

# SECTION 1

## Getting Started

# Exam Guide

## Exam Details

**Format**

Multiple choice, multiple answer

**Type**

Associate

**Delivery Method**

Testing center or online proctored exam

**Time**

130 mins to complete the exam

**Cost**

150 USD (Practice exam: 20 USD)

**Language**

Available in English, Japanese, Korean,  
and Simplified Chinese



## Neal Davis

has successfully completed the AWS Certification requirements and has achieved their:

### AWS Certified Developer - Associate

**Issue Date**  
Feb 14, 2020

**Expiration Date**  
Feb 14, 2023

A handwritten signature in black ink.

Maureen Lonergan  
Director, Training and Certification

Validation Number 3BXJB782314QQT9X  
Validate at: <http://aws.amazon.com/verification>

# Exam Guide

## Abilities Validated by the Certification

- Demonstrate an understanding of core AWS services, uses, and basic AWS architecture best practices
- Demonstrate proficiency in developing, deploying, and debugging cloud-based applications using AWS

## Exam Guide

### Recommended knowledge:

- In-depth knowledge of at least one high-level programming language
- Understanding of core AWS services, uses, and basic AWS architecture best practices
- Proficiency in developing, deploying, and debugging cloud-based applications using AWS
- Ability to use the AWS service APIs, AWS CLI, and SDKs to write applications

## Exam Guide

### Time and Length:

- 130 minutes
- 65 questions
- Scoring:
  - Scaled score between 100 – 1000
  - Minimum passing score of 720

### Question format:

- Multiple-choice: Has one correct response and three incorrect responses
- Multiple-response: Has two or more correct responses out of five or more options

# Exam Guide

## Domain 1: Deployment

- 1.1 Deploy written code in AWS using existing CI/CD pipelines, processes, and patterns
- 1.2 Deploy applications using Elastic Beanstalk
- 1.3 Prepare the application deployment package to be deployed to AWS
- 1.4 Deploy serverless applications

## Domain 2: Security

- 2.1 Make authenticated calls to AWS services
- 2.2 Implement encryption using AWS services
- 2.3 Implement application authentication and authorization

# Exam Guide

## Domain 3: Development with AWS Services

- 3.1 Write code for serverless applications
- 3.2 Translate functional requirements into application design
- 3.3 Implement application design into application code
- 3.4 Write code that interacts with AWS services by using APIs, SDKs, and AWS CLI

## Domain 4: Refactoring

- 4.1 Optimize application to best use AWS services and features
- 4.2 Migrate existing application code to run on AWS

# Exam Guide

## Domain 5: Monitoring and Troubleshooting

- 5.1 Write code that can be monitored
- 5.2 Perform root cause analysis on faults found in testing or production

# Exam Guide

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

## SECTION 2

# Overview of IAM, VPC & AWS Free Tier Account Setup

## Section 2: Identity and Access Management (IAM) Overview

### AWS Identity and Access Management (IAM)



User



IAM Role



Group



Multi-Factor  
Authentication



IAM Policy



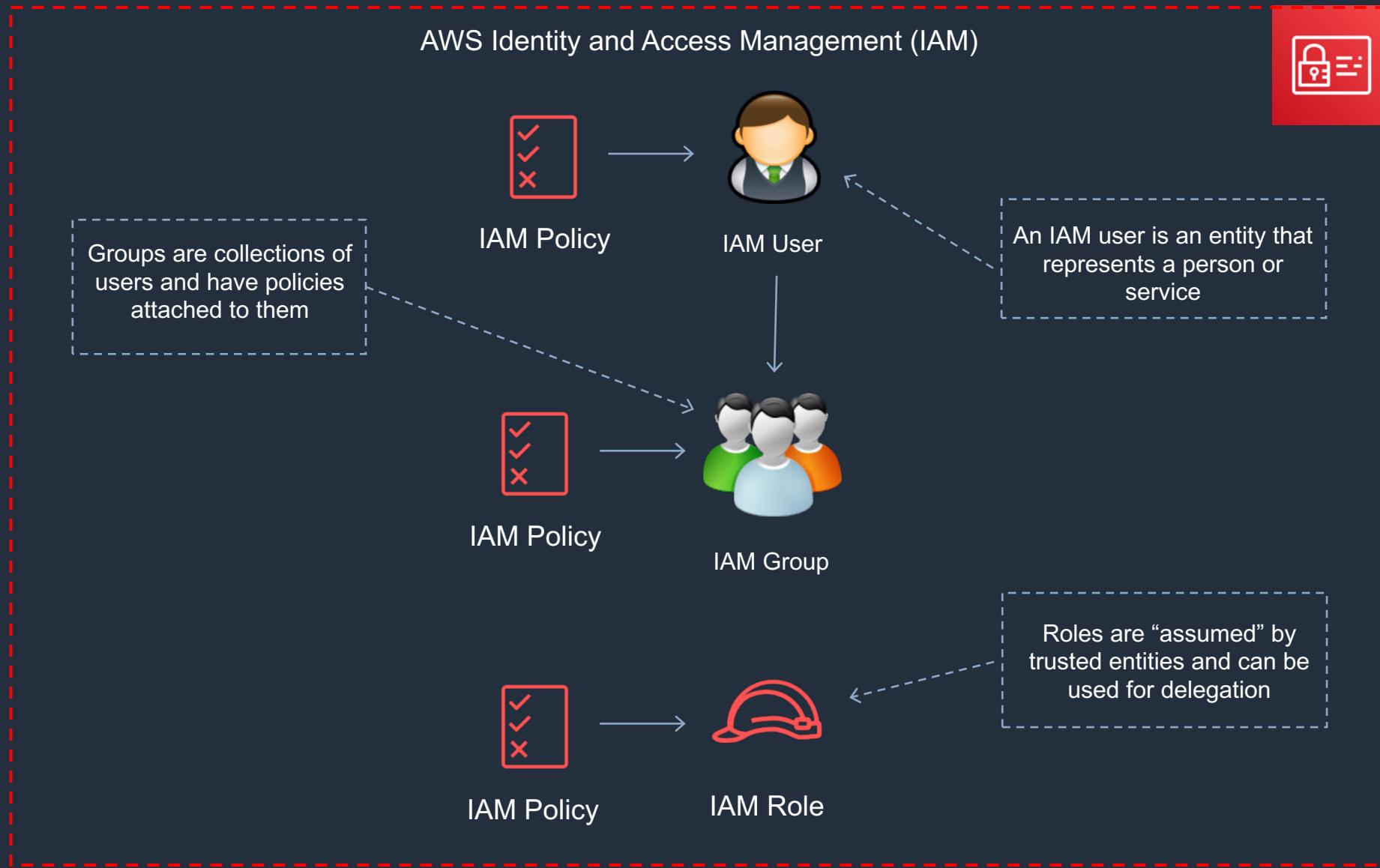
Identity Federation



API Keys  
(programmatic  
access)



## Section 2: IAM Users, Groups, Roles and Policies



## Section 2: IAM Users

- An IAM user is an entity that represents a person or service
- Can be assigned:
  - An access key ID and secret access key for programmatic access to the AWS API, CLI, SDK, and other development tools
  - A password for access to the management console
- By default users cannot access anything in your account
- The account root user credentials are the email address used to create the account and a password
- The root account has full administrative permissions and these cannot be restricted
- Best practice for root accounts:
  - Don't use the root user credentials
  - Don't share the root user credentials
  - Create an IAM user and assign administrative permissions as required
  - Enable Multi-Factor Authentication (MFA)



Eric



Ethan



Andrea

## Section 2: IAM Users

- IAM users can be created to represent applications and these are known as “service accounts”
- You can have up to 5000 users per AWS account
- Each user account has a friendly name and an Amazon Resource Name (ARN) which uniquely identifies the user across AWS
- You should create individual IAM accounts for users (best practice not to share accounts)
- A password policy can be defined for enforcing password length, complexity etc. (applies to all users)



Eric



Ethan



Andrea

## Section 2: IAM Groups

- Groups are collections of users and have policies attached to them
- A group is not an identity and cannot be identified as a principal in an IAM policy
- Use groups to assign permissions to users
- Use the principle of least privilege when assigning permissions
- You cannot nest groups (groups within groups)



Developers



AWS Admins



Operations

## Section 2: IAM Roles

- Roles are created and then “assumed” by trusted entities and define a set of permissions for making AWS service requests
- With IAM Roles you can delegate permissions to resources for users and services without using permanent credentials (e.g. user name and password)
- IAM users or AWS services can assume a role to obtain temporary security credentials that can be used to make AWS API calls



S3 Full Access



DynamoDB Read-Only



AWSLambdaBasicExecutionRole

## Section 2: IAM Policies

- Policies are documents that define permissions and can be applied to users, groups and roles
- Policy documents are written in JSON (key value pair that consists of an attribute and a value)
- All permissions are implicitly denied by default
- The most restrictive policy is applied
- The IAM policy simulator is a tool to help you understand, test, and validate the effects of access control policies
- The Condition element can be used to apply further conditional logic



S3 Full Access

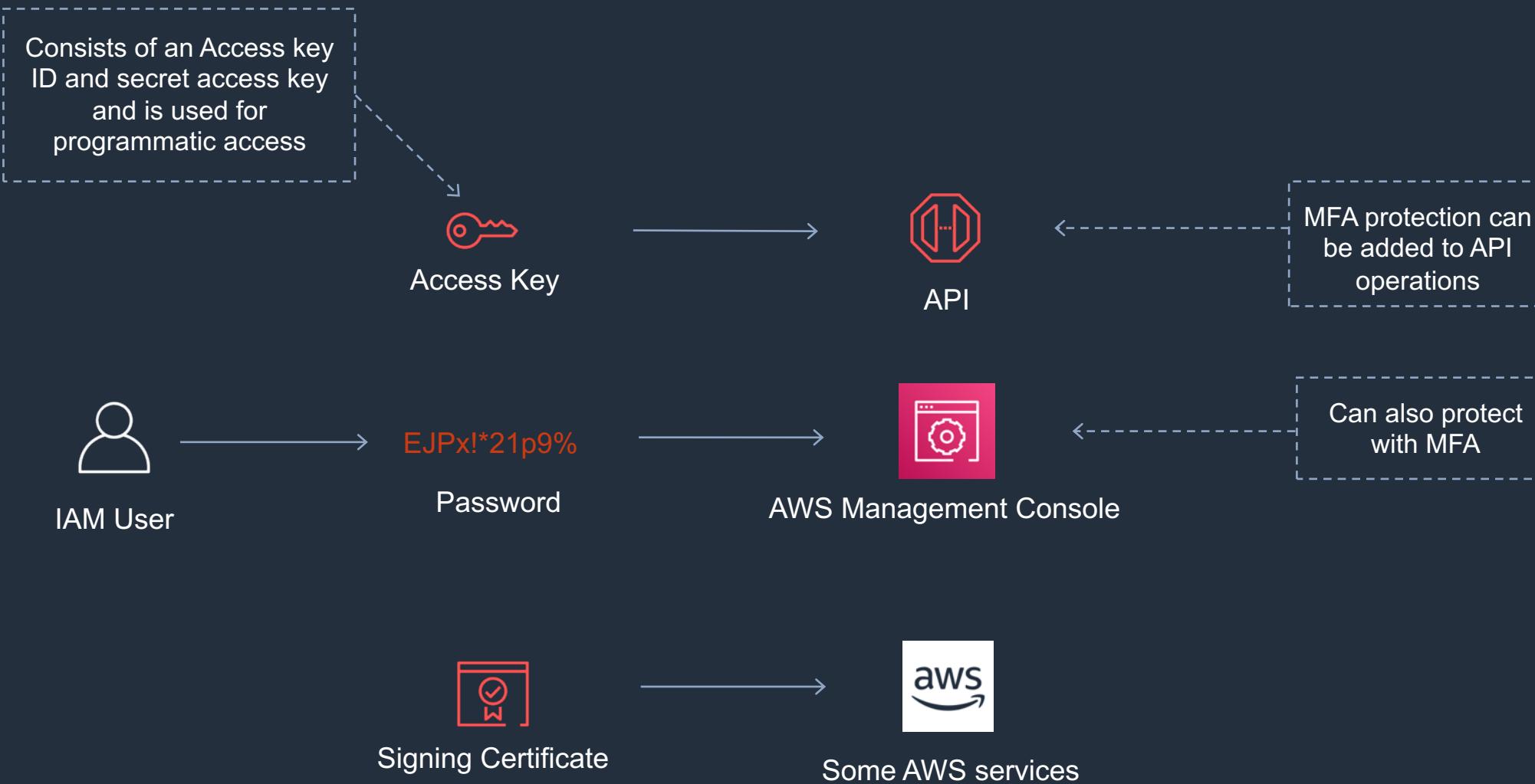


DynamoDB Read-Only



AWSLambdaBasicExecutionRole

## Section 2: Authentication Methods



## Section 2: IAM Access Keys



Access Key



API

- A combination of an access key ID and a secret access key
- These can be used to make programmatic calls to AWS when using the API in program code or at a command prompt when using the AWS CLI or the AWS PowerShell tools
- You can create, modify, view or rotate access keys
- When created IAM returns the access key ID and secret access key
- The secret access is returned only at creation time and if lost a new key must be created
- Ensure access keys and secret access keys are stored securely
- Users can be given access to change their own keys through IAM policy (not from the console)
- You can disable a user's access key which prevents it from being used for API calls

## Section 2: IAM Console Password

- A password that the user can enter to sign into interactive sessions such as the AWS Management Console
- You can allow users to change their own passwords
- You can allow selected IAM users to change their passwords by disabling the option for all users and using an IAM policy to grant permissions for the selected users



## Section 2: IAM Server Certificate / Signing Certificate

- SSL/TLS certificates that you can use to authenticate with some AWS services
- AWS recommends that you use the AWS Certificate Manager (ACM) to provision, manage and deploy your server certificates
- Use IAM only when you must support HTTPS connections in a region that is not supported by ACM



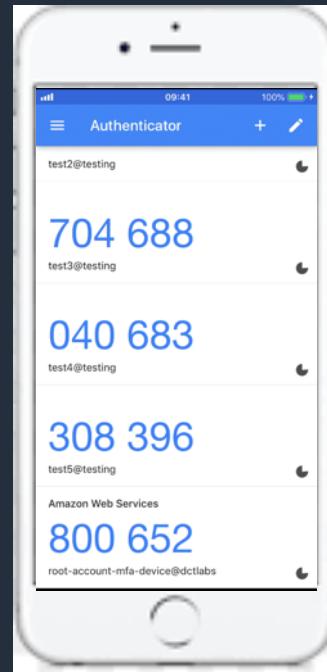
## Section 2: Multi-Factor Authentication

Something you **know**:

EJPx!\*21p9%

Password

Something you **have**:



Something you **are**:



## Section 2: Multi-Factor Authentication in AWS

Something you **know**:



IAM User

EJPx!\*21p9%

Password

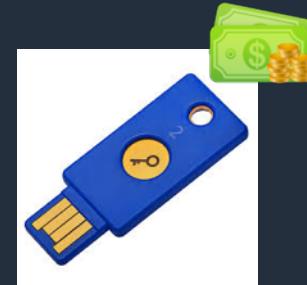
Something you **have**:



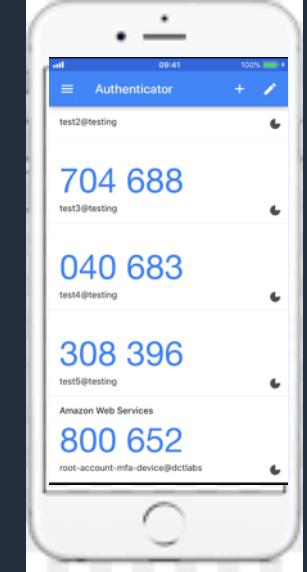
Virtual MFA



Physical MFA



e.g. Google Authenticator on  
your smart phone

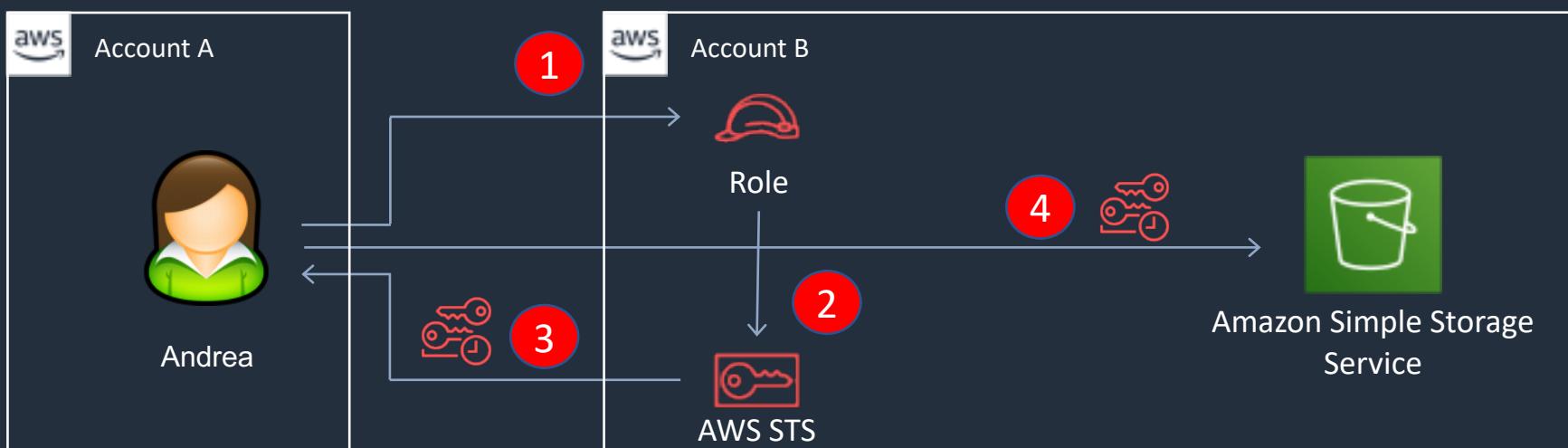


## Section 2: AWS Security Token Service (STS)

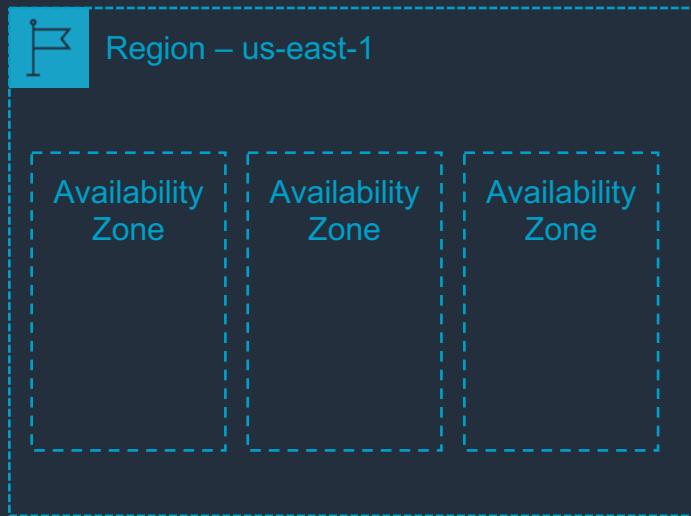
- The AWS Security Token Service (STS) is a web service that enables you to request temporary, limited-privilege credentials for IAM users or for users that you authenticate (federated users)
- By default, AWS STS is available as a global service, and all AWS STS requests go to a single endpoint at <https://sts.amazonaws.com>
- All regions are enabled for STS by default but can be disabled
- The region in which temporary credentials are requested must be enabled
- Credentials will always work globally



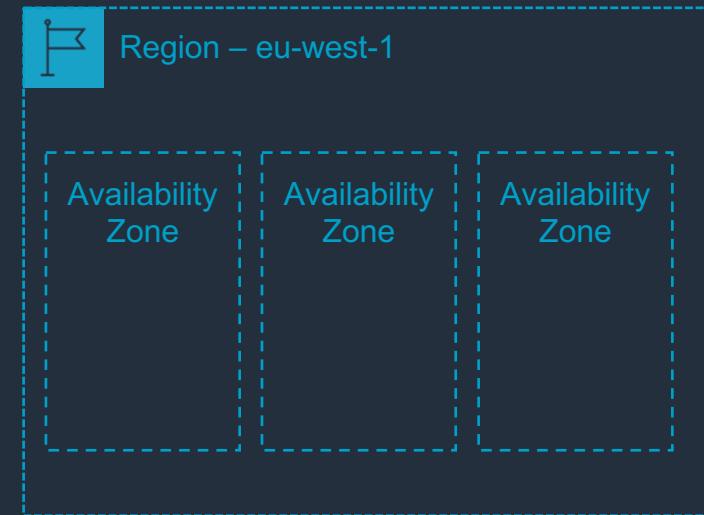
Temporary security credential



## Section 2: AWS Global Infrastructure

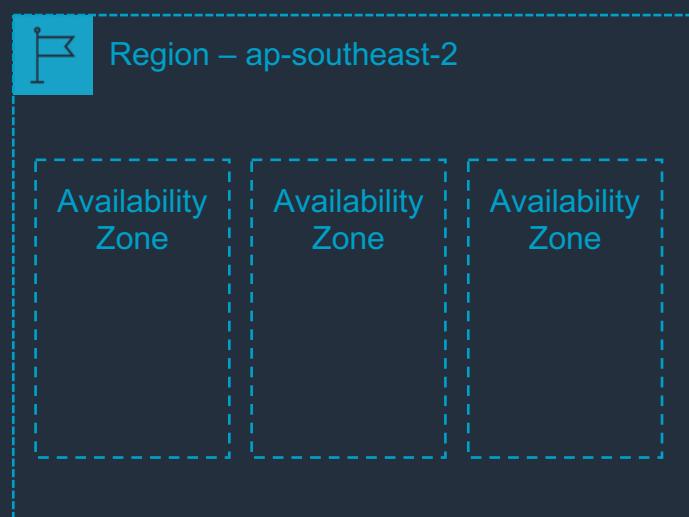


Every region is connected via a high bandwidth, full redundant network



There are 23 regions around the world

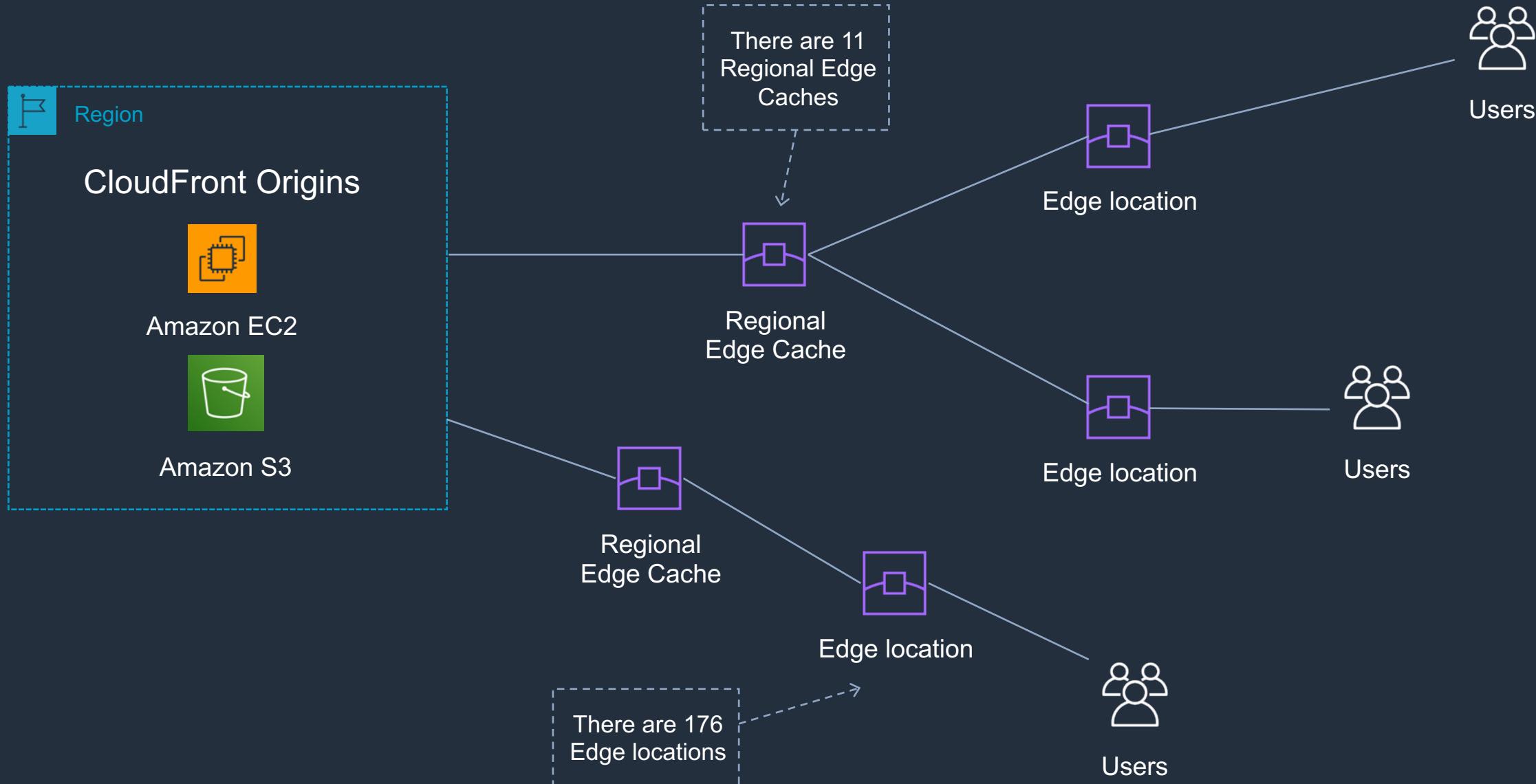
Each region is completely independent



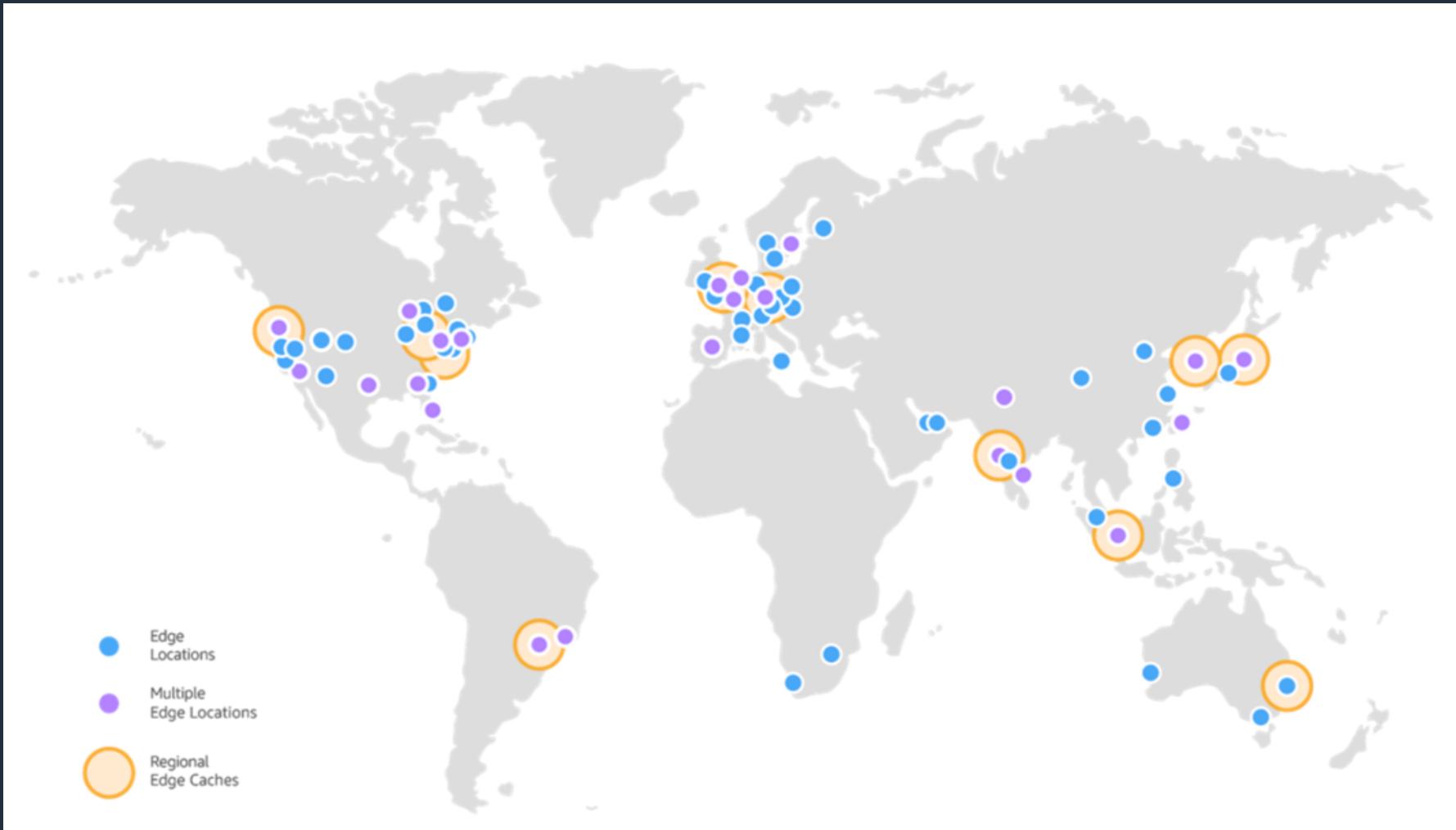
## Section 2: AWS Global Infrastructure



## Section 2: CloudFront Edge Locations



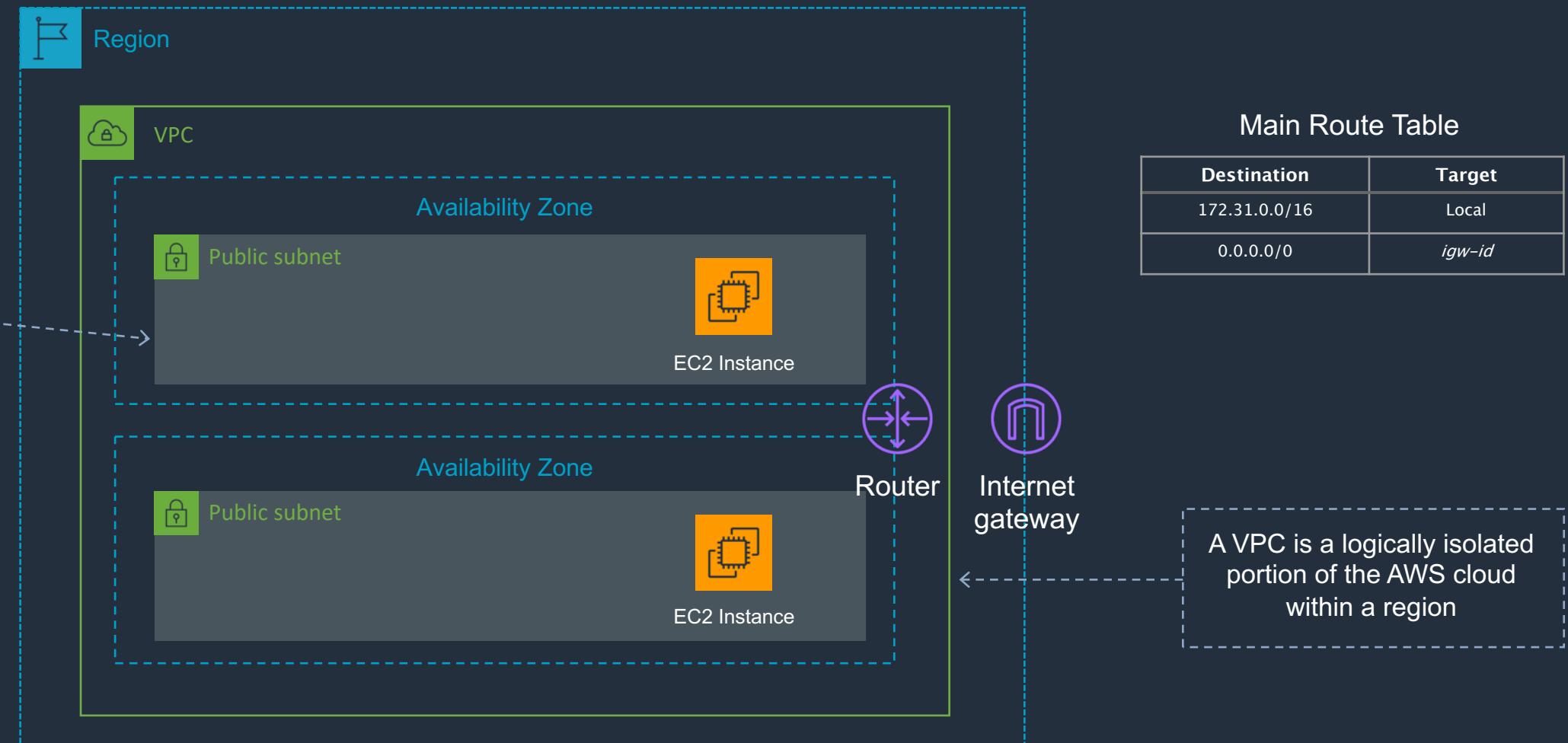
## Section 2: CloudFront Edge Locations



## Section 2: AWS Global Infrastructure

Name	Description
Region	A geographical area with 2 or more AZs, isolated from other AWS regions
Availability Zone (AZ)	One or more data centers that are physically separate and isolated from other AZs
Edge Location	A location with a cache of content that can be delivered at low latency to users – used by CloudFront
Regional Edge Cache	Also part of the CloudFront network. These are larger caches that sit between AWS services and Edge Locations
Global Network	Highly available, low-latency private global network interconnecting every data center, AZ, and AWS region

## Section 2: VPC Overview



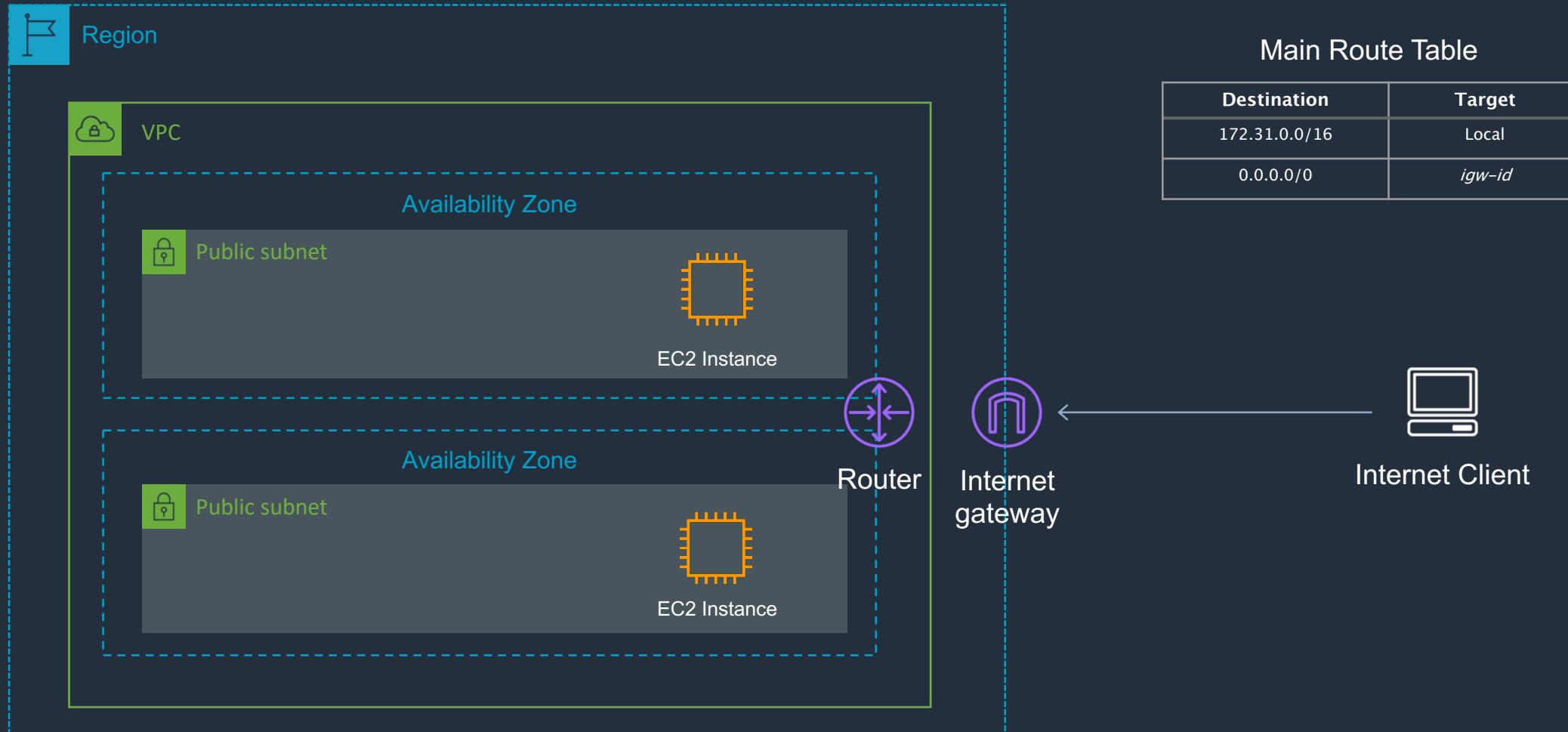
## Section 2: VPC Overview



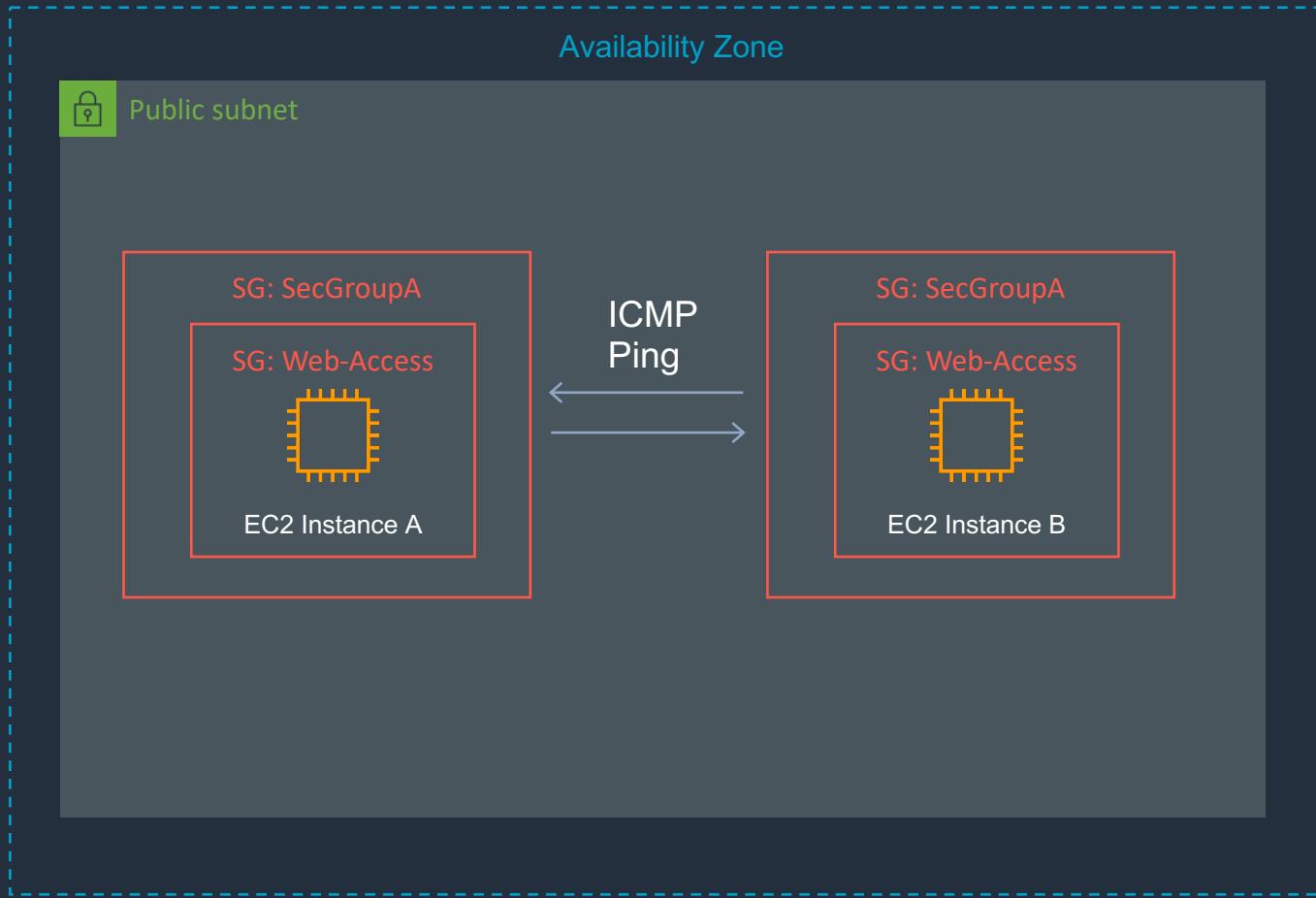
# SECTION 3

# Amazon Elastic Compute Cloud (EC2)

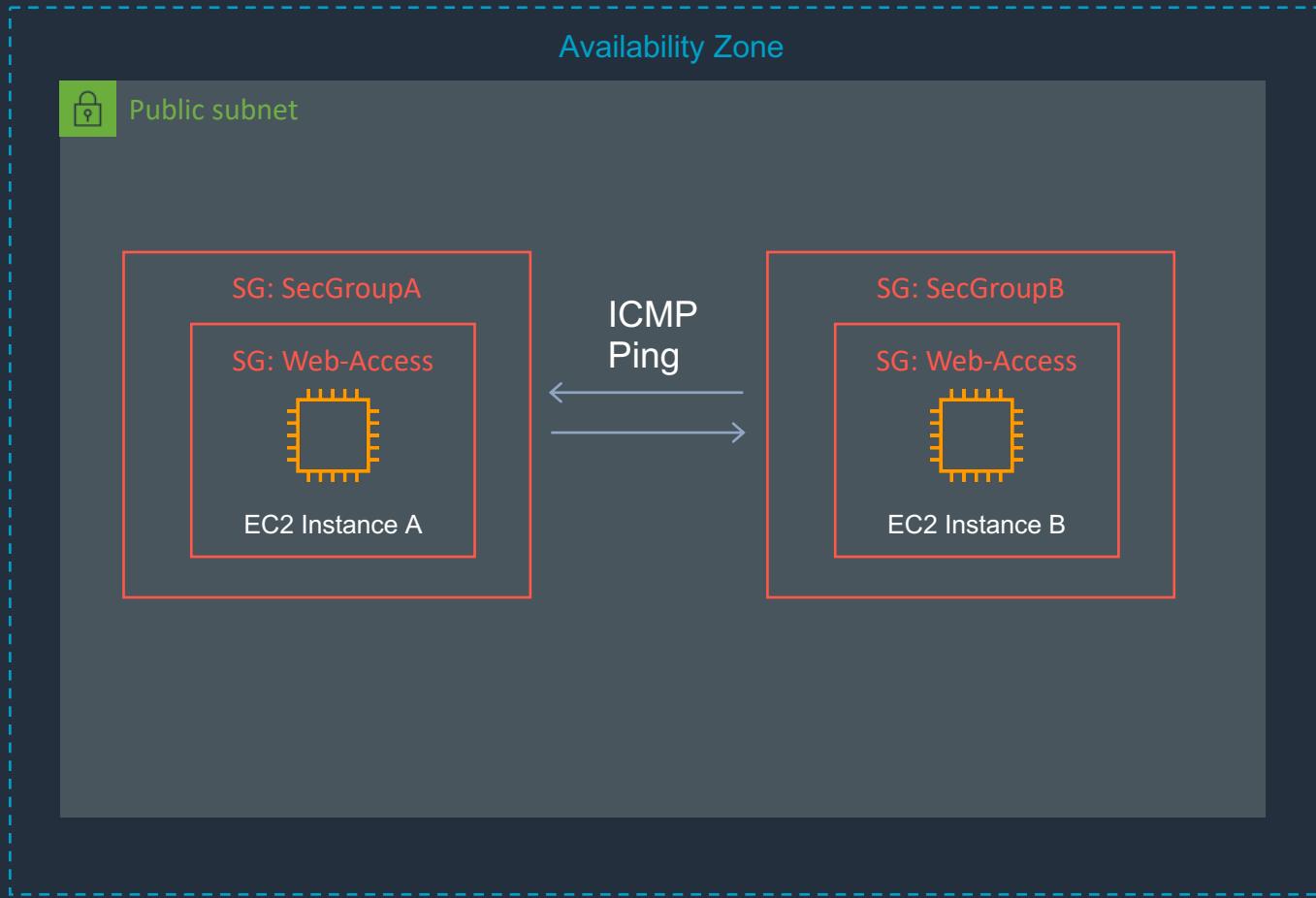
## Section 3: Launch an EC2 Instance



## Section 3: Security Group Slide 1



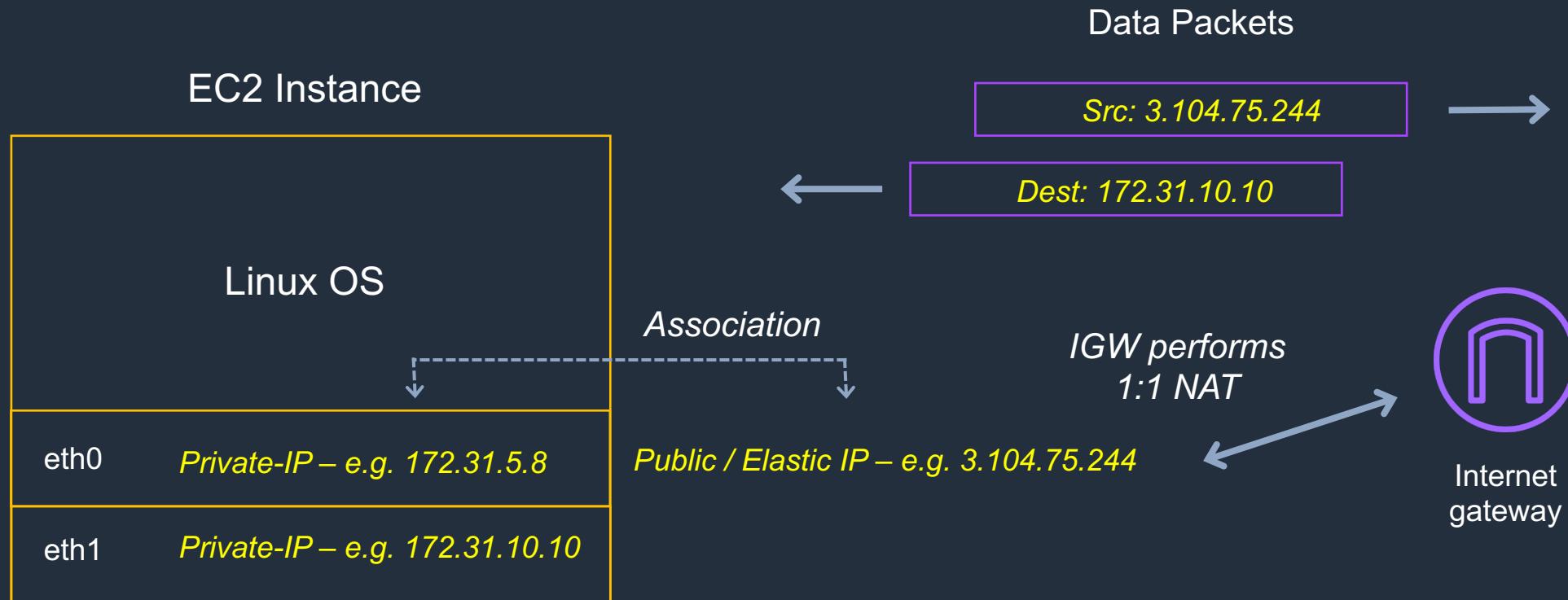
## Section 3: Security Group Slide 2



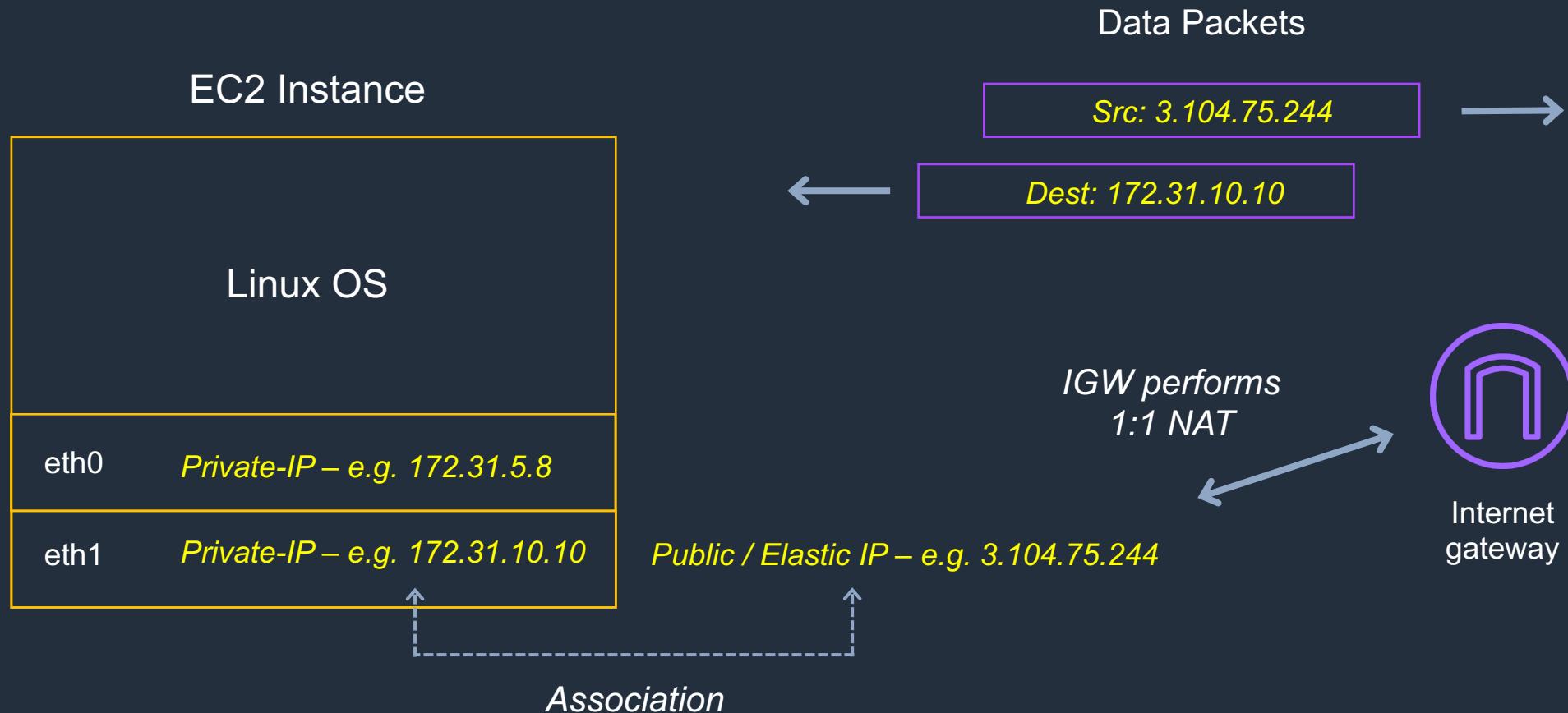
## Section 3: Public, Private, and Elastic IP addresses

Name	Description
Public IP address	<p><b>Lost when the instance is stopped</b></p> <p><b>Used in Public Subnets</b></p> <p><b>No charge</b></p> <p><b>Associated with a private IP address on the instance</b></p> <p><b>Cannot be moved between instances</b></p>
Private IP address	<p><b>Retained when the instance is stopped</b></p> <p><b>Used in Public and Private Subnets</b></p>
Elastic IP address	<p><b>Static Public IP address</b></p> <p><b>You are charged if not used</b></p> <p><b>Associated with a private IP address on the instance</b></p> <p><b>Can be moved between instances and Elastic Network Adapters</b></p>

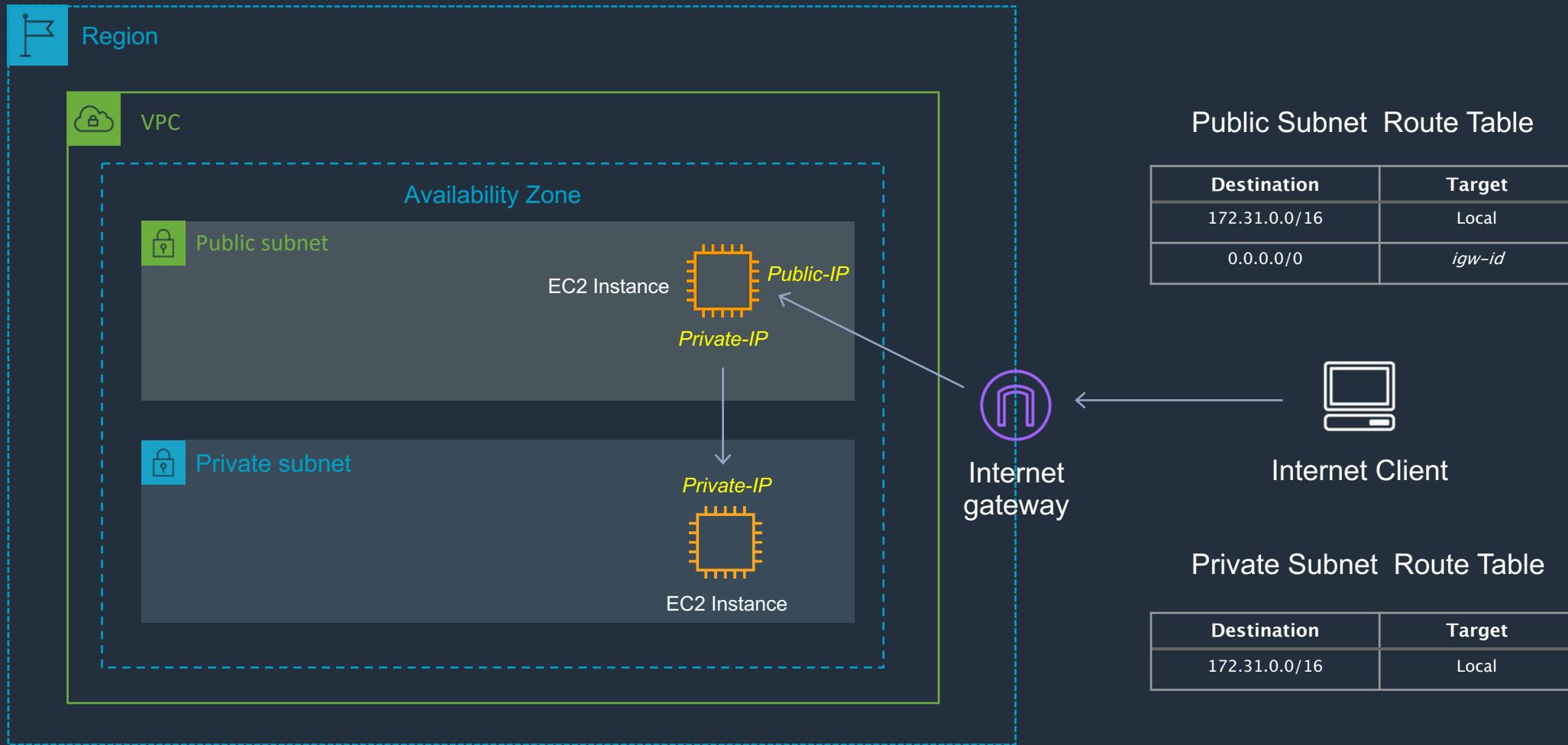
## Section 3: Public, Private and Elastic IPs - Slide 1



## Section 3: Public, Private and Elastic IPs - Slide 2



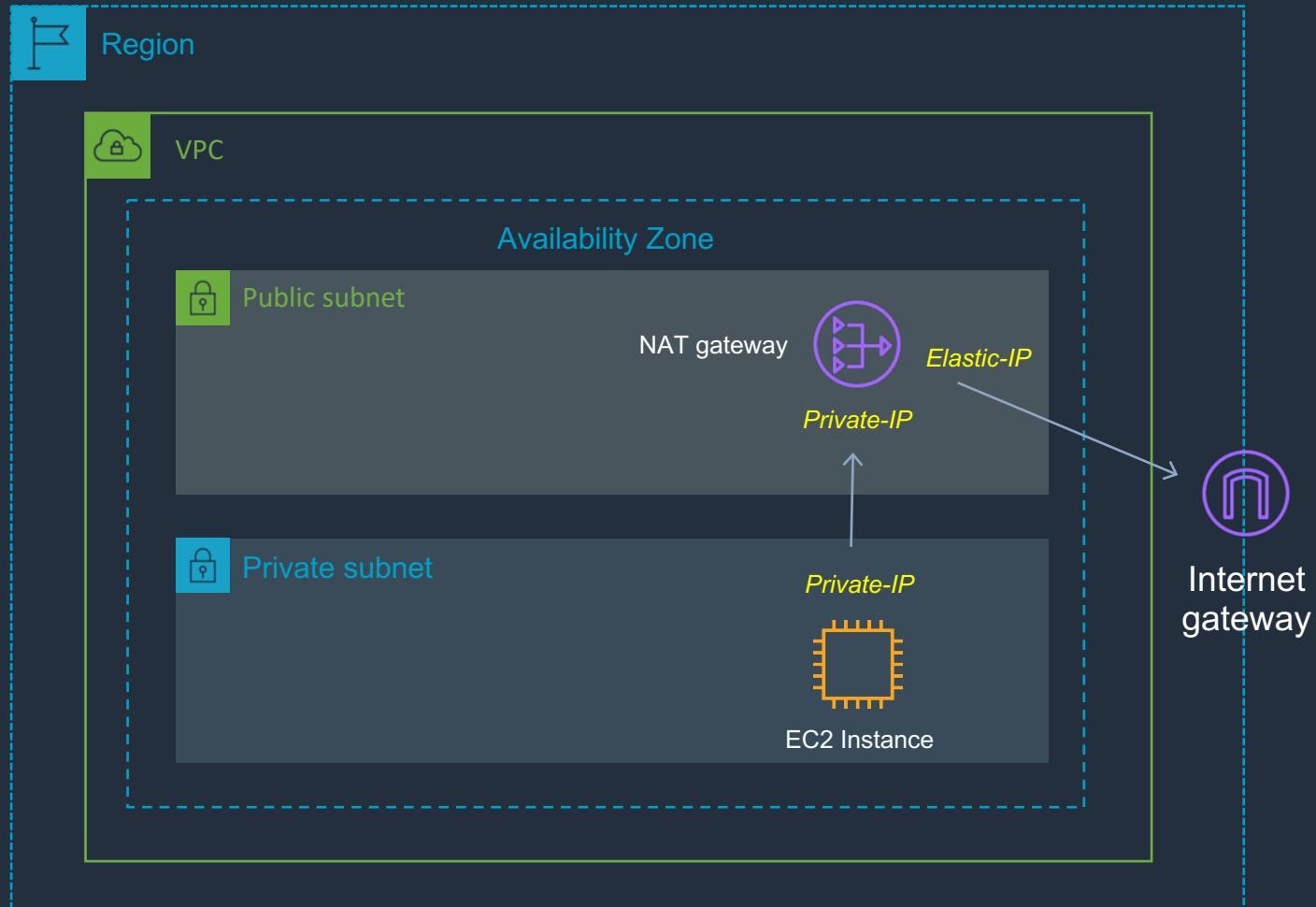
## Section 3: Private Subnets and Bastion Hosts



## Section 3: NAT Instance vs NAT Gateway

NAT Instance	NAT Gateway
Managed by you (e.g. software updates)	Managed by AWS
Scale up (instance type) manually and use enhanced networking	Elastic scalability up to 45 Gbps
No high availability – scripted/auto-scaled HA possible using multiple NATs in multiple subnets	Provides automatic high availability within an AZ and can be placed in multiple AZs
Need to assign Security Group	No Security Groups
Can use as a bastion host	Cannot access through SSH
Use an Elastic IP address or a public IP address with a NAT instance	Choose the Elastic IP address to associate with a NAT gateway at creation
Can implement port forwarding through manual customisation	Does not support port forwarding

## Section 3: Private Subnet with NAT Gateway



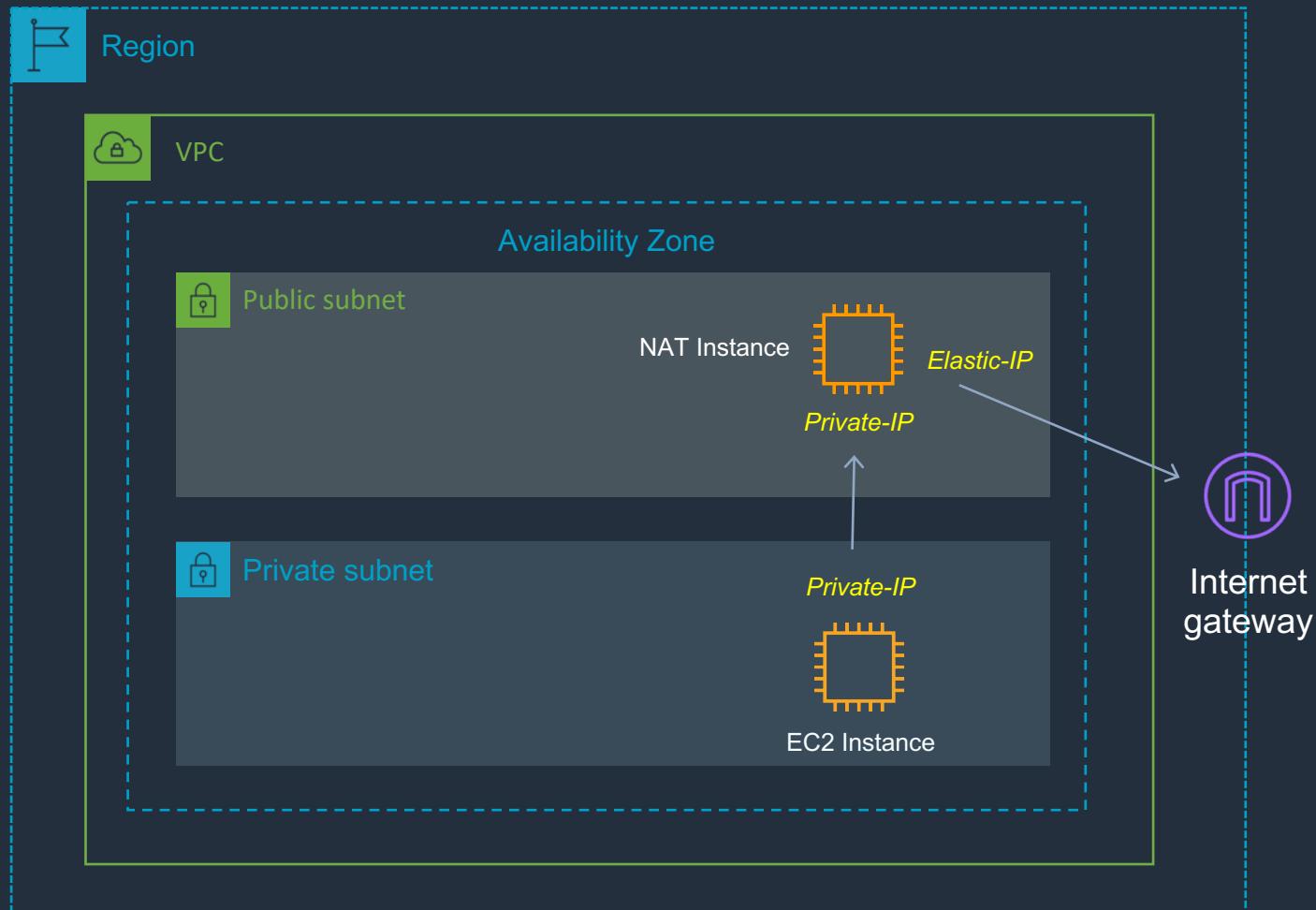
Public Subnet Route Table

Destination	Target
172.31.0.0/16	Local
0.0.0.0/0	<i>igw-id</i>

Private Subnet Route Table

Destination	Target
172.31.0.0/16	Local
0.0.0.0/0	<i>nat-gateway-id</i>

## Section 3: Private Subnet with NAT Instance



Public Subnet Route Table

Destination	Target
172.31.0.0/16	Local
0.0.0.0/0	<i>igw-id</i>

Private Subnet Route Table

Destination	Target
172.31.0.0/16	Local
0.0.0.0/0	<i>nat-instance-id</i>

# SECTION 4

## Elastic Load Balancing and Auto Scaling

## Section 4: Amazon S3 Overview



S3 Bucket

[http://\*bucket\*.s3.\*aws-region\*.amazonaws.com](http://bucket.s3.amazonaws.com)  
[http://s3.\*aws-region\*.amazonaws.com/\*bucket\*](http://s3.amazonaws.com/bucket)



Object

- Key
- Version ID
- Value
- Metadata
- Subresources
- Access control information



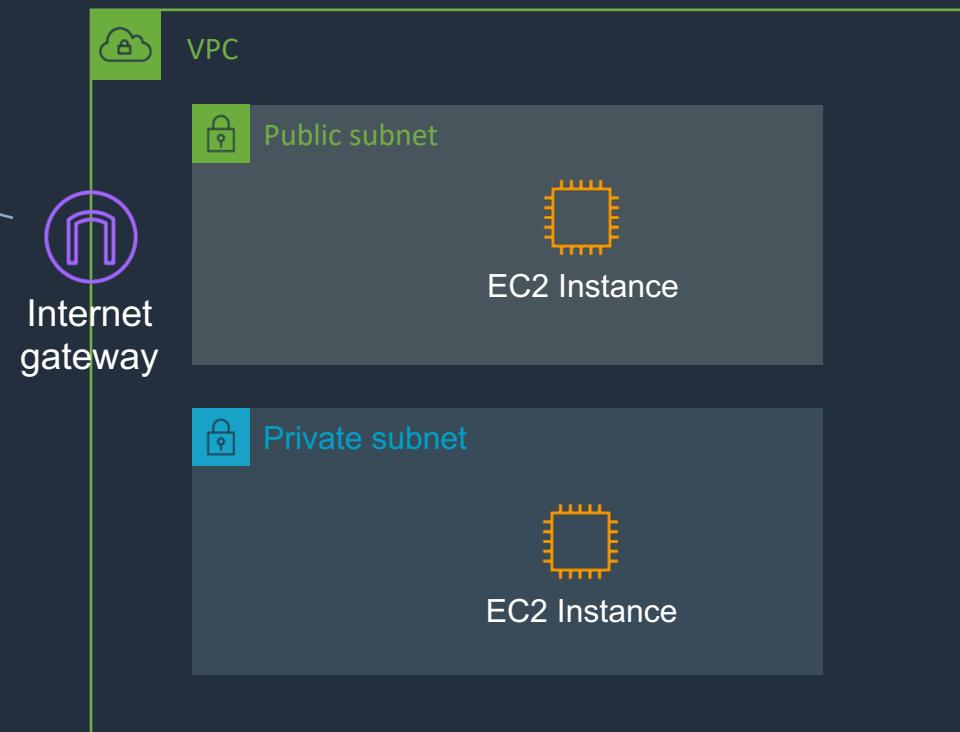
Public Internet



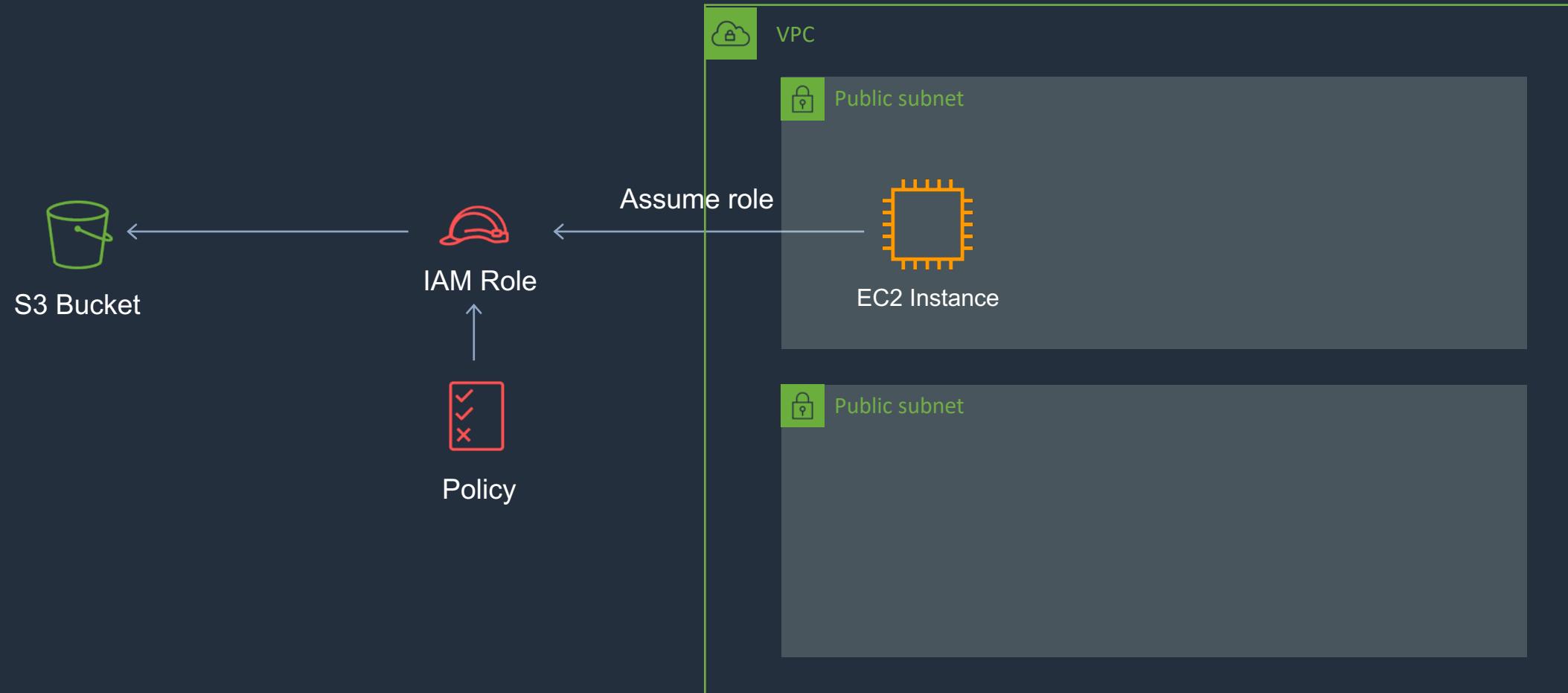
Internet Client



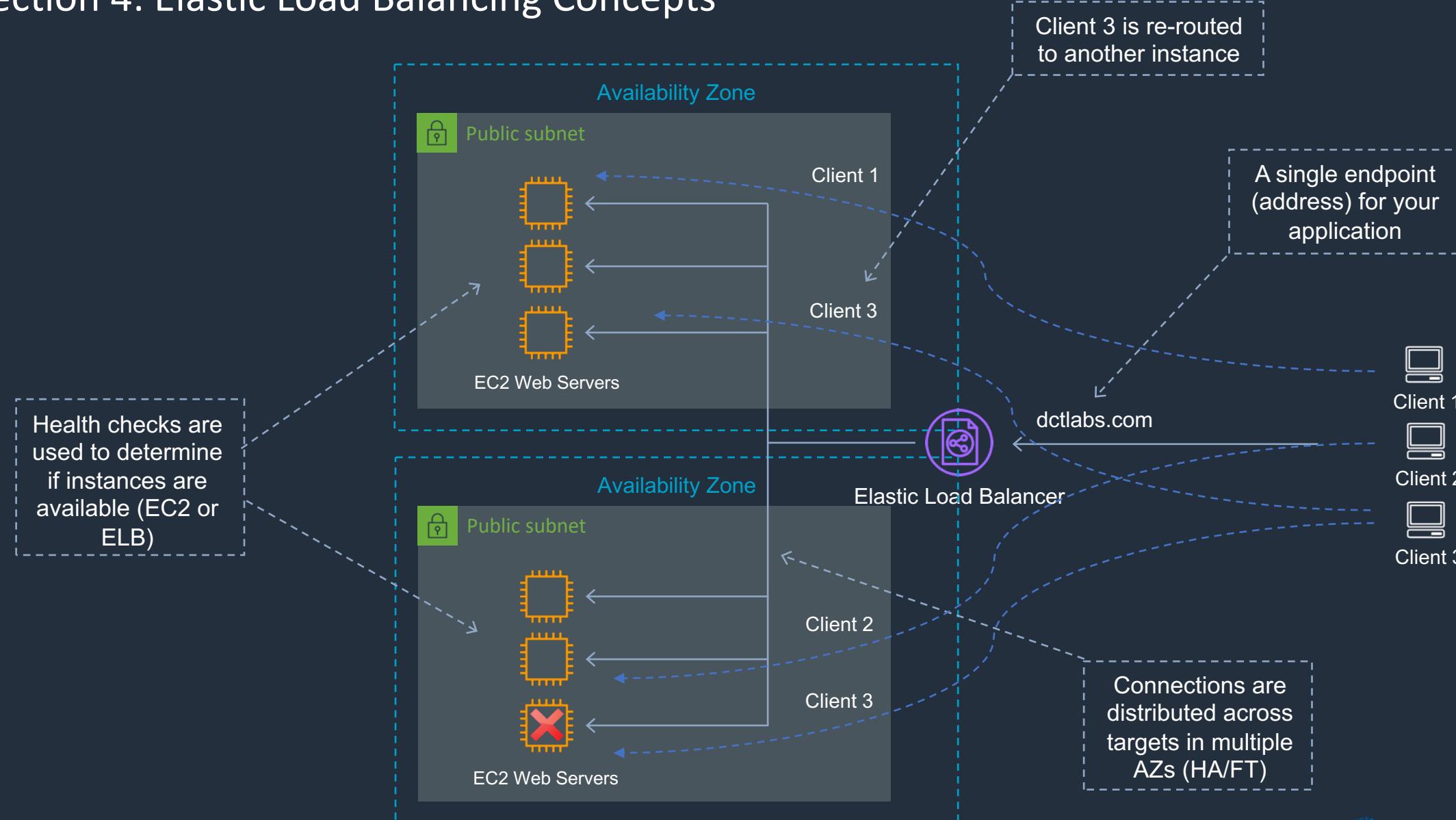
Amazon S3



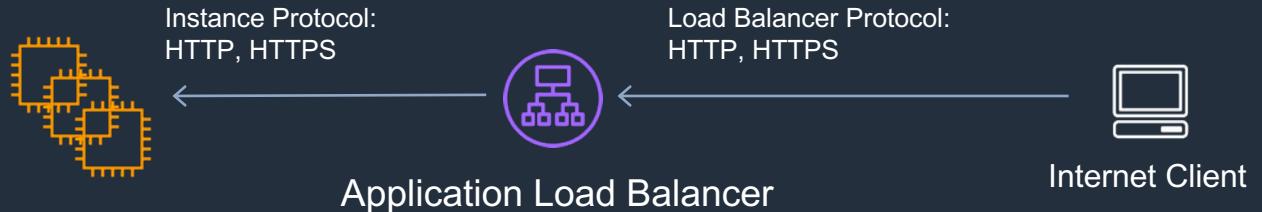
## Section 4: IAM Roles



## Section 4: Elastic Load Balancing Concepts

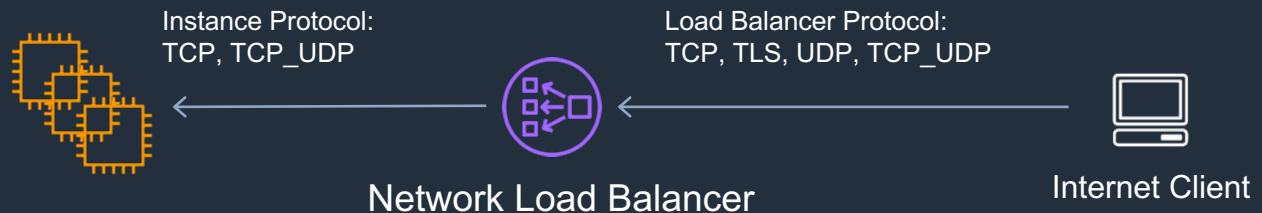


## Section 4: Elastic Load Balancing (ELB) Types



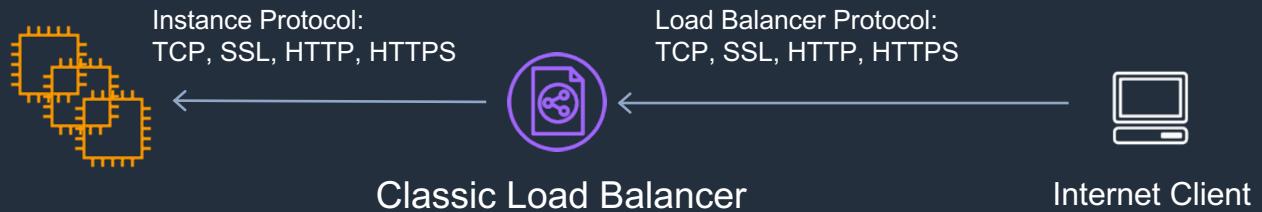
### Application Load Balancer

- Operates at the request level
- Routes based on the content of the request (layer 7)
- Supports path-based routing, host-based routing, query string parameter-based routing, and source IP address-based routing
- Supports IP addresses, Lambda Functions and containers as targets



