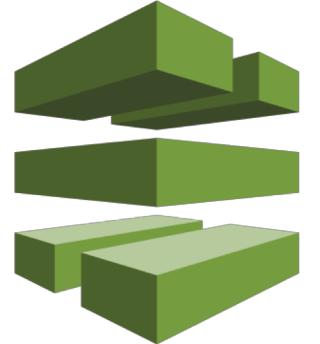
## Differences:

- Security:
  - GitHub: GitHub Users
  - CodeCommit: AWS IAM users & roles,
- Hosted:
  - GitHub: hosted by GitHub
  - GitHub Enterprise: self hosted on your servers
  - CodeCommit: managed & hosted by AWS
- UI:
  - GitHub UI is fully featured
  - CodeCommit UI is minimal

# CodeCommit Notifications

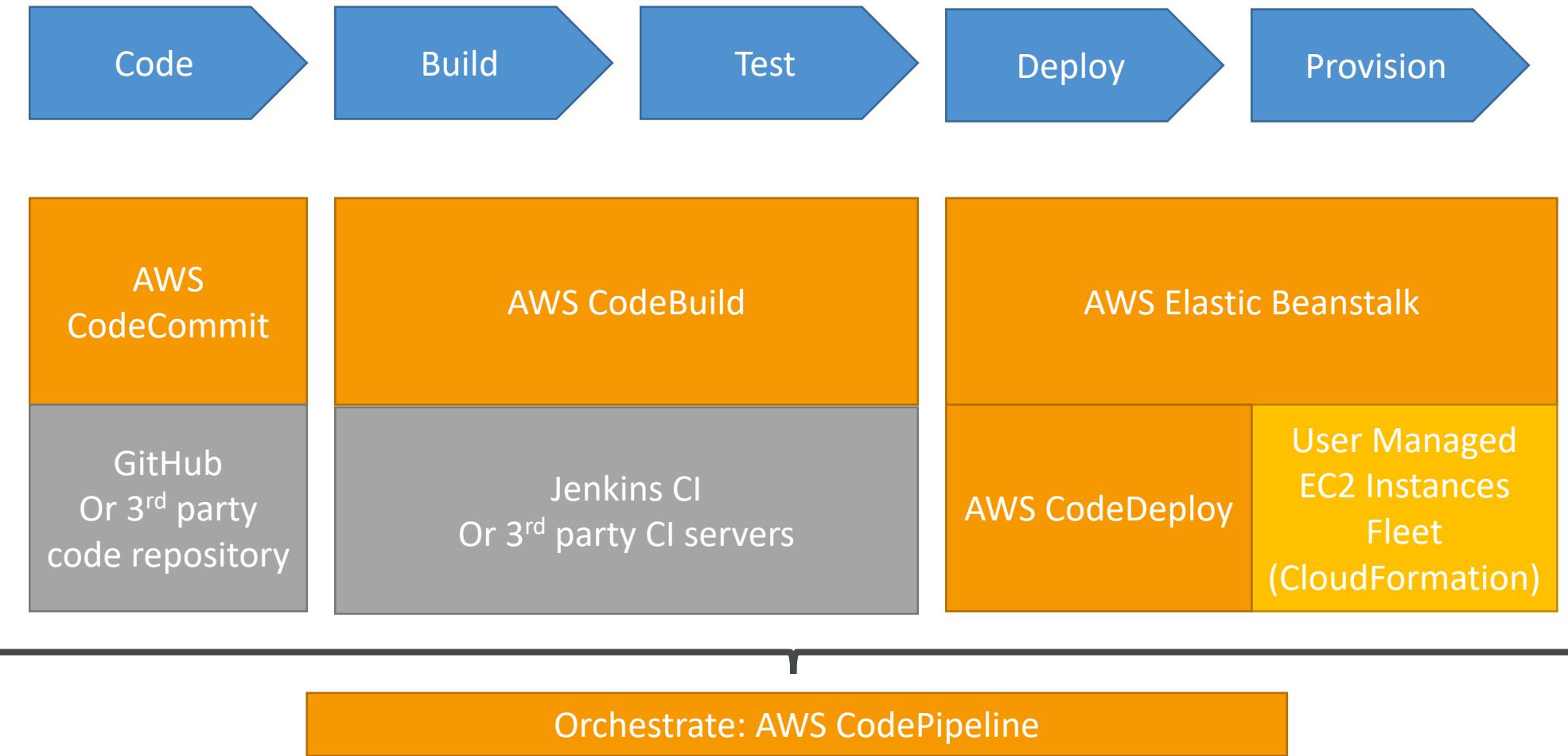
- You can trigger notifications in CodeCommit using AWS SNS (Simple Notification Service) or AWS Lambda or AWS CloudWatch Event Rules
- Use cases for notifications SNS / AWS Lambda notifications:
  - Deletion of branches
  - Trigger for pushes that happens in master branch
  - Notify external Build System
  - Trigger AWS Lambda function to perform codebase analysis (maybe credentials got committed in the code?)
- Use cases for CloudWatch Event Rules:
  - Trigger for pull request updates (created / updated / deleted / commented)
  - Commit comment events
  - CloudWatch Event Rules goes into an SNS topic

# CodePipeline



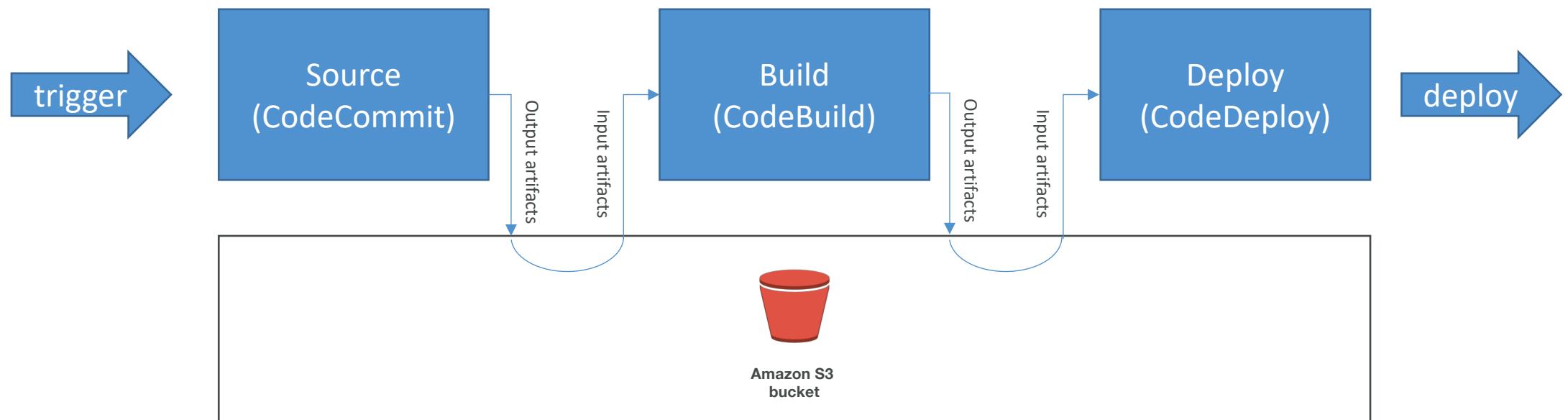
- Continuous delivery
- Visual workflow
- Source: GitHub / CodeCommit / Amazon S3
- Build: CodeBuild / Jenkins / etc...
- Load Testing: 3<sup>rd</sup> party tools
- Deploy: AWS CodeDeploy / Beanstalk / CloudFormation / ECS...
- Made of stages:
  - Each stage can have sequential actions and / or parallel actions
  - Stages examples: Build / Test / Deploy / Load Test / etc...
  - Manual approval can be defined at any stage

# Technology Stack for CICD



# AWS CodePipeline Artifacts

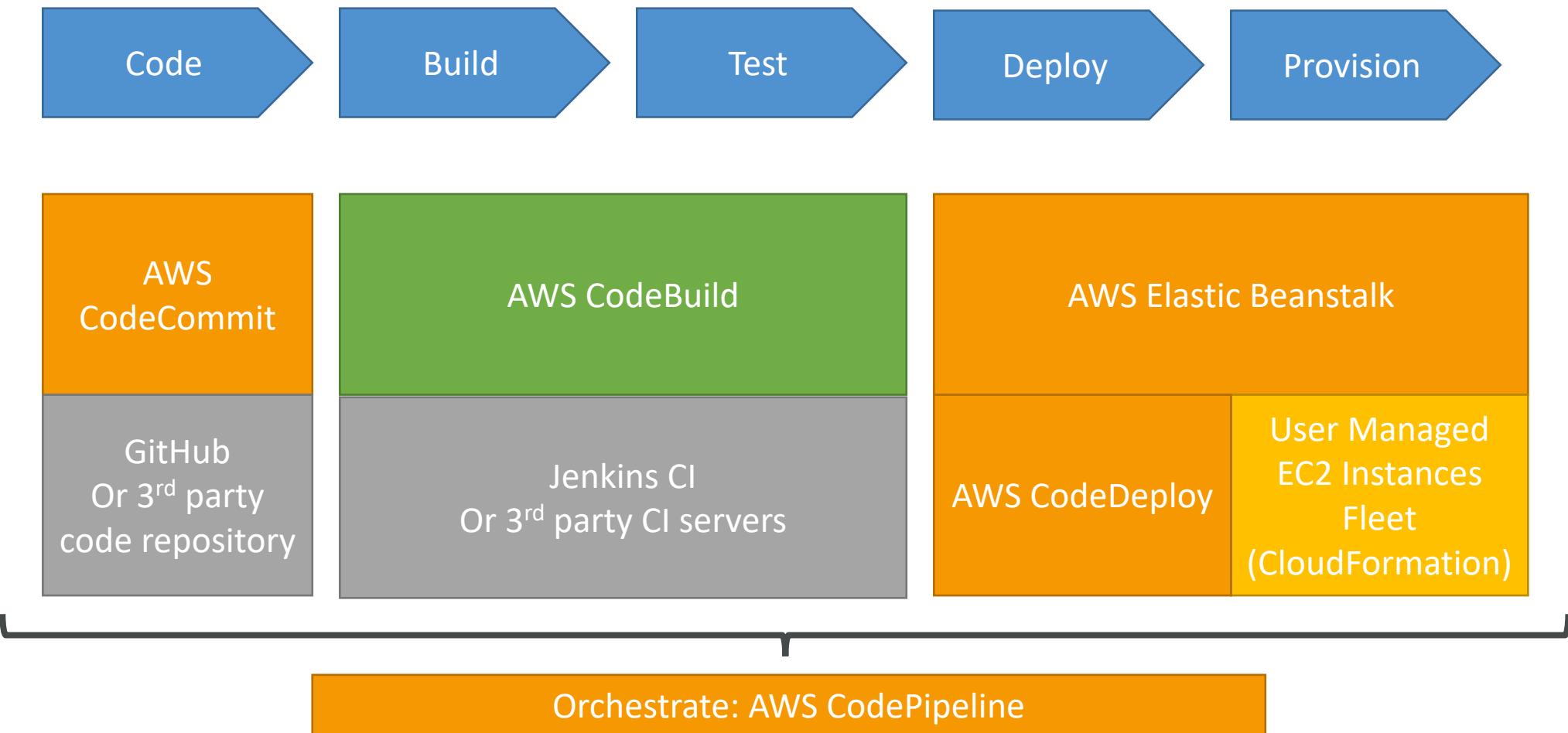
- Each pipeline stage can create "artifacts"
- Artifacts are passed stored in Amazon S3 and passed on to the next stage



# CodePipeline Troubleshooting

- CodePipeline state changes happen in AWS CloudWatch Events, which can in return create SNS notifications.
  - Ex: you can create events for failed pipelines
  - Ex: you can create events for cancelled stages
- If CodePipeline fails a stage, your pipeline stops and you can get information in the console
- AWS CloudTrail can be used to audit AWS API calls
- If Pipeline can't perform an action, make sure the "IAM Service Role" attached does have enough permissions (IAM Policy)

# CodeBuild



# CodeBuild Overview



- Fully managed build service
- Alternative to other build tools such as Jenkins
- Continuous scaling (no servers to manage or provision – no build queue)
- Pay for usage: the time it takes to complete the builds
- Leverages Docker under the hood for reproducible builds
- Possibility to extend capabilities leveraging our own base Docker images
- Secure: Integration with KMS for encryption of build artifacts, IAM for build permissions, and VPC for network security, CloudTrail for API calls logging

# CodeBuild Overview

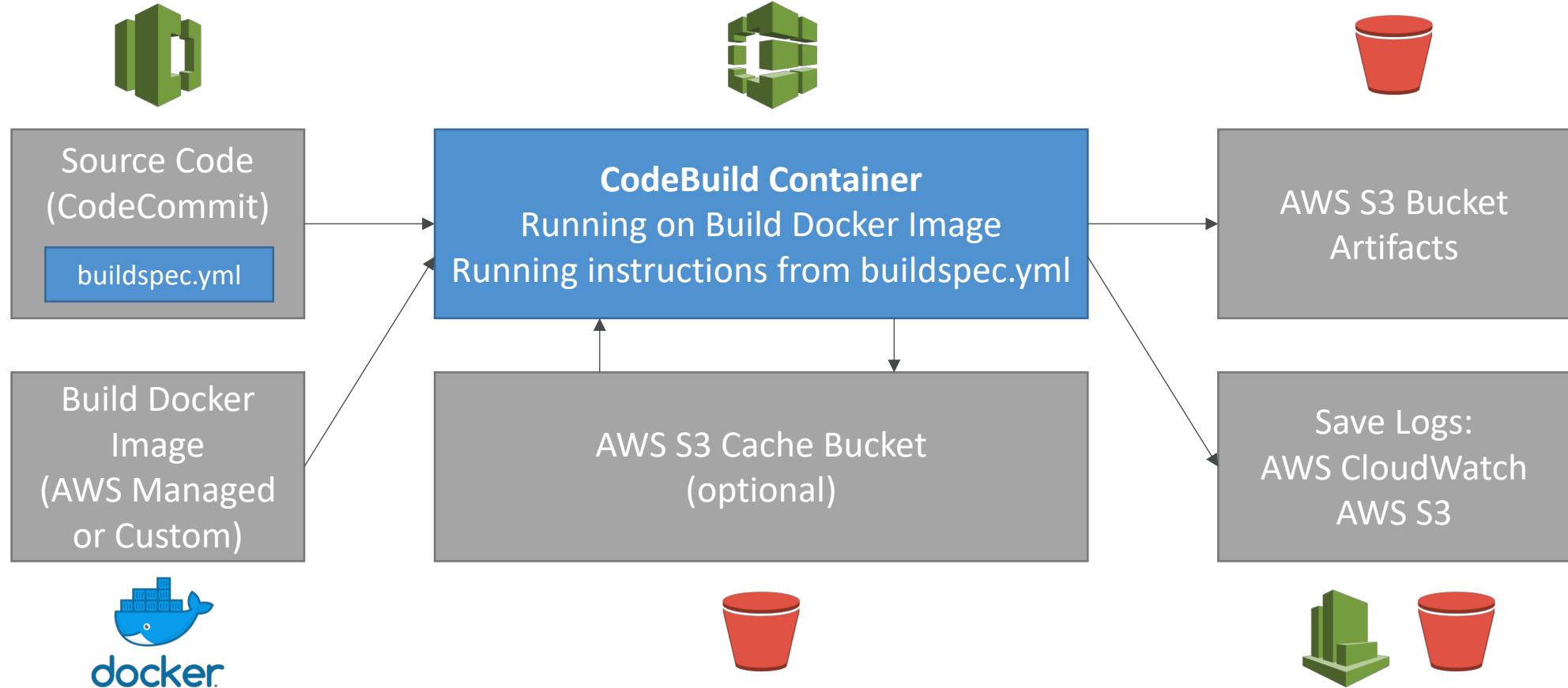


- Source Code from GitHub / CodeCommit / CodePipeline / S3...
- Build instructions can be defined in code (buildspec.yml file)
- Output logs to Amazon S3 & AWS CloudWatch Logs
- Metrics to monitor CodeBuild statistics
- Use CloudWatch Events to detect failed builds and trigger notifications
- Use CloudWatch Alarms to notify if you need “thresholds” for failures
- CloudWatch Events / AWS Lambda as a Glue
- SNS notifications
- Ability to reproduce CodeBuild locally to troubleshoot in case of errors
- Pipelines can be defined within CodePipeline or CodeBuild itself

# CodeBuild Support environments

- Java
- Ruby
- Python
- Go
- Node.js
- Android
- .NET Core
- PHP
- Docker: extend any environment you like

# How CodeBuild works



# CodeBuild BuildSpec

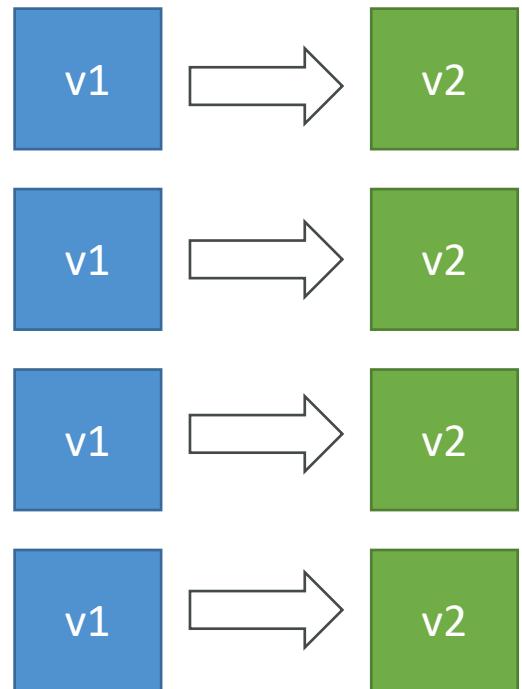
- `buildspec.yml` file must be at the root of your code
- Define environment variables:
  - Plaintext variables
  - Secure secrets: use SSM Parameter store
- Phases (specify commands to run):
  - Install: install dependencies you may need for your build
  - Pre build: final commands to execute before build
  - **Build: actual build commands**
  - Post build: finishing touches (zip output for example)
- Artifacts: What to upload to S3 (encrypted with KMS)
- Cache: Files to cache (usually dependencies) to S3 for future build speedup

# CodeBuild Local Build

- In case of need of deep troubleshooting beyond logs...
  - You can run CodeBuild locally on your desktop (after installing Docker)
  - For this, leverage the CodeBuild Agent
- 
- <https://docs.aws.amazon.com/codebuild/latest/userguide/use-codebuild-agent.html>

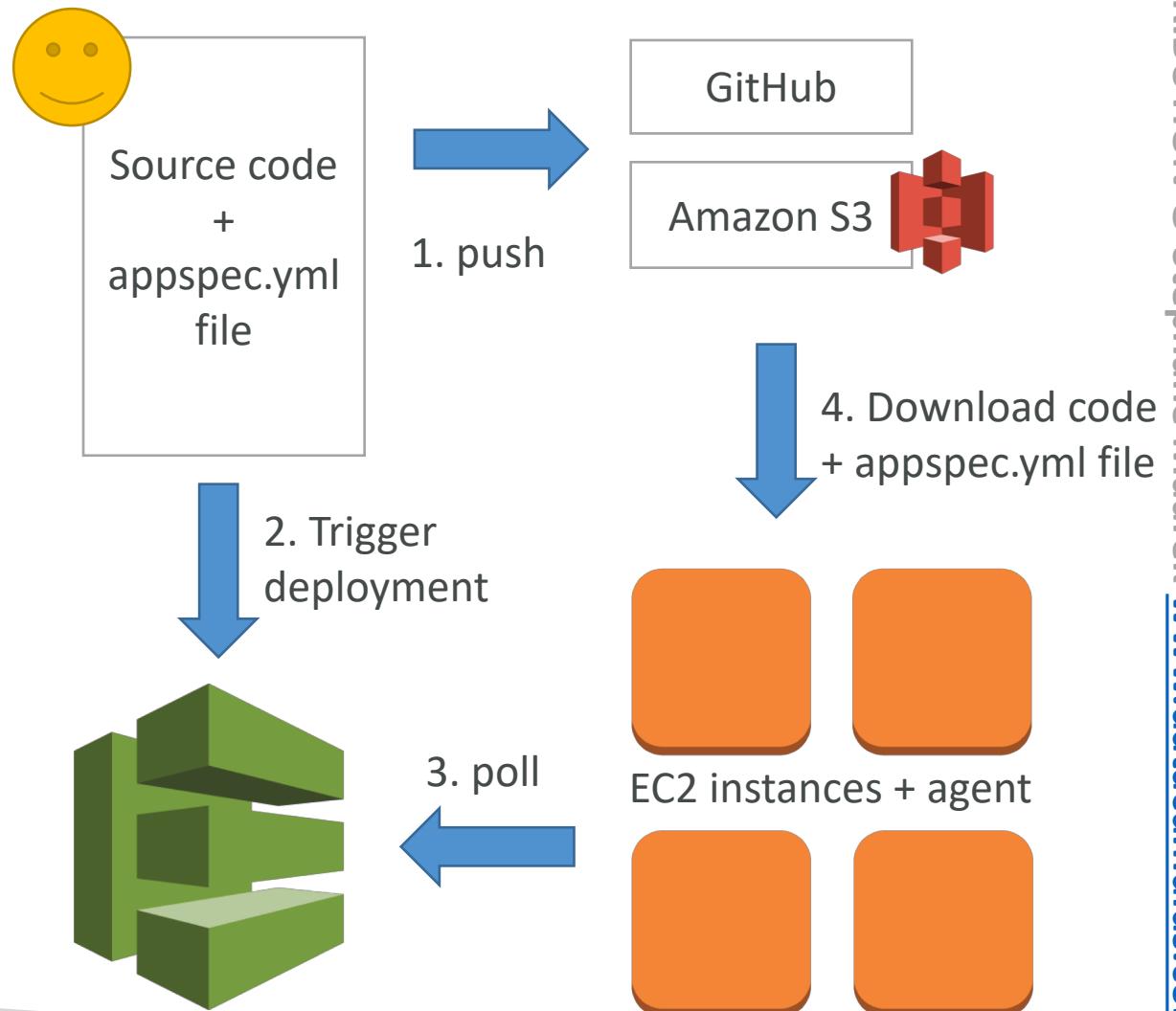
# AWS CodeDeploy

- We want to deploy our application automatically to many EC2 instances
- These instances are not managed by Elastic Beanstalk
- There are several ways to handle deployments using open source tools (Ansible, Terraform, Chef, Puppet, etc...)
- We can use the managed Service AWS CodeDeploy



# AWS CodeDeploy – Steps to make it work

- Each EC2 Machine (or On Premise machine) must be running the CodeDeploy Agent
- The agent is continuously polling AWS CodeDeploy for work to do
- CodeDeploy sends **appspec.yml** file.
- Application is pulled from GitHub or S3
- EC2 will run the deployment instructions
- CodeDeploy Agent will report of success / failure of deployment on the instance



# AWS CodeDeploy – Other

- EC2 instances are grouped by deployment group (dev / test / prod)
- Lots of flexibility to define any kind of deployments
- CodeDeploy can be chained into CodePipeline and use artifacts from there
- CodeDeploy can re-use existing setup tools, works with any application, auto scaling integration
- Note: Blue / Green only works with EC2 instances (not on premise)
- Support for AWS Lambda deployments (we'll see this later)
- CodeDeploy does not provision resources

# AWS CodeDeploy Primary Components

- **Application:** unique name
- **Compute platform:** EC2/On-Premise or Lambda
- **Deployment configuration:** Deployment rules for success / failures
  - EC2/On-Premise: you can specify the minimum number of healthy instances for the deployment.
  - AWS Lambda: specify how traffic is routed to your updated Lambda function versions.
- **Deployment group:** group of tagged instances (allows to deploy gradually)
- **Deployment type:** In-place deployment or Blue/green deployment:
- **IAM instance profile:** need to give EC2 the permissions to pull from S3 / GitHub
- **Application Revision:** application code + appspec.yml file
- **Service role:** Role for CodeDeploy to perform what it needs
- **Target revision:** Target deployment application version

# AWS CodeDeploy AppSpec

- File section: how to source and copy from S3 / GitHub to filesystem
- Hooks: set of instructions to do to deploy the new version (hooks can have timeouts). The order is:
  - ApplicationStop
  - DownloadBundle
  - BeforeInstall
  - AfterInstall
  - ApplicationStart
  - ValidateService: really important

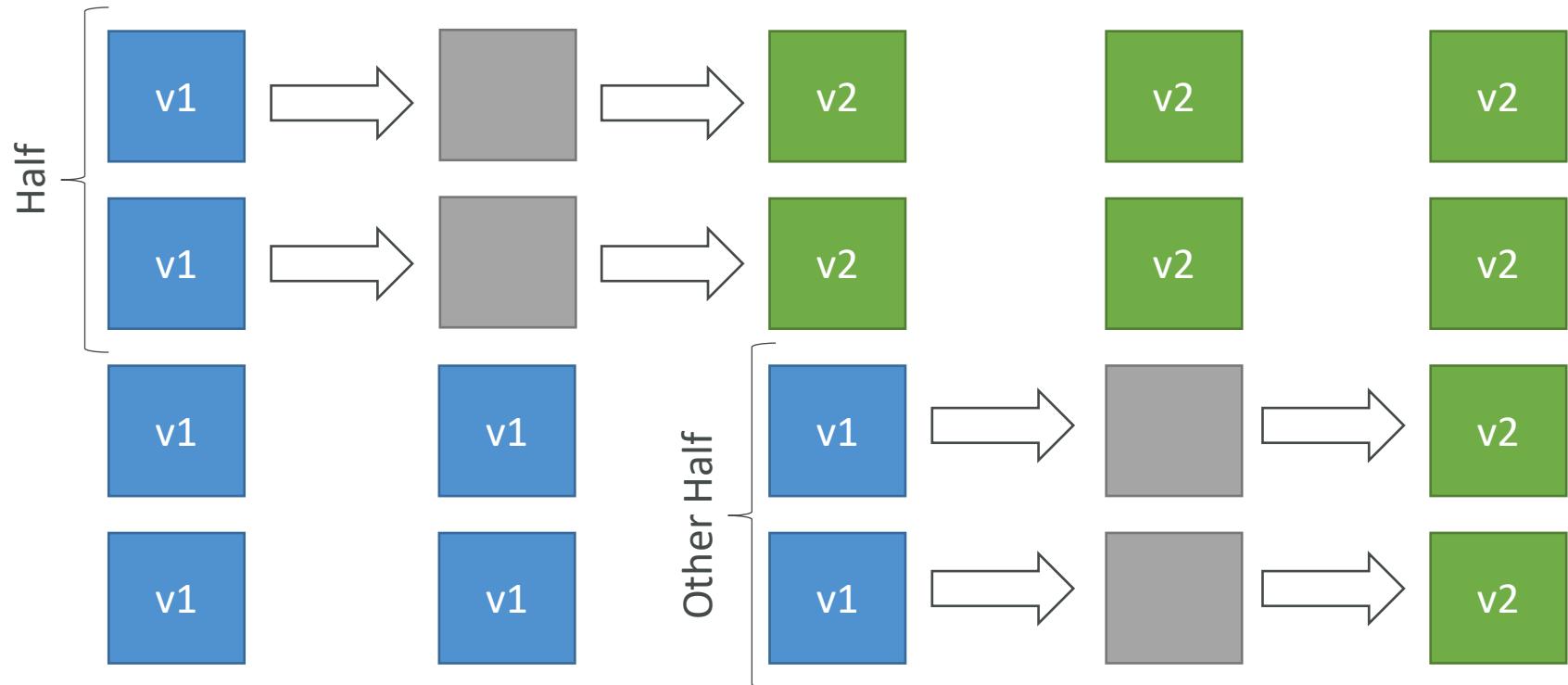
# AWS CodeDeploy Deploy & Hooks Order

Event	Start time	End time	Duration	Status
ApplicationStop	Sept 26, 2018 7:51:29 AM UTC	Sept 26, 2018 7:51:29 AM UTC	less than one second	Succeeded
DownloadBundle	Sept 26, 2018 7:51:30 AM UTC	Sept 26, 2018 7:51:30 AM UTC	less than one second	Succeeded
BeforeInstall	Sept 26, 2018 7:51:31 AM UTC	Sept 26, 2018 7:51:32 AM UTC	2 secs	Succeeded
Install	Sept 26, 2018 7:51:33 AM UTC	Sept 26, 2018 7:51:33 AM UTC	less than one second	Succeeded
AfterInstall	Sept 26, 2018 7:51:34 AM UTC	Sept 26, 2018 7:51:34 AM UTC	less than one second	Succeeded
ApplicationStart	Sept 26, 2018 7:51:35 AM UTC	Sept 26, 2018 7:51:35 AM UTC	less than one second	Succeeded
ValidateService	Sept 26, 2018 7:51:36 AM UTC	Sept 26, 2018 7:51:36 AM UTC	less than one second	Succeeded
BeforeAllowTraffic	Sept 26, 2018 7:51:49 AM UTC	Sept 26, 2018 7:51:49 AM UTC	less than one second	Succeeded
AllowTraffic	Sept 26, 2018 7:51:50 AM UTC	Sept 26, 2018 7:52:11 AM UTC	21 secs	Succeeded
AfterAllowTraffic	Sept 26, 2018 7:52:12 AM UTC	Sept 26, 2018 7:52:12 AM UTC	less than one second	Succeeded