### Network Load Balancer

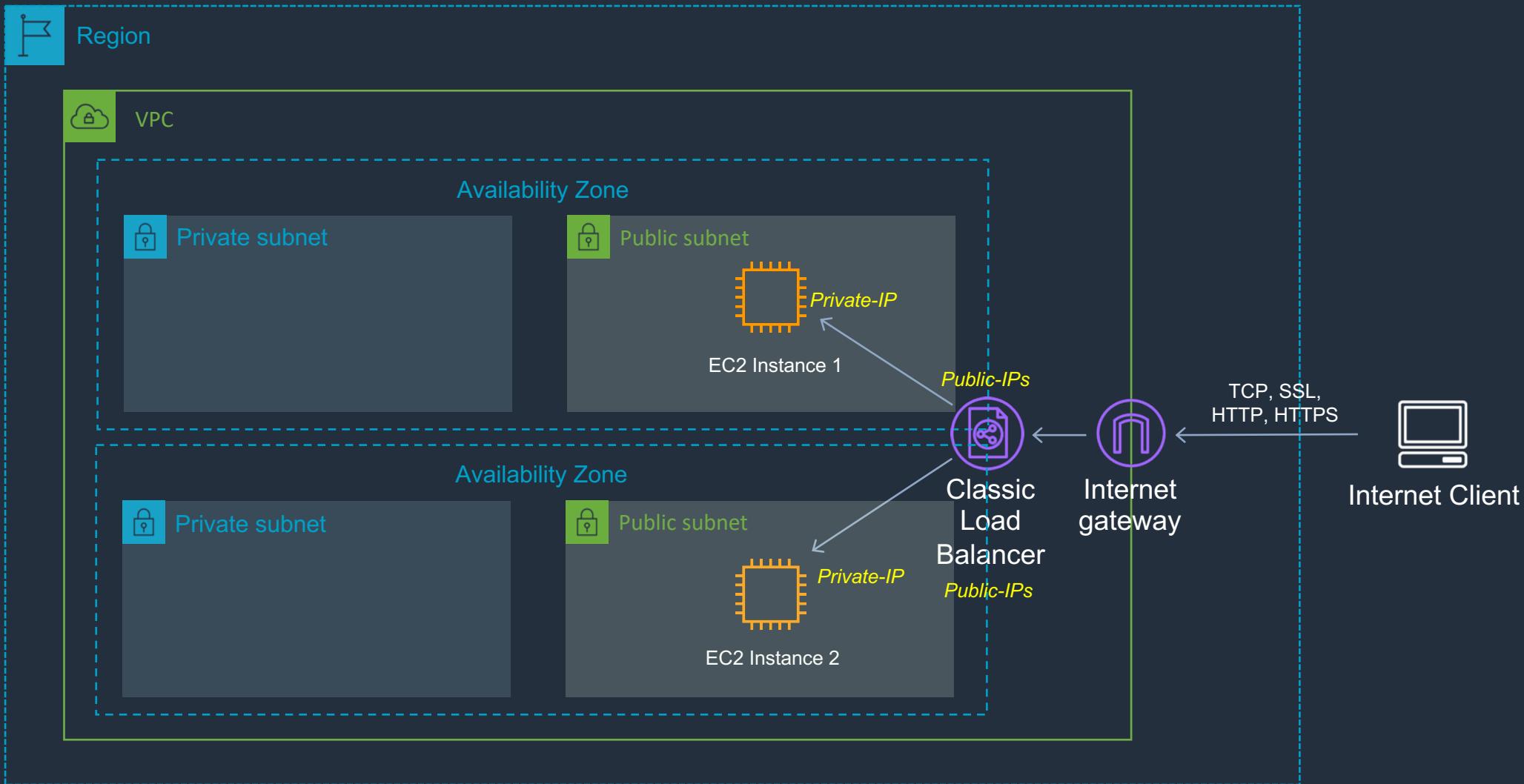
- Operates at the connection level
- Routes connections based on IP protocol data (layer 4)
- Offers ultra high performance, low latency and TLS offloading at scale
- Can have static IP / Elastic IP
- Supports UDP and static IP addresses as targets



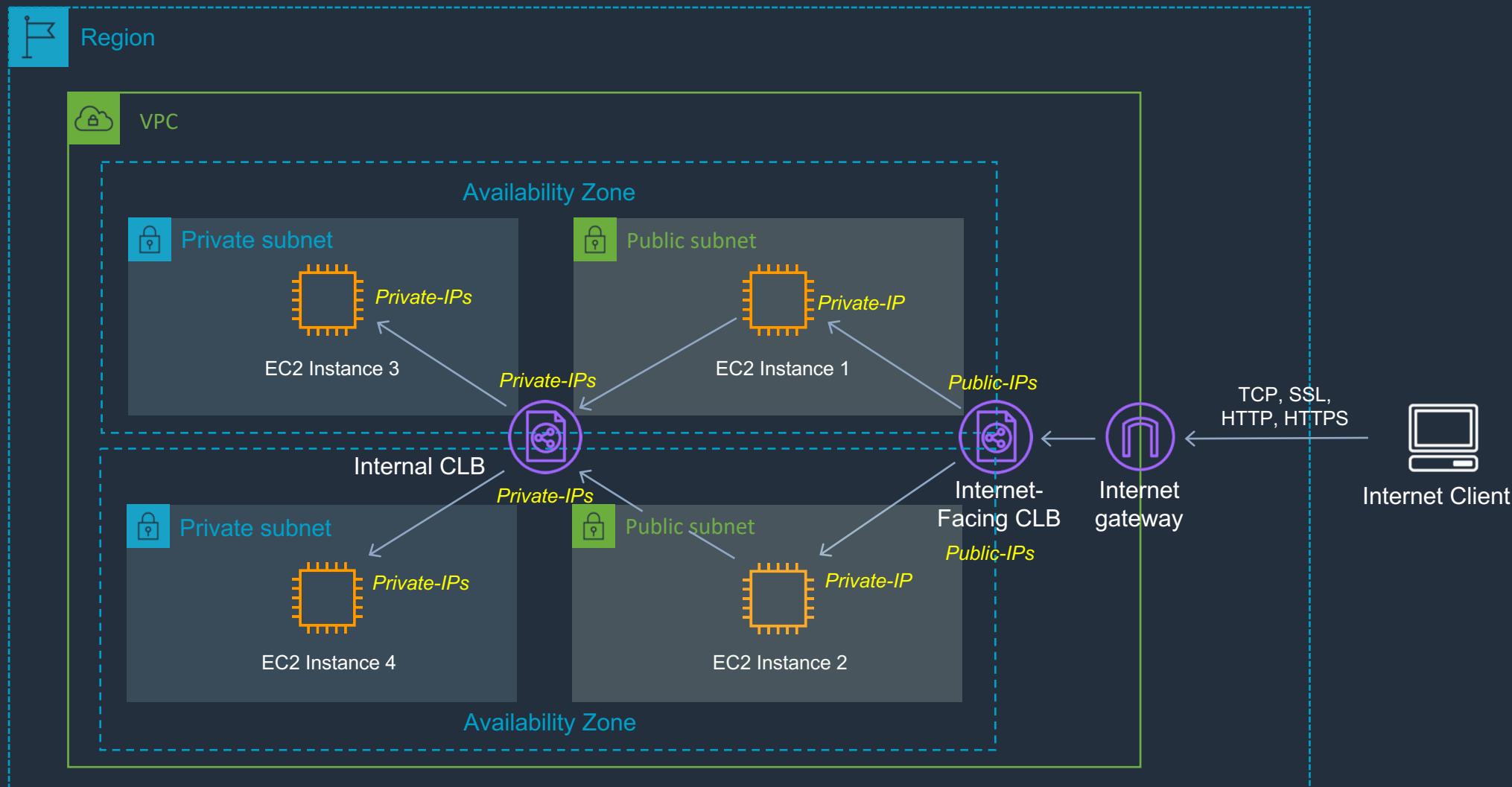
### Classic Load Balancer

- Old generation; not recommended for new applications
- Performs routing at Layer 4 and Layer 7
- Use for existing applications running in EC2-Classic

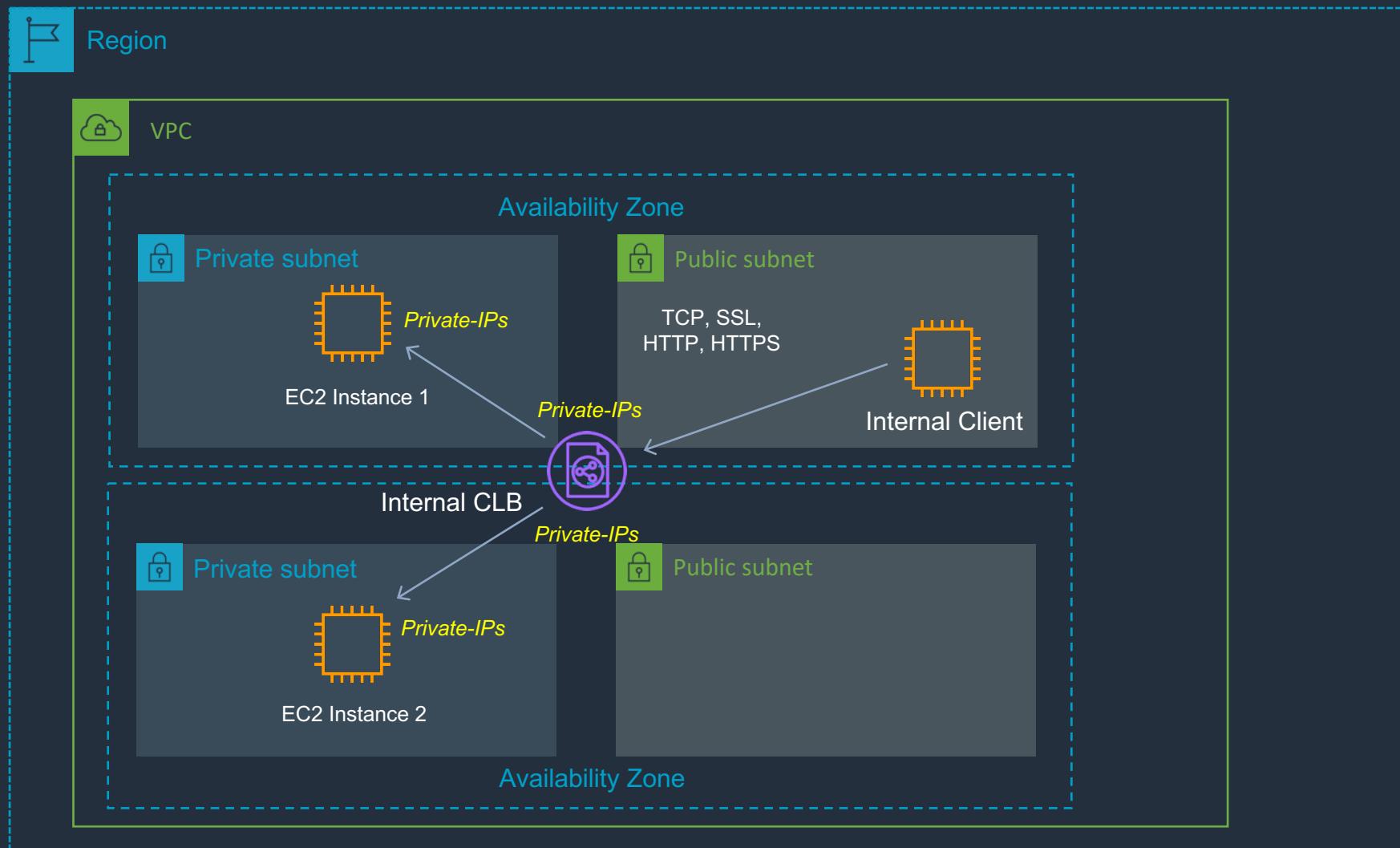
## Section 4: Classic Load Balancer (Internet-Facing)



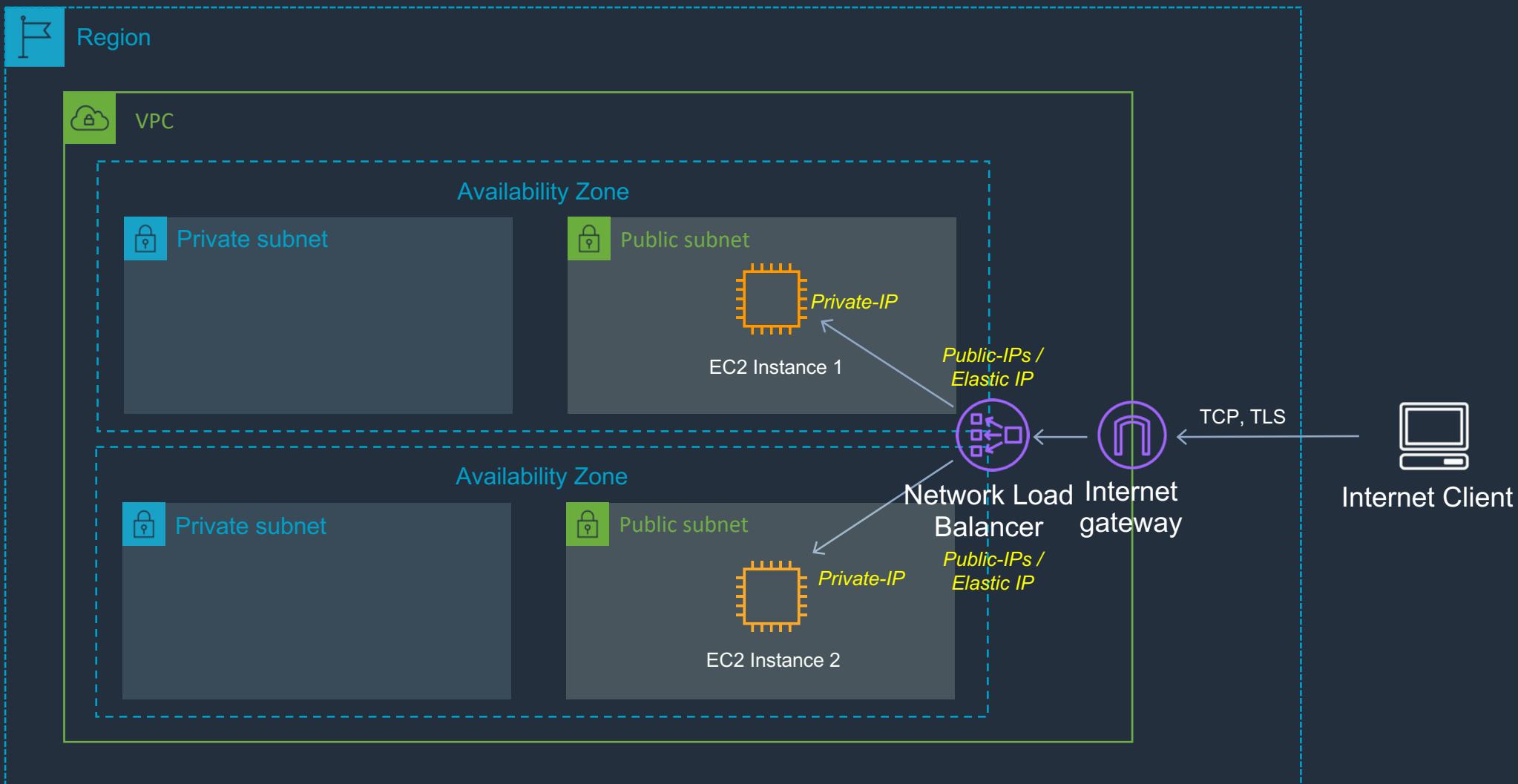
## Section 4: Classic Load Balancer - Multi-tier



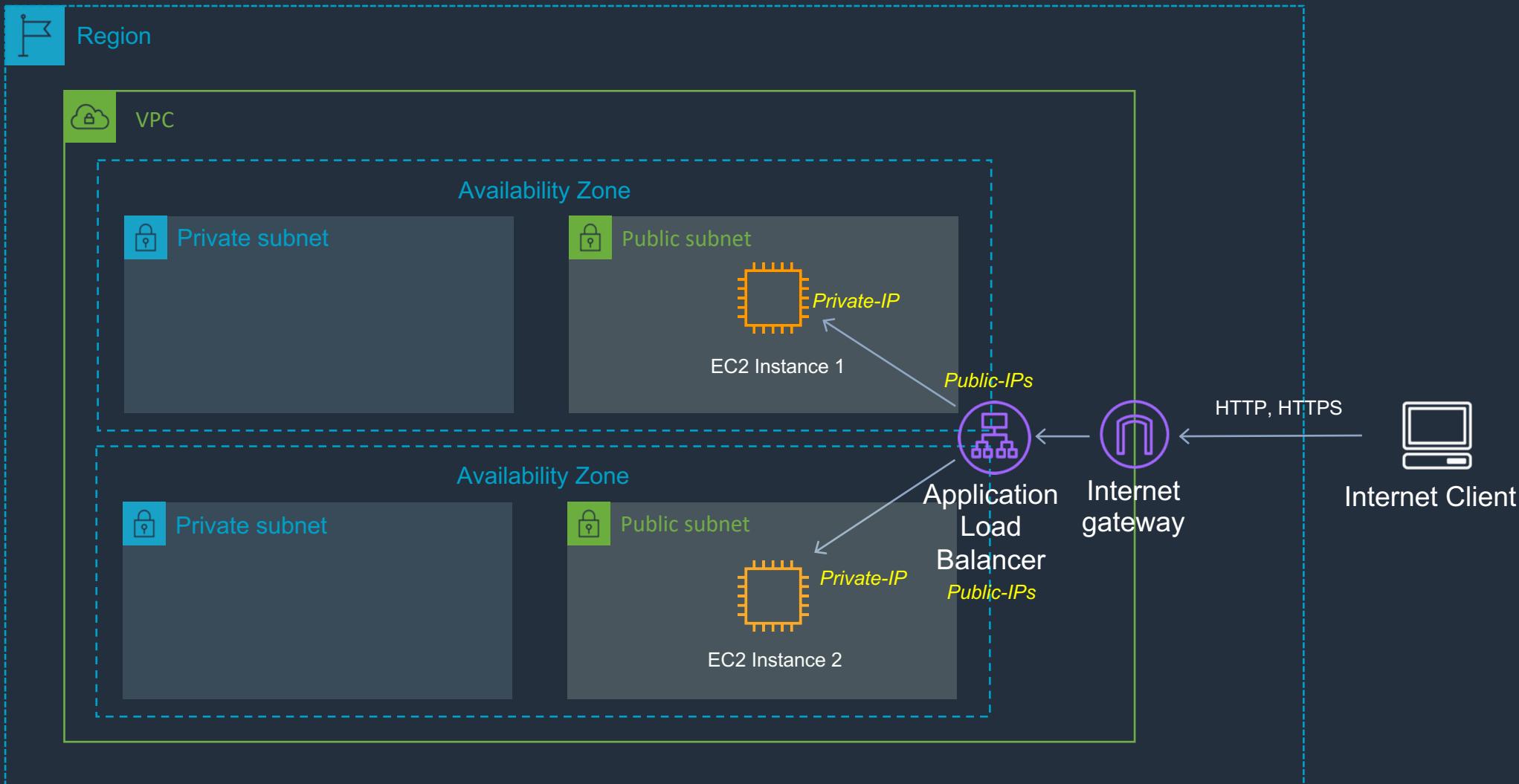
## Section 4: Classic Load Balancer (Internal)



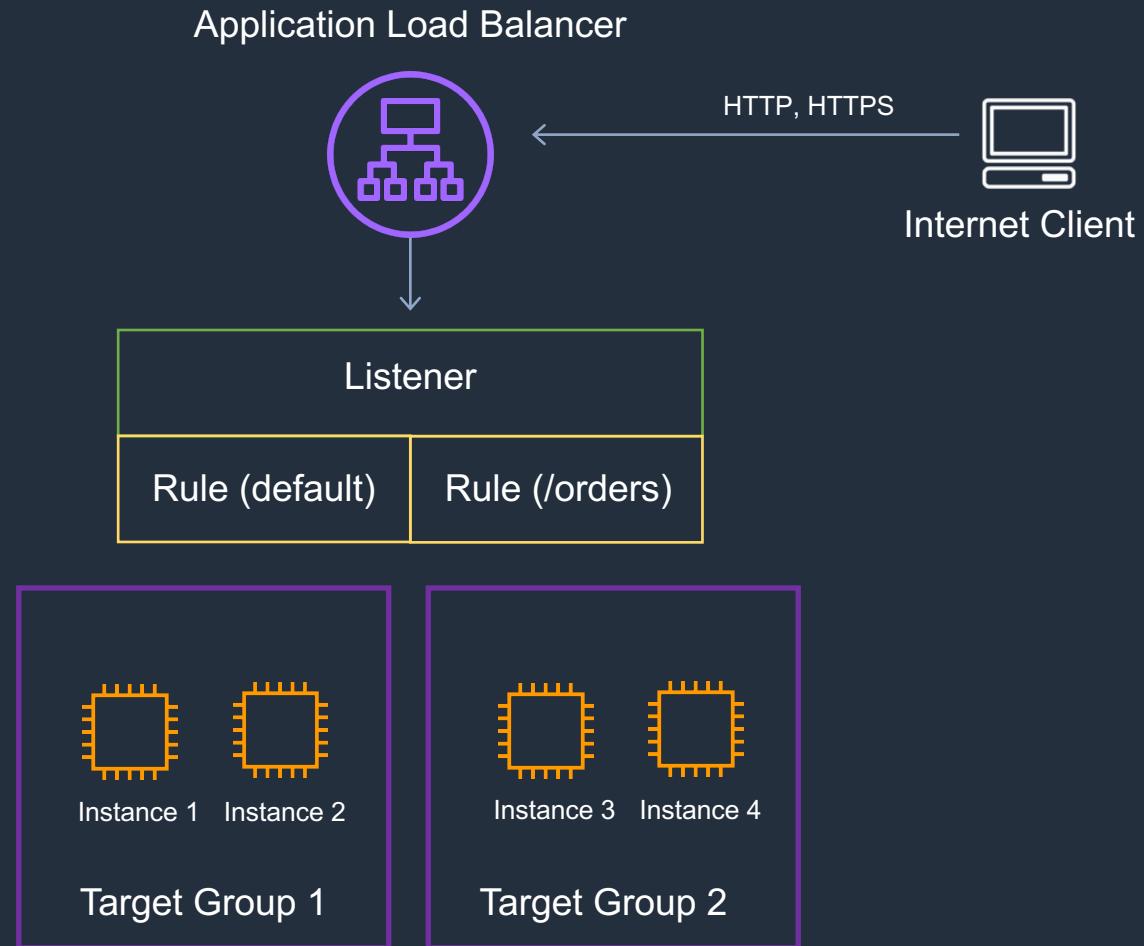
## Section 4: Network Load Balancer (Internet-Facing)



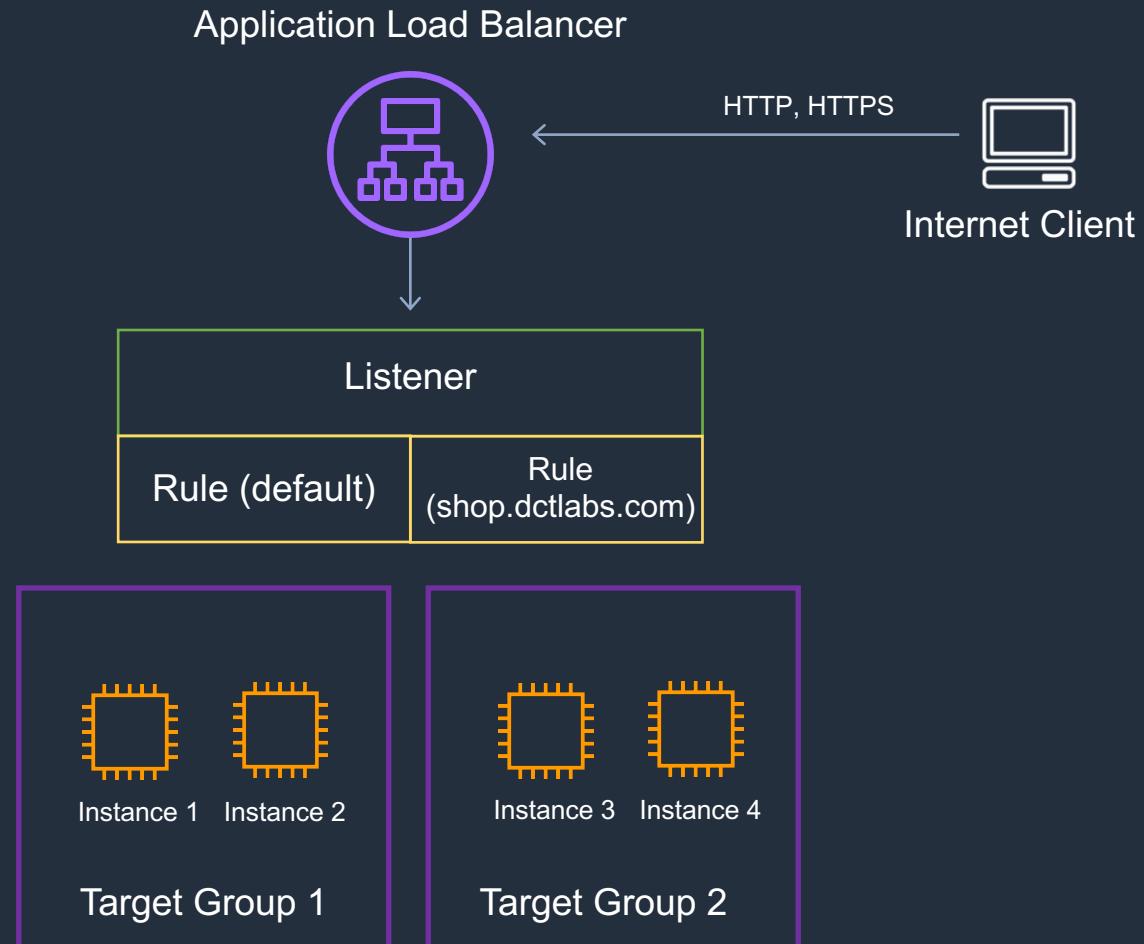
## Section 4: Application Load Balancer (Internet-Facing)



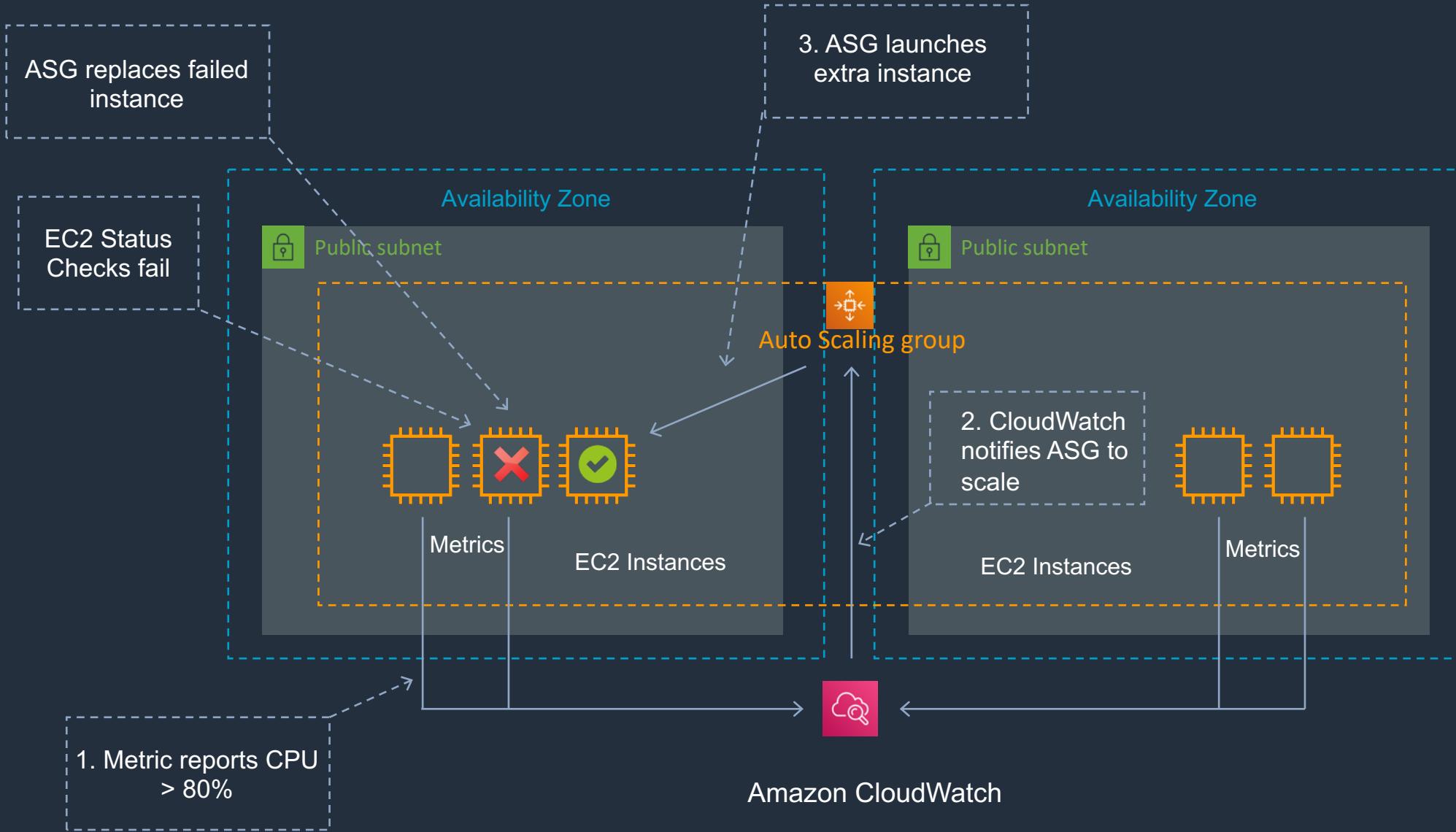
## Section 4: Application Load Balancer – Path-based Routing



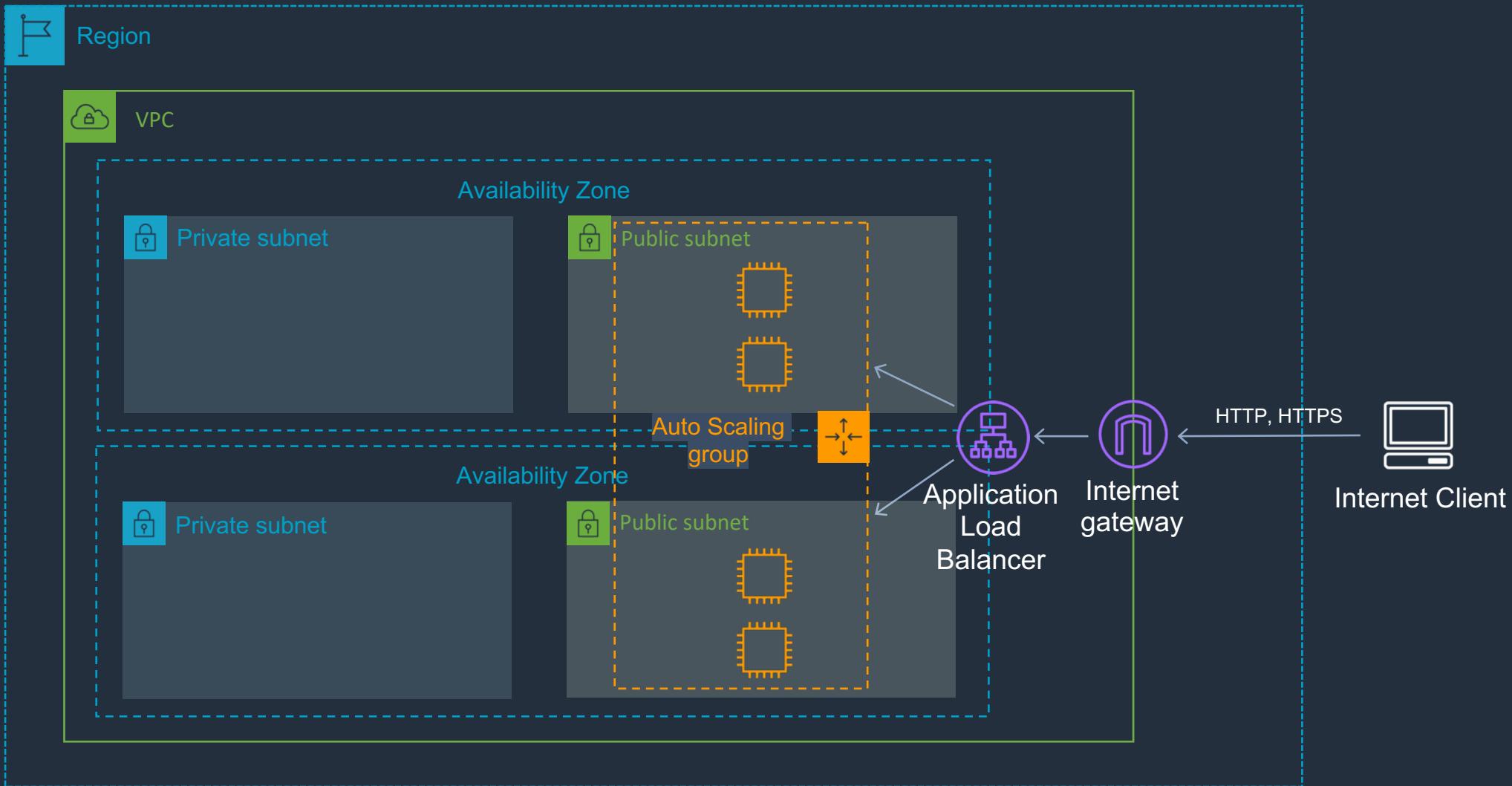
## Section 4: Application Load Balancer – Host-based Routing



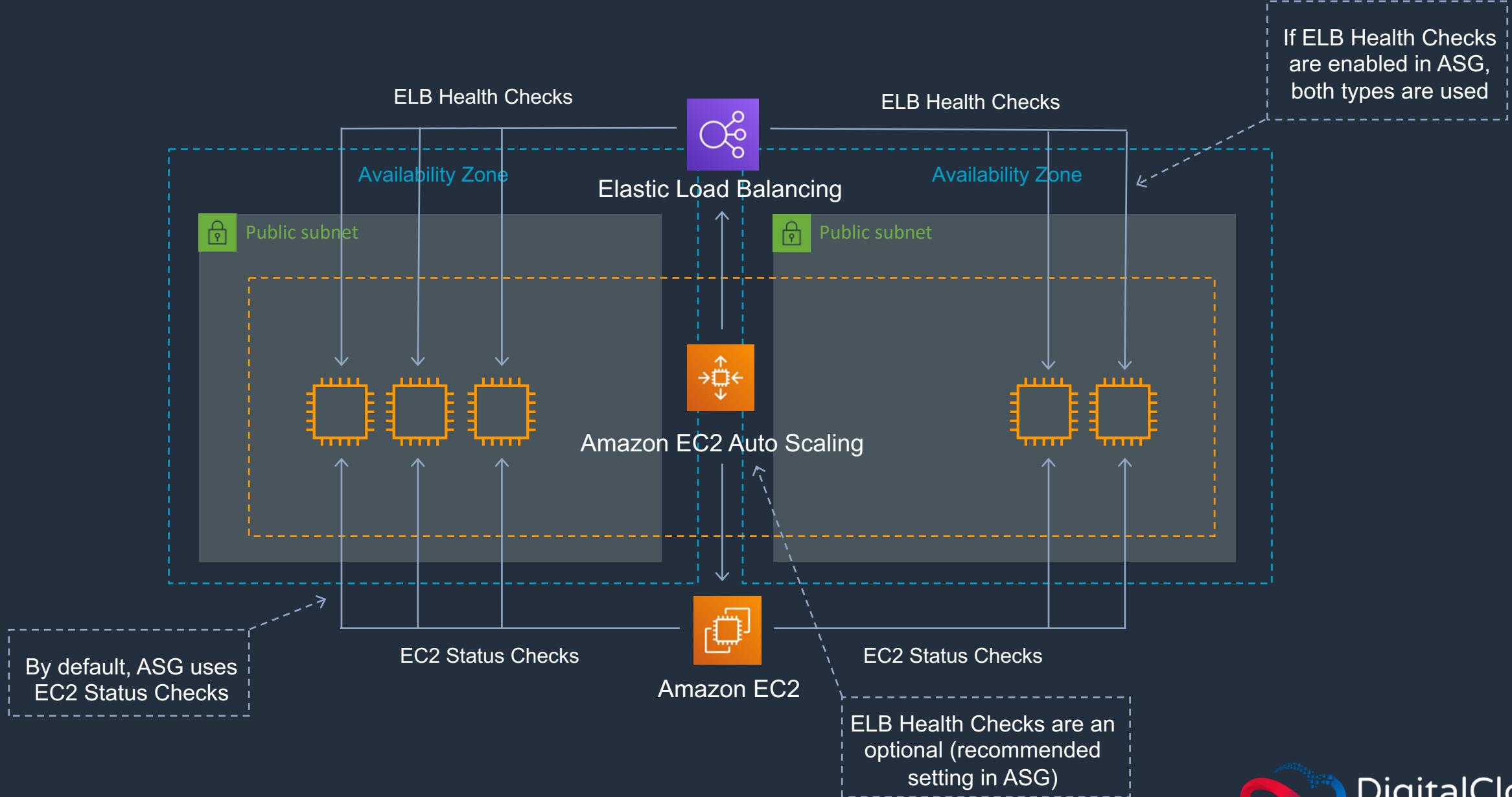
## Section 4: Auto Scaling Overview



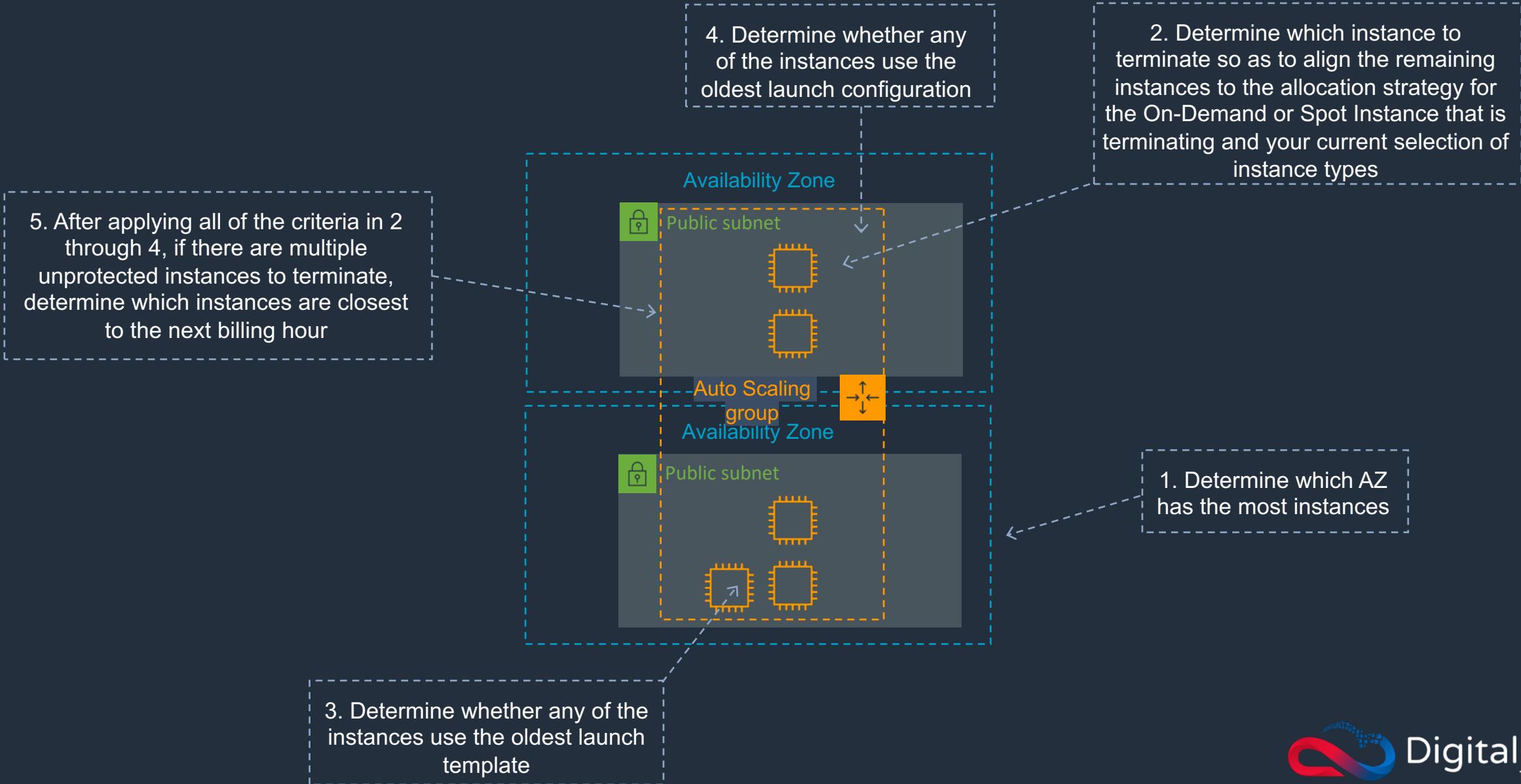
## Section 4: Auto Scaling Group with ALB



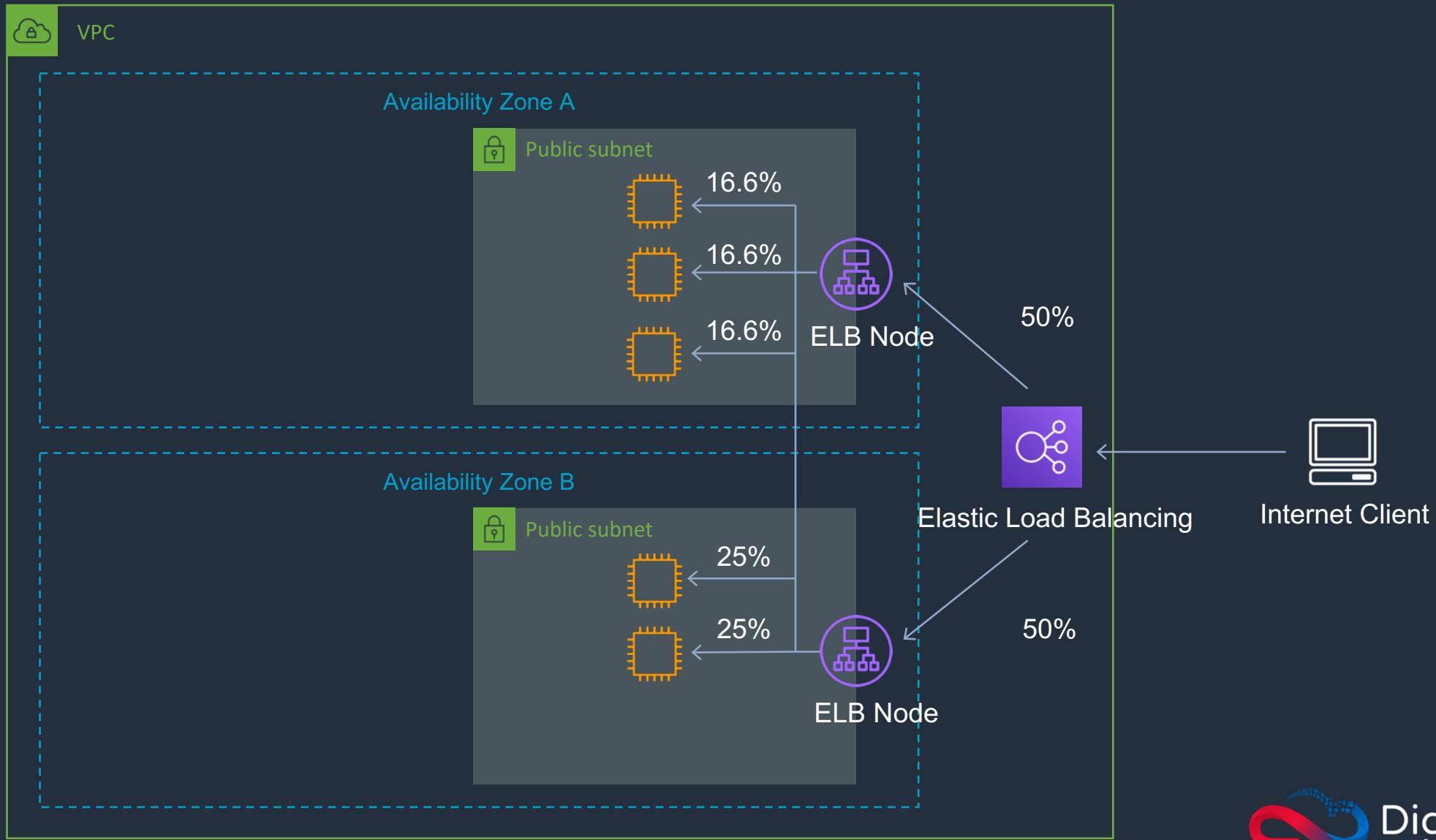
## Section 4: EC2 and ELB Health Checks



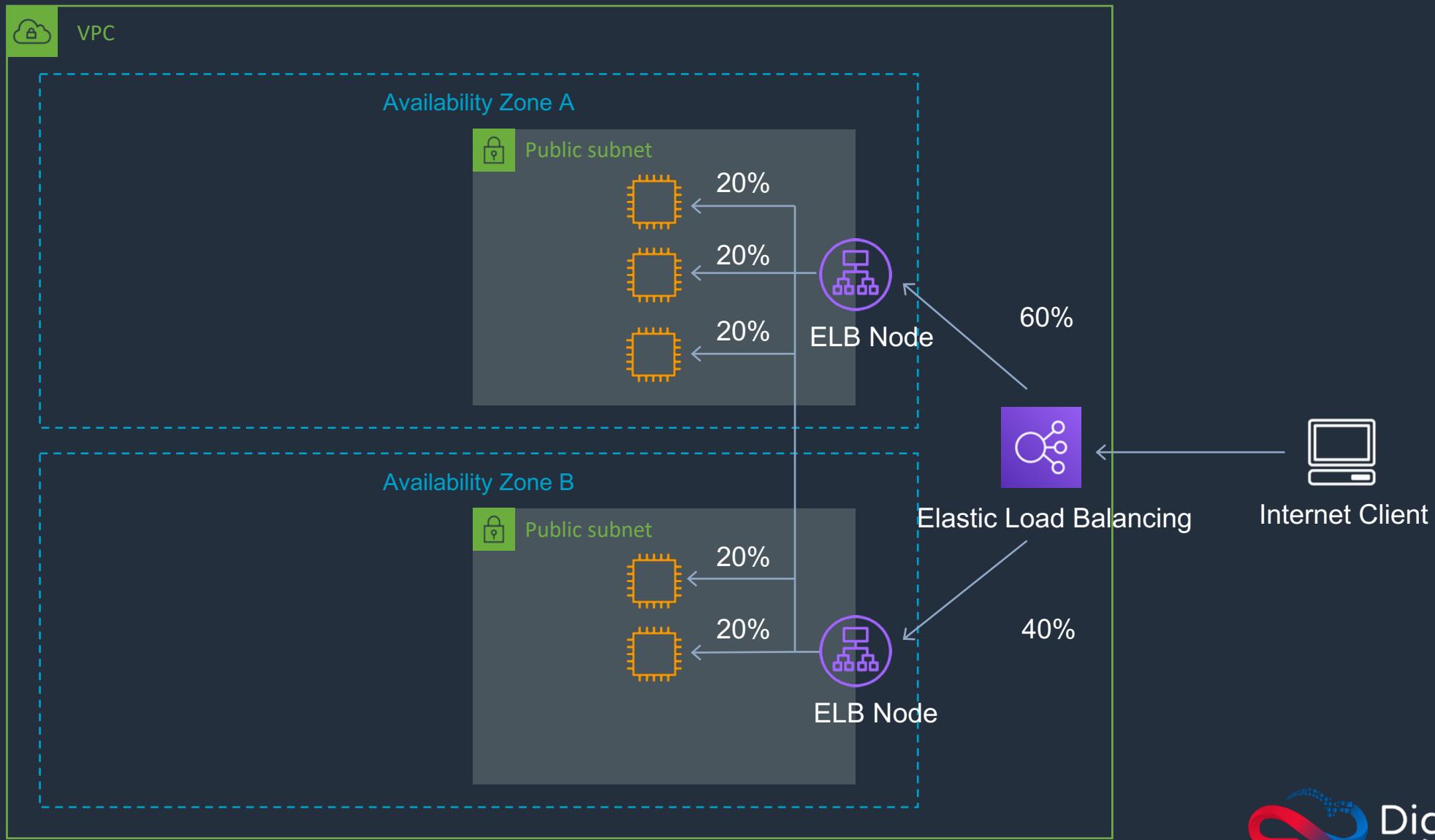
## Section 4: Auto Scaling Termination Policies – Default Policies



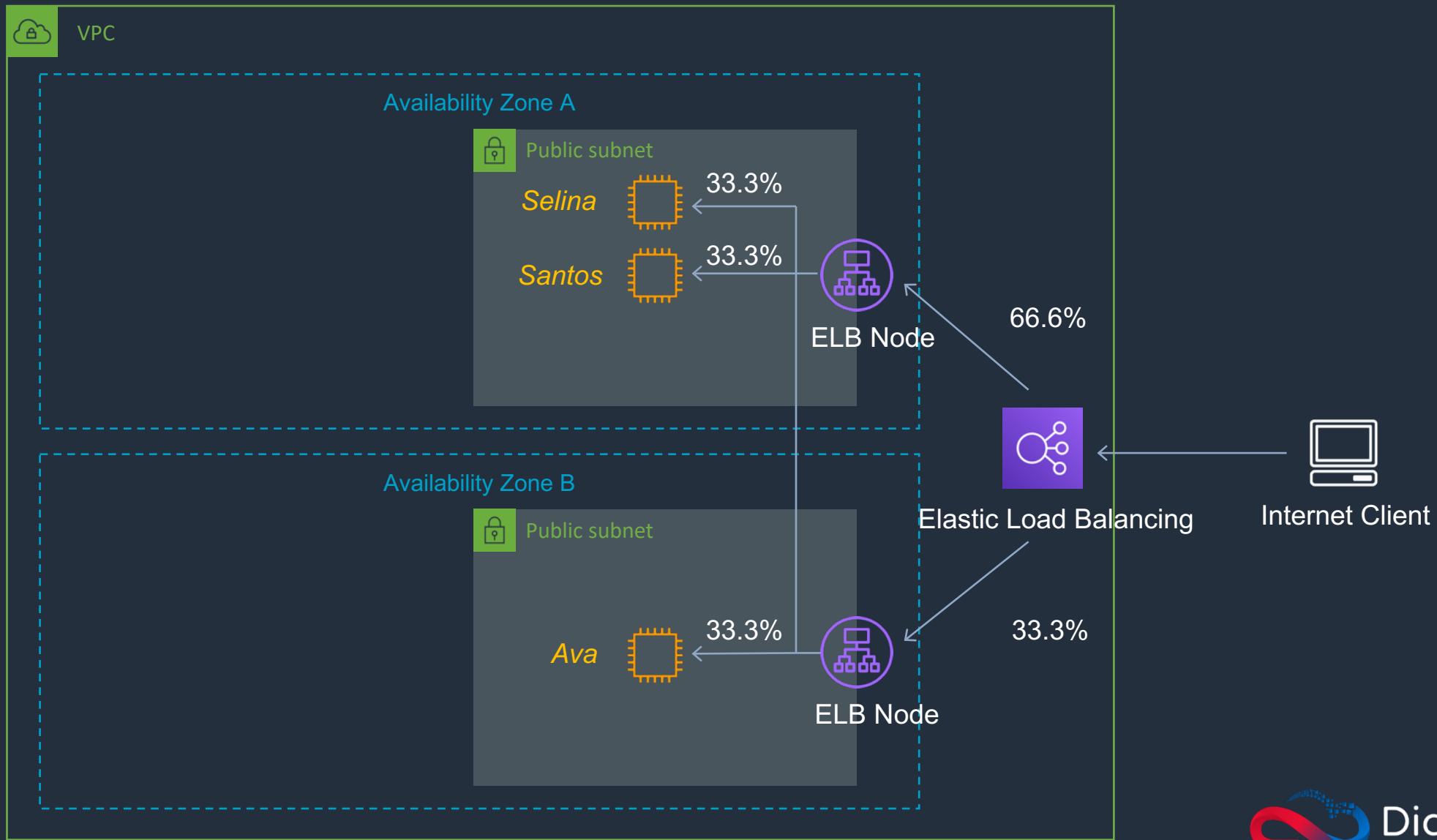
## Section 4: Cross-Zone Load Balancing - Disabled



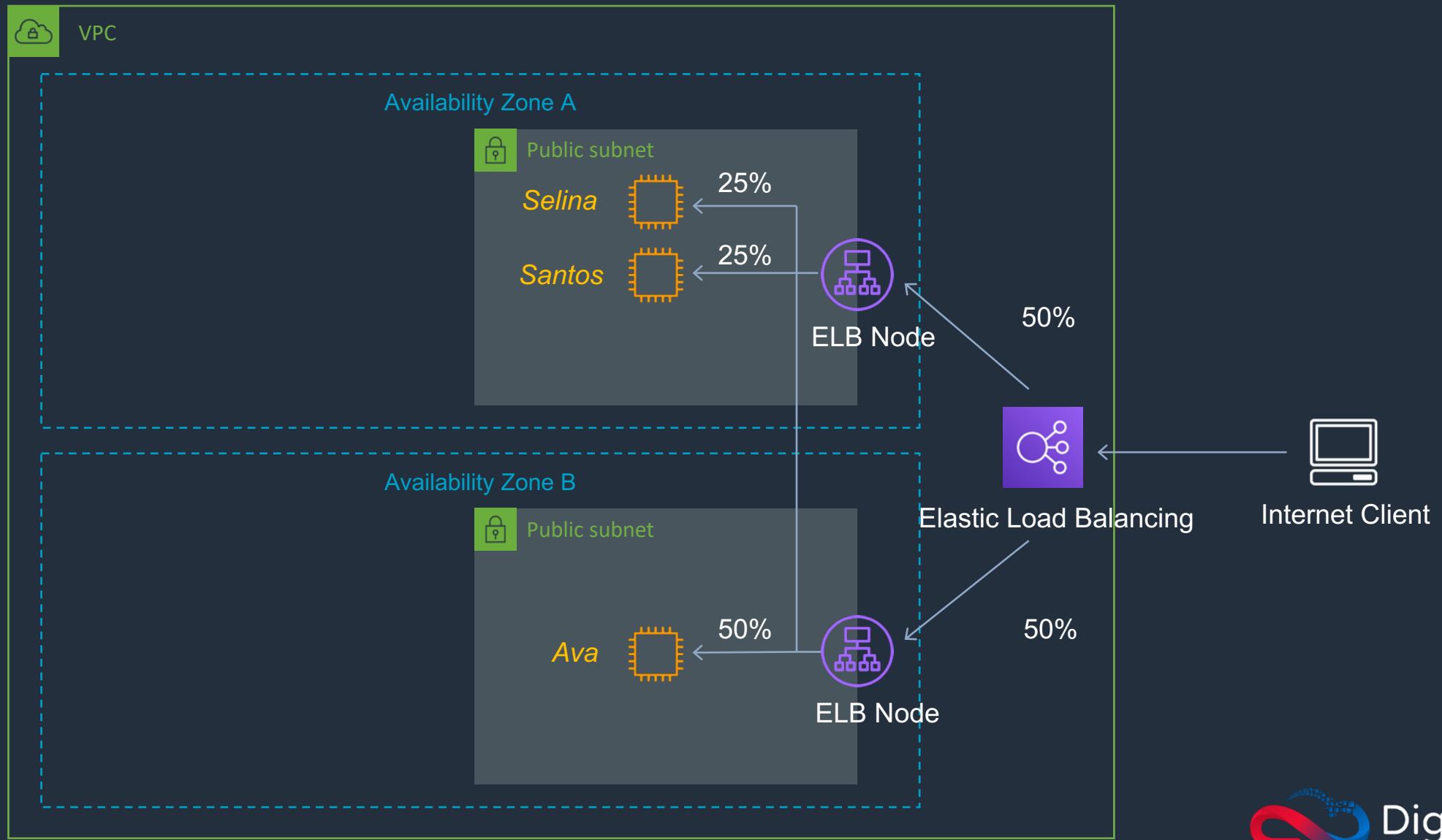
## Section 4: Cross-Zone Load Balancing - Enabled



## Section 4: Cross-Zone Load Balancing - Enabled



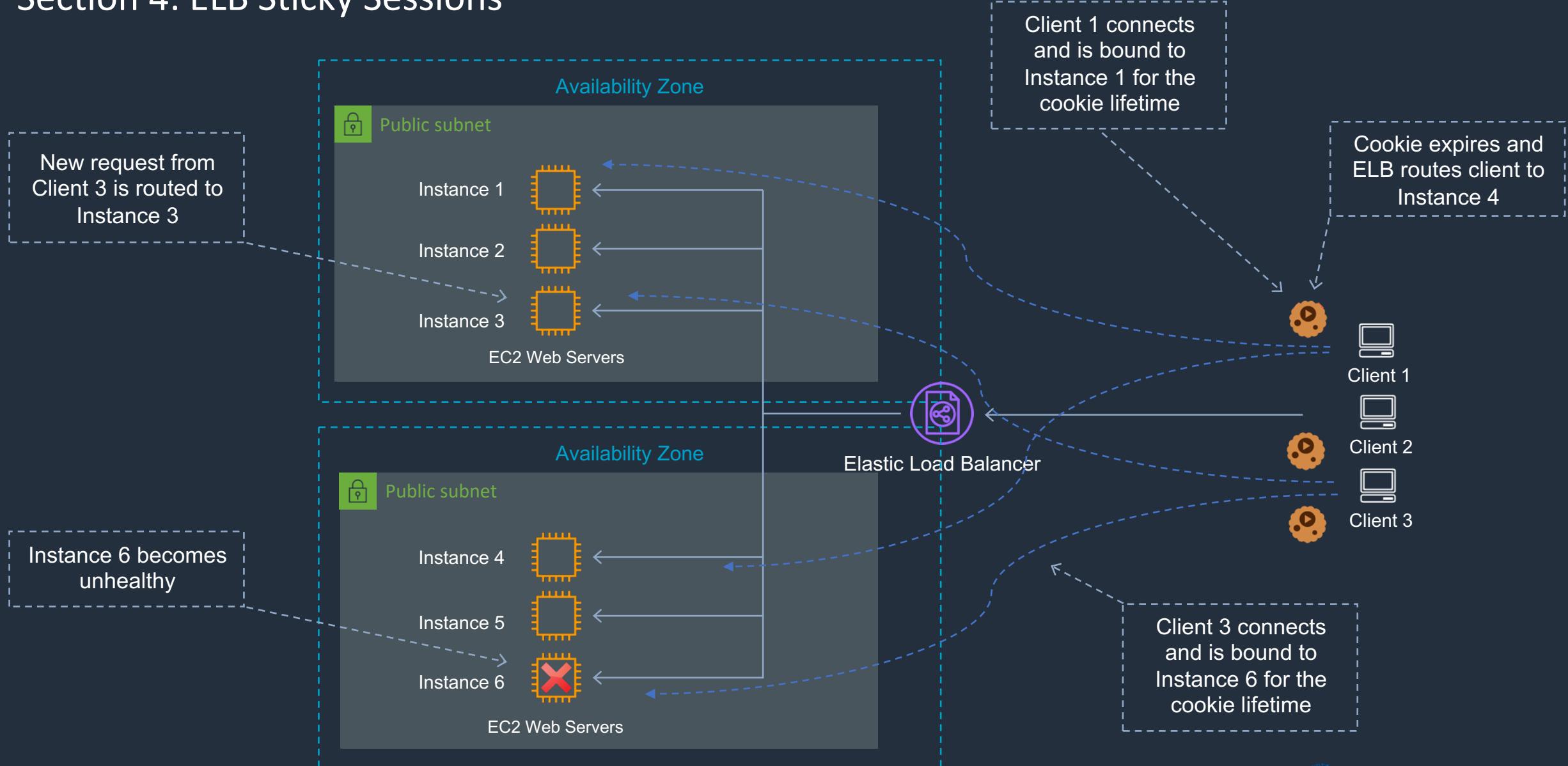
## Section 4: Cross-Zone Load Balancing - Disabled



## Section 4: Cross-Zone Load Balancing

Name	Created through Console	Created through CLI/API	Can be enabled/disabled?
ALB	Enabled	Enabled	No
NLB	Disabled	Disabled	Yes
CLB	Enabled	Disabled	Yes

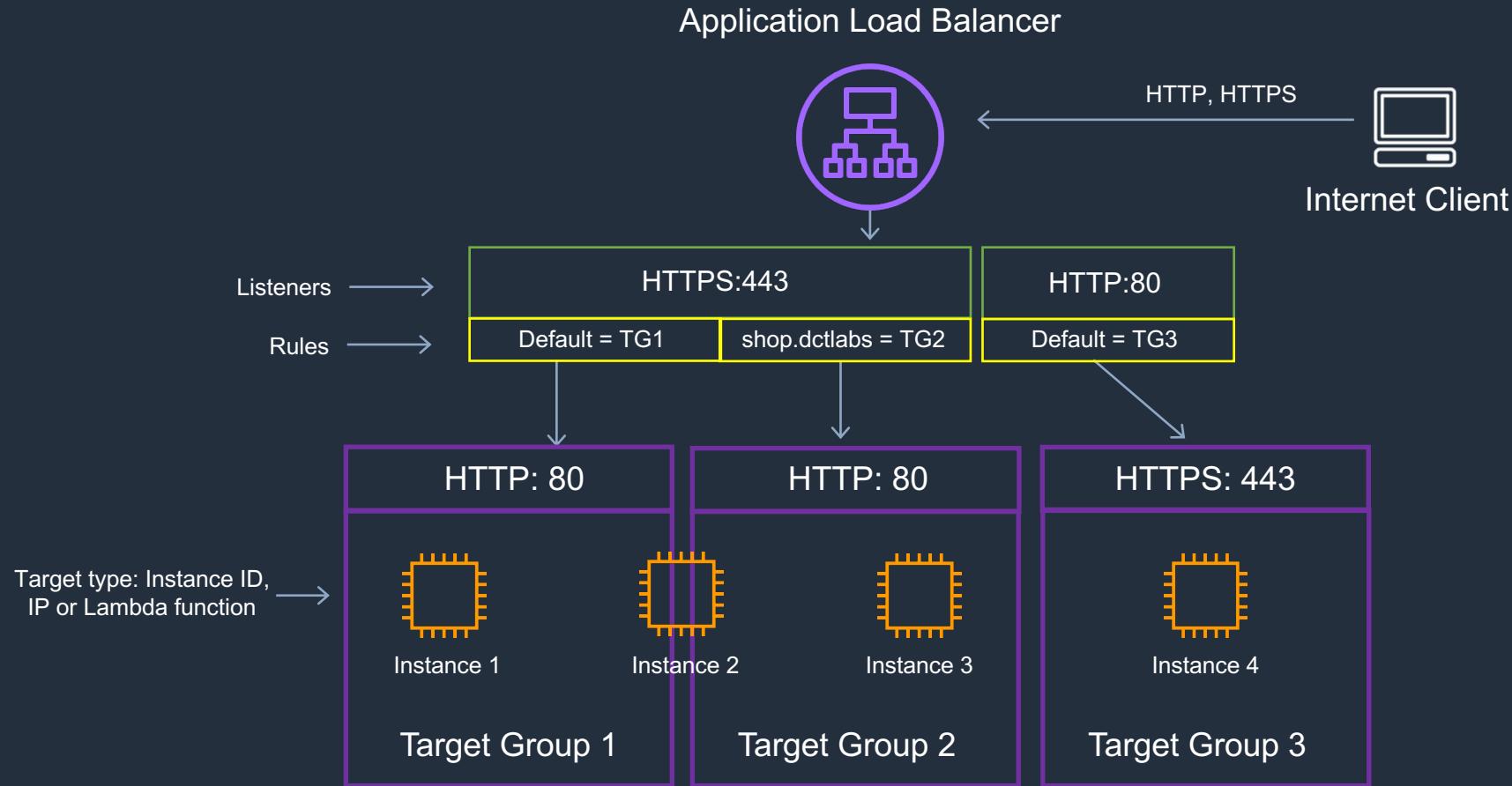
## Section 4: ELB Sticky Sessions



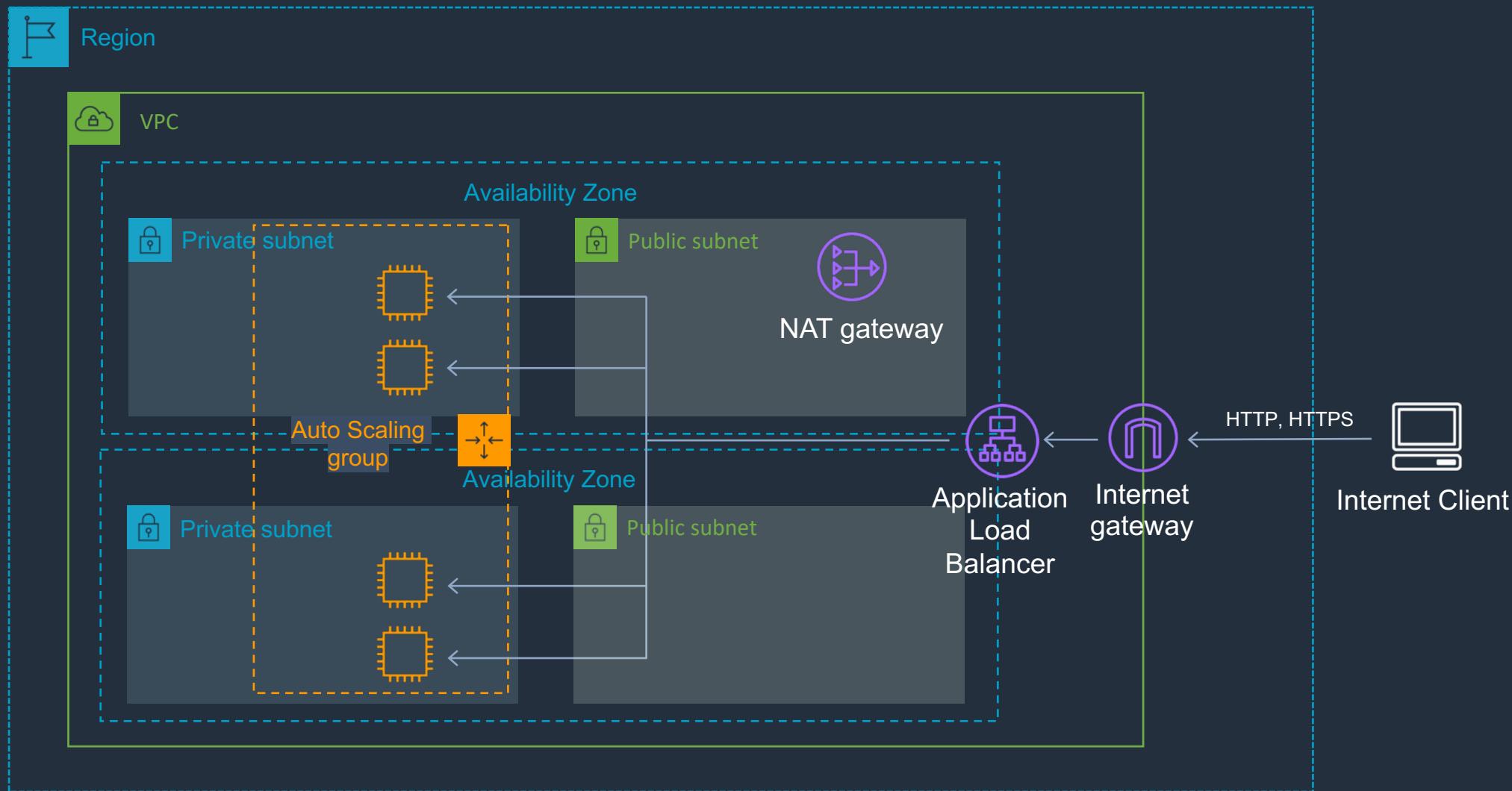
## Section 4: Sticky Sessions

Name	Supported?	Load Balancer Generated Cookie	Application Generated Cookie
ALB	Yes	Yes, "AWSALB"	Not supported
NLB	No	N/A	N/A
CLB	Yes	Yes	Yes

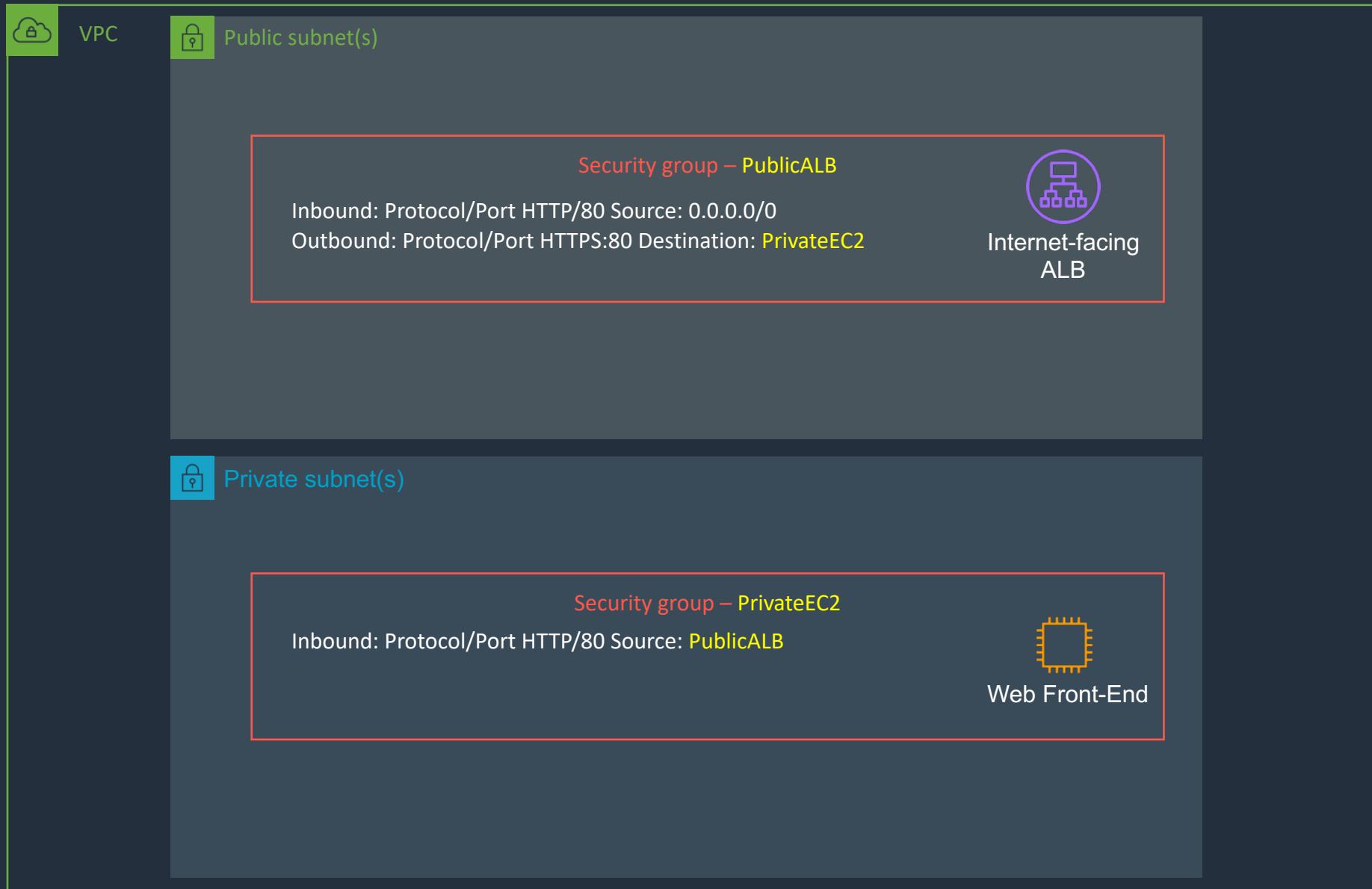
## Section 4: Application Load Balancer – Listeners and SSL/TLS



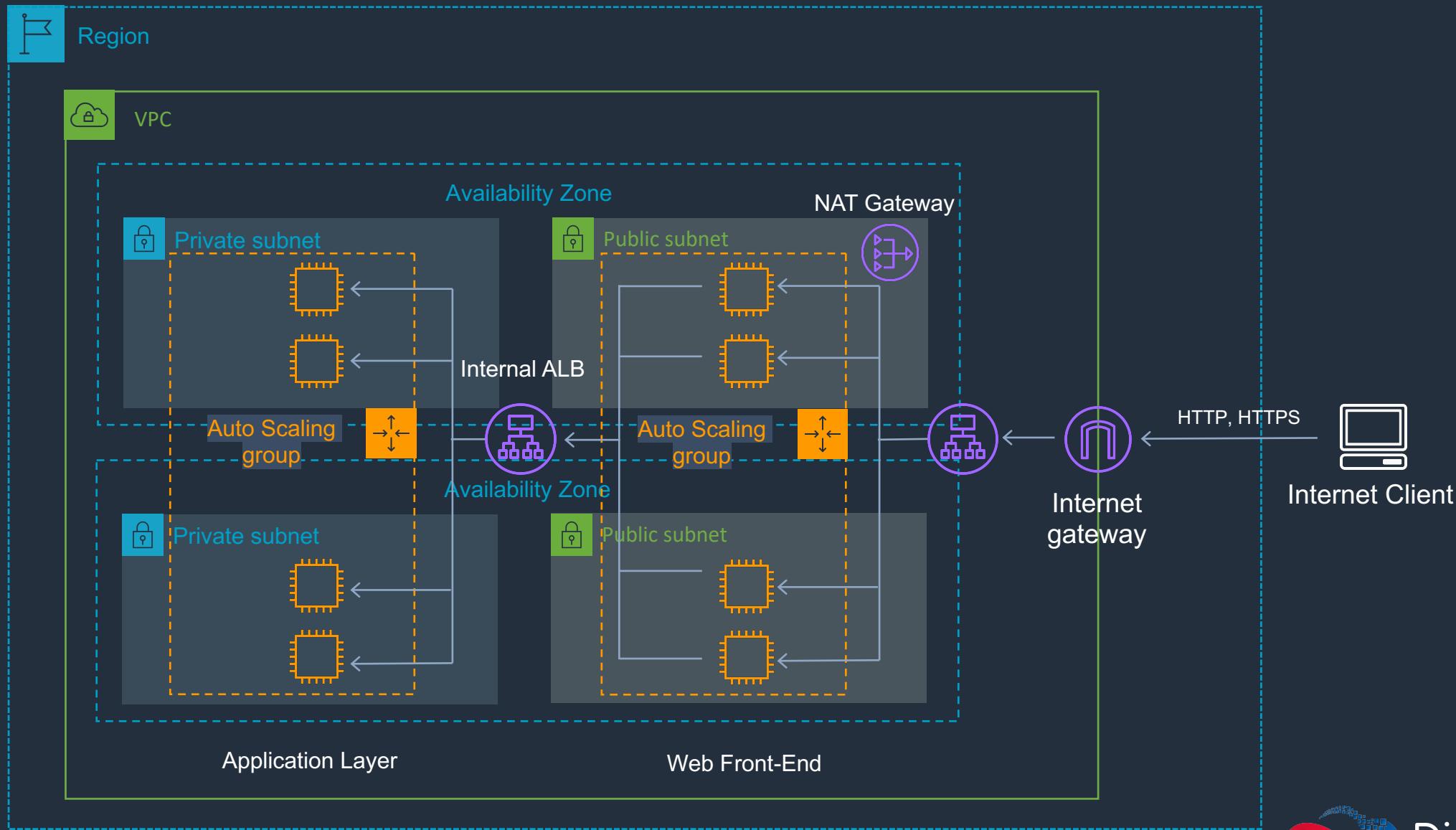
## Section 4: Public ALB with Private Instances



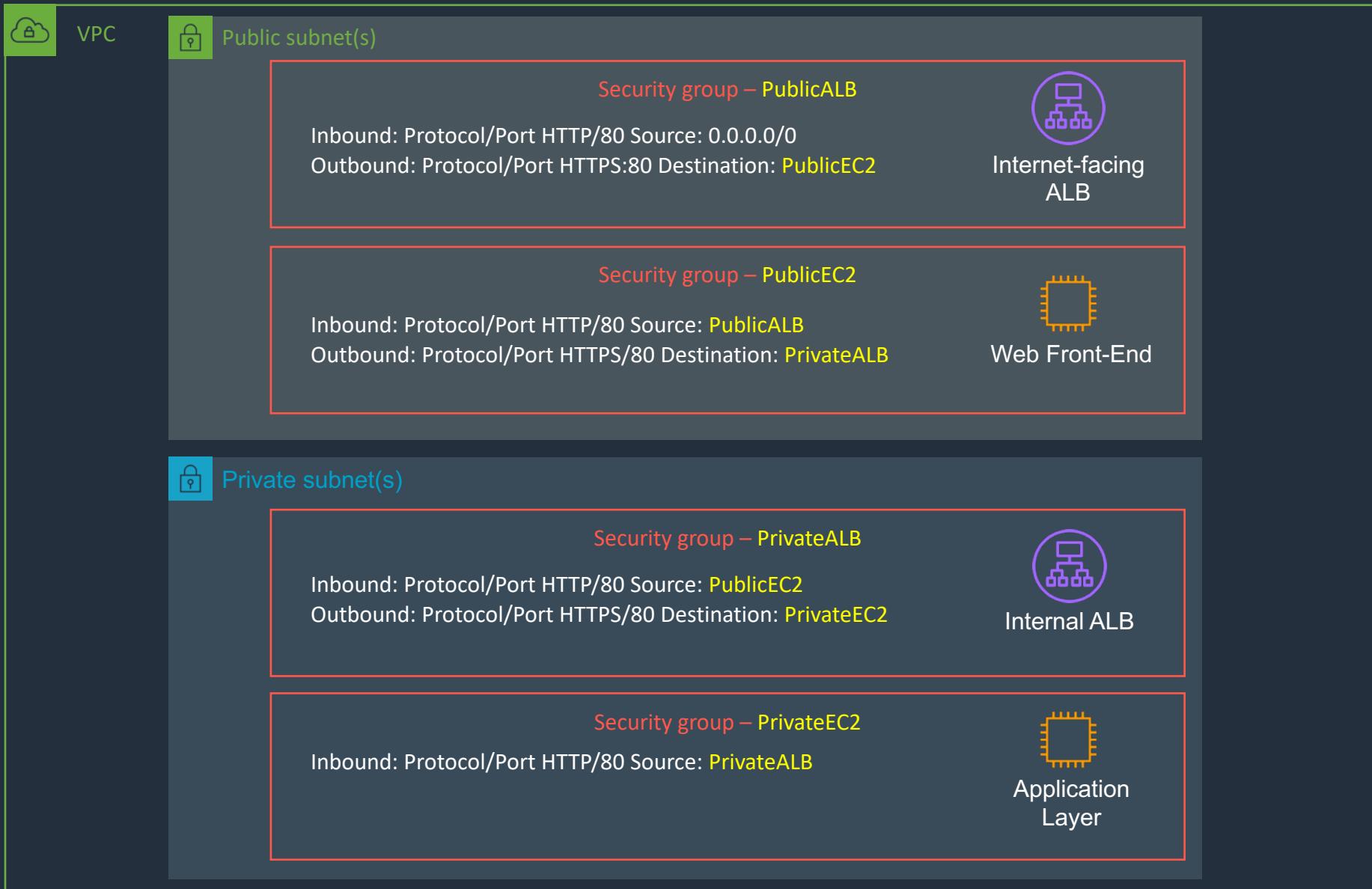
## Section 4: Public ALB with Private Instances– Security Groups