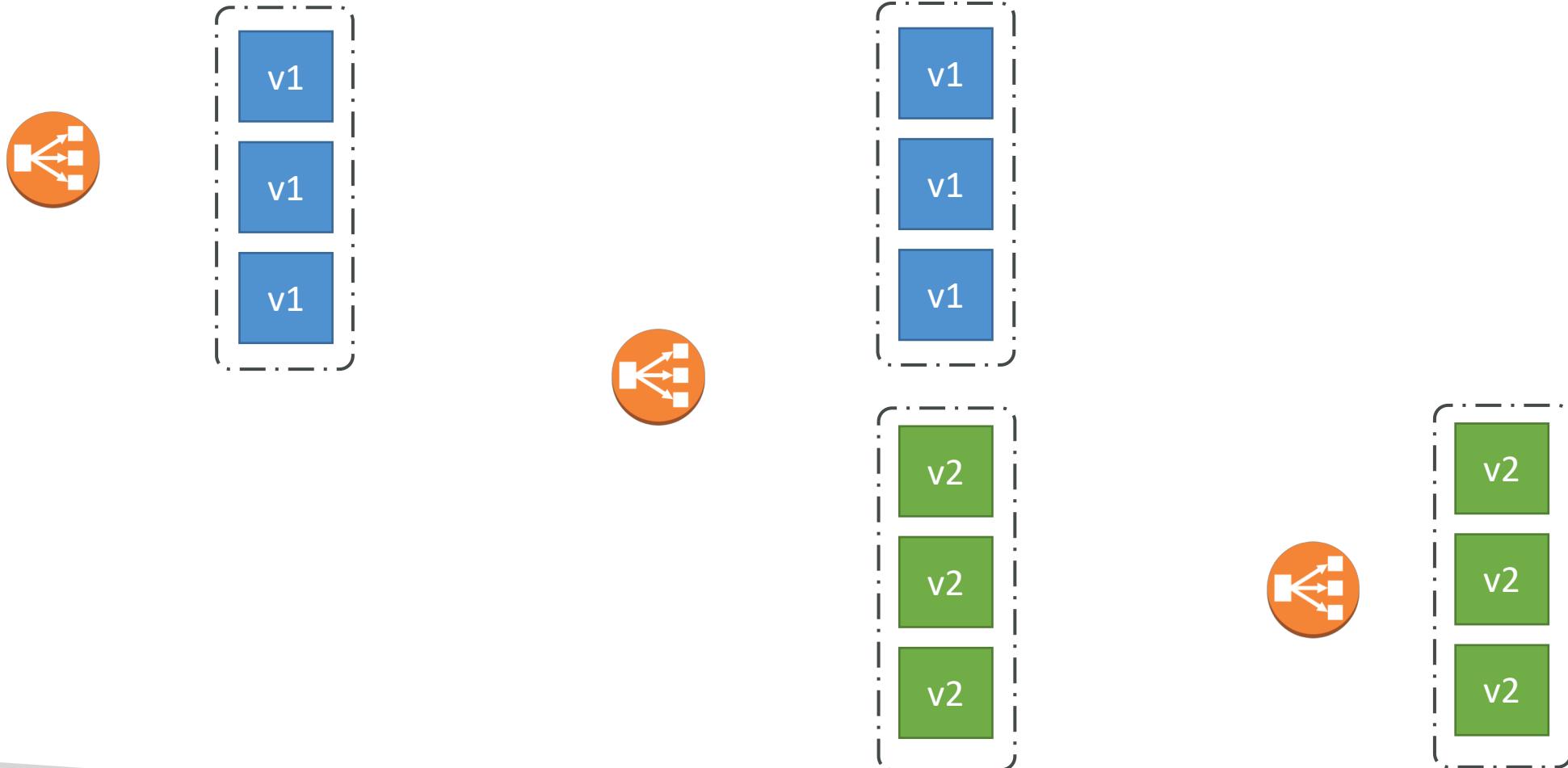
# AWS CodeDeploy Deployment Config

- Configs:
  - One at a time: one instance at a time, one instance fails => deployment stops
  - Half at a time: 50%
  - All at once: quick but no healthy host, downtime. Good for dev
  - Custom: min healthy host = 75%
- Failures:
  - Instances stay in “failed state”
  - New deployments will first be deployed to “failed state” instances
  - To rollback: redeploy old deployment or enable automated rollback for failures
- Deployment Targets:
  - Set of EC2 instances with tags
  - Directly to an ASG
  - Mix of ASG / Tags so you can build deployment segments
  - Customization in scripts with DEPLOYMENT\_GROUP\_NAME environment variables

# In Place Deployment – Half at a time



# Blue Green Deployment



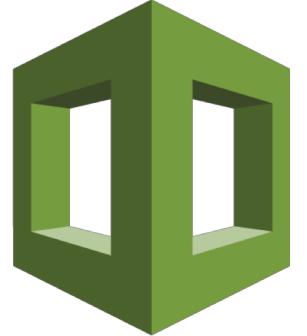
# AWS CloudFormation

Managing your infrastructure as code

# Infrastructure as Code

- Currently, we have been doing a lot of manual work
- All this manual work will be very tough to reproduce:
  - In another region
  - in another AWS account
  - Within the same region if everything was deleted
- Wouldn't it be great, if all our infrastructure was... code?
- That code would be deployed and create / update / delete our infrastructure

# What is 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 machines using this security group
  - I want two Elastic IPs for these EC2 machines
  - I want an S3 bucket
  - I want a load balancer (ELB) in front of these machines
- Then CloudFormation creates those for you, in the **right order**, with the **exact configuration** that you specify

# 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 have to 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 the CloudFormation Designer
  - Using the console to input parameters, etc
- Automated way:
  - Editing templates in a YAML file
  - Using the AWS CLI (Command Line Interface) to deploy the templates
  - Recommended way when you fully want to automate your flow

# CloudFormation Building Blocks

Templates components (one course section for each):

1. Resources: your AWS resources declared in the template (MANDATORY)
2. Parameters: the dynamic inputs for your template
3. Mappings: the static variables for your template
4. Outputs: References to what has been created
5. Conditionals: List of conditions to perform resource creation
6. Metadata

Templates helpers:

1. References
2. Functions

Note:

# This is an introduction to CloudFormation

- It can take over 3 hours to properly learn and master CloudFormation
- This section is meant so you get a good idea of how it works
- We'll be slightly less hands-on than in other sections
  
- We'll learn everything we need to answer questions for the exam
- The exam does not require you to actually write CloudFormation
- The exam expects you to understand how to read CloudFormation

# Introductory Example

- We're going to create a simple EC2 instance.
  - Then we're going to create to add an Elastic IP to it
  - And we're going to add two security groups to it
  - For now, forget about the code syntax.
  - We'll look at the structure of the files later on
- 
- We'll see how in no-time, we are able to get started with CloudFormation!



# YAML Crash Course

```
1  invoice:      34843
2  date   :     2001-01-23
3  bill-to:
4    given  :   Chris
5    family :  Dumars
6    address:
7      lines: |
8        458 Walkman Dr.
9        Suite #292
10       city   : Royal Oak
11       state  : MI
12       postal : 48046
13 product:
14   - sku        : BL394D
15     quantity   : 4
16     description: Basketball
17     price      : 450.00
18   - sku        : BL4438H
19     quantity   : 1
20     description: Super Hoop
21     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!

# What are 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 224 types of resources (!)
- Resource types identifiers are of the form:

**AWS::aws-product-name::data-type-name**

# How do I find resources documentation?

- I can't teach you all of the 224 resources, but I can teach you how to learn how to use them.
- All the resources can be found here:  
<http://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):  
<http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-ec2-instance.html>

# Analysis of CloudFormation Template

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

# FAQ for resources

- Can I create a dynamic amount of resources?
  - No, you can't. Everything in the CloudFormation template has to be declared. You can't perform code generation there
- Is every AWS Service supported?
  - Almost. Only a select few niches are not there yet
  - You can work around that using AWS Lambda Custom Resources

# What are 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?

- 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 ☺

**Parameters:**

**SecurityGroupDescription:**

**Description:** Security Group Description  
**(Simple parameter)**

**Type:** String

# Parameters Settings

Parameters can be controlled by all these settings:

- **Type:**
  - String
  - Number
  - CommaDelimitedList
  - List<Type>
  - AWS Parameter (to help catch invalid values – match against existing values in the AWS Account)
- **Description**
- **Constraints**
  - ConstraintDescription (String)
  - Min/MaxLength
  - Min/MaxValue
  - Defaults
  - AllowedValues (array)
  - AllowedPattern (regexp)
  - NoEcho (Boolean)

# How to Reference a Parameter

- 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

```
DbSubnet1:  
  Type: AWS::EC2::Subnet  
  Properties:  
    VpcId: !Ref MyVPC
```

# Concept: Pseudo Parameters

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

Reference Value	Example Return Value
AWS::AccountId	1234567890
AWS::NotificationARNS	[arn:aws:sns:us-east-1:123456789012:MyTopic]
AWS::NoValue	Does not return a value.
AWS::Region	us-east-2
AWS::StackId	arn:aws:cloudformation:us-east-1:123456789012:stack/MyStack/1c2fa620-982a-11e3-aff7-50e2416294e0
AWS::StackName	MyStack

# What are 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, etc
- All the values are hardcoded within the template
- Example:

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

```
RegionMap:  
  us-east-1:  
    "32": "ami-6411e20d"  
    "64": "ami-7a11e213"  
  us-west-1:  
    "32": "ami-c9c7978c"  
    "64": "ami-cfc7978a"  
  eu-west-1:  
    "32": "ami-37c2f643"  
    "64": "ami-31c2f645"
```

# 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

# Fn::FindInMap

## Accessing Mapping Values

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

```
AWSTemplateFormatVersion: "2010-09-09"
Mappings:
  RegionMap:
    us-east-1:
      "32": "ami-6411e20d"
      "64": "ami-7a11e213"
    us-west-1:
      "32": "ami-c9c7978c"
      "64": "ami-cfc7978a"
    eu-west-1:
      "32": "ami-37c2f643"
      "64": "ami-31c2f645"
    ap-southeast-1:
      "32": "ami-66f28c34"
      "64": "ami-60f28c32"
    ap-northeast-1:
      "32": "ami-9c03a89d"
      "64": "ami-a003a8a1"
Resources:
  myEC2Instance:
    Type: "AWS::EC2::Instance"
    Properties:
      ImageId: !FindInMap [RegionMap, !Ref "AWS::Region", 32]
      InstanceType: m1.small
```

# What are 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
- You can't delete a CloudFormation Stack if its outputs are being referenced by another CloudFormation stack

# Outputs Example

- 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

# 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 too.

```
Resources:  
  MySecureInstance:  
    Type: AWS::EC2::Instance  
    Properties:  
      AvailabilityZone: us-east-1a  
      ImageId: ami-a4c7edb2  
      InstanceType: t2.micro  
      SecurityGroups:  
        - !ImportValue SSHSecurityGroup
```

# What are conditions used for?

- 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

# Using a Condition

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

```
Resources:
```

```
  MountPoint:
```

```
    Type: "AWS::EC2::VolumeAttachment"
```

```
    Condition: CreateProdResources
```

# CloudFormation

## Must Know Intrinsic Functions

- Ref
- Fn::GetAtt
- Fn::FindInMap
- Fn::ImportValue
- Fn::Join
- Fn::Sub
- Condition Functions (Fn::If, Fn::Not, Fn::Equals, etc...)

# 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 (ex: EC2 ID)
- The shorthand for this in YAML is `!Ref`

```
DbSubnet1:  
  Type: AWS::EC2::Subnet  
  Properties:  
    VpcId: !Ref MyVPC
```

# 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.
- For example: the AZ of an EC2 machine!

**Resources:**

**EC2Instance:**

**Type:** "AWS::EC2::Instance"

**Properties:**

**ImageId:** ami-1234567

**InstanceType:** t2.micro

**NewVolume:**

**Type:** "AWS::EC2::Volume"

**Condition:** CreateProdResources

**Properties:**

**Size:** 100

**AvailabilityZone:**

**!GetAtt** EC2Instance.AvailabilityZone

# Fn::FindInMap

## Accessing Mapping Values

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

```
AWSTemplateFormatVersion: "2010-09-09"
Mappings:
  RegionMap:
    us-east-1:
      "32": "ami-6411e20d"
      "64": "ami-7a11e213"
    us-west-1:
      "32": "ami-c9c7978c"
      "64": "ami-cfc7978a"
    eu-west-1:
      "32": "ami-37c2f643"
      "64": "ami-31c2f645"
    ap-southeast-1:
      "32": "ami-66f28c34"
      "64": "ami-60f28c32"
    ap-northeast-1:
      "32": "ami-9c03a89d"
      "64": "ami-a003a8a1"
Resources:
  myEC2Instance:
    Type: "AWS::EC2::Instance"
    Properties:
      ImageId: !FindInMap [RegionMap, !Ref "AWS::Region", 32]
      InstanceType: m1.small
```

# Fn::ImportValue

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

```
Resources:  
  MySecureInstance:  
    Type: AWS::EC2::Instance  
    Properties:  
      AvailabilityZone: us-east-1a  
      ImageId: ami-a4c7edb2  
      InstanceType: t2.micro  
      SecurityGroups:  
        - !ImportValue SSHSecurityGroup
```

# Fn::Join

- Join values with a delimiter

```
!Join [ delimiter, [ comma-delimited list of values ] ]
```

- This creates “a:b:c”

```
!Join [ ":", [ a, b, c ] ]
```

# Function Fn::Sub

- Fn::Sub, or !Sub as a shorthand, is used to substitute variables from a text. It's a very handy function that will allow you to fully customize your templates.
- For example, you can combine Fn::Sub with References or AWS Pseudo variables!
- String must contain \${VariableName} and will substitute them