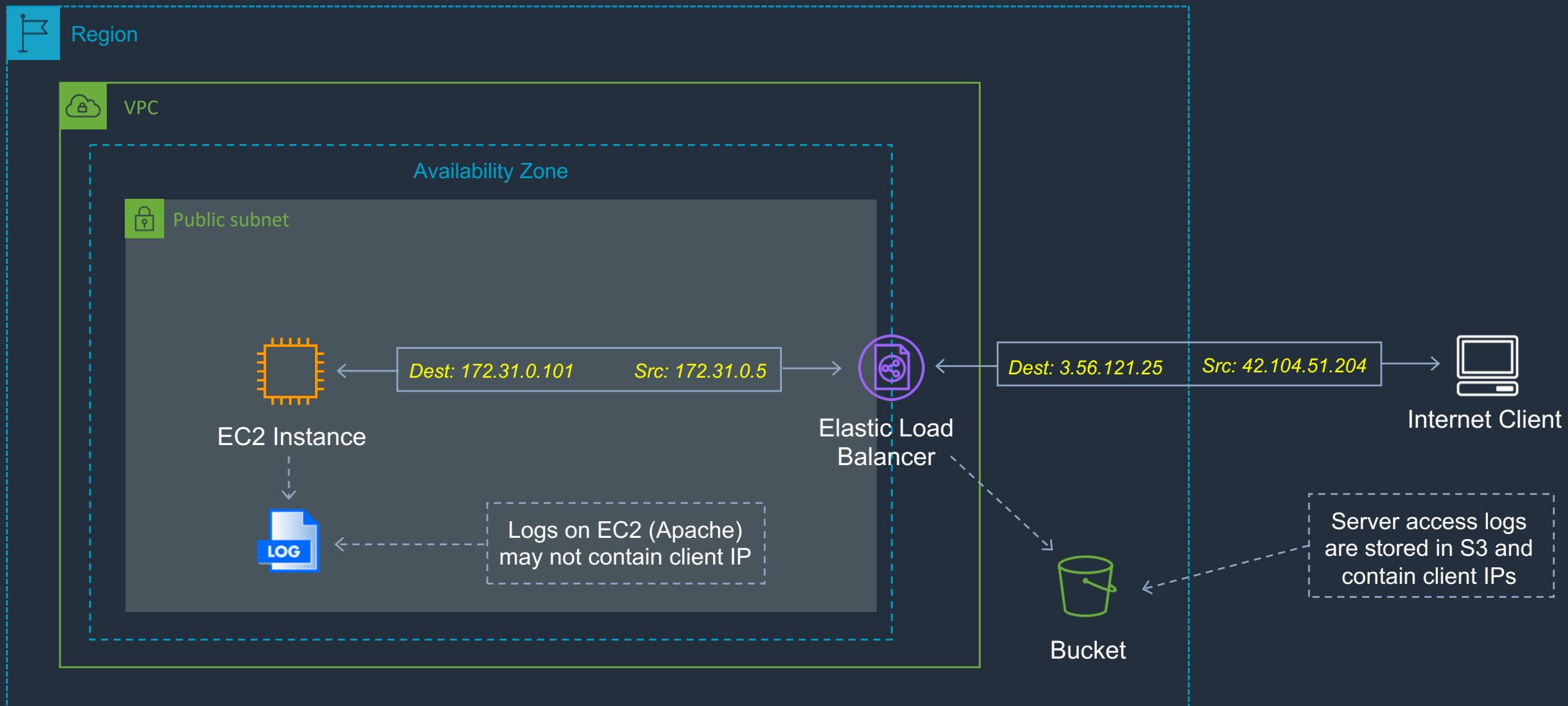
## Section 4: Multi-Tier Web Architecture



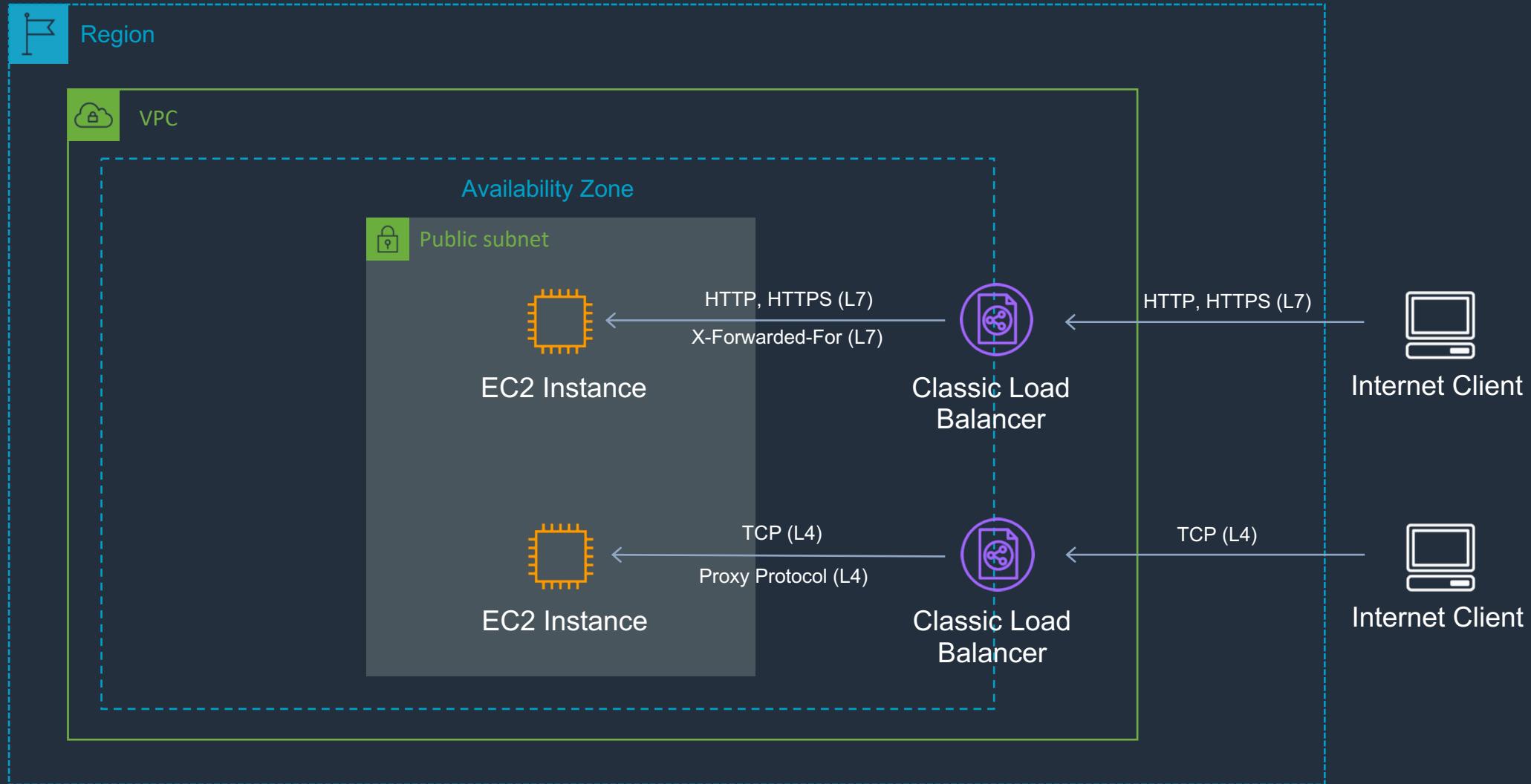
## Section 4: Multi-Tier Web Architecture – Security Groups



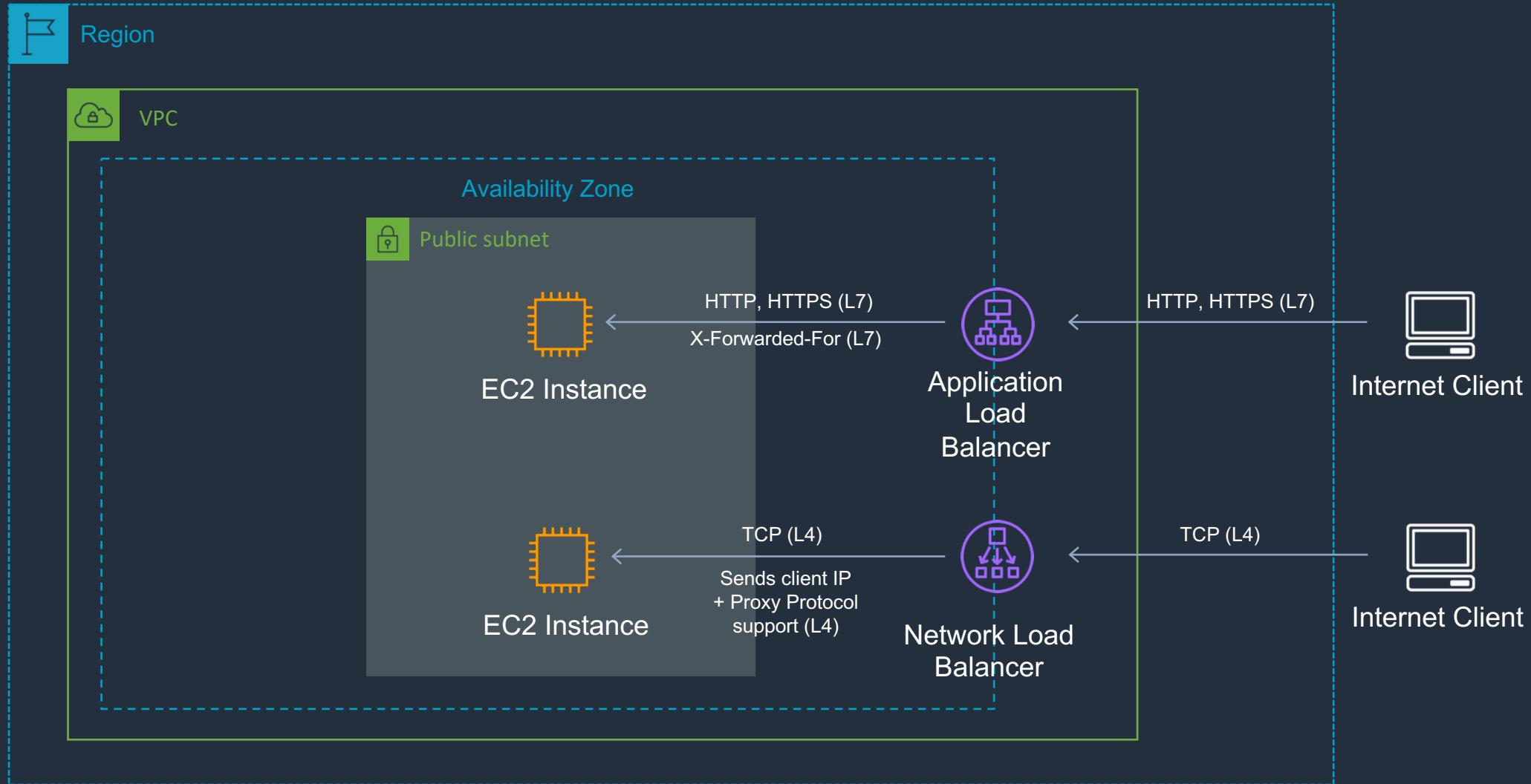
## Section 4: ELB Connections and Logging



## Section 4: CLB - Proxy Protocol and X-Forwarded-For



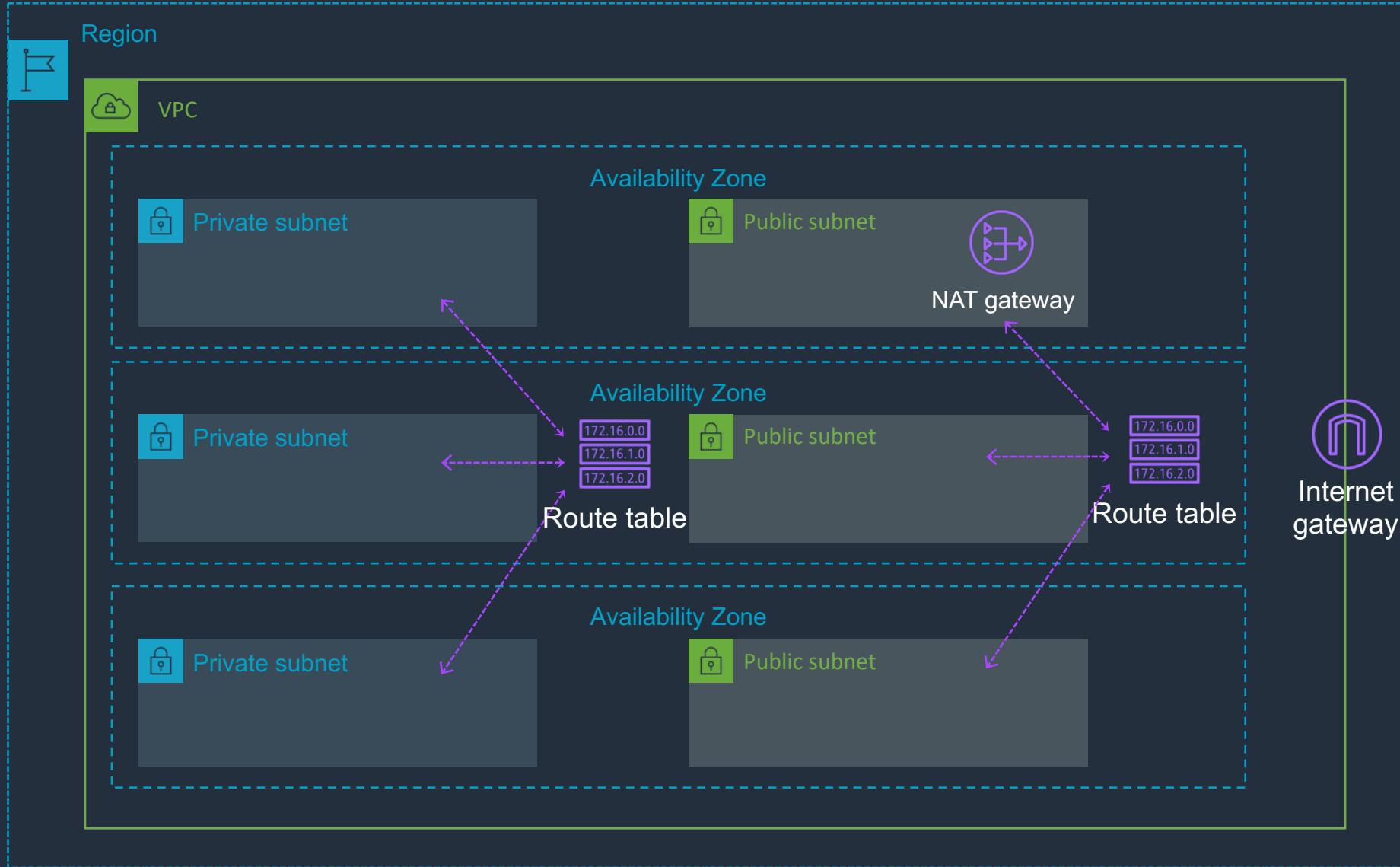
## Section 4: ALB/NLB - Proxy Protocol, X-Forwarded-For and Access Logging



# SECTION 5

## Virtual Private Cloud (VPC)

## Section 5: Creating a Custom VPC



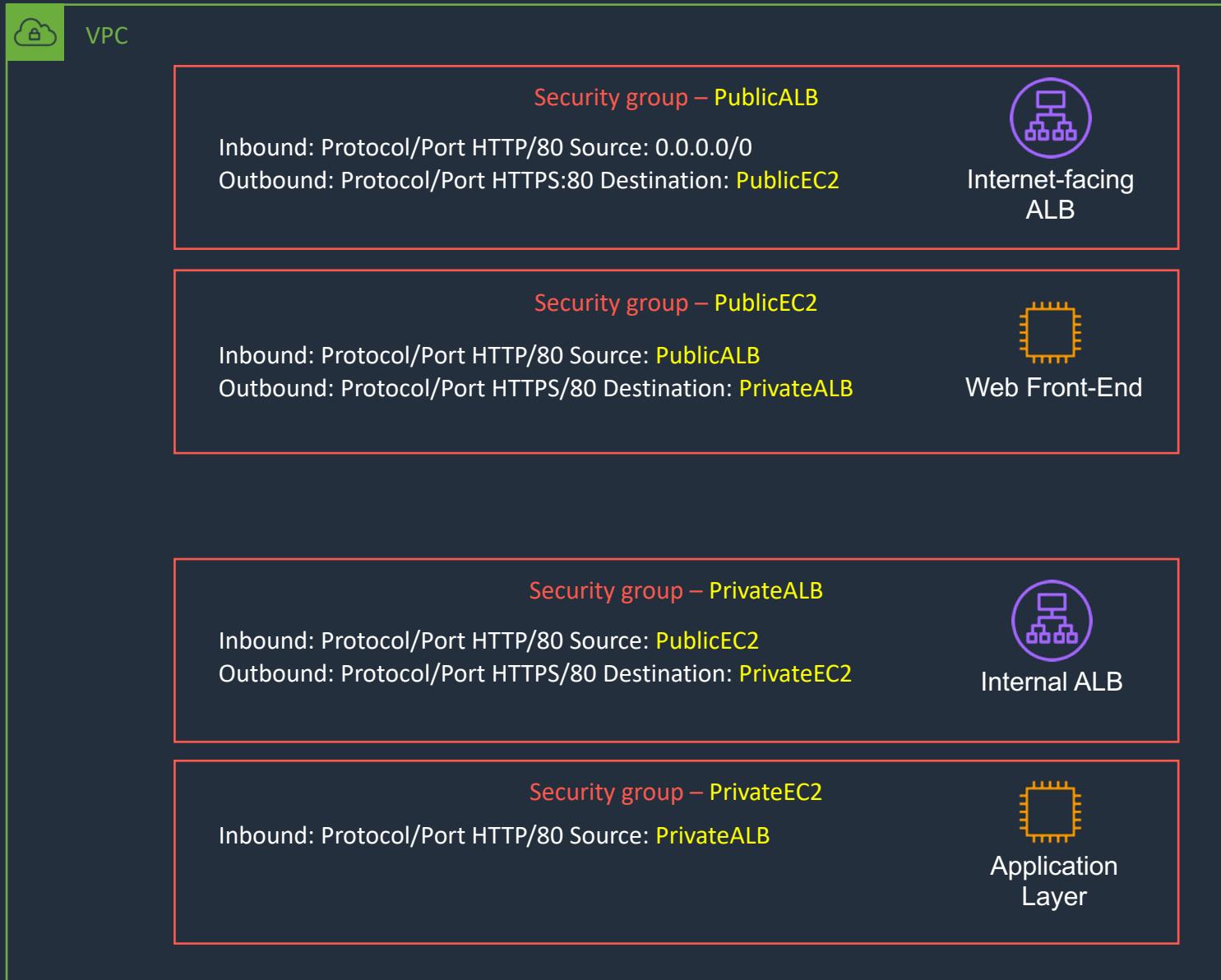
Public Route Table

Destination	Target
10.0.0.0/16	Local
0.0.0.0/0	<i>igw-id</i>

Private Route Table

Destination	Target
10.0.0.0/16	Local
0.0.0.0/0	<i>nat-gateway-id</i>

# Section 5: Security Groups



## Default Security Group

### Inbound:

Source	Protocol	Port
Security Group ID	All	All

### Outbound:

Destination	Protocol	Port
0.0.0.0/0	All	All
::/0	All	All

## Custom Security Group

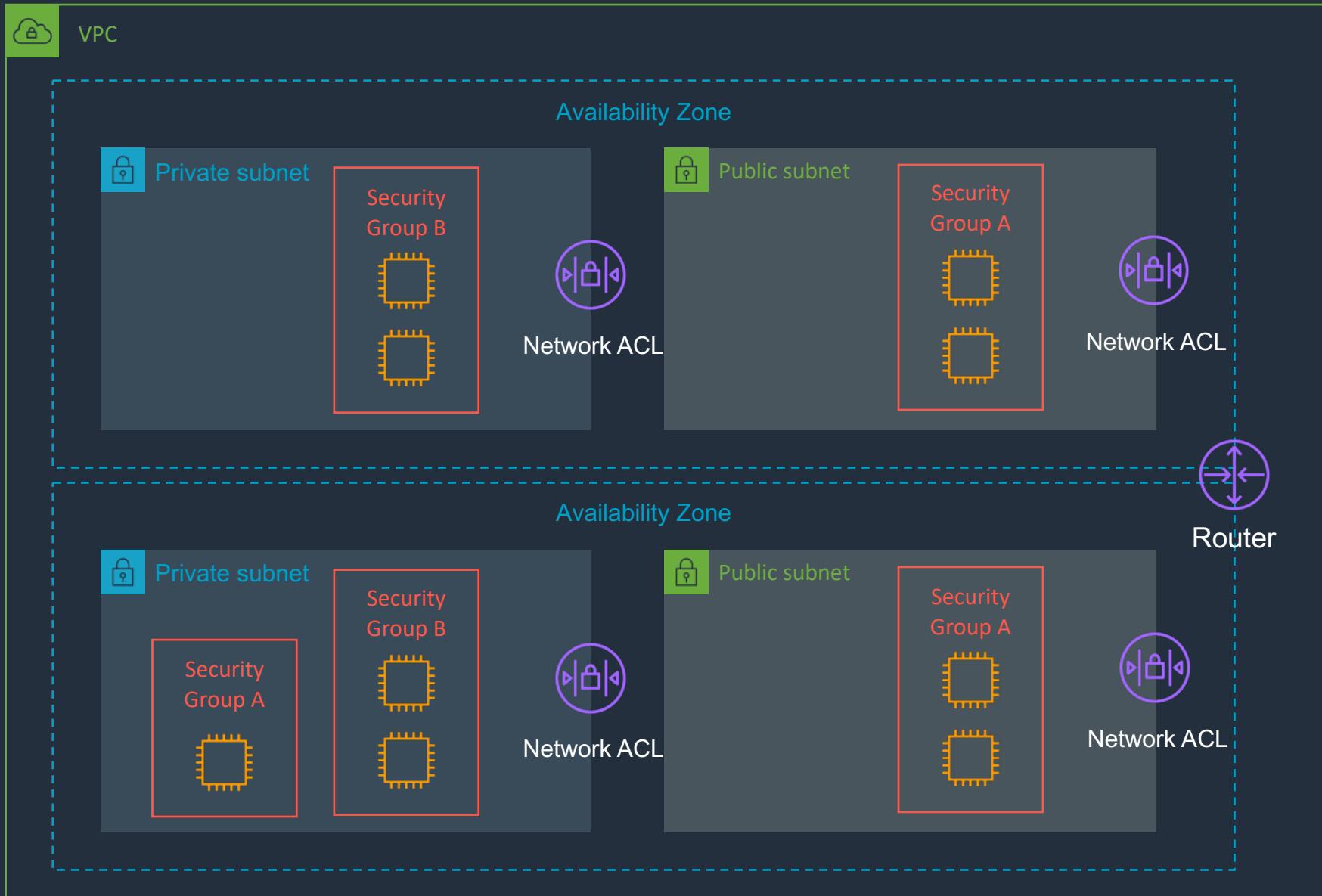
### Inbound:

Source	Protocol	Port

### Outbound:

Destination	Protocol	Port
0.0.0.0/0	All	All
::/0	All	All

# Section 5: Network Access Control Lists (NACLs)



## Default NACL

Inbound:

Protocol	Port	Source	Action
All	All	0.0.0.0/0	ALLOW
All	All	::/0	ALLOW

Outbound:

Protocol	Port	Source	Action
All	All	0.0.0.0/0	ALLOW
All	All	::/0	ALLOW

## Custom NACL

Inbound:

Protocol	Port	Source	Action
All	All	0.0.0.0/0	DENY
All	All	::/0	DENY

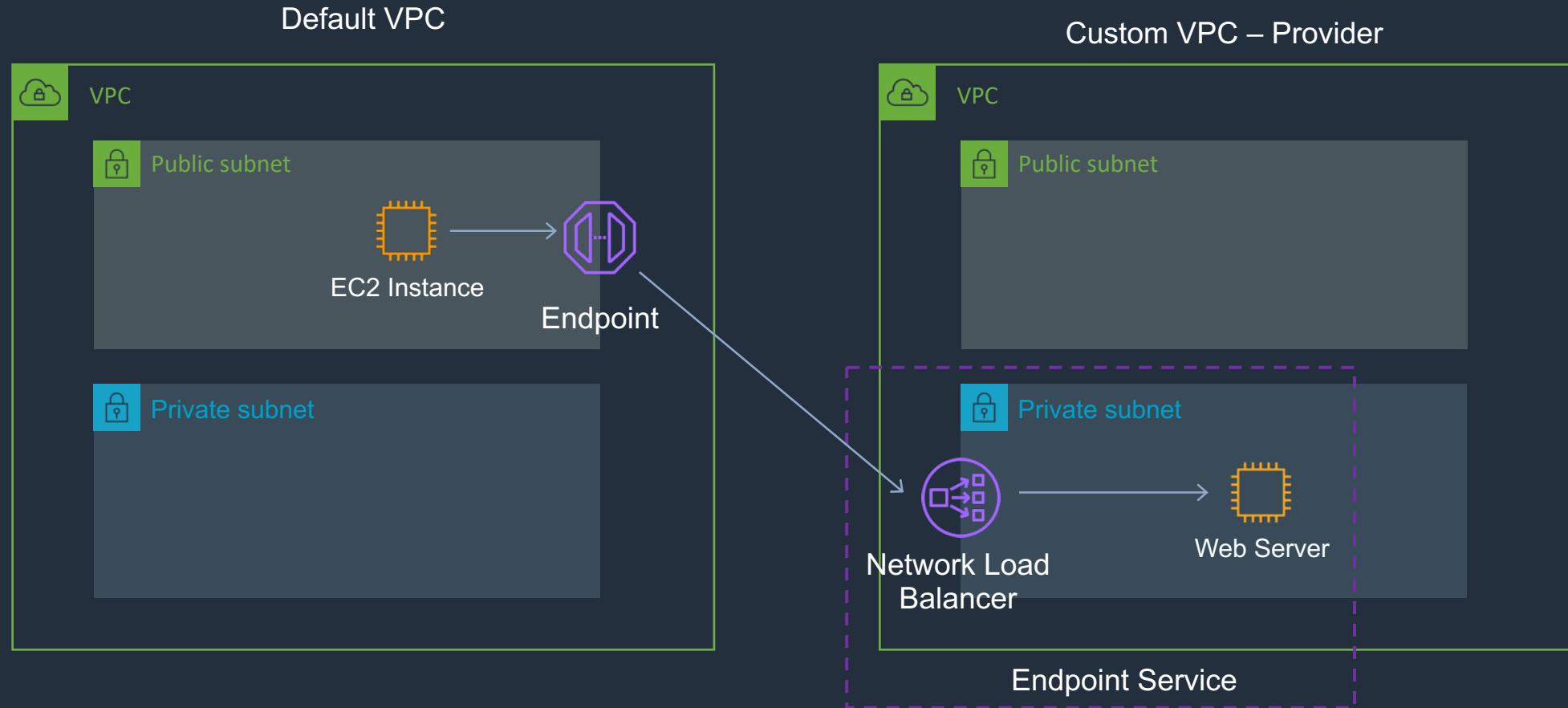
Outbound:

Protocol	Port	Source	Action
All	All	0.0.0.0/0	DENY
All	All	::/0	DENY

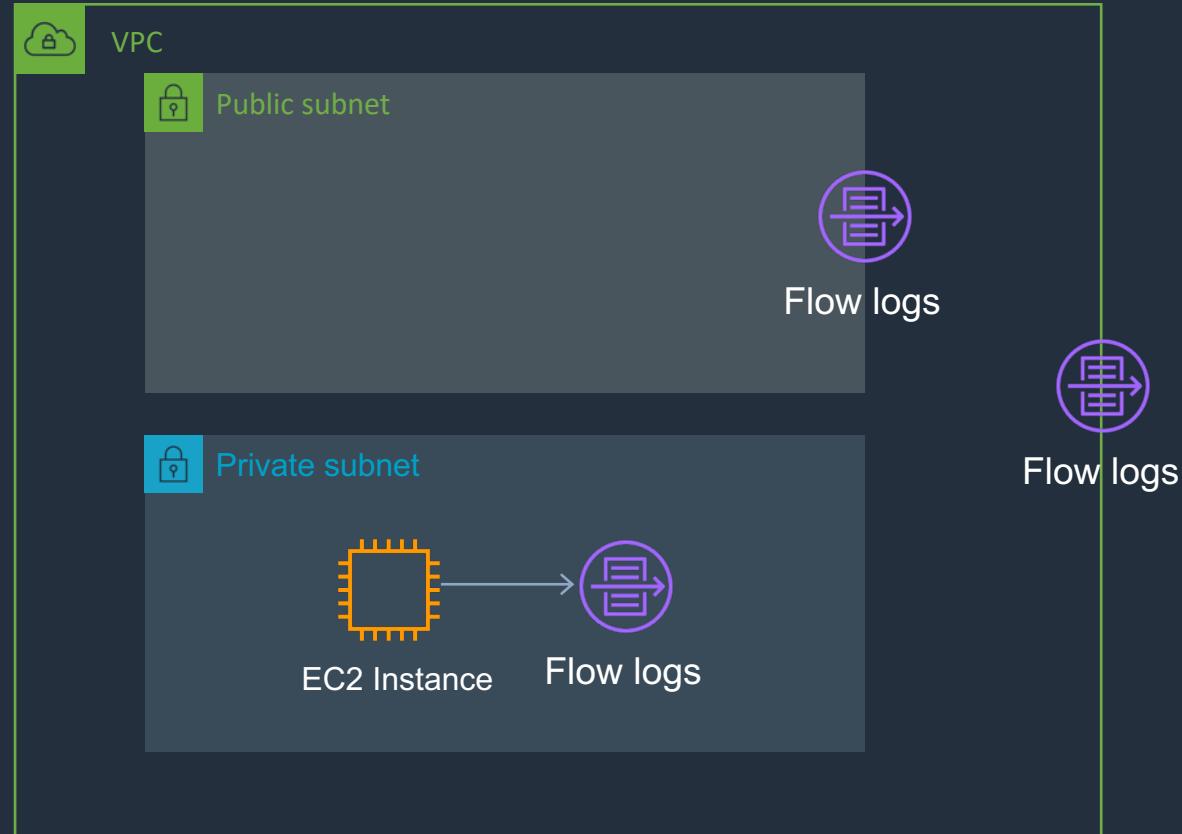
## Section 5: Security Groups vs Network ACLs

Security Group	Network ACL
Operates at the instance level	Operates at the subnet level
Supports allow rules only	Supports allow and deny rules
Stateful	Stateless
Evaluates all rules	Processes rules in order
Applies to an instance only if associated with a group	Automatically applies to all instances in the subnets its associated with

## Section 5: VPC Endpoint Services



## Section 5: VPC Flow Logs



# SECTION 6

# AWS Route 53

## Section 6: Route 53 Overview



Amazon Route 53



## Section 6: Route 53 DNS Record Types

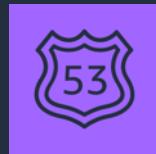
### Supported DNS records

- A (address record)
- AAAA (IPv6 address record)
- CNAME (canonical name record)
- Alias (an Amazon Route 53-specific virtual record)
- CAA (certification authority authorization)
- MX (mail exchange record)
- NAPTR (name authority pointer record)
- NS (name server record)
- PTR (pointer record)
- SOA (start of authority record)
- SPF (sender policy framework)
- SRV (service locator)
- TXT (text record)

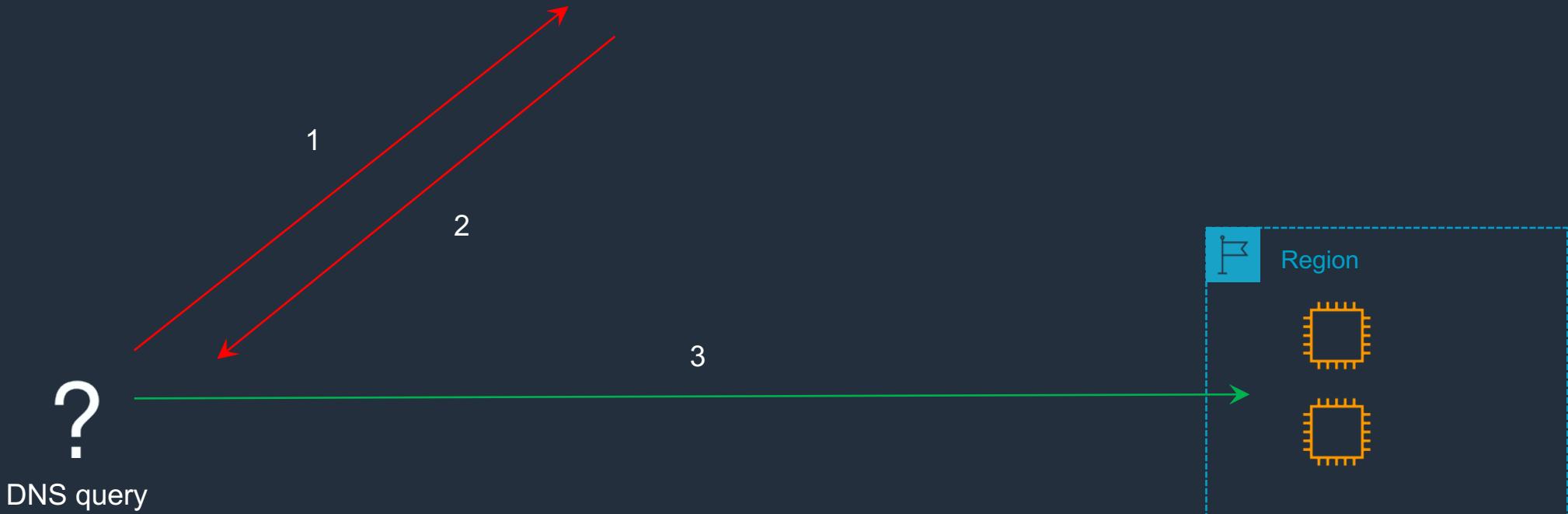
CNAME	Alias
Route 53 charges for CNAME queries	Route 53 doesn't charge for alias queries to AWS resources
You can't create a CNAME record at the top node of a DNS namespace (zone apex)	You can create an alias record at the zone apex (however you can't route to a CNAME at the zone apex)
A CNAME can point to any DNS record that is hosted anywhere	An alias record can only point to a CloudFront distribution, Elastic Beanstalk environment, ELB, S3 bucket as a static website, or to another record in the same hosted zone that you're creating the alias record in

## Section 6: Route 53 - Simple Routing Policy

Name	Type	Value	TTL
simple.dctlabs.com	A	1.1.1.1	60
		2.2.2.2	
simpler.dctlabs.com	A	3.3.3.3	60

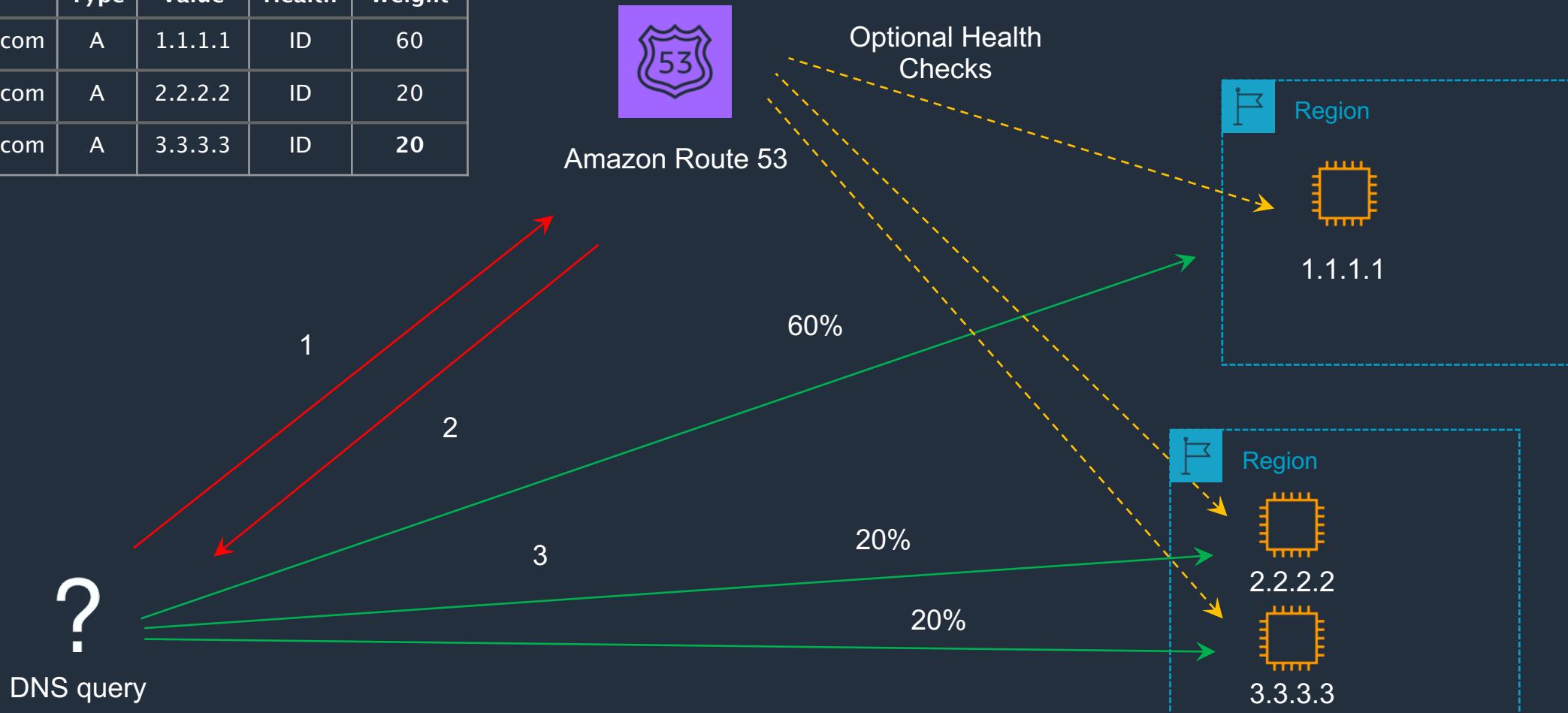


Amazon Route 53



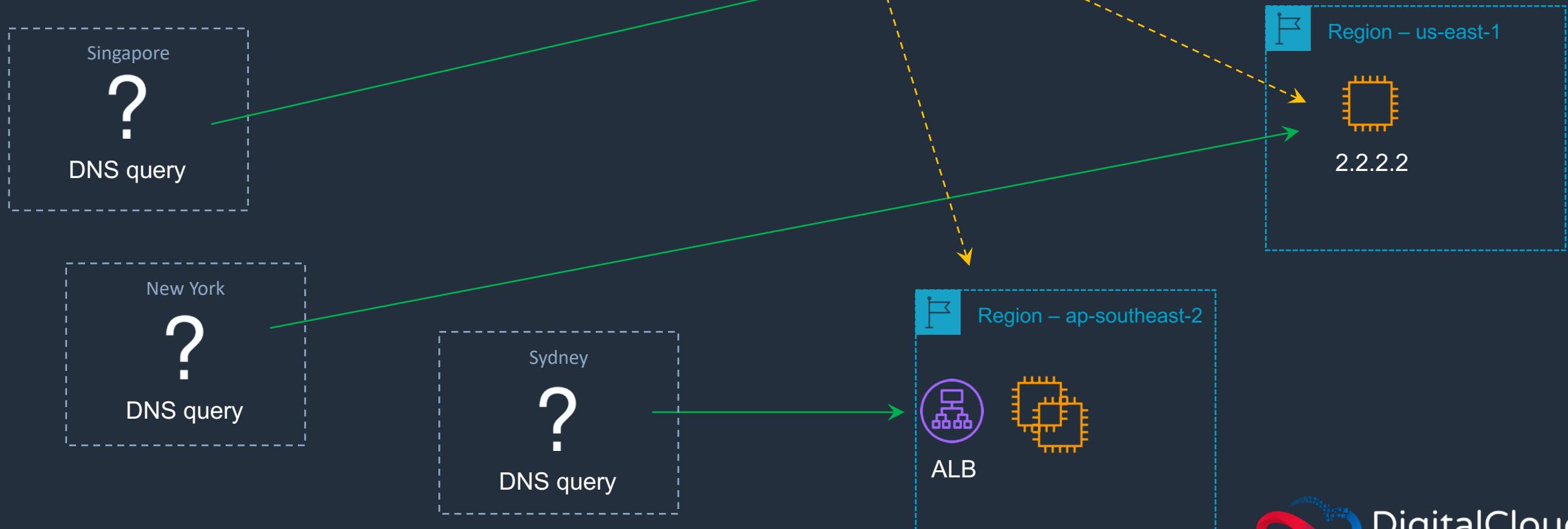
## Section 6: Route 53 - Weighted Routing Policy

Name	Type	Value	Health	Weight
weighted.dctlabs.com	A	1.1.1.1	ID	60
weighted.dctlabs.com	A	2.2.2.2	ID	20
weighted.dctlabs.com	A	3.3.3.3	ID	20



## Section 6: Route 53 - Latency Routing Policy

Name	Type	Value	Health	Region
latency.dctlabs.com	A	1.1.1.1	ID	ap-southeast-1
latency.dctlabs.com	A	2.2.2.2	ID	us-east-1
latency.dctlabs.com	A	<i>alb-id</i>	ID	ap-southeast-2



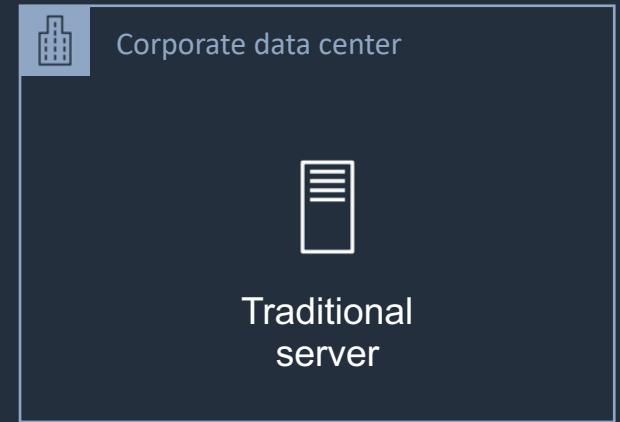
## Section 6: Route 53 - Failover Routing Policy

Name	Type	Value	Health	Record Type
failover.dctlabs.com	A	1.1.1.1	ID	Primary
failover.dctlabs.com	A	<i>alb-id</i>		Secondary

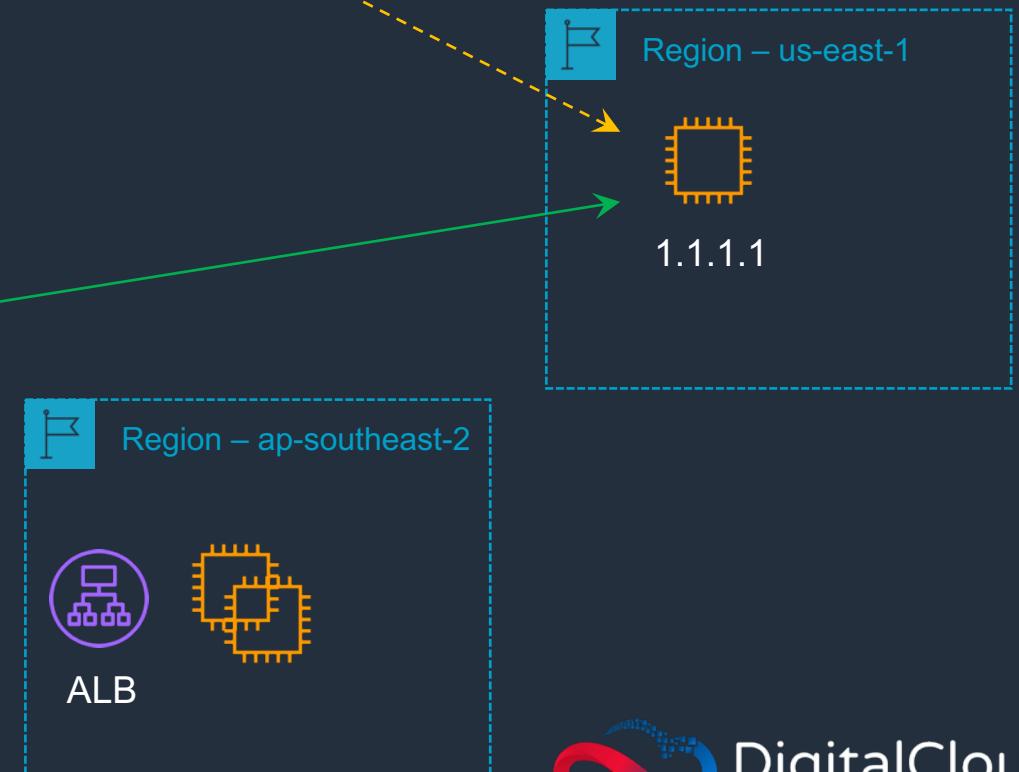


Amazon Route 53

Health Check  
required on  
Primary

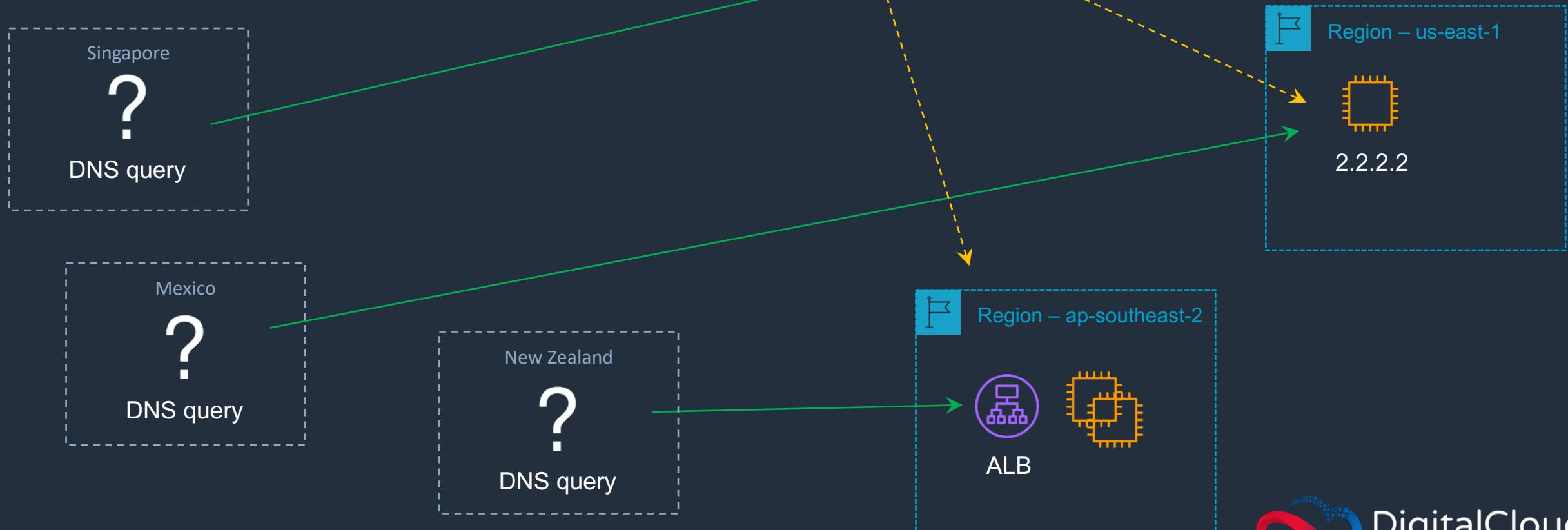


DNS query



## Section 6: Route 53 - Geolocation Routing Policy

Name	Type	Value	Health	Geolocation
geolocation.dctlabs.com	A	1.1.1.1	ID	Singapore
geolocation.dctlabs.com	A	2.2.2.2	ID	Default
geolocation.dctlabs.com	A	<i>alb-id</i>	ID	Oceania



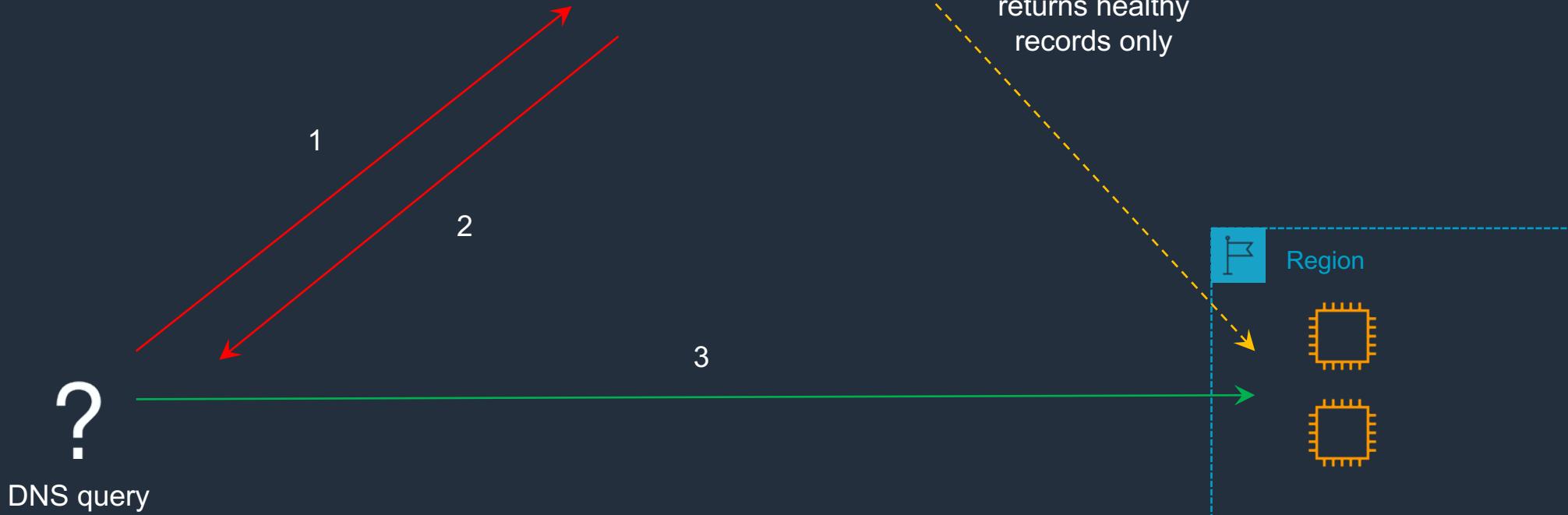
## Section 6: Route 53 - Multivalue Routing Policy

Name	Type	Value	Health	Multi Value
multivalue.dctlabs.com	A	1.1.1.1	ID	Yes
multivalue.dctlabs.com	A	2.2.2.2	ID	Yes
multivalue.dctlabs.com	A	3.3.3.3	ID	Yes



Amazon Route 53

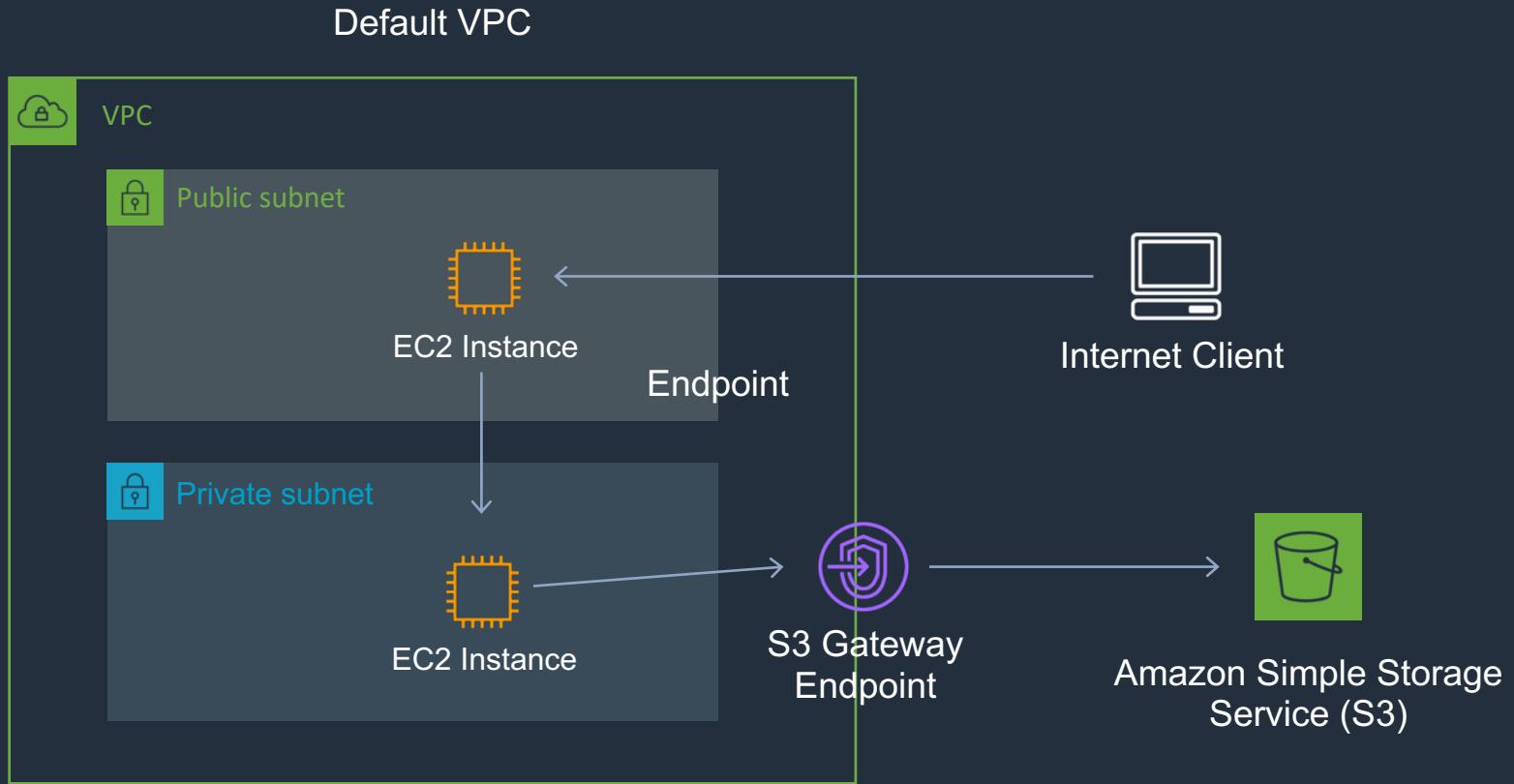
Health Checks:  
returns healthy  
records only



# SECTION 7

## Amazon S3 and CloudFront

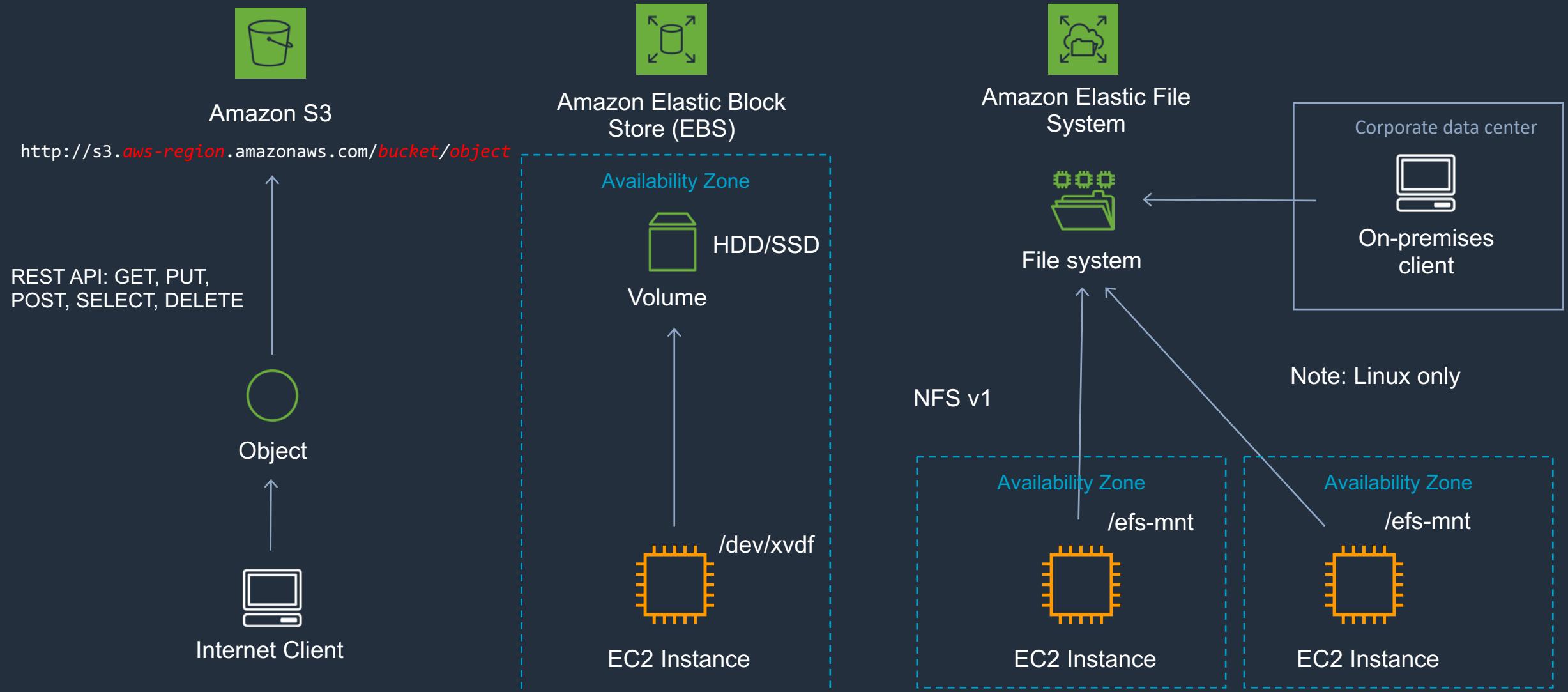
## Section 7: S3 Gateway Endpoints



### Route Table

Destination	Target
<code>pl-6ca54005 (com.amazonaws.ap-southeast-2.s3, 54.231.248.0/22, 54.231.252.0/24, 52.95.128.0/21)</code>	<code>vpce-ID</code>

## Section 7: Block, Object and File Storage



## Section 7: Amazon S3 Overview



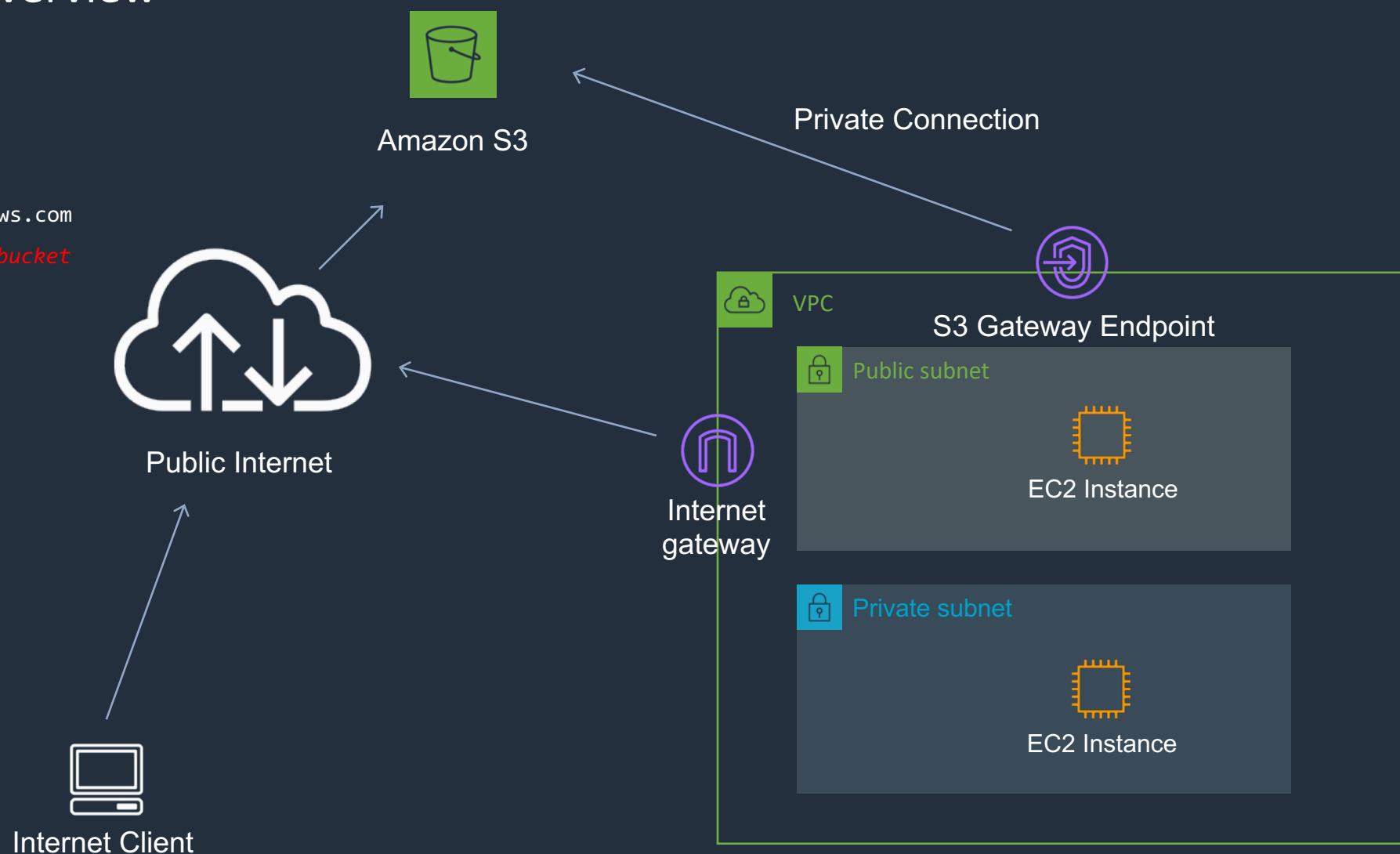
Bucket

[http://\*bucket\*.s3.\*aws-region\*.amazonaws.com](http://bucket.s3.amazonaws.com)  
[http://s3.\*aws-region\*.amazonaws.com/\*bucket\*](http://s3.amazonaws.com/bucket)



Object

- Key
- Version ID
- Value
- Metadata
- Subresources
- Access control information



# Section 7: Identity-Based and Resource-Based Policies

## Example Policy

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Sid": "SeeBucketListInTheConsole",  
      "Action": ["s3>ListAllMyBuckets"],  
      "Effect": "Allow",  
      "Resource": ["arn:aws:s3:::*"]  
    }  
  ]  
}
```

### Identity-based policies



IAM Role      Inline Policy

### Resource-based policy



Bucket      Policy

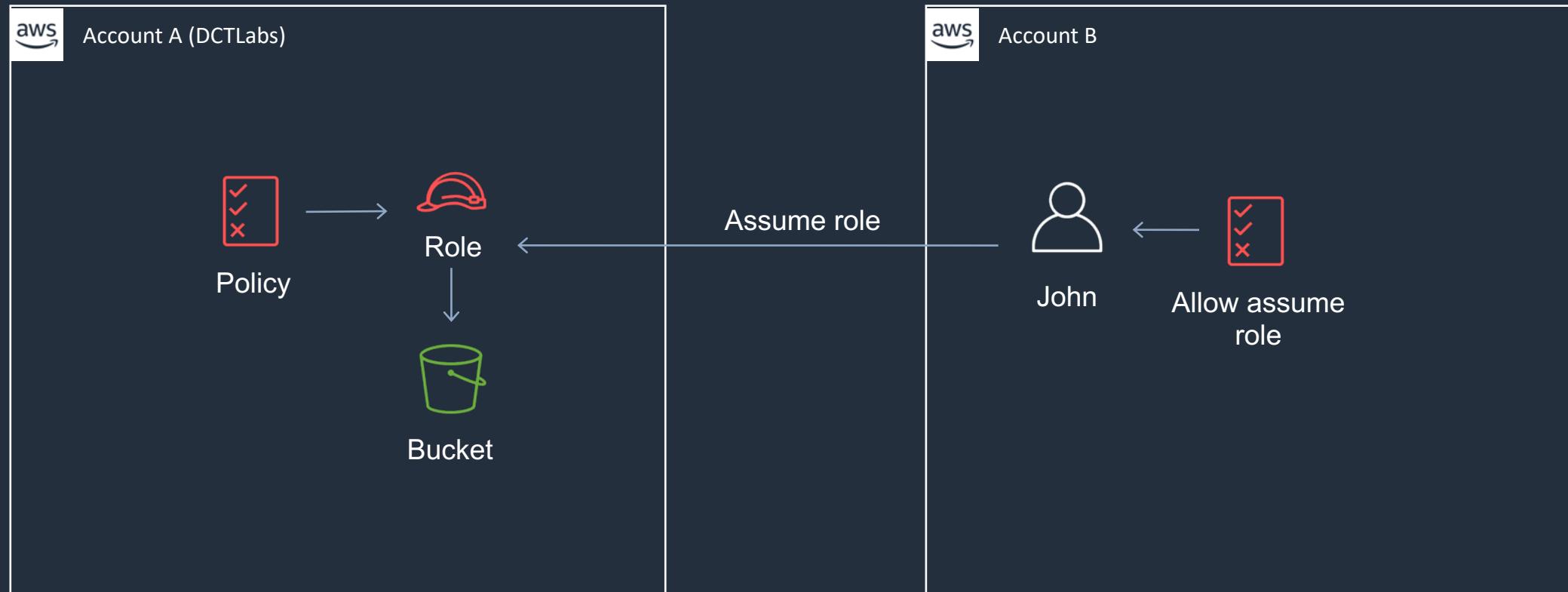


IAM User      Inline Policy



IAM Group      Policy

## Section 7: Cross-Account Access

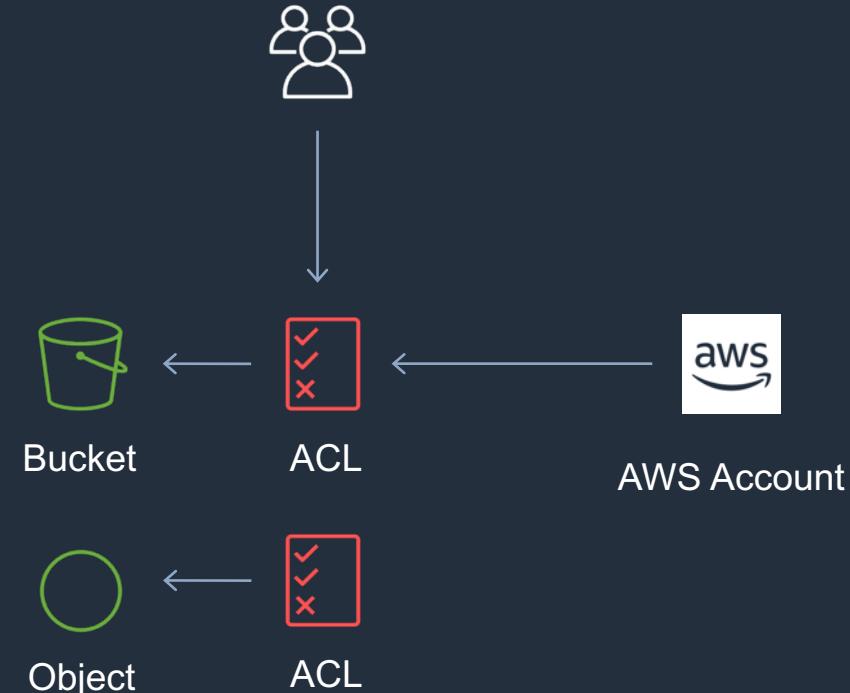


# Section 7: Access Control Lists

## Example ACL

```
... <AccessControlPolicy>
<Owner>
  <ID> AccountACanonicalUserID </ID>
  <DisplayName> AccountADisplayName </DisplayName>
</Owner>
<AccessControlList>
...
  <Grant>
    <Grantee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="CanonicalUser">
      <ID> AccountBCanonicalUserID </ID>
      <DisplayName> AccountBDisplayName </DisplayName>
    </Grantee>
    <Permission> WRITE </Permission>
  </Grant>
...
</AccessControlList>
</AccessControlPolicy>
```

- S3 Predefined Group
- Authenticated Users
- All Users
- Log Delivery Group

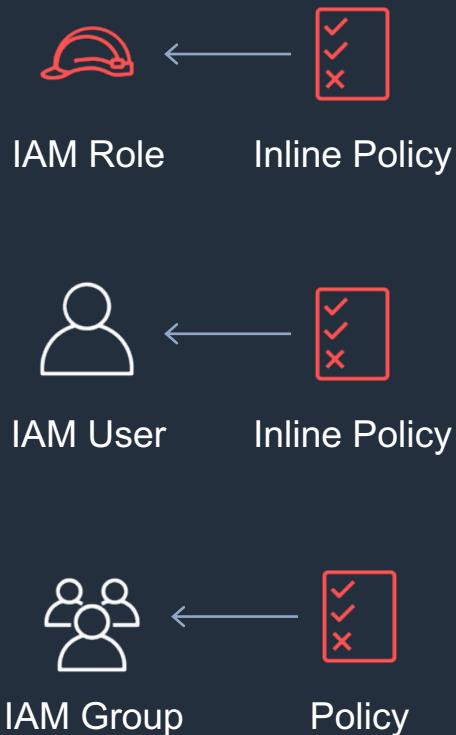


## Section 7: Access Control List Permissions

Permissions	When granted on a bucket	When granted on an object
READ	Allows grantee to list the objects in the bucket	Allows grantee to read the object data and its metadata
WRITE	Allows grantee to create, overwrite, and delete any object in the bucket	Not applicable
READ_ACP	Allows grantee to read the bucket ACL	Allows grantee to read the object ACL
WRITE_ACP	Allows grantee to write the ACL for the applicable bucket	Allows grantee to write the ACL for the applicable object
FULL_CONTROL	Allows grantee the READ, WRITE, READ_ACP, and WRITE_ACP permissions on the bucket	Allows grantee the READ, READ_ACP, and WRITE_ACP permissions on the object

## Section 7: Choosing Access Control Options

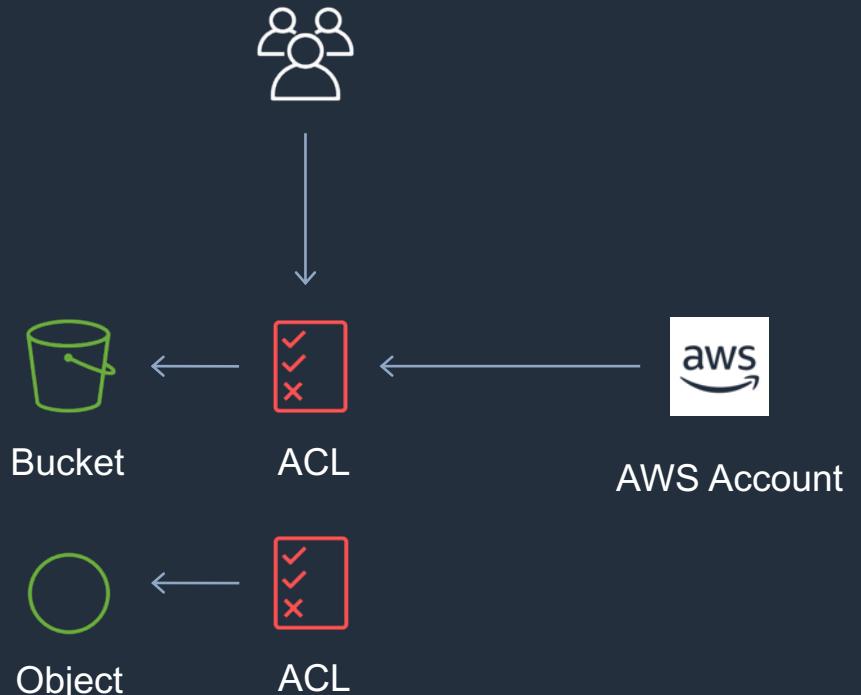
### Identity-based policies



### Resource-based policy

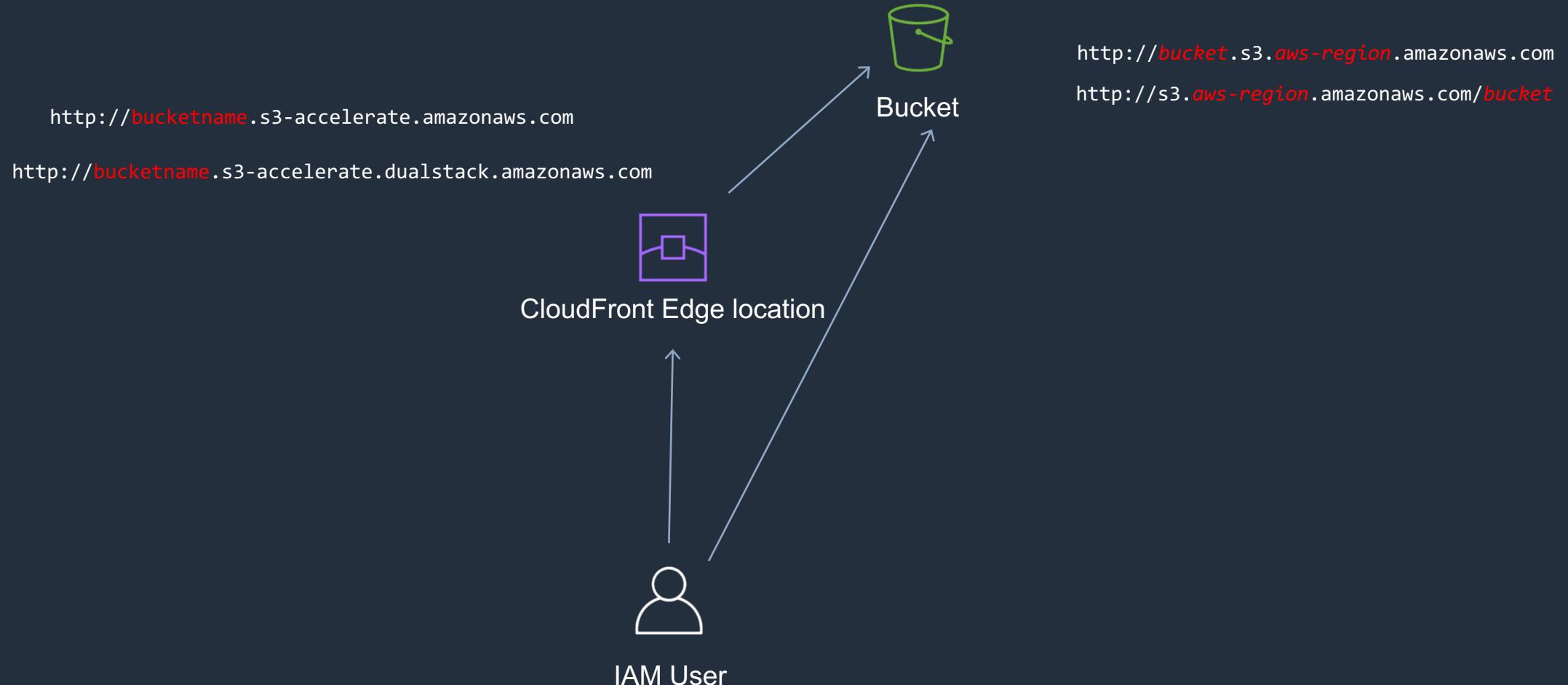


### S3 Predefined Group



- Authenticated Users
- All Users
- Log Delivery Group

## Section 7: Transfer Acceleration



# Section 7: S3 Encryption

## Server-side encryption with S3 managed keys (SSE-S3)

- S3 managed keys
- Unique object keys
- Master key
- AES 256



Encryption / decryption



## Server-side encryption with AWS KMS managed keys (SSE-KMS)



- KMS managed keys
- Customer master keys
- CMK can be customer generated



Encryption / decryption



## Server-side encryption with client provided keys (SSE-C)



Encryption / decryption



- Client managed keys
- AWS KMS CMK



## Client side encryption

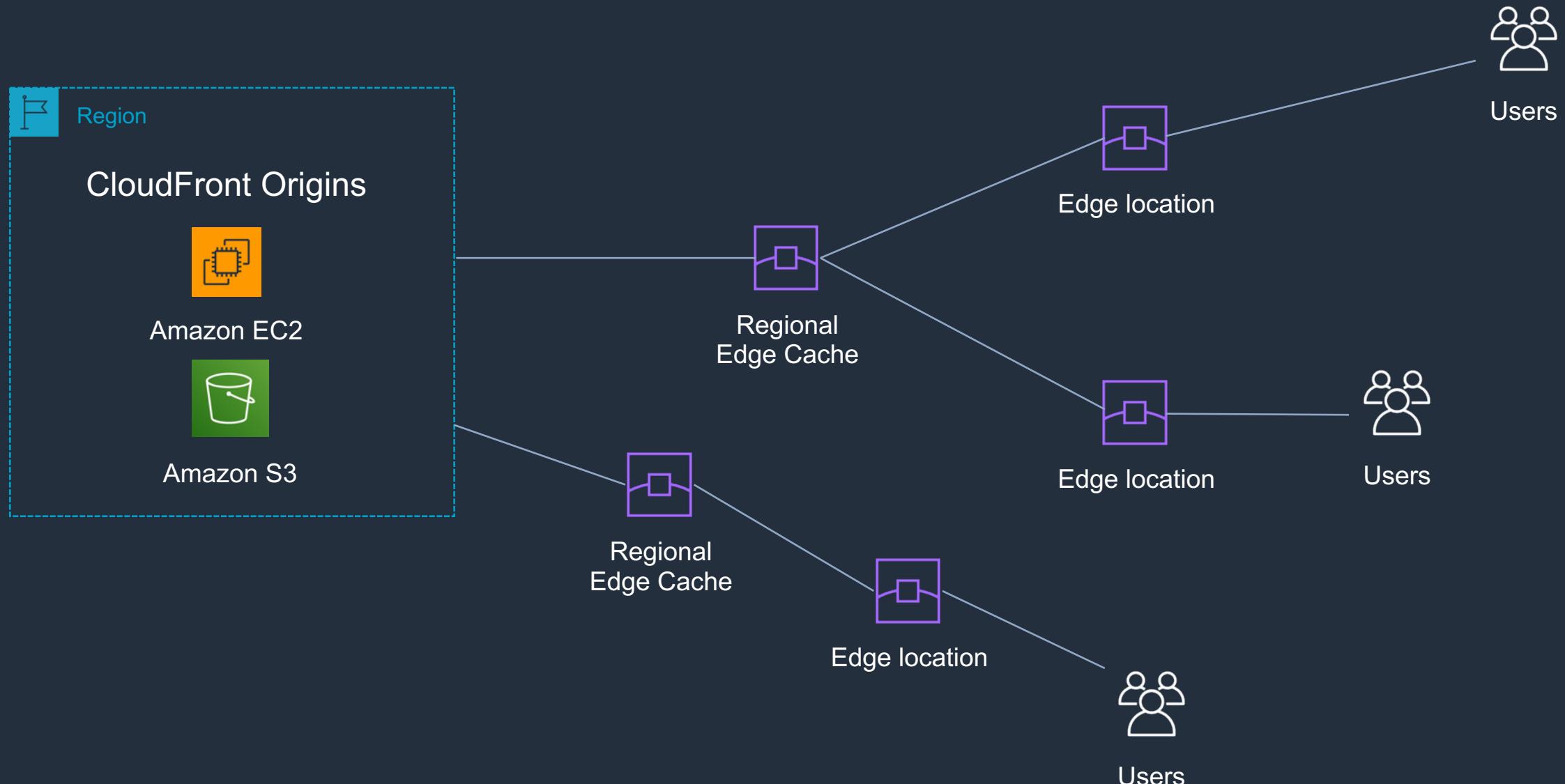


Encryption / decryption

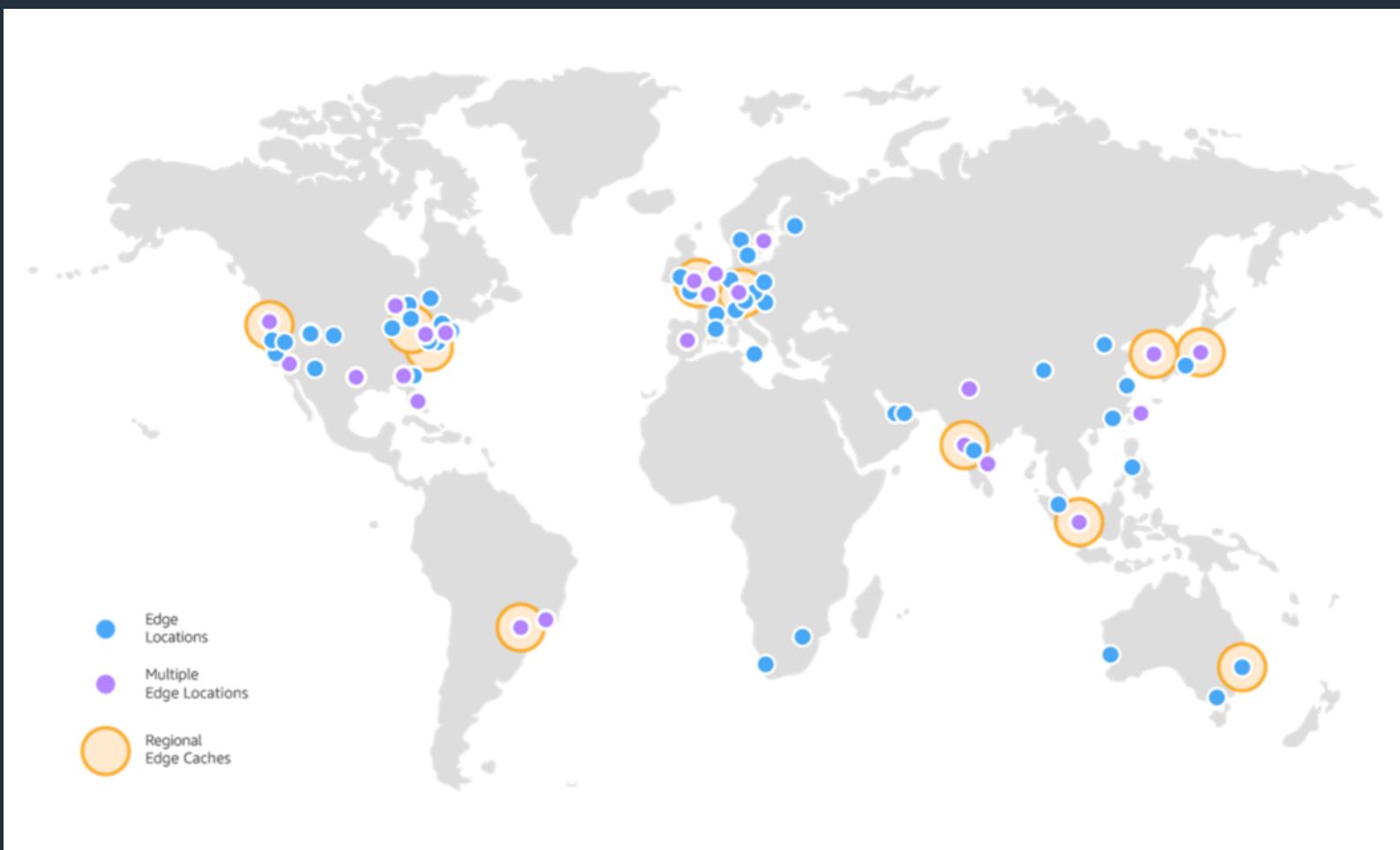


- Client managed keys
- Not stored on AWS

## Section 7: CloudFront Overview

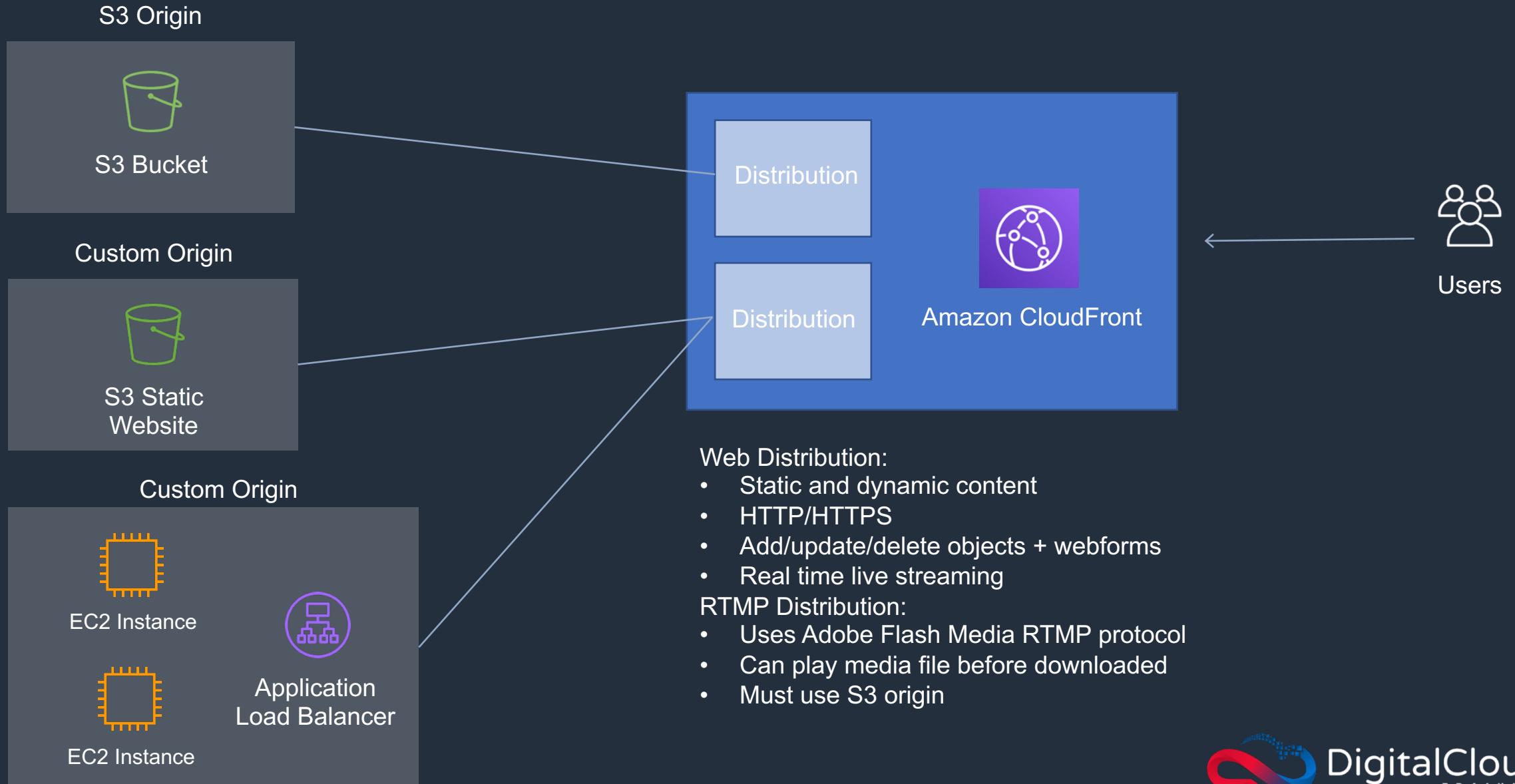


## Section 7: CloudFront – Points of Presence

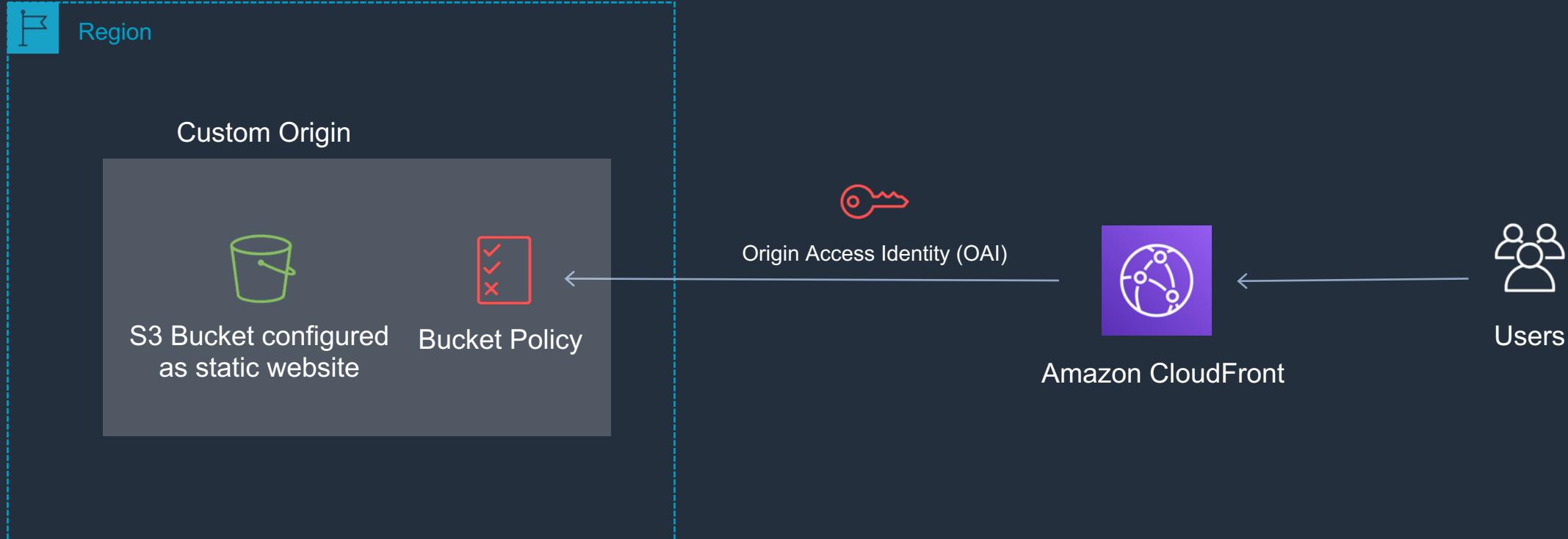


- Points of Presence:
- 176 Edge Locations
  - 11 Regional Edge Caches
  - 69 cities
  - 30 countries

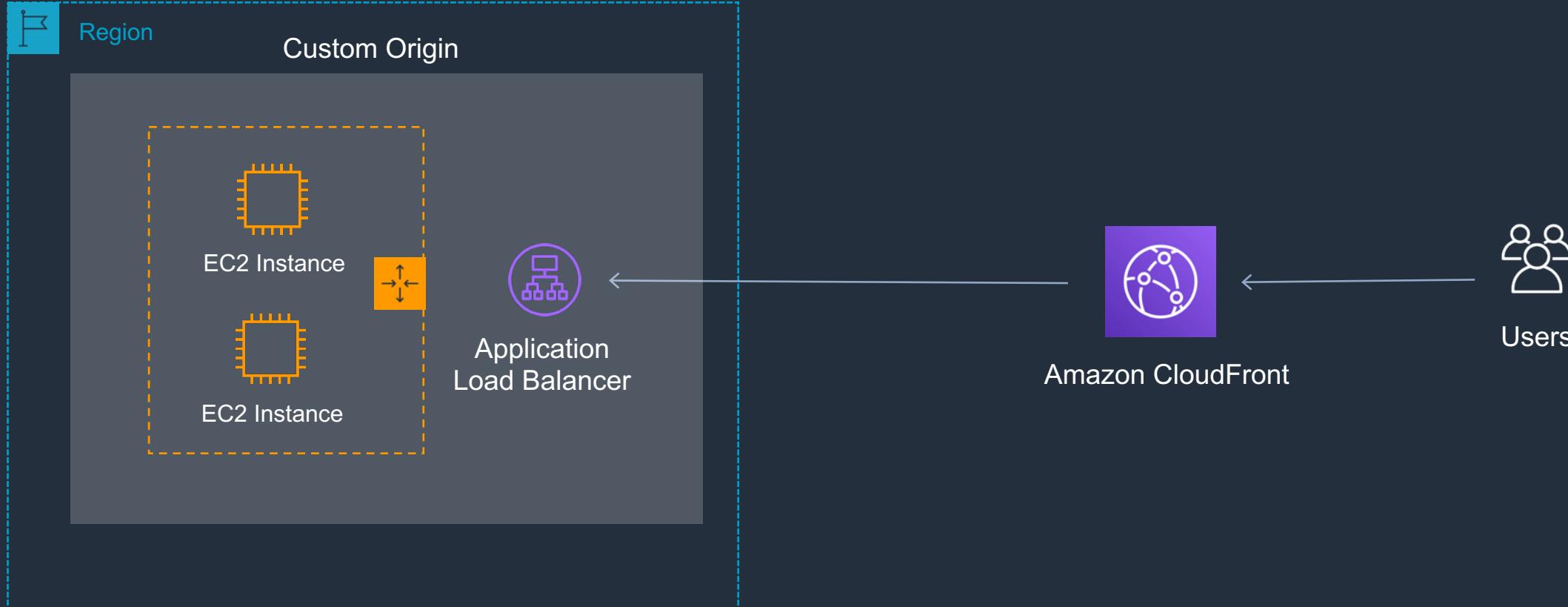
## Section 7: CloudFront Distribution and Origins



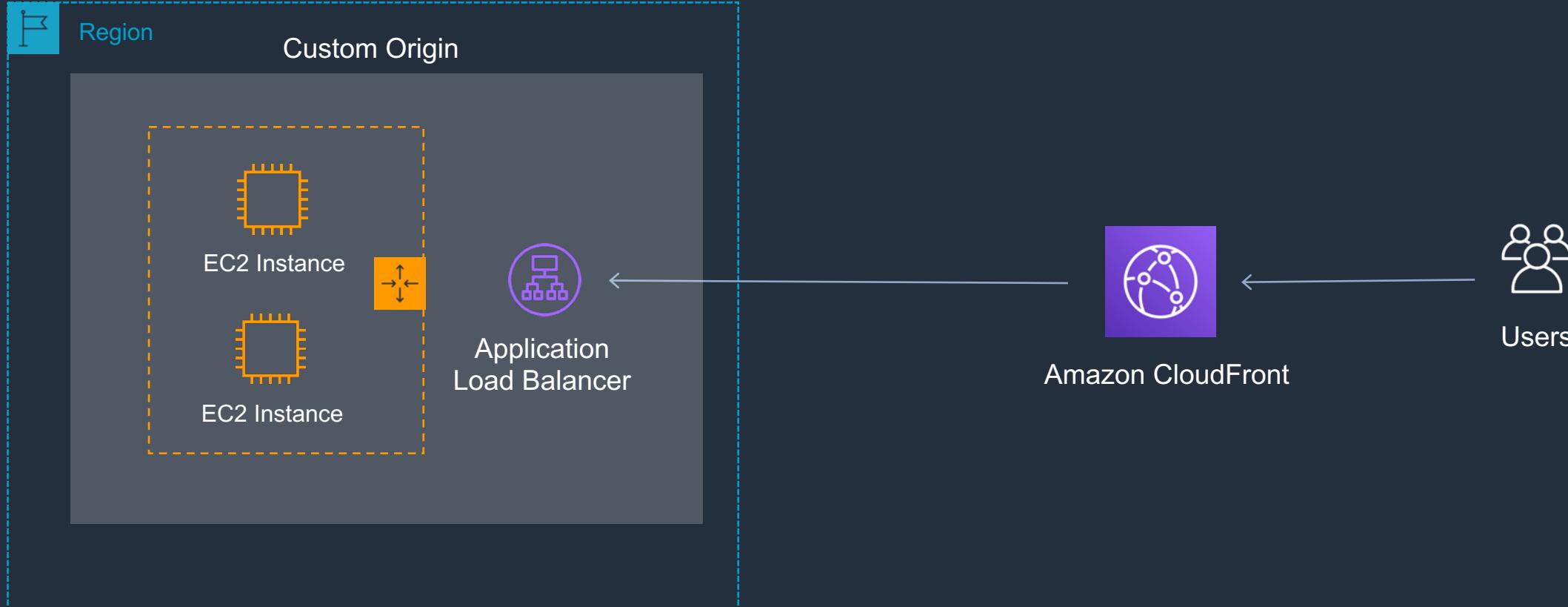
## Section 7: CloudFront with S3 Static Website



## Section 7: CloudFront with ALB and EC2 Custom Origin



## Section 7: CloudFront with Lambda@Edge



# SECTION 9

## Infrastructure as Code: AWS CloudFormation

# AWS CloudFormation

- AWS CloudFormation is a service that allows you to manage, configure and provision your AWS infrastructure as code.
- AWS CloudFormation provides a common language for you to describe and provision all the infrastructure resources in your cloud environment.
- Resources are defined using a CloudFormation template.

```
1  AWSTemplateFormatVersion: 2010-09-09
2  Description: >-
3  AWS CloudFormation Sample Template LAMP_Single_Instance: Create a LAMP stack
4  using a single EC2 instance and a local MySQL database for storage.
5  Parameters:
6    KeyName:
7      Description: Name of an existing EC2 KeyPair to enable SSH access to the instance
8      Type: 'AWS::EC2::KeyPair::KeyName'
9      ConstraintDescription: must be the name of an existing EC2 KeyPair.
10   DBName:
11     Default: MyDatabase
12     Description: MySQL database name
13     Type: String
14     MinLength: '1'
15     MaxLength: '64'
16     AllowedPattern: '[a-zA-Z][a-zA-Z0-9]*'
17     ConstraintDescription: must begin with a letter and contain only alphanumeric characters.
18   DBUser:
19     NoEcho: 'true'
20     Description: Username for MySQL database access
21     Type: String
22     MinLength: '1'
23     MaxLength: '16'
24     AllowedPattern: '[a-zA-Z][a-zA-Z0-9]*'
25     ConstraintDescription: must begin with a letter and contain only alphanumeric characters.
26   DBPassword:
27     NoEcho: 'true'
28     Description: Password for MySQL database access
29     Type: String
```

# AWS CloudFormation

- CloudFormation can be used to provision a broad range of AWS resources.
- Think of CloudFormation as deploying infrastructure as code.

```
1 "AWSTemplateFormatVersion" : "2010-09-09",
2
3 "Description" : "AWS CloudFormation Sample Template WordPress_Multi_AZ: WordPress is web
4
5 "Parameters" : {
6     "VpcId" : {
7         "Type" : "AWS::EC2::VPC::Id",
8         "Description" : "VpcId of your existing Virtual Private Cloud (VPC)",
9         "ConstraintDescription" : "must be the VPC Id of an existing Virtual Private Cloud."
10    },
11
12 "Subnets" : {
13     "Type" : "List<AWS::EC2::Subnet::Id>",
14     "Description" : "The list of SubnetIds in your Virtual Private Cloud (VPC)",
15     "ConstraintDescription" : "must be a list of at least two existing subnets associated
16    },
```



# Comparing AWS CloudFormation to Elastic Beanstalk

CloudFormation	Elastic Beanstalk
“Template-driven provisioning”	“Web apps made easy”
Deploys infrastructure using code	Deploys applications on EC2 (PaaS)
Can be used to deploy almost any AWS service	Deploys web applications based on Java, .NET, PHP, Node.js, Python, Ruby, Go, and Docker
Uses JSON or YAML template files	Uses ZIP or WAR files
CloudFormation can deploy Elastic Beanstalk environments	Elastic Beanstalk cannot deploy using CloudFormation
Similar to Terraform	Similar to Google App Engine

## AWS CloudFormation – Key Benefits

- Infrastructure is provisioned consistently, with fewer mistakes (human error).
- Less time and effort than configuring resources manually.
- You can use version control and peer review for your CloudFormation templates.
- Free to use (you're only charged for the resources provisioned).
- Can be used to manage updates and dependencies.
- Can be used to rollback and delete the entire stack as well.

# AWS CloudFormation – Key Concepts

Component	Description
Templates	The JSON or YAML text file that contains the instructions for building out the AWS environment
Stacks	The entire environment described by the template and created, updated, and deleted as a single unit
StackSets	AWS CloudFormation StackSets extends the functionality of stacks by enabling you to create, update, or delete stacks across multiple accounts and regions with a single operation
Change Sets	A summary of proposed changes to your stack that will allow you to see how those changes might impact your existing resources before implementing them

## AWS CloudFormation – Templates

- A template is a YAML or JSON template used to describe the end-state of the infrastructure you are either provisioning or changing.
- After creating the template, you upload it to CloudFormation directly or using Amazon S3.
- CloudFormation reads the template and makes the API calls on your behalf.
- The resulting resources are called a "Stack".
- Logical IDs are used to reference resources within the template.
- Physical IDs identify resources outside of AWS CloudFormation templates, but only after the resources have been created.

## AWS CloudFormation – Intrinsic Functions

- AWS CloudFormation provides several built-in functions that help you manage your stacks.
- Use intrinsic functions in your templates to assign values to properties that are not available until runtime.
- Full list can be found at:

<https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/intrinsic-function-reference.html>

## AWS CloudFormation – Intrinsic Functions - Ref

- The intrinsic function Ref returns the value of the specified parameter or resource.
- When you specify a parameter's logical name, it returns the value of the parameter.
- When you specify a resource's logical name, it returns a value that you can typically use to refer to that resource, such as a physical ID.
- The following resource declaration for an Elastic IP address needs the instance ID of an EC2 instance and uses the Ref function to specify the instance ID of the MyEC2Instance resource:

```
MyEIP:  
  Type: "AWS::EC2::EIP"  
  Properties:  
    InstanceId: !Ref MyEC2Instance
```

## Intrinsic Functions - Ref

- The intrinsic function Ref returns the value of the specified parameter or resource.
- When you specify a parameter's logical name, it returns the value of the parameter.
- When you specify a resource's logical name, it returns a value that you can typically use to refer to that resource, such as a physical ID.
- The following resource declaration for an Elastic IP address needs the instance ID of an EC2 instance and uses the Ref function to specify the instance ID of the MyEC2Instance resource:

```
MyEIP:  
  Type: "AWS::EC2::EIP"  
  Properties:  
    InstanceId: !Ref MyEC2Instance
```

## Intrinsic Functions - Fn::GetAtt

- The Fn::GetAtt intrinsic function returns the value of an attribute from a resource in the template.
- Full syntax (YAML): Fn::GetAtt: [ logicalNameOfResource, attributeName ]
- Short form (YAML): !GetAtt logicalNameOfResource.attributeName

## Intrinsic Functions - Fn::GetAtt

- The following example template returns the SourceSecurityGroup.OwnerAlias and SourceSecurityGroup.GroupName of the load balancer with the logical name myELB.

```
AWSTemplateFormatVersion: 2010-09-09
Resources:
  myELB:
    Type: AWS::ElasticLoadBalancing::LoadBalancer
    Properties:
      AvailabilityZones:
        - eu-west-1a
      Listeners:
        - LoadBalancerPort: '80'
          InstancePort: '80'
          Protocol: HTTP
  myELBIngressGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: ELB ingress group
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: '80'
          ToPort: '80'
      SourceSecurityGroupOwnerId: !GetAtt myELB.SourceSecurityGroup.OwnerAlias
      SourceSecurityGroupName: !GetAtt myELB.SourceSecurityGroup.GroupName
```

## Intrinsic Functions - Fn::FindInMap

- The intrinsic function Fn::FindInMap returns the value corresponding to keys in a two-level map that is declared in the Mappings section.
- Full syntax (YAML): Fn::FindInMap: [ MapName, TopLevelKey, SecondLevelKey ]
- Short form (YAML): !FindInMap [ MapName, TopLevelKey, SecondLevelKey ]

## Intrinsic Functions - Fn::FindInMap

- The following example shows how to use Fn::FindInMap for a template with a `Mappings` section that contains a single map, `RegionMap`, that associates AMIs with AWS regions:

```
Mappings:  
  RegionMap:  
    us-east-1:  
      HVM64: "ami-0ff8a91507f77f867"  
      HVMG2: "ami-0a584ac55a7631c0c"  
    us-west-1:  
      HVM64: "ami-0bdb828fd58c52235"  
      HVMG2: "ami-066ee5fd4a9ef77f1"  
  
Resources:  
  myEC2Instance:  
    Type: "AWS::EC2::Instance"  
    Properties:  
      ImageId: !FindInMap  
        - RegionMap  
        - !Ref 'AWS::Region'  
        - HVM64  
    InstanceType: m1.small
```

## AWS CloudFormation – Resources

- Resources - the required Resources section declares the AWS resources that you want to include in the stack, such as an Amazon EC2 instance or an Amazon S3 bucket.
- Mandatory.
- Resources are declared and can reference each other.

```
Resources:  
  MyEC2Instance:  
    Type: "AWS::EC2::Instance"  
    Properties:  
      ImageId: "ami-0ff8a91507f77f867"
```

## AWS CloudFormation – Parameters

- Parameters – use the optional Parameters section to customize your templates.
- Parameters enable you to input custom values to your template each time you create or update a stack.
- Useful for template reuse.

```
Parameters:  
  InstanceTypeParameter:  
    Type: String  
    Default: t2.micro  
    AllowedValues:  
      - t2.micro  
      - m1.small  
      - m1.large  
    Description: Enter t2.micro, m1.small, or m1.large. Default is t2.micro.
```

## AWS CloudFormation – Mappings

- Mappings – the optional Mappings section matches a key to a corresponding set of named values.

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

- Exam tip: with mappings you can, for example, set values based on a region.

You can create a mapping that uses the region name as a key and contains the values you want to specify for each specific region.

## AWS CloudFormation – Outputs

- Outputs – the optional Outputs section declares output values that you can import into other stacks (to create cross-stack references), return in response (to describe stack calls), or view on the AWS CloudFormation console.
- In the following example YAML code, the output named StackVPC returns the ID of a VPC, and then exports the value for cross-stack referencing with the name VPCID appended to the stack's name

```
Outputs:  
  StackVPC:  
    Description: The ID of the VPC  
    Value: !Ref MyVPC  
    Export:  
      Name: !Sub "${AWS::StackName}-VPCID"
```

## AWS CloudFormation – Conditions

- Conditions – the optional Conditions section contains statements that define the circumstances under which entities are created or configured.
- In the sample YAML code below, resources are created only if the EnvType parameter is equal to prod:

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

## AWS CloudFormation – Transform

- Transform – the optional Transform section specifies one or more macros that AWS CloudFormation uses to process your template.
- The transform section can be used to reference additional code stored in S3, such as Lambda code or reusable snippets of CloudFormation code.

## AWS CloudFormation – Transform

- The AWS::Serverless transform specifies the version of the AWS Serverless Application Model (AWS SAM) to use. This model defines the AWS SAM syntax that you can use and how AWS CloudFormation processes it.
- The AWS::Include transform works with template snippets that are stored separately from the main AWS CloudFormation template.
- In the following example, the template uses AWS SAM syntax to simplify the declaration of a Lambda function and its execution role:

```
Transform: AWS::Serverless-2016-10-31
Resources:
  MyServerlessFunctionLogicalID:
    Type: AWS::Serverless::Function
    Properties:
      Handler: index.handler
      Runtime: nodejs8.10
      CodeUri: 's3://testBucket/mySourceCode.zip'
```

## AWS CloudFormation – Stacks

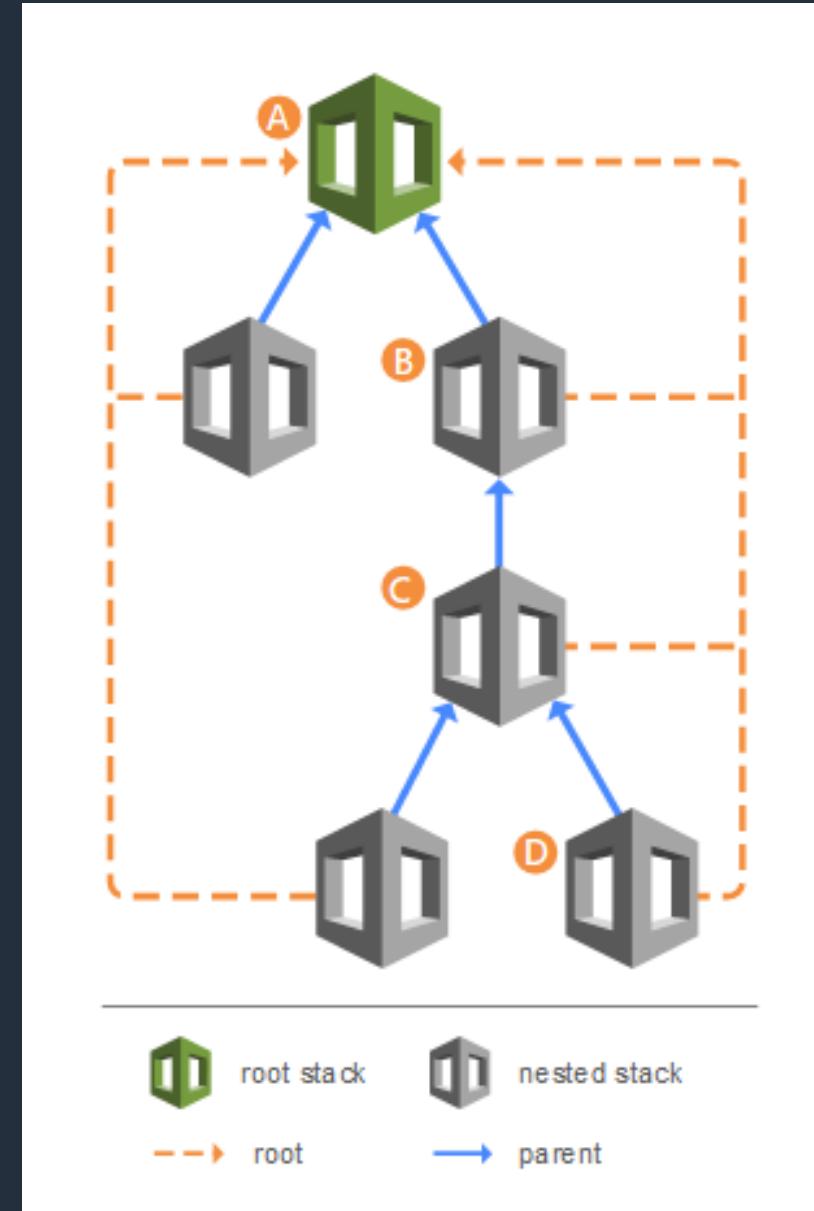
- Deployed resources based on templates.
- Create, update and delete stacks using templates.
- Deployed through the Management Console, CLI or APIs.
- Stack creation errors:
  - Automatic rollback on error is enabled by default.
  - You will be charged for resources provisioned even if there is an error.

## AWS CloudFormation – StackSets

- AWS CloudFormation StackSets extends the functionality of stacks by enabling you to create, update, or delete stacks across multiple accounts and regions with a single operation.
- Using an administrator account, you define and manage an AWS CloudFormation template, and use the template as the basis for provisioning stacks into selected target accounts across specified regions.
- An administrator account is the AWS account in which you create stack sets.
- A stack set is managed by signing in to the AWS administrator account in which it was created.
- A target account is the account into which you create, update, or delete one or more stacks in your stack set.

## AWS CloudFormation – Nested Stacks

- Nested stacks allow re-use of CloudFormation code for common use cases.
- For example standard configuration for a load balancer, web server, application server etc.
- Instead of copying out the code each time, create a standard template for each common use case and reference from within your CloudFormation template.



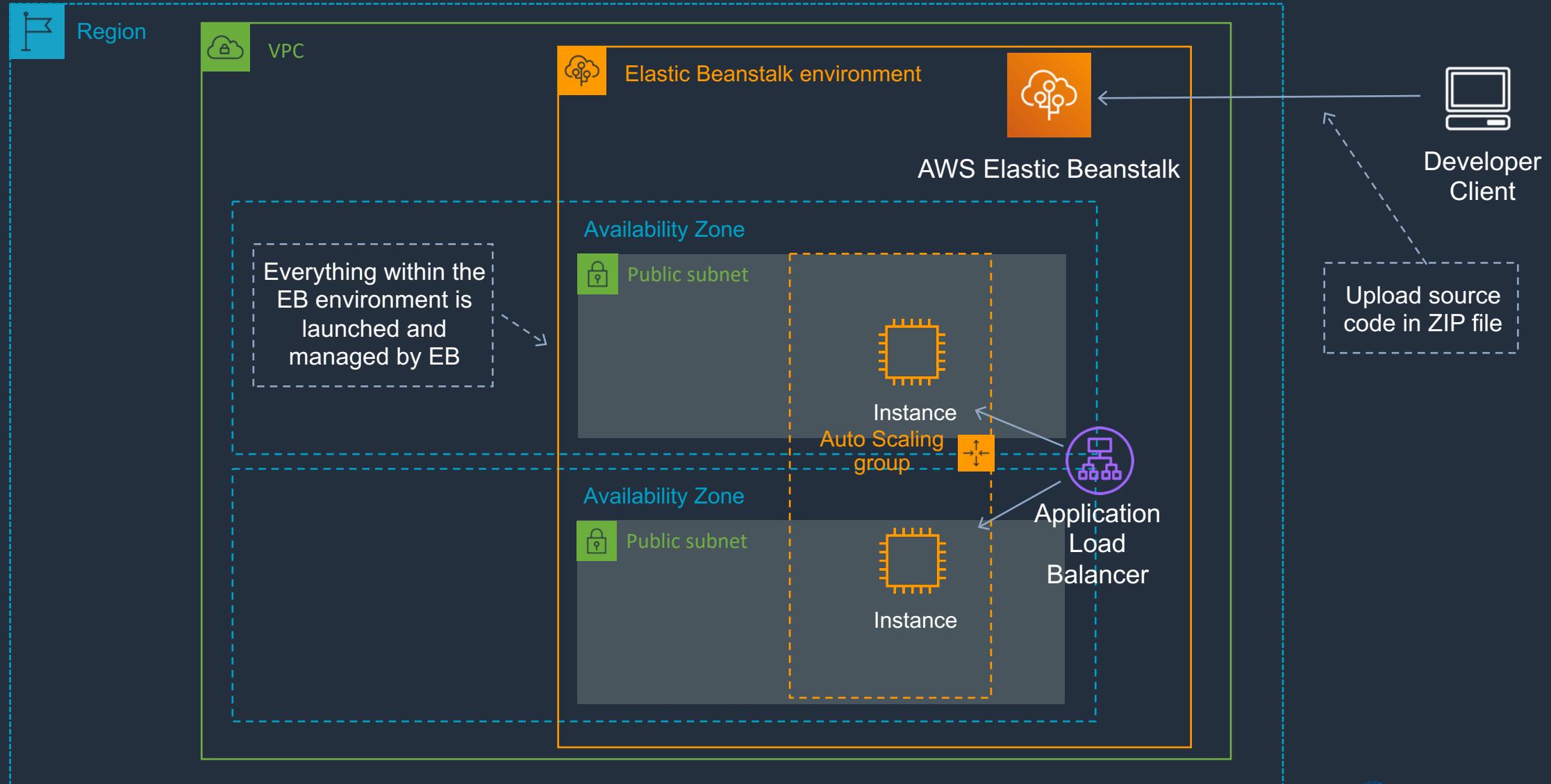
## AWS CloudFormation – Change Sets

- AWS CloudFormation provides two methods for updating stacks: direct update or creating and executing change sets.
- When you directly update a stack, you submit changes and AWS CloudFormation immediately deploys them.
- Use direct updates when you want to quickly deploy your updates.
- With change sets, you can preview the changes AWS CloudFormation will make to your stack, and then decide whether to apply those changes.

# SECTION 10

## Platform services: AWS Elastic Beanstalk

# AWS Elastic Beanstalk



## AWS Elastic Beanstalk

- AWS Elastic Beanstalk can be used to quickly deploy and manage applications in the AWS Cloud.
- Developers upload applications and Elastic Beanstalk handles the deployment details of capacity provisioning, load balancing, auto-scaling, and application health monitoring.
- AWS Elastic Beanstalk leverages Elastic Load Balancing and Auto Scaling to automatically scale your application in and out based on your application's specific needs.
- Considered a Platform as a Service (PaaS) solution.

## AWS Elastic Beanstalk

- Supports Java, .NET, PHP, Node.js, Python, Ruby, Go, and Docker web applications.
- Supports the following languages and development stacks:
  - Apache Tomcat for Java applications.
  - Apache HTTP Server for PHP applications.
  - Apache HTTP Server for Python applications.
  - Nginx or Apache HTTP Server for Node.js applications.
  - Passenger or Puma for Ruby applications.
  - Microsoft IIS 7.5, 8.0, and 8.5 for .NET applications.
  - Java SE.
  - Docker.
  - Go.

## AWS Elastic Beanstalk

- You maintain full control of the underlying resources.
- You pay only for the resources provisioned, not for Elastic Beanstalk itself.
- Elastic Beanstalk automatically scales your application up and down.
- The Managed Platform Updates feature automatically applies updates for your operating system, Java, PHP, Node.js etc.
- Elastic Beanstalk monitors and manages application health and information is viewable via a dashboard.
- AWS CloudFormation is used by Elastic Beanstalk to deploy the resources.
- Integrated with CloudWatch and X-Ray for performance data and metrics.

# AWS Elastic Beanstalk

There are several layers:

Applications:

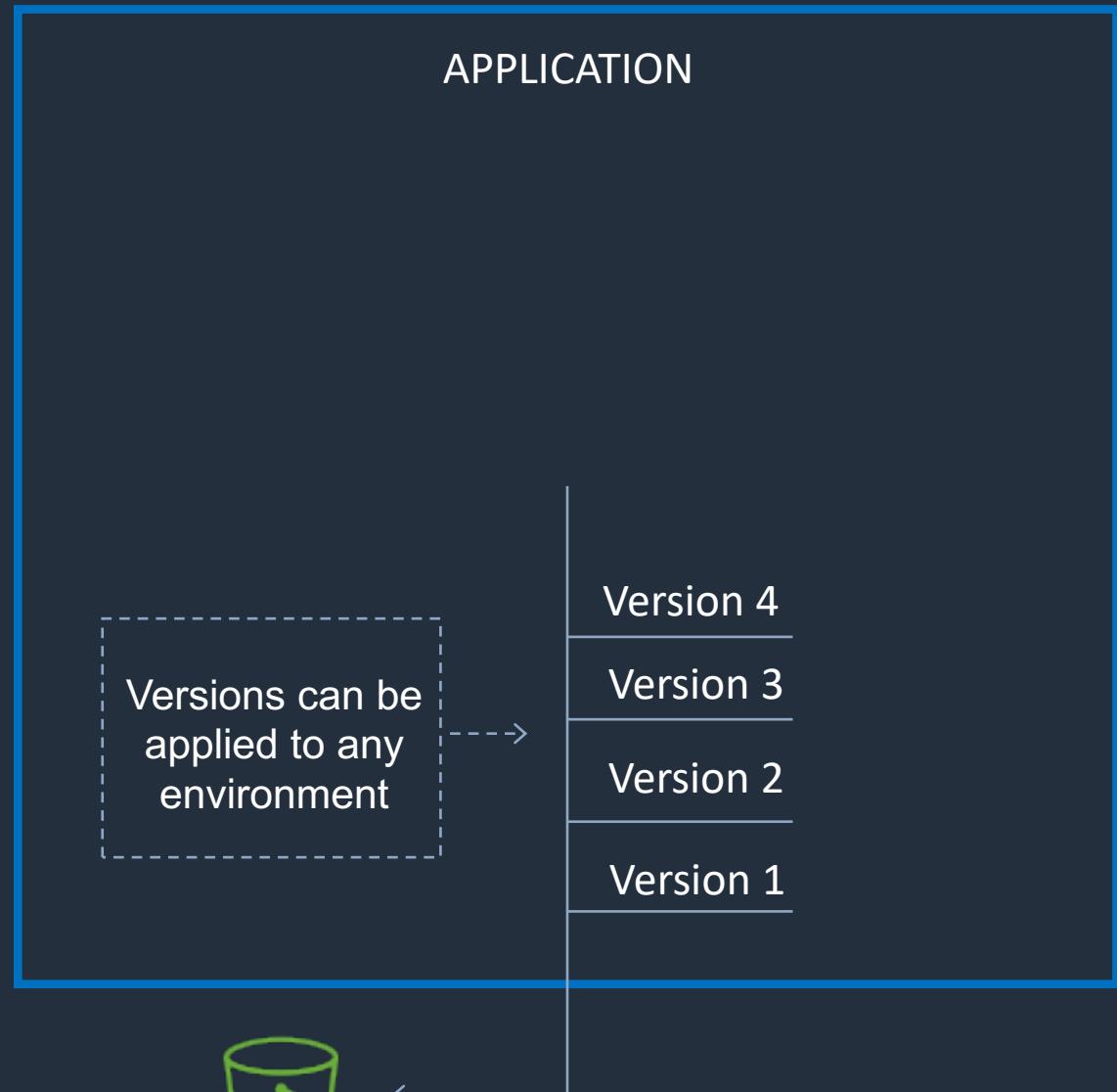
- Contain environments, environment configurations, and application versions.
- You can have multiple application versions held within an application.

APPLICATION

# AWS Elastic Beanstalk

## Application version

- A specific reference to a section of deployable code.
- The application version will point typically to an Amazon S3 bucket containing the code.

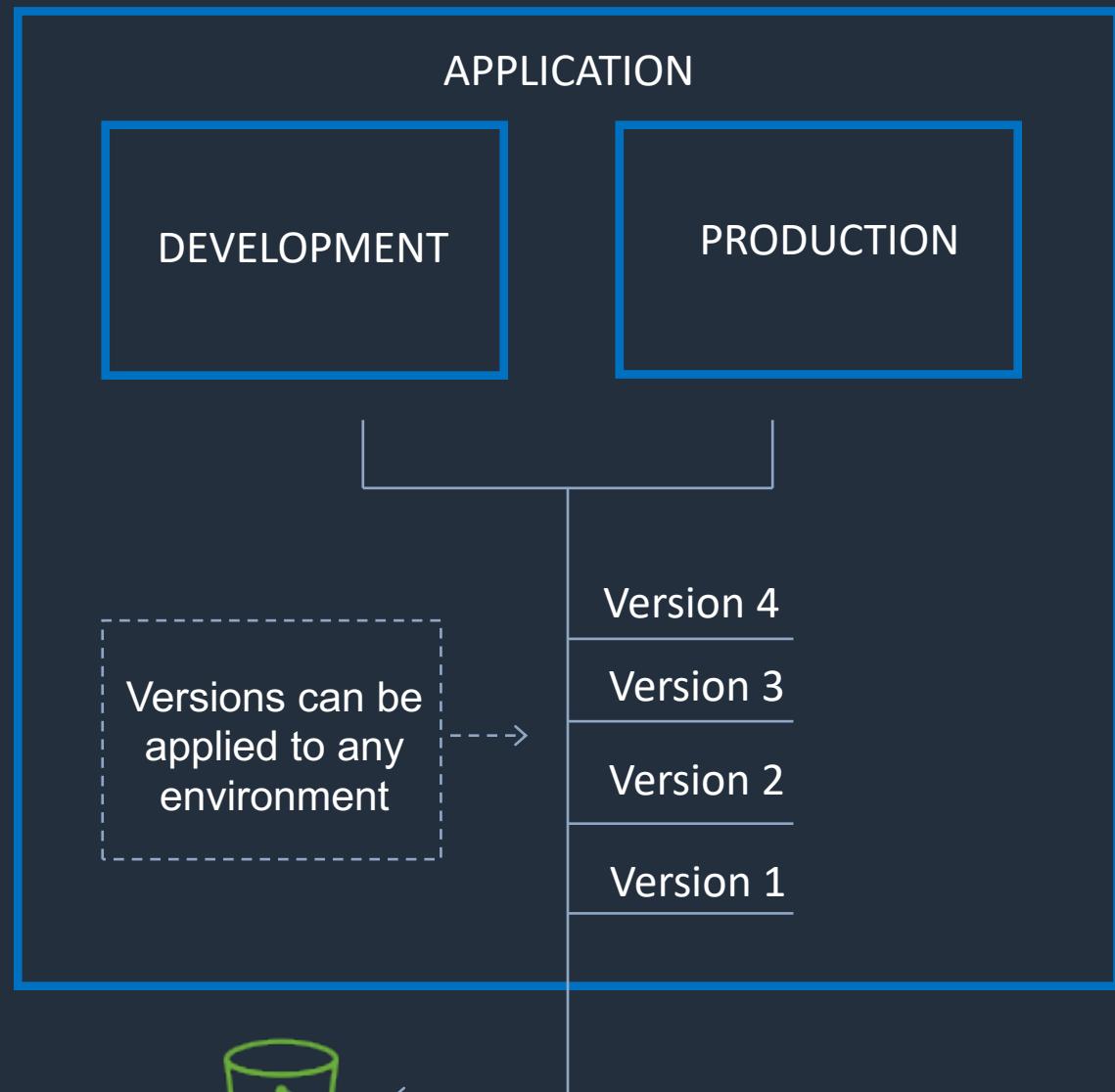


S3 Bucket

# AWS Elastic Beanstalk

## Environments:

- An application version that has been deployed on AWS resources.
- The resources are configured and provisioned by AWS Elastic Beanstalk.
- The environment is comprised of all the resources created by Elastic Beanstalk and not just an EC2 instance with your uploaded code.

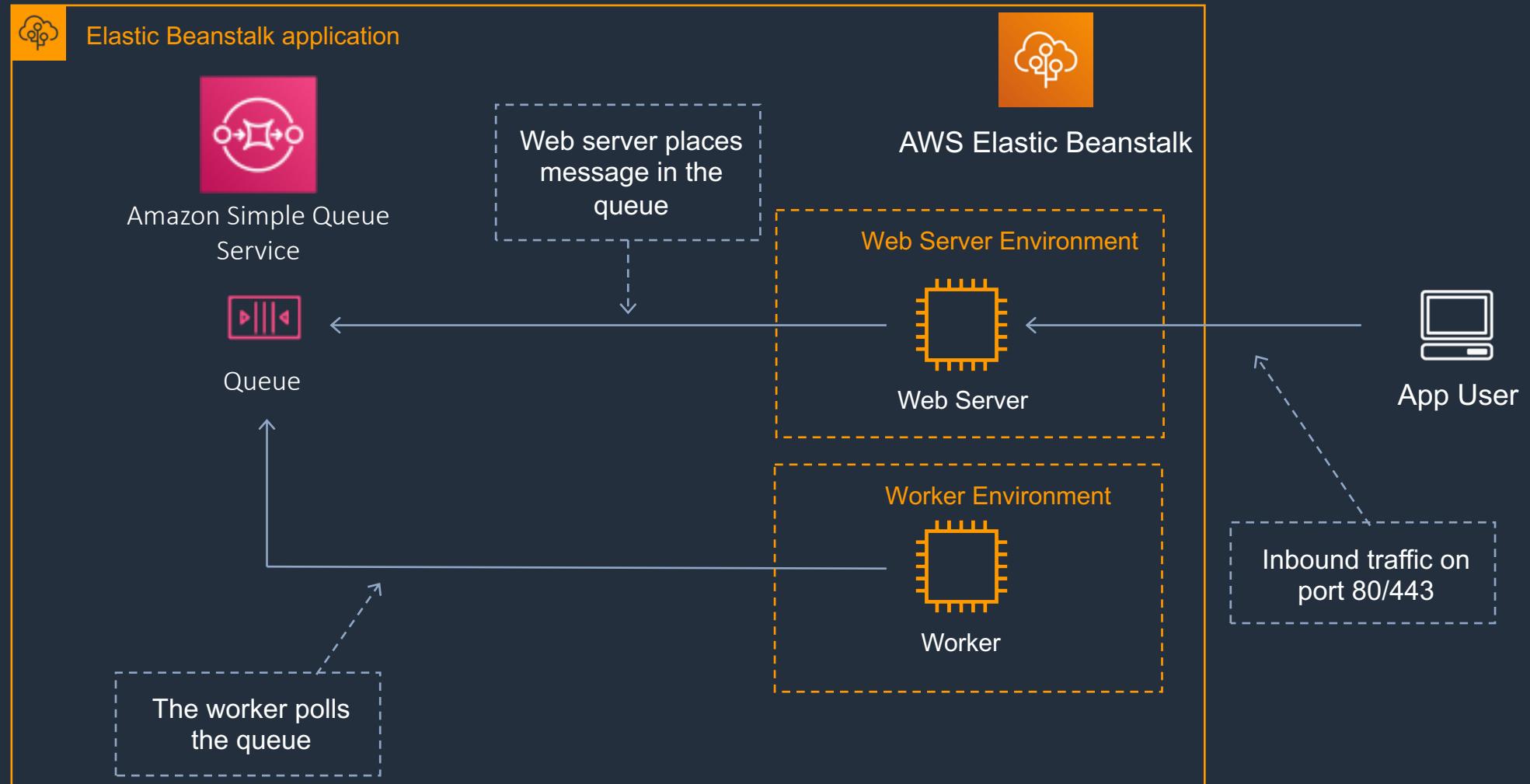


S3 Bucket

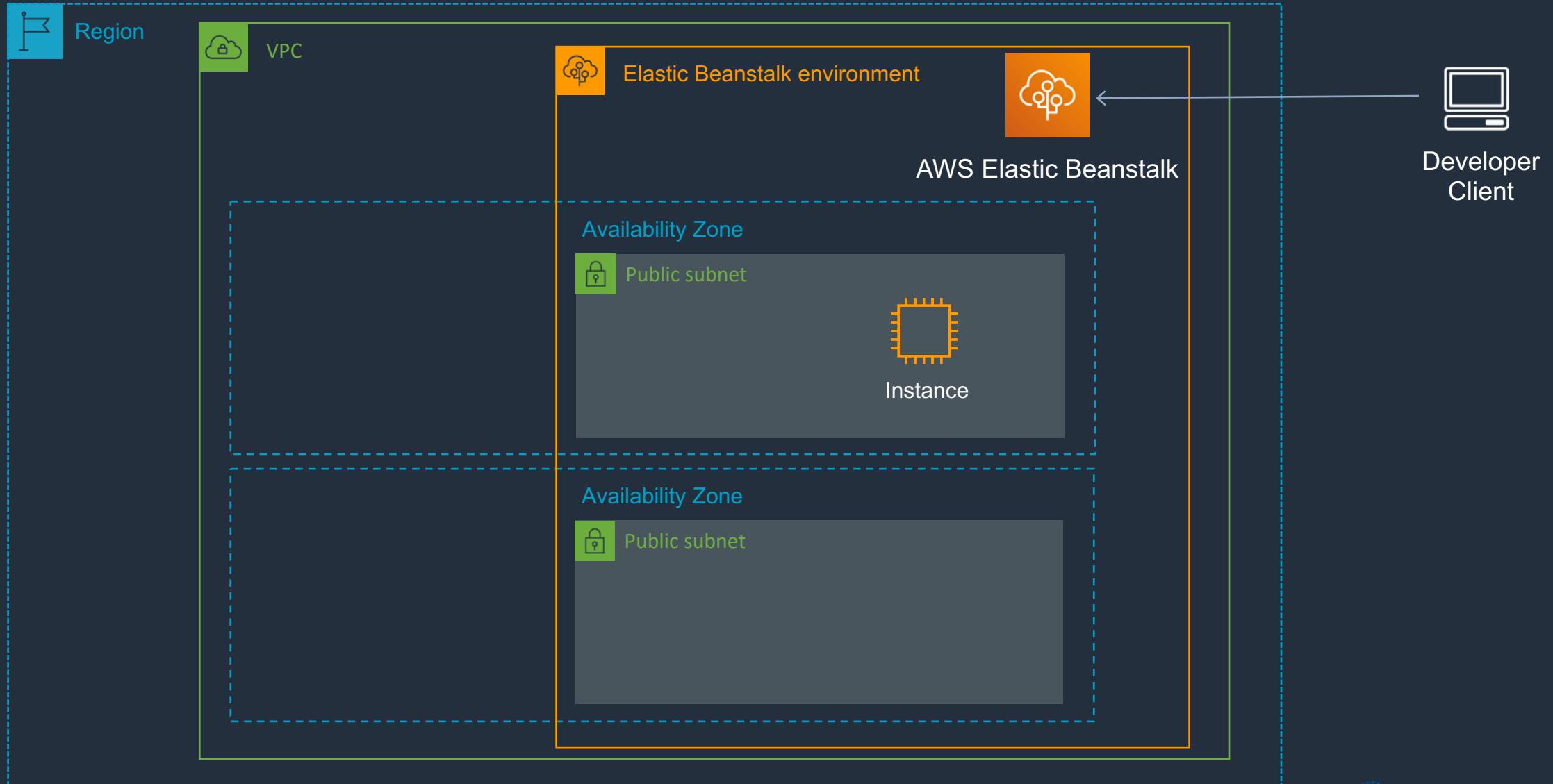
## AWS Elastic Beanstalk Environment Tiers

- Determines how Elastic Beanstalk provisions resources based on what the application is designed to do.
- Consists of Web Servers and Workers:
  - Web servers are standard applications that listen for and then process HTTP requests, typically over port 80.
  - Workers are specialized applications that have a background processing task that listens for messages on an Amazon SQS queue.
  - Workers should be used for long-running tasks.

# AWS Elastic Beanstalk Web Servers and Workers



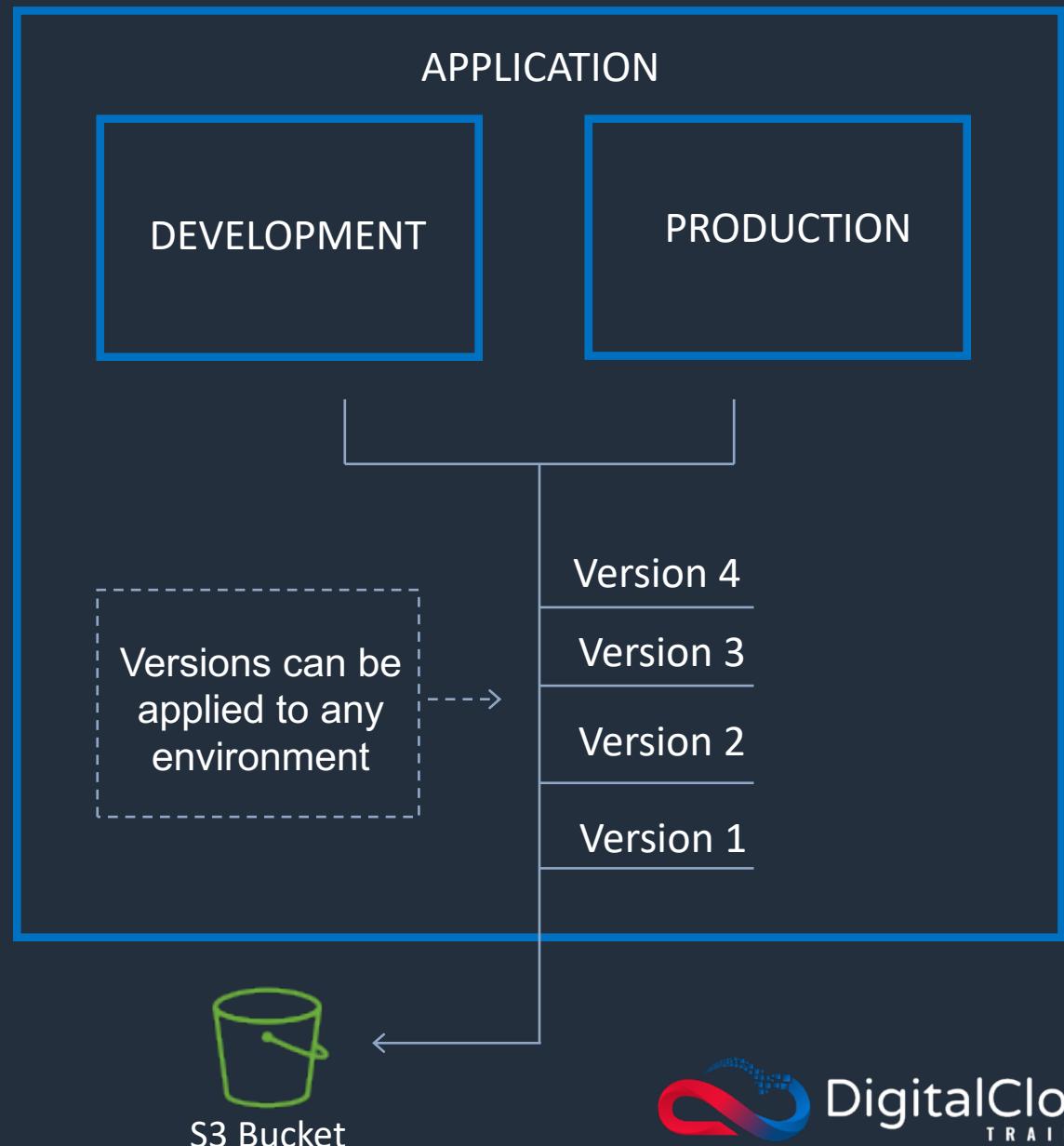
# AWS Elastic Beanstalk



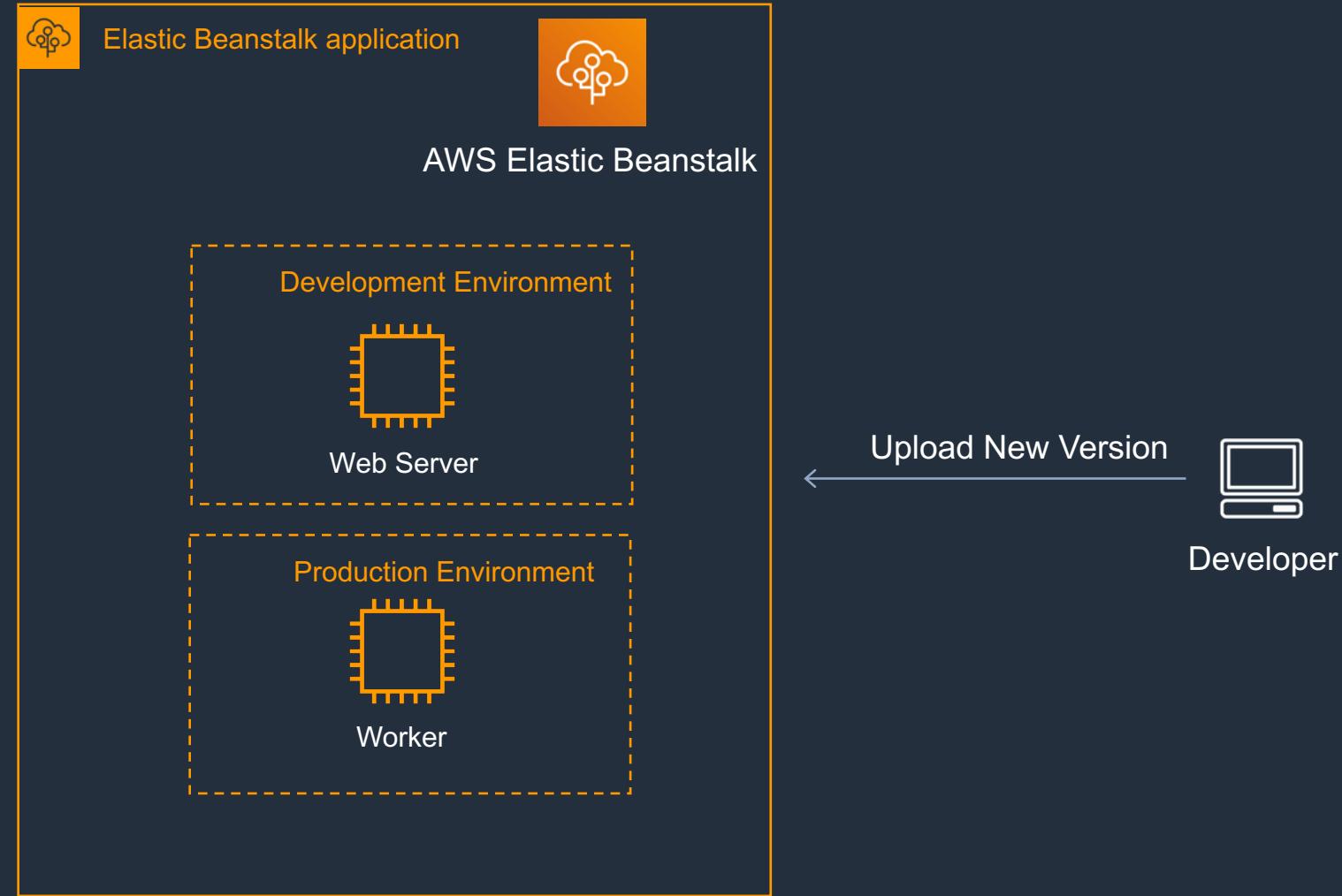
# AWS Elastic Beanstalk

## Core components:

- Application
- Application Versions
- Environments



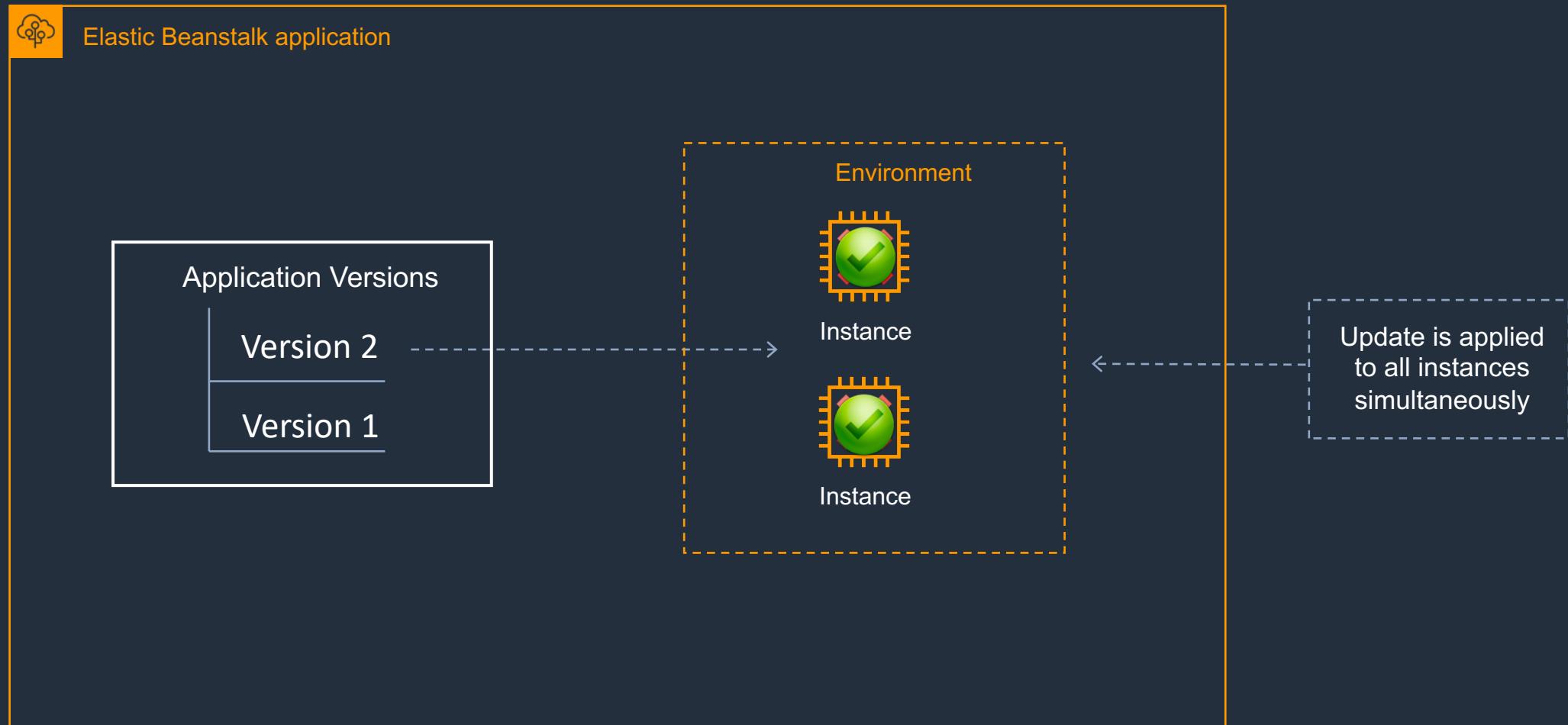
# AWS Elastic Beanstalk - Multiple Environments



# AWS Elastic Beanstalk Deployment Policies

- All at once:
  - Deploys the new version to all instances simultaneously.
- Rolling:
  - Update a batch of instances, and then move onto the next batch once the first batch is healthy.
- Rolling with additional batch:
  - Like Rolling but launches new instances in a batch ensuring that there is full availability.
- Immutable:
  - Launches new instances in a new ASG and deploys the version update to these instances before swapping traffic to these instances once healthy.
- Blue/green:
  - Create a new "stage" environment and deploy updates there.

# AWS Elastic Beanstalk – All at Once Update



## AWS Elastic Beanstalk – All at Once Update

- Deploys the new version to all instances simultaneously.
- All of your instances are out of service while the deployment takes place.
- Fastest deployment.
- Good for quick iterations in development environment.
- You will experience an outage while the deployment is taking place - not ideal for mission-critical systems.
- If the update fails, you need to roll back the changes by re-deploying the original version to all of your instances.
- No additional cost.

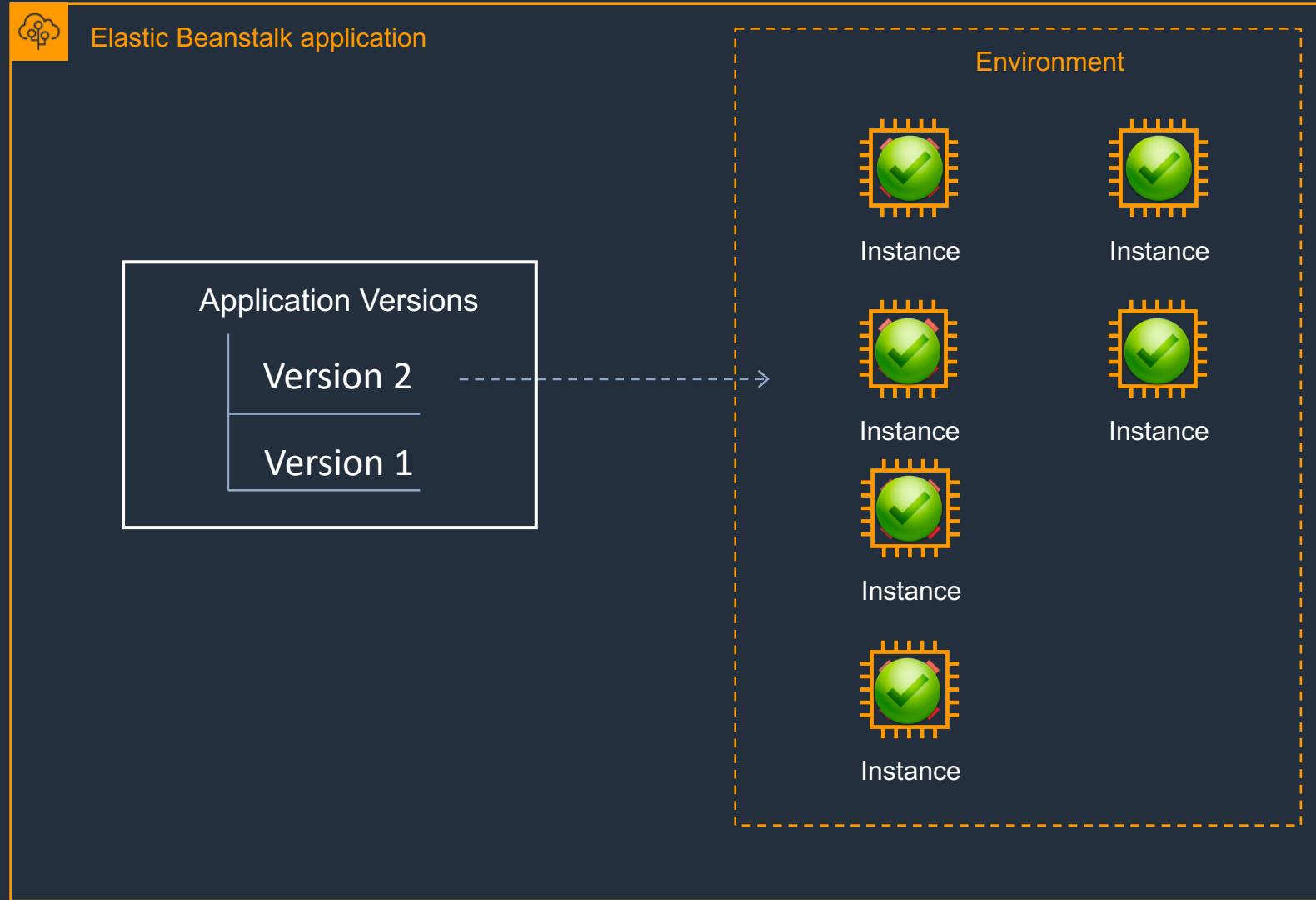
# AWS Elastic Beanstalk – Rolling Update



## AWS Elastic Beanstalk – Rolling Update

- Update a few instances at a time (batch), and then move onto the next batch once the first batch is healthy (downtime for 1 batch at a time).
- Application is running both versions simultaneously.
- Each batch of instances is taken out of service while the deployment takes place.
- Your environment capacity will be reduced by the number of instances in a batch while the deployment takes place.
- Not ideal for performance-sensitive systems.
- If the update fails, you need to perform an additional rolling update to roll back the changes.
- No additional cost.
- Long deployment time.

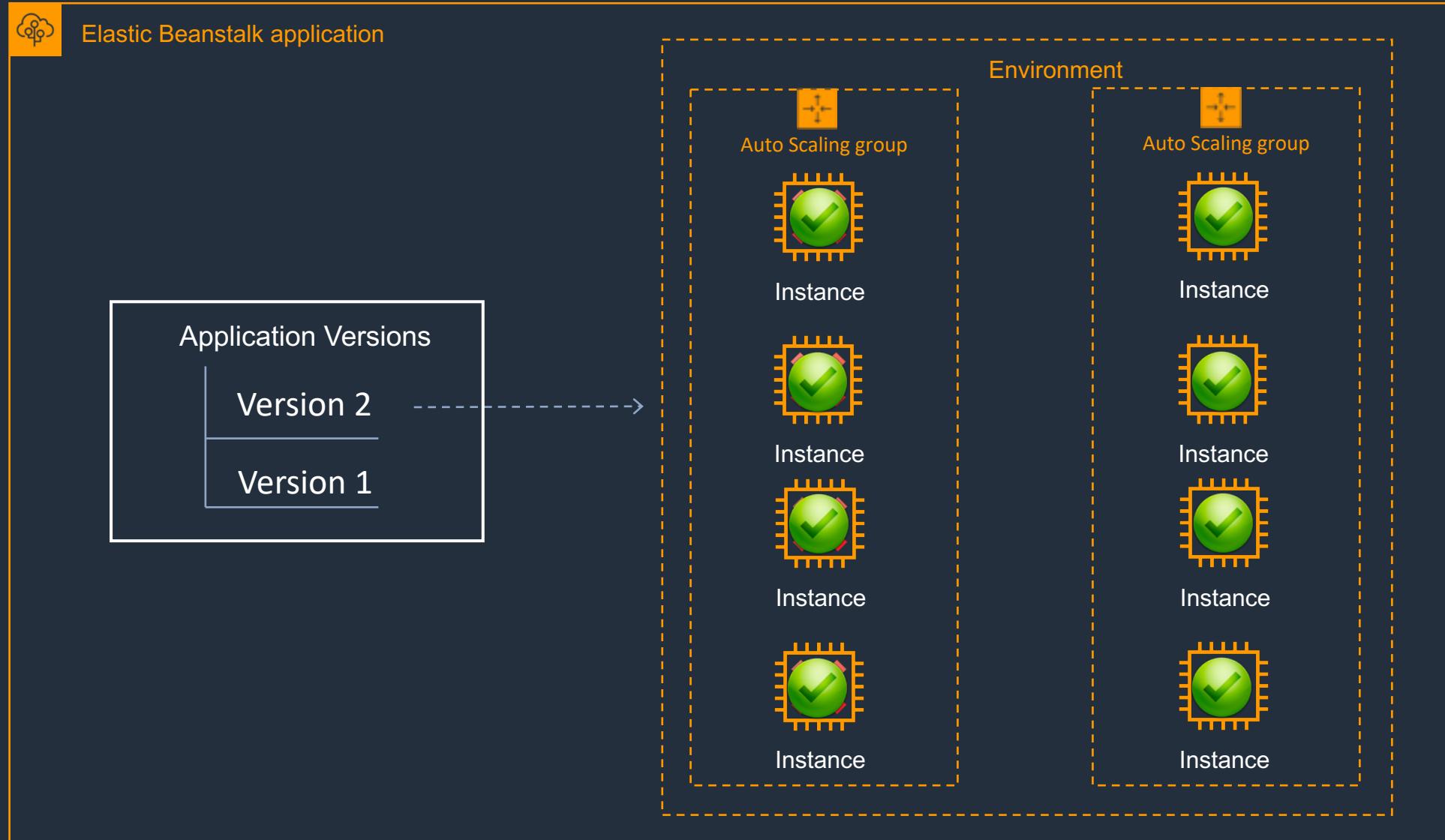
# AWS Elastic Beanstalk – Rolling with Additional Batch Update



## AWS Elastic Beanstalk – Rolling with Additional Batch Update

- Like Rolling but launches new instances in a batch ensuring that there is full availability.
- Application is running at capacity.
- Can set the batch size.
- Application is running both versions simultaneously.
- Small additional cost.
- Additional batch is removed at the end of the deployment.
- Longer deployment.
- Good for production environments.

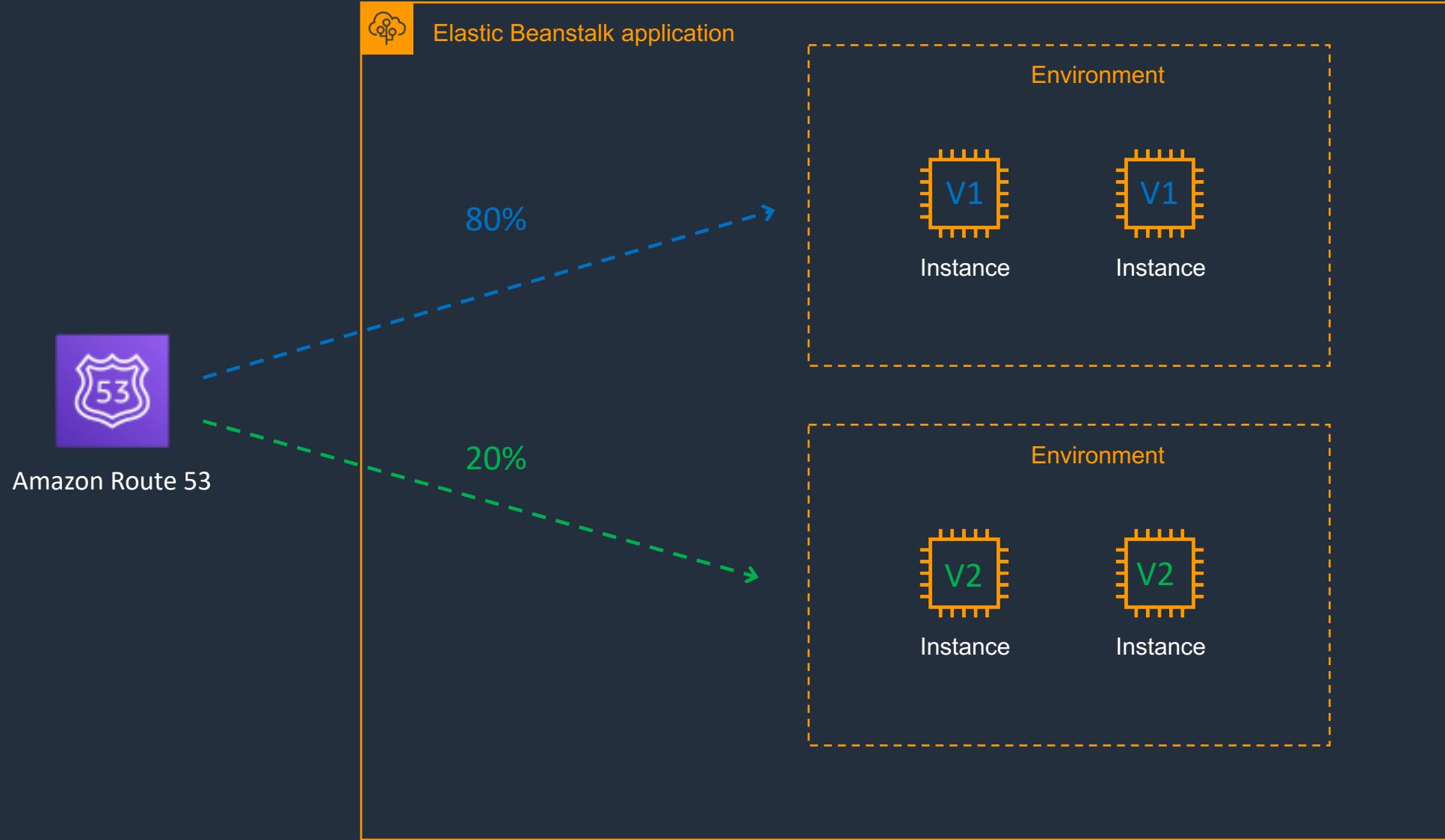
# AWS Elastic Beanstalk – Immutable Update



## AWS Elastic Beanstalk – Immutable Update

- Launches new instances in a new ASG and deploys the version update to these instances before swapping traffic to these instances once healthy.
- Zero downtime.
- New code is deployed to new instances using an ASG.
- High cost as double the number of instances running during updates.
- Longest deployment.
- Quick rollback in case of failures.
- Great for production environments.

# AWS Elastic Beanstalk – Blue/green



## AWS Elastic Beanstalk – Blue/Green Update

- This is not a feature within Elastic Beanstalk
- You create a new "staging" environment and deploy updates there.
- The new environment (green) can be validated independently and you can roll back if there are issues.
- Route 53 can be setup using weighted policies to redirect a percentage of traffic to the staging environment.
- Using Elastic Beanstalk, you can "swap URLs" when done with the environment test.
- Zero downtime.

## AWS Elastic Beanstalk – .ebextensions

- You can add AWS Elastic Beanstalk configuration files (.ebextensions) to your web application's source code to configure your environment and customize the AWS resources that it contains.
- Configuration files are YAML- or JSON-formatted documents with a .config file extension that you place in a folder named .ebextensions and deploy in your application source bundle.