```
!Sub
- String
- { Var1Name: Var1Value, Var2Name: Var2Value }
```

```
!Sub String
```

# 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

# AWS Monitoring, Troubleshooting & Audit

CloudWatch, X-Ray and CloudTrail

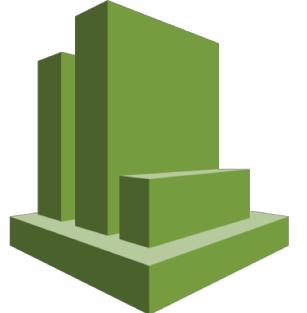
# Why Monitoring is Important

- 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 10 dimensions per metric
- Metrics have **timestamps**
- Can create CloudWatch dashboards of metrics

# AWS CloudWatch 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 more prompt scale 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)

# AWS CloudWatch Custom Metrics

- Possibility to define and send your own custom metrics to CloudWatch
- Ability to use dimensions (attributes) to segment metrics
  - Instance.id
  - Environment.name
- Metric resolution:
  - Standard: 1 minute
  - High Resolution: up to 1 second (*StorageResolution* API parameter) – Higher cost
- Use API call **PutMetricData**
- Use exponential back off in case of throttle errors

# AWS CloudWatch Alarms



- Alarms are used to trigger notifications for any metric
- Alarms can go to Auto Scaling, EC2 Actions, SNS notifications
- 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: can only choose 10 sec or 30 sec

# AWS CloudWatch Logs

- Applications can send logs to CloudWatch using the SDK
- CloudWatch can collect log from:
  - 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
  - CloudWatch log agents: for example on EC2 machines
  - Route53: Log DNS queries
- CloudWatch Logs can go to:
  - Batch exporter to S3 for archival
  - Stream to ElasticSearch cluster for further analytics

# AWS CloudWatch Logs

- CloudWatch Logs can use filter expressions
- Logs storage architecture:
  - Log groups: arbitrary name, usually representing an application
  - Log stream: instances within application / log files / containers
- Can define log expiration policies (never expire, 30 days, etc..)
- Using the AWS CLI we can tail CloudWatch logs
- To send logs to CloudWatch, make sure IAM permissions are correct!
- Security: encryption of logs using KMS at the Group Level

# AWS CloudWatch Events

- Schedule: Cron jobs
- Event Pattern: Event rules to react to a service doing something
  - Ex: CodePipeline state changes!
- Triggers to Lambda functions, SQS/SNS/Kinesis Messages
- CloudWatch Event creates a small JSON document to give information about the change

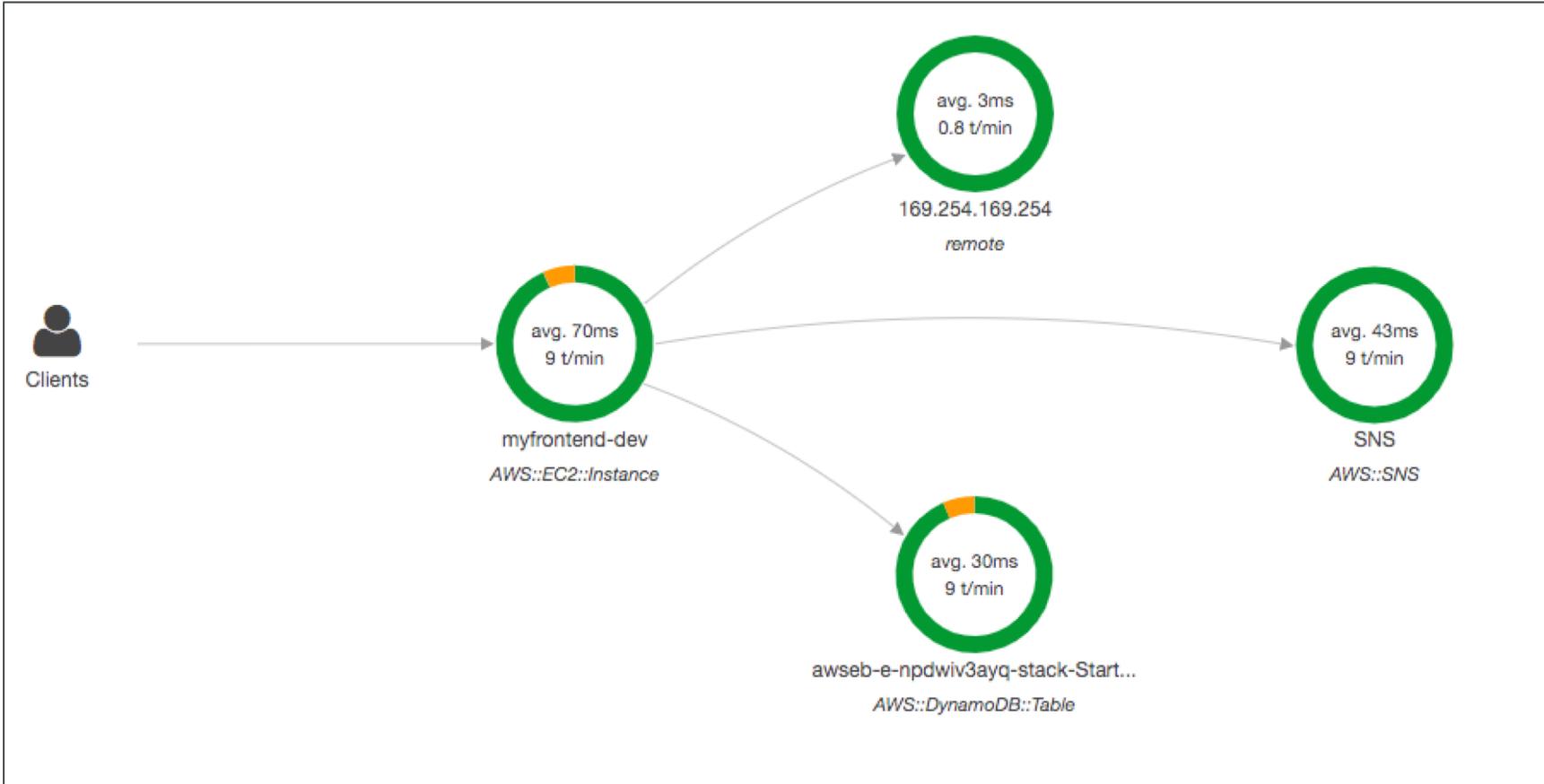
# AWS X-Ray



- 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

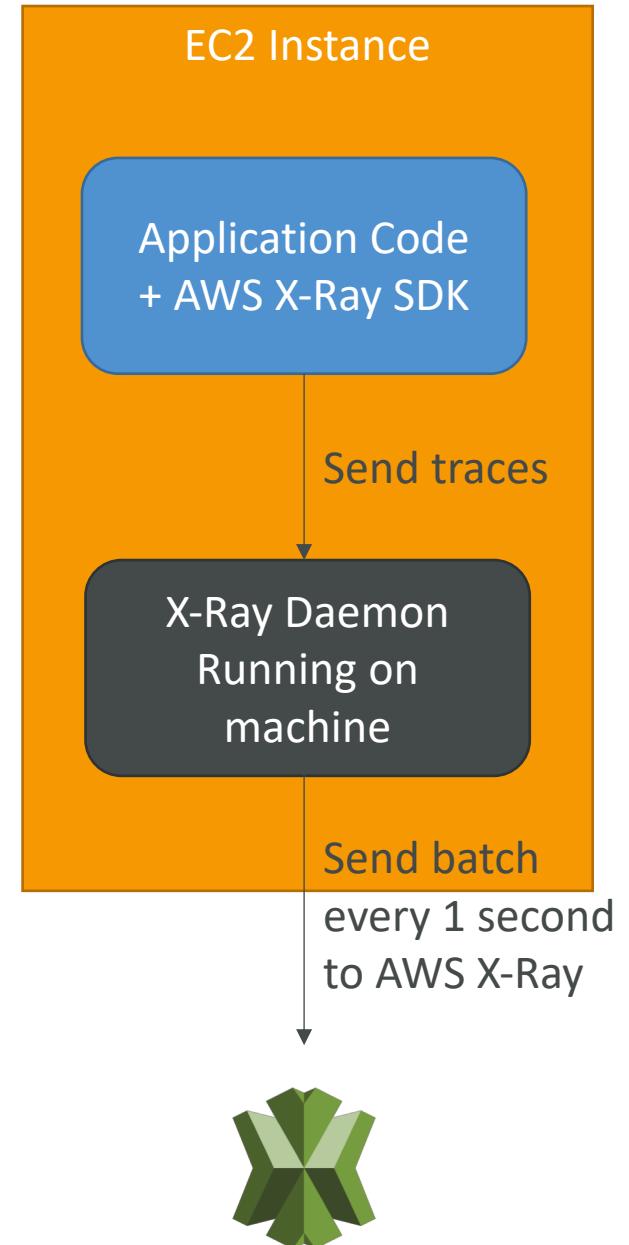
## How to enable it?

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)

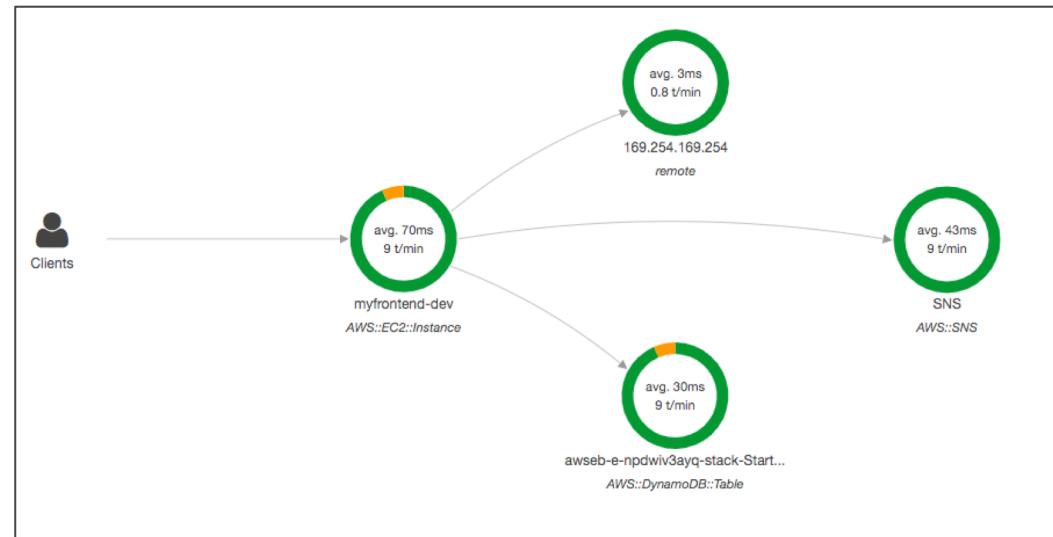
2) Install the X-Ray daemon or enable X-Ray AWS Integration

- 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

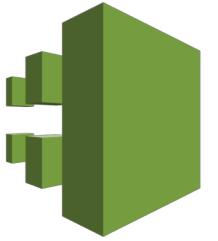
- X-Ray service collects data from all the different services
- Service map is computed from all the segments and traces
- X-Ray is graphical, so even non technical people can help troubleshoot



# 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

# 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
- If a resource is deleted in AWS, look into CloudTrail first!

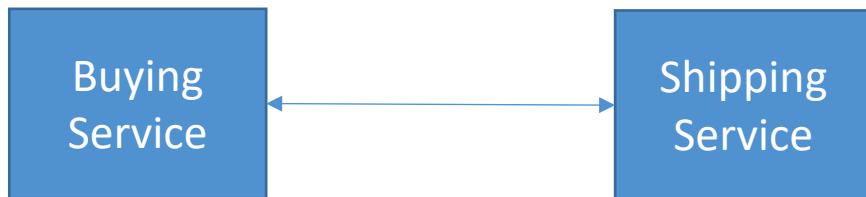
# AWS Integration & Messaging

SQS, SNS & Kinesis

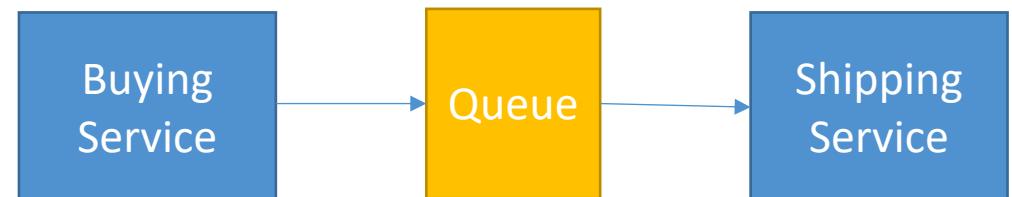
# Section Introduction

- 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)**

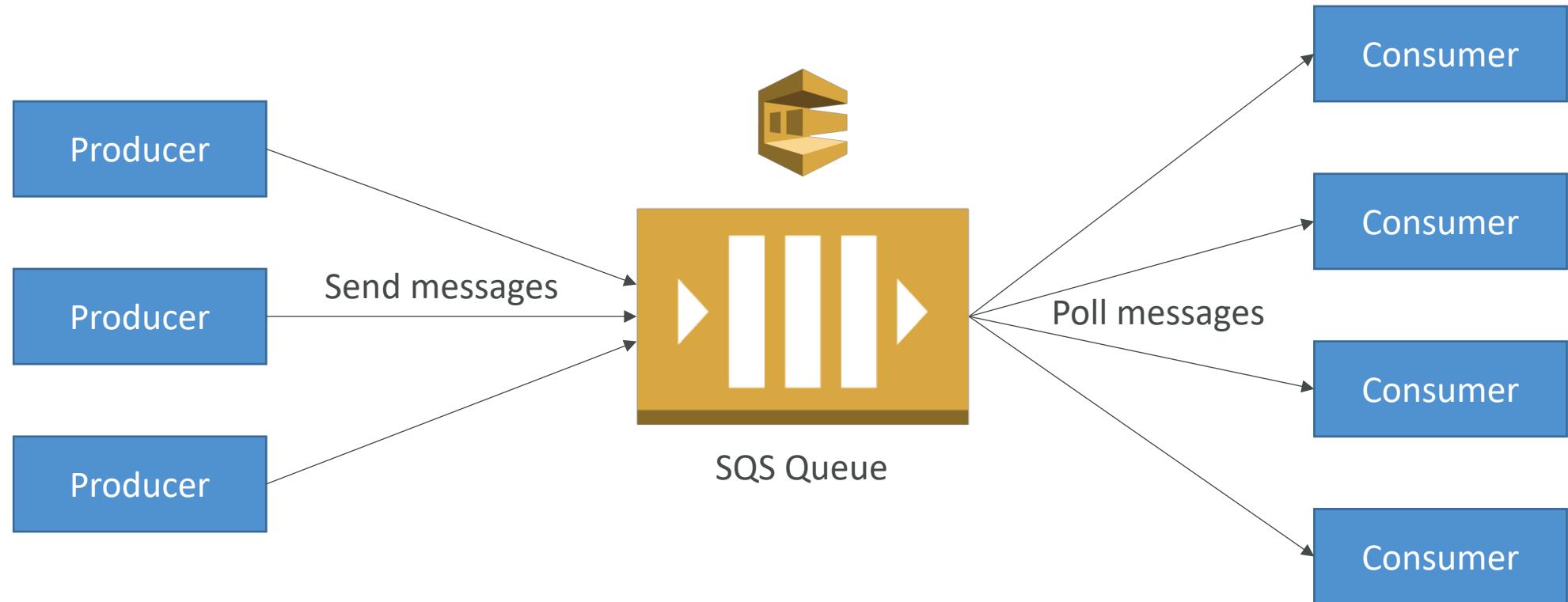


# Section Introduction

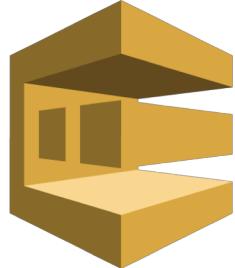
- Synchronous between applications can be problematic if there are sudden spikes of traffic
- What if you need to suddenly encode 1000 videos but usually it's 10?
- In that case, it's better to **decouple** your applications,
  - using SQS: queue model
  - using SNS: pub/sub model
  - using Kinesis: real-time streaming model
- These services can scale independently from our application!

# AWS SQS

## What's a queue?



# AWS SQS – Standard Queue



- Oldest offering (over 10 years old)
- Fully managed
- Scales from 1 message per second to 10,000s per second
- Default retention of messages: 4 days, maximum of 14 days
- No limit to how many messages can be in the queue
- Low latency (<10 ms on publish and receive)
- Horizontal scaling in terms of number of consumers
- Can have duplicate messages (at least once delivery, occasionally)
- Can have out of order messages (best effort ordering)
- Limitation of 256KB per message sent

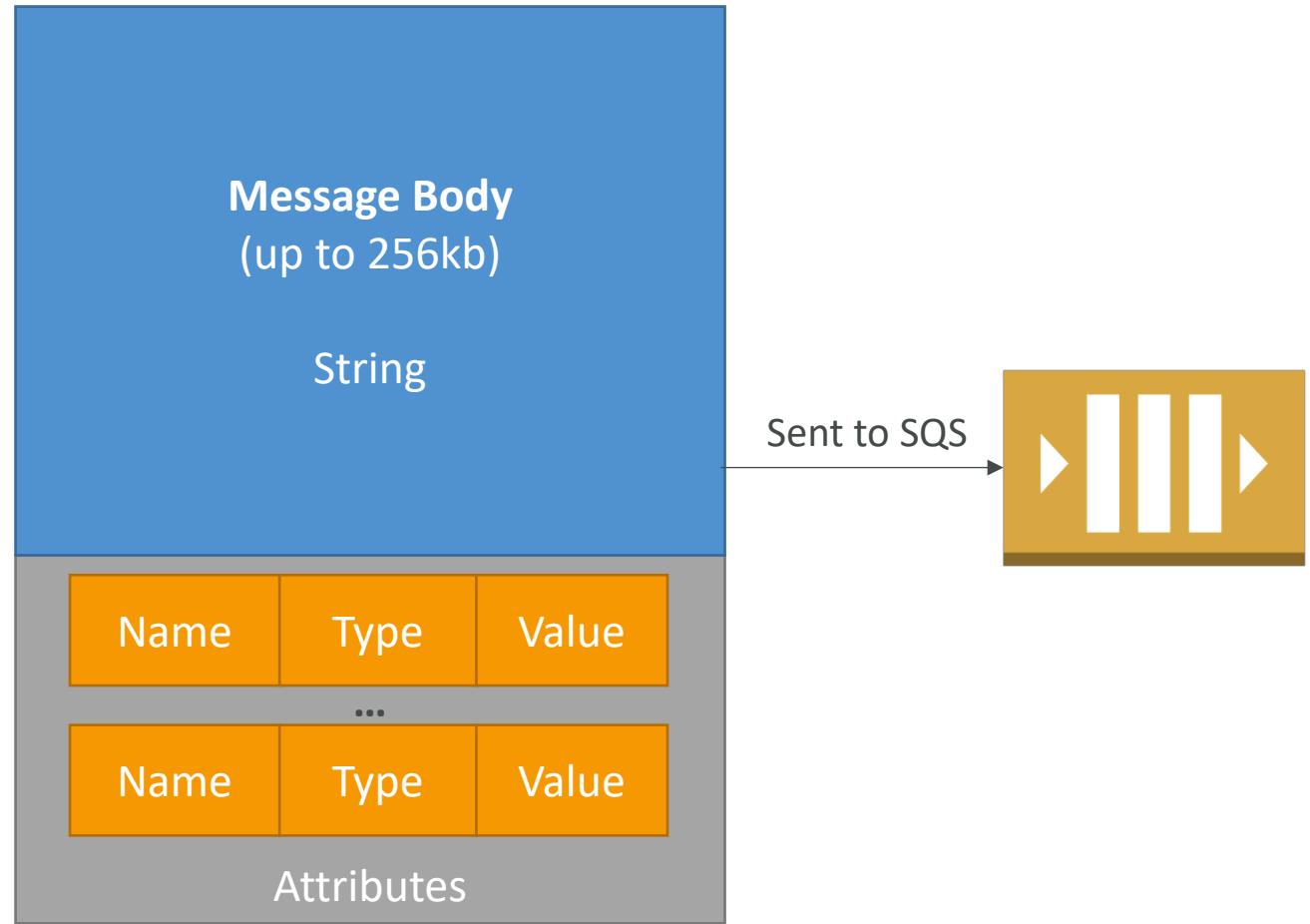
# AWS SQS – Delay Queue

- 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 using the `DelaySeconds` parameter



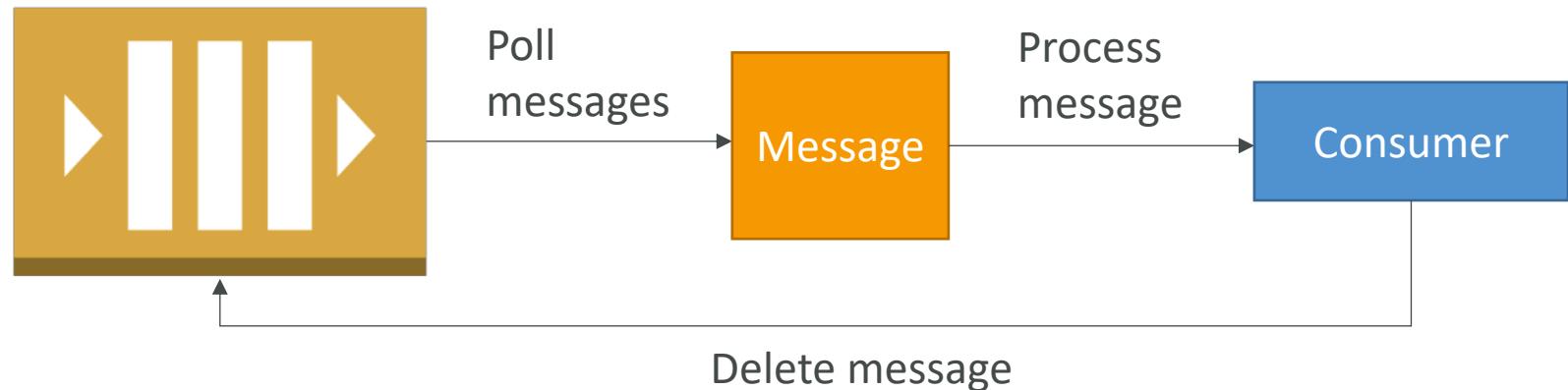
# SQS – Producing Messages

- Define Body
- Add message attributes (metadata – optional)
- Provide Delay Delivery (optional)
- Get back
  - Message identifier
  - MD5 hash of the body



# SQS – Consuming Messages

- Consumers...
- Poll SQS for messages (receive up to 10 messages at a time)
- Process the message within the visibility timeout
- Delete the message using the message ID & receipt handle

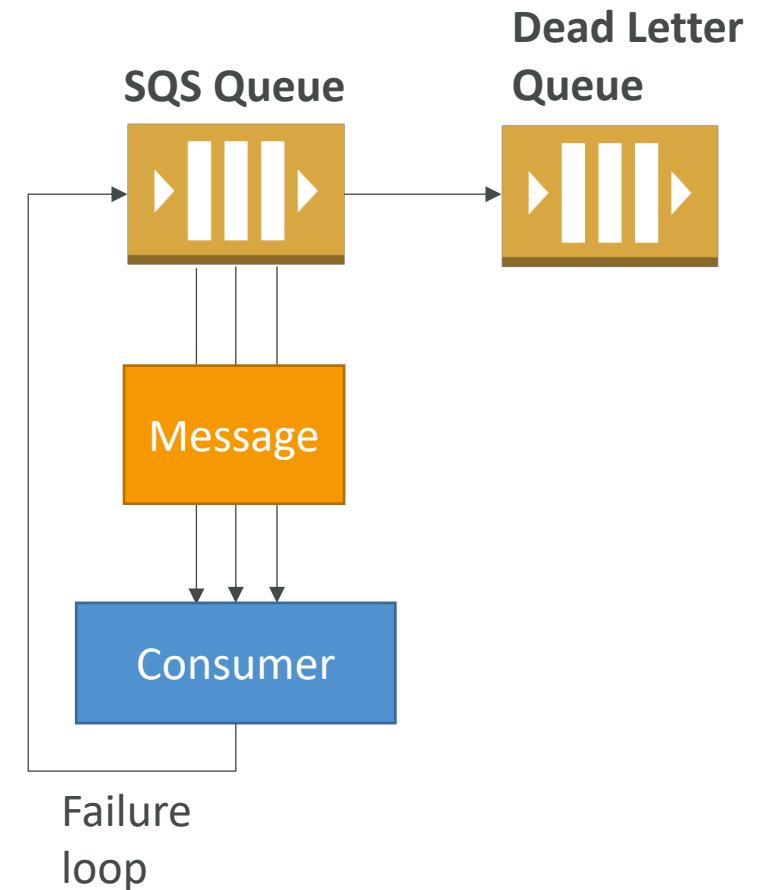


# SQS – Visibility timeout

- When a consumer polls a message from a queue, the message is “invisible” to other consumers for a defined period... the **VisibilityTimeout**:
  - Set between 0 seconds and 12 hours (default 30 seconds)
  - If too high (15 minutes) and consumer fails to process the message, you must wait a long time before processing the message again
  - If too low (30 seconds) and consumer needs time to process the message (2 minutes), another consumer will receive the message and the message will be processed more than once
- **ChangeMessageVisibility** API to change the visibility while processing a message
- **DeleteMessage** API to tell SQS the message was successfully processed

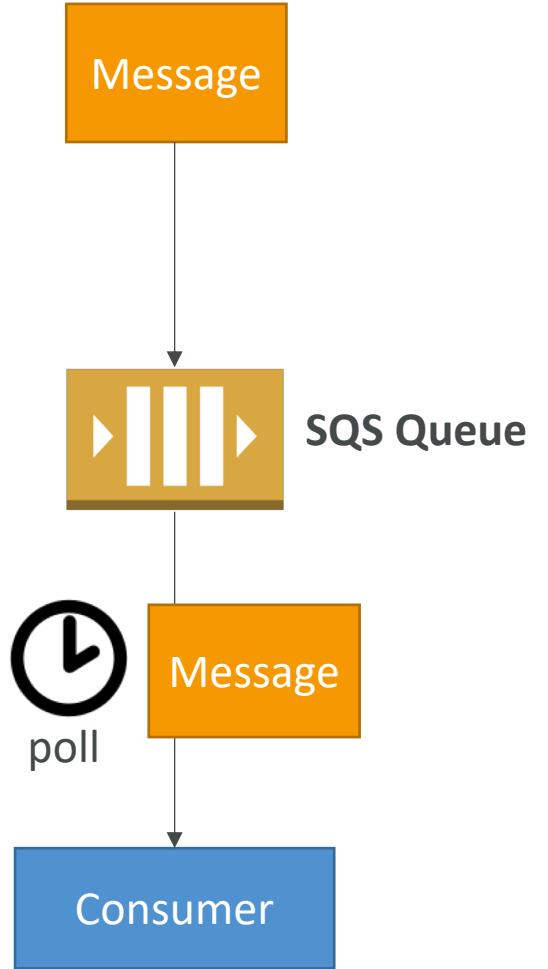
# AWS SQS – Dead Letter Queue

- 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 – it's called a "redrive policy"
- After the threshold is exceeded, the message goes into a dead letter queue (DLQ)
- We have to create a DLQ first and then designate it dead letter queue
- Make sure to process the messages in the DLQ before they expire!

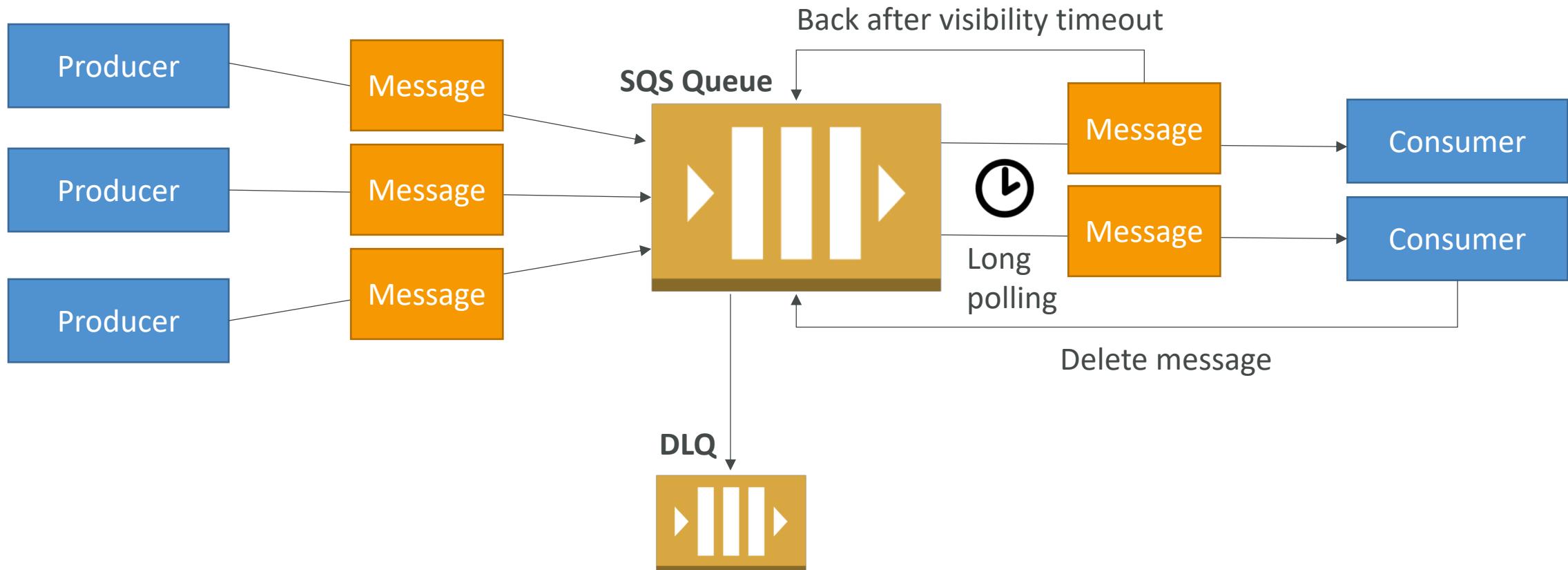


# AWS SQS - Long Polling

- When a consumer requests message from the queue, it can optionally “wait” for messages to arrive if there are none in the queue
- This is called Long Polling
- Long Polling decreases the number of API calls made to SQS while increasing the efficiency and 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 `WaitTimeSeconds`

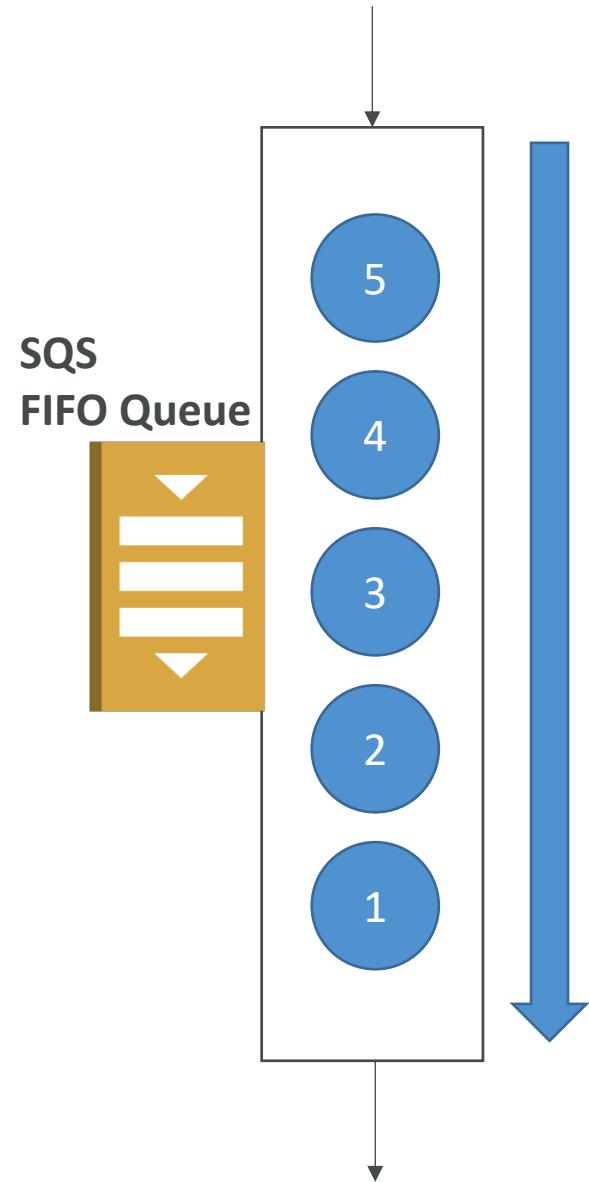


# SQS Message consumption flow diagram



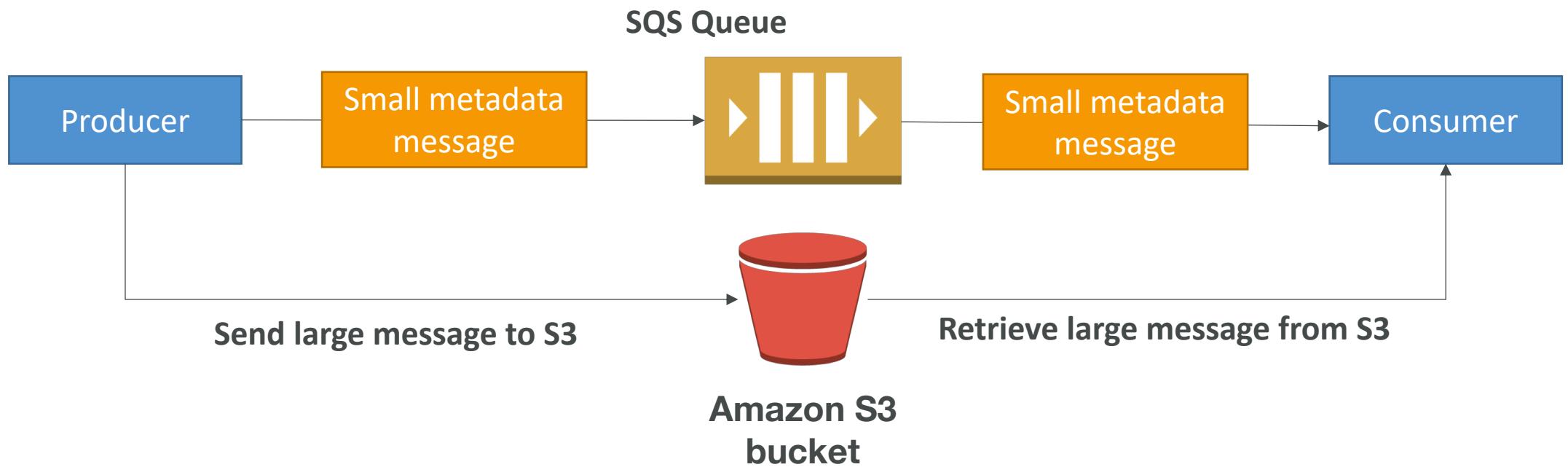
# AWS SQS – FIFO Queue

- Newer offering (First In - First out) – not available in all regions!
- Name of the queue must end in .fifo
- Lower throughput (up to 3,000 per second with batching, 300/s without)
- Messages are processed in order by the consumer
- Messages are sent exactly once
- No per message delay (only per queue delay)
- Ability to do content based de-duplication
- 5-minute interval de-duplication using “Duplication ID”
- Message Groups:
  - Possibility to group messages for FIFO ordering using “Message GroupID”
  - Only one worker can be assigned per message group so that messages are processed in order
  - Message group is just an extra tag on the message!



# SQS Extended Client

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



# AWS SQS Security

- Encryption in flight using the HTTPS endpoint
- Can enable SSE (Server Side Encryption) using KMS
  - Can set the CMK (Customer Master Key) we want to use
  - Can set the data key reuse period (between 1 minute and 24 hours)
    - Lower and KMS API will be used often
    - Higher and KMS API will be called less
  - SSE only encrypts the body, not the metadata (message ID, timestamp, attributes)
- IAM policy must allow usage of SQS
- SQS queue access policy
  - Finer grained control over IP
  - Control over the time the requests come in
- No VPC Endpoint, must have internet access to access SQS

# SQS – Must know API

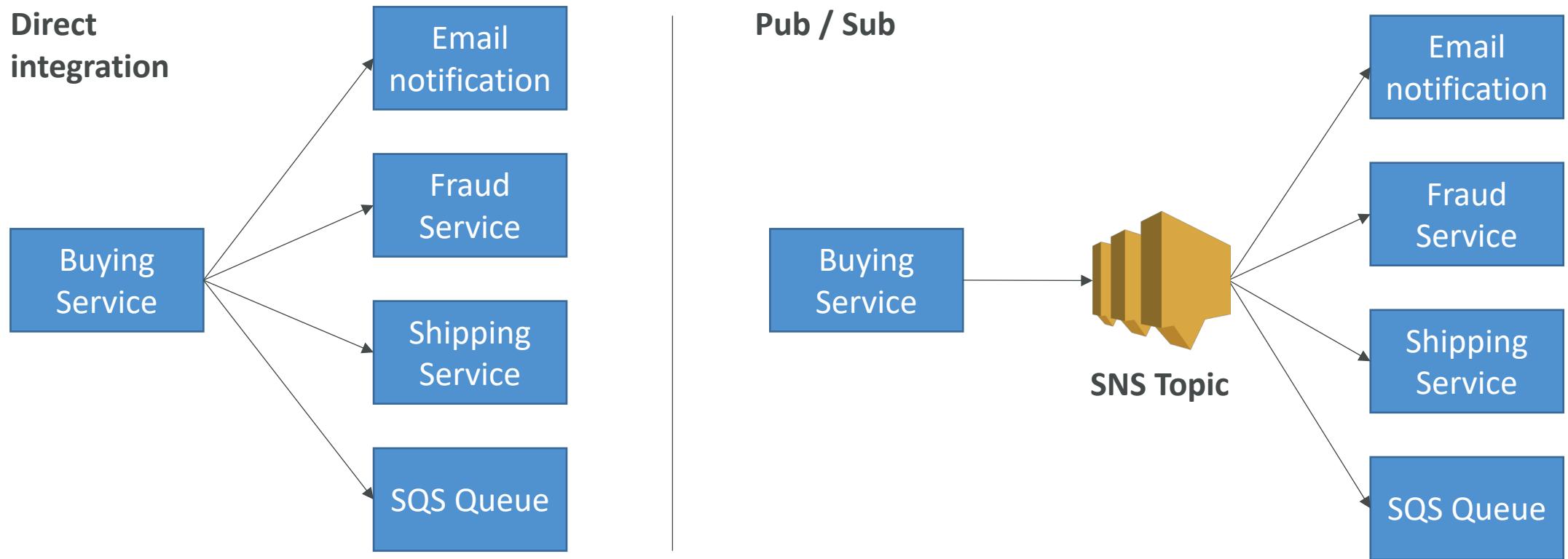
- CreateQueue, DeleteQueue
  - PurgeQueue: delete all the messages in queue
  - SendMessage, ReceiveMessage, DeleteMessage
  - ChangeMessageVisibility: change the timeout
- 
- Batch APIs for SendMessage, DeleteMessage, ChangeMessageVisibility helps decrease your costs

# AWS SQS Use Cases

- Decouple applications  
(for example to handle payments asynchronously)
- Buffer writes to a database  
(for example a voting application)
- Handle large loads of messages coming in  
(for example an email sender)
- SQS can be integrated with Auto Scaling through CloudWatch!

# AWS SNS

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



# AWS 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 10,000,000 subscriptions per topic
- 100,000 topics limit
- Subscribers can be:
  - SQS
  - HTTP / HTTPS (with delivery retries – how many times)
  - Lambda
  - Emails
  - SMS messages
  - Mobile Notifications

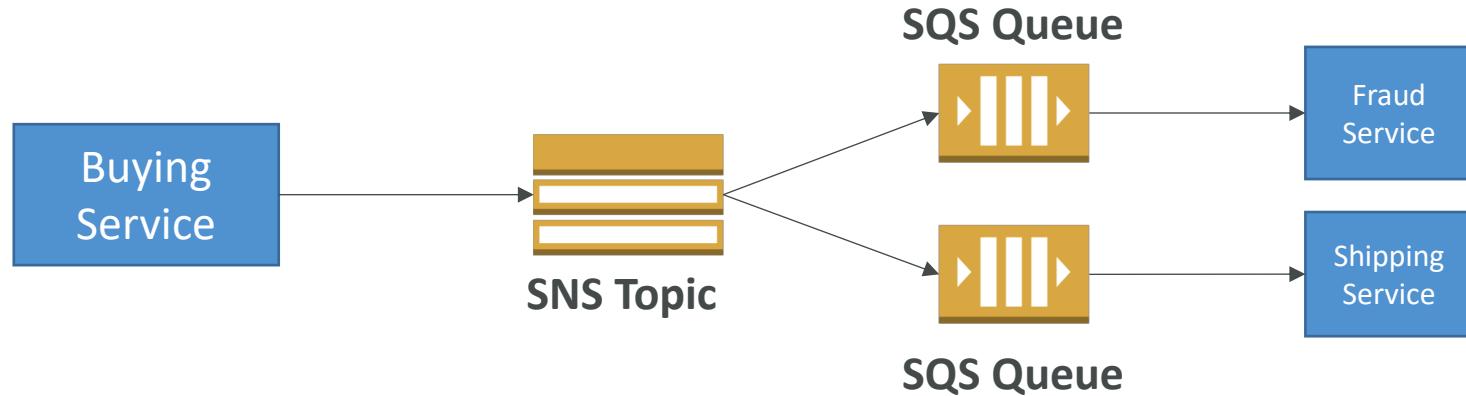
# SNS integrates with a lot of Amazon Products

- Some services can send data directly to SNS for notifications
- CloudWatch (for alarms)
- Auto Scaling Groups notifications
- Amazon S3 (on bucket events)
- CloudFormation (upon state changes => failed to build, etc)
- Etc...

# AWS SNS – How to publish

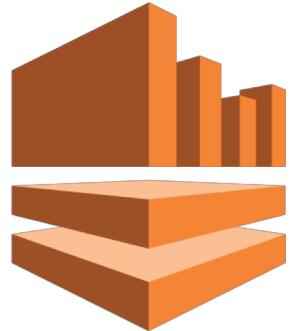
- Topic Publish (within your AWS Server – using the SDK)
  - Create a topic
  - Create a subscription (or many)
  - Publish to the topic
- Direct Publish (for mobile apps SDK)
  - Create a platform application
  - Create a platform endpoint
  - Publish to the platform endpoint
  - Works with Google GCM, Apple APNS, Amazon ADM...

# SNS + SQS: Fan Out



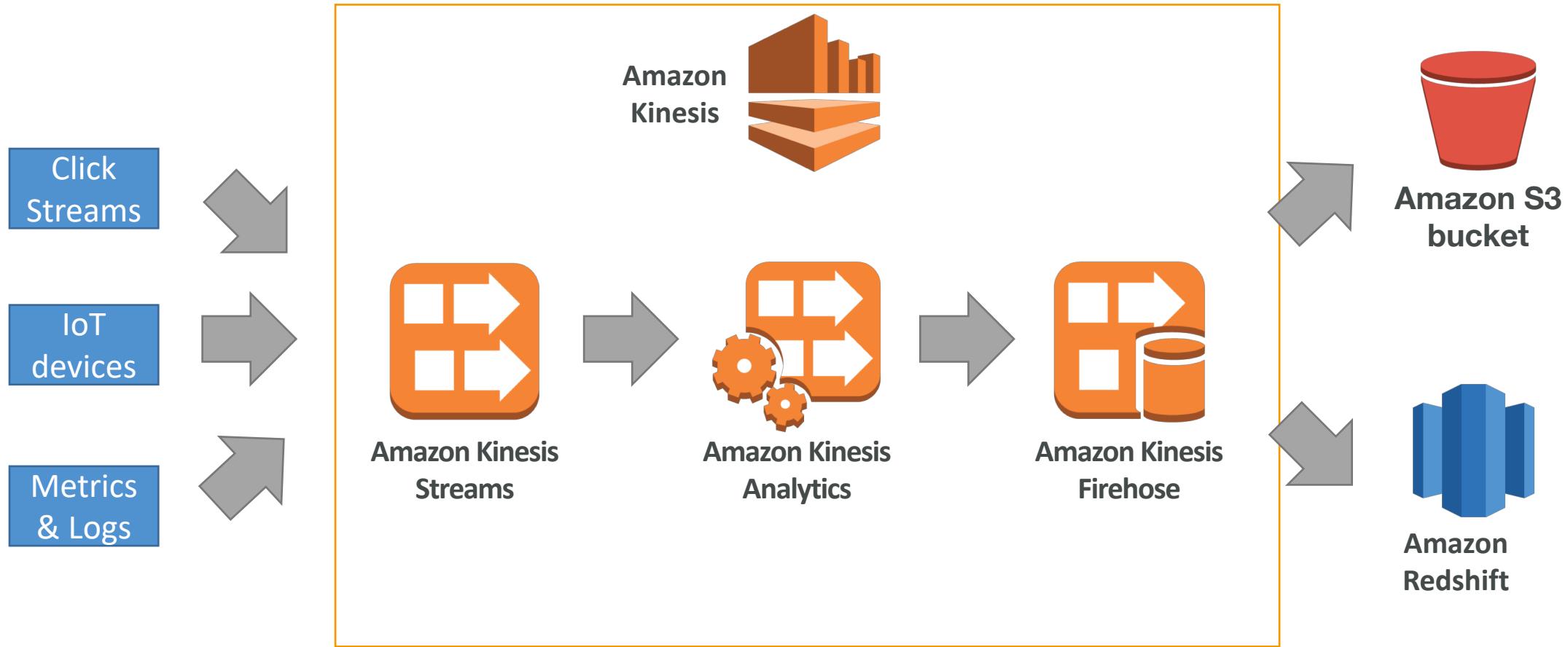
- Push once in SNS, receive in many SQS
- Fully decoupled
- No data loss
- Ability to add receivers of data later
- SQS allows for delayed processing
- SQS allows for retries of work
- May have many workers on one queue and one worker on the other queue

# AWS Kinesis Overview



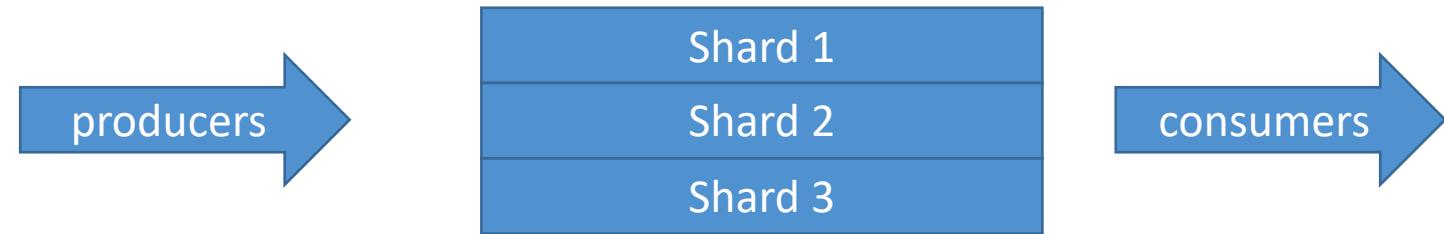
- Kinesis is a managed alternative to Apache Kafka
  - Great for application logs, metrics, IoT, clickstreams
  - Great for “real-time” big data
  - Great for streaming processing frameworks (Spark, NiFi, etc...)
  - Data is automatically replicated to 3 AZ
- 
- **Kinesis Streams:** low latency streaming ingest at scale
  - **Kinesis Analytics:** perform real-time analytics on streams using SQL
  - **Kinesis Firehose:** load streams into S3, Redshift, ElasticSearch...

# Kinesis



# Kinesis Streams Overview

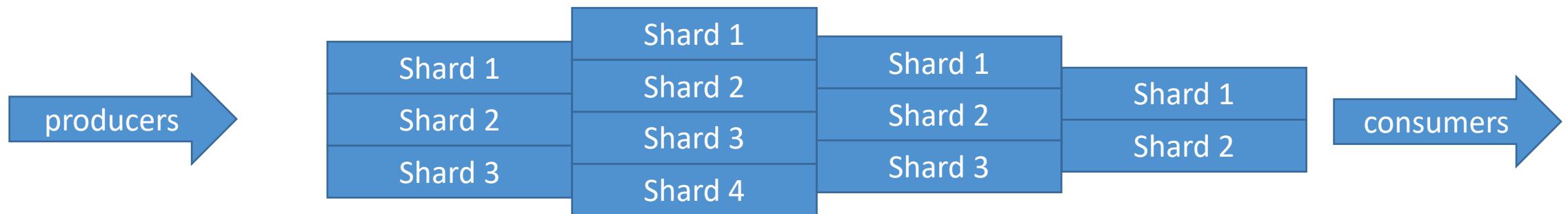
- Streams are divided in ordered Shards / Partitions



- Data retention is 1 day by default, can go up to 7 days
- Ability to reprocess / replay data
- Multiple applications can consume the same stream
- Real-time processing with scale of throughput
- Once data is inserted in Kinesis, it can't be deleted (immutability)

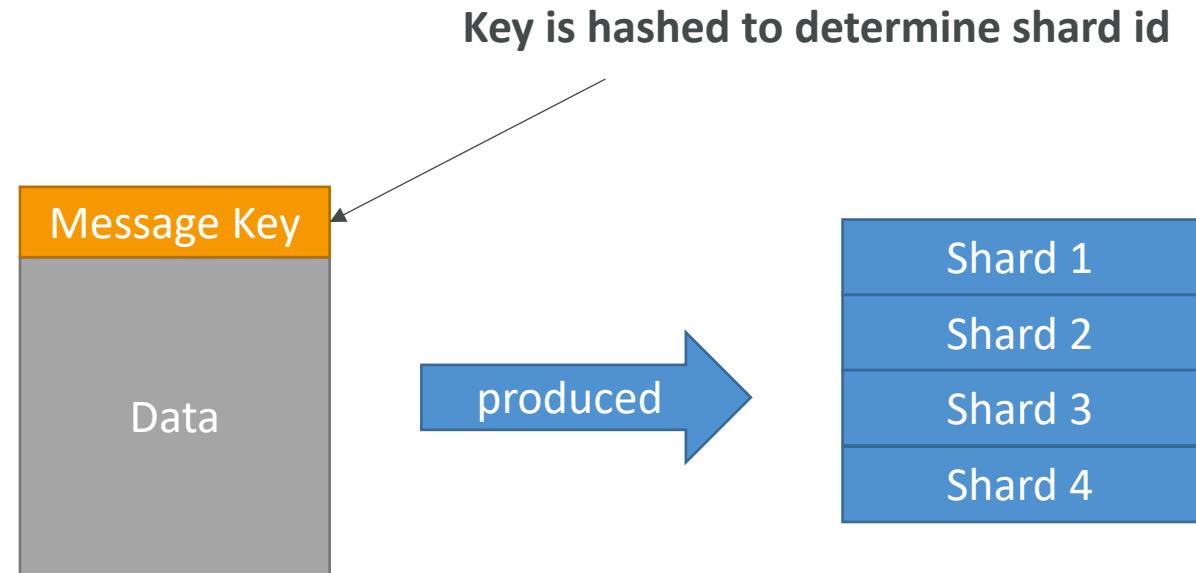
# Kinesis Streams Shards

- One stream is made of many different shards
- 1MB/s or 1000 messages/s at write PER SHARD
- 2MB/s at read PER SHARD
- Billing is per shard provisioned, can have as many shards as you want
- Batching available or per message calls.
- The number of shards can evolve over time (reshard / merge)
- Records are ordered per shard



# AWS Kinesis API – Put records

- PutRecord API + Partition key that gets hashed
- The same key goes to the same partition (helps with ordering for a specific key)
- Messages sent get a “sequence number”
- Choose a partition key that is highly distributed (helps prevent “hot partition”)
  - user\_id if many users
  - **Not** country\_id if 90% of the users are in one country
- Use Batching with PutRecords to reduce costs and increase throughput
- ProvisionedThroughputExceeded if we go over the limits
- Can use CLI, AWS SDK, or producer libraries from various frameworks

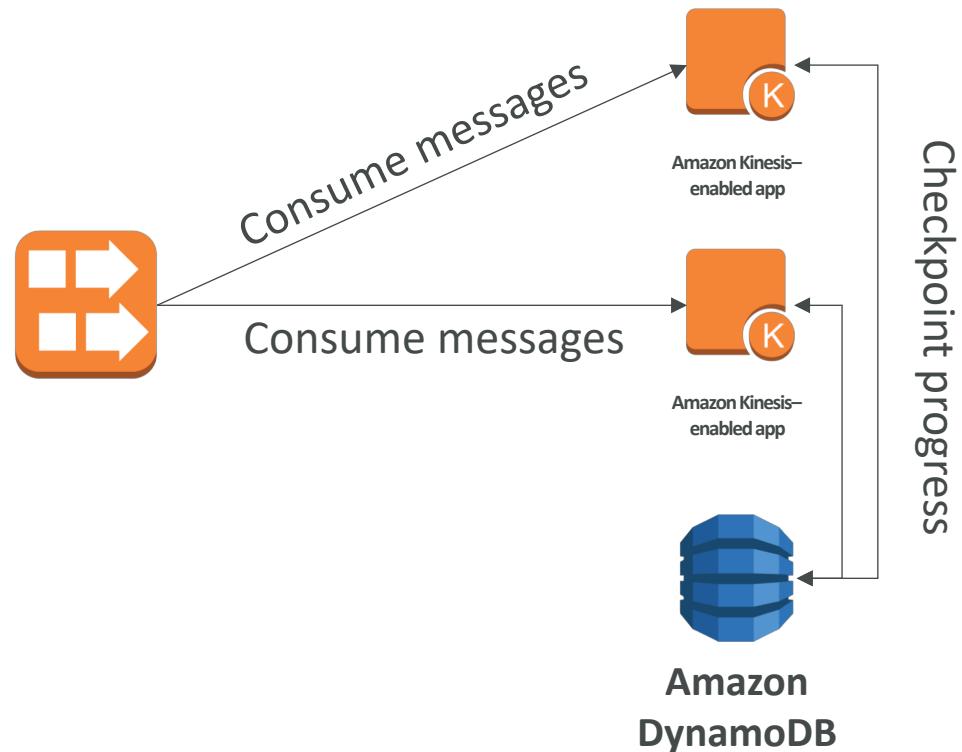


# AWS Kinesis API – Exceptions

- ProvisionedThroughputExceeded Exceptions
  - Happens when sending more data (exceeding MB/s or TPS for any shard)
  - Make sure you don't have a hot shard (such as your partition key is bad and too much data goes to that partition)
- Solution:
  - Retries with backoff
  - Increase shards (scaling)
  - Ensure your partition key is a good one

# AWS Kinesis API – Consumers

- Can use a normal consumer (CLI, SDK, etc...)
- Can use Kinesis Client Library (in Java, Node, Python, Ruby, .Net)
  - KCL uses DynamoDB to checkpoint offsets
  - KCL uses DynamoDB to track other workers and share the work amongst shards



# Kinesis Security

- Control access / authorization using IAM policies
- Encryption in flight using HTTPS endpoints
- Encryption at rest using KMS
- Possibility to encrypt / decrypt data client side (harder)
- VPC Endpoints available for Kinesis to access within VPC

# AWS Kinesis Data Analytics



- Perform real-time analytics on Kinesis Streams using SQL
- Kinesis Data Analytics:
  - Auto Scaling
  - Managed: no servers to provision
  - Continuous: real time
- Pay for actual consumption rate
- Can create streams out of the real-time queries

# AWS Kinesis Firehose

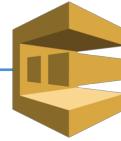


- Fully Managed Service, no administration
- Near Real Time (60 seconds latency)
- Load data into Redshift / Amazon S3 / ElasticSearch / Splunk
- Automatic scaling
- Support many data format (pay for conversion)
- Pay for the amount of data going through Firehose

# SQS vs SNS vs Kinesis

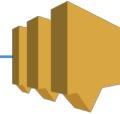
## SQS:

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



## SNS:

- Push data to many subscribers
- Up to 10,000,000 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



## Kinesis:

- Consumers “pull data”
- As many consumers as we want
- Possibility to replay data
- Meant for real-time big data, analytics and ETL
- Ordering at the shard level
- Data expires after X days
- Must provision throughput



# AWS Lambda

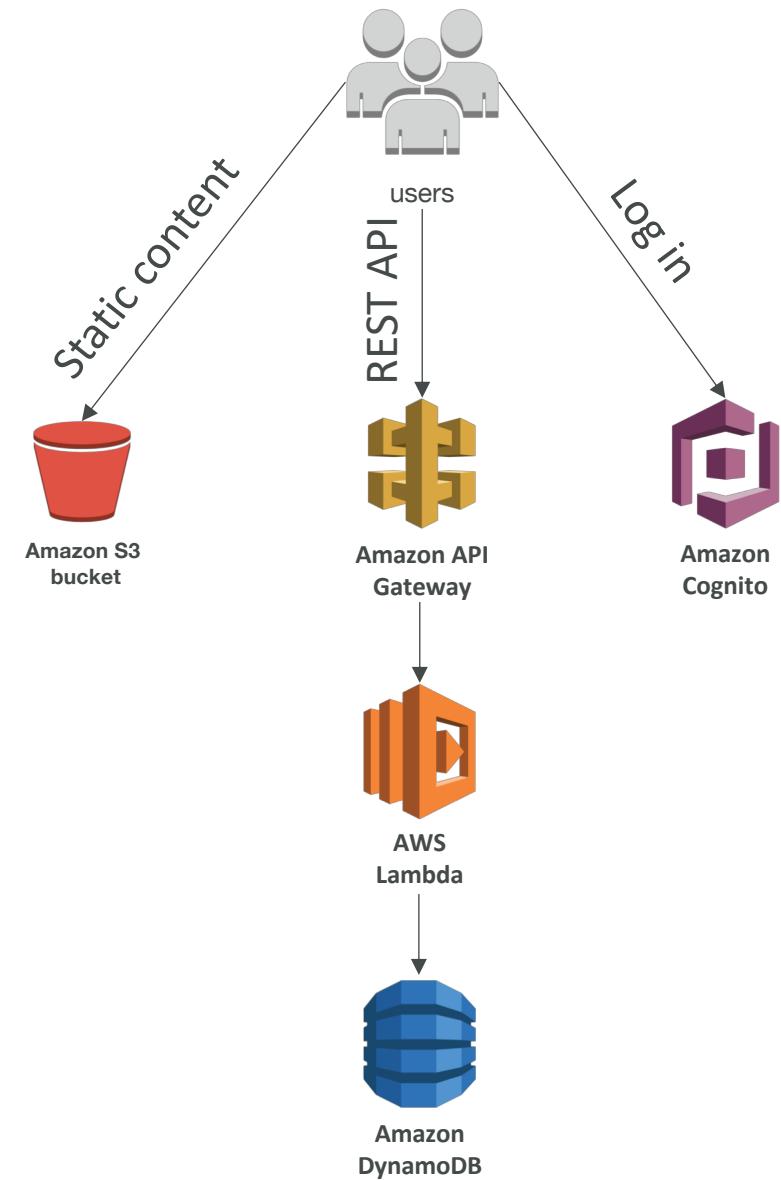
It's a serverless world

# 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 & Step Functions
- DynamoDB
- AWS Cognito
- AWS API Gateway
- Amazon S3
- AWS SNS & SQS
- AWS Kinesis
- Aurora Serverless



# 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 Stack
- Integrated with many programming languages
- Easy monitoring through AWS CloudWatch
- Easy to get more resources per functions (up to 3GB of RAM!)
- Increasing RAM will also improve CPU and network!

# AWS Lambda language support

- Node.js (JavaScript)
- Python
- Java (Java 8 compatible)
- C# (.NET Core)
- Golang
- C# / Powershell

# AWS Lambda Integrations

## Main ones



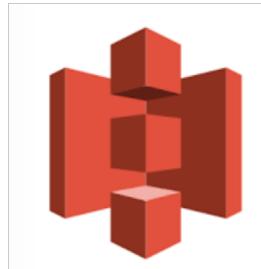
API Gateway



Kinesis



DynamoDB



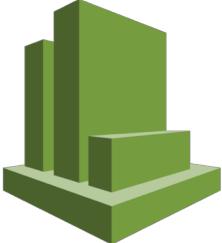
AWS S3 –  
Simple Storage Service



AWS IoT  
Internet of Things



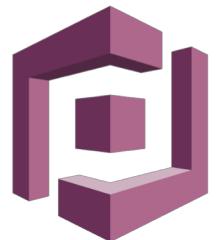
CloudWatch Events



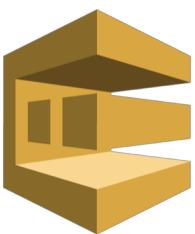
CloudWatch Logs



AWS SNS

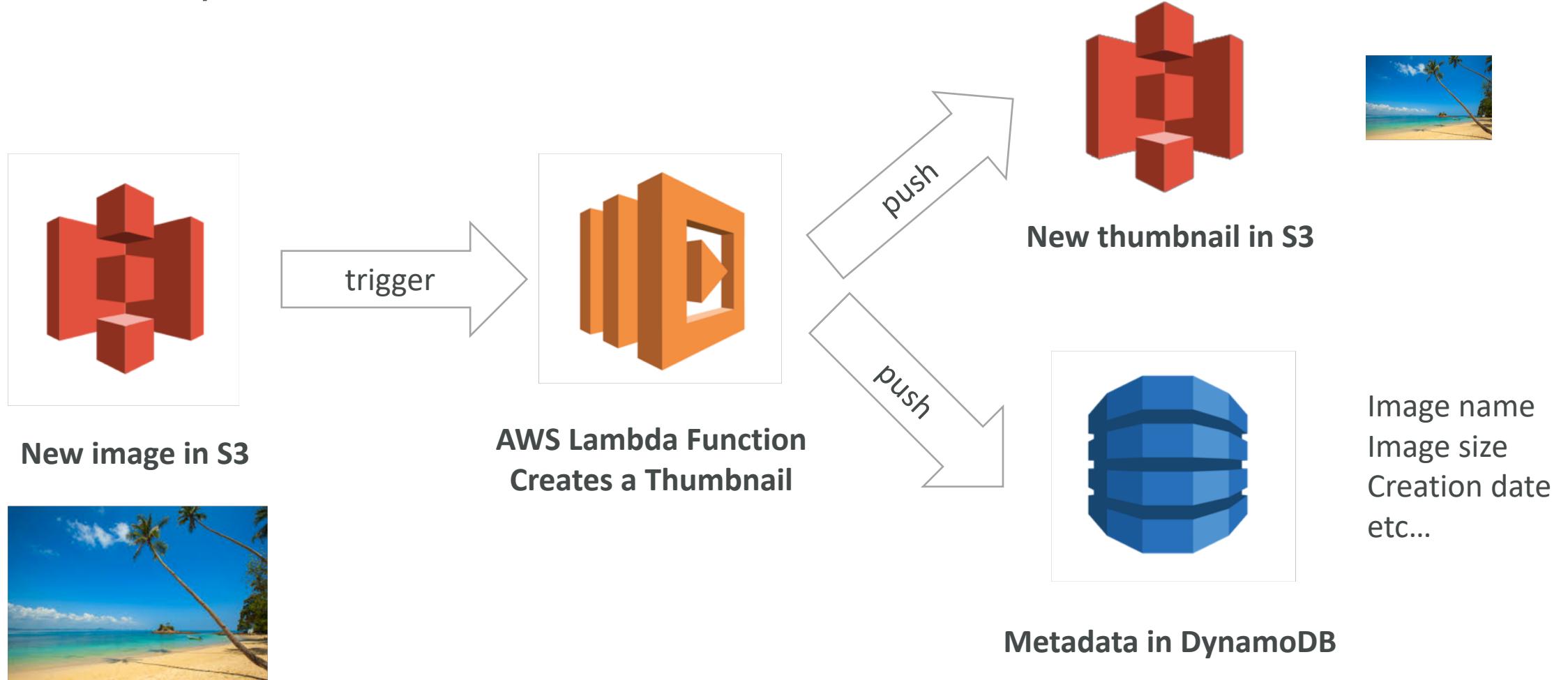


AWS Cognito



Amazon  
SQS

# Example: Serverless Thumbnail creation



# AWS Lambda Pricing (as of June 2018, us-east-1 region)

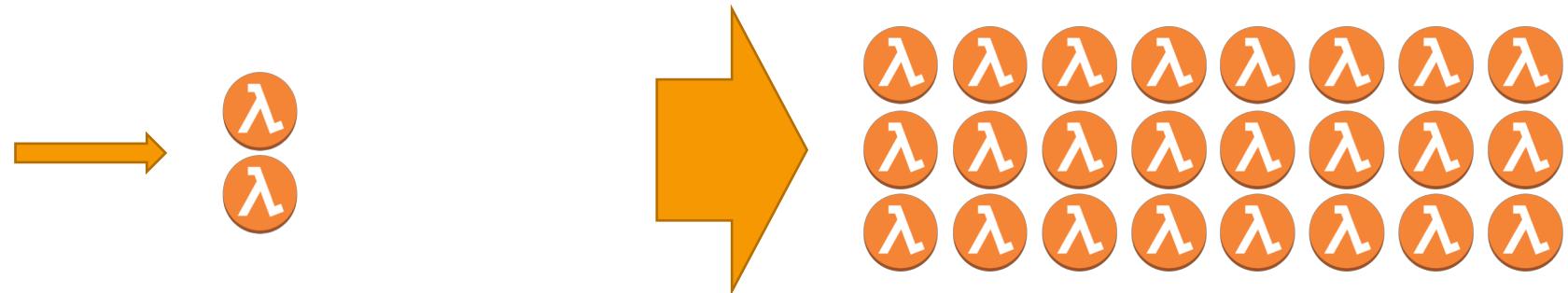
- You can find overall pricing information here:  
<https://aws.amazon.com/lambda/pricing/>
- Pay per calls:
  - First 1,000,000 requests are free
  - \$0.20 per 1 million requests thereafter (\$0.0000002 per request)
- Pay per duration: (in increment of 100ms)
  - 400,000 GB-seconds of compute time per month if 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

# AWS Lambda Configuration

- Timeout: default 3 seconds, max of 300s (Note: new limit 15 minutes)
- Environment variables
- Allocated memory (128M to 3G)
- Ability to deploy within a VPC + assign security groups
- IAM execution role must be attached to the Lambda function

# AWS Lambda Concurrency and Throttling

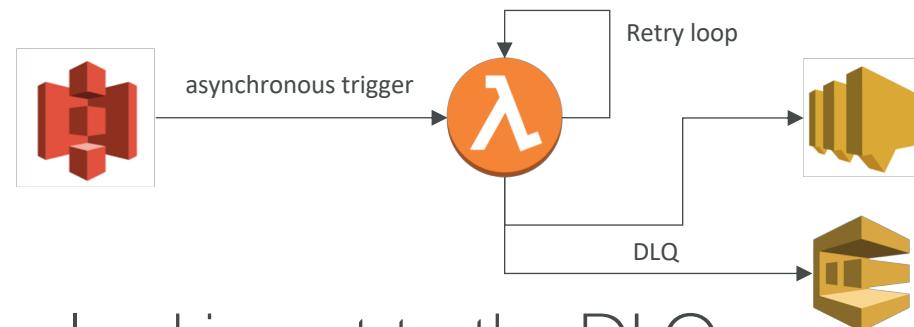
- Concurrency: up to 1000 executions (can be increased through ticket)



- Can set a “reserved concurrency” at the function level
- 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

# AWS Lambda Retries and DLQ

- If a lambda function asynchronous invocation fails, it will be retried twice
- After all retries, unprocessed events go to the Dead Letter Queue
- DLQ can be a SNS topic or SQS queue



- The original event payload is sent to the DLQ
- This is an easy way to debug what's wrong with your functions in production without changing the code
- Make sure the IAM execution role is correct for your Lambda function

# AWS Lambda Logging, Monitoring and Tracing

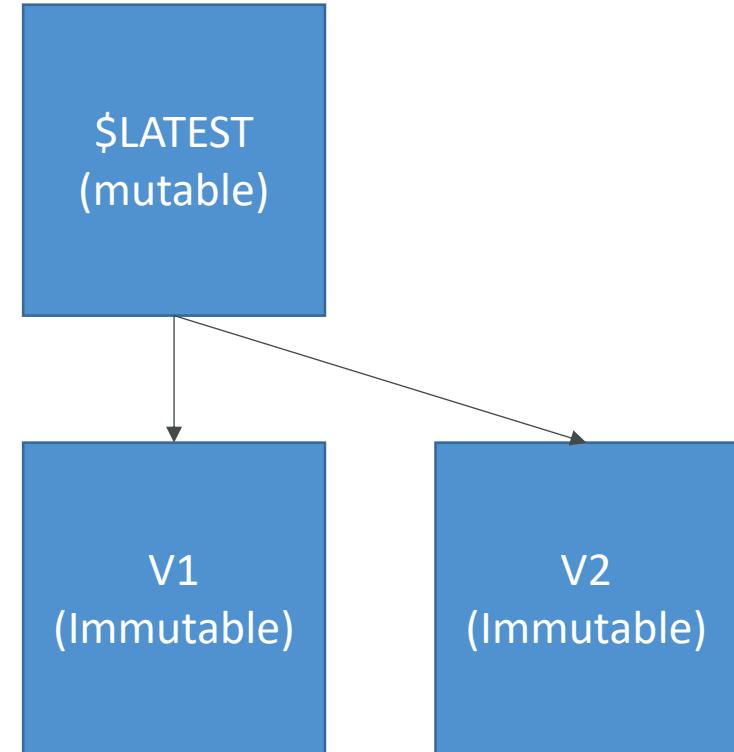
- CloudWatch:
  - AWS Lambda execution logs are stored in AWS CloudWatch Logs
  - AWS Lambda metrics are displayed in AWS CloudWatch Metrics
  - Make sure your AWS Lambda function has an execution role with an IAM policy that authorizes writes to CloudWatch
- X-Ray:
  - It's possible to trace Lambda with X-Ray
  - Enable in Lambda configuration (runs the X-Ray daemon for you)
  - Use AWS SDK in Code
  - Ensure Lambda Function has correct IAM Execution Role

# AWS Lambda Limits to Know

- Execution:
  - Memory allocation: 128 MB – 3008 MB (64 MB increments)
  - Maximum execution time: 300 seconds (5 minutes), now 15 minutes but 5 for exam
  - Disk capacity in the “function container” (in /tmp): 512 MB
  - Concurrency limits: 1000
- 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

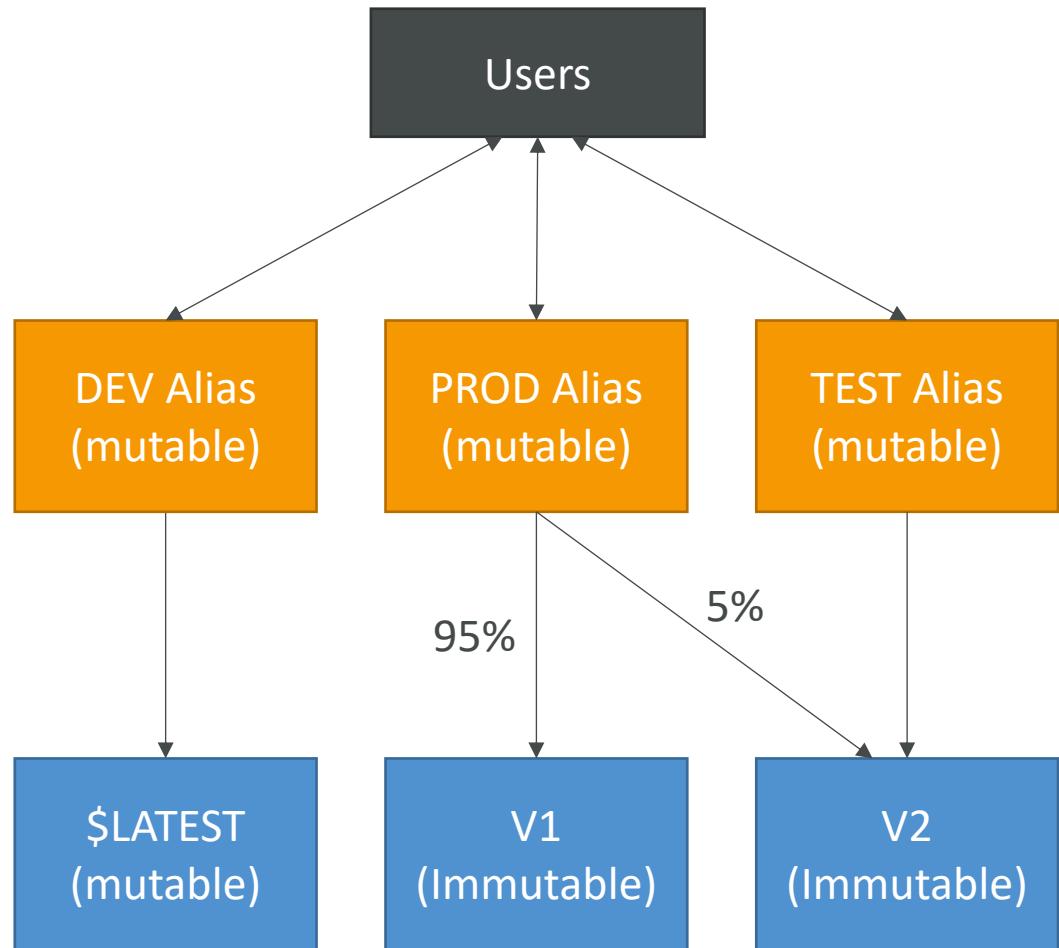
# AWS Lambda Versions

- 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 Blue / Green deployment by assigning weights to lambda functions
- Aliases enable stable configuration of our event triggers / destinations
- Aliases have their own ARNs





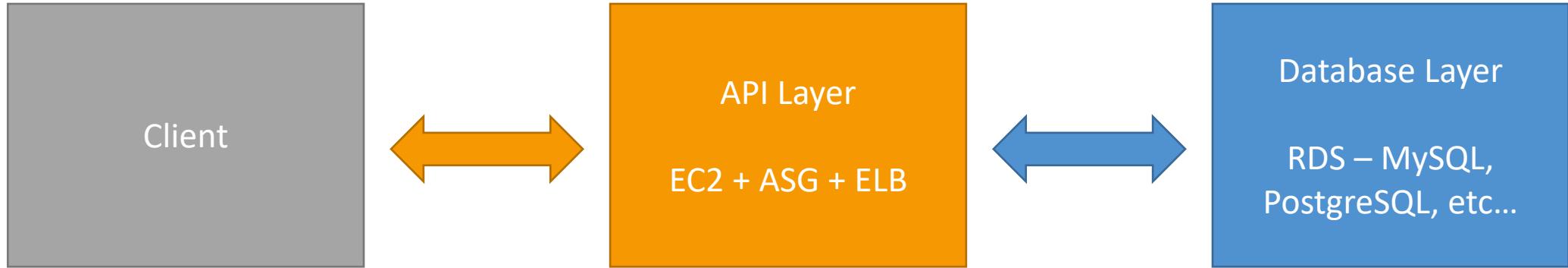
# AWS Lambda Best Practices

- 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... do not put these values directly 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
- Avoid using recursive code, never have a Lambda function call itself
- Don't put your Lambda function in a VPC unless you have to

# 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 join, aggregations, computations
- Vertical scaling (means usually getting a more powerful CPU / RAM / IO)

# NoSQL databases

- NoSQL databases are non-relational databases and are **distributed**
- NoSQL databases include MongoDB, DynamoDB, etc.
- NoSQL databases do not support join
- All the data that is needed for a query is present in one row
- NoSQL databases don't perform aggregations such as "SUM"
- **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

# DynamoDB



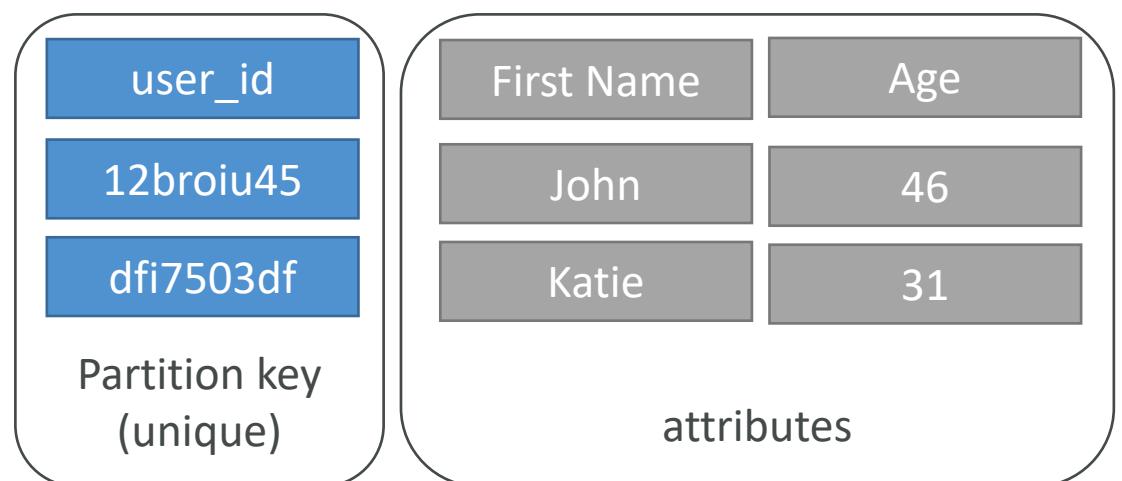
- Fully Managed, Highly available with replication across 3 AZ
- NoSQL database - not a relational database
- Scales to massive workloads, distributed database
- Millions of requests per seconds, trillions of row, 100s of TB of storage
- Fast and consistent in performance (low latency on retrieval)
- Integrated with IAM for security, authorization and administration
- Enables event driven programming with DynamoDB Streams
- Low cost and auto scaling capabilities

# 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 a 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

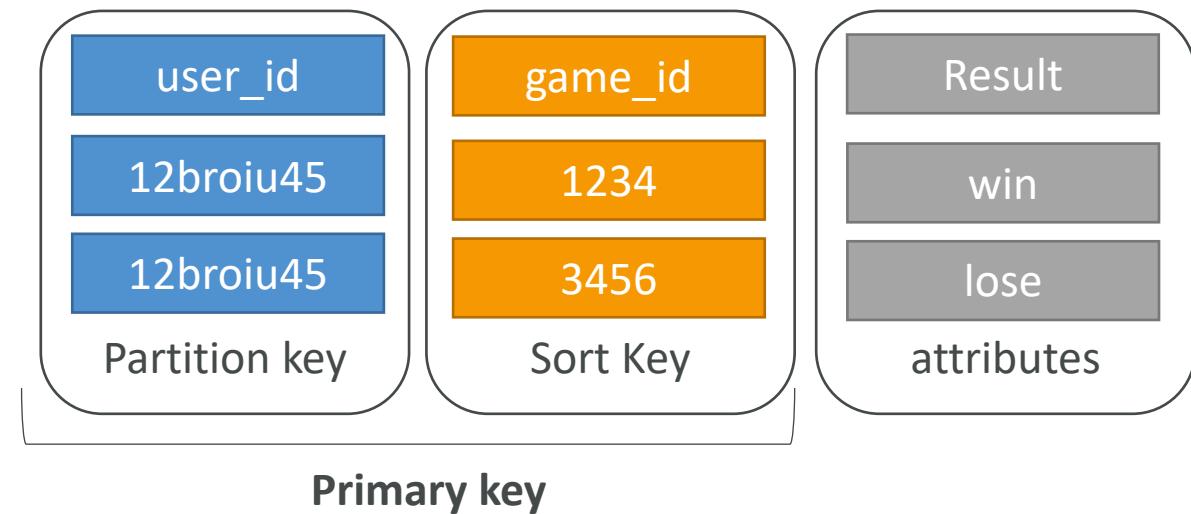
# DynamoDB – Primary Keys

- Option 1: Partition key only (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
- The combination must be unique
- Data is grouped by partition key
- Sort key == range key
- Example: users-games table
  - user\_id for the partition key
  - game\_id for the 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
- moving\_language doesn't take many values and may be skewed towards English so it's not a great partition key

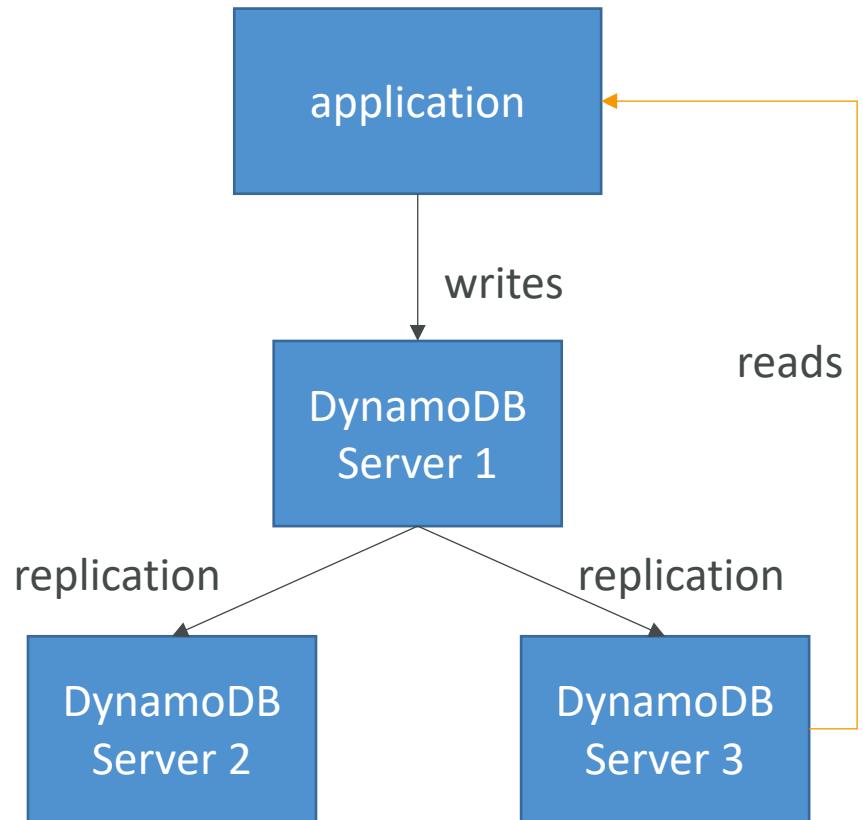
# DynamoDB – Provisioned Throughput

- 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 credit”
- If burst credit are empty, you’ll get a “ProvisionedThroughputException”.
- It’s then advised to do an exponential back-off retry

# DynamoDB – Write Capacity Units

- One *write capacity unit* represents one write per second for an item up to 1 KB in size.
- If the items are larger than 1 KB, more WCU are consumed
- **Example 1:** we write 10 objects per seconds of 2 KB each.
  - We need  $2 * 10 = 20$  WCU
- **Example 2:** we write 6 objects per second of 4.5 KB each
  - We need  $6 * 5 = 30$  WCU (4.5 gets rounded to the upper KB)
- **Example 3:** we write 120 objects per minute of 2 KB each
  - We need  $120 / 60 * 2 = 4$  WCU

# Strongly Consistent Read vs Eventually Consistent Read



- **Eventually Consistent Read:** If we read just after a write, it's possible we'll get unexpected response because of replication
- **Strongly Consistent Read:** If we read just after a write, we will get the correct data
- **By default:** DynamoDB uses Eventually Consistent Reads, but GetItem, Query & Scan provide a “ConsistentRead” parameter you can set to True

# DynamoDB – Read Capacity Units

- One *read capacity unit* 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 RCU are consumed
- **Example 1:** 10 strongly consistent reads per seconds of 4 KB each
  - We need  $10 * 4 \text{ KB} / 4 \text{ KB} = 10 \text{ RCU}$
- **Example 2:** 16 eventually consistent reads per seconds of 12 KB each
  - We need  $(16 / 2) * (12 / 4) = 24 \text{ RCU}$
- **Example 3:** 10 strongly consistent reads per seconds of 6 KB each
  - We need  $10 * 8 \text{ KB} / 4 = 20 \text{ RCU}$  (we have to round up 6 KB to 8 KB)

# DynamoDB – Partitions Internal

- Data is divided in partitions
- Partition keys go through a hashing algorithm to know to which partition they go to
- To compute the number of partitions:
  - By capacity:  $(\text{TOTAL RCU} / 3000) + (\text{TOTAL WCU} / 1000)$
  - By size: Total Size / 10 GB
  - Total partitions = CEILING(MAX(Capacity, Size))
- WCU and RCU are spread evenly between partitions

# DynamoDB - Throttling

- If we exceed our RCU or WCU, we get **ProvisionedThroughputExceededExceptions**
- Reasons:
  - Hot keys: one partition key is being read too many times (popular item for ex)
  - Hot partitions:
  - Very large items: remember RCU and WCU depends on size of items
- Solutions:
  - Exponential back-off when exception is encountered (already in SDK)
  - Distribute partition keys as much as possible
  - If RCU issue, we can use DynamoDB Accelerator (DAX)

# DynamoDB – Writing Data

- **PutItem** - Write data to DynamoDB (create data or full replace)
  - Consumes WCU
- **UpdateItem** – Update data in DynamoDB (partial update of attributes)
  - Possibility to use Atomic Counters and increase them
- **Conditional Writes:**
  - Accept a write / update only if conditions are respected, otherwise reject
  - Helps with concurrent access to items
  - No performance impact

# DynamoDB – Deleting Data

- **DeleteItem**
  - Delete an individual row
  - 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 – Batching Writes

- **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
- Batching allows you to save in latency by reducing the number of API calls done against DynamoDB
- Operations are done in parallel for better efficiency
- It's possible for part of a batch to fail, in which case we have to try the failed items (using exponential back-off algorithm)

# DynamoDB – Reading Data

- **GetItem:**
  - Read based on Primary key
  - Primary Key = HASH or HASH-RANGE
  - Eventually consistent read by default
  - Option to use strongly consistent reads (more RCU - might take longer)
  - **ProjectionExpression** can be specified to include only certain attributes
- **BatchGetItem:**
  - Up to 100 items
  - Up to 16 MB of data
  - Items are retrieved in parallel to minimize latency

# DynamoDB – Query

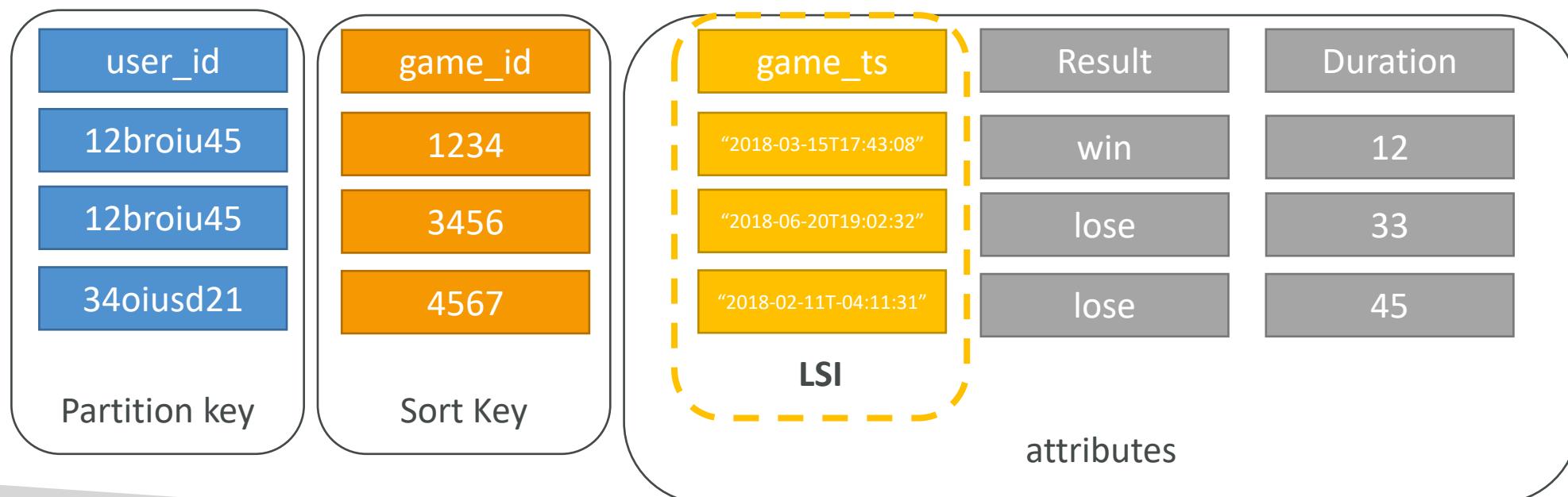
- **Query** returns items based on:
  - PartitionKey value (**must be = operator**)
  - SortKey value (=, <, <=, >, >=, Between, Begin) – optional
  - FilterExpression to further filter (client side filtering)
- Returns:
  - Up to 1 MB of data
  - Or number of items specified in **Limit**
- Able to do pagination on the results
- Can query table, a local secondary index, or a global secondary index

# DynamoDB - 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 scans**:
  - Multiple instances scan multiple partitions at the same time
  - Increases the throughput and RCU consumed
  - Limit the impact of parallel scans just like you would for Scans
- Can use a **ProjectionExpression** + **FilterExpression** (no change to RCU)

# DynamoDB – LSI (Local Secondary Index)

- Alternate range key for your table, local to the hash key
- Up to five local secondary indexes per table.
- The sort key consists of exactly one scalar attribute.
- The attribute that you choose must be a scalar String, Number, or Binary
- LSI must be defined at table creation time



# DynamoDB – GSI (Global Secondary Index)

- To speed up queries on non-key attributes, use a Global Secondary Index
- GSI = partition key + optional sort key
- The index is a new “table” and we can project attributes on it