```
option_settings:  
  aws:elasticbeanstalk:environment:  
    LoadBalancerType: network
```

## AWS Elastic Beanstalk – .ebextensions

- The option\_settings section of a configuration file defines values for configuration options.
- The Resources section lets you further customize the resources in your application's environment and define additional AWS resources beyond the functionality provided by configuration options.
- The other sections of a configuration file (packages, sources, files, users, groups, commands, container\_commands, and services) let you configure the EC2 instances that are launched in your environment.

## AWS Elastic Beanstalk – HTTPS

- Can assign a server certificate to your environment's load balancer :
  - Can be configured through the console.
  - This secures the connection between the client (app user) and the load balancer.
  - The backend connections between the load balancer and EC2 are not secured.
  - You can use AWS Certificate Manager (ACM) certificates.
  - Can also configure through .ebextensions:

`option_settings:`

`aws:elbv2:listener:443:`

`ListenerEnabled: 'true'`

`Protocol: HTTPS`

`SSLCertificateArns: arnXXX`

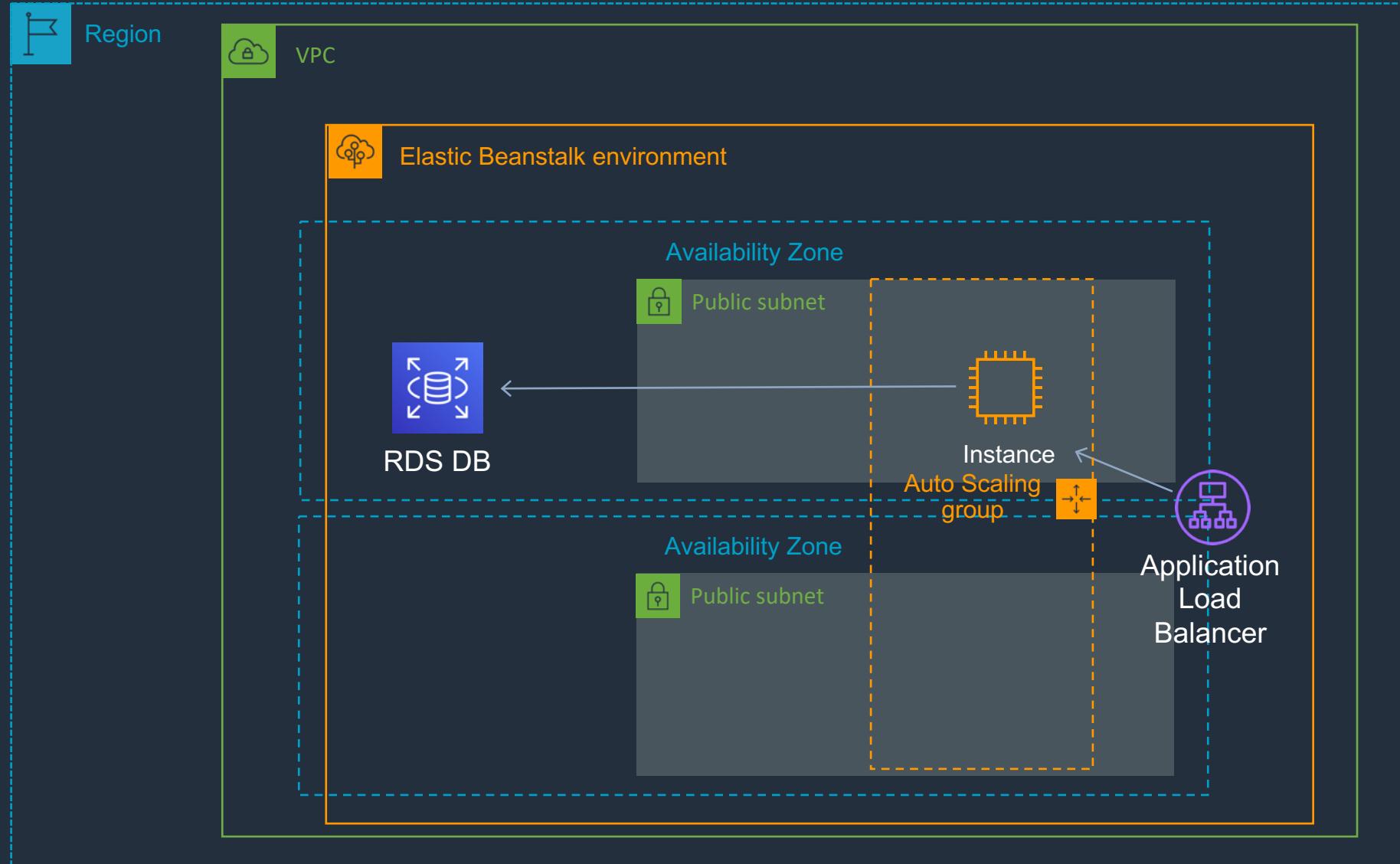
## AWS Elastic Beanstalk – HTTPS

- For end-to-end or single instance environments, configure instances to terminate HTTPS:
  - Must be configured through .ebextensions configuration files.

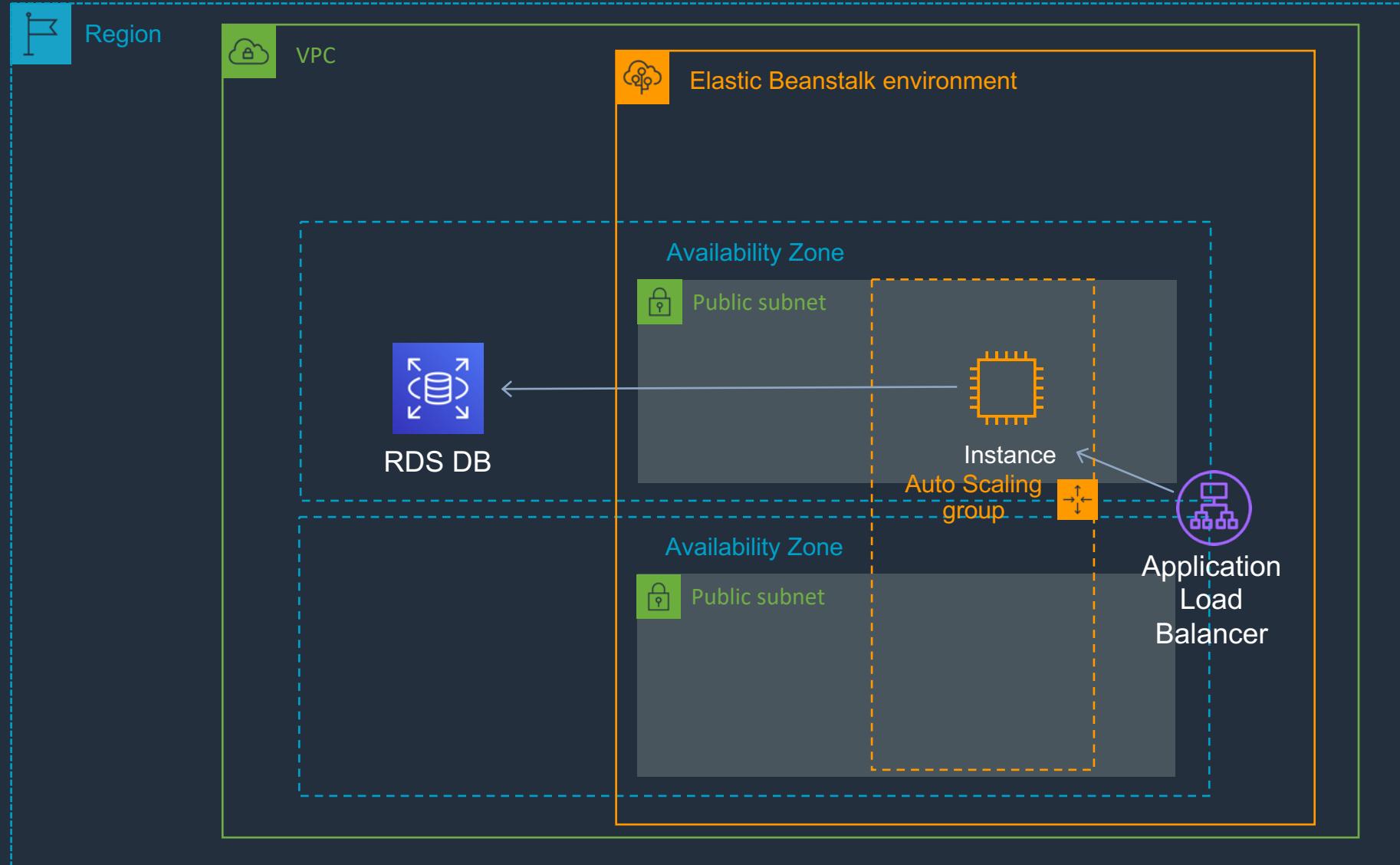
## AWS Elastic Beanstalk – HTTP to HTTPS Redirection

- To configure HTTP to HTTPS redirection, do one of the following:
- If your environment is load balanced – Configure your load balancer to terminate HTTPS.
- If you're running a single instance environment – Configure your application to terminate HTTPS connections at the instance.
- Configure your Amazon Elastic Compute Cloud (Amazon EC2) instances to redirect HTTP traffic to HTTPS (platform dependent). Code available on GitHub

# AWS Elastic Beanstalk and RDS



# AWS Elastic Beanstalk and RDS



## AWS Elastic Beanstalk and Amazon Relational Database Service (RDS)

- You can deploy Amazon RDS within an Elastic Beanstalk environment.
- However, if you terminate your Elastic Beanstalk environment you also lose the database.
- Use case is only for development environments, typically not suitable for production.
- For production it is preferable to create the Amazon RDS database outside of Elastic Beanstalk.

## AWS Elastic Beanstalk – Connecting to RDS

- When the environment update is complete, the DB instance's hostname and other connection information are available to your application through the following environment properties:
  - RDS\_HOSTNAME – The hostname of the DB instance.
  - RDS\_PORT – The port on which the DB instance accepts connections. The default value varies among DB engines.
  - RDS\_DB\_NAME – The database name, ebdb.
  - RDS\_USERNAME – The user name that you configured for your database.
  - RDS\_PASSWORD – The password that you configured for your database.

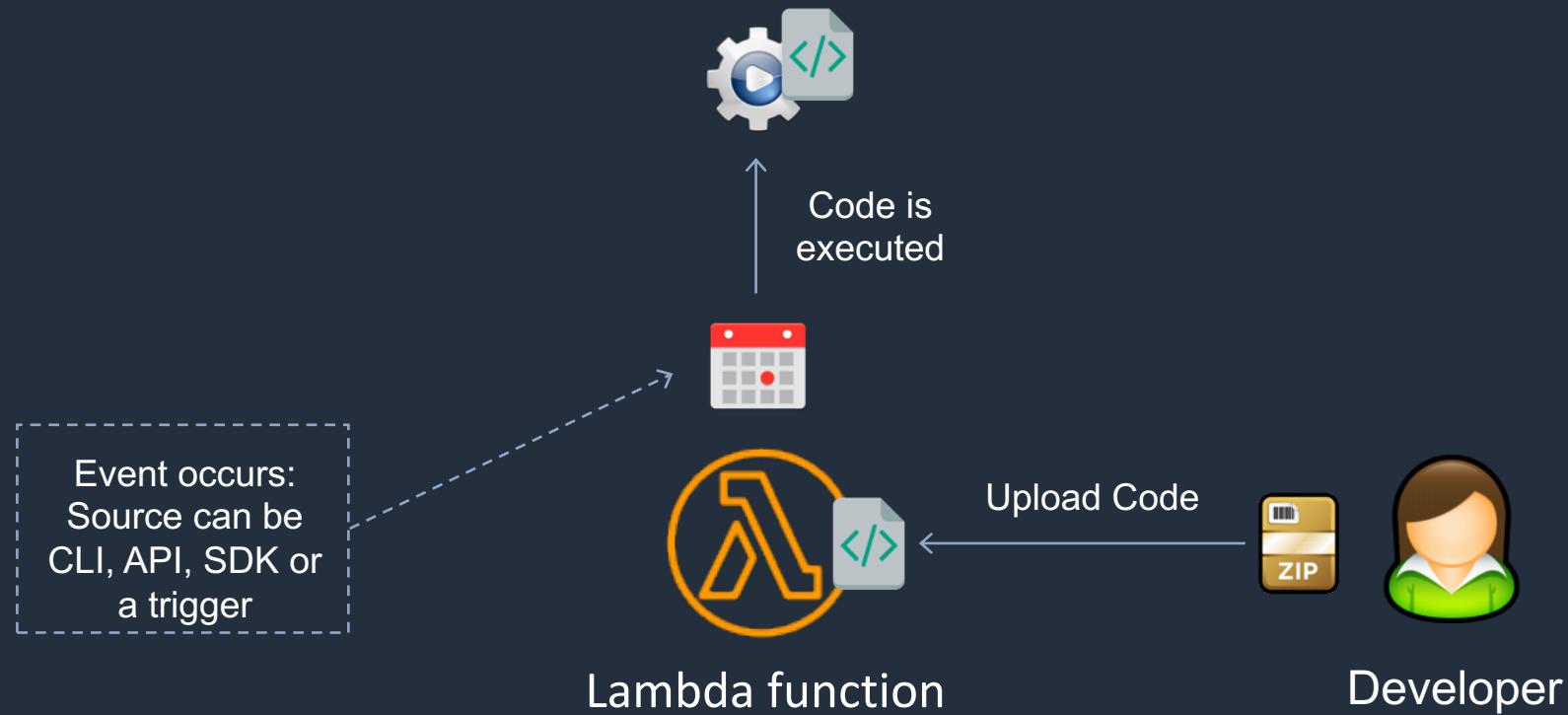
## AWS Elastic Beanstalk and Amazon Relational Database Service (RDS)

- Steps to migrate from RDS within a Beanstalk environment to standalone RDS:
  - Take a snapshot of the RDS DB.
  - Enable deletion protection on the RDS DB.
  - Create a new environment without an RDS DB and point applications to the existing RDS DB.
  - Perform a blue/green deployment and swap the new and old environments.
  - Terminate the old environment (RDS will not be deleted due to termination protection).
  - Delete the CloudFormation stack (will be in the DELETE\_FAILED state).

# SECTION 11

## Serverless Functions: AWS Lambda

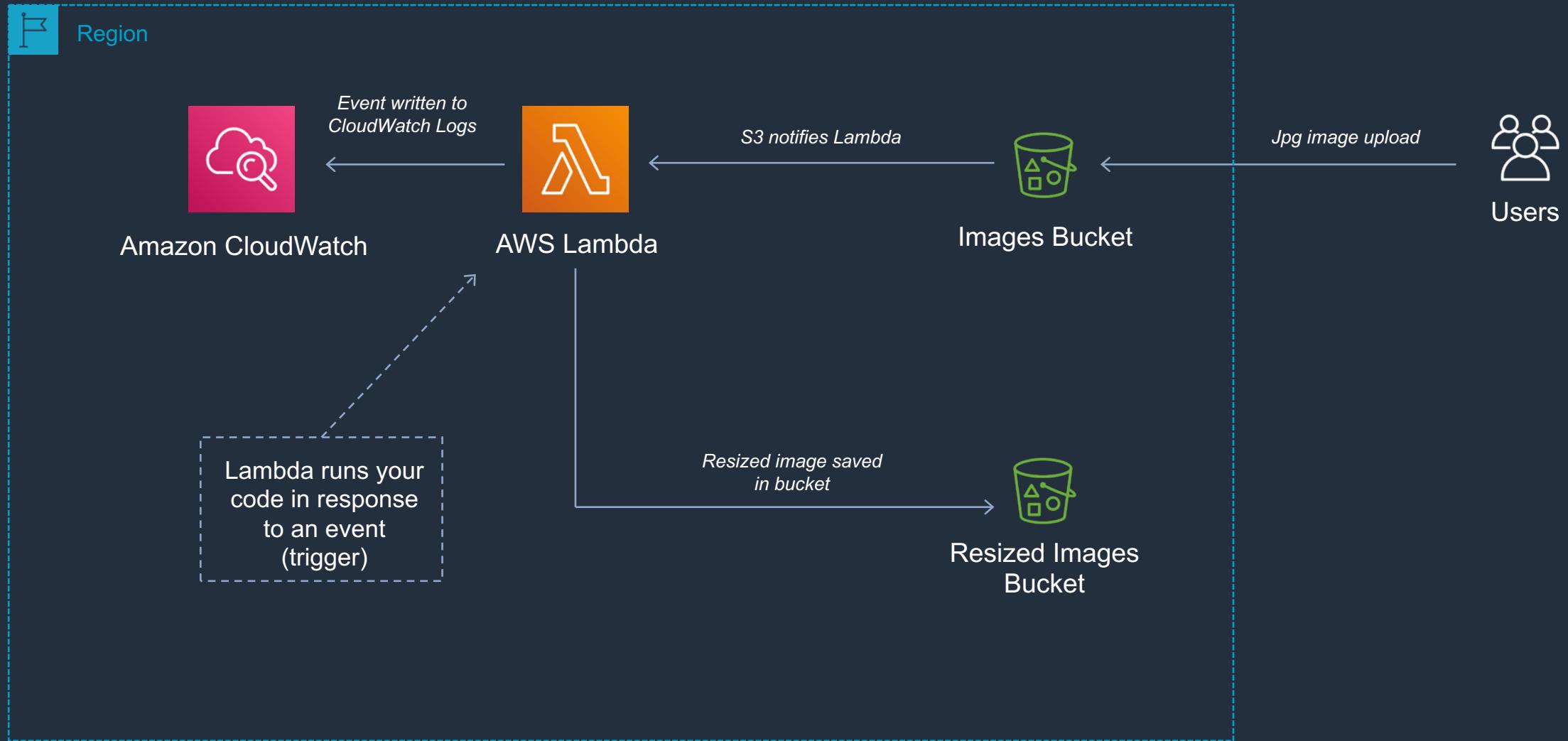
# AWS Lambda



## AWS Lambda – Overview

- AWS Lambda is a serverless compute service that runs your code in response to events and automatically manages the underlying compute resources for you.
- AWS Lambda can automatically run code in response to multiple events, such as HTTP requests via Amazon API Gateway, modifications to objects in Amazon S3 buckets, table updates in Amazon DynamoDB, and state transitions in AWS Step Functions.
- Lambda runs your code on high-availability compute infrastructure and performs all the administration of the compute resources, including server and operating system maintenance, capacity provisioning and automatic scaling, code and security patch deployment, and code monitoring and logging.
- Lambda is a “serverless” service.

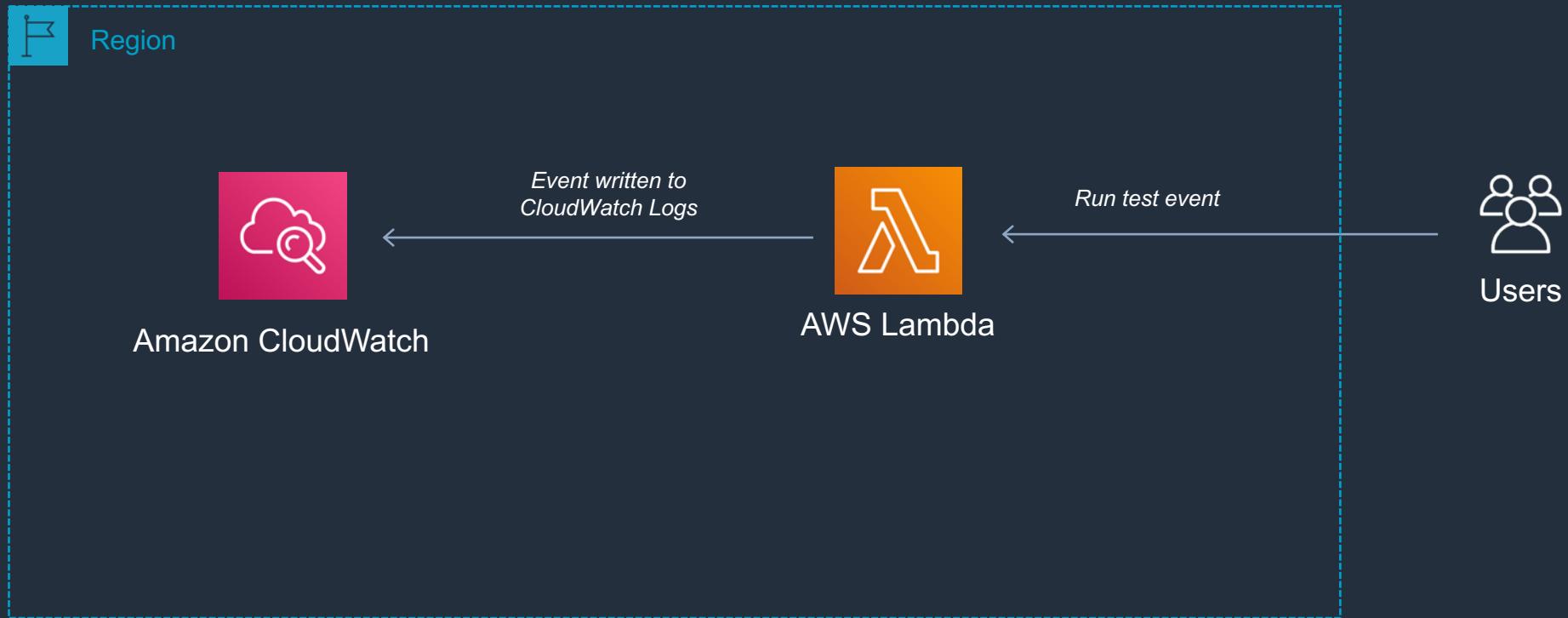
# AWS Lambda – Function to Resize Images



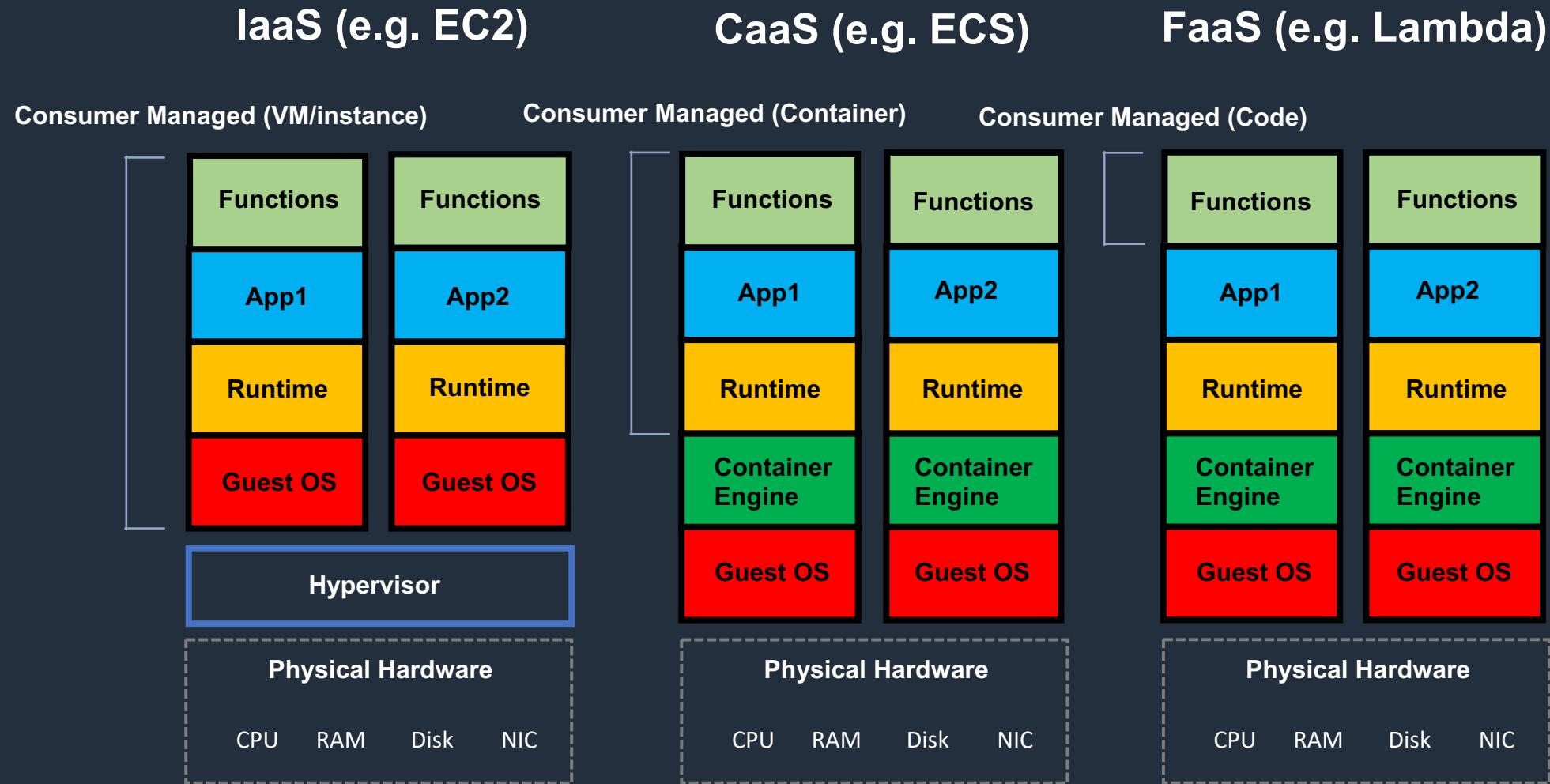
## AWS Lambda – Overview

- All you need to do is supply the code.
- You are charged based on the number of requests for your functions and the duration, the time it takes for your code to execute.
- The AWS Lambda free usage tier includes 1M free requests per month and 400,000 GB-seconds of compute time per month.
- The price depends on the amount of memory you allocate to your function.
- In the AWS Lambda resource model, you choose the amount of memory you want for your function, and are allocated proportional CPU power and other resources.
- An increase in memory size triggers an equivalent increase in CPU available to your function.

# AWS Lambda – Hello World



# Comparing IaaS, CaaS, and FaaS



# Comparing Compute Options

EC2	ECS (EC2 Launch Type)	ECS (Fargate Launch Type)	Lambda
You manage the operating system	You manage container instance (EC2) and the containers (tasks)	You manage the containers (tasks)	You manage the code
Scale vertically – more CPU/Mem/HDD or scale horizontally (automatic) with Auto Scaling	Manually add container instances or use ECS Services and EC2 Auto Scaling	AWS scales the cluster automatically	Lambda automatically scales concurrent executions up to default limit (1000)
Use for traditional applications and long running tasks	Use for microservices and batch use cases where you need containers and need to retain management of underlying platform	Use for microservices and batch use cases	Use for ETL, infrastructure automation, data validation, mobile backends
No timeout issues	No timeout issues	No timeout issues	Limited to 900 seconds execution time for single execution (3 second default)
Pay for instance run time based on family/type	Pay for instance run time based on family/type	Pay for container run time based on allocated resources	Pay only for execution time based on memory allocation

## AWS Lambda – Invoking Functions

- You can invoke Lambda functions directly with the Lambda console, the Lambda API, the AWS SDK, the AWS CLI, and AWS toolkits.
- You can also configure other AWS services to invoke your function, or you can configure Lambda to read from a stream or queue and invoke your function.
- When you invoke a function, you can choose to invoke it synchronously or asynchronously.
- Other AWS services and resources invoke your function directly.
- For example, you can configure CloudWatch Events to invoke your function on a timer, or you can configure Amazon S3 to invoke your function when an object is created.
- Each service varies in the method it uses to invoke your function, the structure of the event, and how you configure it.

## AWS Lambda – Synchronous Invocation

- You wait for the function to process the event and return a response.
- When you invoke a function synchronously, Lambda runs the function and waits for a response.
- When the function execution ends, Lambda returns the response from the function's code with additional data, such as the version of the function that was executed.
- To invoke a function synchronously with the AWS CLI, use the **invoke** command.

```
$ aws lambda invoke --function-name my-function --payload '{ "key": "value" }' response.json
{
  "ExecutedVersion": "$LATEST",
  "StatusCode": 200
}
```

## AWS Lambda – Synchronous Invocation

```
$ aws lambda invoke --function-name my-function --payload '{ "key": "value" }' response.json
{
  "ExecutedVersion": "$LATEST",
  "StatusCode": 200
}
```

- The payload is a string that contains an event in JSON format (base64 encoded).
- The name of the file where the AWS CLI writes the response from the function is response.json.
- To get logs for an invocation from the command line, use the --log-type option. The response includes a LogResult field that contains up to 4 KB of base64-encoded logs from the invocation.

```
$ aws lambda invoke --function-name my-function out --log-type Tail
{
  "StatusCode": 200,
  "LogResult": "U1RBUlQgUmVxdWVzdElk0iA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
  "ExecutedVersion": "$LATEST"
}
```

## AWS Lambda – Asynchronous Invocation

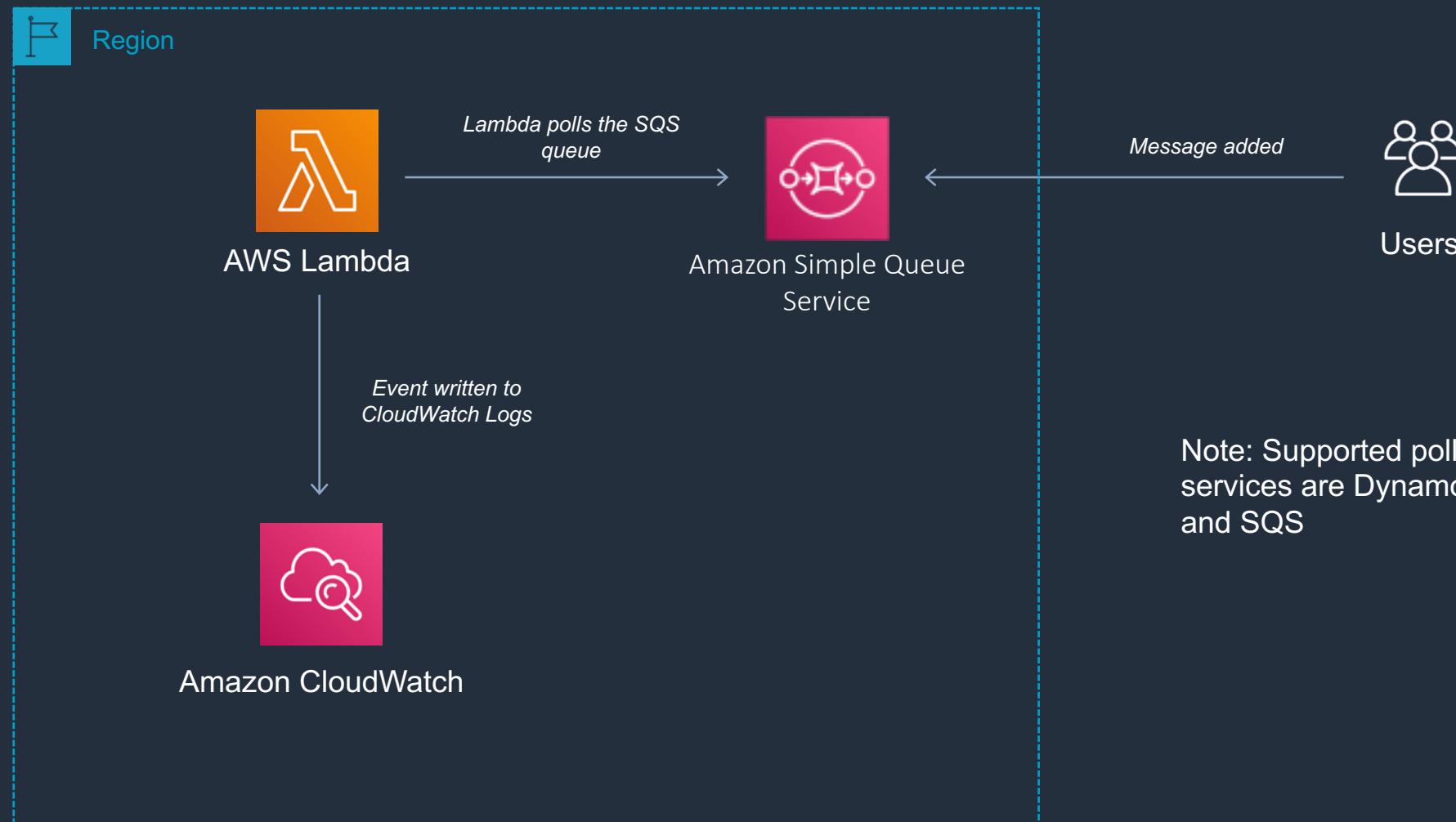
- When you invoke a function asynchronously, you don't wait for a response from the function code.
- For asynchronous invocation, Lambda handles retries and can send invocation records to a destination.
- For asynchronous invocation, Lambda places the event in a queue and returns a success response without additional information.
- To invoke a function asynchronously, set the invocation type parameter to **Event**.

## AWS Lambda – Asynchronous Invocation

- To invoke a function asynchronously, set the invocation type parameter to **Event**.

```
$ aws lambda invoke --function-name my-function --invocation-type Event --payload '{ "key": "value" }' response.json
{
  "StatusCode": 202
}
```

# AWS Lambda – DynamoDB Event Source Mapping



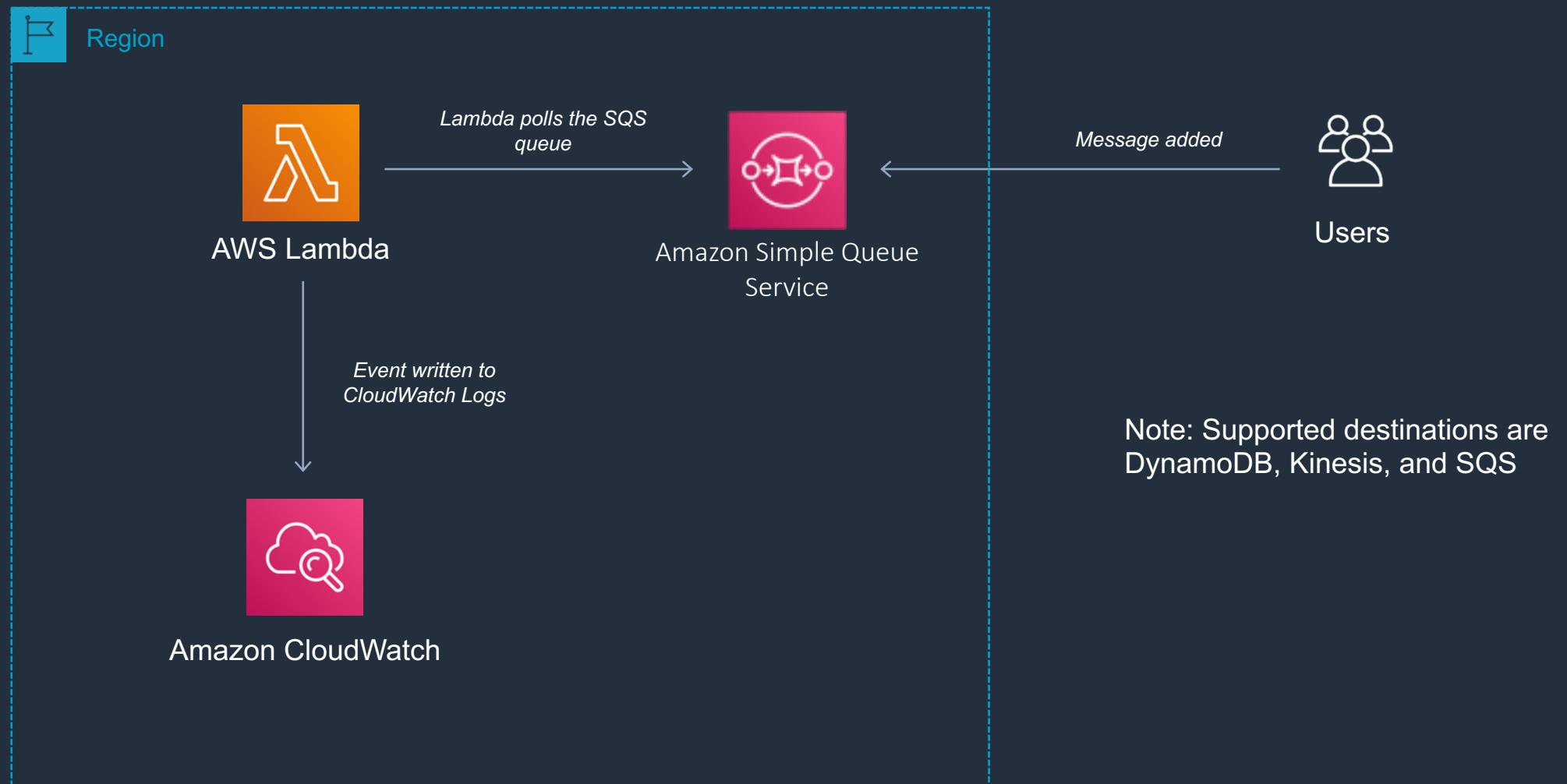
## AWS Lambda – Event Source Mappings

- To process items from a stream or queue, you can create an event source mapping.
- An event source mapping is an AWS Lambda resource that reads from an event source and invokes a Lambda function.
- You can use event source mappings to process items from a stream or queue in services that don't invoke Lambda functions directly.
- An event source mapping uses permissions in the function's execution role to read and manage items in the event source.
- Permissions, event structure, settings, and polling behavior vary by event source.

## AWS Lambda – Event Source Mappings

- The configuration of the event source mapping for stream and queue-based services (DynamoDB, Kinesis), and Amazon SQS, is made on the Lambda side.
- Note: for other services such as Amazon S3 and SNS, the function is invoked asynchronously and the configuration is made on the source (S3/SNS) rather than Lambda.

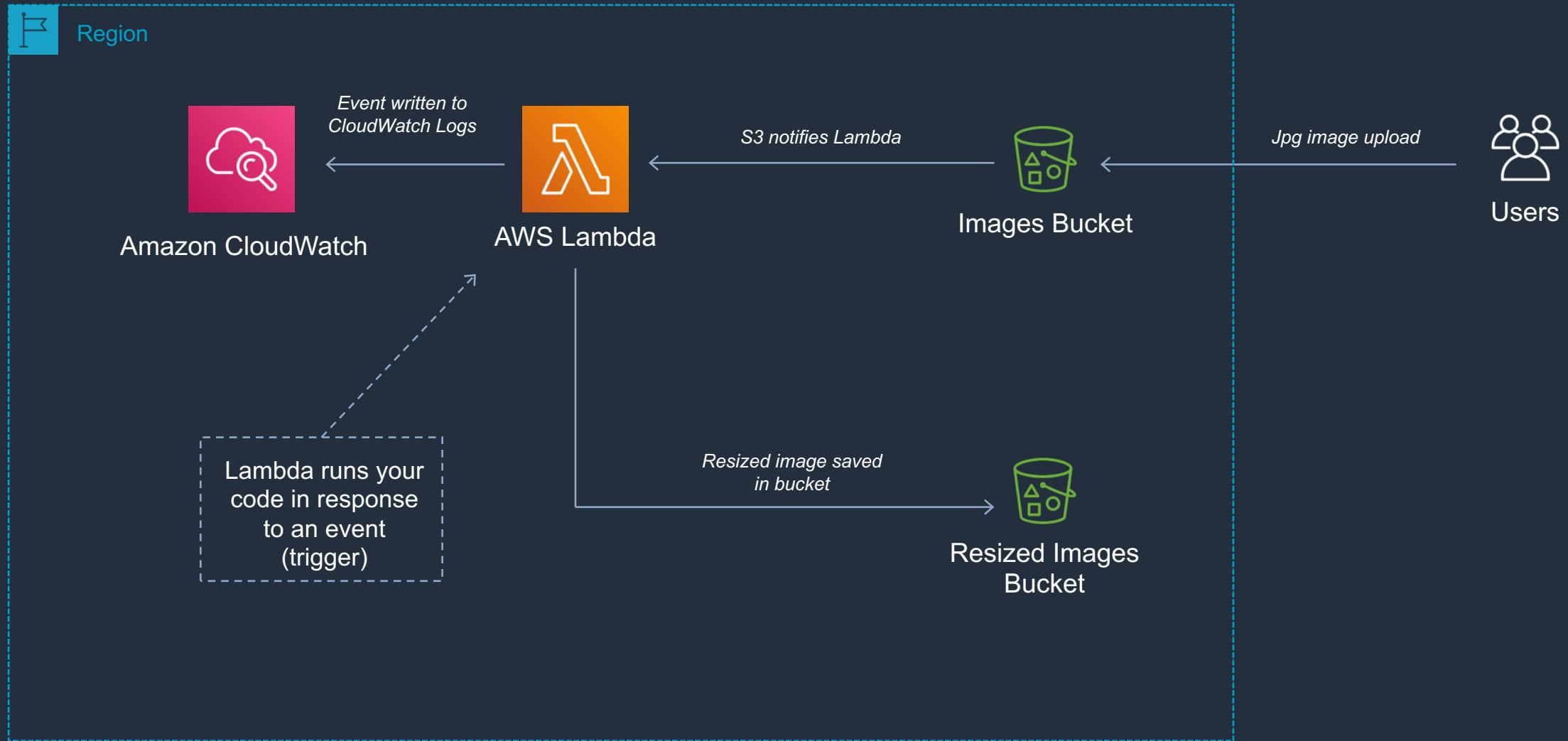
# AWS Lambda – DynamoDB Event Source Mapping



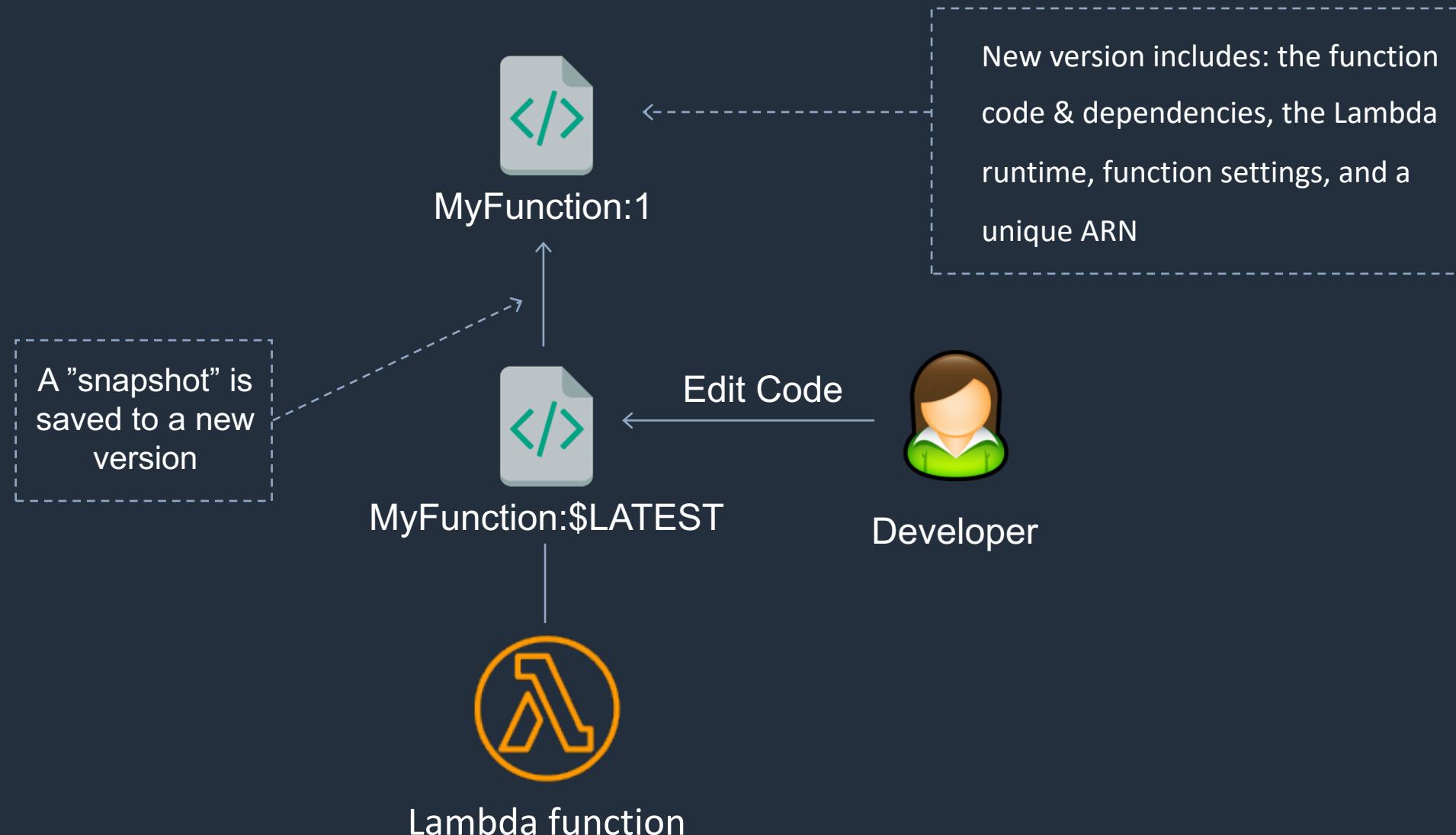
## AWS Lambda – Event Notifications

- You can use Lambda to process event notifications from Amazon Simple Storage Service.
- Amazon S3 can send an event to a Lambda function when an object is created or deleted.
- You configure notification settings on a bucket, and grant Amazon S3 permission to invoke a function on the function's resource-based permissions policy.
- Amazon S3 invokes your function asynchronously with an event that contains details about the object.

# AWS Lambda – Function to Resize Images



# AWS Lambda – Versions



## AWS Lambda – Versions

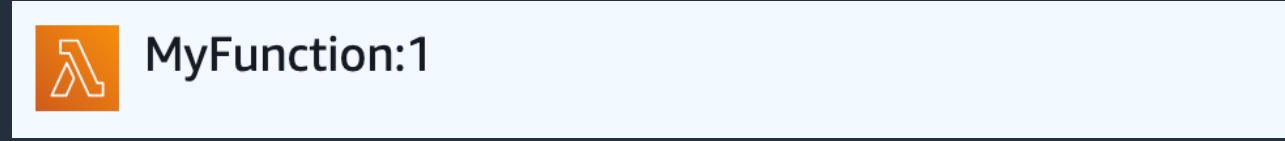
- Versioning means you can have multiple versions of your function.
- You can use versions to manage the deployment of your AWS Lambda functions. For example, you can publish a new version of a function for beta testing without affecting users of the stable production version.
- The function version includes the following information:
  - The function code and all associated dependencies.
  - The Lambda runtime that executes the function.
  - All of the function settings, including the environment variables.
  - A unique Amazon Resource Name (ARN) to identify this version of the function.

## AWS Lambda – Versions

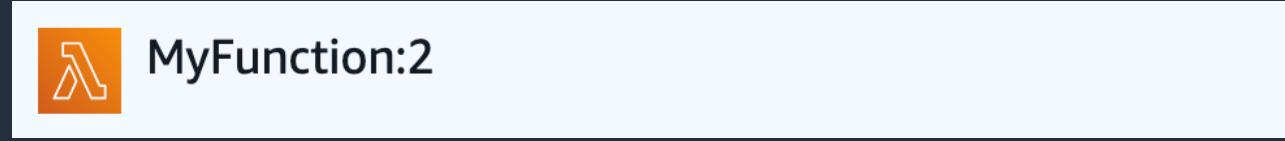
- You work on \$LATEST which is the latest version of the code - this is mutable (changeable).



- When you're ready to publish a Lambda function you create a version (these are numbered).



- Numbered versions are assigned a number starting with 1 and subsequent versions are incremented by 1.



- Versions are immutable (code cannot be edited).

## AWS Lambda – Versions

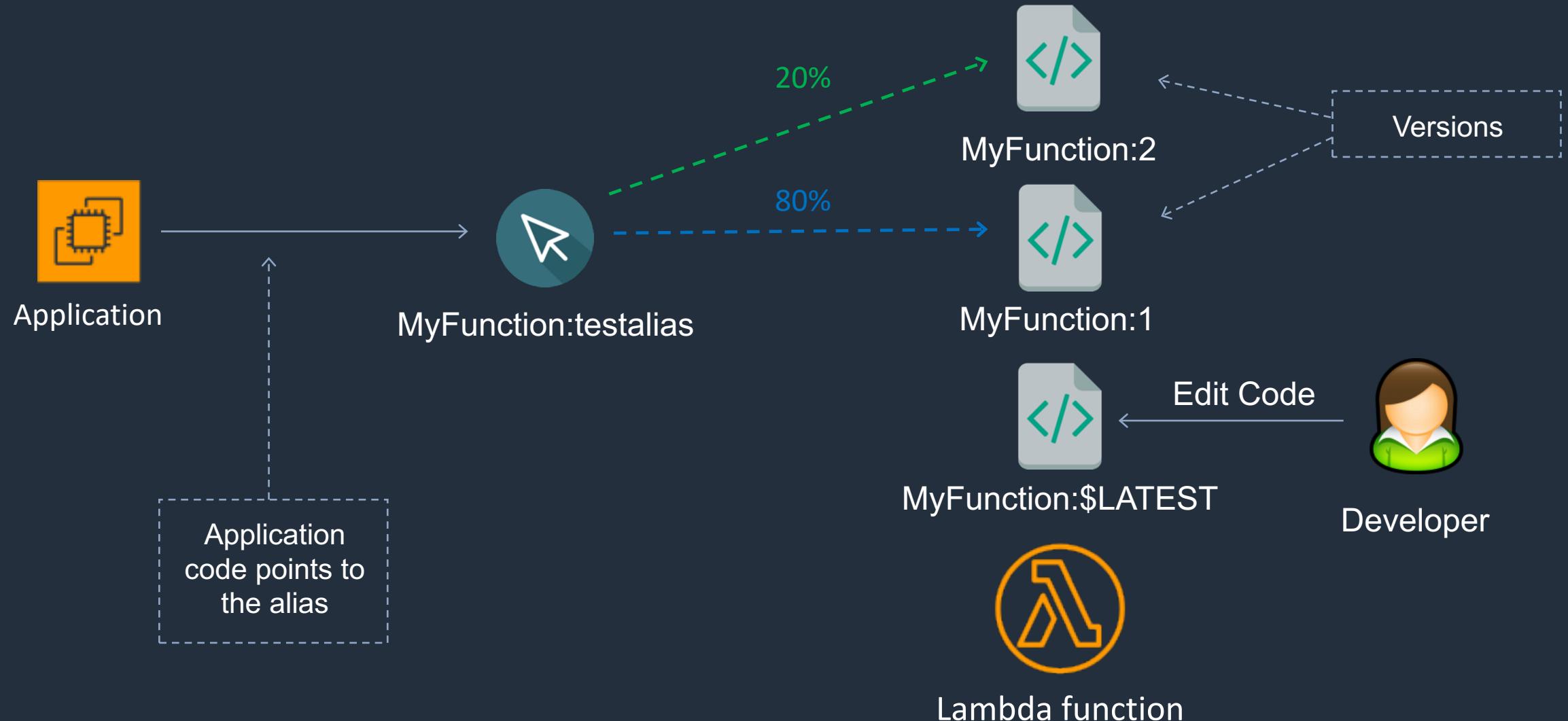
- Each version has it's own ARN.

**ARN - arn:aws:lambda:ap-southeast-2:515148227241:function:MyFunction:\$LATEST**

**ARN - arn:aws:lambda:ap-southeast-2:515148227241:function:MyFunction:1**

- Because different versions have unique ARNs this allows you to effectively manage them for different environments like Production, Staging or Development.
- A qualified ARN has a version suffix.
- An unqualified ARN does not have a version suffix.
- You cannot create an alias from an unqualified ARN.

# AWS Lambda – Aliases



# AWS Lambda – Aliases

- Lambda aliases are pointers to a specific Lambda version.
  - Using an alias you can invoke a function without having to know which version of the function is being referenced.
  - Aliases are mutable.

## Create a new alias

An alias is a pointer to one or two versions. Choose each version that you want the alias to point to.

Name\*

Description

Version\*

1	▼	Weight: 80%
---	---	-------------

You can shift traffic between two versions, based on weights (%) that you assign. Click [here](#) to learn more.

Additional version	Weight
2	20 %

## AWS Lambda – Aliases

- Aliases also have static ARNs but can point to any version of the same function.



**ARN - arn:aws:lambda:ap-southeast-2:515148227241:function:MyFunction:testalias**

- Aliases enable stable configuration of event triggers / destinations.
- Aliases enable blue / green deployment by assigning weights to Lambda version (doesn't work for \$LATEST, you need to create an alias for \$LATEST).

## AWS Lambda – Lambda Handler

- A handler is a function which Lambda will invoke to execute your code - it is an entry point.
- When you create a Lambda function, you specify a handler that AWS Lambda can invoke when the service executes the function on your behalf.
- You define a Lambda function handler as an instance or static method in a class.

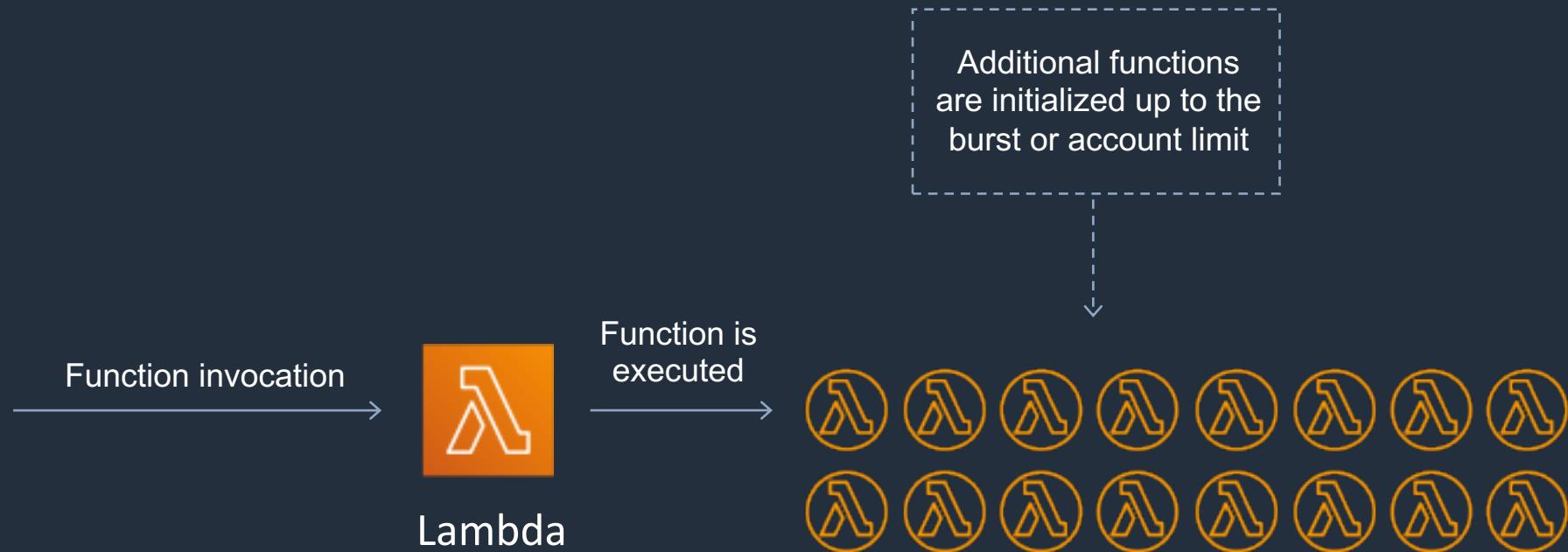
## AWS Lambda – Function Dependencies

- If your Lambda function depends on external libraries such as AWS X-Ray SDK, database clients etc. you need to install the packages with the code and zip it all up.
  - For Node.js use npm & "node modules" directory.
  - For Python use pip -- target options.
  - For Java include the relevant .jar files.
- Upload the zip file straight to Lambda if it's less than 50MB, otherwise upload to S3.
- Native libraries work: they need to be compiled on Amazon Linux.
- AWS SDK comes with every Lambda function by default.

## AWS Lambda – Concurrency

- The first time you invoke your function, AWS Lambda creates an instance of the function and runs its handler method to process the event.
- When the function returns a response, it stays active and waits to process additional events.
- If you invoke the function again while the first event is being processed, Lambda initializes another instance, and the function processes the two events concurrently.

# AWS Lambda - Concurrency



## AWS Lambda – Concurrency

- Burst Concurrency Limits:
  - 3000 – US West (Oregon), US East (N. Virginia), Europe (Ireland).
  - 1000 – Asia Pacific (Tokyo), Europe (Frankfurt).
  - 500 – Other Regions.
- After the initial burst, your functions' concurrency can scale by an additional 500 instances each minute.
- This continues until the account limit (default 1000 exec/sec is reached).

## AWS Lambda – Throttling

- Each invocation over the concurrency limit will trigger a throttle.
- TooManyRequestsException may be experienced if the concurrent execution limit is exceeded.
- You may receive a HTTP status code: 429 and the message is "Request throughput limit exceeded".
- Throttle behavior:
  - For synchronous invocations returns throttle error 429.
  - For asynchronous invocations retries automatically (twice) then goes to a Dead Letter Queue (DLQ).
- A DLQ can be an SNS topic or SQS queue.
- The original event payload is sent to the DLQ.

## AWS Lambda – Reserved Concurrency

- You can set a reserved concurrency at the function level to guarantee a set number of concurrent executions will be available for a critical function.
- You can reserve up to the Unreserved account concurrency value that is shown in the console, minus 100 for functions that don't have reserved concurrency.
- To throttle a function, set the reserved concurrency to zero. This stops any events from being processed until you remove the limit.

## AWS Lambda –Provisioned Concurrency

- When provisioned concurrency is allocated, the function scales with the same burst behavior as standard concurrency.
- After it's allocated, provisioned concurrency serves incoming requests with very low latency.
- When all provisioned concurrency is in use, the function scales up normally to handle any additional requests.
- Application Auto Scaling takes this a step further by providing autoscaling for provisioned concurrency.
- Provisioned concurrency runs continually and is billed in addition to standard invocation costs.

## AWS Lambda – Success and Failure Destinations

- Lambda asynchronous invocations can put an event or message on Amazon Simple Notification Service (SNS), Amazon Simple Queue Service (SQS), or Amazon EventBridge for further processing.
- With Destinations, you can route asynchronous function results as an execution record to a destination resource without writing additional code.
- An execution record contains details about the request and response in JSON format including version, timestamp, request context, request payload, response context, and response payload.
- For each execution status such as Success or Failure you can choose one of four destinations:
  - Another Lambda function.
  - Amazon Simple Notification Service (SNS).
  - Amazon Simple Queue Service (SQS).
  - Amazon EventBridge.

## AWS Lambda – Dead Letter Queue

- You can configure a dead letter queue (DLQ) on AWS Lambda to give you more control over message handling for all asynchronous invocations, including those delivered via AWS events (S3, SNS, IoT, etc).
- A dead-letter queue saves discarded events for further processing. A dead-letter queue acts the same as an on-failure destination in that it is used when an event fails all processing attempts or expires without being processed.
- However, a dead-letter queue is part of a function's version-specific configuration, so it is locked in when you publish a version.
- You can setup a DLQ by configuring the 'DeadLetterConfig' property when creating or updating your Lambda function.

## AWS Lambda – Monitoring and Logging

- Performance metrics are displayed in Amazon CloudWatch
- Execution logs are stored in Amazon CloudWatch Logs.
- Must ensure a Lambda function execution role has permissions (IAM) that allows writes to CloudWatch Logs.

```
1 1 {
2   "Version": "2012-10-17",
3   "Statement": [
4     {
5       "Effect": "Allow",
6       "Action": "logs:CreateLogGroup",
7       "Resource": "arn:aws:logs:ap-southeast-2:515148227241:*log-group:/aws/lambda/MyFunction:*"
8     },
9     {
10       "Effect": "Allow",
11       "Action": [
12         "logs:CreateLogStream",
13         "logs:PutLogEvents"
14       ],
15       "Resource": [
16         "arn:aws:logs:ap-southeast-2:515148227241:log-group:/aws/lambda/MyFunction:*"
17       ]
18     }
19   ]
20 }
```

## AWS Lambda – Tracing with X-Ray

- You can use AWS X-Ray to visualize the components of your application, identify performance bottlenecks, and troubleshoot requests that resulted in an error.
- Your Lambda functions send trace data to X-Ray, and X-Ray processes the data to generate a service map and searchable trace summaries.
- The AWS X-Ray Daemon is a software application that gathers raw segment data and relays it to the AWS X-Ray service.
- The daemon works in conjunction with the AWS X-Ray SDKs so that data sent by the SDKs can reach the X-Ray service.
- When you trace your Lambda function, the X-Ray daemon automatically runs in the Lambda environment to gather trace data and send it to X-Ray.
- Must have permissions to write to X-Ray in the execution role.

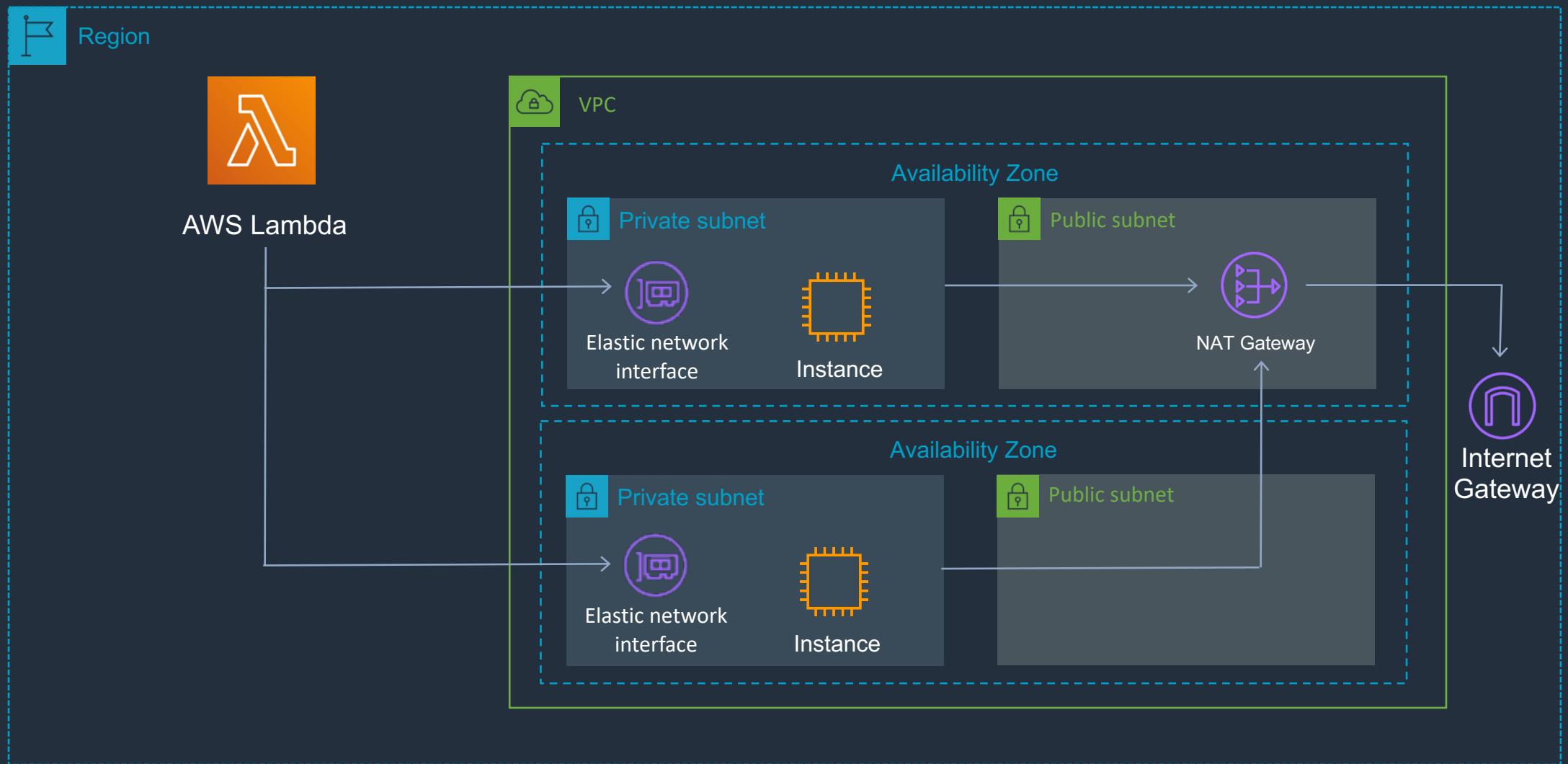
## AWS Lambda – Deploy Lambda through AWS CloudFormation

- The function code zip file must be stored in Amazon S3
- The S3 bucket must be in the same region where you're running CloudFormation.
- Also need a CloudFormation template file.

## AWS Lambda in a Virtual Private Cloud (VPC)

- You can configure a function to connect to private subnets in a virtual private cloud (VPC) in your account.
- Connect your function to the VPC to access private resources during execution.
- To enable this, you need to allow the function to connect to the private subnet.
- Lambda needs the following VPC configuration information so that it can connect to the VPC:
  - Private subnet ID.
  - Security Group ID (with required access).
- Lambda uses this information to setup an Elastic Network Interface (ENI) using an available IP address from your private subnet.

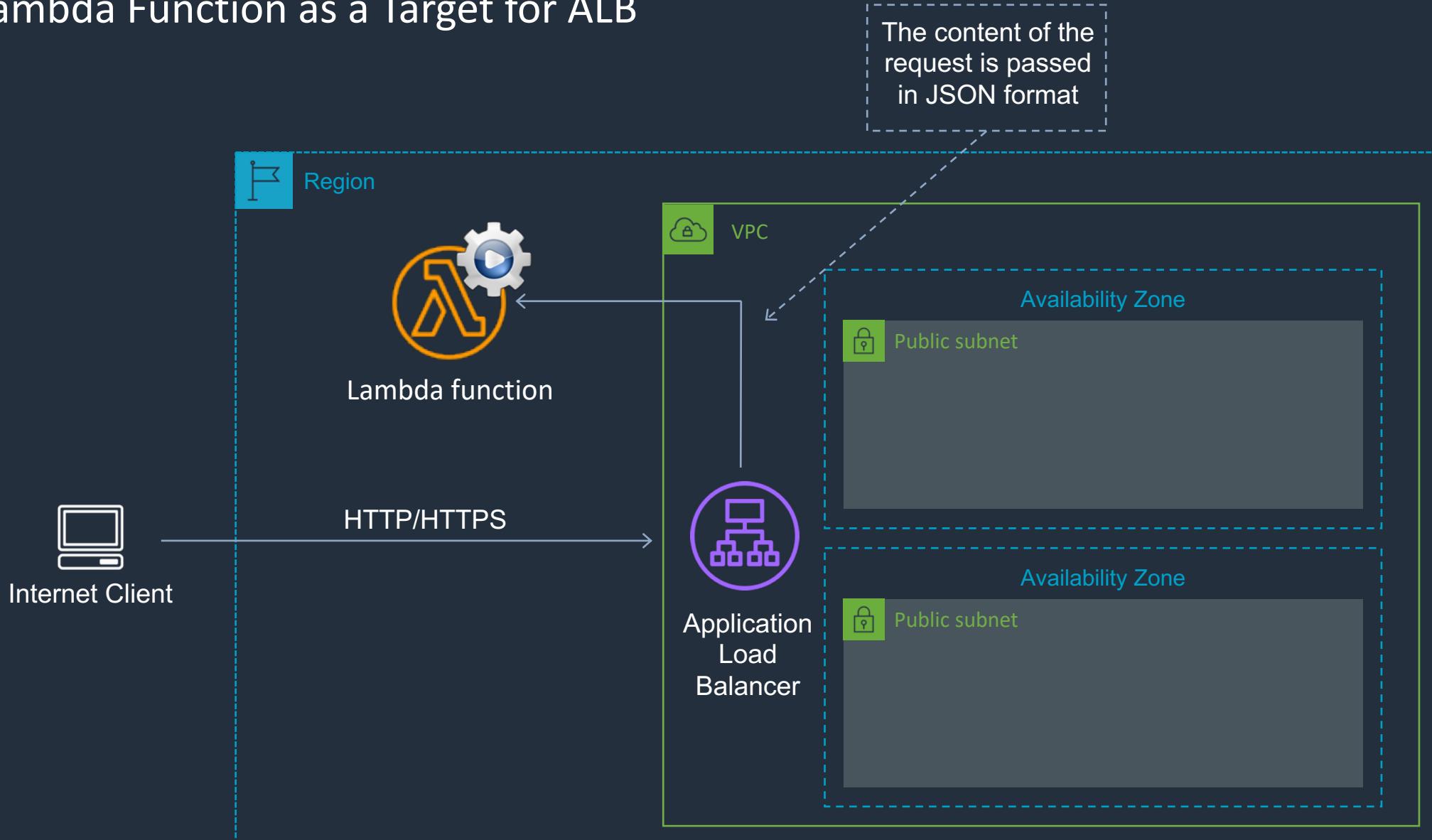
# AWS Lambda in a Virtual Private Cloud (VPC)



## AWS Lambda in a Virtual Private Cloud (VPC)

- You must connect to a private subnet with a NAT Gateway for Internet access (no public IP).
- Careful with DNS resolution of public hostnames as it could add to function running time (cost).
- Cannot connect to a dedicated tenancy VPC.
- Only connect to a VPC if you need to, can slow down function execution.
- Lambda uses your function's permissions to create and manage network interfaces. To connect to a VPC, your function's execution role must have the following permissions:
  - ec2:CreateNetworkInterface
  - ec2:DescribeNetworkInterfaces
  - ec2:DeleteNetworkInterface
- These permissions are included in the AWSLambdaVPCAccessExecutionRole managed policy.

# AWS Lambda Function as a Target for ALB



## AWS Lambda – Functions as Targets for Application Load Balancers (ALB)

- Application Load Balancers (ALBs) support AWS Lambda functions as targets.
- You can register your Lambda functions as targets and configure a listener rule to forward requests to the target group for your Lambda function (CLI, API or Management Console).
- When the load balancer forwards the request to a target group with a Lambda function as a target, it invokes your Lambda function and passes the content of the request to the Lambda function, in JSON format.
- Limits:
  - The Lambda function and target group must be in the same account and in the same Region.
  - The maximum size of the request body that you can send to a Lambda function is 1 MB.
  - The maximum size of the response JSON that the Lambda function can send is 1 MB.
  - WebSockets are not supported. Upgrade requests are rejected with an HTTP 400 code.

## AWS Lambda – Limits

- Memory allocation 128MB - 3008MB in 64MB increments.
- Maximum execution time is 15 minutes (900 seconds).
- Size of environment variables maximum 4KB.
- Disk capacity in the "function container" (/tmp) is 512 MB.
- Concurrency limits: 1000 concurrent executions.
- Function burst concurrency 500 -3000 (region dependent).
- Invocation payload:
  - Synchronous 6 MB.
  - Asynchronous 256 KB
- Lambda function deployment size is 50 MB (zipped), 250 MB unzipped.

## AWS Lambda – Layers

- You can configure your Lambda function to pull in additional code and content in the form of layers.
- A layer is a ZIP archive that contains libraries, a custom runtime, or other dependencies.
- With layers, you can use libraries in your function without needing to include them in your deployment package.
- A function can use up to 5 layers at a time.
- Layers are extracted to the /opt directory in the function execution environment.
- Each runtime looks for libraries in a different location under /opt, depending on the language.

## AWS Lambda – Layers

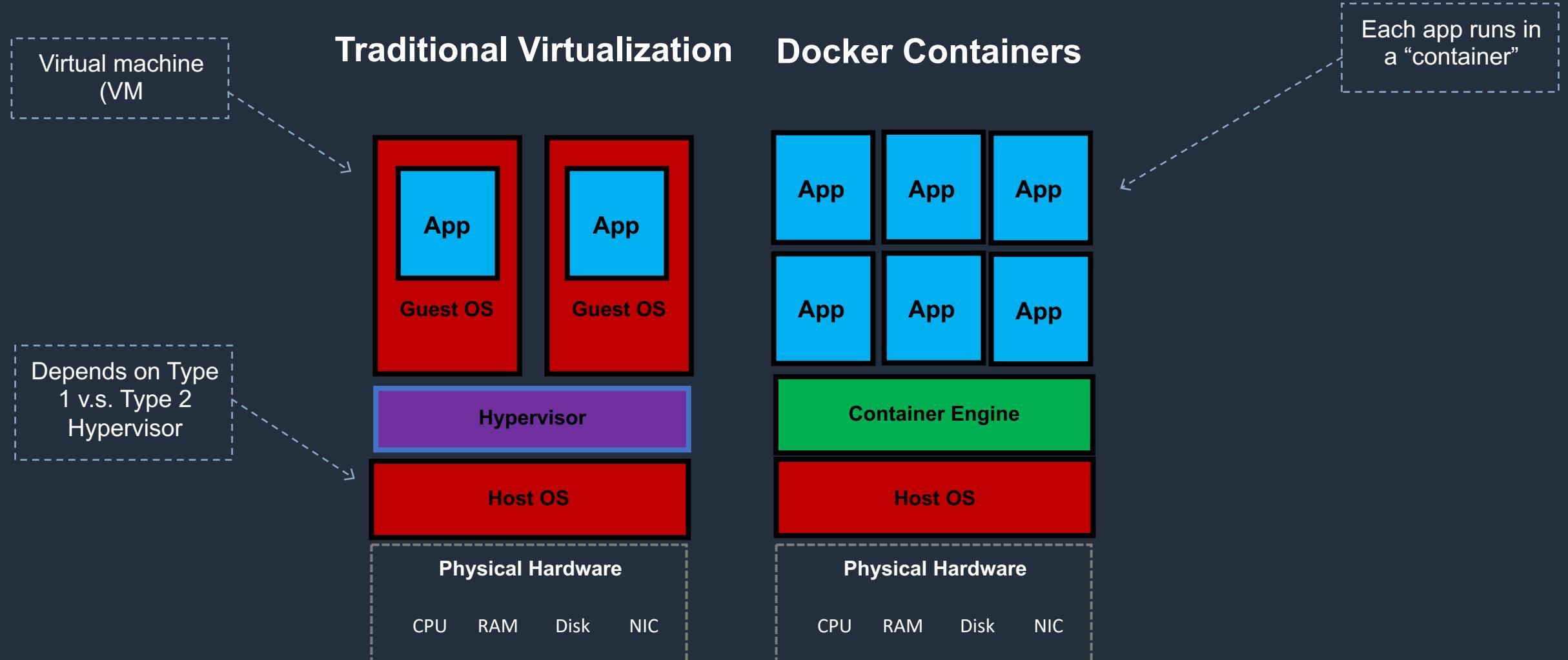
- To add layers to your function, use the update-function-configuration command.
- The following example adds two layers: one from the same account as the function, and one from a different account.