  - The partition key and sort key of the original table are always projected (KEYS\_ONLY)
  - Can specify extra attributes to project (INCLUDE)
  - Can use all attributes from main table (ALL)
- Must define RCU / WCU for the index
- Possibility to add / modify GSI (not LSI)

# DynamoDB – GSI (Global Secondary Index)

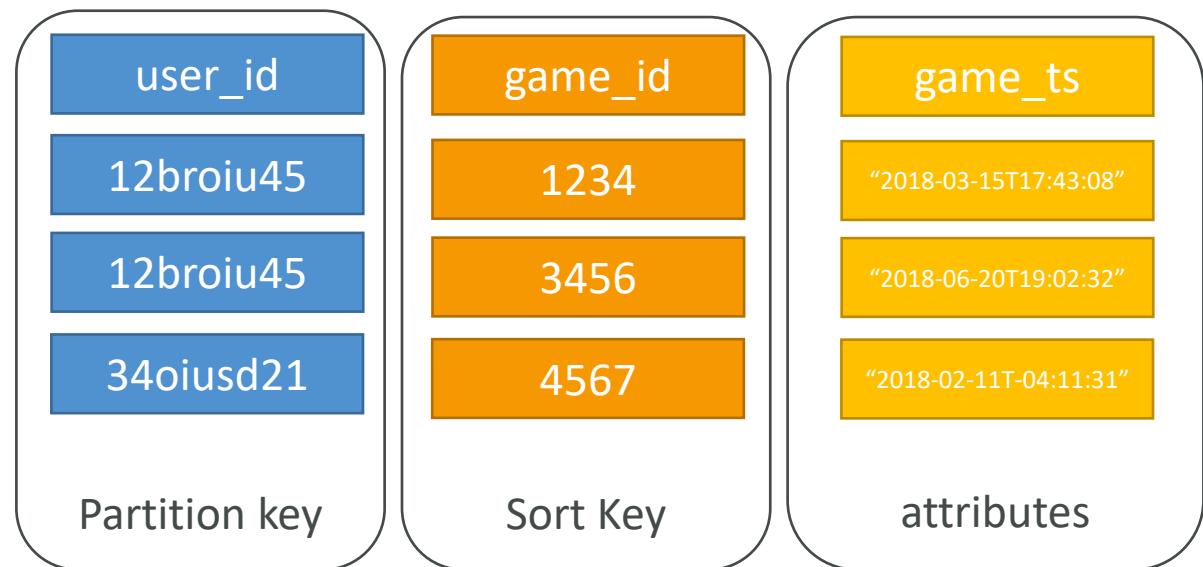
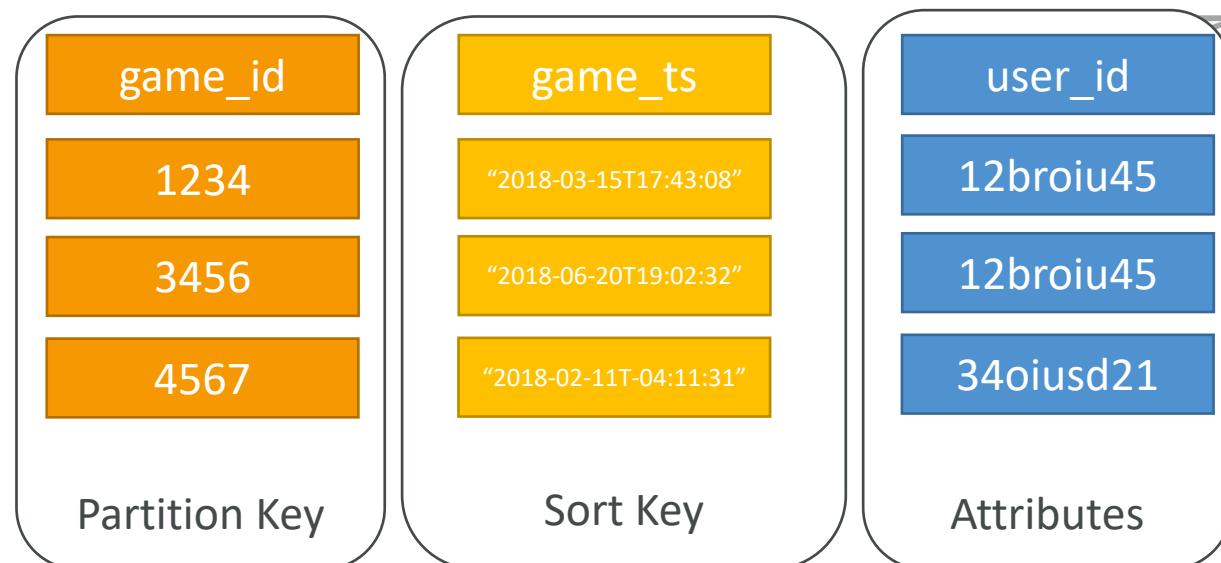


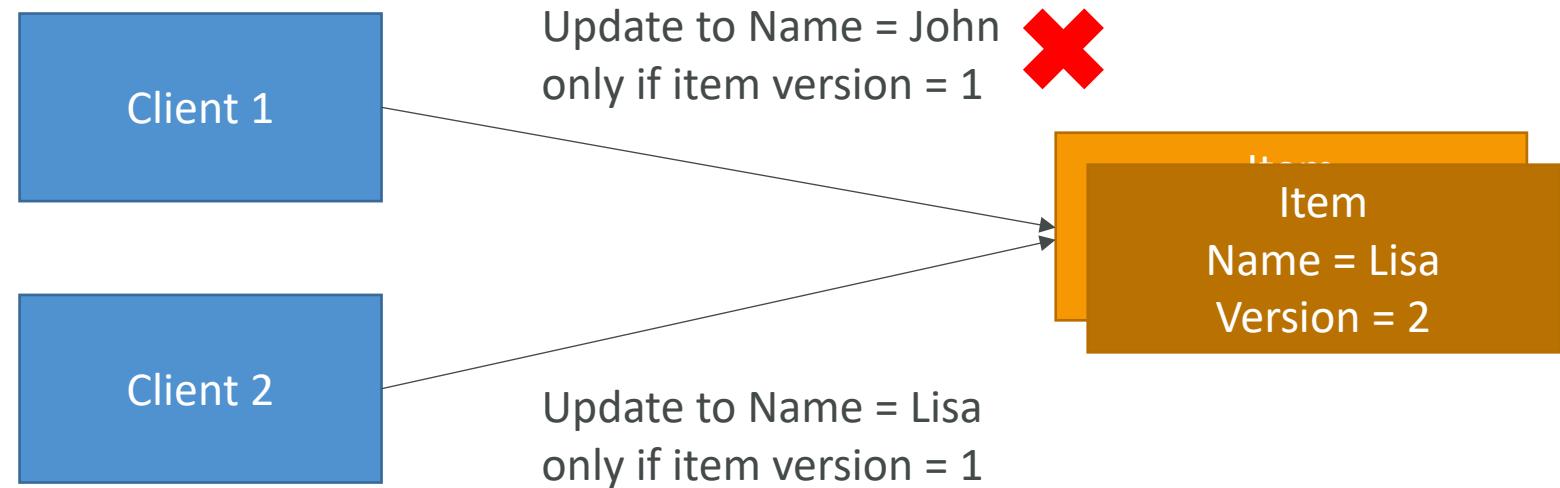
TABLE – query by user\_id

INDEX – queries by game\_id



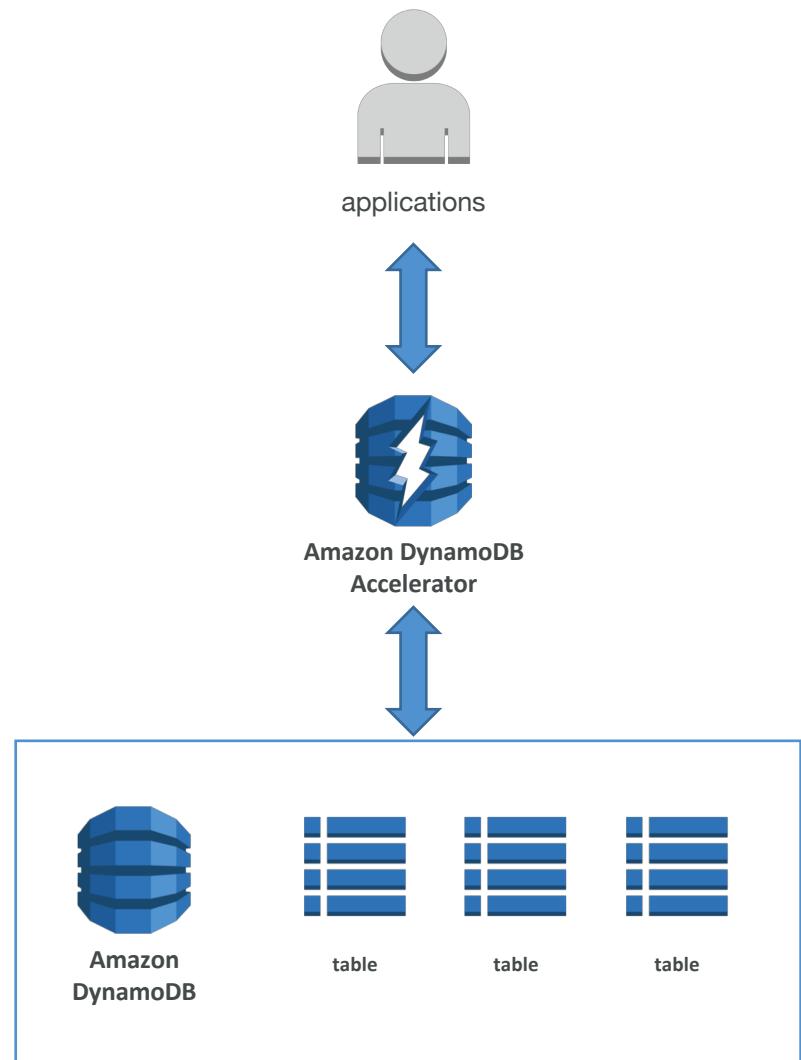
# DynamoDB Concurrency

- DynamoDB has a feature called “Conditional Update / Delete”
- That means that you can ensure an item hasn’t changed before altering it
- That makes DynamoDB an **optimistic locking / concurrency** database



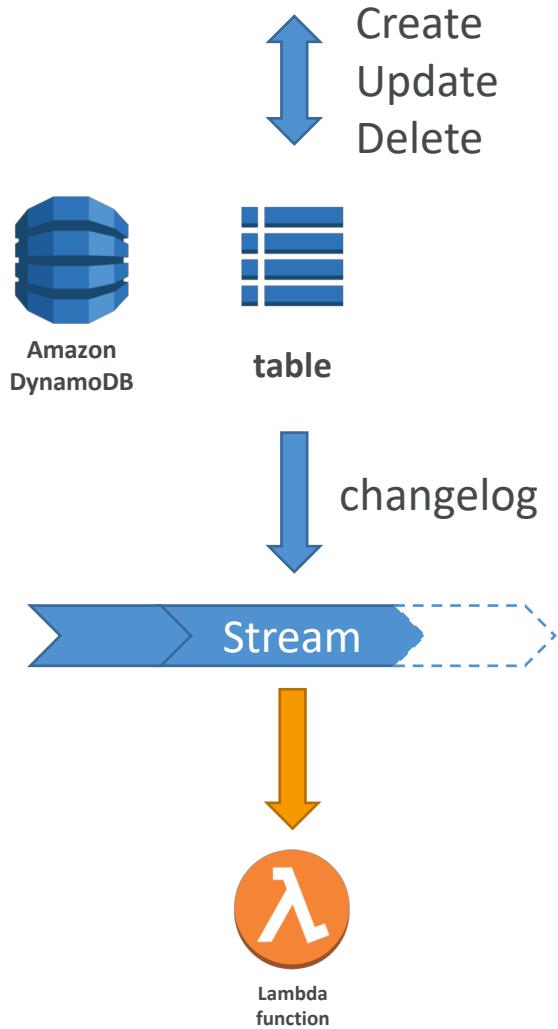
# DynamoDB - DAX

- DAX = DynamoDB Accelerator
- Seamless cache for DynamoDB, no application re-write
- Writes go through DAX to DynamoDB
- Micro second latency for cached reads & queries
- Solves the Hot Key problem (too many reads)
- 5 minutes TTL for cache by 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 Streams

- Changes in DynamoDB (Create, Update, Delete) can end up in a DynamoDB Stream
- This stream can be read by AWS Lambda, and we can then do:
  - React to changes in real time (welcome email to new users)
  - Analytics
  - Create derivative tables / views
  - Insert into ElasticSearch
- Could implement cross region replication using Streams
- Stream has 24 hours of data retention



# DynamoDB – Security & Other Features

- Security:
  - VPC Endpoints available to access DynamoDB without internet
  - Access fully controlled by IAM
  - Encryption at rest using KMS
  - Encryption in transit using SSL / TLS
- Backup and Restore feature available
  - Point in time restore like RDS
  - No performance impact
- Global Tables
  - Multi region, fully replicated, high performance
- Amazon DMS can be used to migrate to DynamoDB (from Mongo, Oracle, MySQL, S3, etc...)
- You can launch a local DynamoDB on your computer for development purposes

# API Gateway

Build, Deploy and Manage APIs

# Building a Serverless API



# AWS API Gateway



- AWS Lambda + API Gateway: No infrastructure to manage
- 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

- Outside of VPC:
  - AWS Lambda (most popular / powerful)
  - Endpoints on EC2
  - Load Balancers
  - Any AWS Service
  - External and publicly accessible HTTP endpoints
- Inside of VPC:
  - AWS Lambda in your VPC
  - EC2 endpoints in your VPC

# 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 – 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

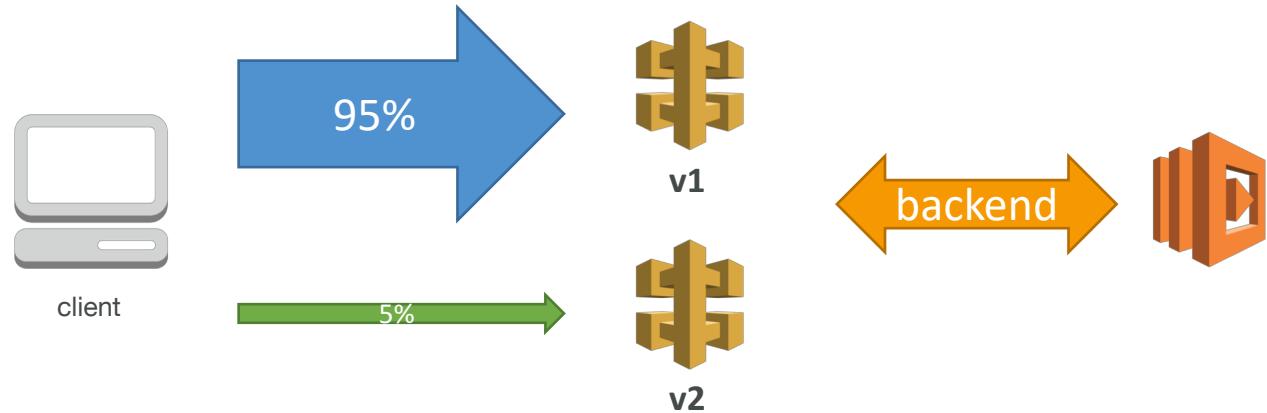
# API Gateway Stage Variables & Lambda Aliases

- 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

# Mapping Templates

- Mapping templates can be used to modify request / responses
- Rename parameters
- Modify body content
- Add headers
- Map JSON to XML for sending to backend or back to client
- Uses Velocity Template Language (VTL): for loop, if etc...
- Filter output results (remove unnecessary data)

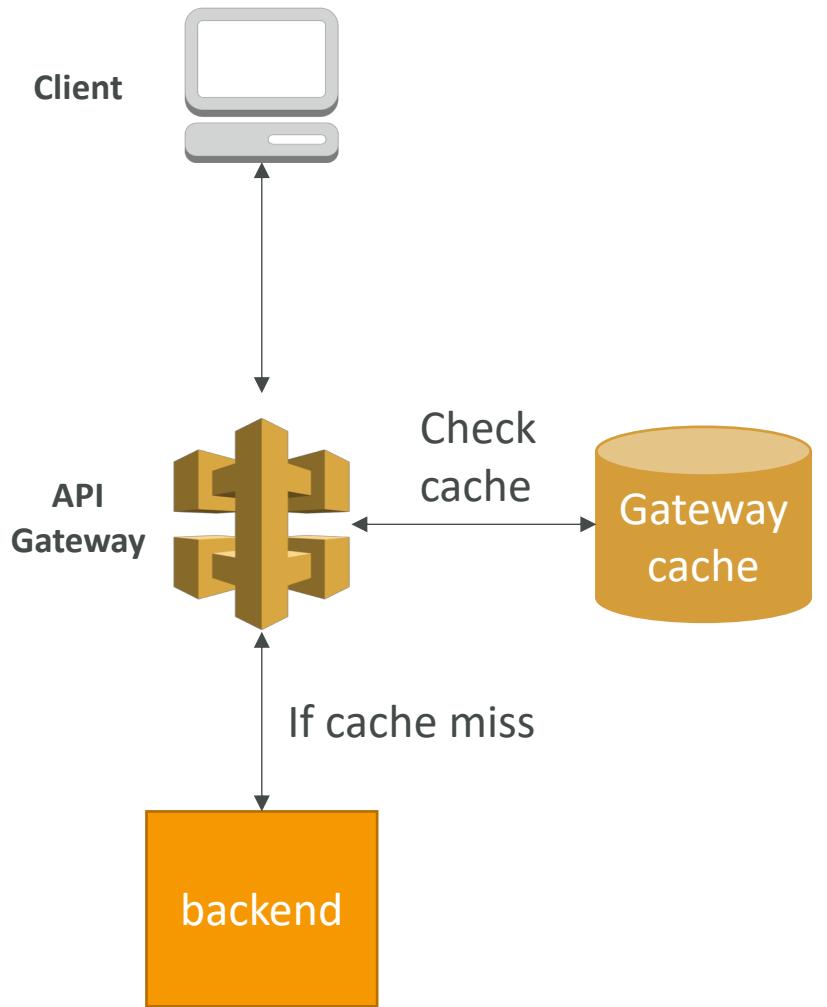
# AWS API Gateway Swagger / Open API spec

- Common way of defining REST APIs, using API definition as code
- Import existing Swagger / 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 Swagger / OpenAPI spec
- Swagger can be written in YAML or JSON
- Using Swagger we can generate SDK for our applications



# Caching API responses

- Caching reduces the number of calls made to the backend
- Default time TTL (time to live) is 300 seconds (min: 0s, max: 3600s)
- Caches are defined per stage
- Cache encryption option
- Cache capacity between 0.5GB to 237GB
- Possible to override cache settings for specific methods
- 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)



# API Gateway – Logging, Monitoring, Tracing

- **CloudWatch Logs:**
  - Enable CloudWatch logging at the Stage level (with Log Level)
  - Can override settings on a per API basis (ex: ERROR, DEBUG, INFO)
  - Log contains information about request / response body
- **CloudWatch Metrics:**
  - Metrics are by stage
  - Possibility to enable detailed metrics
- **X-Ray:**
  - Enable tracing to get extra information about requests in API Gateway
  - X-Ray API Gateway + AWS Lambda gives you the full picture

# AWS API Gateway - CORS

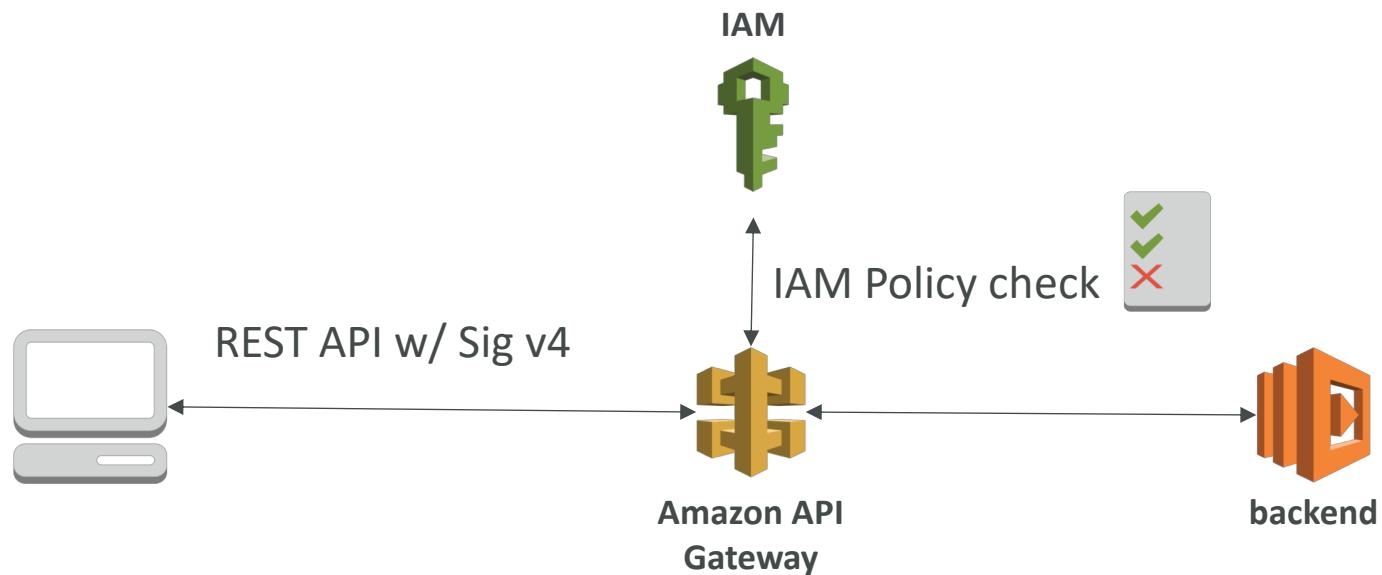
- CORS must be enabled when you receive API calls from another domain.
- The OPTIONS pre-flight request must contain the following headers:
  - Access-Control-Allow-Methods
  - Access-Control-Allow-Headers
  - Access-Control-Allow-Origin
- CORS can be enabled through the console

# API Gateway – Usage plans & API Keys

- What if you want to limit your customers usage of your API?
- **Usage plans:**
  - Throttling: set overall capacity and burst capacity
  - Quotas: number of requests made per day / week / month
  - Associate with desired API Stages
- **API Keys:**
  - Generate one per customer
  - Associate with usage plans
- Ability to track usage for API Keys

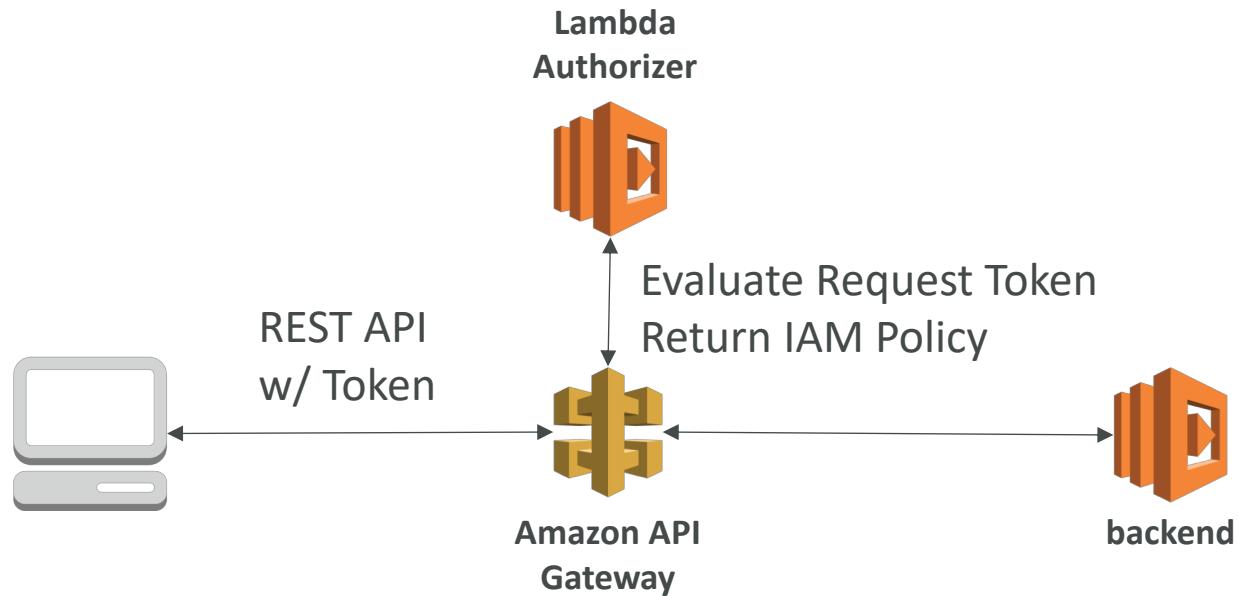
# API Gateway – Security IAM Permissions

- Create an IAM policy authorization and attach to User / Role
- API Gateway verifies IAM permissions passed by the calling application
- Good to provide access within your own infrastructure
- Leverages “Sig v4” capability where IAM credential are in headers



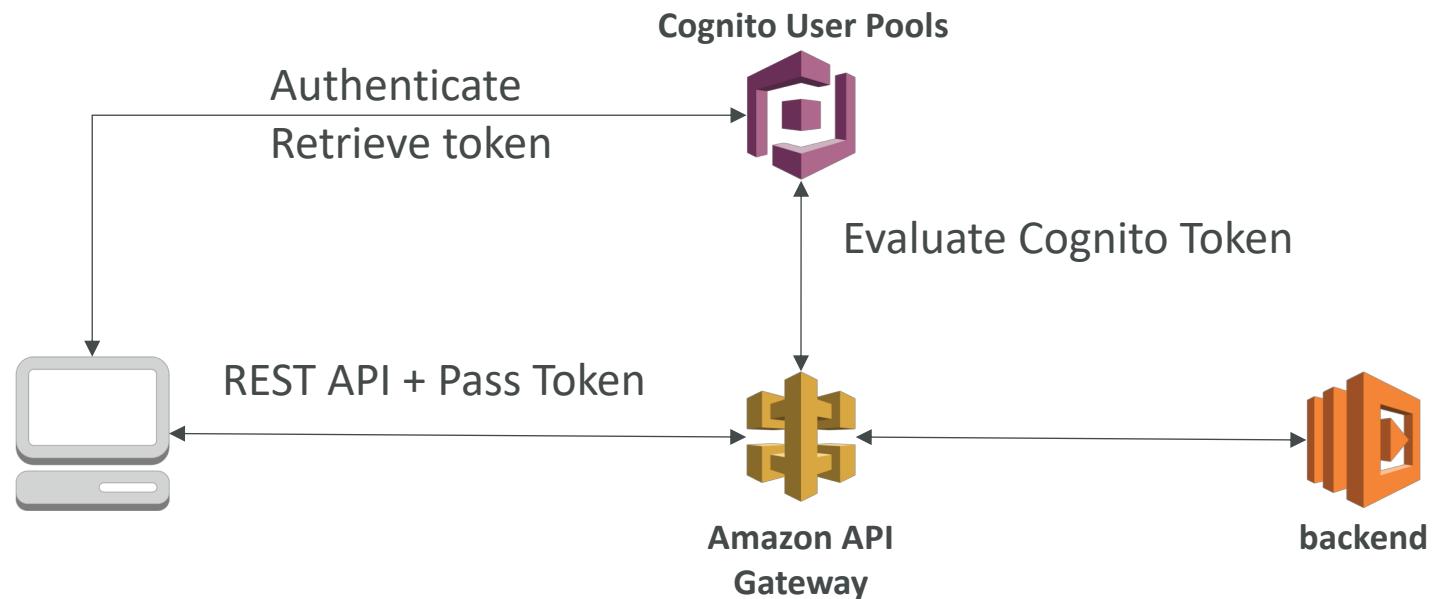
# API Gateway – Security Lambda Authorizer (formerly Custom Authorizers)

- Uses AWS Lambda to validate the token in header being passed
- Option to cache result of authentication
- Helps to use OAuth / SAML / 3<sup>rd</sup> party type of authentication
- Lambda must return an IAM policy for the user



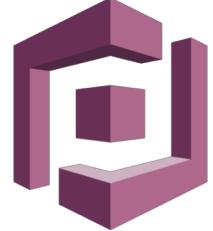
# API Gateway – Security Cognito User Pools

- Cognito fully manages user lifecycle
- API gateway verifies identity automatically from AWS Cognito
- No custom implementation required
- Cognito only helps with authentication, not authorization



# API Gateway – Security – Summary

- **IAM:**
  - Great for users / roles already within your AWS account
  - Handle authentication + authorization
  - Leverages Sig v4
- **Custom Authorizer:**
  - Great for 3<sup>rd</sup> party tokens
  - Very flexible in terms of what IAM policy is returned
  - Handle Authentication + Authorization
  - Pay per Lambda invocation
- **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

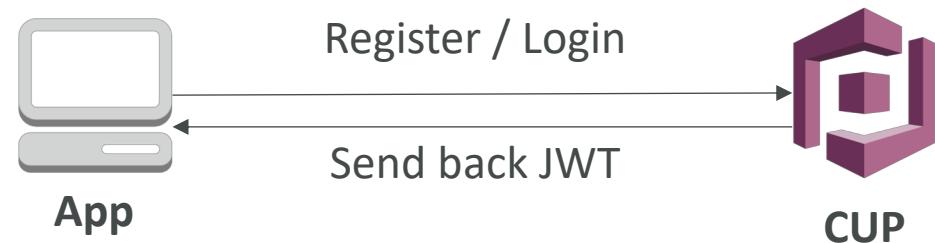


# AWS Cognito

- We want to give our users an identity so that they can interact with our application.
- **Cognito User Pools:**
  - Sign in functionality for app users
  - Integrate with API Gateway
- **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 Sync:**
  - Synchronize data from device to Cognito.
  - May be deprecated and replaced by AppSync

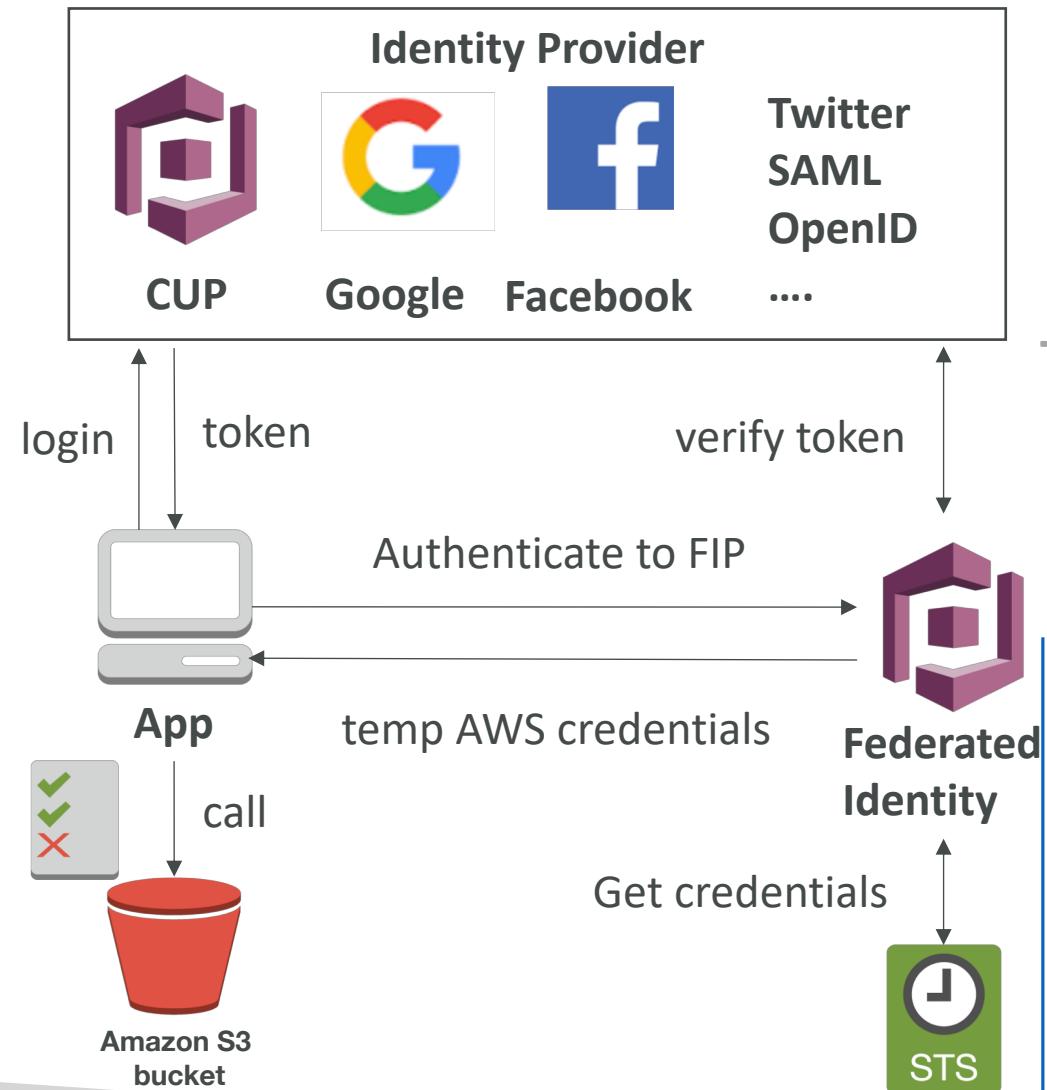
# AWS Cognito User Pools (CUP)

- Create a serverless database of user for your mobile apps
- Simple login: Username (or email) / password combination
- Possibility to verify emails / phone numbers and add MFA
- Can enable Federated Identities (Facebook, Google, SAML...)
- Sends back a JSON Web Tokens (JWT)
- Can be integrated with API Gateway for authentication



# AWS Cognito – Federated Identity Pools

- Goal:
  - Provide direct access to AWS Resources from the Client Side
- How:
  - Log in to federated identity provider – or remain anonymous
  - Get temporary AWS credentials back from the Federated Identity Pool
  - These credentials come with a pre-defined IAM policy stating their permissions
- Example:
  - provide (temporary) access to write to S3 bucket using Facebook Login



# AWS Cognito Sync

- Deprecated – use AWS AppSync now
- Store preferences, configuration, state of app
- Cross device synchronization (any platform – iOS, Android, etc...)
- Offline capability (synchronization when back online)
- Requires Federated Identity Pool in Cognito (not User Pool)
- Store data in datasets (up to 1 MB)
- Up to 20 datasets to synchronise

# 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...
- Only two commands to deploy to AWS
- 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:
  - aws cloudformation package / sam package
  - aws cloudformation deploy / sam deploy

# Deep dive into SAM deployment

aws cloudformation package



SAM Template  
YAML file

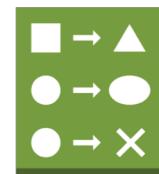
Transform

aws cloudformation deploy



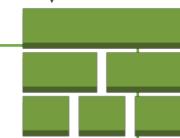
Generated Template  
CloudFormation YAML

Create and execute change set



AWS  
CloudFormation

change set



stack



Application Code  
+ Swagger File (optional)

Zip and upload

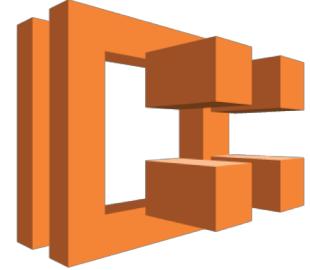


Code S3  
bucket

# AWS ECS - Essentials

Docker containers in AWS

# ECS Introduction



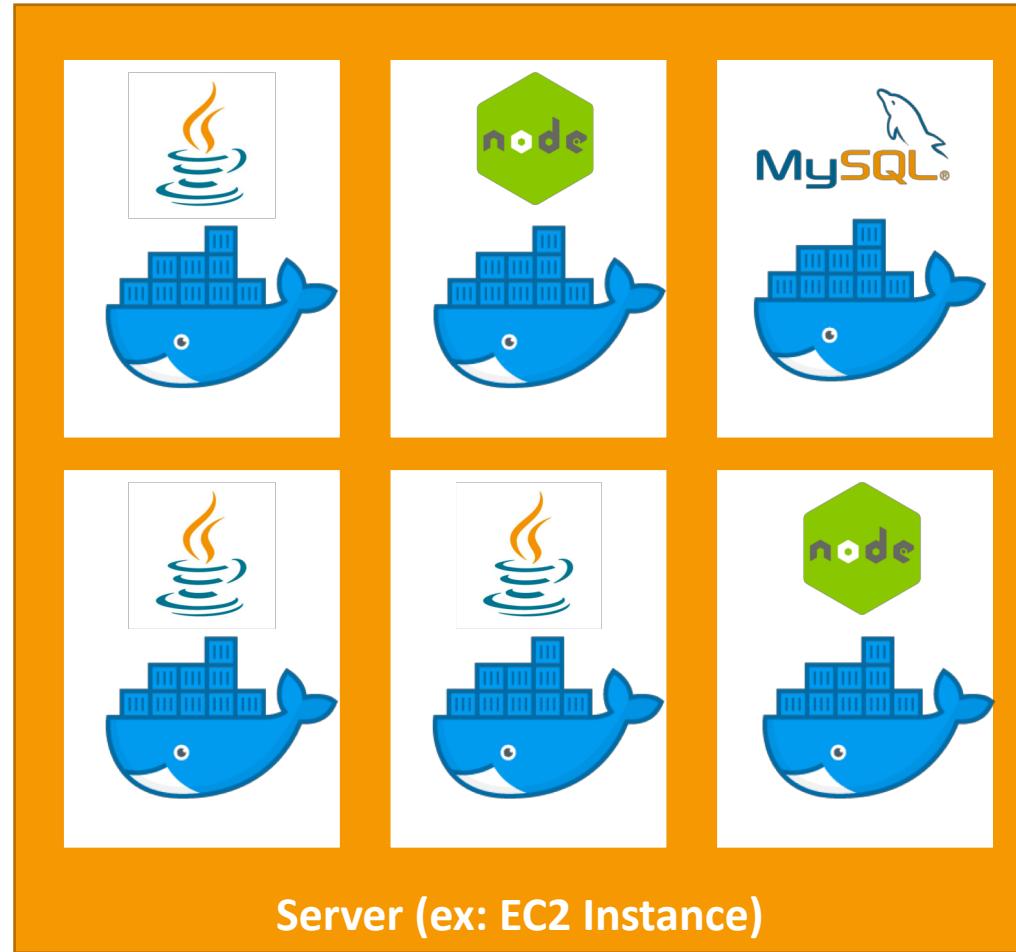
- New section at the exam – some tricky questions
- Docker Introduction
- ECS
  - Cluster
  - Services
  - Tasks
  - Tasks Definition
- ECR
- Fargate
- Exam Tips

# 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

# Docker on an OS

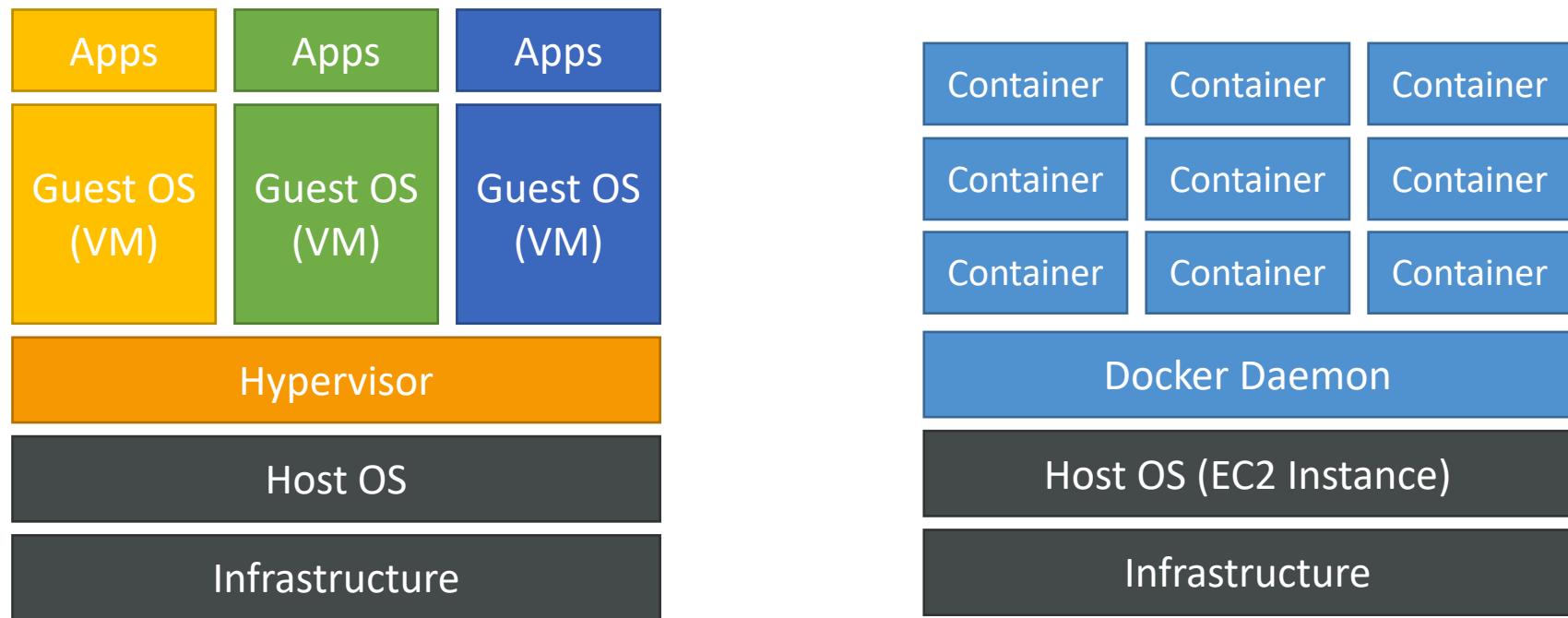


# Where Docker images are stored?

- Docker images are stored in Docker Repositories
- Public: Docker Hub <https://hub.docker.com/>
  - Find base images for many technologies or OS:
  - Ubuntu
  - MySQL
  - NodeJS, Java...
- Private: Amazon ECR (Elastic Container Registry)

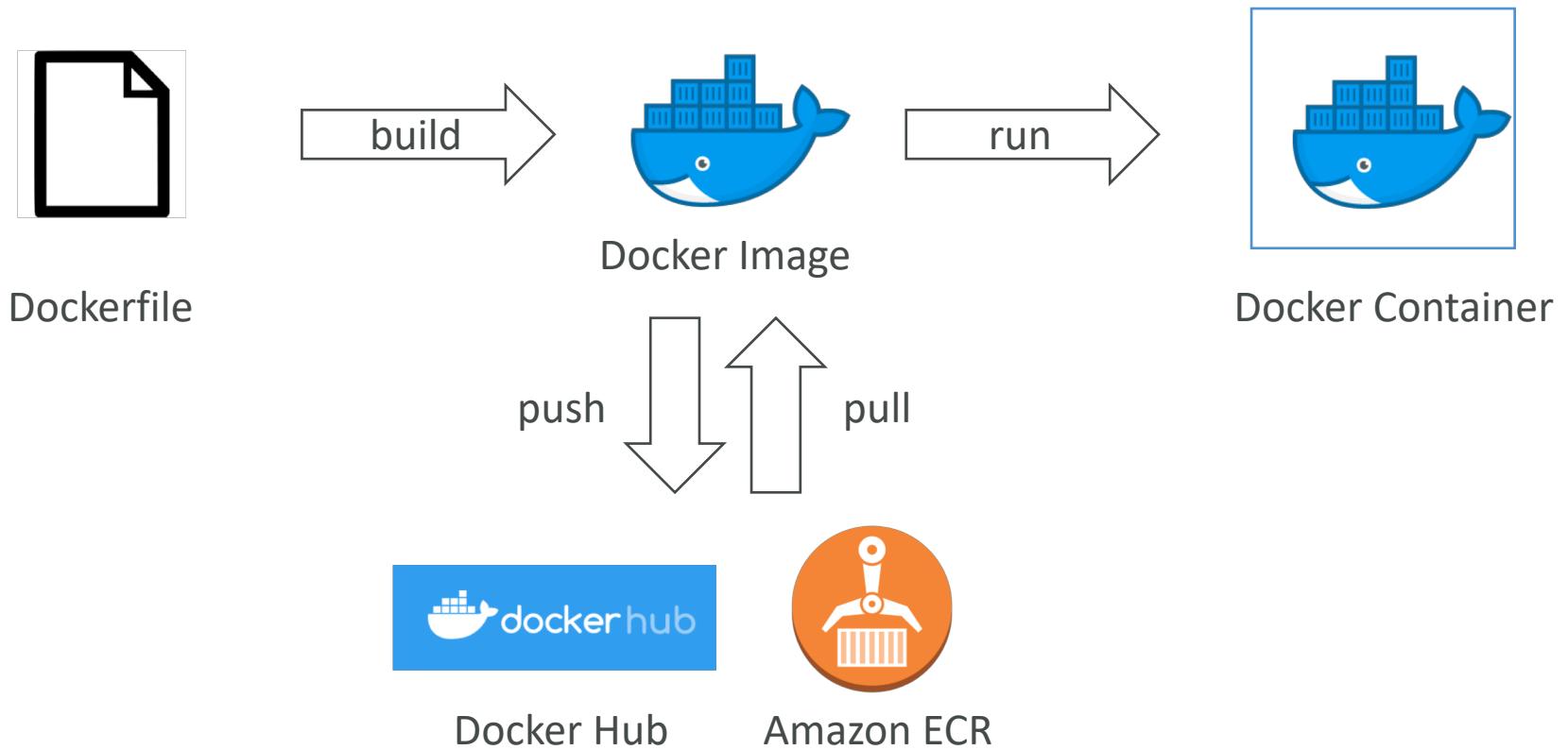
# Docker versus Virtual Machines

- 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

- Download Docker at: <https://www.docker.com/get-started>

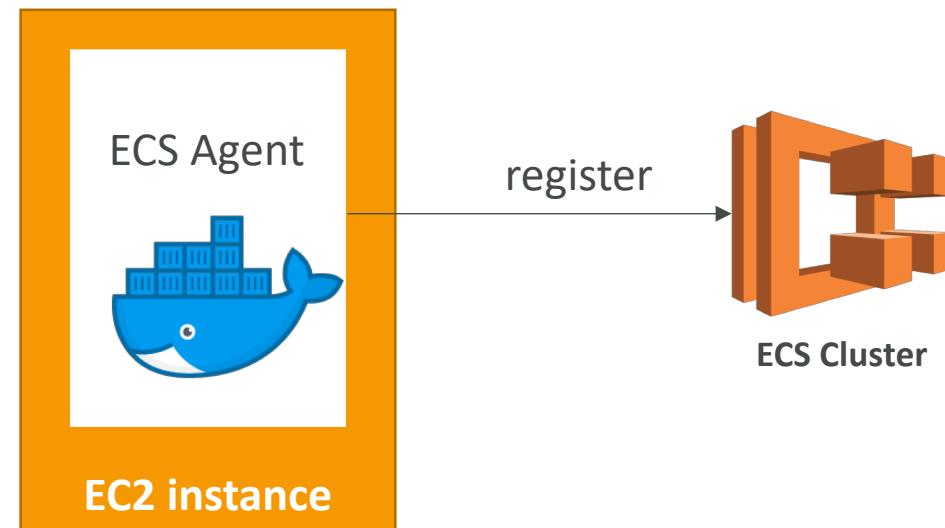


# Docker Containers Management

- To manage containers, we need a container management platform
- Three choices:
- ECS: Amazon's own platform
- Fargate: Amazon's own Serverless platform
- EKS: Amazon's managed Kubernetes (open source)

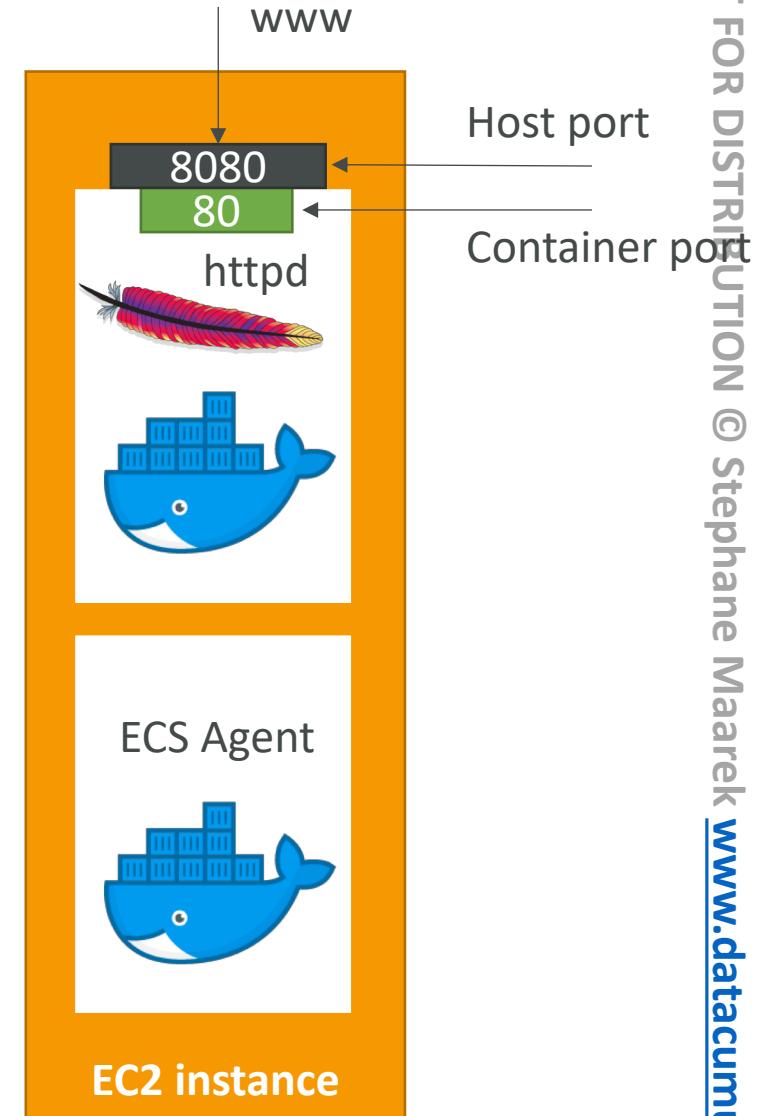
# ECS Clusters Overview

- ECS Clusters are logical grouping of EC2 instances
- EC2 instances run the ECS agent (Docker container)
- The ECS agents registers the instance to the ECS cluster
- The EC2 instances run a special AMI, made specifically for ECS



# ECS Task Definitions

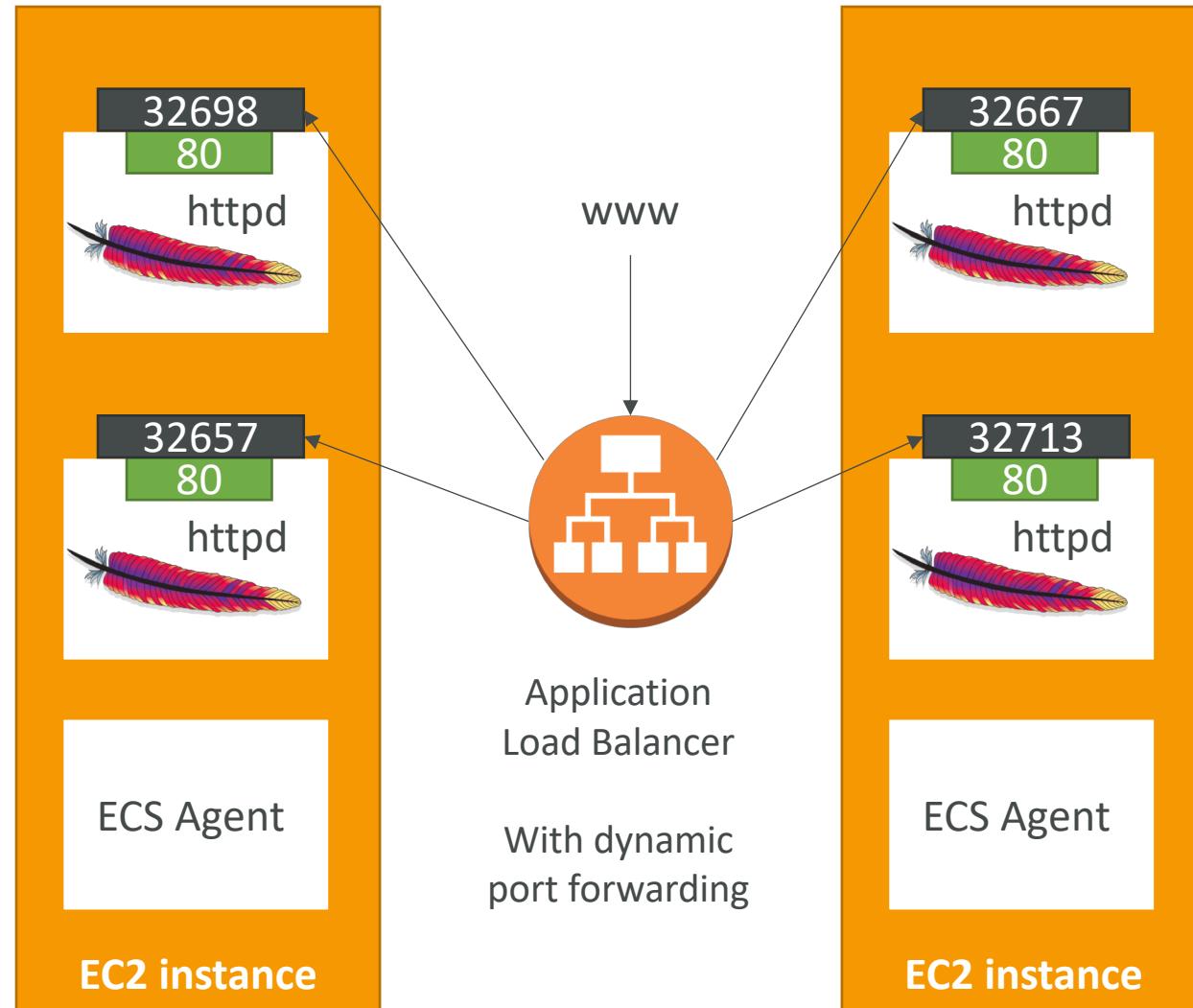
- Tasks definitions are metadata in **JSON form** to tell ECS how to run a Docker Container
- It contains crucial information around:
  - Image Name
  - Port Binding for Container and Host
  - Memory and CPU required
  - Environment variables
  - Networking information
  - IAM Role
  - Logging configuration (ex CloudWatch)



# ECS Service

- ECS Services help define how many tasks should run and how they should be run
- They ensure that the number of tasks desired is running across our fleet of EC2 instances.
- They can be linked to ELB / NLB / ALB if needed
- Let's make our first service!

# ECS Service with Load Balancer





# ECR

- So far we've been using Docker images from Docker Hub (public)
- ECR is a private Docker image repository
- Access is controlled through IAM (permission errors => policy)
- You need to run some commands to push pull:
  - \$(aws ecr get-login --no-include-email --region eu-west-1 )
  - docker push 1234567890.dkr.ecr.eu-west-1.amazonaws.com/demo:latest
  - docker pull 1234567890.dkr.ecr.eu-west-1.amazonaws.com/demo:latest

# Fargate

- When launching an ECS Cluster, we have to create our EC2 instances
  - If we need to scale, we need to add EC2 instances
  - So we manage infrastructure...
- 
- With Fargate, it's all Serverless!
  - We don't provision EC2 instances
  - We just create task definitions, and AWS will run our containers for us
  - To scale, just increase the task number. Simple! No more EC2 ☺

# ECS Summary + Exam Tips

- ECS is used to run Docker containers and has 3 flavors:
- ECS “Classic”: provision EC2 instances to run containers onto
- Fargate: ECS Serverless, no more EC2 to provision
- EKS: Managed Kubernetes by AWS

# ECS Classic

- EC2 instances must be created
- We must configure the file `/etc/ecs/ecs.config` with the cluster name
- The EC2 instance must run an ECS agent
- EC2 instances can run multiple containers on the same type:
  - You must not specify a host port (only container port)
  - You should use an Application Load Balancer with the dynamic port mapping
  - The EC2 instance security group must allow traffic from the ALB on all ports
- ECS tasks can have IAM Roles to execute actions against AWS
- Security groups operate at the instance level, not task level

# ECR is used to store Docker Images

- ECR is tightly integrated with IAM
- To push:
  - \$(aws ecr get-login --no-include-email --region eu-west-1)
  - docker push 1234567890.dkr.ecr.eu-west-1.amazonaws.com/demo:latest
- To pull:
  - \$(aws ecr get-login --no-include-email --region eu-west-1)
  - docker pull 1234567890.dkr.ecr.eu-west-1.amazonaws.com/demo:latest
- “aws ecr get-login” generates a “docker login” command
- In case an EC2 instance (or you) cannot pull a Docker image, check IAM

# Fargate

- Fargate is Serverless (no EC2 to manage)
- AWS provisions containers for us and assigns them ENI
- Fargate containers are provisioned by the container spec (CPU / RAM)
  
- Fargate tasks can have IAM Roles to execute actions against AWS

# ECS Integrations

- ECS does integrate with X-Ray
  - To make it work, X-Ray must be running as a 2<sup>nd</sup> container within the task definition
  - You can use the ready-to-use AWS image: <https://hub.docker.com/r/amazon/aws-xray-daemon/>
  - More information here: <https://docs.aws.amazon.com/xray/latest/devguide/xray-daemon-ecs.html>
- ECS does integrate with CloudWatch Logs:
  - You need to setup logging at the task definition level
  - Each container will have a different log stream

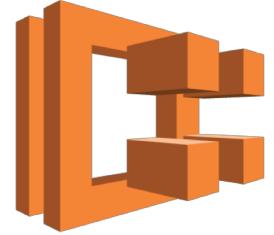
# CLI to know

- Obtain a “docker login” command against ECR:
  - aws ecr get-login
- Pull:
  - docker pull 1234567890.dkr.ecr.eu-west-1.amazonaws.com/demo:latest
- Push:
  - docker push 1234567890.dkr.ecr.eu-west-1.amazonaws.com/demo:latest
- Create a service on ECS:
  - aws ecs create-service
- Build a docker image:
  - docker build -t demo .

# Previous ECS Slides below

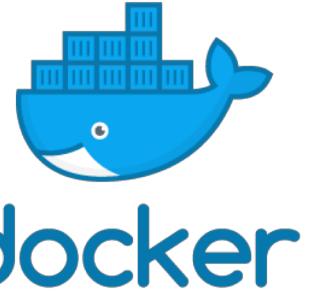
- Will keep them for reference, have a look too!

# AWS ECS – Elastic Container Service



- ECS is a container orchestration service
- ECS helps you run Docker containers on EC2 machines
- ECS is complicated, and made of:
  - “ECS Core”: Running ECS on user-provisioned EC2 instances
  - Fargate: Running ECS tasks on AWS-provisioned compute (serverless)
  - EKS: Running ECS on AWS-powered Kubernetes (running on EC2)
  - ECR: Docker Container Registry hosted by AWS
- ECS & Docker are very popular for microservices
- For now, for the exam, only “ECS Core” & ECR is in scope
- IAM security and roles at the ECS task level

# What's Docker?



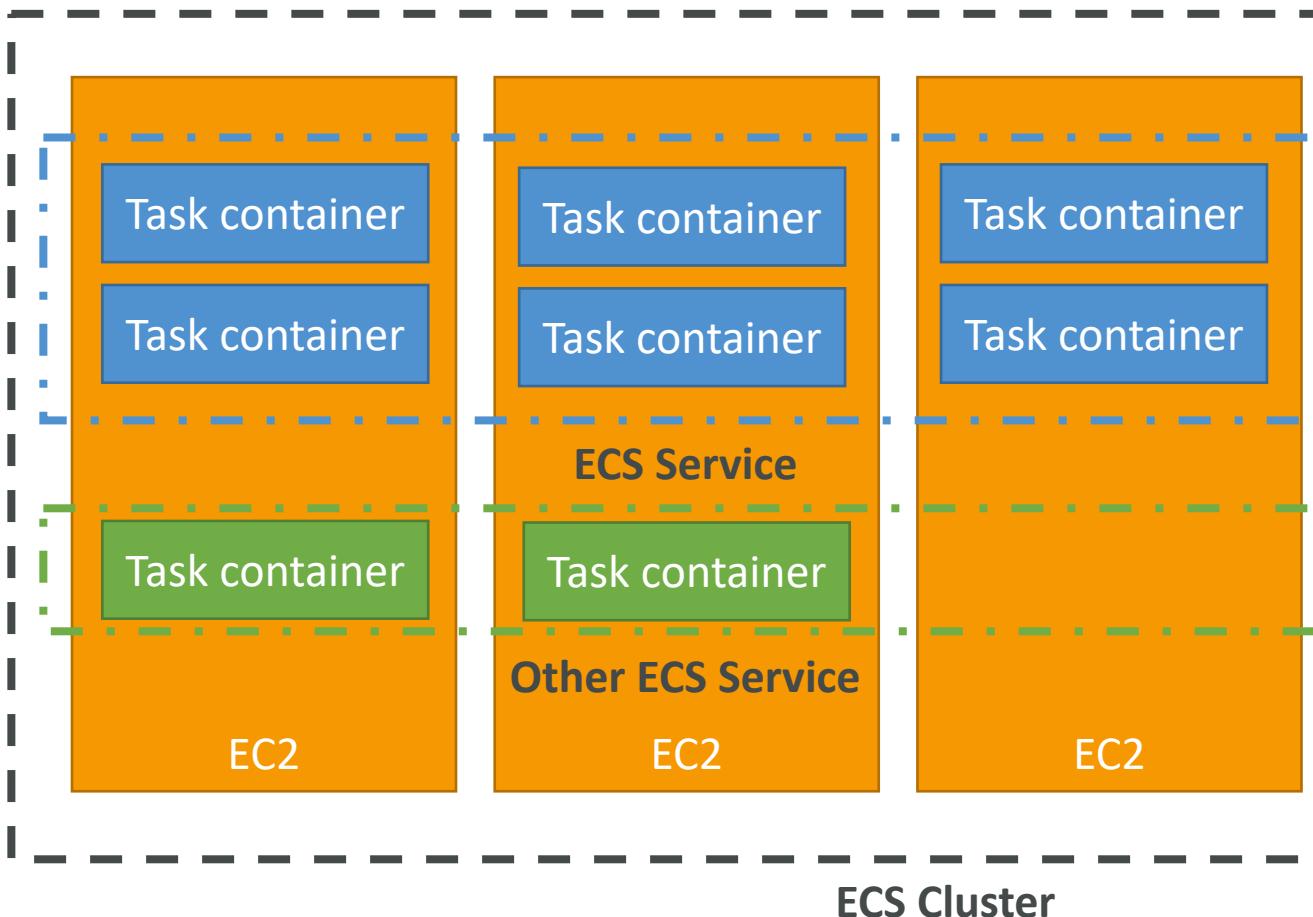
- Docker is a “container technology”
- Run a containerized application on any machine with Docker installed
- Containers allows our application to work the same way anywhere
- Containers are isolated from each other
- Control how much memory / CPU is allocated to your container
- Ability to restrict network rules
- More efficient than Virtual machines
- Scale containers up and down very quickly (seconds)

# AWS ECS – Use cases

- Run microservices
  - Ability to run multiple docker containers on the same machine
  - Easy service discovery features to enhance communication
  - Direct integration with Application Load Balancers
  - Auto scaling capability
- Run batch processing / scheduled tasks
  - Schedule ECS containers to run on On-demand / Reserved / Spot instances
- Migrate applications to the cloud
  - Dockerize legacy applications running on premise
  - Move Docker containers to run on ECS

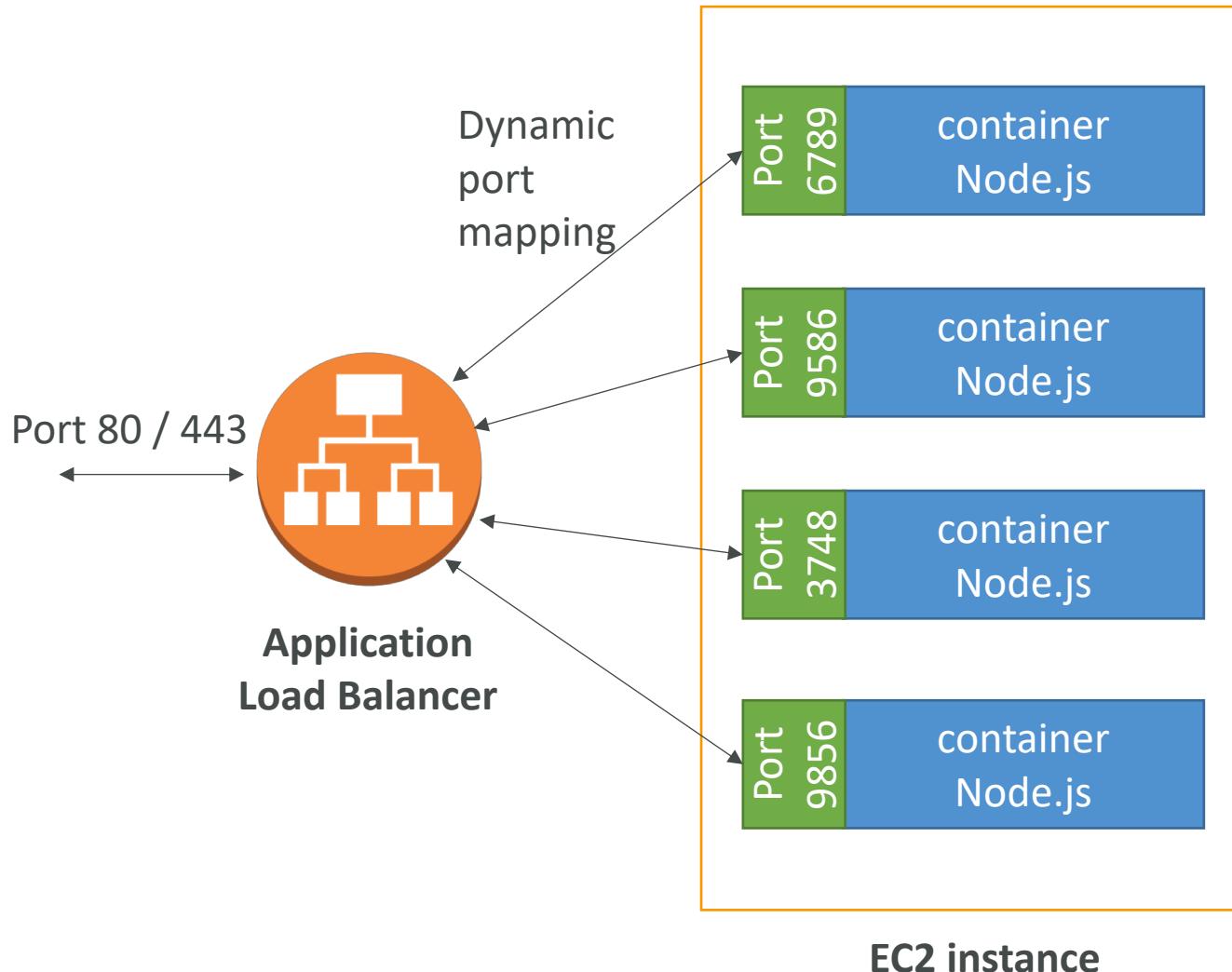
# AWS ECS – Concepts

- **ECS cluster:** set of EC2 instances
- **ECS service:** applications definitions running on ECS cluster
- **ECS tasks + definition:** containers running to create the application
- **ECS IAM roles:** roles assigned to tasks to interact with AWS



# AWS ECS – ALB integration

- Application Load Balancer (ALB) has a direct integration feature with ECS called “port mapping”
- This allows you to run multiple instances of the same application on the same EC2 machine
- Use cases:
  - Increased resiliency even if running on one EC2 instance
  - Maximize utilization of CPU / cores
  - Ability to perform rolling upgrades without impacting application uptime



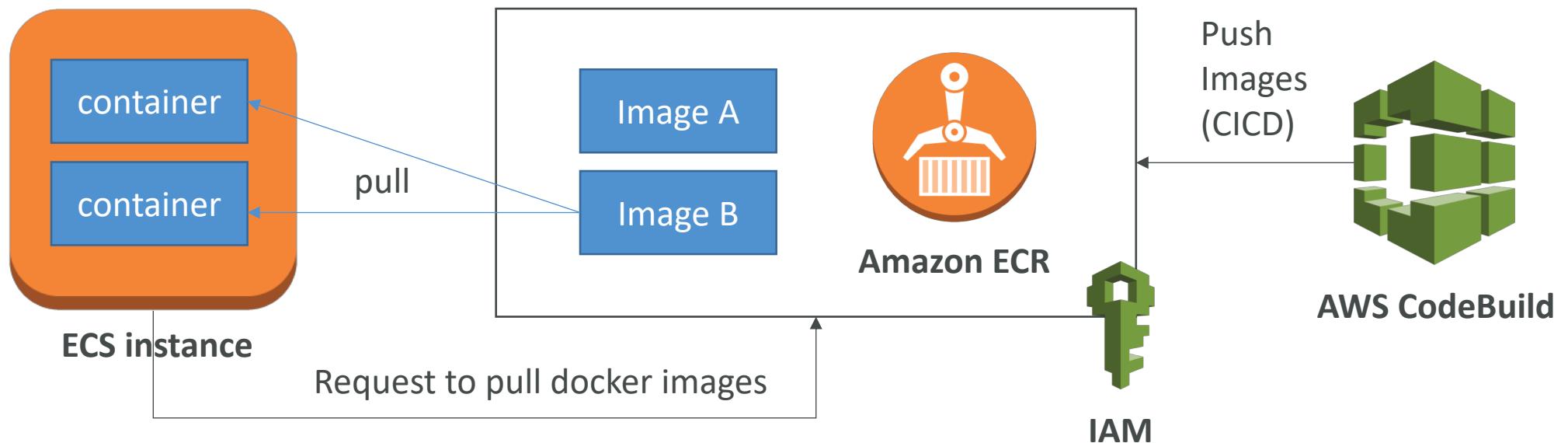
# AWS ECS – ECS Setup & Config file

- Run an EC2 instance, install the ECS agent with ECS config file
- Or use an ECS-ready Linux AMI (still need to modify config file)
- ECS Config file is at **/etc/ecs/ecs.config**