```
$ aws lambda update-function-configuration --function-name my-function \
--layers arn:aws:lambda:us-east-2:123456789012:layer:my-layer:3 \
arn:aws:lambda:us-east-2:210987654321:layer:their-layer:2
{
    "FunctionName": "test-layers",
    "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
    "Runtime": "nodejs12.x",
    "Role": "arn:aws:iam::123456789012:role/service-role/lambda-role",
    "Layers": [
        {
            "Arn": "arn:aws:lambda:us-east-2:123456789012:layer:my-layer:3",
            "CodeSize": 169
        },
        {
            "Arn": "arn:aws:lambda:us-east-2:210987654321:layer:their-layer:2",
            "CodeSize": 169
        }
    ],
    "RevisionId": "81cc64f5-5772-449a-b63e-12330476bcc4",
    ...
}
```

# SECTION 12

Docker Containers on  
AWS: ECS, Fargate and  
ECR

# Comparing Virtualization and Containers



## Definition of a Container (from Docker)

“A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another. A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.”

# Advantages and Use Cases for Docker Containers

## Advantages:

- Efficient use of system resources - less overhead as there's no OS.
- Portability – containers can be easily deployed to different platforms (hardware/software).
- Speed – containers deploy rapidly and can be quickly patched.
- Enhanced development lifecycle - can be scaled and updated quickly and efficiently.

## Use Cases:

- Microservices – this software patterns involves decomposing “monolithic” applications into many separate services that are loosely coupled.
- Lift & Shift or refactor existing applications into cloud architectures.

# Amazon Elastic Container Services (ECS) Overview



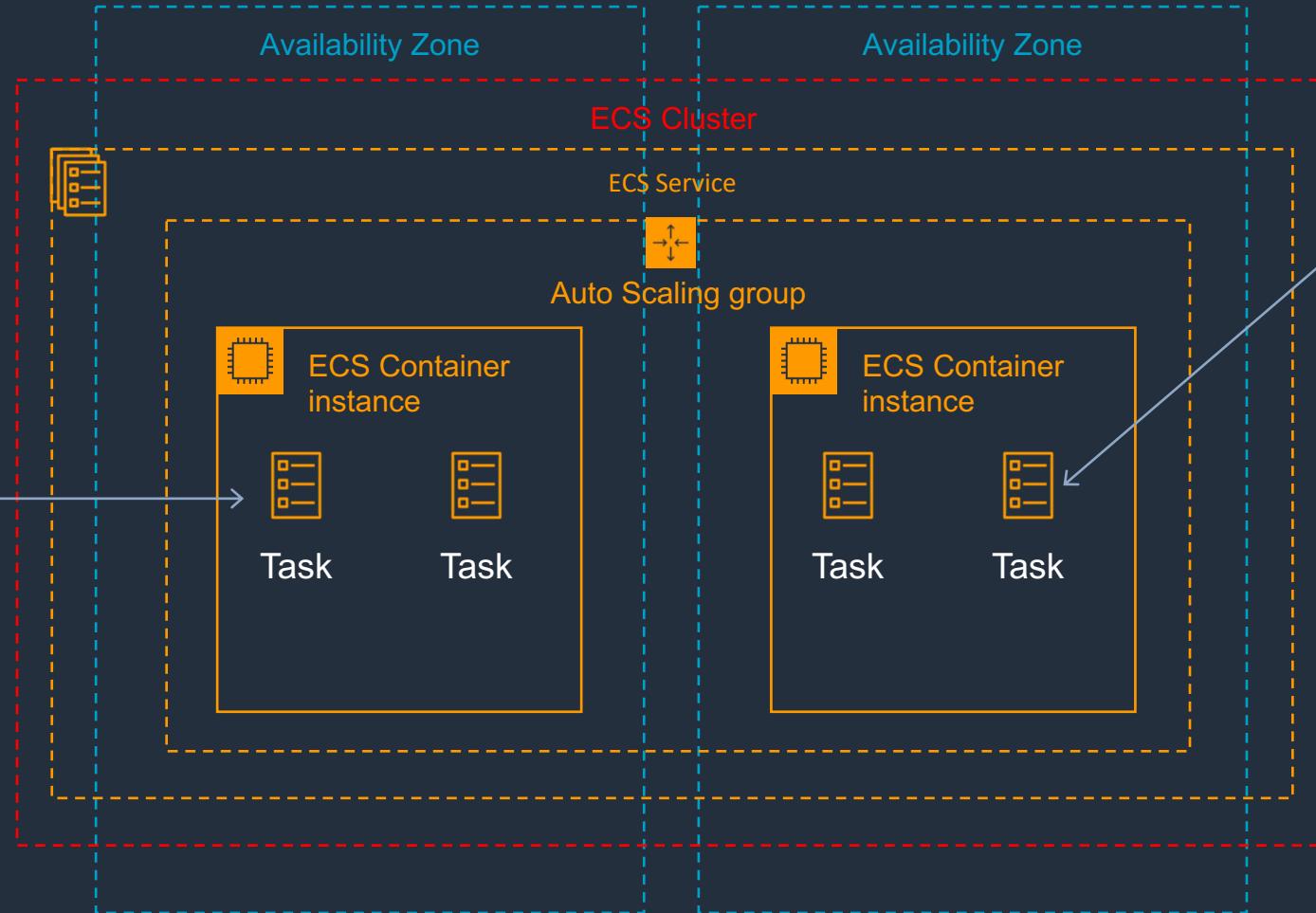
Amazon Elastic Container Registry



Amazon Elastic Container Service

Task Definition

```
{
  "containerDefinitions": [
    {
      "name": "wordpress",
      "links": [
        "mysql"
      ],
      "image": "wordpress",
      "essential": true,
      "portMappings": [
        {
          "containerPort": 80,
          "hostPort": 80
        }
      ],
      "memory": 500,
      "cpu": 10
    }
  ]
}
```



# ECS Terminology

Elastic Container Service (ECS)	Description
Cluster	Logical grouping of EC2 instances
Container instance	EC2 instance running the the ECS agent
Image	Contains the instructions for creating a Docker container. Stored in a registry
Task Definition	Blueprint that describes how a Docker container should launch
Task	A running container using settings in a Task Definition
Service	Defines long running tasks – can control task count with Auto Scaling and attach an ELB
Registry	Storage repository for images - e.g. DockerHub and Amazon Elastic Container Registry (ECR)

## Amazon ECS Overview

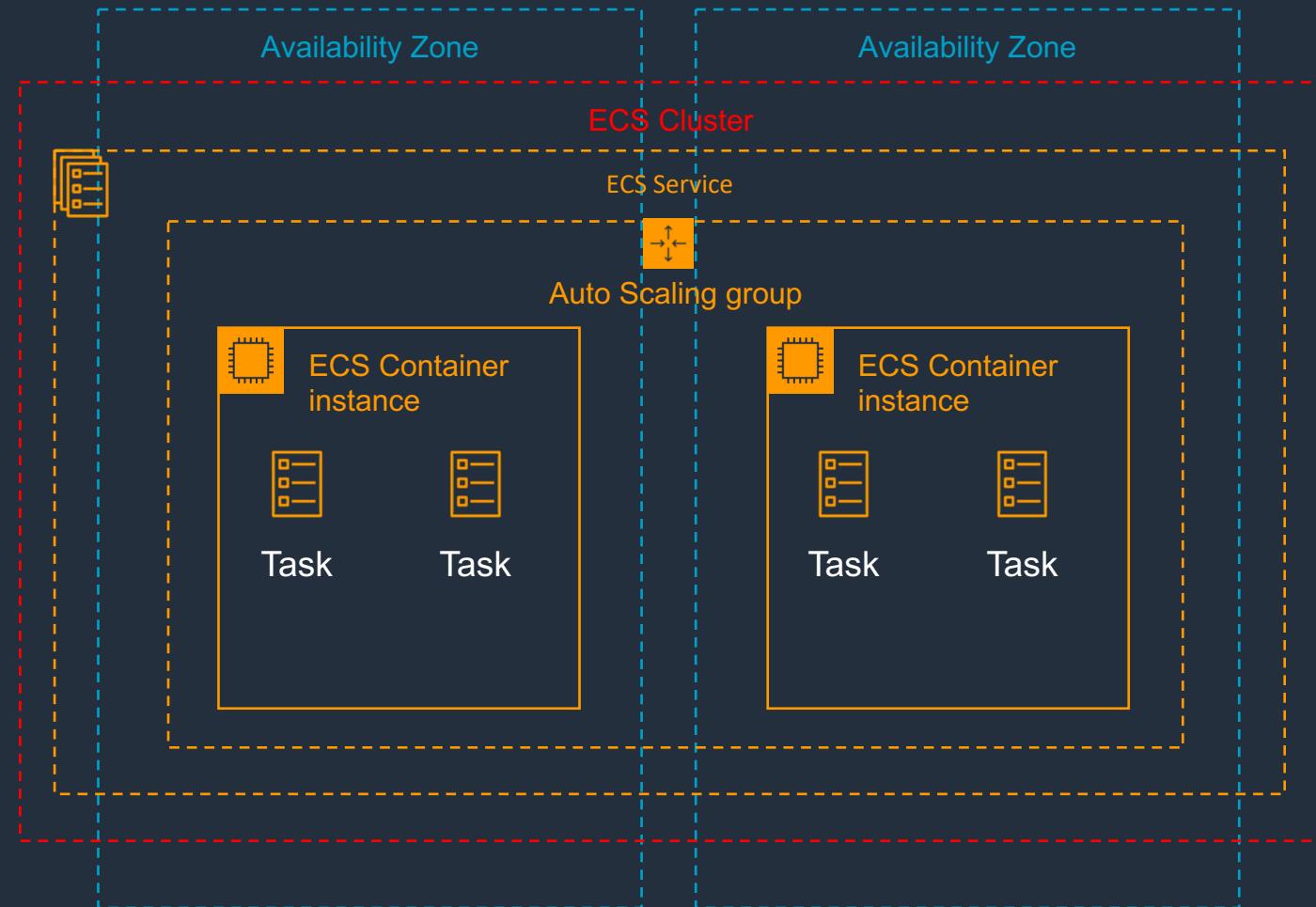
- Amazon Elastic Container Service (ECS) is a highly scalable, high performance container management service that supports Docker containers and allows you to easily run applications on a managed cluster of Amazon EC2 instances.
- There is no additional charge for Amazon ECS. You pay for:
  - Resources created with the EC2 Launch Type (e.g. EC2 instances and EBS volumes).
  - The number and configuration of tasks you run for the Fargate Launch Type.
- You can use Elastic Beanstalk to handle the provisioning of an Amazon ECS cluster, balancing load, auto-scaling, monitoring, and placing your containers across your cluster.
- Alternatively use ECS directly for more fine-grained control for customer application architectures.

## Amazon ECS Overview

- It is possible to associate a service on Amazon ECS to an Application Load Balancer (ALB) for the Elastic Load Balancing (ELB) service.
- The ALB supports a target group that contains a set of instance ports. You can specify a dynamic port in the ECS task definition which gives the container an unused port when it is scheduled on the EC2 instance.

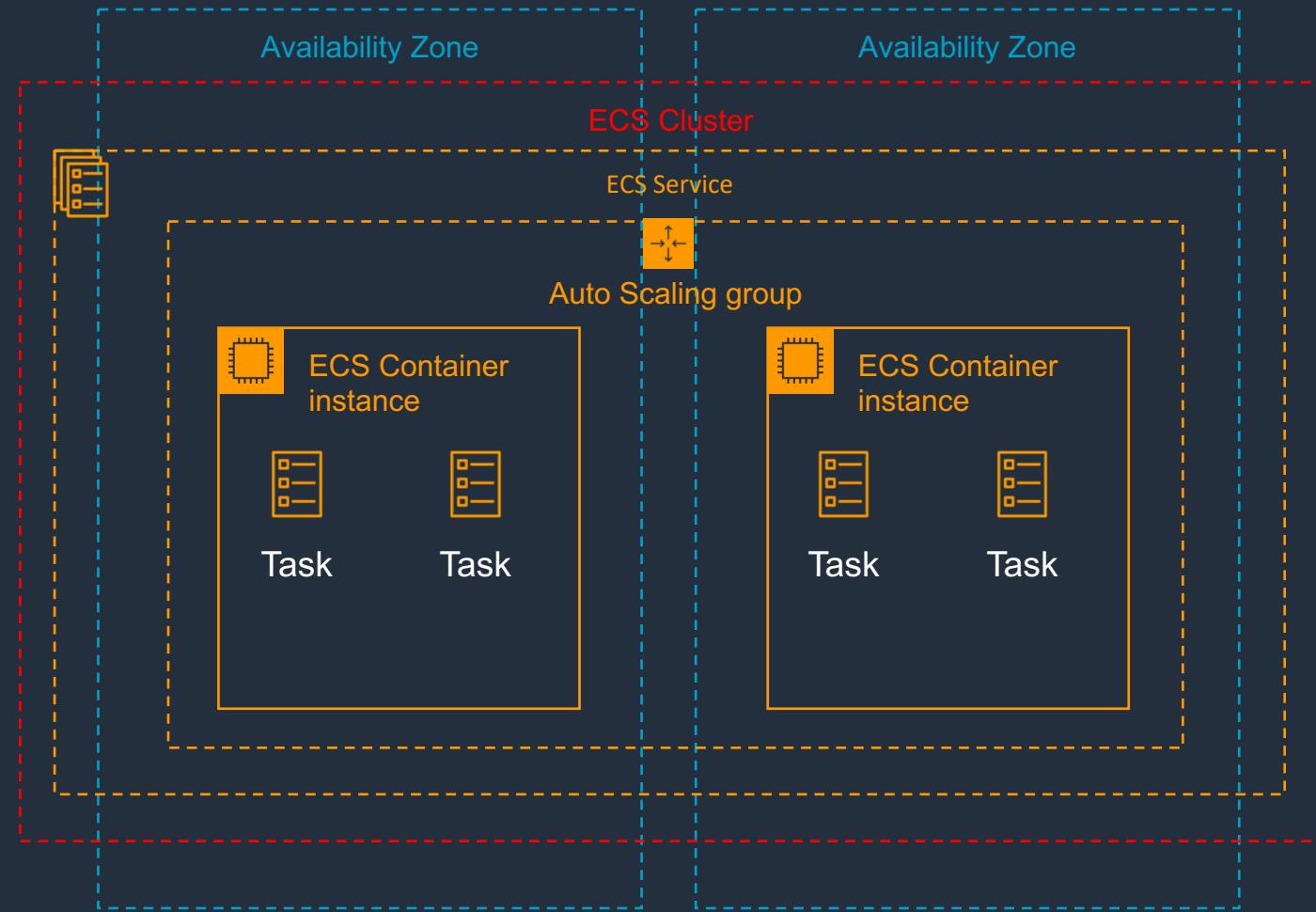
# Amazon ECS Clusters

- ECS Clusters are a logical grouping of container instances that you can place tasks on.
- A default cluster is created but you can then create multiple clusters to separate resources.
- ECS allows the definition of a specified number (desired count) of tasks to run in the cluster.



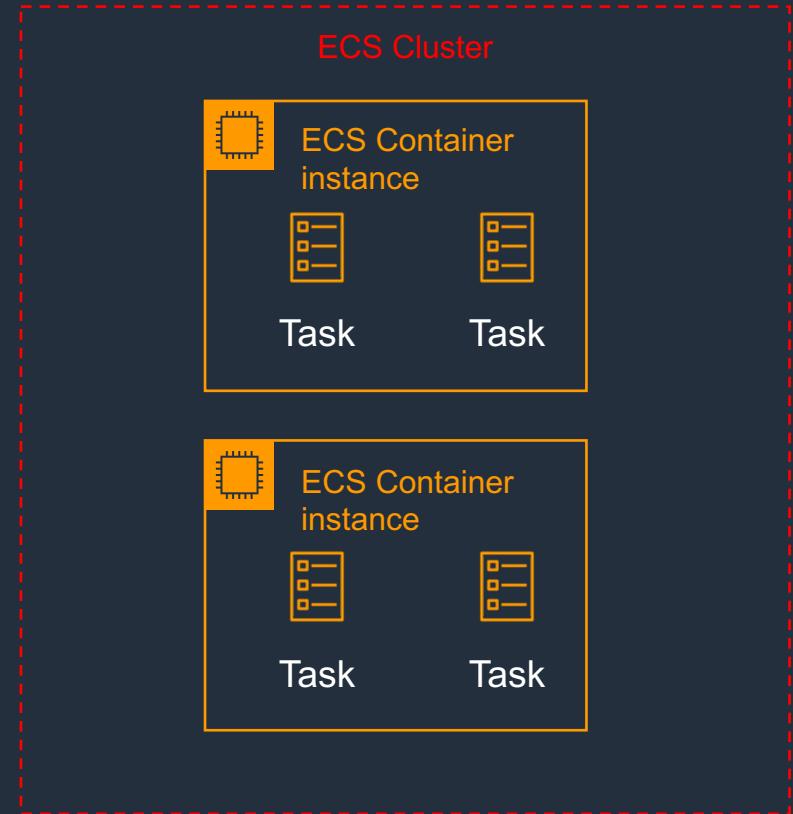
# Amazon ECS Clusters

- Clusters can contain tasks using the Fargate and EC2 launch type.
- EC2 launch type clusters can contain different container instance types.
- Each container instance may only be part of one cluster at a time.
- Clusters are region specific.
- You can create IAM policies for your clusters to allow or restrict users' access to specific clusters.



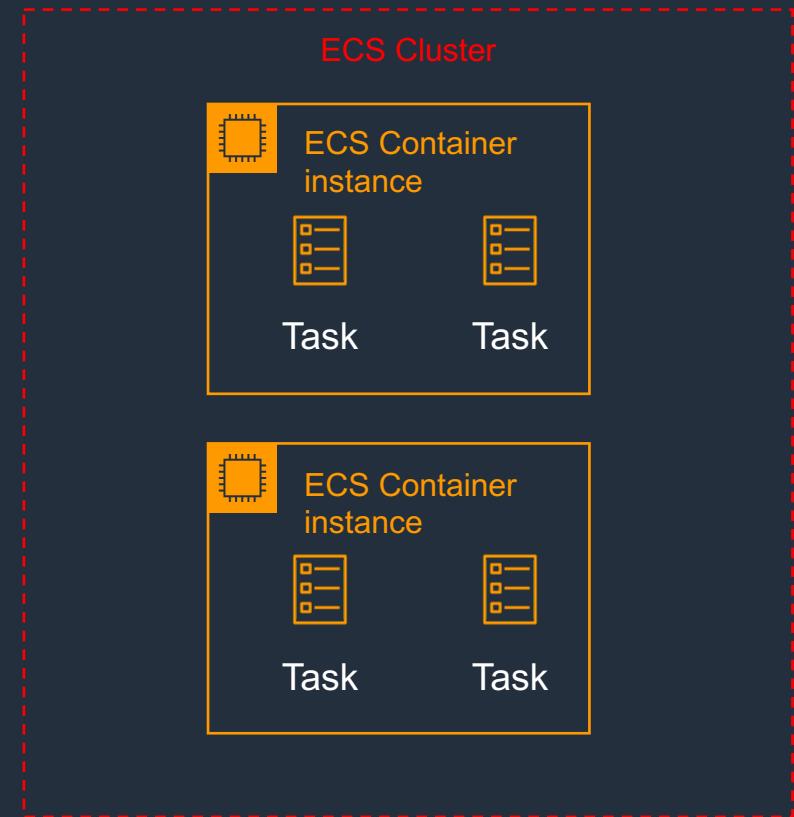
## Amazon ECS Container Instances & Container Agent

- You can use any AMI that meets the Amazon ECS AMI specification.
- The EC2 instances used as container hosts must run an ECS agent.
- The ECS container agent allows container instances to connect to the cluster.
- The container agent runs on each infrastructure resource on an ECS cluster.



## Amazon ECS Container Instances & Container Agent

- The ECS container agent is included in the Amazon ECS optimized AMI and can also be installed on any EC2 instance that supports the ECS specification (only supported on EC2 instances).
- Linux and Windows based.
- For non-AWS Linux instances to be used on AWS you must manually install the ECS container agent.
- The agent is configured in `/etc/ecs/ecs.config`.



## Amazon ECS Images

- Containers are created from a read-only template called an image which has the instructions for creating a Docker container.
- Images are built from a Dockerfile.
- Only Docker containers are currently supported on ECS.
- Images are stored in a registry such as DockerHub or AWS Elastic Container Registry (ECR).
- ECR is a managed AWS Docker registry service that is secure, scalable and reliable.
- ECR supports private Docker repositories with resource-based permissions using AWS IAM in order to access repositories and images.
- Can use the Docker CLI to push, pull and manage images.

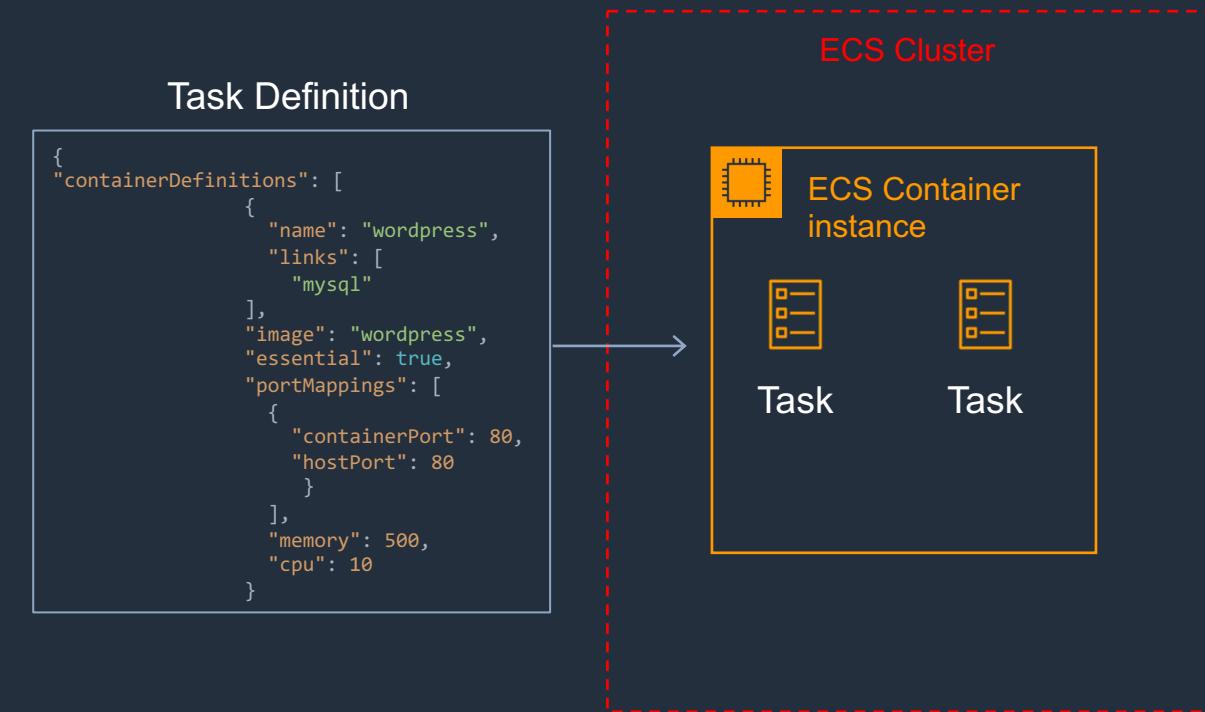


Amazon Elastic Container Registry



# Amazon ECS Tasks and Task Definitions

- A task definition is required to run Docker containers in Amazon ECS.
- A task definition is a text file in JSON format that describes one or more containers, up to a maximum of 10.
- Task definitions use Docker images to launch containers.
- You specify the number of tasks to run (i.e. the number of containers).



## Amazon ECS Tasks and Task Definitions

Some of the parameters you can specify in a task definition include:

- Which Docker images to use with the containers in your task.
- How much CPU and memory to use with each container.
- Whether containers are linked together in a task.
- The Docker networking mode to use for the containers in your task.
- What (if any) ports from the container are mapped to the host container instances.
- Whether the task should continue if the container finished or fails.
- The commands the container should run when it is started.
- Environment variables that should be passed to the container when it starts.
- Data volumes that should be used with the containers in the task.
- IAM role the task should use for permissions.

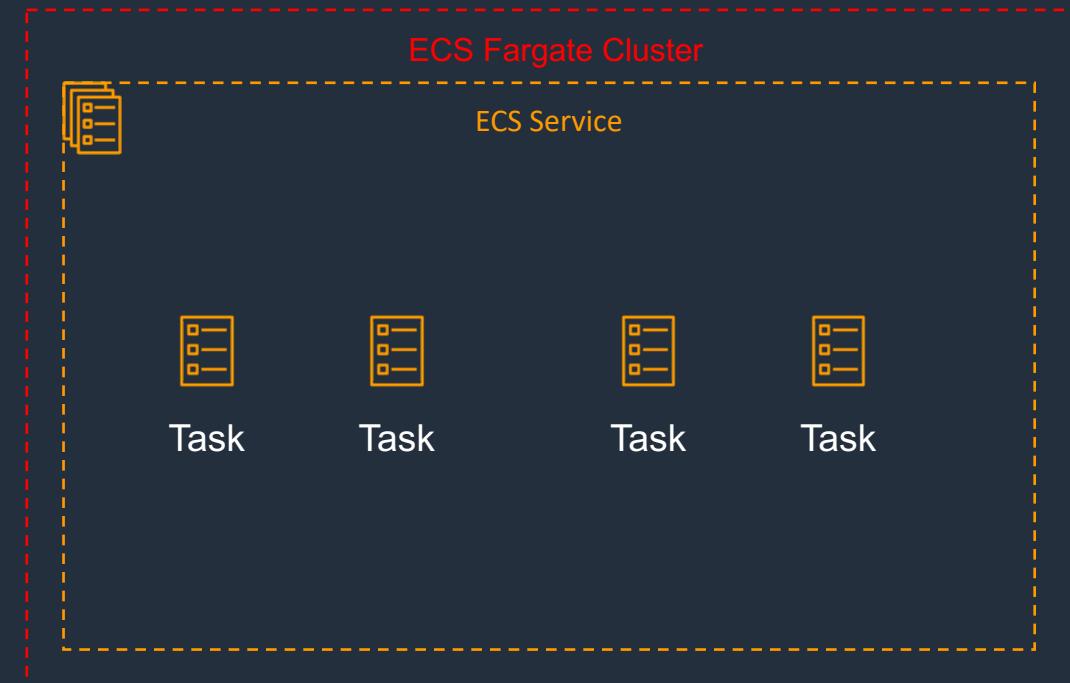
# Launch Types – EC2 and Fargate



Registry:  
ECR, Docker Hub, Self-hosted



Registry:  
ECR, Docker Hub



## EC2 Launch Type

- You explicitly provision EC2 instances
- You're responsible for managing EC2 instances
- Charged per running EC2 instance
- EFS and EBS integration
- You handle cluster optimization
- More granular control over infrastructure

## Fargate Launch Type

- Fargate automatically provisions resources
- Fargate provisions and manages compute
- Charged for running tasks
- No EFS and EBS integration
- Fargate handles cluster optimization
- Limited control, infrastructure is automated

# Amazon ECS Launch Types

## Fargate Launch Type

- The Fargate launch type allows you to run your containerized applications without the need to provision and manage the backend infrastructure. Just register your task definition and Fargate launches the container for you.
- Fargate Launch Type is a serverless infrastructure managed by AWS.
- Fargate only supports container images hosted on Elastic Container Registry (ECR) or Docker Hub.

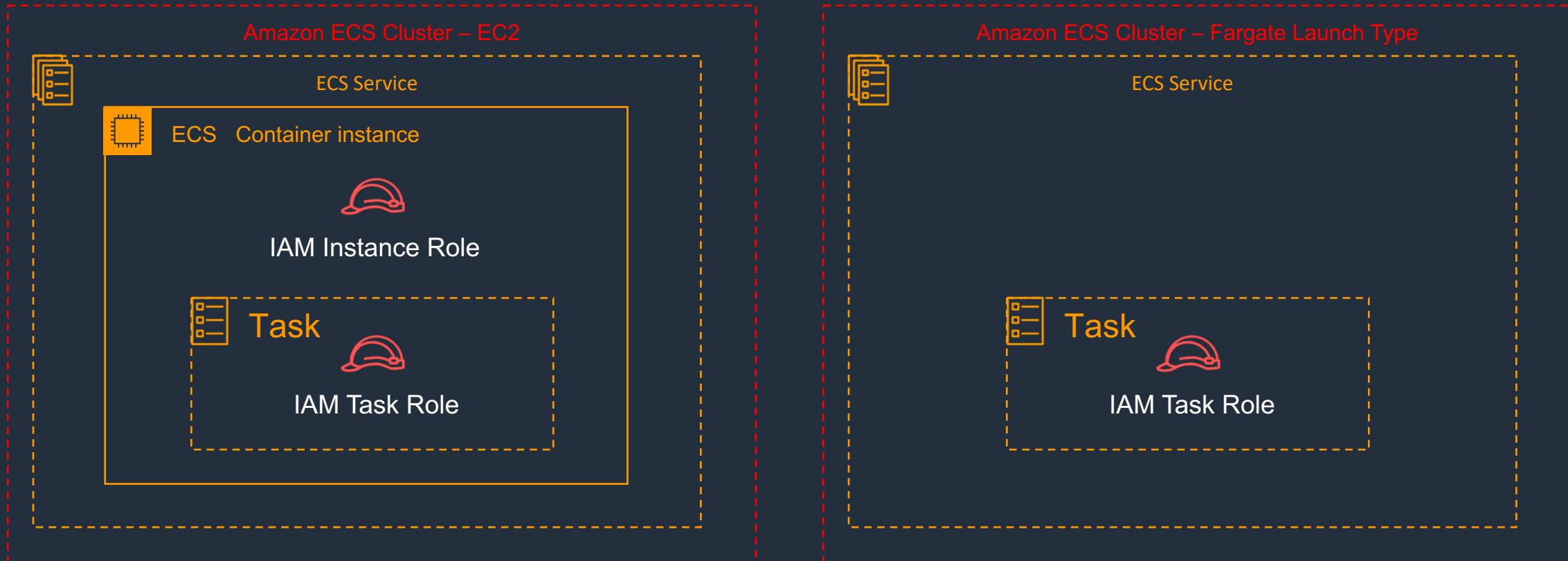
## EC2 Launch Type

- The EC2 launch type allows you to run your containerized applications on a cluster of Amazon EC2 instances that you manage.
- Private repositories are only supported by the EC2 Launch Type.

# Amazon ECS Launch Types

Amazon EC2	Amazon Fargate
You explicitly provision EC2 instances	The control plane asks for resources and Fargate automatically provisions
You're responsible for upgrading, patching, care of EC2 pool	Fargate provisions compute as needed
You must handle cluster optimization	Fargate handles cluster optimization
More granular control over infrastructure	Limited control, as infrastructure is automated

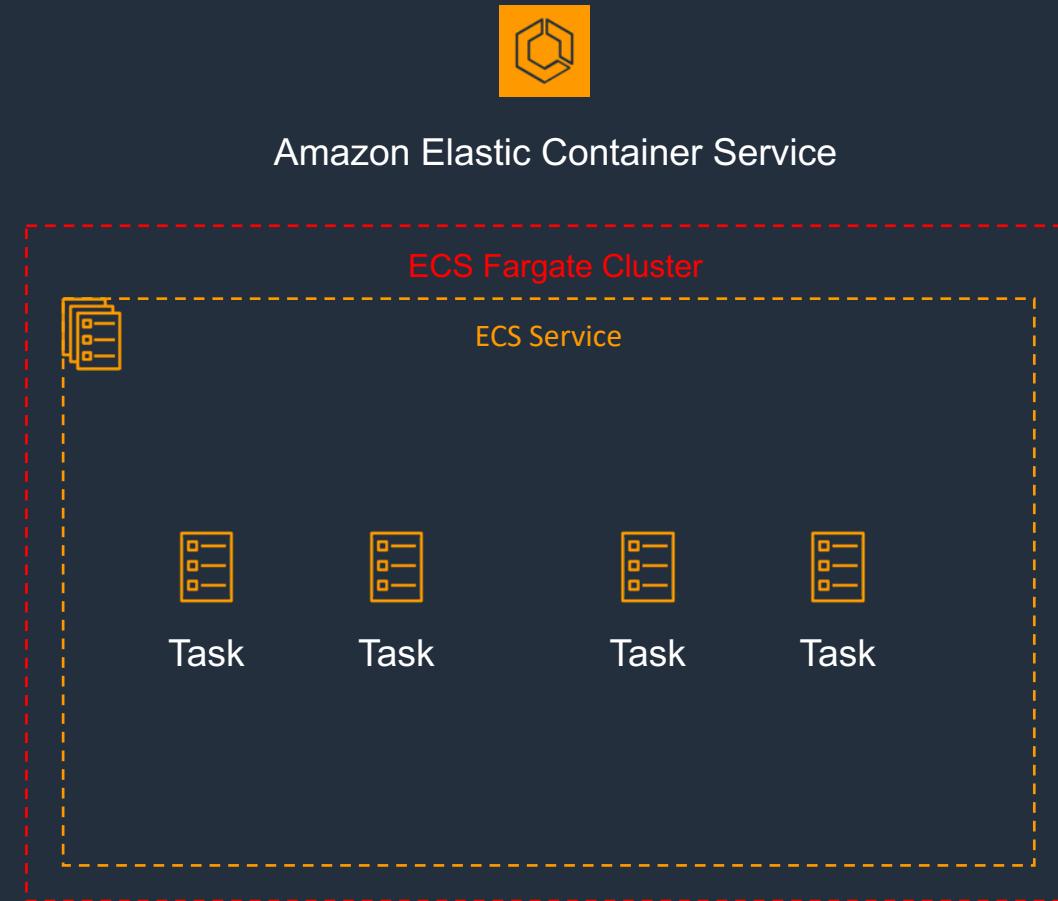
# Amazon ECS - IAM Roles



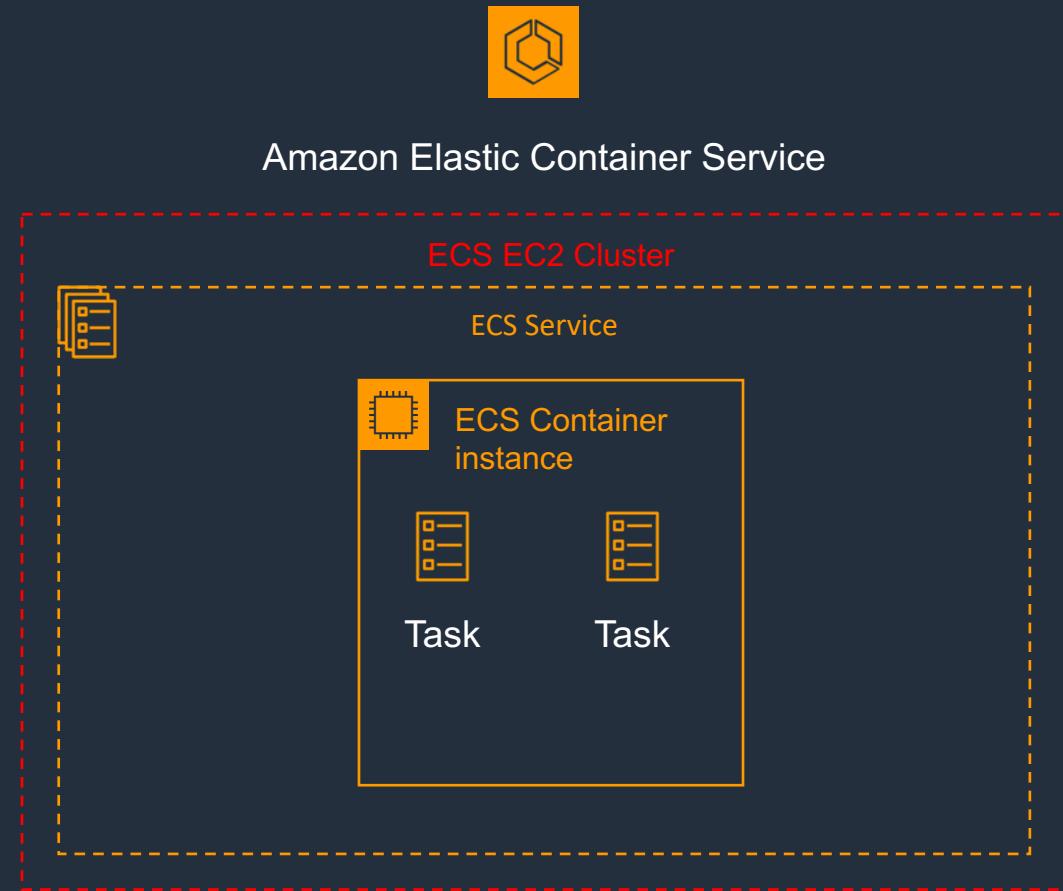
## IAM Roles for Tasks

- EC2 instances use an IAM role to access ECS.
- ECS tasks can have IAM Roles attached (including Fargate tasks).
- ECS tasks use the IAM role to access services and resources.
- The container agent makes calls to the ECS API on your behalf through the applied IAM roles and policies.
- You need to apply IAM roles to container instances before they are launched (EC2 launch type).
- Best practices:
  - AWS recommend limiting the permissions that are assigned to the container instance's IAM roles.
  - Assign extra permissions to tasks through separate IAM roles (IAM Roles for Tasks).

# Create AWS Fargate Cluster using the Console



# Create Amazon ECS Cluster and Task using the Console



# Create Amazon ECS Cluster and Task using the ECS CLI

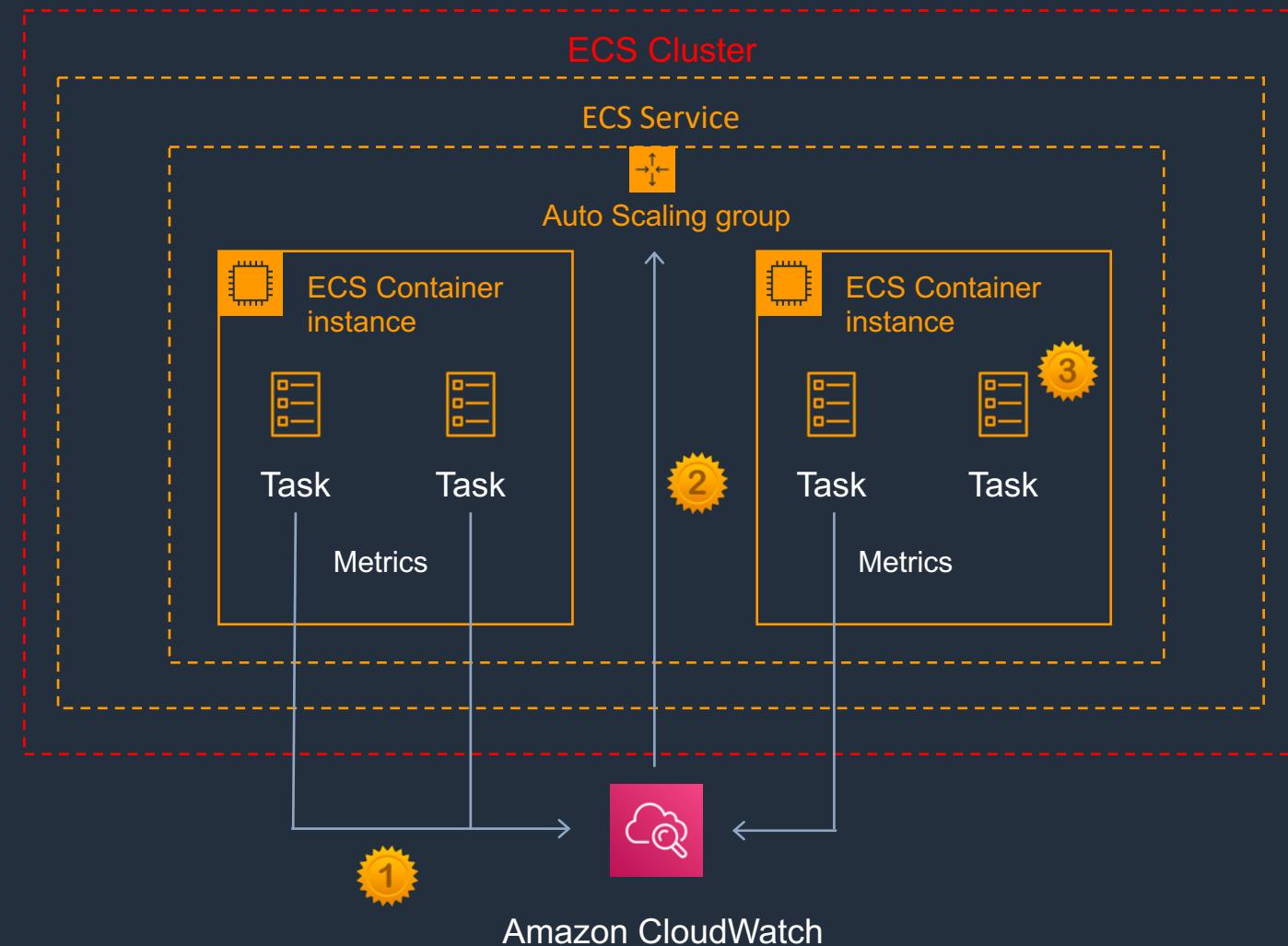


# Scaling the Amazon ECS Cluster



# Amazon ECS – Service Auto Scaling

- 1. Metric reports CPU > 80%
- 2. CloudWatch notifies ASG
- 3. AWS launches additional task

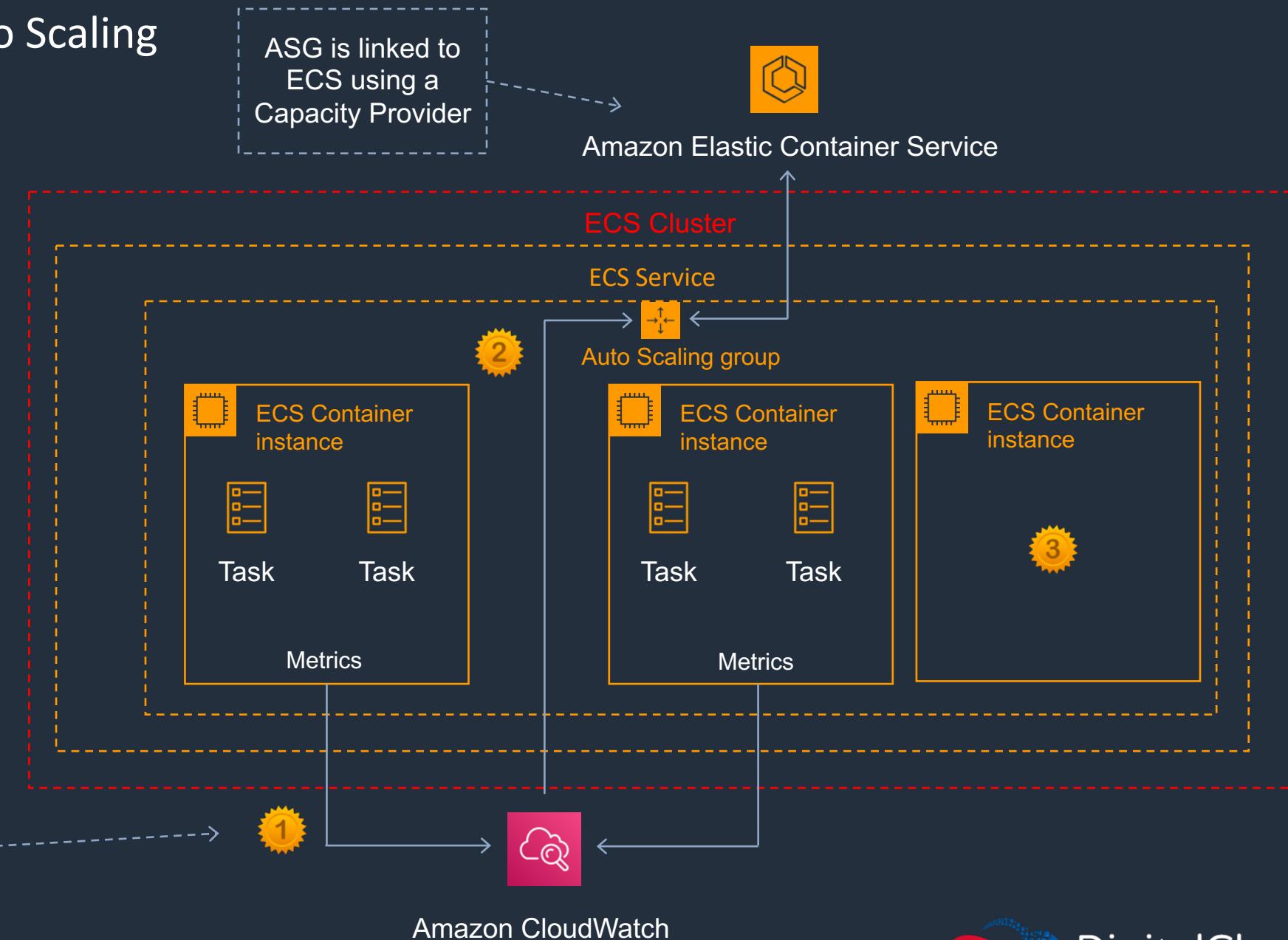


## Amazon ECS – Service Auto Scaling

- Amazon ECS service can optionally be configured to use Service Auto Scaling to adjust the desired task count up or down automatically.
- Service Auto Scaling leverages the Application Auto Scaling service to provide this functionality.
- Amazon ECS Service Auto Scaling supports the following types of scaling policies:
  - Target Tracking Scaling Policies—Increase or decrease the number of tasks that your service runs based on a target value for a specific CloudWatch metric.
  - Step Scaling Policies—Increase or decrease the number of tasks that your service runs in response to CloudWatch alarms. Step scaling is based on a set of scaling adjustments, known as step adjustments, which vary based on the size of the alarm breach.
  - Scheduled Scaling—Increase or decrease the number of tasks that your service runs based on the date and time.

# Amazon ECS – Cluster Auto Scaling

- 1. Metric reports target capacity > 80%
- 2. CloudWatch notifies ASG
- 3. AWS launches additional container instance



A “Capacity provider reservation” metric measures the total percentage of cluster resources needed by all ECS workloads in the cluster

Amazon CloudWatch

## Amazon ECS – Cluster Auto Scaling

- Uses a new ECS resource type called a Capacity Provider.
- A Capacity Provider can be associated with an EC2 Auto Scaling Group (ASG).
- When you associate an ECS Capacity Provider with an ASG and add the Capacity Provider to an ECS cluster, the cluster can now scale your ASG automatically by using two new features of ECS:
  - Managed scaling, with an automatically-created scaling policy on your ASG, and a new scaling metric (Capacity Provider Reservation) that the scaling policy uses; and
  - Managed instance termination protection, which enables container-aware termination of instances in the ASG when scale-in happens.

# Amazon ECS – Cleaning up

**May need to manually clean up some resources first:**

Delete ECS service

Stop ECS tasks

Deactivate capacity provider

Delete the manually created ASG

**ecs-cli commands:**

```
ecs-cli compose service rm --cluster-config ec2-lesson --ecs-profile ec2-lesson-profile
```

```
ecs-cli down --force --cluster-config ec2-lesson --ecs-profile ec2-lesson-profile
```

**Lastly, check the following are deleted (or delete):**

ASG

EC2 instances

CloudFormation Stack

## Amazon ECS – Task Placement Strategy

- A task placement strategy is an algorithm for selecting instances for task placement or tasks for termination. Task placement strategies can be specified when either running a task or creating a new service.
- This is relevant only to the EC2 launch type.
- Amazon ECS supports the following task placement strategies:
  - binpack** - place tasks based on the least available amount of CPU or memory. This minimizes the number of instances in use.
  - random** - place tasks randomly.

## Amazon ECS – Task Placement Strategy

**spread** - place tasks evenly based on the specified value.

- Accepted values are instanceId (or host, which has the same effect), or any platform or custom attribute that is applied to a container instance, such as attribute:ecs.availability-zone.
- Service tasks are spread based on the tasks from that service.
- Standalone tasks are spread based on the tasks from the same task group.

## Amazon ECS – Task Placement Strategy

- The following strategy distributes tasks evenly across Availability Zones:

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

## Amazon ECS – Task Placement Strategy

- The following strategy distributes tasks evenly across all instances:

```
"placementStrategy": [  
    {  
        "field": "instanceId",  
        "type": "spread"  
    }  
]
```

## Amazon ECS – Task Placement Strategy

- The following strategy bin packs tasks based on memory:

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

## Amazon ECS – Task Placement Strategy

- The following strategy distributes tasks evenly across Availability Zones and then bin packs tasks based on memory within each Availability Zone:

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

## Amazon ECS – Task Placement Constraints

- A task placement constraint is a rule that is considered during task placement.
- Amazon ECS supports the following types of task placement constraints:
  - **distinctInstance** - Place each task on a different container instance.
  - **memberOf** - Place tasks on container instances that satisfy an expression.

## Amazon ECS – Cluster Query Language

- Cluster queries are expressions that enable you to group objects.
- For example, you can group container instances by attributes such as Availability Zone, instance type, or custom metadata.
- Expressions have the following syntax: **subject operator [argument]**
- Example 1: The following expression selects instances with the specified instance type.

```
attribute:ecs.instance-type == t2.small
```

- Example 2: The following expression selects instances in the us-east-1a or us-east-1b Availability Zone.

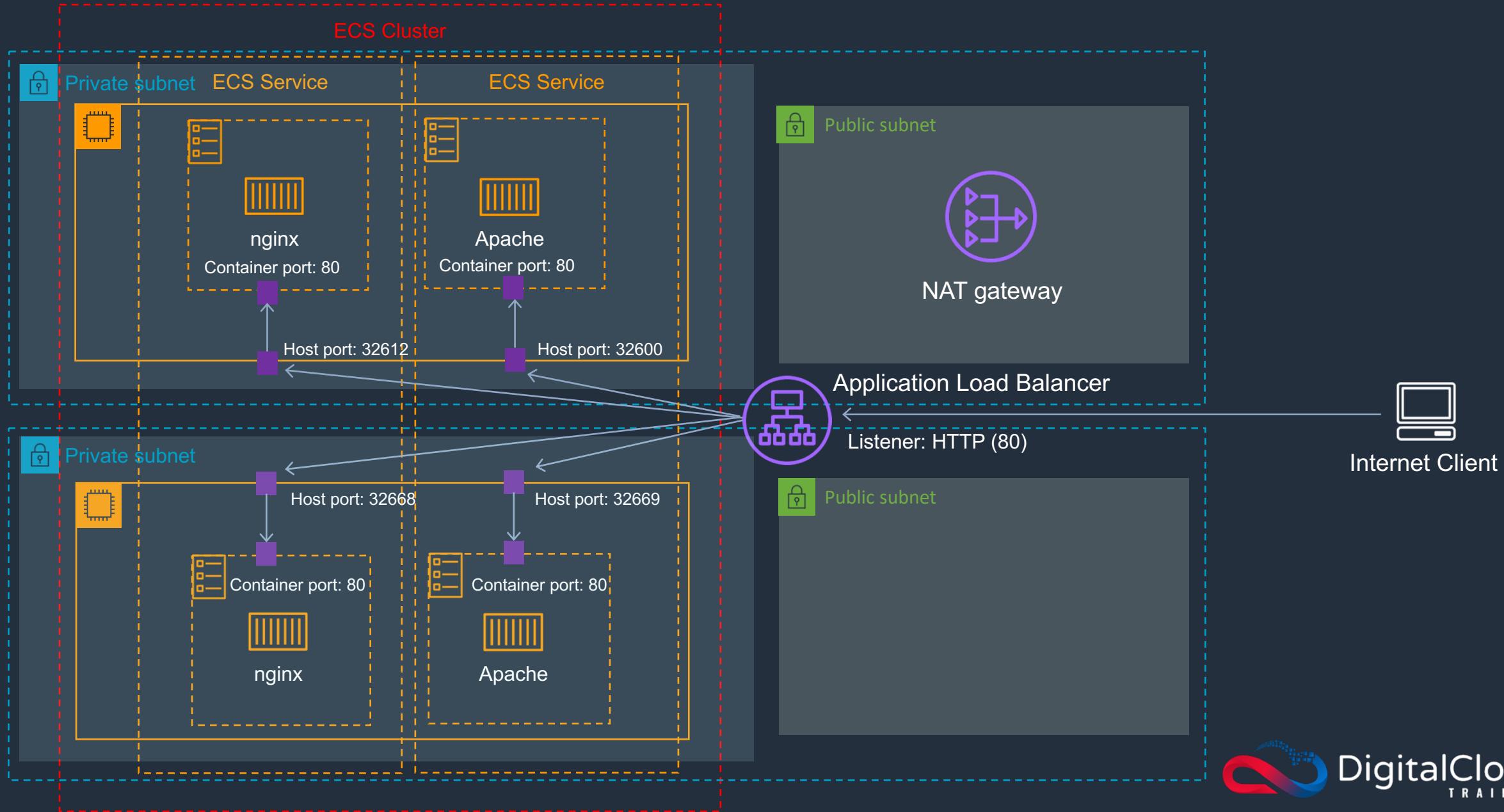
```
attribute:ecs.availability-zone in [us-east-1a, us-east-1b]
```

## Amazon ECS – Cluster Query Language

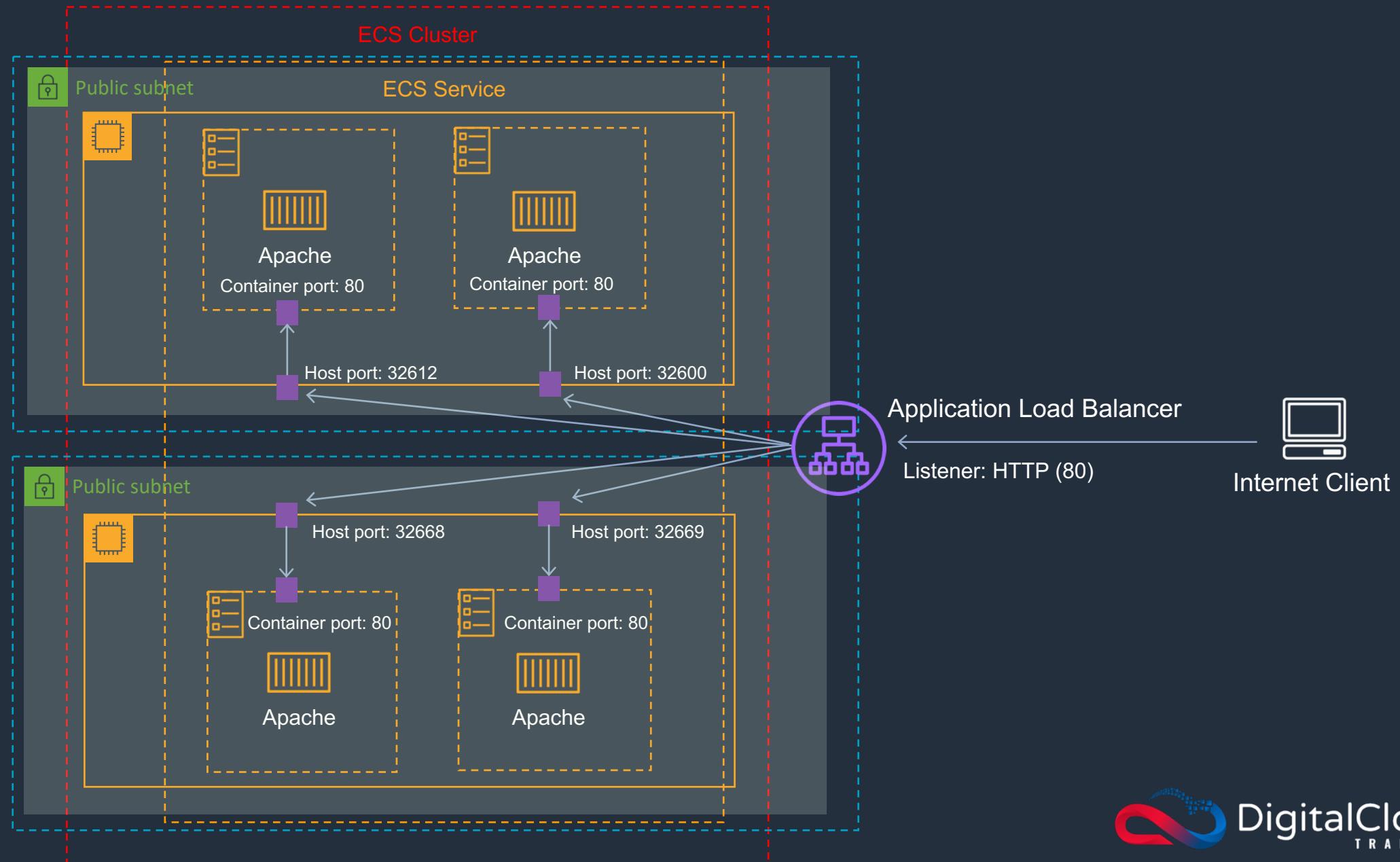
- Example 3: The following expression selects instances that are hosting tasks in the **service:production** group.

```
task:group == service:production
```

# ECS with Application Load Balancer (ALB)



# ECS with Application Load Balancer (ALB)



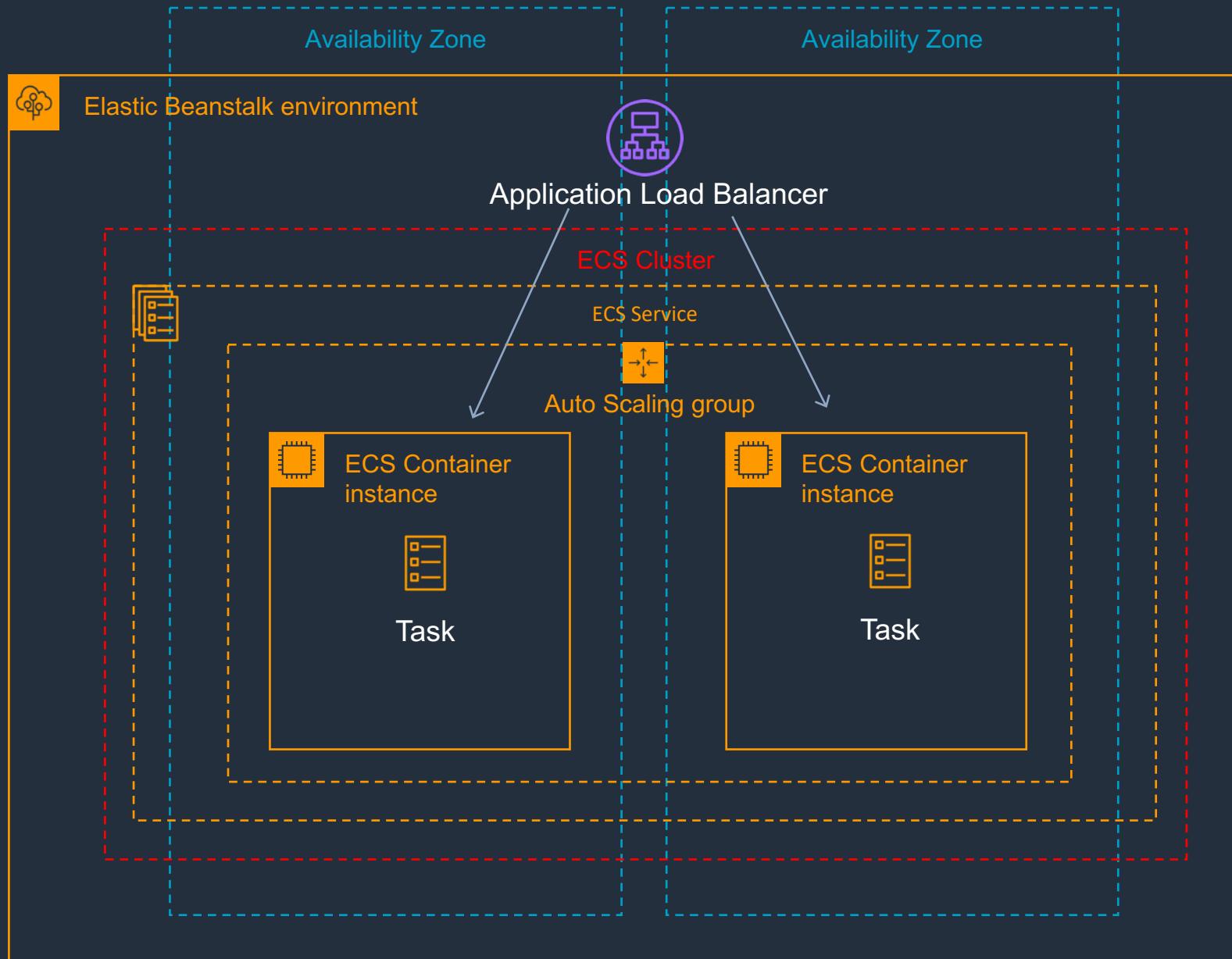
# Amazon ECS with Elastic Beanstalk

## Single Container Docker

- The single container platform can be used to deploy a Docker image (described in a Dockerfile or Dockerrun.aws.json definition) and source code to EC2 instances running in an Elastic Beanstalk environment.
- Use the single container platform when you only need to run one container per instance.



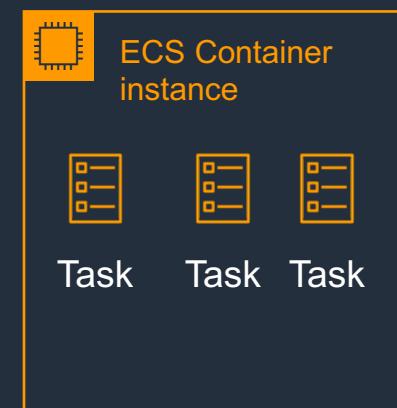
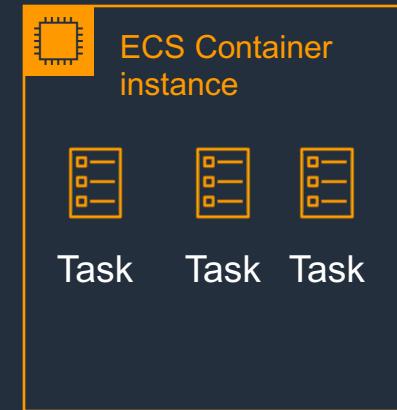
# Amazon ECS with Elastic Beanstalk – Single Container Docker



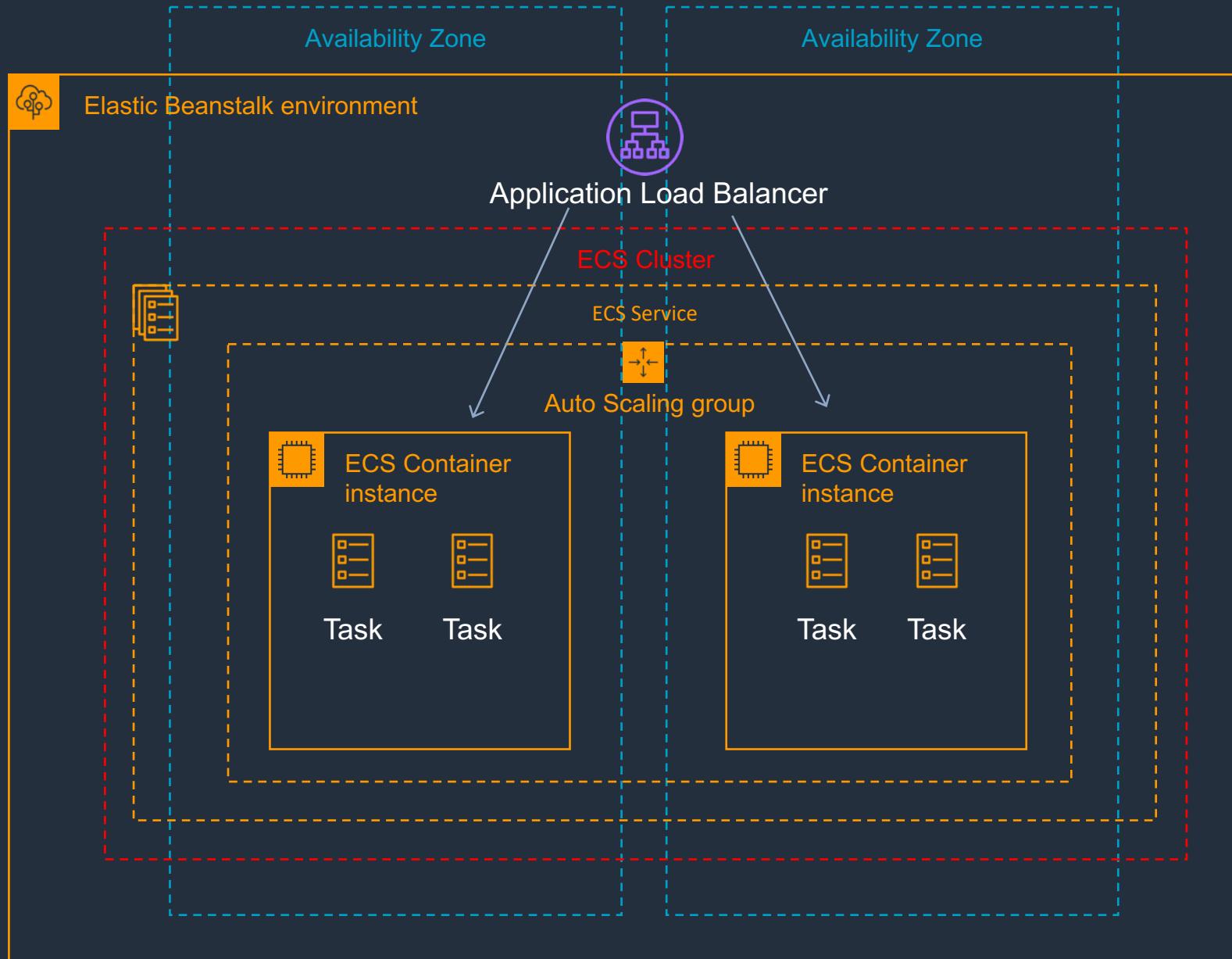
# Amazon ECS with Elastic Beanstalk

## Multicontainer Docker

- The other basic platform, Multicontainer Docker, uses the Amazon Elastic Container Service to coordinate a deployment of multiple Docker containers to an Amazon ECS cluster in an Elastic Beanstalk environment.
- The instances in the environment each run the same set of containers, which are defined in a `Dockerrun.aws.json` file.
- Use the multicontainer platform when you need to deploy multiple Docker containers to each instance.



# Amazon ECS with Elastic Beanstalk – Multicontainer Docker

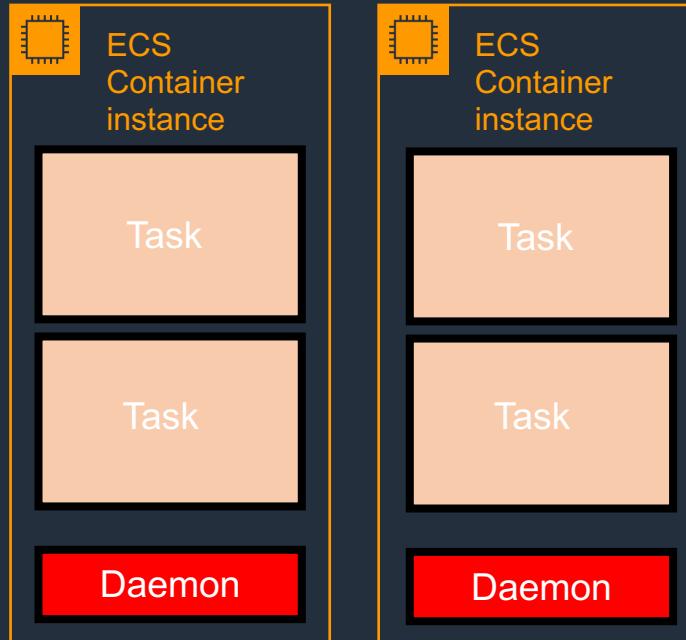


## Amazon ECS – Tracing with AWS X-Ray

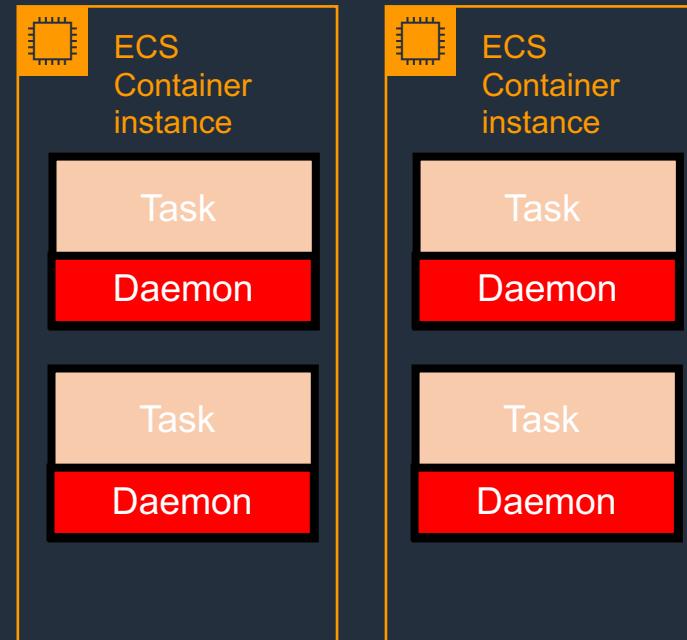
- There are two ways to run X-Ray:
  - X-Ray agent runs in a daemon container on each EC2 container instance
  - X-Ray agent runs locally in a Docker container.
- For Fargate the X-Ray daemon only runs as locally in a Docker container.
- Task definition:
  - X-Ray runs on port 2000 UDP.
  - Must specify the daemon address.
  - Must link the containers together.
- X-Ray provides a Docker container image that you can deploy alongside your application:
  - Command: *docker pull amazon/aws-xray-daemon*

# AWS X-Ray with Amazon ECS

Daemon Container



Local Daemon in Container



Local Daemon in Container



## Amazon ECS – Tracing with AWS X-Ray

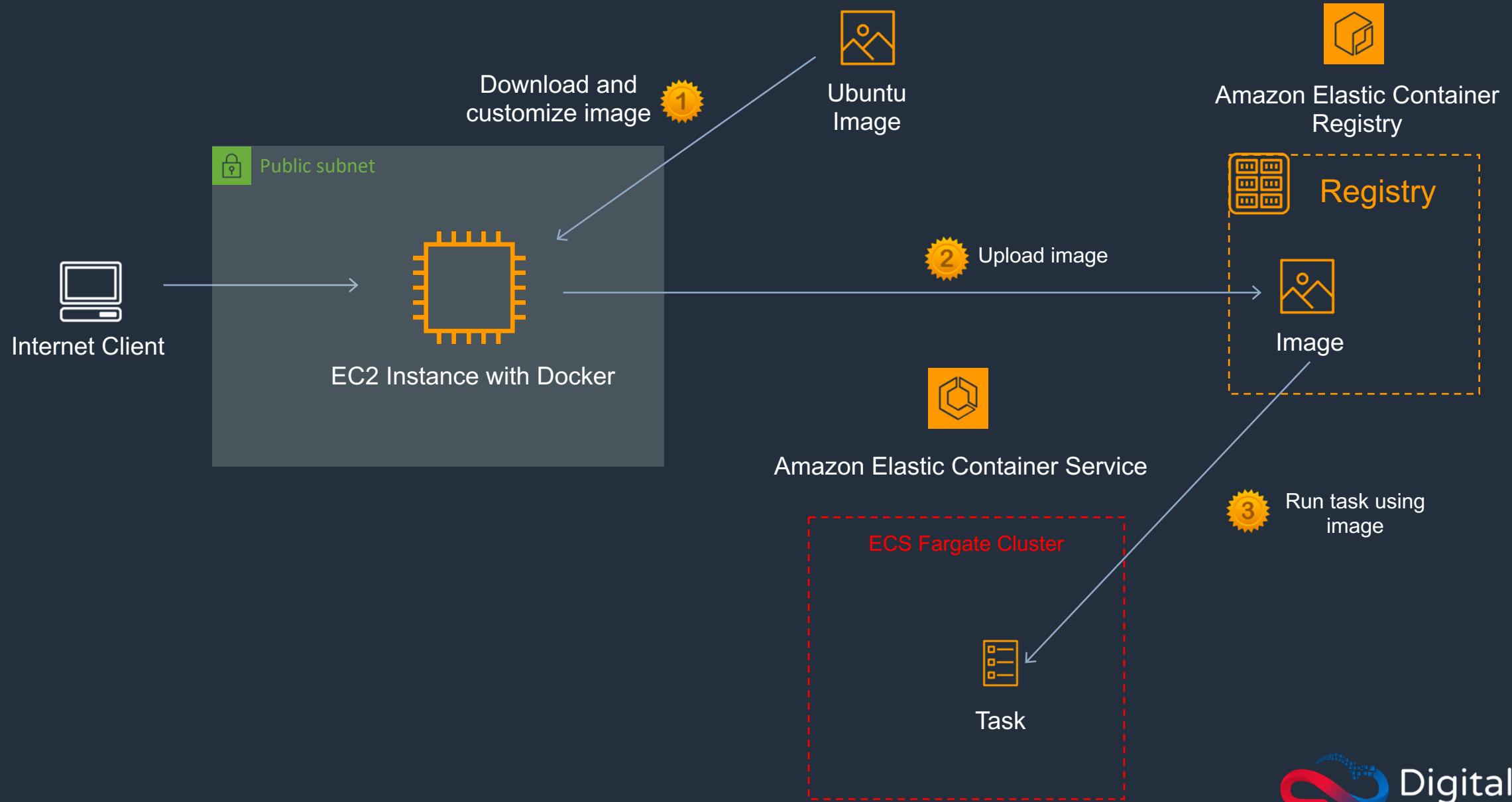
- Example Task Definition:

```
{  
    "name": "xray-daemon",  
    "image": "amazon/aws-xray-daemon",  
    "cpu": 32,  
    "memoryReservation": 256,  
    "portMappings" : [  
        {  
            "hostPort": 0,  
            "containerPort": 2000,  
            "protocol": "udp"  
        }  
    ]  
}
```

## Amazon Elastic Container Registry (ECR)

- Amazon Elastic Container Registry (ECR) is a fully-managed Docker container registry that makes it easy for developers to store, manage, and deploy Docker container images.
- Amazon ECR is integrated with Amazon Elastic Container Service (ECS).
- Amazon ECR hosts your images in a highly available and scalable architecture, allowing you to reliably deploy containers for your applications.
- Integration with AWS Identity and Access Management (IAM) provides resource-level control of each repository.

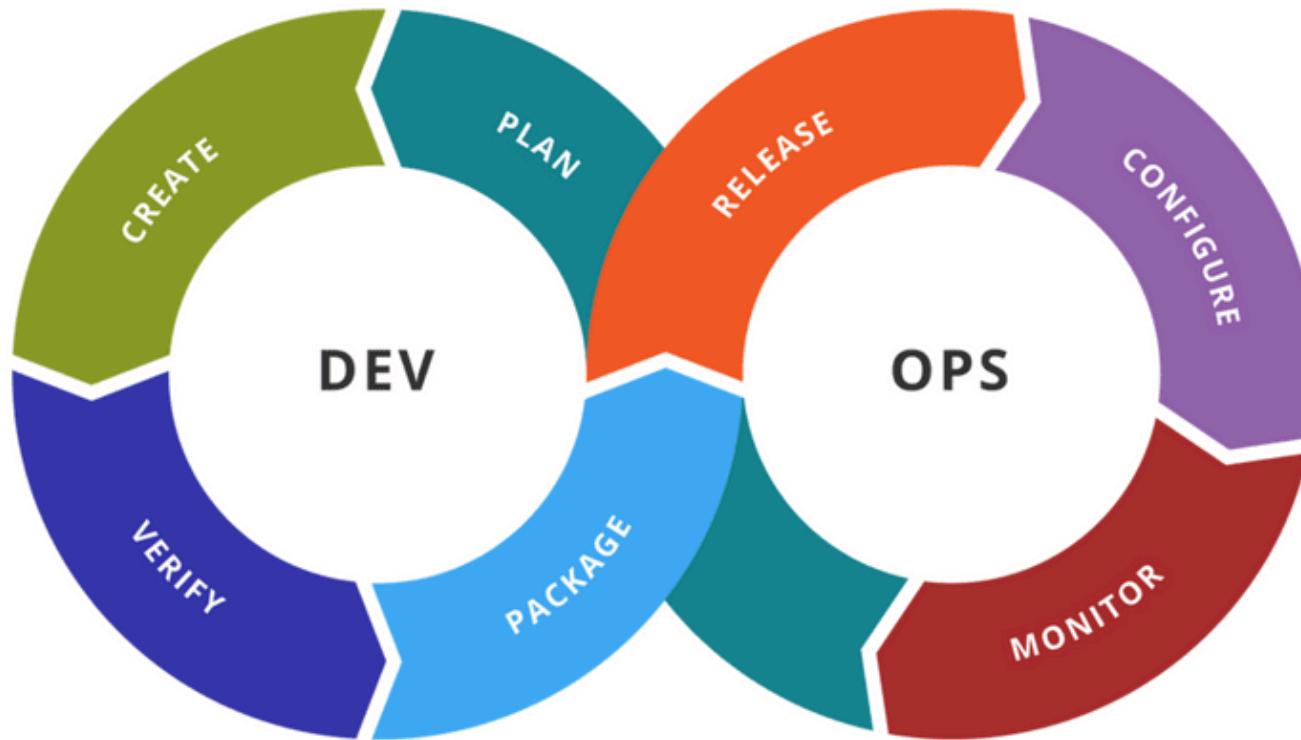
# Amazon Elastic Container Registry (ECR)



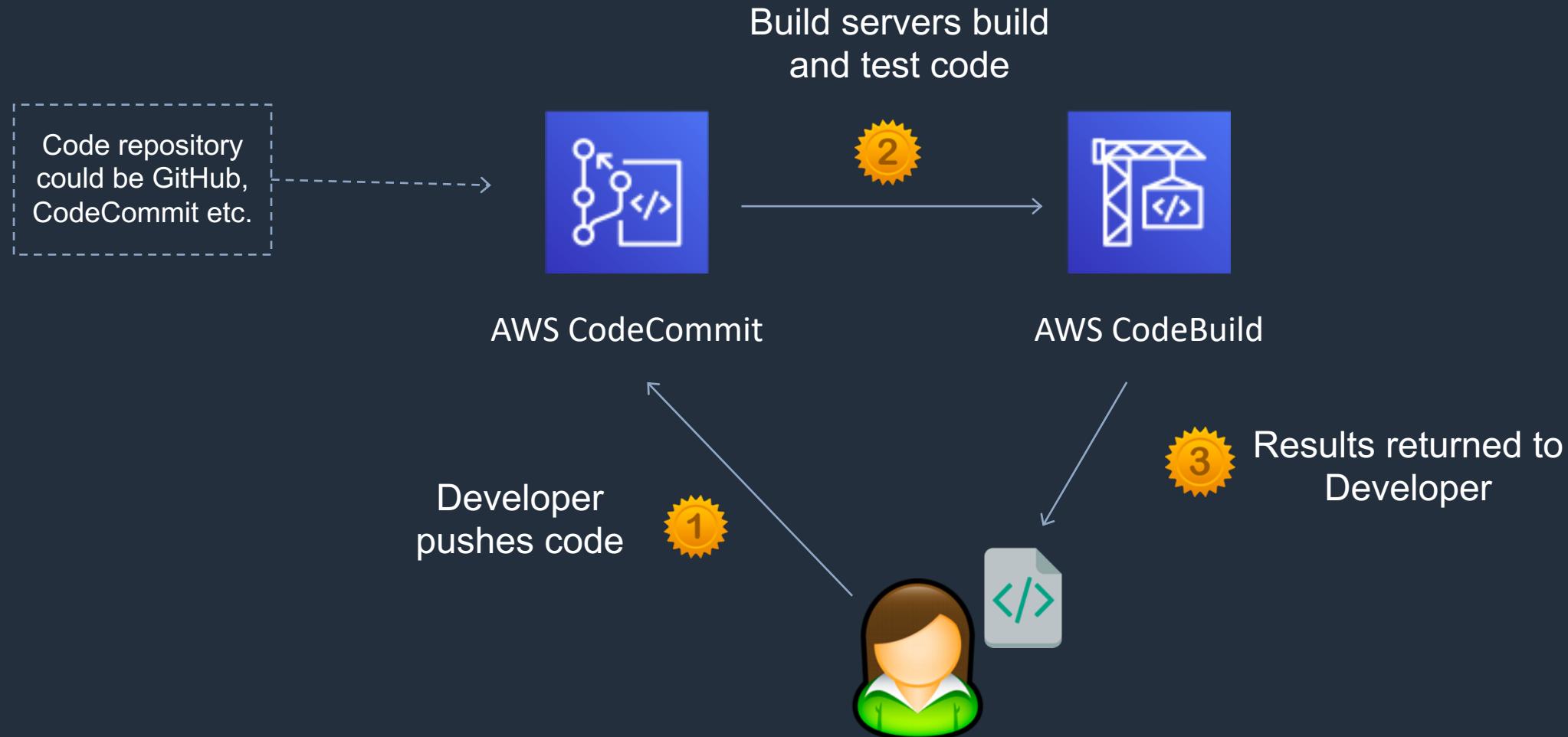
# SECTION 13

## AWS Developer Tools: CI/CD

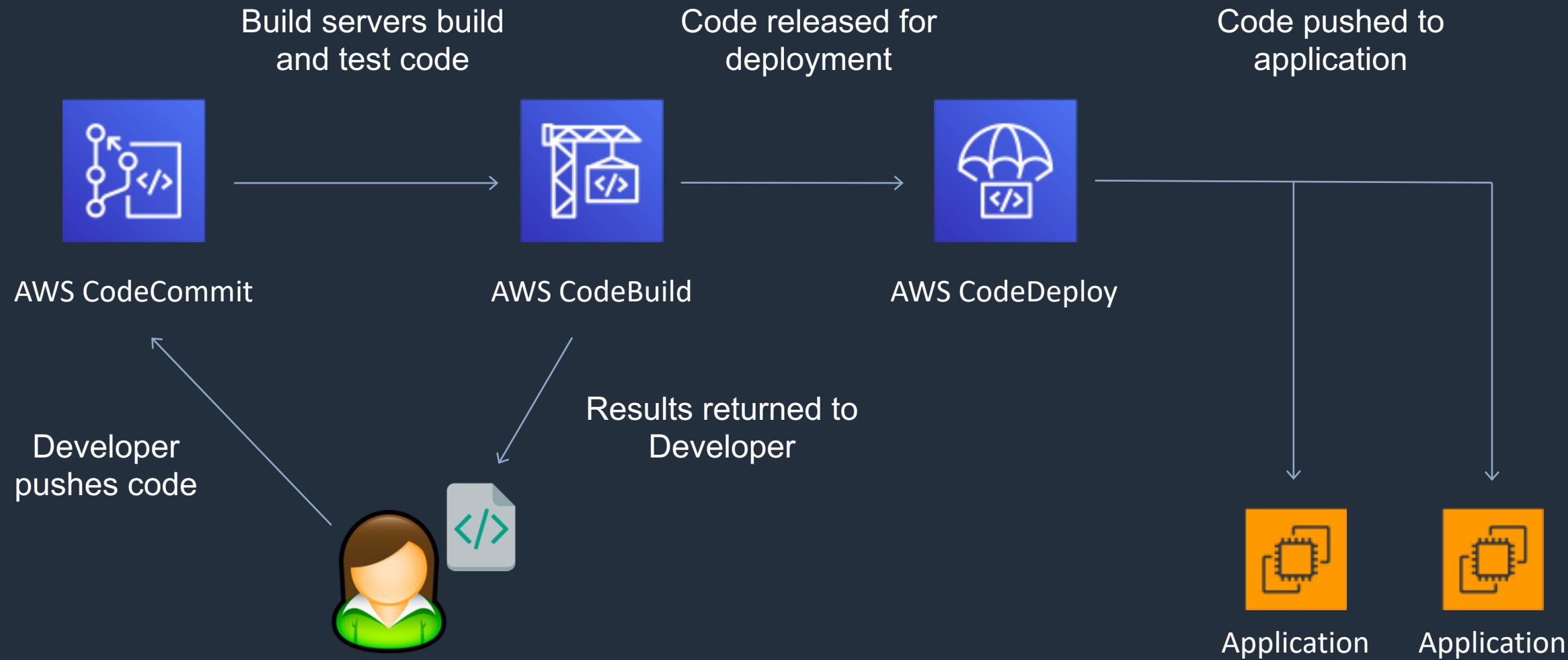
# Continuous Integration and Continuous Delivery (CI/CD)



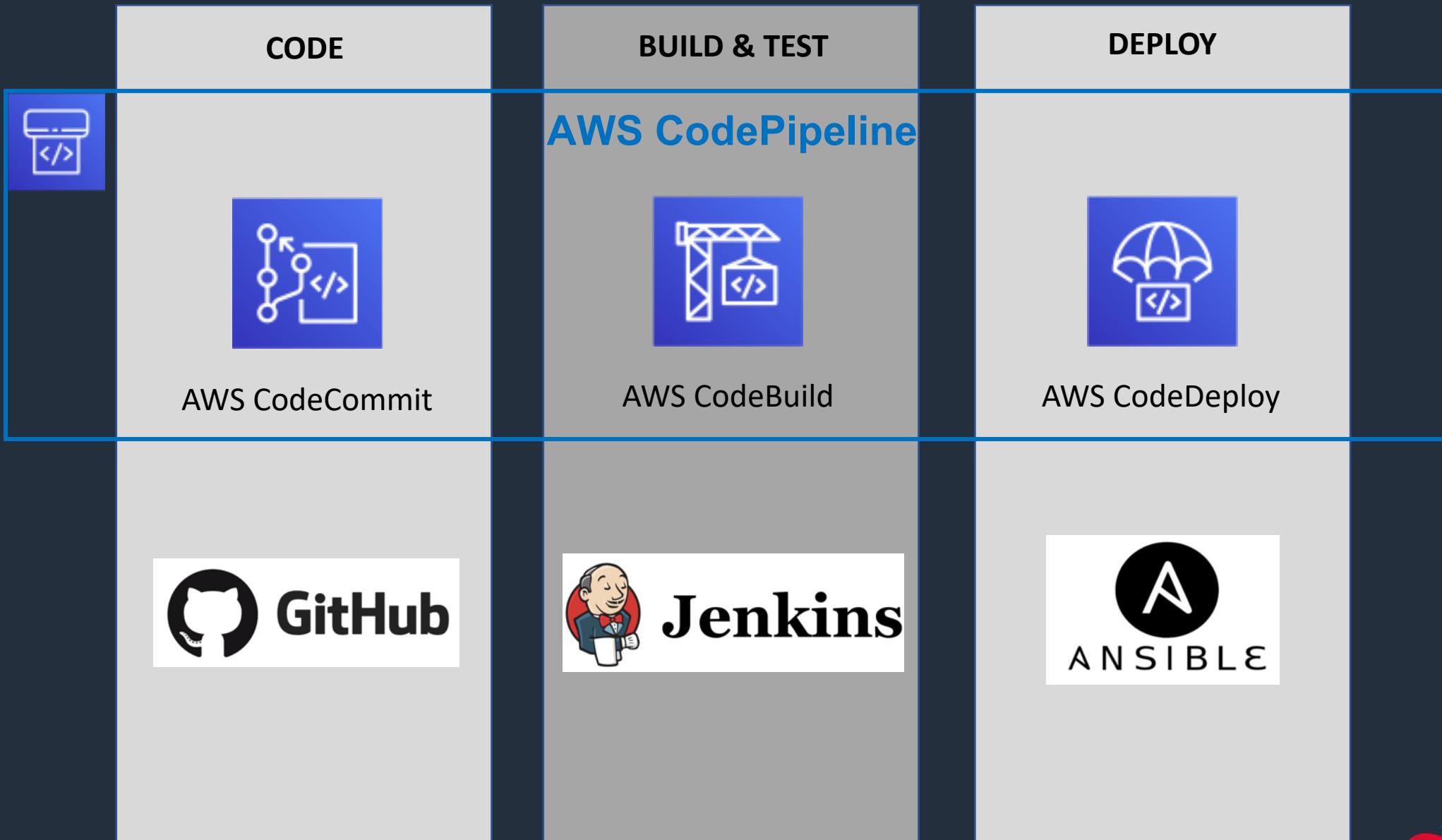
# Continuous Integration



# Continuous Integration & Continuous Delivery



# Continuous Integration & Continuous Delivery



## AWS CodeCommit

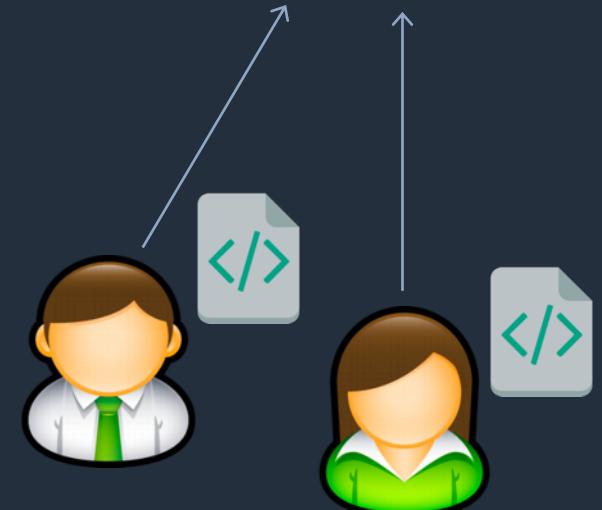
- AWS CodeCommit is a fully-managed source control service that hosts secure Git-based repositories.

- Git is an Open Source distributed source control system:

- Centralized repository for all of your code, binaries, images, and libraries.
- Tracks and manages code changes.
- Maintains version history.
- Manages updates from multiple sources.
- Enables collaboration.

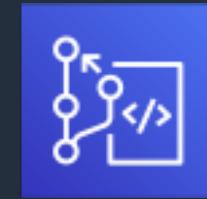


AWS CodeCommit

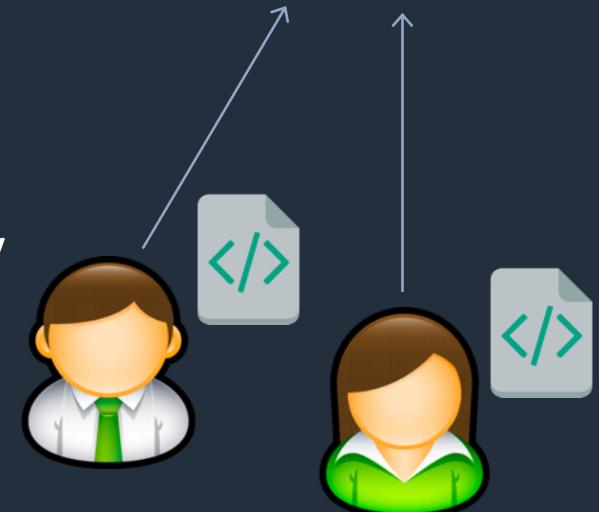


## AWS CodeCommit

- CodeCommit repositories are private.
- CodeCommit scales seamlessly.
- CodeCommit is integrated with Jenkins, CodeBuild and other CI tools.
- You can transfer your files to and from AWS CodeCommit using HTTPS or SSH.
- Repositories are automatically encrypted at rest through AWS Key Management Service (AWS KMS) using customer-specific keys.



AWS CodeCommit



## AWS CodeCommit - Authentication

- You need to configure your Git client to communicate with CodeCommit repositories.
- As part of this configuration, you provide IAM credentials that CodeCommit can use to authenticate you.
- IAM supports CodeCommit with three types of credentials:
  - Git credentials, an IAM-generated user name and password pair you can use to communicate with CodeCommit repositories over HTTPS.
  - SSH keys, a locally generated public-private key pair that you can associate with your IAM user to communicate with CodeCommit repositories over SSH.
  - AWS access keys, which you can use with the credential helper included with the AWS CLI to communicate with CodeCommit repositories over HTTPS.

## AWS CodePipeline

- AWS CodePipeline is a fully managed continuous delivery service that helps you automate your release pipelines for fast and reliable application and infrastructure updates.
- CodePipeline automates the build, test, and deploy phases of your release process every time there is a code change, based on the release model you define.
- CodePipeline provides tooling integrations for many AWS and third-party software at each stage of the pipeline including:
  - Source stage – S3, CodeCommit, Github, ECR, Bitbucket Cloud (beta).
  - Build – CodeBuild, Jenkins.
  - Deploy stage – CloudFormation, CodeDeploy, ECS, Elastic Beanstalk, AWS Service Catalog, S3.

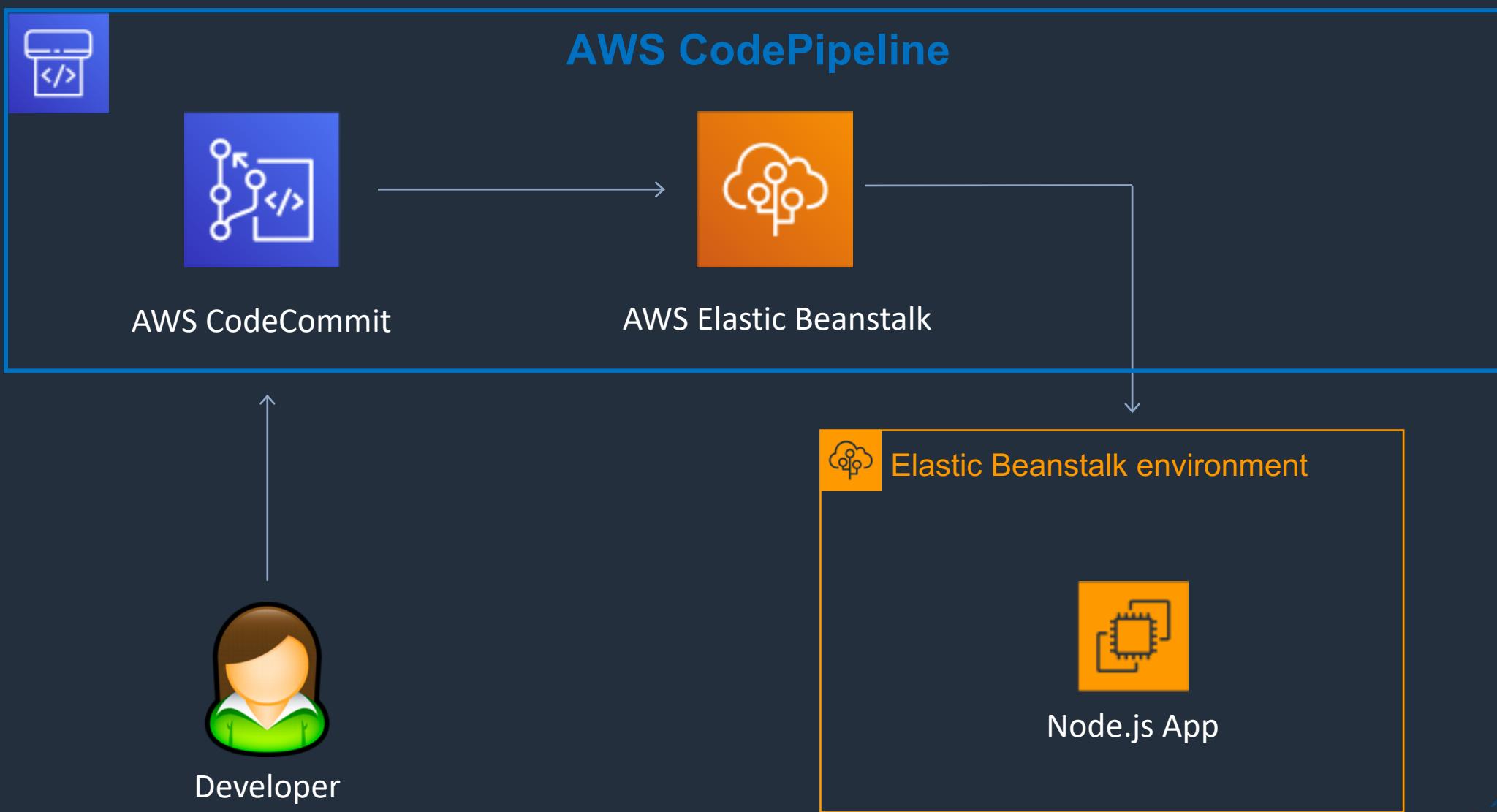
# AWS CodePipeline

- Pipelines:
  - A workflow that describes how software changes go through the release process.
- Artifacts:
  - Files or changes that will be worked on by the actions and stages in the pipeline.
  - Each pipeline stage can create "artifacts".
  - Artifacts are passed, stored in Amazon S3 and then passed on to the next stage.
- Stages:
  - Pipelines are broken up into stages.
  - Each stage can have sequential actions and or parallel actions.
  - Stage examples would be build, test, deploy, load test etc.
  - Manual approval can be defined at any stage.

# AWS CodePipeline

- Actions:
  - Stages contain at least one action.
  - Actions affect artifacts and will have artifacts as either an input, and output, or both.
- Transitions:
  - The progression from one stage to another inside of a pipeline.

# AWS CodePipeline with Elastic Beanstalk



## AWS CodeBuild

- AWS CodeBuild is a fully managed continuous integration (CI) service that compiles source code, runs tests, and produces software packages that are ready to deploy.
- With CodeBuild, you don't need to provision, manage, and scale your own build servers.
- CodeBuild scales continuously and processes multiple builds concurrently, so your builds are not left waiting in a queue.
- You pay based on the time it takes to complete the builds.
- CodeBuild takes source code from GitHub, CodeCommit, CodePipeline, S3 etc.
- Build instructions can be defined in the code (buildspec.yml).
- Output logs can be sent to Amazon S3 & AWS CloudWatch Logs.

## AWS CodeBuild

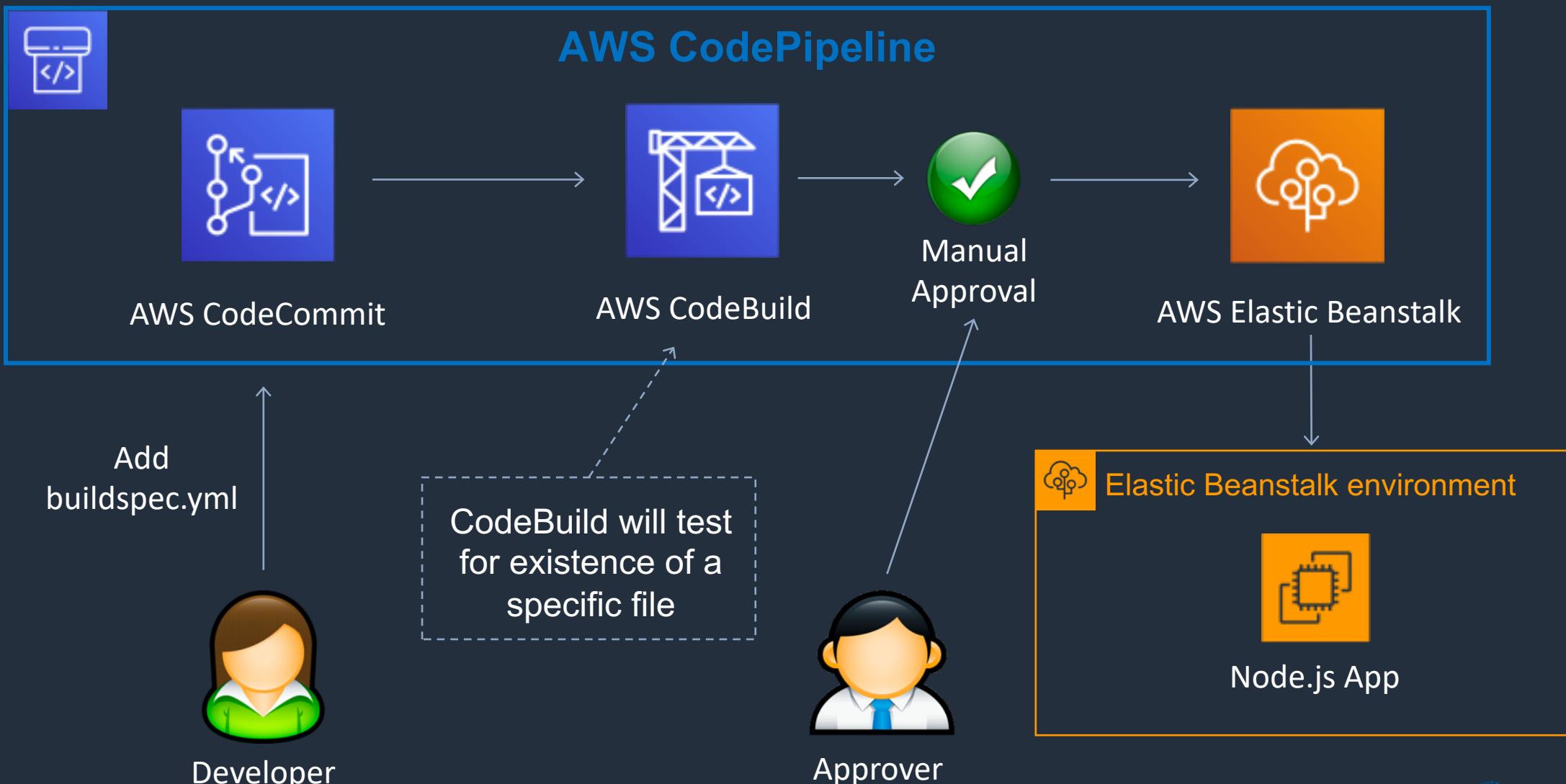
- **Build project** - defines how CodeBuild will run a build defines settings including:
  - Location of the source code.
  - The build environment to use.
  - The build commands to run.
  - Where to store the output of the build.
- **Build environment** - the operating system, language runtime, and tools that CodeBuild uses for the build.
- **Build Specification** - a YAML file that describes the collection of commands and settings for CodeBuild to run a build.

## AWS CodeBuild – buildspec.yml

```
version: 0.2

phases:
  install:
    commands:
      - echo "Entered the install phase..."
  pre_build:
    commands:
      - echo "Entered the pre_build phase..."
  build:
    commands:
      - echo "Entered the build phase..."
      - echo "Build started on `date`"
      - find production.txt
  post_build:
    commands:
      - echo "Entered the post_build phase..."
      - echo "Build completed on `date`"
```

# Continuous Integration & Continuous Delivery



## AWS CodeDeploy

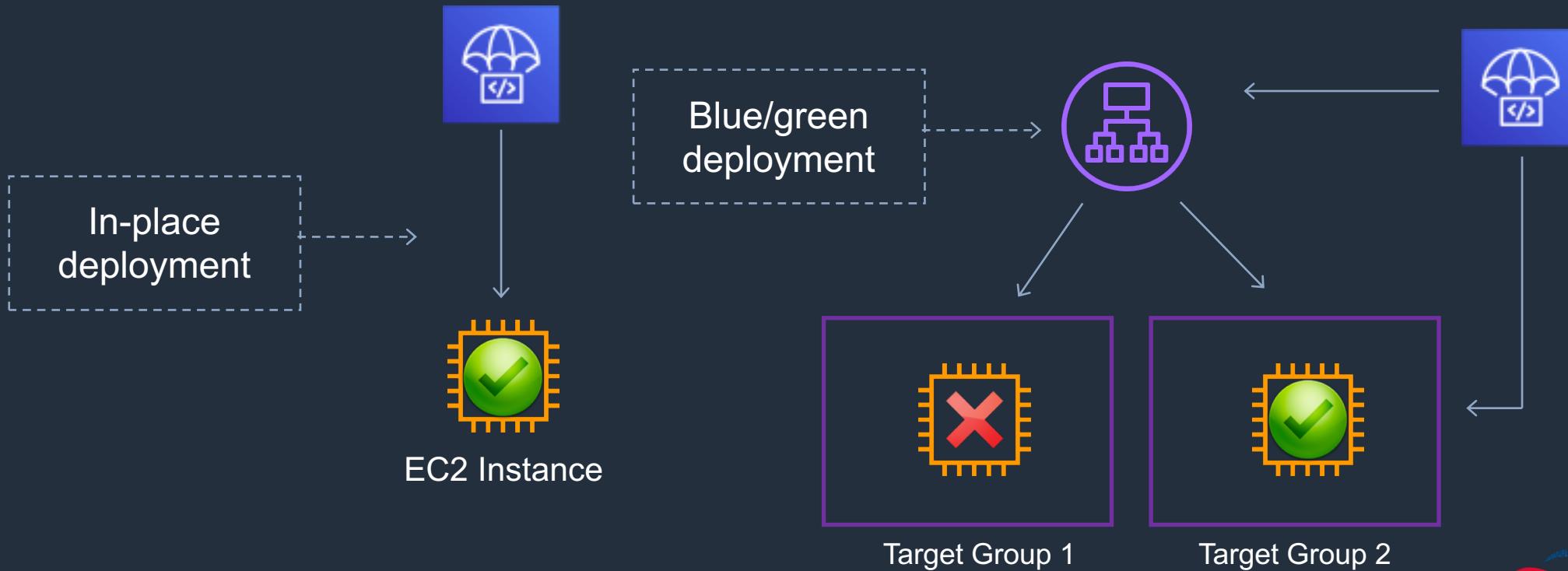
- CodeDeploy is a deployment service that automates application deployments to Amazon EC2 instances, on-premises instances, serverless Lambda functions, or Amazon ECS services.
- You can deploy a nearly unlimited variety of application content, including:
  - Serverless AWS Lambda functions.
  - Web and configuration files.
  - Executables.
  - Packages.
  - Scripts.
  - Multimedia files.

## AWS CodeDeploy

- An AWS CodeDeploy application contains information about what to deploy and how to deploy it.
- Need to choose the compute platform:
  - EC2/On-premises.
  - AWS Lambda.
  - Amazon ECS.

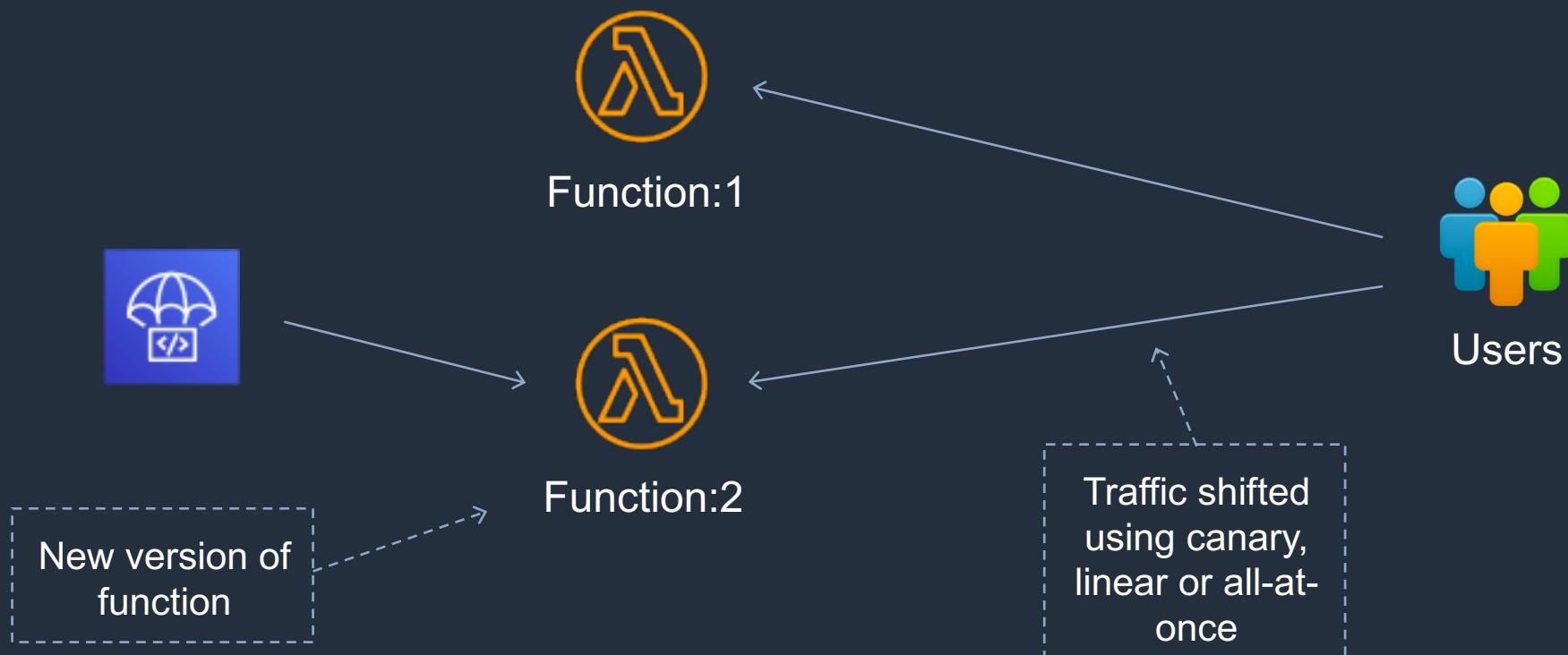
# AWS CodeDeploy

- EC2/On-Premises:
  - Amazon EC2 cloud instances, on-premises servers, or both.
  - Deployments that use the EC2/On-Premises compute platform manage the way in which traffic is directed to instances by using an in-place or blue/green deployment type.



# AWS CodeDeploy

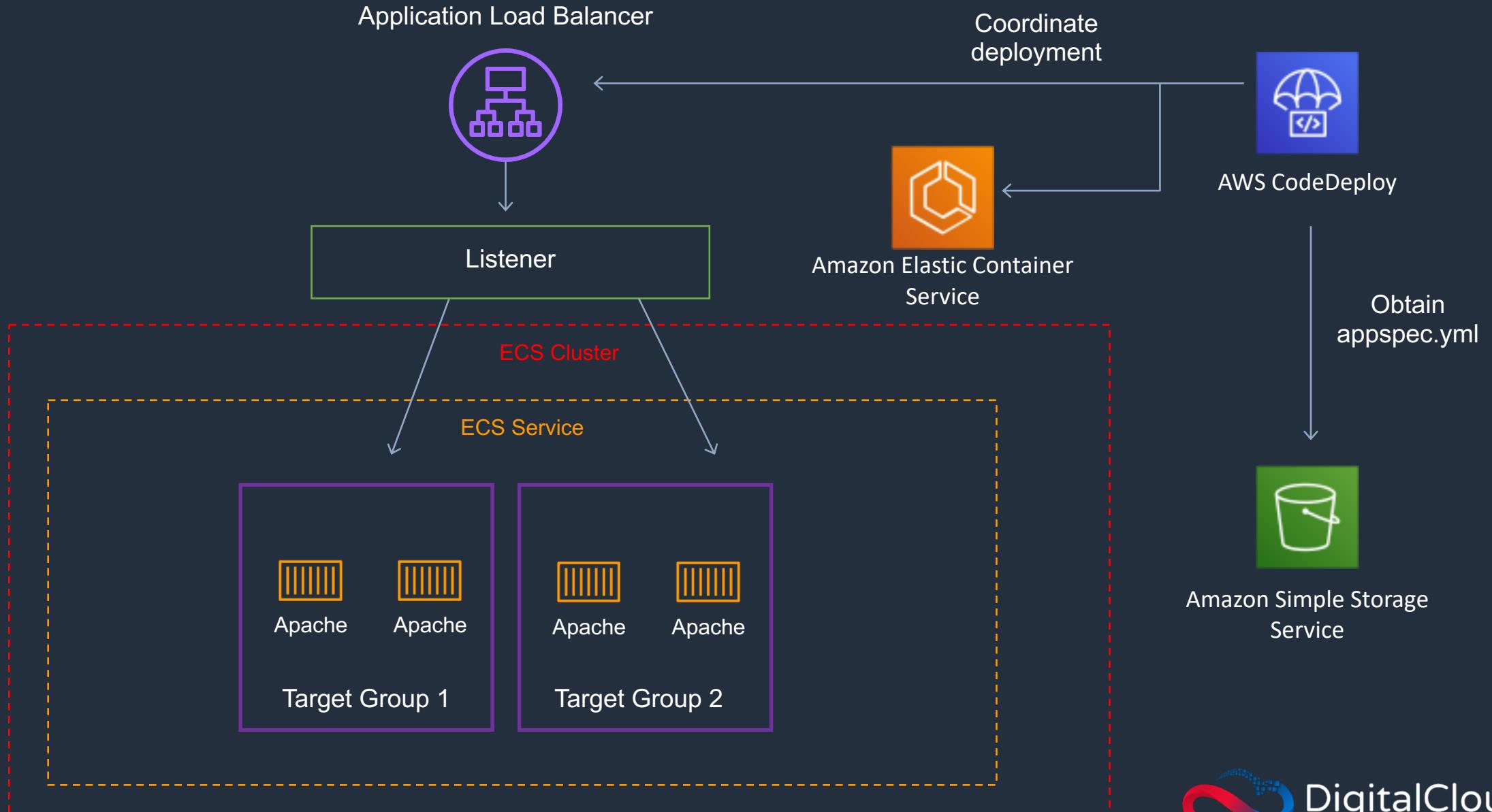
- AWS Lambda:
  - Used to deploy applications that consist of an updated version of a Lambda function.
  - You can manage the way in which traffic is shifted to the updated Lambda function versions during a deployment by choosing a canary, linear, or all-at-once configuration.



## AWS CodeDeploy

- Amazon ECS:
  - Used to deploy an Amazon ECS containerized application as a task set.
  - CodeDeploy performs a blue/green deployment by installing an updated version of the application as a new replacement task set.
  - CodeDeploy reroutes production traffic from the original application task set to the replacement task set.
  - The original task set is terminated after a successful deployment.
  - You can manage the way in which traffic is shifted to the updated task set during a deployment by choosing a canary, linear, or all-at-once configuration.

# AWS CodeDeploy – ECS Deployment



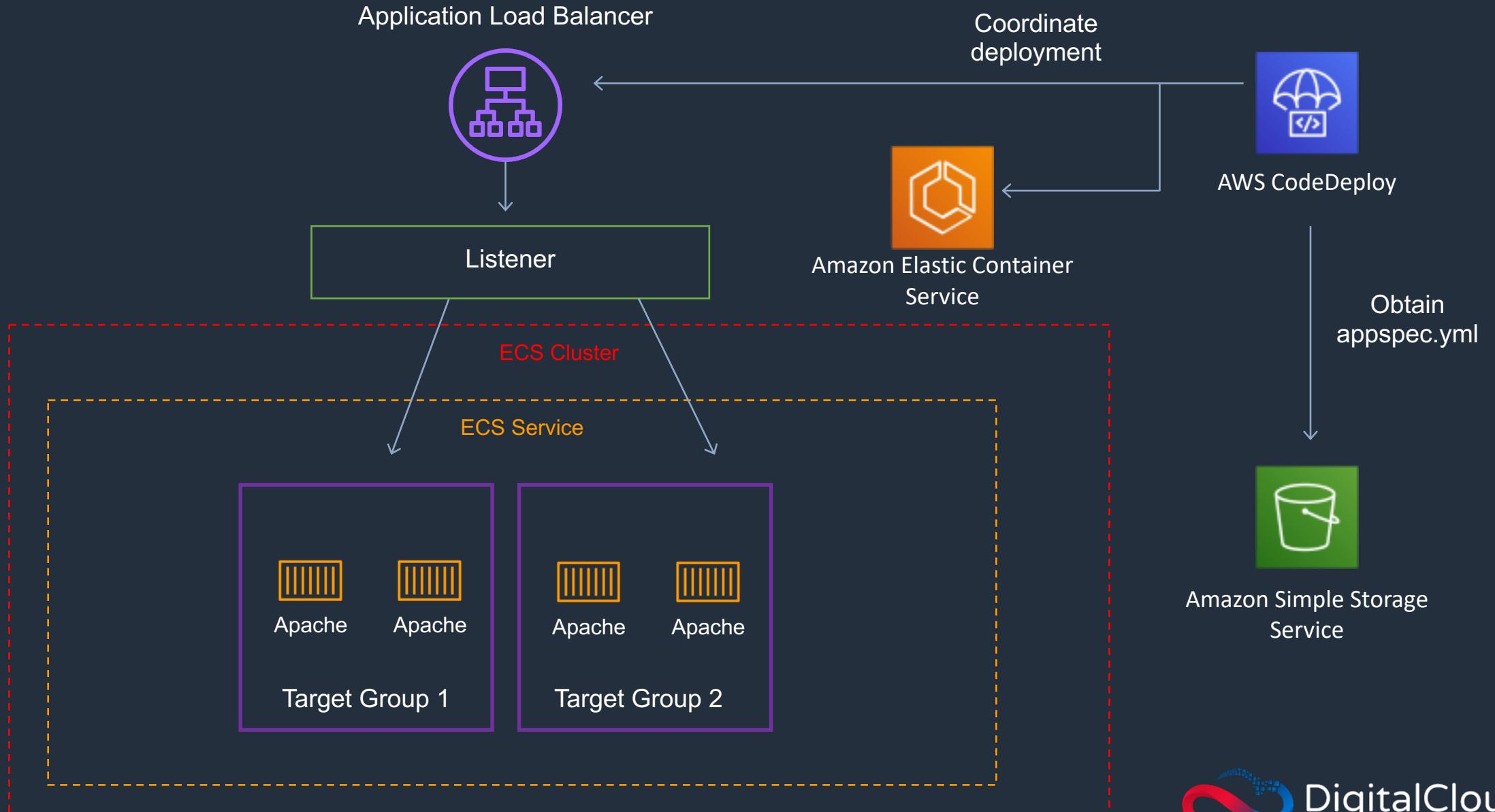
## AWS CodeDeploy – Blue/Green traffic shifting

- AWS Lambda: Traffic is shifted from one version of a Lambda function to a new version of the same Lambda function.
- Amazon ECS: Traffic is shifted from a task set in your Amazon ECS service to an updated, replacement task set in the same Amazon ECS service.
- EC2/On-Premises: Traffic is shifted from one set of instances in the original environment to a replacement set of instances.
- Note: All AWS Lambda and Amazon ECS deployments are blue/green. An EC2/On-Premises deployment can be in-place or blue/green.

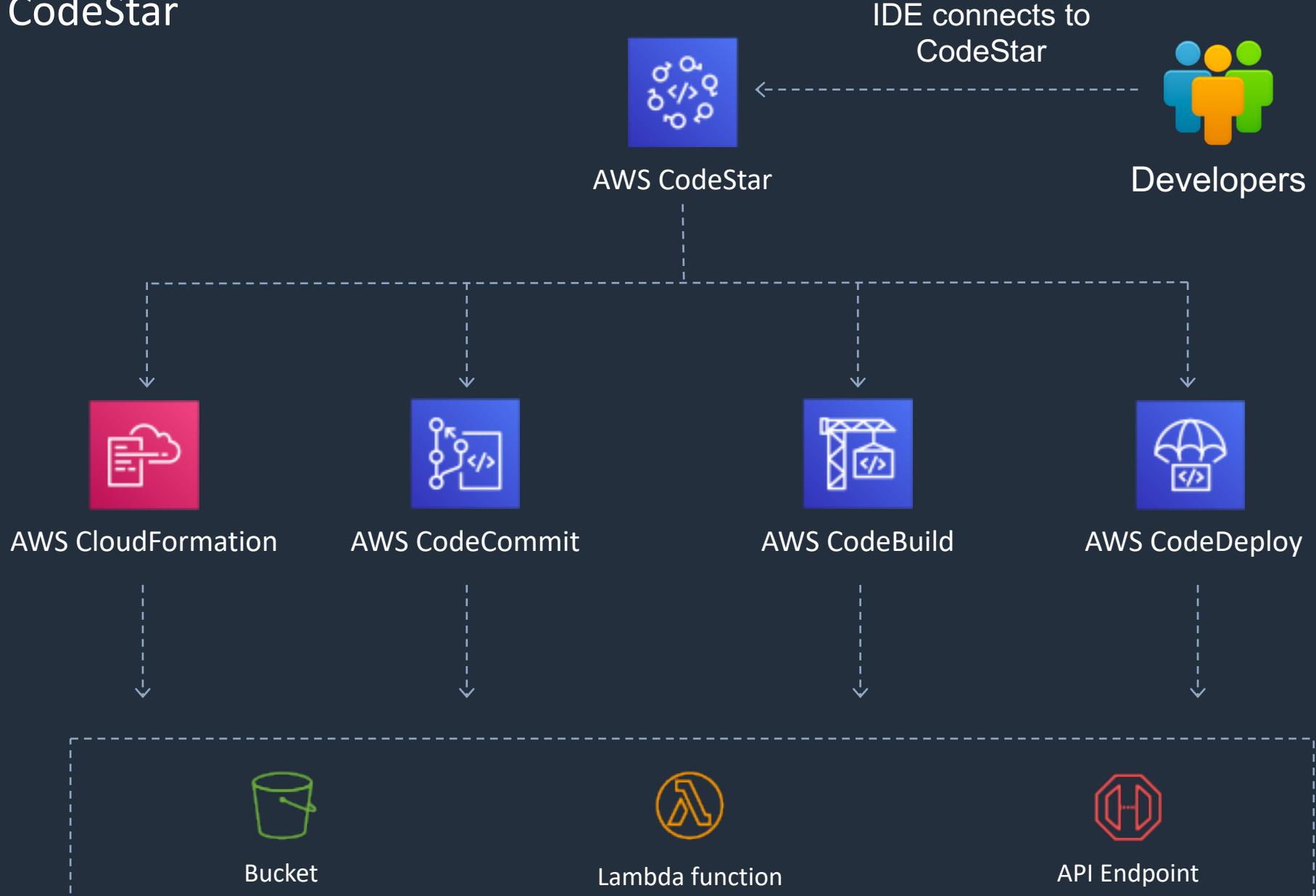
## AWS CodeDeploy – Blue/Green traffic shifting

- For Amazon ECS and AWS Lambda there are three ways traffic can be shifted during a deployment:
- Canary: Traffic is shifted in two increments. You can choose from predefined canary options that specify the percentage of traffic shifted to your updated Amazon ECS task set / Lambda function in the first increment and the interval, in minutes, before the remaining traffic is shifted in the second increment.
- Linear: Traffic is shifted in equal increments with an equal number of minutes between each increment. You can choose from predefined linear options that specify the percentage of traffic shifted in each increment and the number of minutes between each increment.
- All-at-once: All traffic is shifted from the original Amazon ECS task set / Lambda function to the updated Amazon ECS task set / Lambda function all at once.

# AWS CodeDeploy – ECS Deployment



# AWS CodeStar



## AWS CodeStar

- AWS CodeStar enables you to quickly develop, build, and deploy applications on AWS. AWS CodeStar provides a unified user interface, enabling you to easily manage your software development activities in one place.
- With AWS CodeStar, you can set up your entire continuous delivery toolchain in minutes, allowing you to start releasing code faster.
- AWS CodeStar makes it easy for your whole team to work together securely, allowing you to easily manage access and add owners, contributors, and viewers to your projects.
- Each AWS CodeStar project comes with a project management dashboard, including an integrated issue tracking capability powered by Atlassian JIRA Software.
- With the AWS CodeStar project dashboard, you can easily track progress across your entire software development process, from your backlog of work items to teams' recent code deployments.

# SECTION 14

## Amazon RDS and ElastiCache

# Database Types – Relational vs Non-Relational

Key differences are how data are **managed** and how data are **stored**

Relational	Non-Relational
Organized by tables, rows and columns	Varied data storage models
Rigid schema (SQL)	Flexible schema (NoSQL) – data stored in key-value pairs, columns, documents or graphs
Rules enforced within database	Rules can be defined in application code (outside database)
Typically scaled vertically	Scales horizontally
Supports complex queries and joins	Unstructured, simple language that supports any kind of schema
ACID (Atomicity, Consistency, Isolation, Durability) compliance typically enforced	Performance is typically prioritized, can use ACID transactions in some cases
Amazon RDS, Oracle, MySQL, IBM DB2, PostgreSQL	Amazon DynamoDB, MongoDB, Redis, Neo4j

# Relational Database (e.g. Amazon RDS)

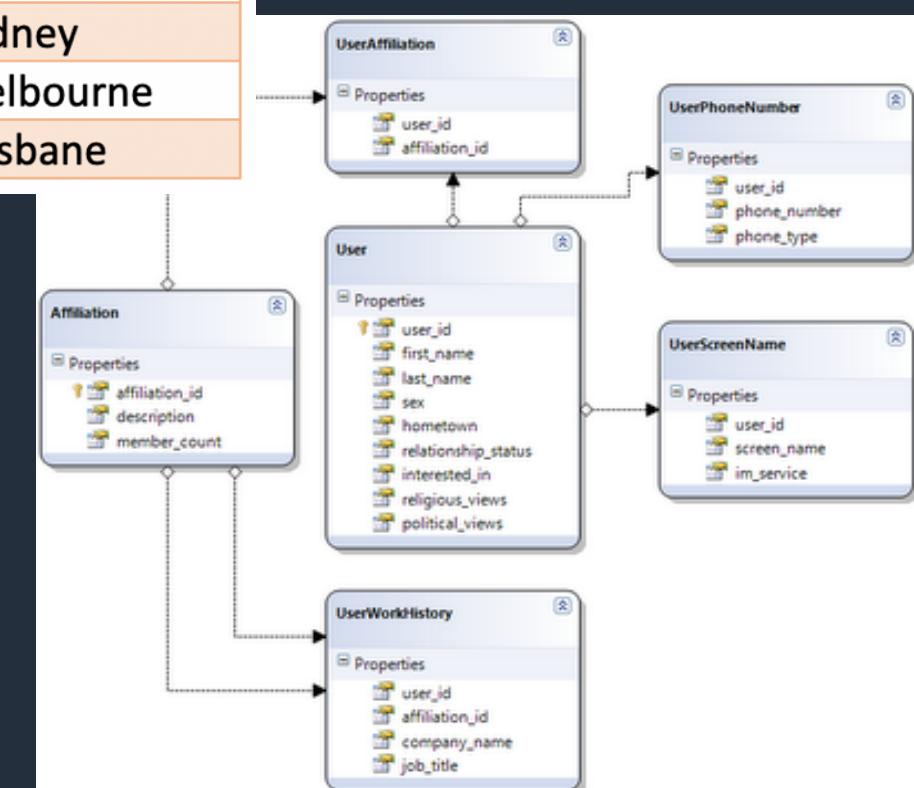
EmployeeID	FirstName	LastName	JobRole	Location
00001	Paul	Peterson	Senior Developer	Sydney
00002	Kaleigh	Annette	Assistant Manager	Brisbane
00003	Carl	Wood	Sales Support	Sydney
00004	Vinni	Jones	Customer Services	Melbourne
00005	Stefanie	Howard	IT Architect	Brisbane

Relational Database

Structured Query Language (SQL) query:

```
SELECT FirstName  
FROM employees  
WHERE Location = Sydney
```

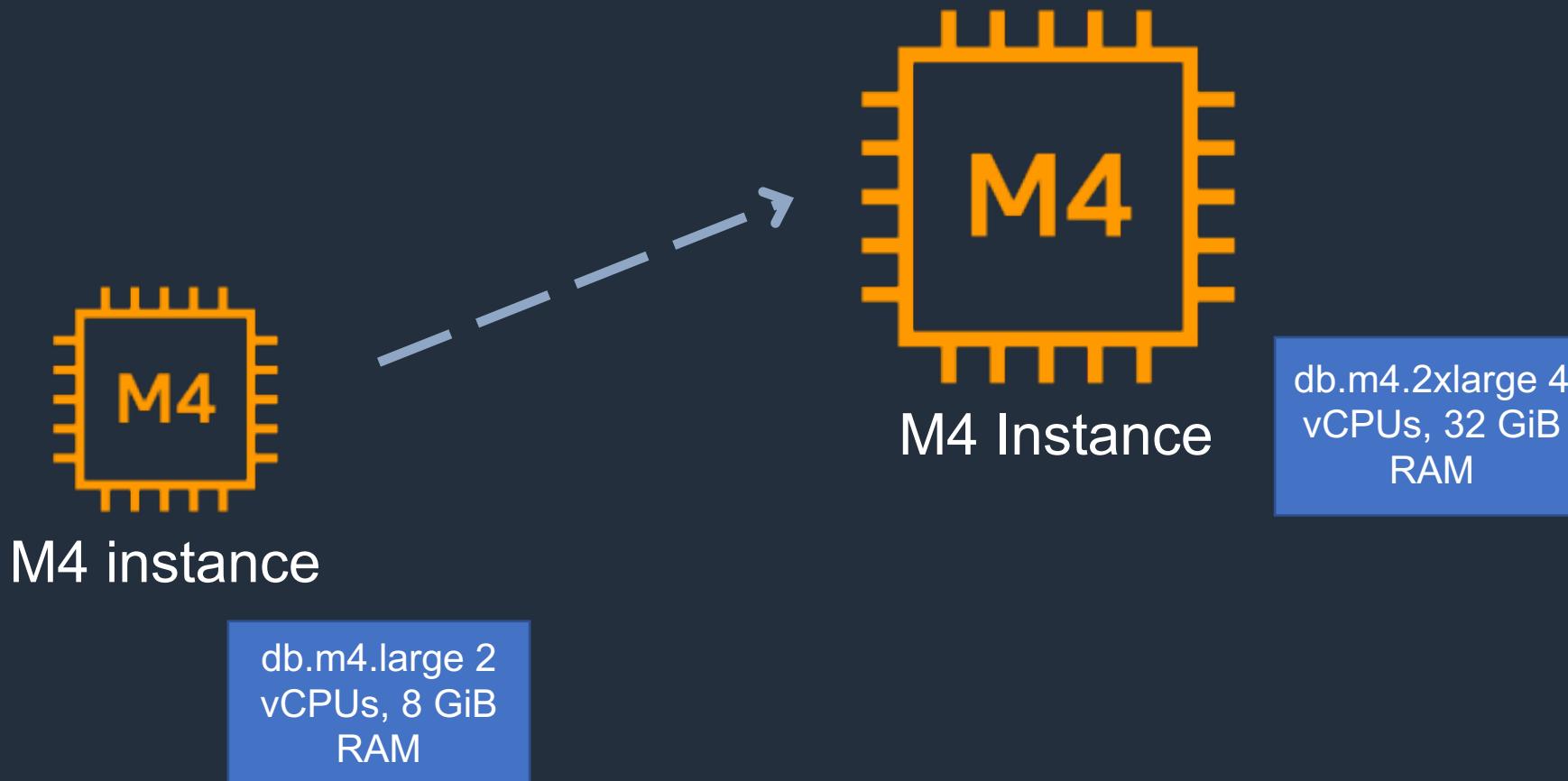
Relational Database – Multiple Tables



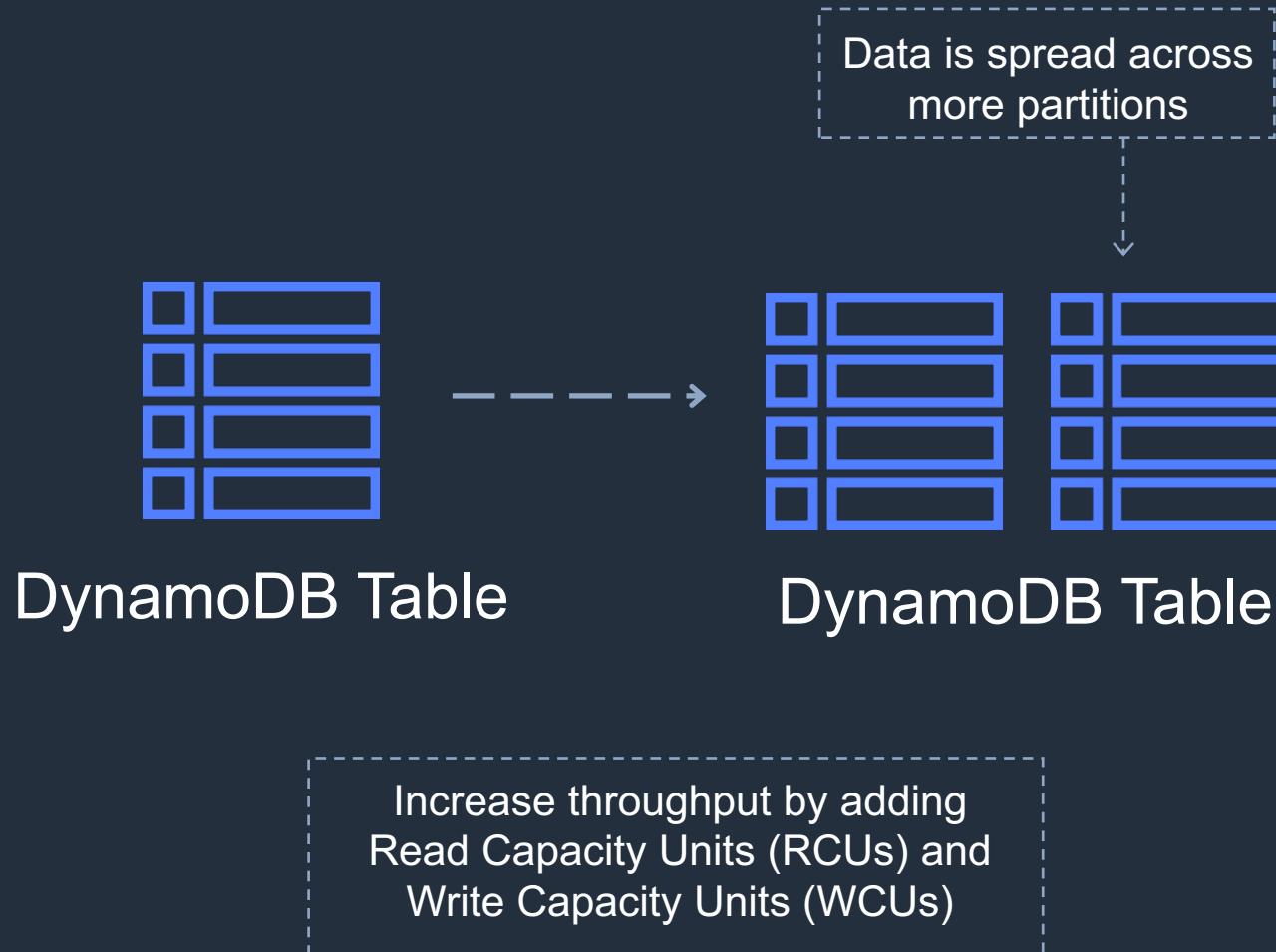
# Non-Relational Database (e.g. Amazon DynamoDB)

Primary Key		Attributes				
Partition Key	Sort Key	sku	category	size	colour	weight
clientid	created	SKU-S523	T-Shirt	Small	Red	Light
john@example.com	1583975308	SKU-J091	Pen		Blue	
chris@example.com	1583975613	SKU-A234	Mug			4011
sarah@example.com	1583976311	SKU-R873	Chair			
jenny@example.com	1583976323	SKU-I019	Plate	30		

# Scaling Vertically with Amazon RDS



# Scaling Horizontally with DynamoDB

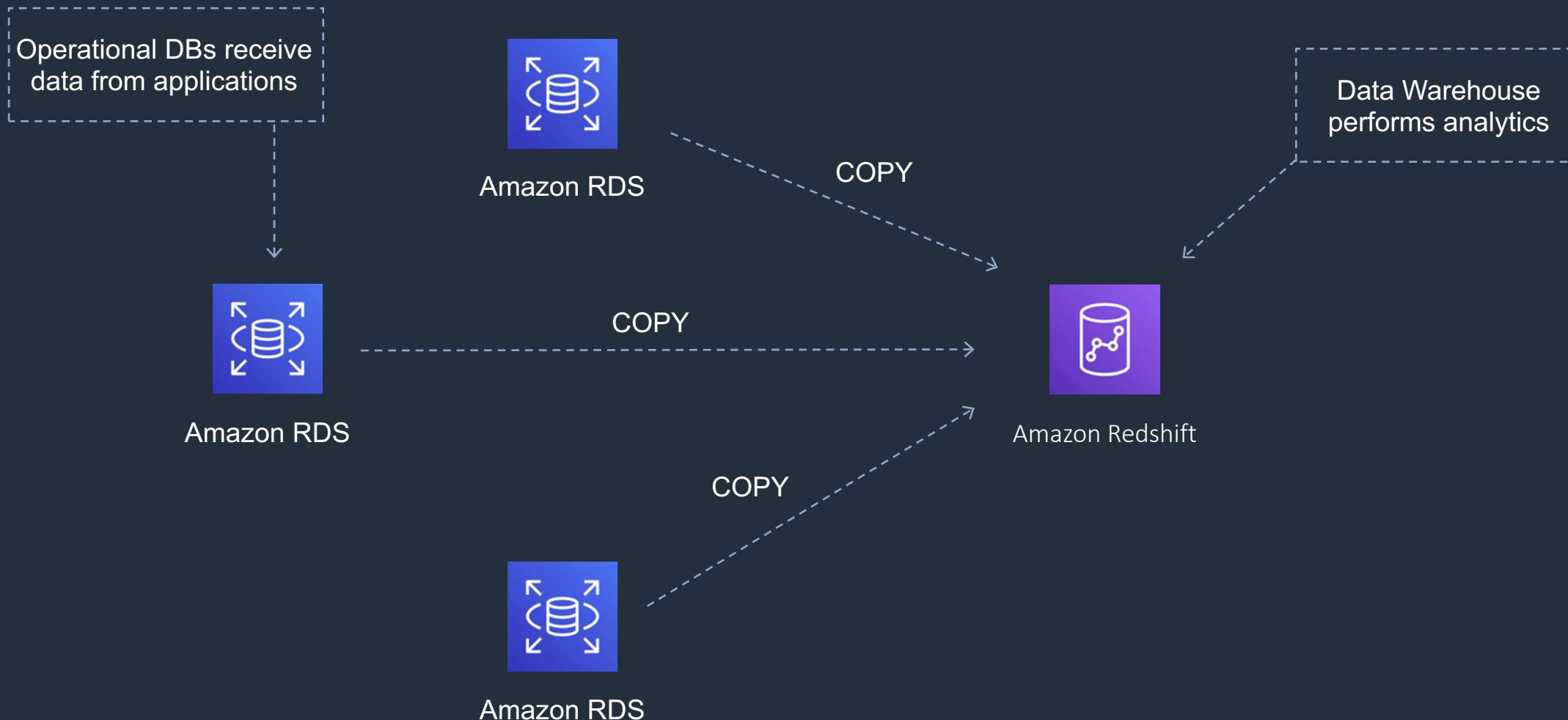


# Database Types – Operational vs Analytical

Key differences are **use cases** and how the database is **optimized**

Operational / transactional	Analytical
<b>Online Transaction Processing (OLTP)</b>	<b>Online Analytics Processing (OLAP)</b> – the source data comes from OLTP DBs
Production DBs that process transactions. E.g. adding customer records, checking stock availability ( <b>INSERT, UPDATE, DELETE</b> )	Data warehouse. Typically, separated from the customer facing DBs. Data is extracted for decision making
Short transactions and simple queries	Long transactions and complex queries
Relational examples: Amazon RDS, Oracle, IBM DB2, MySQL	Relational examples: Amazon RedShift, Teradata, HP Vertica
Non-relational examples: Amazon DynamoDB, MongoDB, Cassandra	Non-relational examples: Amazon EMR, MapReduce

# Database Types – Operational vs Analytical



# Databases – Architecture Discussion

Data Store	When to Use
Database on EC2	<ul style="list-style-type: none"><li>• Full control over instance and database</li><li>• Preferred DB not available under RDS</li></ul>
Amazon RDS	<ul style="list-style-type: none"><li>• Need traditional relational database for OLTP</li><li>• Your data is well-formed and structured</li></ul>
Amazon DynamoDB	<ul style="list-style-type: none"><li>• Name/value pair data</li><li>• Unpredictable data structure</li><li>• In-memory performance with persistence</li><li>• High I/O needs</li><li>• Require dynamic scaling</li></ul>
Amazon RedShift	<ul style="list-style-type: none"><li>• Data warehouse for large volumes of aggregated data</li><li>• Primarily OLAP workloads</li></ul>
Amazon ElastiCache	<ul style="list-style-type: none"><li>• Fast temporary storage for small amounts of data</li><li>• Highly volatile data (non-persistent)</li></ul>

## Amazon Relational Database Service (RDS)

- Amazon Relational Database Service (Amazon RDS) is a managed service that makes it easy to set up, operate, and scale a relational database in the cloud.
- RDS is an Online Transaction Processing (OLTP) type of database.
- Primary use case is a transactional database (rather than analytical).
- Best for structured, relational data store requirements.
- Aims to be drop-in replacement for existing on-premise instances of the same databases.
- Automated backups and patching applied in customer-defined maintenance windows.
- Push-button scaling, replication and redundancy.

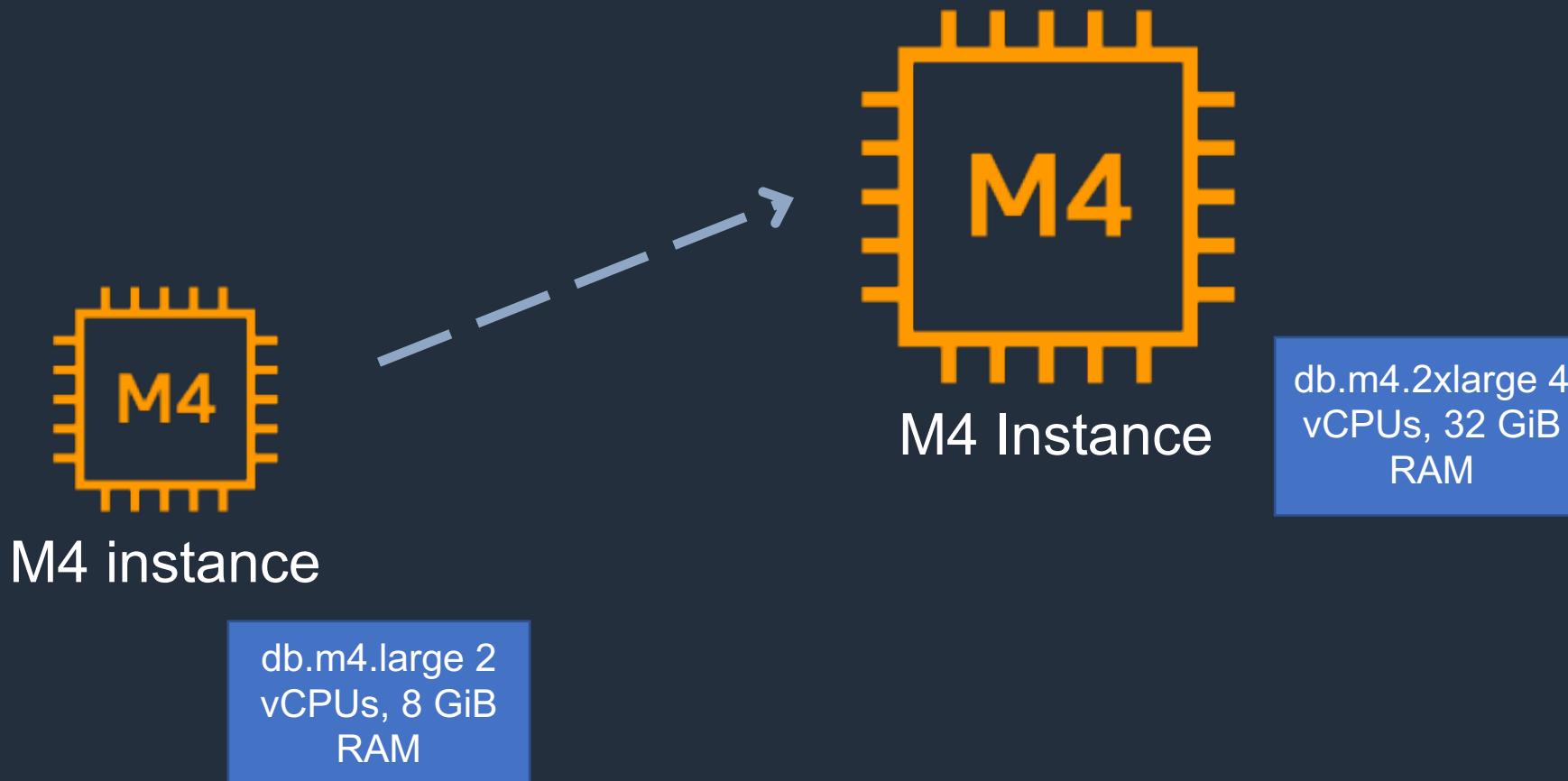
## Amazon Relational Database Service (RDS)

- Amazon RDS supports the following database engines:
- Amazon Aurora (proprietary AWS database engine).
- MySQL.
- MariaDB.
- Oracle.
- SQL Server.
- PostgreSQL.
- RDS is a managed service and you do not have access to the underlying EC2 instance (no root access).

## Amazon RDS – Scalability

- You can only scale RDS up (compute and storage).
- You cannot decrease the allocated storage for an RDS instance.
- You can scale storage and change the storage type for all DB engines except MS SQL.
- For MS SQL the workaround is to create a new instance from a snapshot with the new configuration.
- Scaling storage can happen while the RDS instance is running without outage however there may be performance degradation.
- Scaling compute will cause downtime.
- You can choose to have changes take effect immediately, however the default is within the maintenance window.
- Scaling requests are applied during the the specified maintenance window unless “apply immediately” is used.

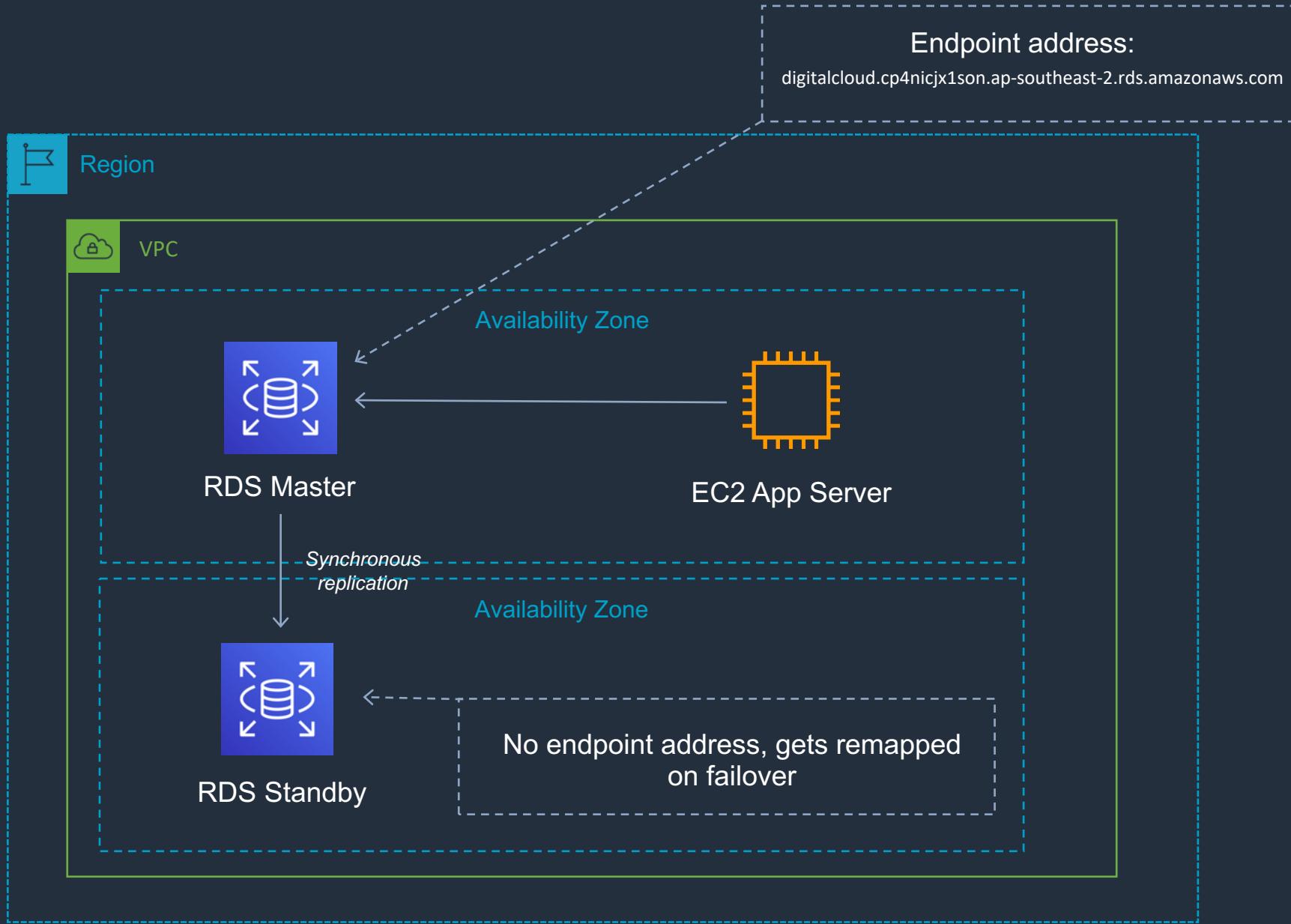
# Scaling Vertically with Amazon RDS



# Amazon RDS – Multi-AZ and Read Replicas

Multi-AZ Deployments	Read Replicas
Synchronous replication – highly durable	Asynchronous replication – highly scalable
Only database engine on primary instance is active	All read replicas are accessible and can be used for read scaling
Automatic failover to standby when a problem is detected	Can be manually promoted to a standalone database instance
Always span two Availability Zones within a single Region	Can be within an Availability Zone, Cross-AZ, or Cross-Region
Automated backups are taken from standby	No backups configured by default
Database engine version upgrades happen on primary	Database engine version upgrade is independent from source instance

# Amazon RDS Multi-AZ



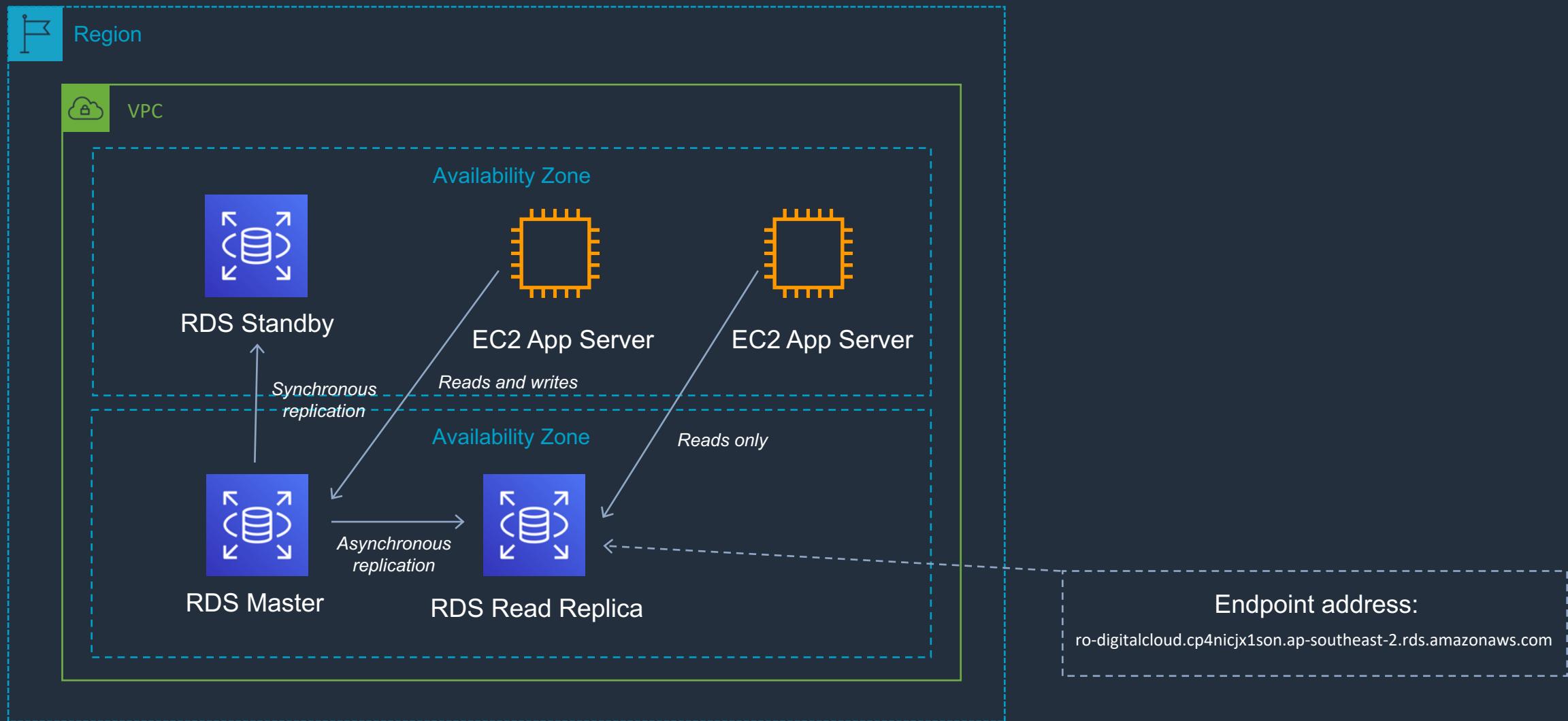
## Amazon RDS – Multi-AZ

- Multi-AZ RDS creates a replica in another AZ and synchronously replicates to it (DR only).
- Each AZ runs on its own physically distinct, independent infrastructure, and is engineered to be highly reliable.
- You cannot choose which AZ in the region will be chosen to create the standby DB instance.
- During failover RDS automatically updates configuration (including DNS endpoint) to use the second node.
- Depending on the instance class it can take 1 to a few minutes to failover to a standby DB instance.
- It is recommended to implement DB connection retries in your application.
- Recommended to use the endpoint rather than the IP address to point applications to the RDS DB.

## Amazon RDS – Multi-AZ

- The secondary DB in a multi-AZ configuration cannot be used as an independent read node (read or write).
- System upgrades like OS patching, DB Instance scaling and system upgrades, are applied first on the standby, before failing over and modifying the other DB Instance.
- In multi-AZ configurations snapshots and automated backups are performed on the standby to avoid I/O suspension on the primary instance.

# Amazon RDS Read Replicas



## Amazon RDS – Read Replicas

- Read replicas are used for read heavy DBs and replication is asynchronous.
- Read replicas are for workload sharing and offloading.
- Read replicas provide read-only DR.
- You can have 5 read replicas of a database.
- Read replicas are created from a snapshot of the master instance.
- Must have automated backups enabled on the primary (retention period > 0).
- Only supported for transactional database storage engines (InnoDB not MyISAM).
- Read replicas are available for MySQL, PostgreSQL, MariaDB, Oracle and Aurora (not SQL Server).
- In a multi-AZ failover the read replicas are switched to the new primary.

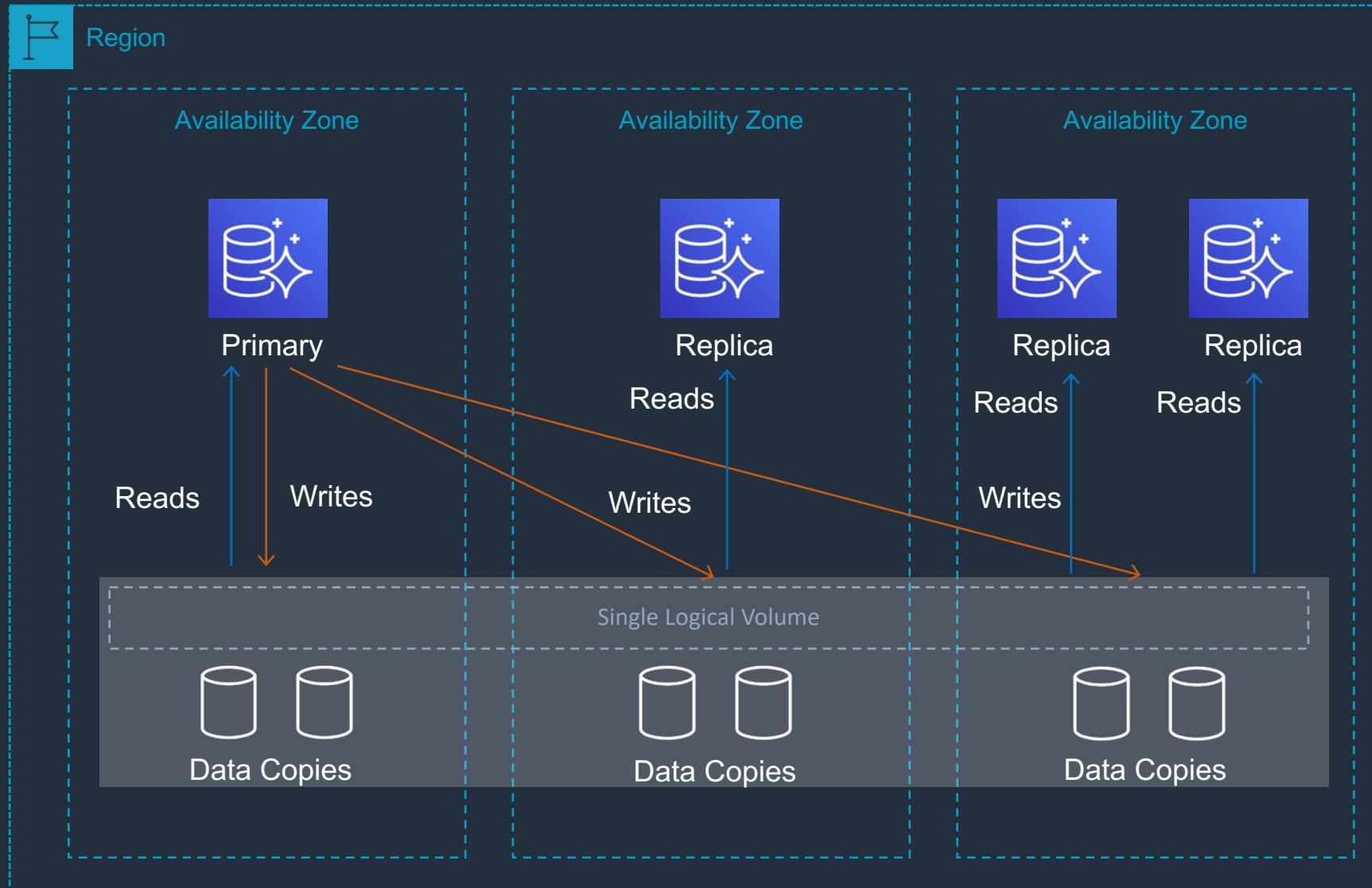
# Amazon RDS Aurora Key Features

Aurora Feature	Benefit
High performance and scalability	Offers high performance, self-healing storage that scales up to 64TB, point-in-time recovery and continuous backup to S3
DB compatibility	Compatible with existing MySQL and PostgreSQL open source databases
Aurora Replicas	In-region read scaling and failover target – up to 15 (can use Auto Scaling)
MySQL Read Replicas	Cross-region cluster with read scaling and failover target – up to 5 (each can have up to 15 Aurora Replicas)
Global Database	Cross-region cluster with read scaling (fast replication / low latency reads). Can remove secondary and promote
Multi-Master	Scales out writes within a region. In preview currently and will not appear on the exam
Serverless	On-demand, autoscaling configuration for Amazon Aurora - does not support read replicas or public IPs (can only access through VPC or Direct Connect - not VPN)

# Amazon RDS Aurora Replicas

Feature	Aurora Replica	MySQL Replica
Number of replicas	Up to 15	Up to 5
Replication type	Asynchronous (milliseconds)	Asynchronous (seconds)
Performance impact on primary	Low	High
Replica location	In-region	Cross-region
Act as failover target	Yes (no data loss)	Yes (potentially minutes of data loss)
Automated failover	Yes	No
Support for user-defined replication delay	No	Yes
Support for different data or schema vs. primary	No	Yes

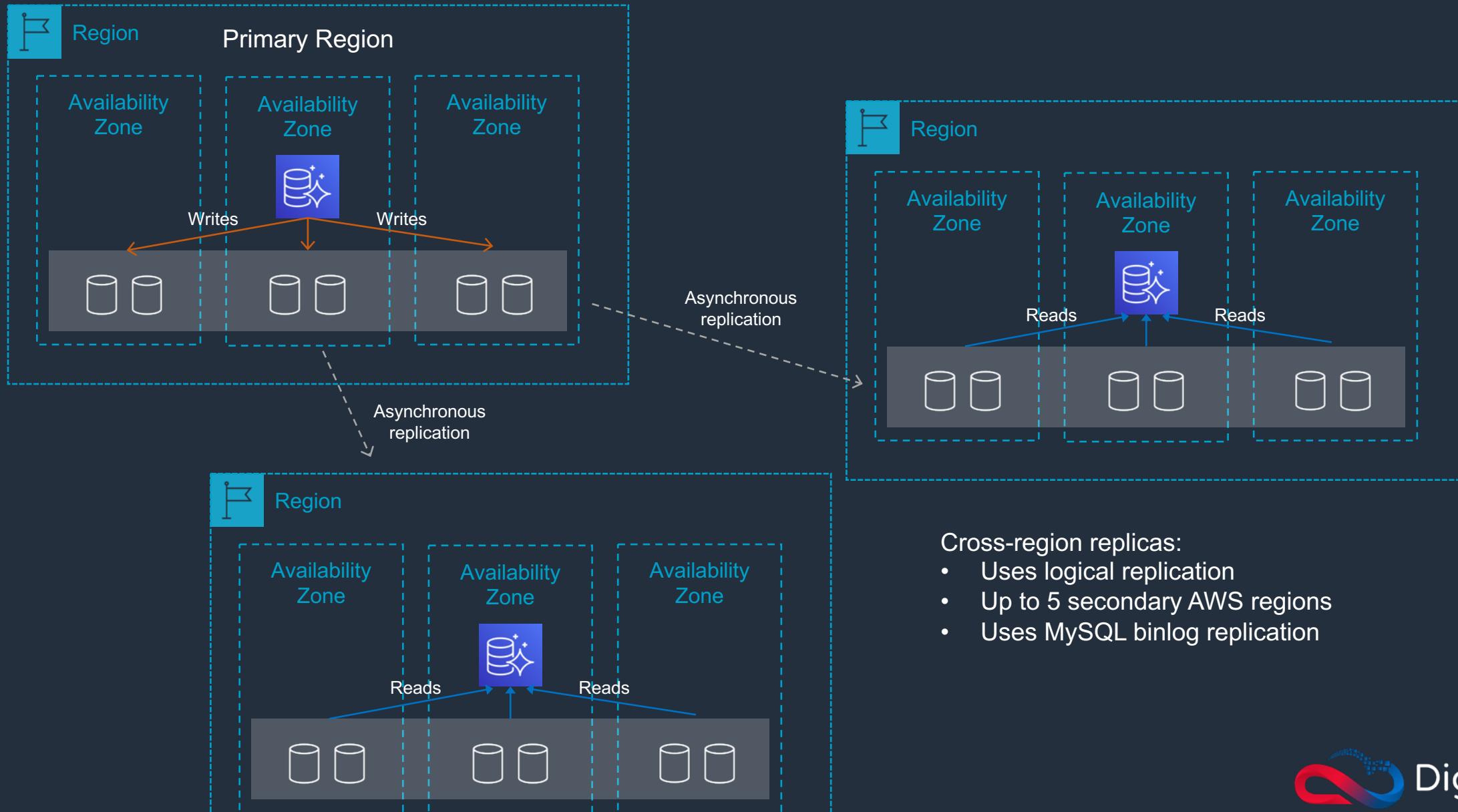
# Aurora Fault Tolerance and Aurora Replicas



## Aurora Fault Tolerance

- Fault tolerance across 3 AZs
- Single logical volume
- Aurora Replicas scale-out read requests
- Up to 15 Aurora Replicas with sub-10ms replica lag
- Aurora Replicas are independent endpoints
- Can promote Aurora Replica to be a new primary or create new primary
- Set priority (tiers) on Aurora Replicas to control order of promotion
- Can use Auto Scaling to add replicas

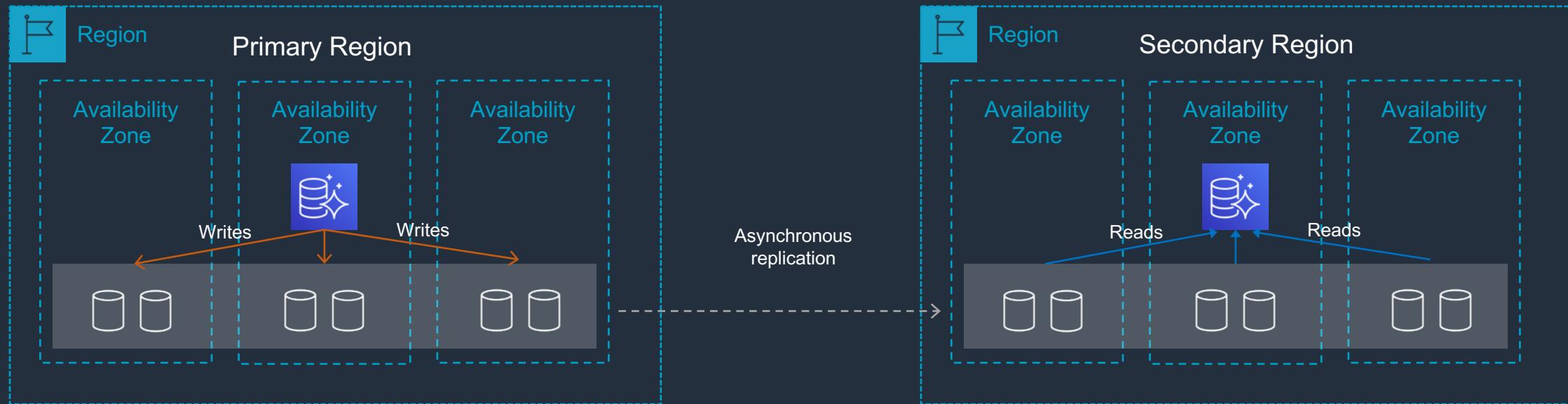
# Cross-Region Replica with Aurora MySQL



## Cross-region replicas:

- Uses logical replication
- Up to 5 secondary AWS regions
- Uses MySQL binlog replication

# Aurora Global Database

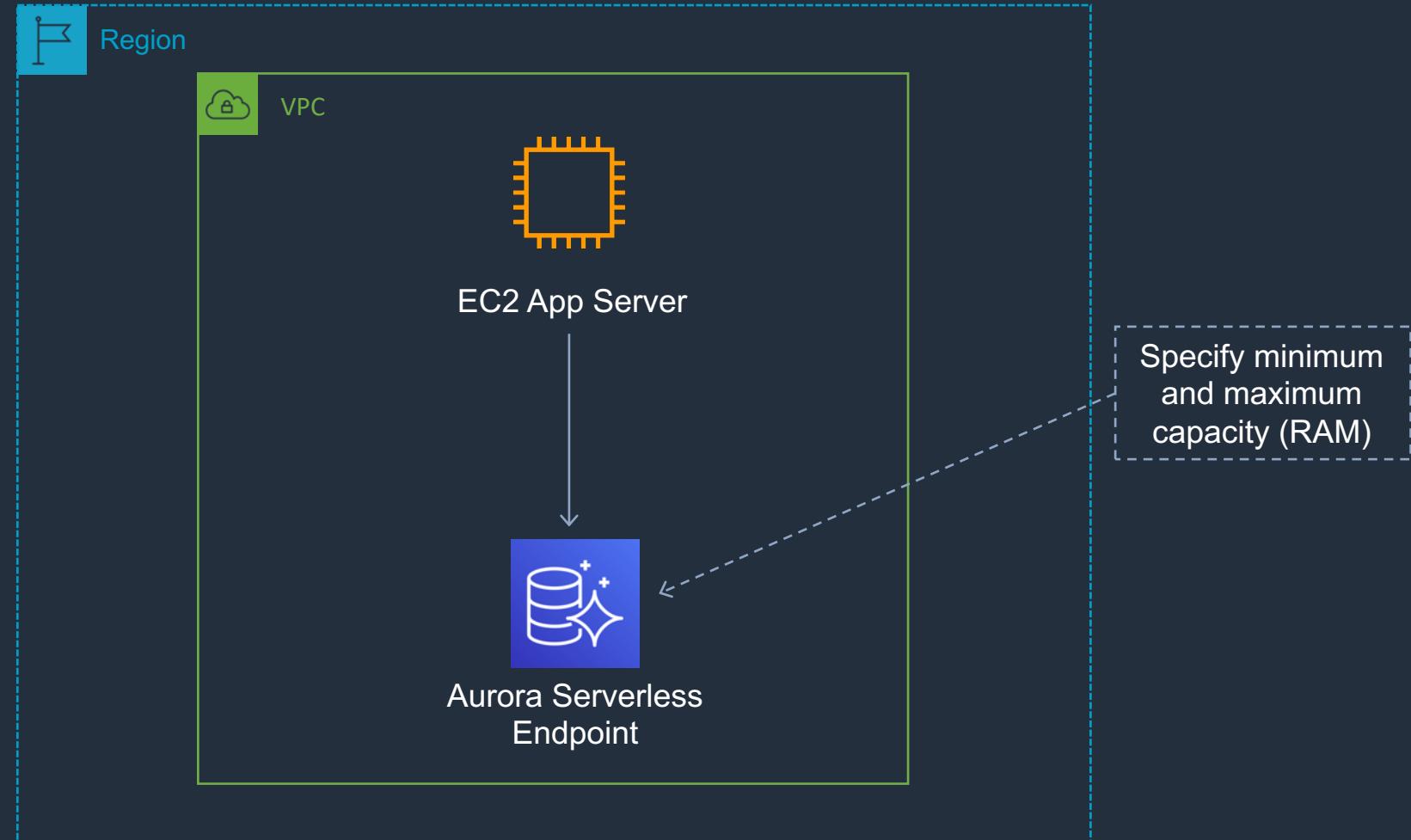


## Aurora Global Database:

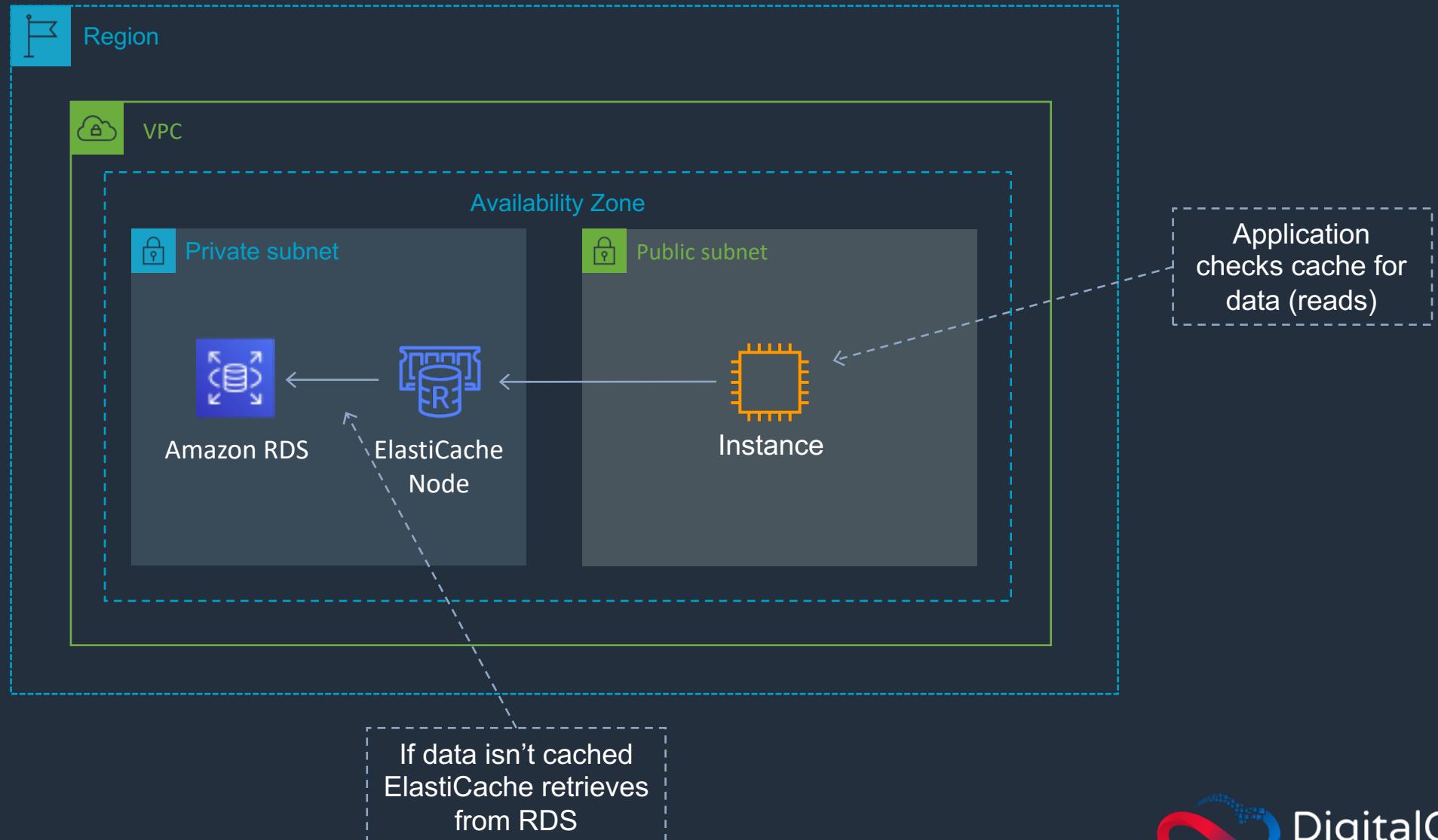
- Uses physical replication
- One secondary AWS region
- Uses dedicated infrastructure
- No impact on DB performance
- Good for disaster recovery

# Aurora Serverless

- On-demand auto-scaling.
- No DB instances to manage.
- Use for infrequently-used applications.
- Pay per second only when the DB is active.



# Amazon ElastiCache



## Amazon ElastiCache

- Fully managed implementations of two popular in-memory data stores – Redis and Memcached.
- ElastiCache is a web service that makes it easy to deploy and run Memcached or Redis protocol-compliant server nodes in the cloud.
- The in-memory caching provided by ElastiCache can be used to significantly improve latency and throughput for many read-heavy application workloads or compute-intensive workloads.
- Can be put in front of databases such as RDS and DynamoDB – sits between the application and the database.
- Good if your database is particularly read-heavy and the data does not change frequently.

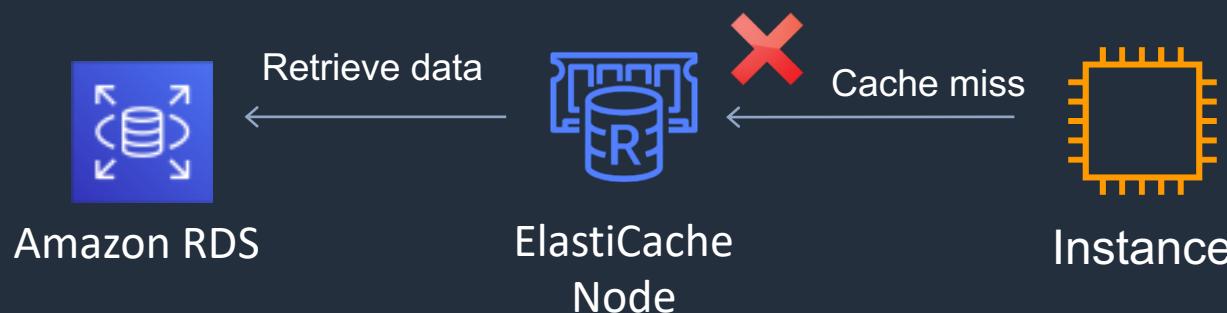
## Amazon ElastiCache

- Frequently-accessed data is stored in-memory for low-latency access, improving the overall performance of your application.
- Also good for compute heavy workloads such as recommendation engines and it can be used to store the results of I/O intensive database queries of compute-intensive calculations.
- ElastiCache can be used for storing session state.
- Push-button scalability for memory, writes and reads.
- In-memory key/value store – not persistent in the traditional sense.
- Billed by node size and hours of use.
- ElastiCache EC2 nodes cannot be accessed from the Internet, nor can they be accessed by EC2 instances in other VPCs.

# Amazon ElastiCache – Caching Strategies

## Lazy Loading

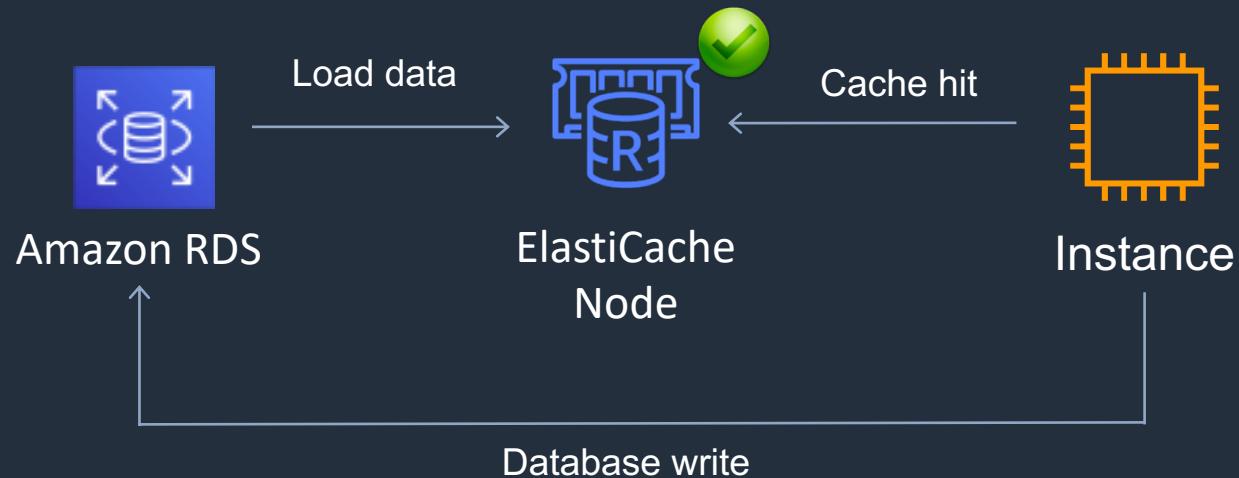
- Loads the data into the cache only when necessary (if a cache miss occurs).
- Lazy loading avoids filling up the cache with data that won't be requested.
- If requested data is in the cache, ElastiCache returns the data to the application.
- If the data is not in the cache or has expired, ElastiCache returns a null.
- The application then fetches the data from the database and writes the data received into the cache so that it is available for next time.
- Data in the cache can become stale if Lazy Loading is implemented without other strategies (such as TTL).



# Amazon ElastiCache – Caching Strategies

## Write Through

- When using a write through strategy, the cache is updated whenever a new write or update is made to the underlying database.
- Allows cache data to remain up-to-date.
- Can add wait time to write operations in your application.
- Without a Time To Live (TTL) you can end up with a lot of cached data that is never read.



# Amazon ElastiCache – Caching Strategies

## Dealing with stale data - Time to Live (TTL)

- The drawbacks of lazy loading and write through techniques can be mitigated by a TTL.
- The TTL specifies the number of seconds until the key (data) expires to avoid keeping stale data in the cache.
- When reading an expired key, the application checks the value in the underlying database.
- Lazy Loading treats an expired key as a cache miss and causes the application to retrieve the data from the database and subsequently write the data into the cache with a new TTL.
- Depending on the frequency with which data changes this strategy may not eliminate stale data - but helps to avoid it.

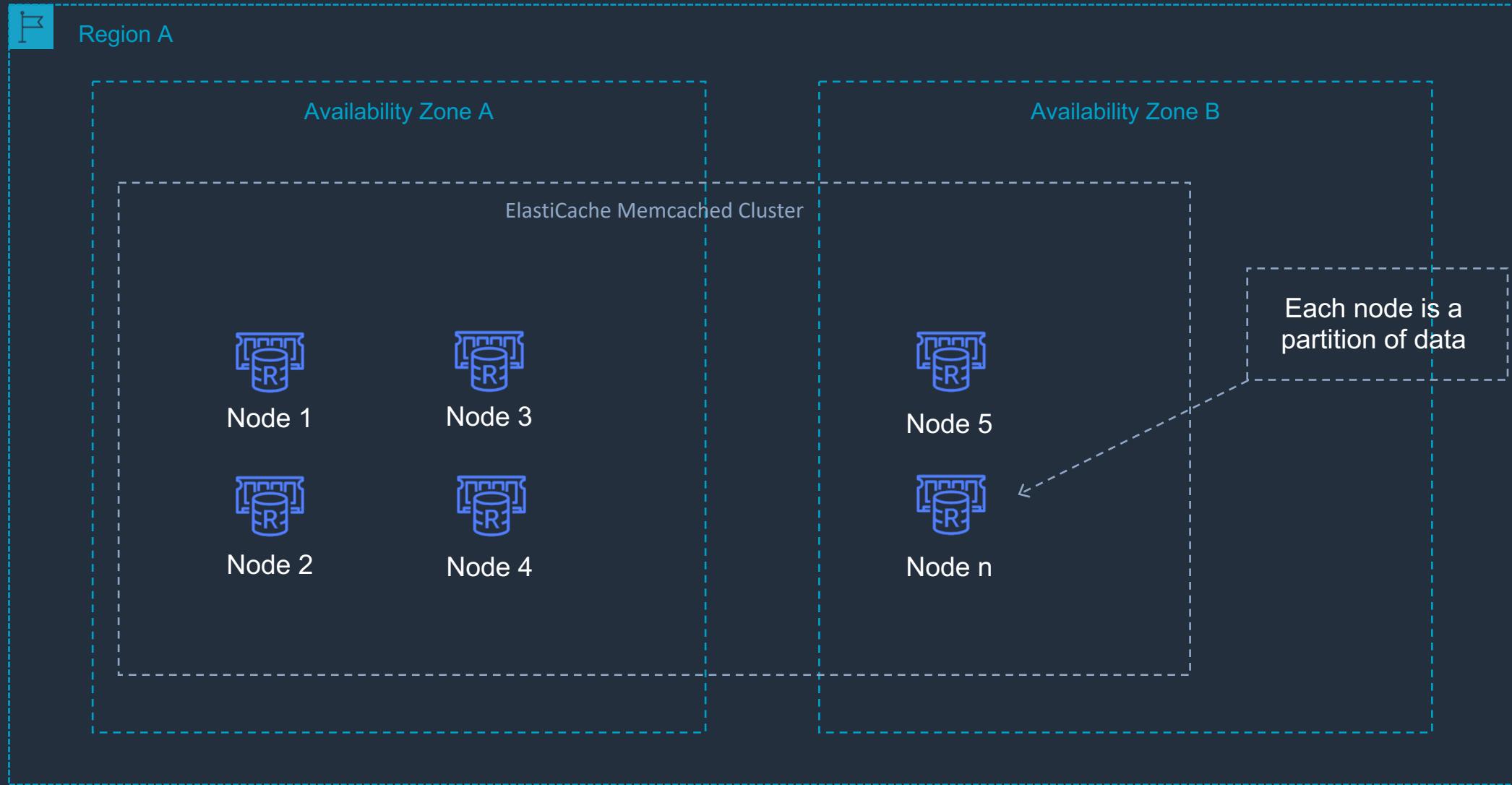
## Amazon ElastiCache

- Exam tip: the key use cases for ElastiCache are offloading reads from a Database, and storing the results of computations and session state. Also, remember that ElastiCache is an in-memory database and it's a managed service (so you can't run it on EC2).

# Amazon ElastiCache - Engines

Feature	Memcached	Redis (cluster mode disabled)	Redis (cluster mode enabled)
Data persistence	No	Yes	Yes
Data types	Simple	Complex	Complex
Data partitioning	Yes	No	Yes
Encryption	No	Yes	Yes
High availability (replication)	No	Yes	Yes
Multi-AZ	Yes, place nodes in multiple AZs. No failover or replication	Yes, with auto-failover. Uses read replicas (0-5 per shard)	Yes, with auto-failover. Uses read replicas (0-5 per shard)
Scaling	Up (node type); out (add nodes)	Single shard (can add replicas)	Add shards
Multithreaded	Yes	No	No
Backup and restore	No (and no snapshots)	Yes, automatic and manual snapshots	Yes, automatic and manual snapshots

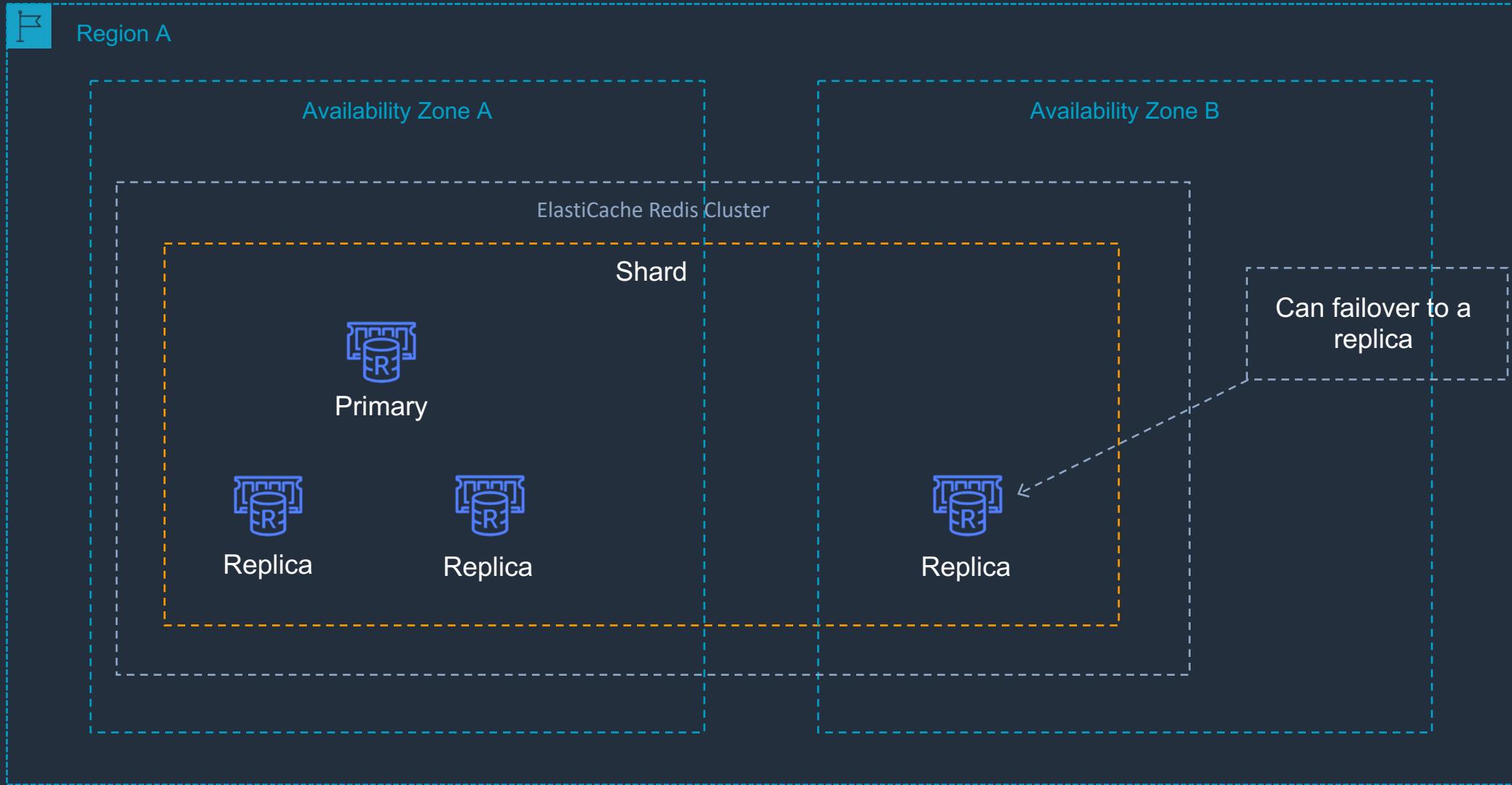
# Amazon ElastiCache Memcached



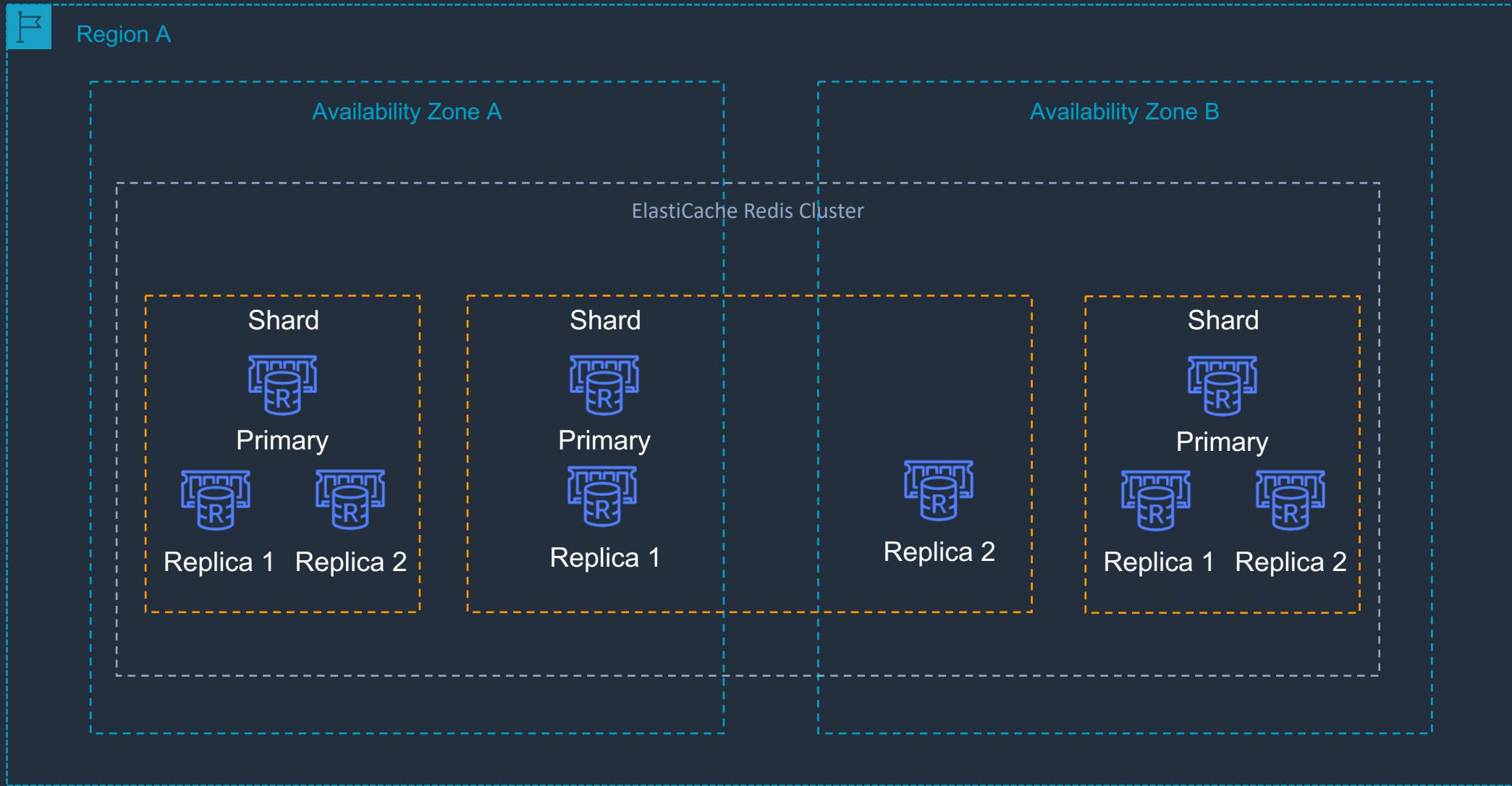
## Amazon ElastiCache - Memcached

- Simplest model and you can run large nodes.
- Memcached can be scaled in and out.
- Widely adopted memory object caching system.
- Multi-threaded.

# Amazon ElastiCache Redis (Cluster mode disabled)



# Amazon ElastiCache Redis (Cluster mode enabled)



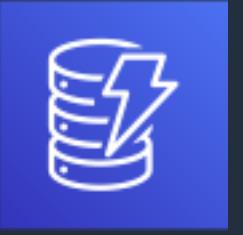
## Amazon ElastiCache - Redis

- Open-source in-memory key-value store.
- Supports more complex data structures: sorted sets and lists.
- Supports master / slave replication and multi-AZ for cross-AZ redundancy.
- Support automatic failover and backup/restore.

# SECTION 15

## Amazon DynamoDB

# Amazon DynamoDB Overview



- Amazon DynamoDB is a fully managed NoSQL database service that provides fast and predictable performance with seamless scalability.
- It is a non-relational, key-value type of database.
- DynamoDB is a serverless service – there are no instances to provision or manage.
- Push button scaling means that you can scale the DB at any time without incurring downtime.



DynamoDB Table



DynamoDB Table

## Amazon DynamoDB Overview



- Provides very low latency.
- Data is stored on SSD storage.
- Multi-AZ redundancy and Cross-Region Replication option.
- The underlying hardware storing data is spread across 3 geographically distinct data centres.

# Amazon DynamoDB Overview

- DynamoDB is made up of:

- Tables.
- Items.
- Attributes.

userid	orderid	book	price	date
user001	1000092	ISBN100..	9.99	2020.04..
user002	1000102	ISBN100..	24.99	2020.03..
user003	1000168	ISBN2X0..	12.50	2020.04..

## Amazon DynamoDB Overview

- DynamoDB Supports key-value and document structures.
- A key-value database stores data as a collection of key-value pairs in which a key serves as a unique identifier.
- Key = the name of the data; Value = the data itself.
- Documents can be written in JSON, HTML, or XML.