```
1  ECS_CLUSTER=MyCluster      #Assign EC2 instance to an ECS cluster
2  ECS_ENGINE_AUTH_DATA={...}  #to pull images from private registries
3  ECS_AVAILABLE_LOGGING_DRIVERS=[...]  #CloudWatch container logging
4  ECS_ENABLE_TASK_IAM_ROLE=true      #Enable IAM roles for ECS tasks
```

# AWS ECR – Elastic Container Registry

- Store, managed and deploy your containers on AWS
- Fully integrated with IAM & ECS
- Sent over HTTPS (encryption in flight) and encrypted at rest



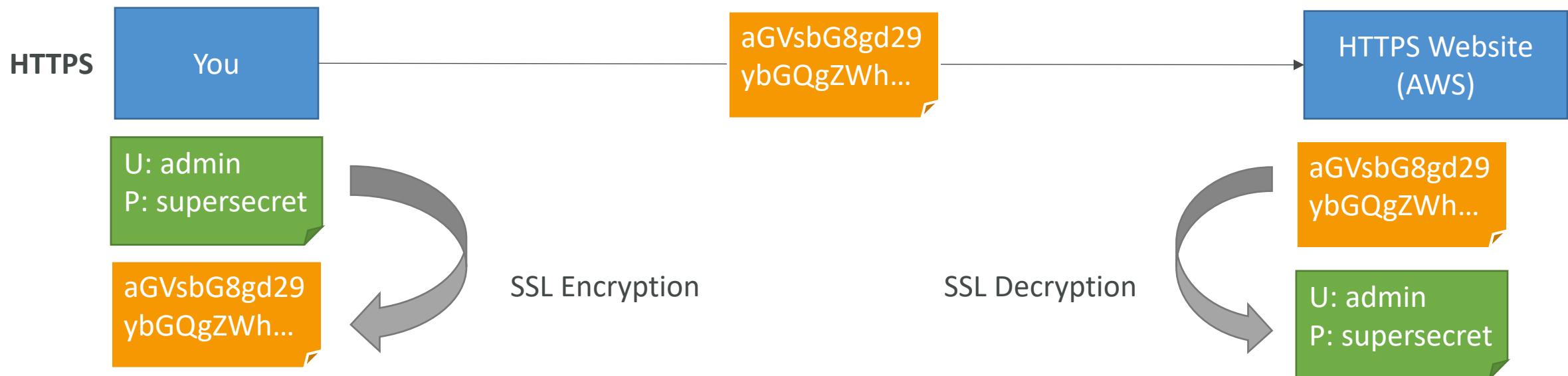
# AWS Security & Encryption

KMS, Encryption SDK, SSM Parameter Store

# Why encryption?

## Encryption in flight (SSL)

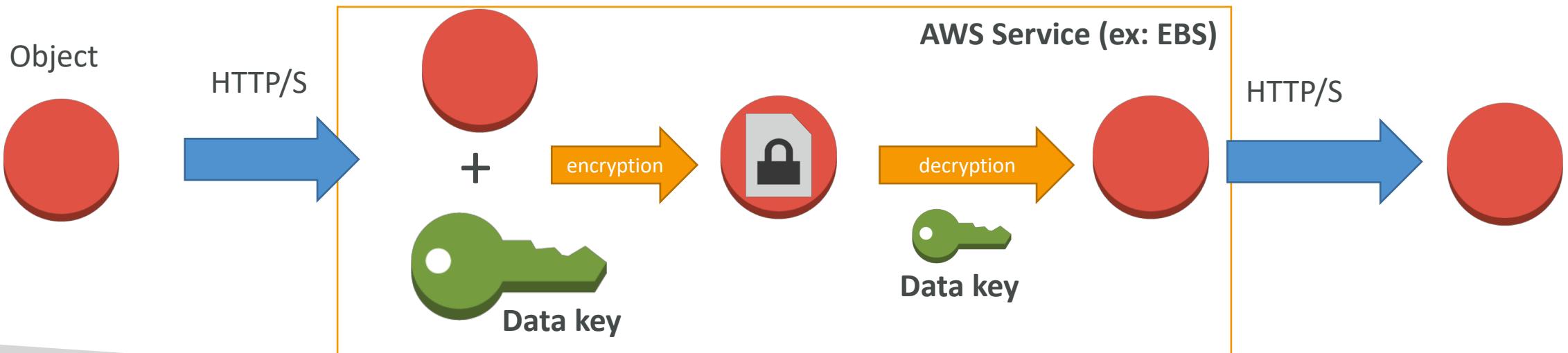
- Data is encrypted before sending and decrypted after receiving
- SSL certificates help with encryption (HTTPS)
- Encryption in flight ensures no MITM (man in the middle attack) can happen



# Why encryption?

## Server side encryption at rest

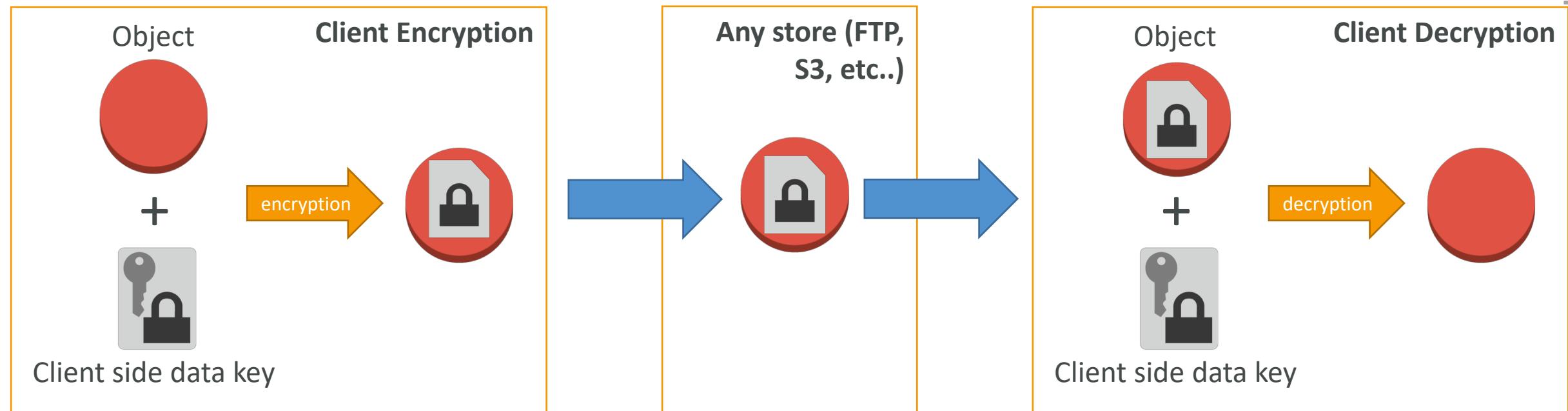
- Data is encrypted after being received by the server
- Data is decrypted before being sent
- It is stored in an encrypted form thanks to a key (usually a data key)
- The encryption / decryption keys must be managed somewhere and the server must have access to it

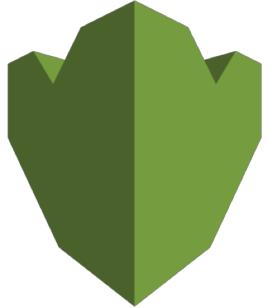


# Why encryption?

## 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
- Easy way to control access to your data, AWS manages keys for us
- Fully integrated with IAM for authorization
- Seamlessly integrated into:
  - Amazon EBS: encrypt volumes
  - Amazon S3: Server side encryption of objects
  - Amazon Redshift: encryption of data
  - Amazon RDS: encryption of data
  - Amazon SSM: Parameter store
  - Etc...
- But you can also use the CLI / SDK

# AWS KMS 101

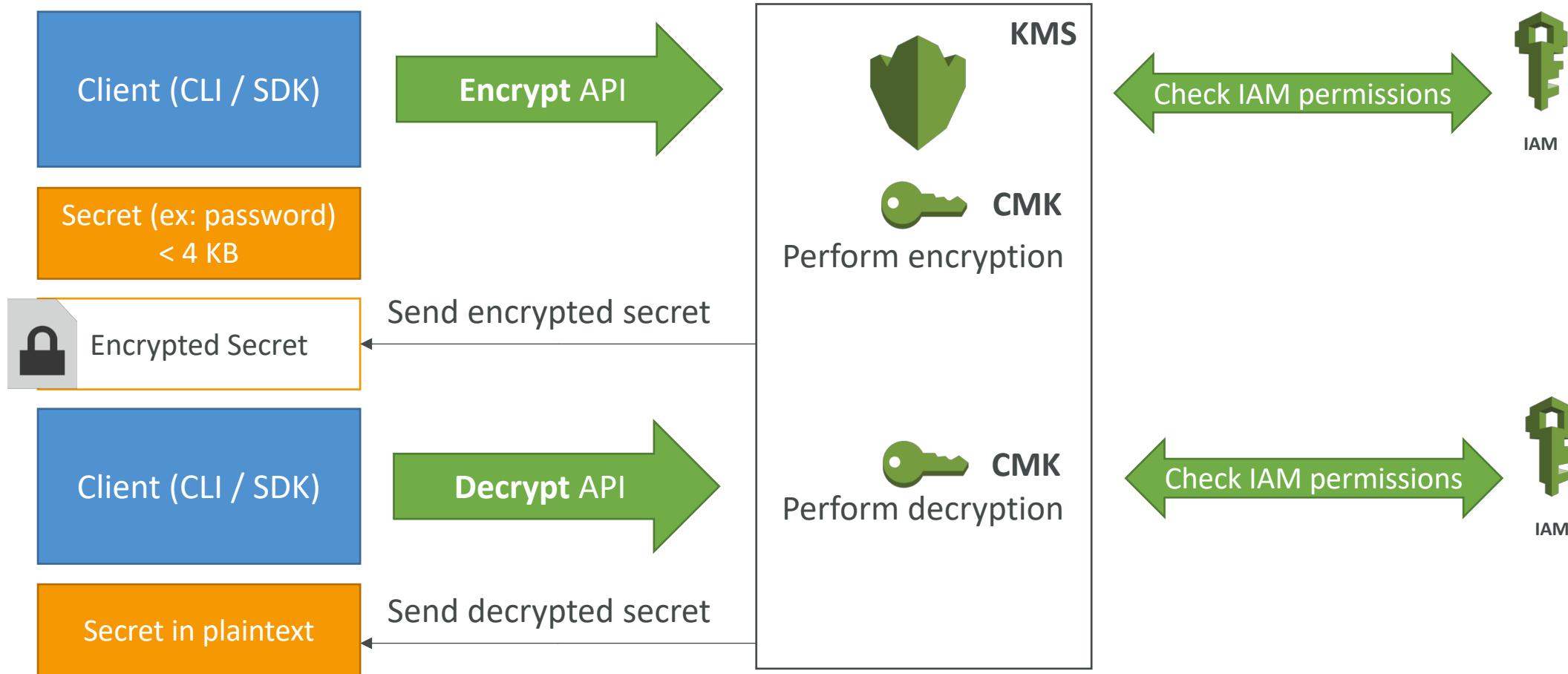
- Anytime you need to share sensitive information... use KMS
  - Database passwords
  - Credentials to external service
  - Private Key of SSL certificates
- The value in KMS is that the CMK used to encrypt data can never be retrieved by the user; and the CMK can be rotated for extra security
- **Never ever store your secrets in plaintext, especially in your code!**
- Encrypted secrets can be stored in the code / environment variables
- **KMS can only help in encrypting up to 4KB of data per call**
- If data > 4 KB, use envelope encryption
- To give access to KMS to someone:
  - Make sure the Key Policy allows the user
  - Make sure the IAM Policy allows the API calls

# AWS KMS (Key Management Service)

- Able to fully manage the keys & policies:
  - Create
  - Rotation policies
  - Disable
  - Enable
- Able to audit key usage (using CloudTrail)
- Three types of Customer Master Keys (CMK):
  - AWS Managed Service Default CMK: **free**
  - User Keys created in KMS: **\$1 / month**
  - User Keys imported (must be 256-bit symmetric key): **\$1 / month**
- + pay for API call to KMS (**\$0.03 / 10000 calls**)

# How does KMS work?

## API – Encrypt and Decrypt

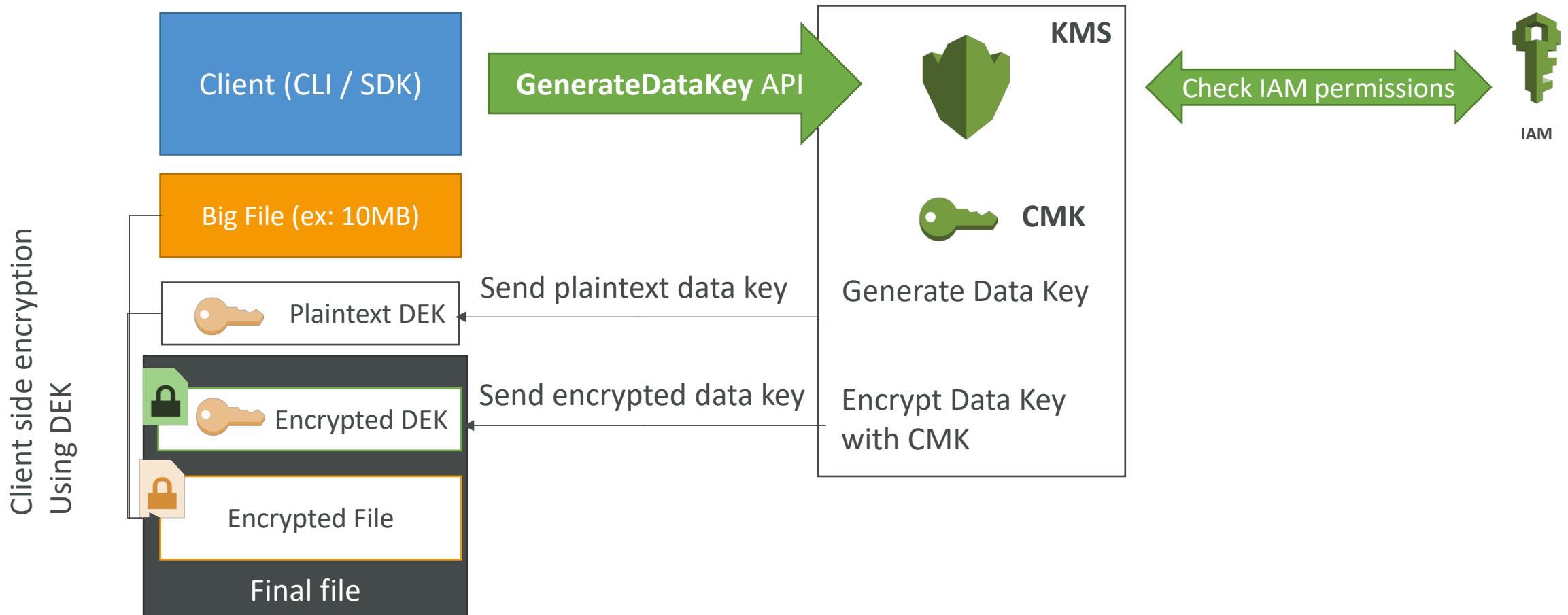


# AWS Encryption SDK

- What if you want to encrypt over 4 KB using KMS?
- For this, we need to use Envelope Encryption.
- Envelope Encryption is a bit cumbersome to implement
- The AWS Encryption SDK helps us use Envelope Encryption
- Note: It is different from the S3 Encryption SDK.
- The Encryption SDK also exists as a CLI tool we can install
  
- For the exam: anything over 4 KB of data that needs to be encrypted must use the Encryption SDK == Envelope Encryption == GenerateDataKey API

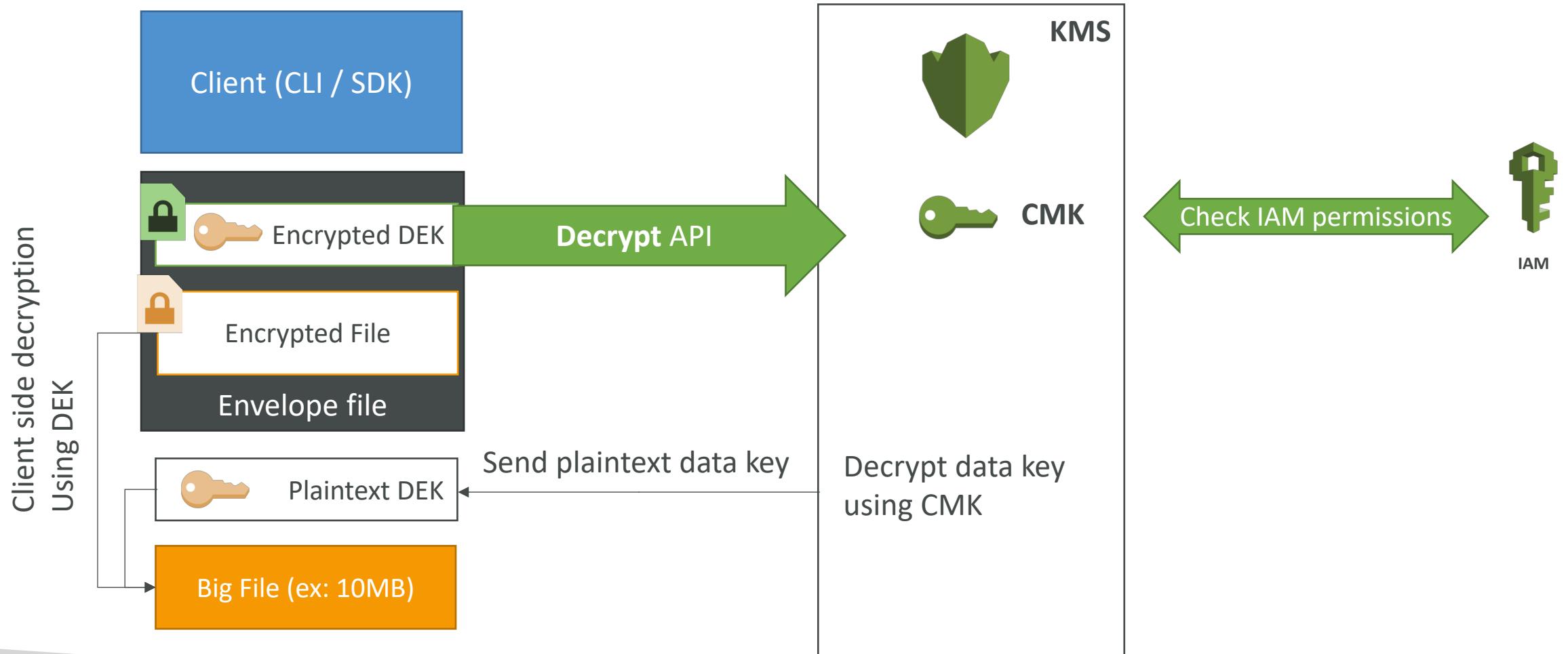
# Deep dive into Envelope Encryption

## GenerateDataKey API



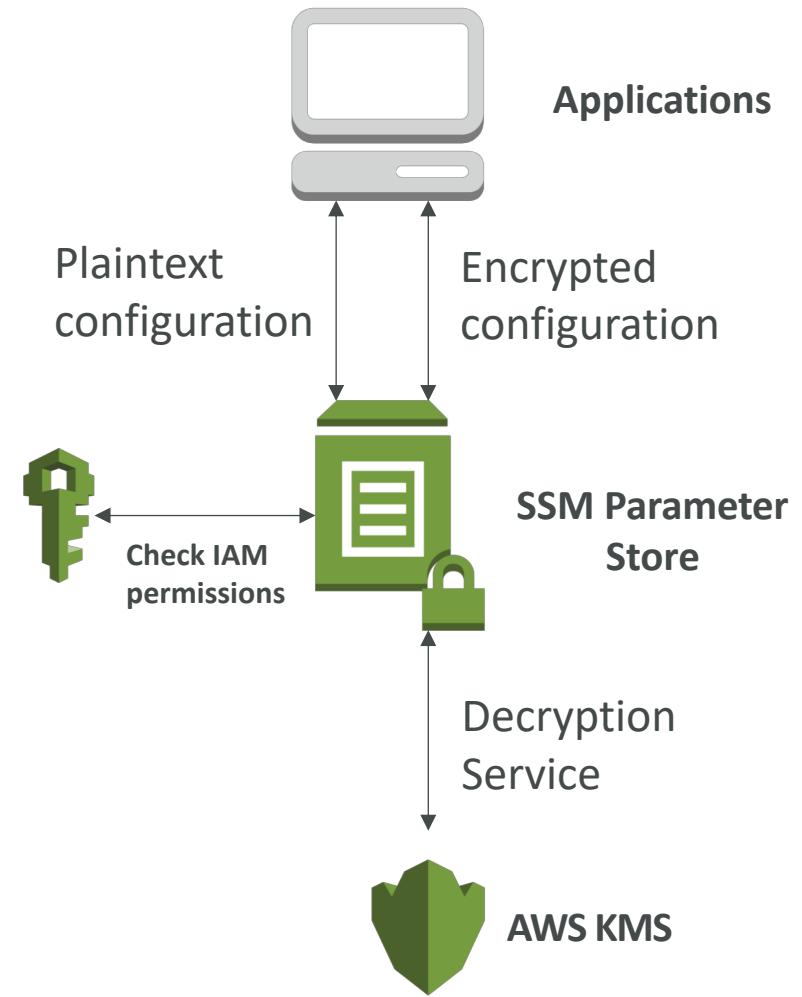
# Deep dive into Envelope Encryption

## Decrypt envelope data



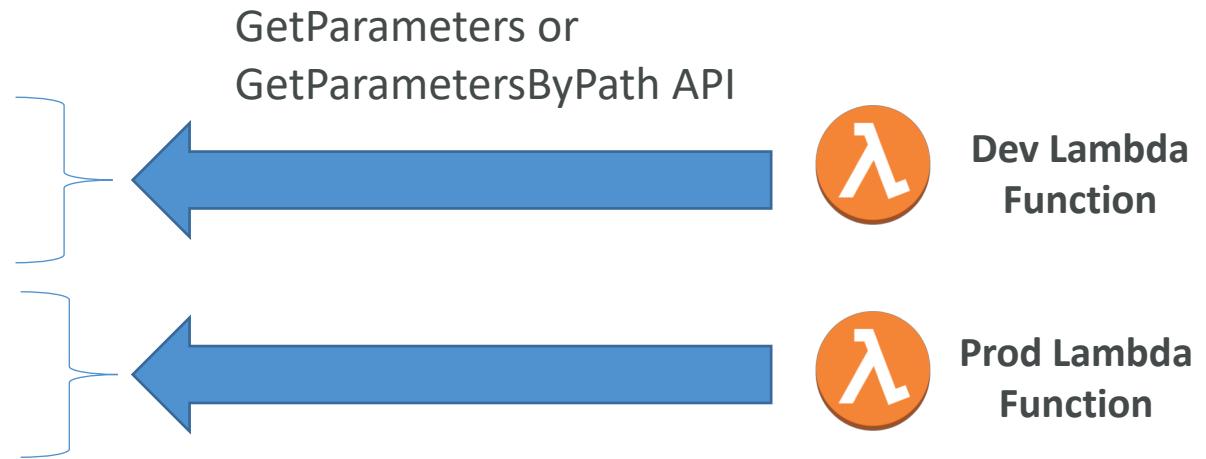
# AWS Parameter Store

- Secure storage for configuration and secrets
- Optional Seamless Encryption using KMS
- Serverless, scalable, durable, easy SDK, free
- Version tracking of configurations / secrets
- Configuration management using path & IAM
- Notifications with CloudWatch Events
- Integration with CloudFormation



# AWS Parameter Store Hierarchy

- /my-department/
  - my-app/
    - dev/
      - db-url
      - db-password
    - prod/
      - db-url
      - db-password
  - other-app/
  - /other-department/



# 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
- Probably not asked on the developer exam (yet)

# IAM Best Practices – General

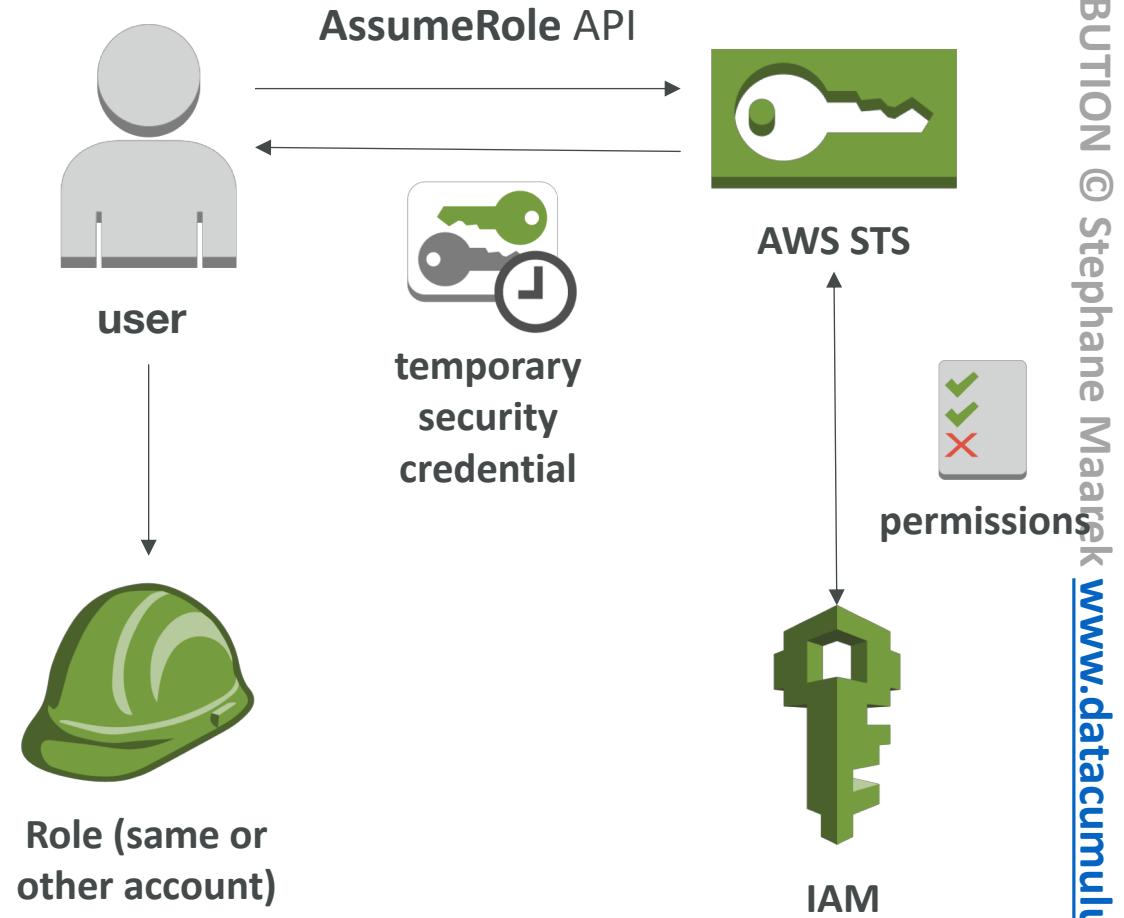
- Never use Root Credentials, enable MFA for Root Account
- Grant Least Privilege
  - Each Group / User / Role should only have the minimum level of permission it needs
  - Never grant a policy with “\*” access to a service
  - Monitor API calls made by a user in CloudTrail (especially Denied ones)
- 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)

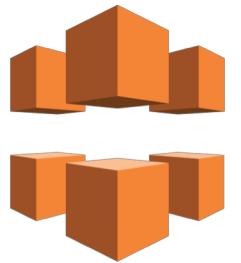
# IAM Best Practices – Cross Account Access

- 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



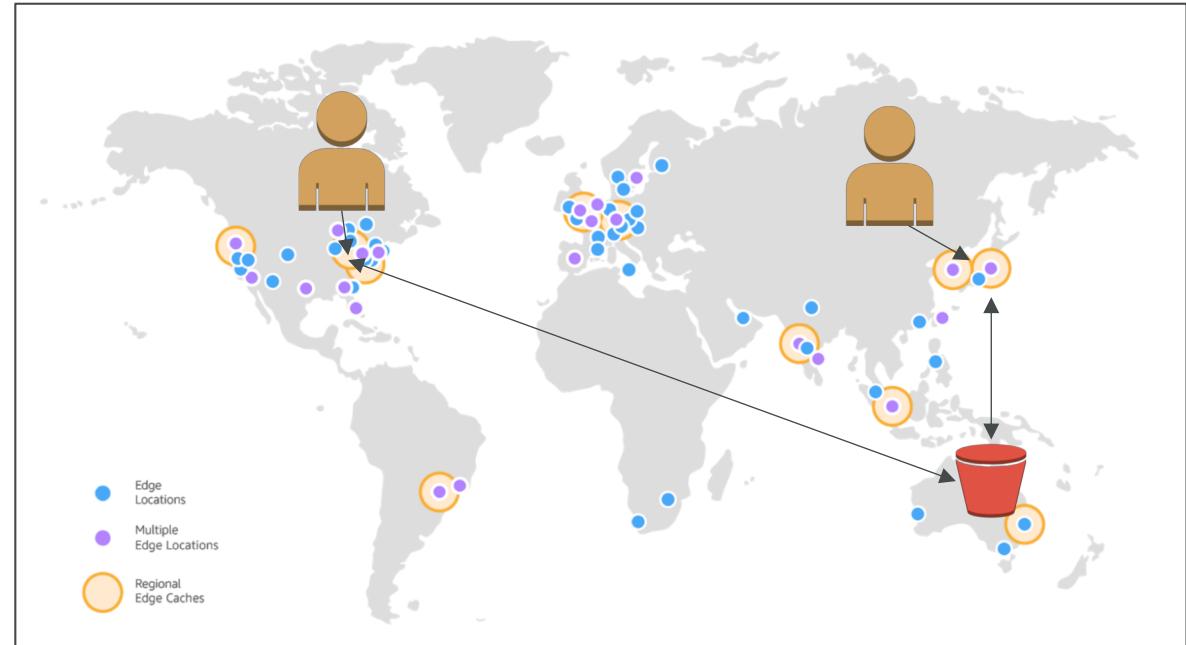
# Other AWS Services

Quick overview of other services that might have questions on at the exam



# AWS CloudFront

- Content Delivery Network (CDN)
- Improves read performance, content is cached at the edge
- 136 Point of Presence globally (edge locations)
- Popular with S3 but works with EC2, Load Balancing
- Can help protect against network attacks
- Can provide SSL encryption (HTTPS) at the edge using ACM
- CloudFront can use SSL encryption (HTTPS) to talk to your applications
- Support RTMP Protocol (videos / media)



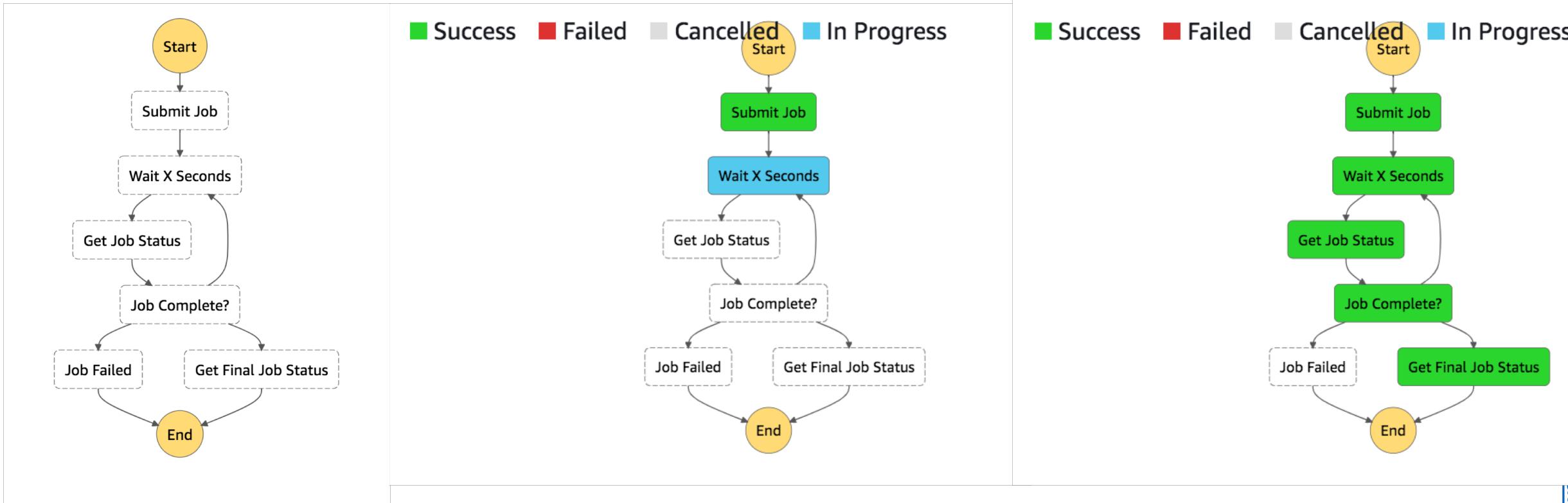
Source: <https://aws.amazon.com/cloudfront/features/?nc=sn&loc=2>



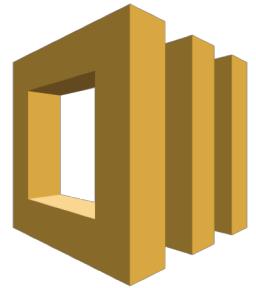
# AWS Step Functions

- Build serverless visual workflow to orchestrate your Lambda functions
- Represent flow as a **JSON state machine**
- Features: sequence, parallel, conditions, timeouts, error handling...
- Can also integrate with EC2, ECS, On premise servers, API Gateway
- Maximum execution time of 1 year
- Possibility to implement human approval feature
- Use cases:
  - Order fulfillment
  - Data processing
  - Web applications
  - Any workflow

# Visual workflow in Step Functions



# AWS SWF – Simple Workflow Service



- Coordinate work amongst applications
- Code runs on EC2 (not serverless)
- 1 year max runtime
- Concept of “activity step” and “decision step”
- Has built-in “human intervention” step
- Example: order fulfilment from web to warehouse to delivery
- **Step Functions is recommended to be used for new applications, except:**
  - If you need external signals to intervene in the processes
  - If you need child processes that return values to parent processes

# AWS SES – Simple Email Service

- Send emails to people using:
  - SMTP interface
  - Or AWS SDK
- Ability to receive email. Integrates with:
  - S3
  - SNS
  - Lambda
- Integrated with IAM for allowing to send emails

# AWS Databases Summary

- RDS: Relational databases, OLTP
  - PostgreSQL, MySQL, Oracle...
  - Aurora + Aurora Serverless
  - Provisioned database
- DynamoDB: NoSQL DB
  - Managed, Key Value, Document
  - Serverless
- ElastiCache: In memory DB
  - Redis / Memcached
  - Cache capability
- Redshift: OLAP – Analytic Processing
  - Data Warehousing / Data Lake
  - Analytics queries
- Neptune: Graph Database
- DMS: Database Migration Service

# 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 CI/CD 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 two identity documents (ID, Credit Card, 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
- At the end you can optionally review all the questions / answers
  
- You will know right away if you passed / failed the exams
- You will not know which answers were right / wrong
- You will know the overall score a few days later (email notification)
- To pass you need a score of at least 720 out of 1000
- If you fail, you can retake the exam again 14 days later

# 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!