```
userid String : userid004
orderid String : 1000200
book String : ISBN100100
price Number : 9.99
date String : 2020-04-01T08:00Z
```

# Amazon DynamoDB Overview

DynamoDB Feature	Benefit
Serverless	Fully managed, fault tolerant, service
Highly available	99.99% availability SLA – 99.999% for Global Tables!
NoSQL type of database with Name / Value structure	Flexible schema, good for when data is not well structured or unpredictable
Horizontal scaling	Seamless scalability to any scale with push button scaling or Auto Scaling
DynamoDB Streams	Captures a time-ordered sequence of item-level modifications in a DynamoDB table and durably stores the information for up to 24 hours. Often used with Lambda and the Kinesis Client Library (KCL)
DynamoDB Accelerator (DAX)	Fully managed in-memory cache for DynamoDB that increases performance (microsecond latency)
Transaction options	Strongly consistent or eventually consistent reads, support for ACID transactions
Backup	Point-in-time recovery down to the second in last 35 days; On-demand backup and restore
Global Tables	Fully managed multi-region, multi-master solution

## Amazon DynamoDB – Access Control

- All authentication and access control is managed using IAM.
- DynamoDB supports identity-based policies:
  - Attach a permissions policy to a user or a group in your account.
  - Attach a permissions policy to a role (grant cross-account permissions).
- DynamoDB doesn't support resource-based policies.
- You can use a special IAM condition to restrict user access to only their own records.

## Amazon DynamoDB – Access Control

- In DynamoDB, the primary resources are tables.
- DynamoDB also supports additional resource types, indexes, and streams.
- You can create indexes and streams only in the context of an existing DynamoDB table (subresources).
- These resources and subresources have unique Amazon Resource Names (ARNs) associated with them, as shown in the following table.

Resource Type	ARN Format
Table	arn:aws:dynamodb:region:account-id:table/ <b>table-name</b>
Index	arn:aws:dynamodb:region:account-id:table/ <b>table-name</b> /index/ <b>index-name</b>
Stream	arn:aws:dynamodb:region:account-id:table/ <b>table-name</b> /stream/ <b>stream-label</b>

## Amazon DynamoDB – Access Control

- The following example policy grants permissions for one DynamoDB action (dynamodb>ListTables):

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "ListTables",  
            "Effect": "Allow",  
            "Action": [  
                "dynamodb>ListTables"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

## Amazon DynamoDB – Access Control

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "DescribeQueryScanBooksTable",  
            "Effect": "Allow",  
            "Action": [  
                "dynamodb:DescribeTable",  
                "dynamodb:Query",  
                "dynamodb:Scan"  
            ],  
            "Resource": "arn:aws:dynamodb:us-west-2:account-id:table/Books"  
        }  
    ]  
}
```

# Amazon DynamoDB – Supported Actions

- BatchGetItem
- BatchWriteItem
- CreateBackup
- CreateGlobalTable
- CreateTable
- DeleteBackup
- DeleteItem
- DeleteTable
- DescribeBackup
- DescribeContinuousBackups
- DescribeContributorInsights
- DescribeEndpoints
- DescribeGlobalTable
- DescribeGlobalTableSettings
- DescribeLimits
- DescribeTable
- DescribeTableReplicaAutoScaling
- DescribeTimeToLive
- GetItem
- ListBackups
- ListContributorInsights
- ListGlobalTables
- ListTables
- ListTagsOfResource
- PutItem
- Query
- RestoreTableFromBackup
- RestoreTableToPointInTime
- Scan
- TagResource
- TransactGetItems
- TransactWriteItems
- UntagResource
- UpdateContinuousBackups
- UpdateContributorInsights
- UpdateGlobalTable
- UpdateGlobalTableSettings
- UpdateItem
- UpdateTable
- UpdateTableReplicaAutoScaling
- UpdateTimeToLive

## Amazon DynamoDB – Access Control

- In addition to DynamoDB permissions, the console requires permissions from the following services:
  - Amazon CloudWatch permissions to display metrics and graphs.
  - AWS Data Pipeline permissions to export and import DynamoDB data.
  - AWS Identity and Access Management permissions to access roles necessary for exports and imports.
  - Amazon Simple Notification Service permissions to notify you whenever a CloudWatch alarm is triggered.
  - AWS Lambda permissions to process DynamoDB Streams records.

## Amazon DynamoDB – Access Control

- AWS Managed (Predefined) Policies for Amazon DynamoDB:
  - [AmazonDynamoDBReadOnlyAccess](#) – Grants read-only access to DynamoDB resources by using the AWS Management Console.
  - [AmazonDynamoDBFullAccess](#) – Grants full access to DynamoDB resources by using the AWS Management Console.
  - [AmazonDynamoDBFullAccesswithDataPipeline](#) – Grants full access to DynamoDB resources, including export and import using AWS Data Pipeline, by using AWS Management Console.

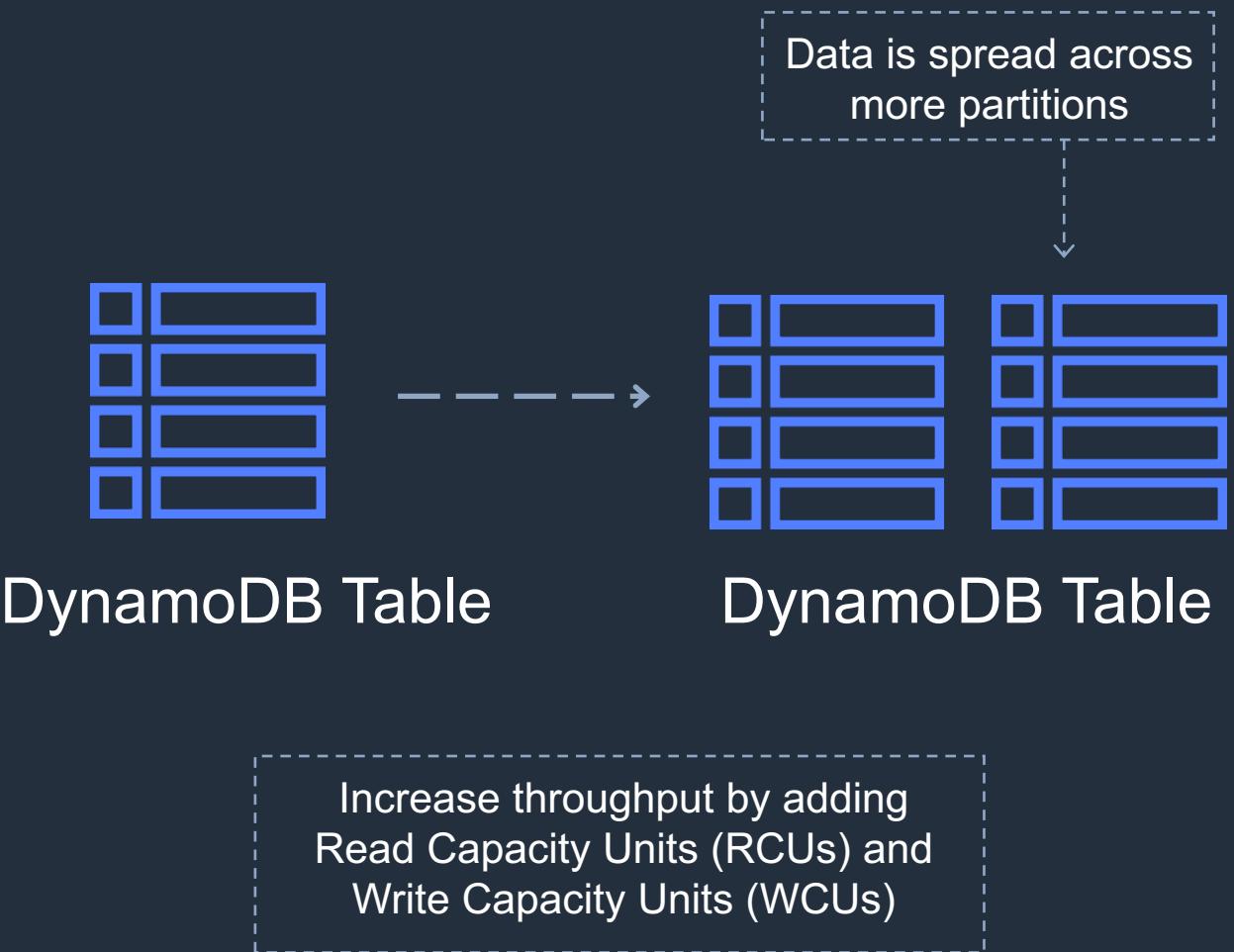
## Amazon DynamoDB – Partitions and Primary Keys

- Amazon DynamoDB stores data in partitions.
- A partition is an allocation of storage for a table that is automatically replicated across multiple AZs within an AWS Region.
- Partition management is handled entirely by DynamoDB—you never have to manage partitions yourself.
- DynamoDB allocates sufficient partitions to your table so that it can handle your provisioned throughput requirements.

# Amazon DynamoDB – Partitions and Primary Keys

DynamoDB allocates additional partitions to a table in the following situations:

- If you increase the table's provisioned throughput settings beyond what the existing partitions can support.
- If an existing partition fills to capacity and more storage space is required.



## Amazon DynamoDB – Partitions and Primary Keys

- There are two types of Primary key – Partition keys and composite keys
- Partition key - unique attribute (e.g. user ID).
  - Value of the Partition key is input to an internal hash function which determines the partition or physical location on which the data is stored.
  - If you are using the Partition key as your Primary key, then no two items can have the same partition key.

# Amazon DynamoDB – Table with a Partition Key

Partition Key	Attributes				
	postid	subject	message	lastposttime	replies
1000000572	Hello World	This code..	2020-04..	12	
1000001982	Advice on..	How can..	2020-03..	39	
1000000239	Help!	I need...	2020-01..	52	

## Amazon DynamoDB – Partitions and Primary Keys

- Composite key - Partition key + Sort key in combination.
- Example is user posting to a forum. Partition key would be the user ID, Sort key would be the timestamp of the post.
- 2 items may have the same Partition key, but they must have a different Sort key.
- All items with the same Partition key are stored together, then sorted according to the Sort key value.
- Allows you to store multiple items with the same partition key.

# Amazon DynamoDB – Table with a Composite Key

Primary Key		Attributes				
Partition Key	Sort Key	sku	category	size	colour	weight
clientid	created	SKU-S523	T-Shirt	Small	Red	Light
john@example.com	2020-03-9T08:12Z	SKU-J091	Pen		Blue	
chris@example.com	2020-03-10T14:30Z	SKU-A234	Mug	12		
sarah@example.com	2020-03-12T7:42Z	SKU-R873	Chair	94		4011
jenny@example.com	2020-03-13T18:29Z	SKU-I019	Plate	30		

## Amazon DynamoDB – Partitions and Primary Keys

- DynamoDB evenly distributes provisioned throughput—read capacity units (RCUs) and write capacity units (WCUs) among partitions
- If your access pattern exceeds 3000 RCU or 1000 WCU for a single partition key value, your requests might be throttled.
- Reading or writing above the limit can be caused by these issues:
  - Uneven distribution of data due to the wrong choice of partition key.
  - Frequent access of the same key in a partition (the most popular item, also known as a hot key).
  - A request rate greater than the provisioned throughput.

## Amazon DynamoDB – Partitions and Primary Keys

Best practices for partition keys:

- Use high-cardinality attributes – e.g. e-mailid, employee\_no, customerid, sessionid, orderid, and so on.
- Use composite attributes – e.g. customerid+productid+countrycode as the partition key and order\_date as the sort key.
- Cache popular items – use DynamoDB accelerator (DAX) for caching reads.
- Add random numbers or digits from a predetermined range for write-heavy use cases
  - e.g. add a random suffix to an invoice number such as INV00023-**04593**

## Amazon DynamoDB – Consistency Models and Transactions

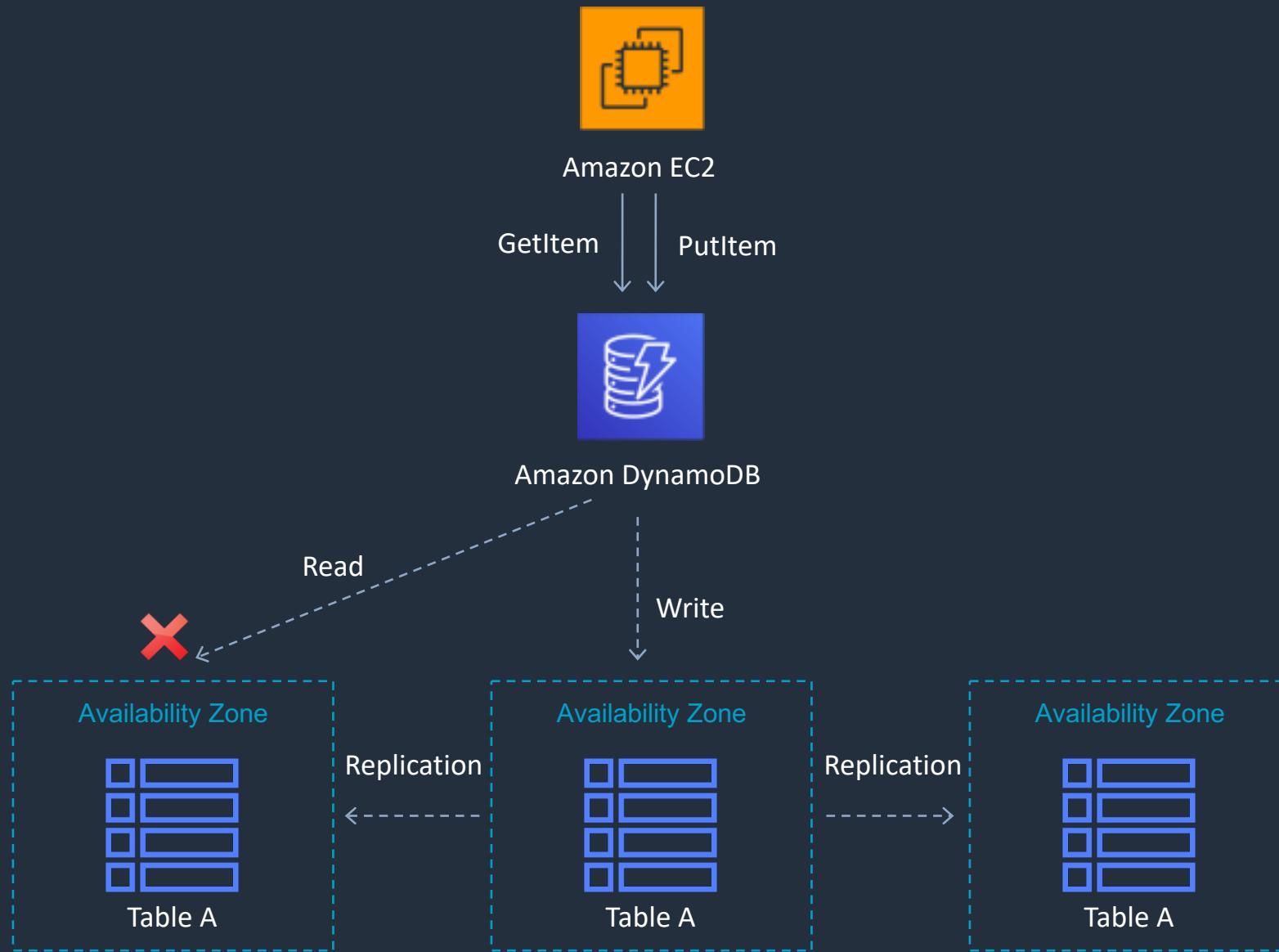
DynamoDB supports eventually consistent and strongly consistent reads.

- Eventually consistent reads:
  - When you read data from a DynamoDB table, the response might not reflect the results of a recently completed write operation.
  - The response might include some stale data.
  - If you repeat your read request after a short time, the response should return the latest data.

## Amazon DynamoDB – Consistency Models and Transactions

- Strongly consistent read:
  - When you request a strongly consistent read, DynamoDB returns a response with the most up-to-date data, reflecting the updates from all prior write operations that were successful.
  - A strongly consistent read might not be available if there is a network delay or outage. In this case, DynamoDB may return a server error (HTTP 500).
  - Strongly consistent reads may have higher latency than eventually consistent reads.
  - Strongly consistent reads are not supported on global secondary indexes.
  - Strongly consistent reads use more throughput capacity than eventually consistent reads.

# Amazon DynamoDB – Consistency Models and Transactions



## Amazon DynamoDB – Consistency Models and Transactions

- With a strongly consistent read, data will always be returned when reading after a successful write.
- DynamoDB uses eventually consistent reads by default.
- You can configure strongly consistent reads with the GetItem, Query and Scan APIs by setting the --consistent-read (or ConsistentRead) parameter to “true”.

## Amazon DynamoDB – Consistency Models and Transactions

- Amazon DynamoDB transactions simplify the developer experience of making coordinated, all-or-nothing changes to multiple items both within and across tables.
- Transactions provide atomicity, consistency, isolation, and durability (ACID) in DynamoDB.
- Enables reading and writing of multiple items across multiple tables as an all or nothing operation.
- Checks for a pre-requisite condition before writing to a table.

## Amazon DynamoDB – Consistency Models and Transactions

- With the transaction write API, you can group multiple **Put**, **Update**, **Delete**, and **ConditionCheck** actions.
- You can then submit the actions as a single **TransactWriteItems** operation that either succeeds or fails as a unit.
- The same is true for multiple **Get** actions, which you can group and submit as a single **TransactGetItems** operation.

## Amazon DynamoDB – Consistency Models and Transactions

- There is no additional cost to enable transactions for DynamoDB tables.
- You pay only for the reads or writes that are part of your transaction.
- DynamoDB performs two underlying reads or writes of every item in the transaction:  
one to prepare the transaction and one to commit the transaction.
- These two underlying read/write operations are visible in your Amazon CloudWatch  
metrics.

# Amazon DynamoDB – Consistency Models and Transactions



# Amazon DynamoDB – Consistency Models and Transactions



## Amazon DynamoDB – Provisioned Capacity

- With provisioned capacity mode (default), you specify the number of data reads and writes per second that you require for your application.
- You can use auto scaling to automatically adjust your table's capacity based on the specified utilization rate to ensure application performance while reducing costs.
- When you create your table you specify your requirements using Read Capacity Units (RCUs) and Write Capacity Units (WCUs).
- WCUs and RCUs are spread between partitions evenly.

## Amazon DynamoDB – Provisioned Capacity

- Can also use Auto Scaling with provisioned capacity.
- DynamoDB auto scaling uses the AWS Application Auto Scaling service to dynamically adjust provisioned throughput capacity on your behalf, in response to traffic patterns.
- This enables a table or a global secondary index to increase its provisioned read and write capacity to handle sudden increases in traffic, without throttling.

# Amazon DynamoDB – Provisioned Capacity

Provisioned (free-tier eligible)  
 On-demand

### Provisioned capacity

	Read capacity units	Write capacity units
Table	5	5

Estimated cost \$3.31 / month ([Capacity calculator](#))

### Auto Scaling

<input checked="" type="checkbox"/> Read capacity	<input checked="" type="checkbox"/> Write capacity
<input type="checkbox"/> Same settings as read	
Target utilization 70 %	70 %
Minimum provisioned capacity 5 units	5 units
Maximum provisioned capacity 40000 units	40000 units
<input checked="" type="checkbox"/> Apply same settings to global secondary indexes	<input checked="" type="checkbox"/> Apply same settings to global secondary indexes

## Amazon DynamoDB – Provisioned Capacity

Read capacity unit (RCU):

- Each API call to read data from your table is a read request.
- Read requests can be strongly consistent, eventually consistent, or transactional.
- For items up to 4 KB in size, one RCU equals:
  - One strongly consistent read request per second.
  - Two eventually consistent read requests per second.
  - 0.5 transactional read requests per second.
- Items larger than 4 KB require additional RCUs.



## Amazon DynamoDB – Provisioned Capacity

Read capacity unit (RCU):

- Example 1: 10 strongly consistent reads per second of 4 KB each

$$10 * 4 \text{ KB} / 4 \text{ KB} = 10 \text{ RCU}$$

- Example 2: 10 strongly consistent reads per second of 11 KB each

$$10 * 11 \text{ KB} / 4 \text{ KB} = 30 \text{ RCU}$$

- Example 3: 20 eventually consistent reads per second of 12 KB each

$$(20 / 2) * (12 / 4) = 30 \text{ RCU}$$

- Example 4: 36 eventually consistent reads per second of 16 KB each

$$(36 / 2) * (16 / 4) = 72 \text{ RCU}$$

## Amazon DynamoDB – Provisioned Capacity

Write capacity unit (WCU):

- Each API call to write data to your table is a write request.
- For items up to 1 KB in size, one WCU can perform:
  - One standard write request per second.
  - 0.5 transactional writes requests (one transactional write requires two WCUs)
- Items larger than 1 KB require additional WCUs.

## Amazon DynamoDB – Provisioned Capacity

Write capacity unit (WCU):

- Example 1: 10 standard writes per second of 4 KB each

$$10 * 4 = 40 \text{ WCU}$$

- Example 2: 12 standard writes per second of 9.5 KB each

$$12 * 10 = 120 \text{ WCU}$$

- Example 1: 10 transactional writes per second of 4 KB each

$$10 * 2 * 4 = 80 \text{ WCU}$$

- Example 2: 12 transactional writes per second of 9.5 KB each

$$12 * 2 * 10 = 240 \text{ RCU}$$

## Amazon DynamoDB – On-Demand Capacity

- With on-demand, you don't need to specify your requirements.
- DynamoDB instantly scales up and down based on the activity of your application.
- Great for unpredictable / spiky workloads or new workloads that aren't well understood.
- You pay for what you use (pay per request).
- You can switch between the provisioned capacity and on-demand pricing models once per day.

## Amazon DynamoDB – Performance and Throttling

- Throttling occurs when the configured RCU or WCU are exceeded.
- May receive the **ProvisionedThroughputExceededException** error.
- This error indicates that your request rate is too high for the read / write capacity provisioned for the table.
- The AWS SDKs for DynamoDB automatically retry requests that receive this exception.
- Your request is eventually successful, unless your retry queue is too large to finish.

## Amazon DynamoDB – Performance and Throttling

- Possible causes of performance issues:
  - Hot keys – one partition key is being read too often.
  - Hot partitions - when data access is imbalanced, a "hot" partition can receive a higher volume of read and write traffic compared to other partitions.
  - Large items – large items consume more RCUs and WCUs.

## Amazon DynamoDB – Performance and Throttling

- Resolution:
  - Reduce the frequency of requests and use exponential backoff.
  - Try to design your application for uniform activity across all logical partition keys in the table and its secondary indexes.
  - Use burst capacity effectively - DynamoDB currently retains up to 5 minutes (300 seconds) of unused read and write capacity which can be consumed quickly.

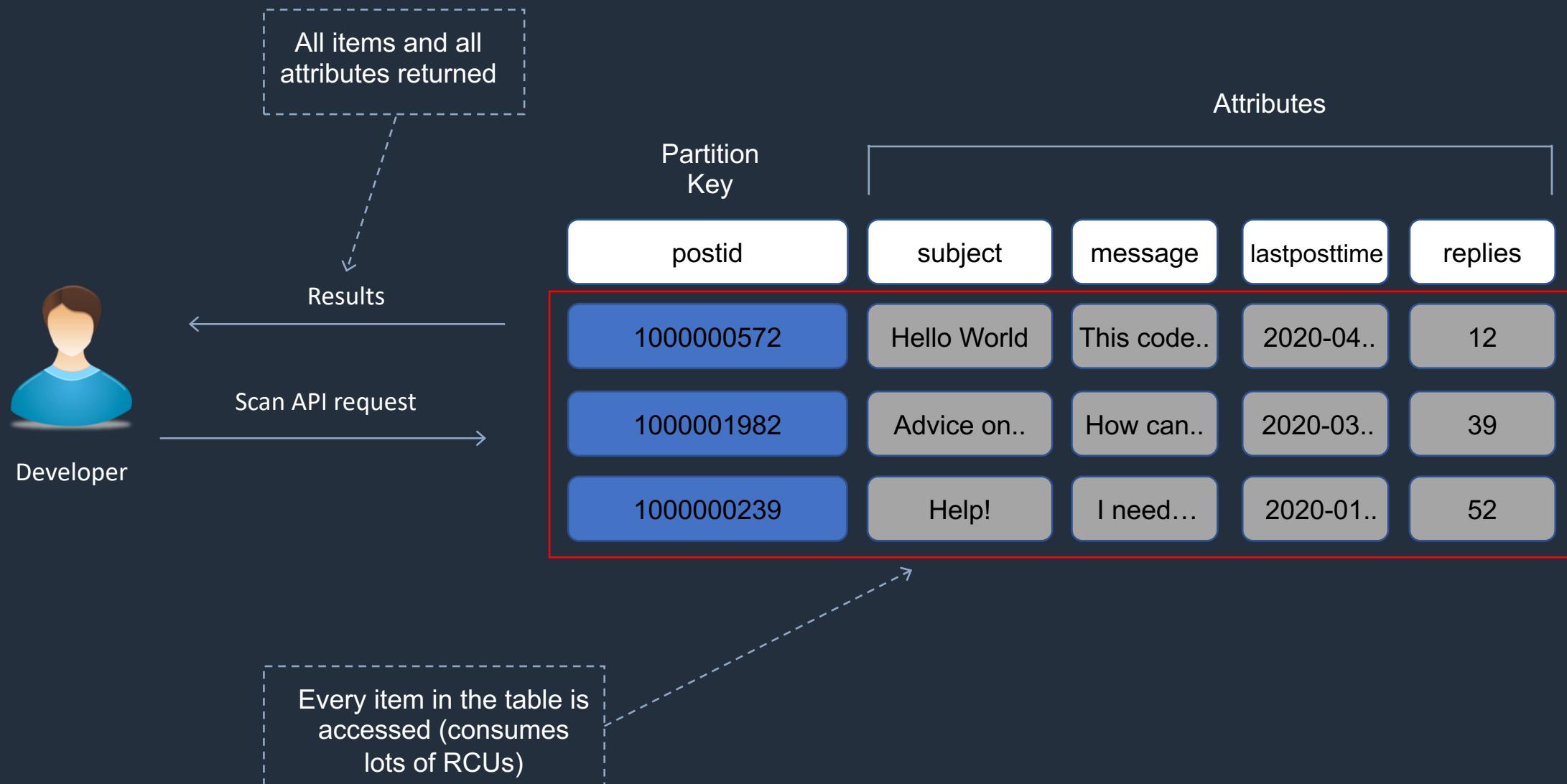
## Amazon DynamoDB – Scan API

- The Scan operation returns one or more items and item attributes by **accessing** every item in a table or a secondary index.
- To have DynamoDB return fewer items, you can provide a FilterExpression operation.
- A single Scan operation reads up to the maximum number of items set (if using the **Limit** parameter) or a maximum of 1 MB.
- Scan API calls can use a lot of RCUs as they access every item in the table.

## Amazon DynamoDB – Scan API

- Scan operations proceed sequentially.
- For faster performance on a large table or secondary index, applications can request a parallel Scan operation by providing the Segment and TotalSegments parameters.
- Scan uses eventually consistent reads when accessing the data in a table.
- If you need a consistent copy of the data, as of the time that the Scan begins, you can set the **ConsistentRead** parameter to true.

# Amazon DynamoDB – Scan API



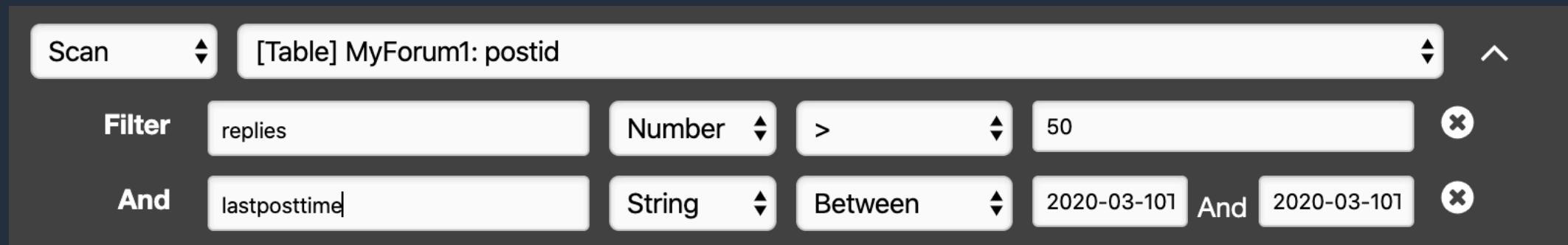
## Amazon DynamoDB – Scan API with Projection Expression

- In this example the scan will return all posts in the forum that were posted within a date range and that have more than 50 replies?

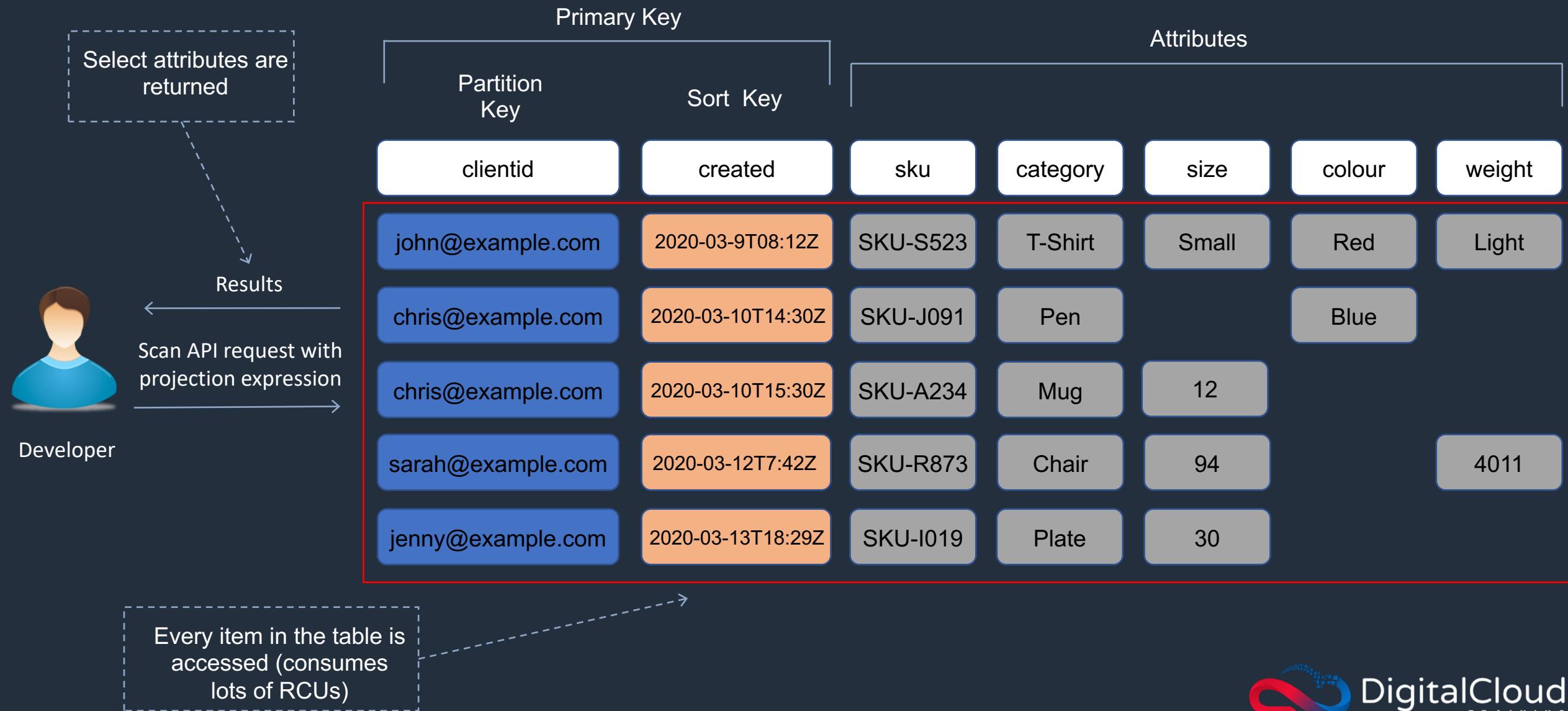
Scan [Table] MyForum1: postid

Filter replies Number > 50

And lastposttime String Between 2020-03-101 And 2020-03-101



# Amazon DynamoDB – Scan API with Projection Expression



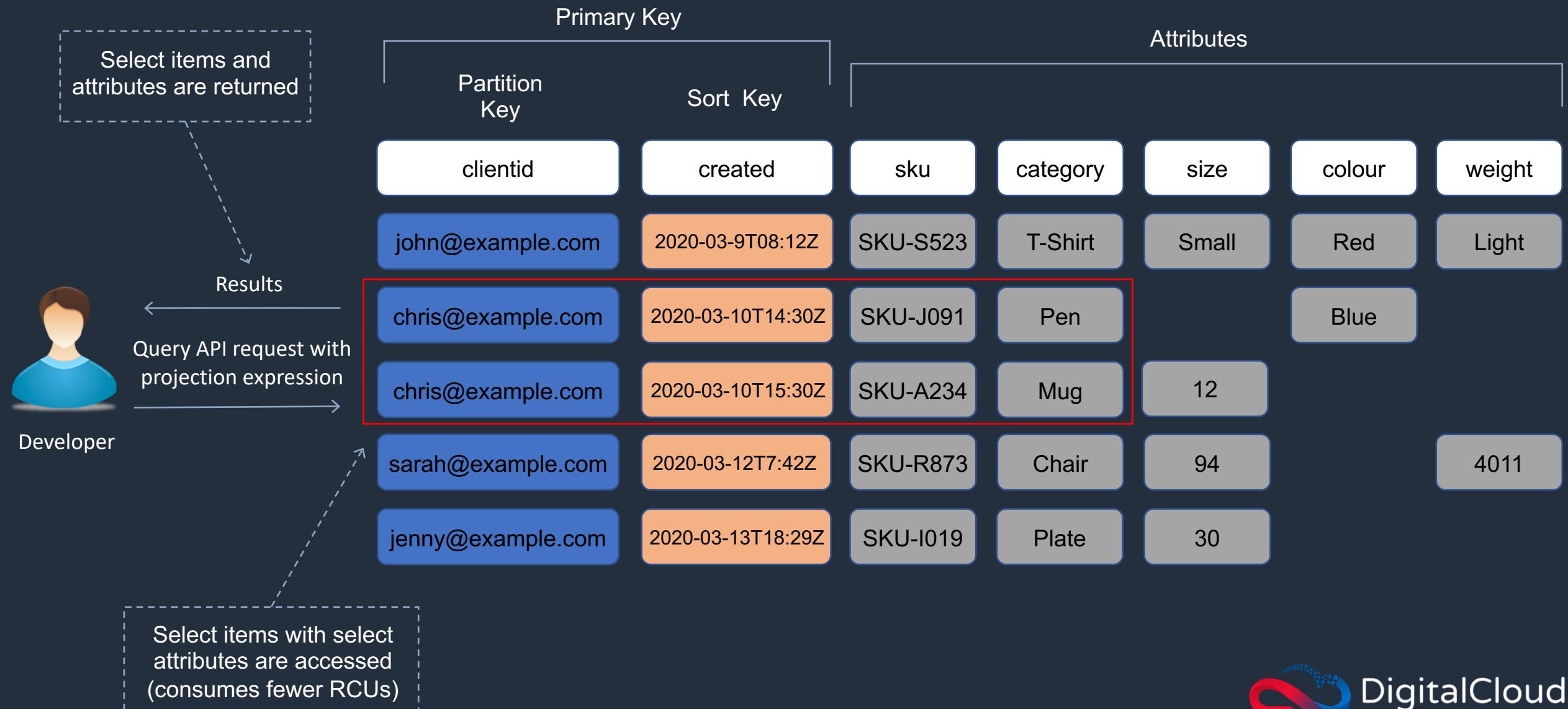
## Amazon DynamoDB – Query API

- A query operation finds items in your table based on the primary key attribute and a distinct value to search for.
- For example you might search for a user ID value and all attributes related to that item would be returned.
- Can use an optional sort key name and value to refine the results.
- For example if you sort key is a timestamp, you can refine the query to only select items with a timestamp of the last 7 days.
- By default, a query returns all the attributes for the items, but you can use the ProjectionExpression parameter if you want to query to only return the attributes you want to see.

## Amazon DynamoDB – Query API

- By default, queries are eventually consistent.
- To use strongly consistent you need to explicitly set this in the query.

# Amazon DynamoDB – Query API with Projection Expression



## Amazon DynamoDB – Query API

- In this example the query returns only items with the client id of “chris@example.com” that were created within a certain date range and that are in the category “pen”:

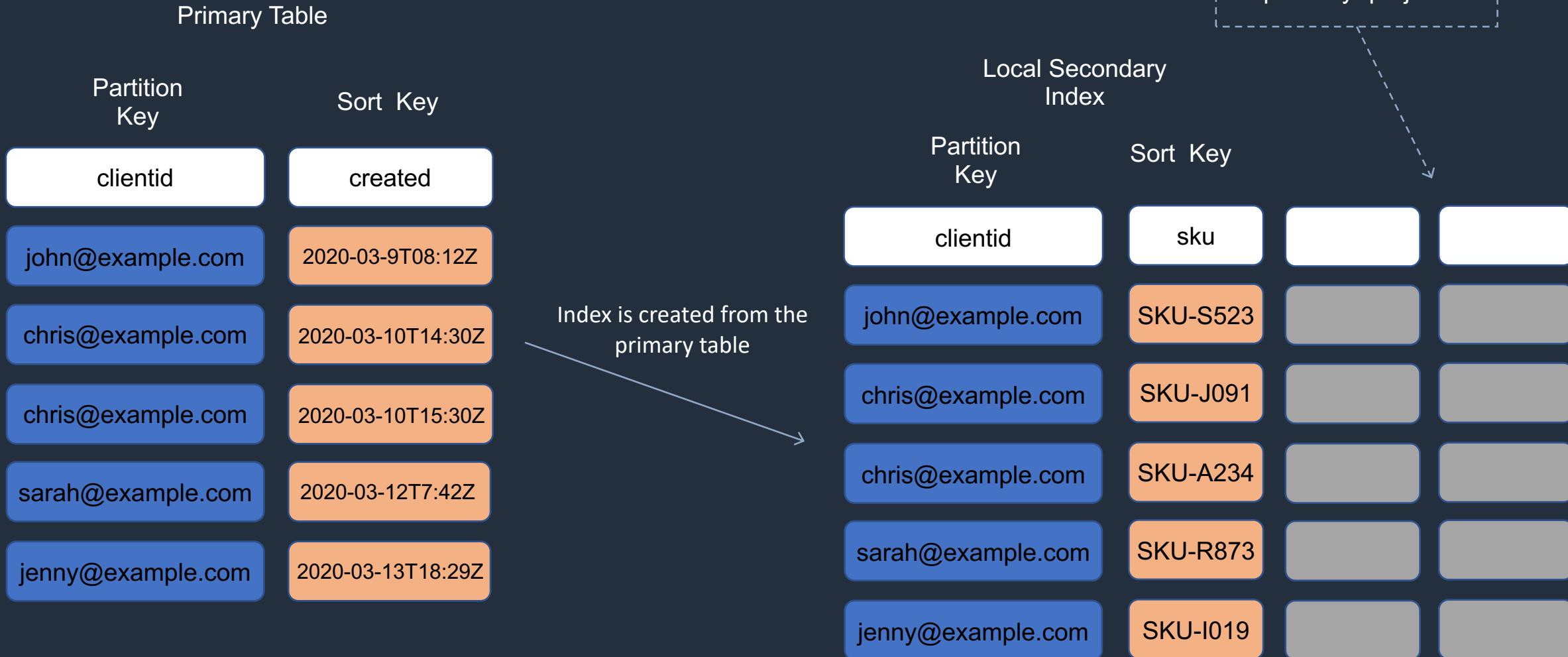
Query [Table] ProductOrders: clientid, created

<b>Partition key</b>	clientid	String	=	chris@example.com
<b>Sort key</b>	created	String	Between	2020-03-101 And 2020-03-101
<b>Filter</b>	category	String	=	pen

## Amazon DynamoDB – Local Secondary Index (LSI)

- Provides an alternative sort key to use for scans and queries.
- Can create up to 5 LSIs per table.
- Must be created at table creation time.
- You cannot add, remove, or modify it later.
- It has the same partition key as your original table (different sort key).
- Gives you a different view of your data, organized by alternative sort key.
- Any queries based on this sort key are much faster using the index than the main table.

# Amazon DynamoDB – Local Secondary Index (LSI)



## Amazon DynamoDB – Local Secondary Index (LSI)

- In this example querying the main table, we must use the partition key “clientid” and the sort key “created”:

Query [Table] mystore: clientid, created

<b>Partition key</b>	clientid	String	=	chris@example.com
<b>Sort key</b>	created	String	>	2020-03-10T14:30Z
<a href="#">+ Add filter</a>				

- In this example with an LSI, we can query the index for any orders made by [chris@example.com](mailto:chris@example.com) for the product "SKU-1922":

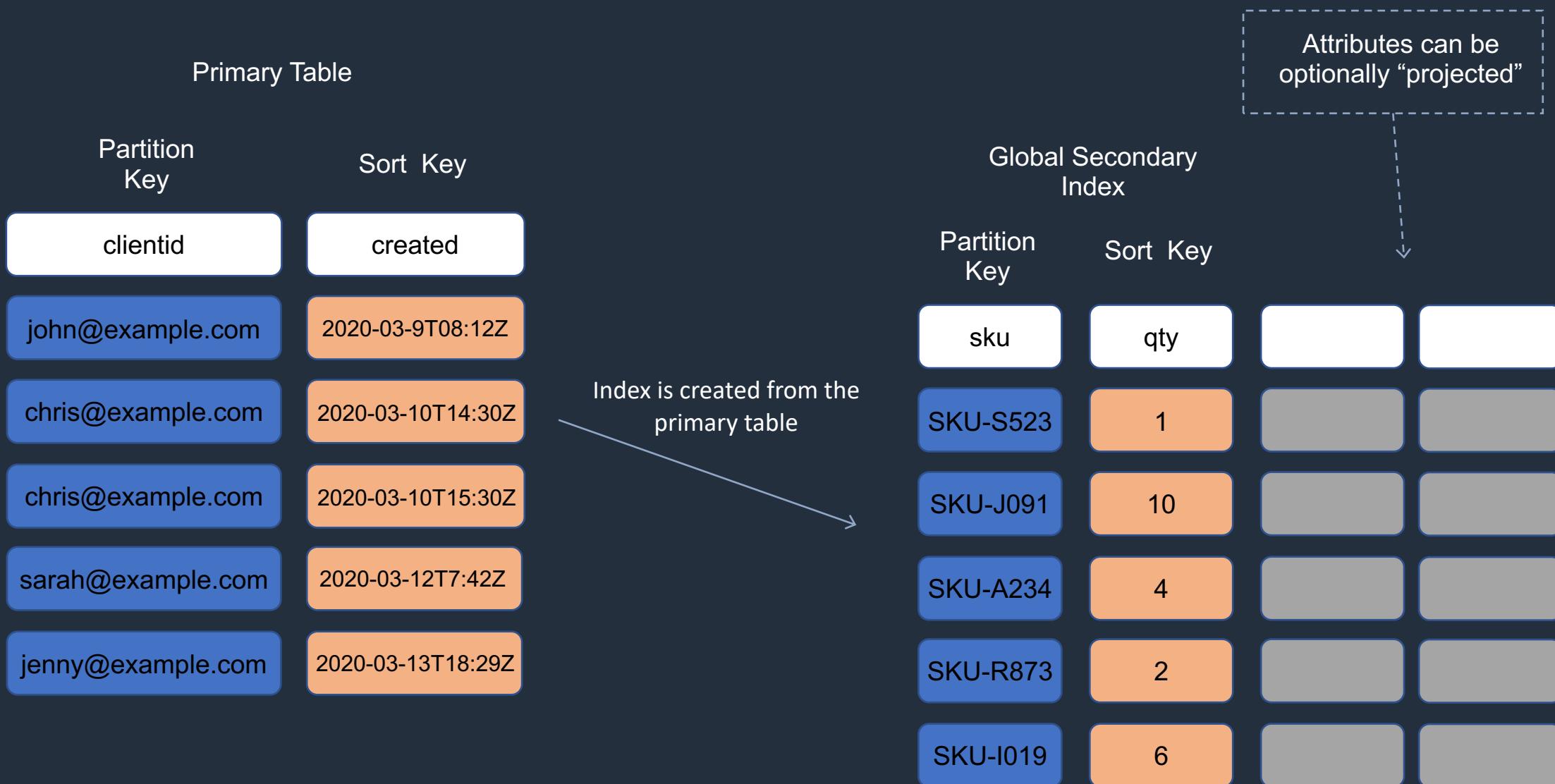
Query [Index] clientid-sku-index: clientid, sku

<b>Partition key</b>	clientid	String	=	chris@example.com
<b>Sort key</b>	sku	String	=	SKU-P122
<a href="#">+ Add filter</a>				

## Amazon DynamoDB – Global Secondary Index (GSI)

- Used to speed up queries on non-key attributes.
- You can create when you create your table or at any time later on.
- Can specify a different partition key as well as a different sort key.
- Gives a completely different view of the data.
- Speeds up any queries relating to this alternative partition and sort key.

# Amazon DynamoDB – Global Secondary Index (GSI)



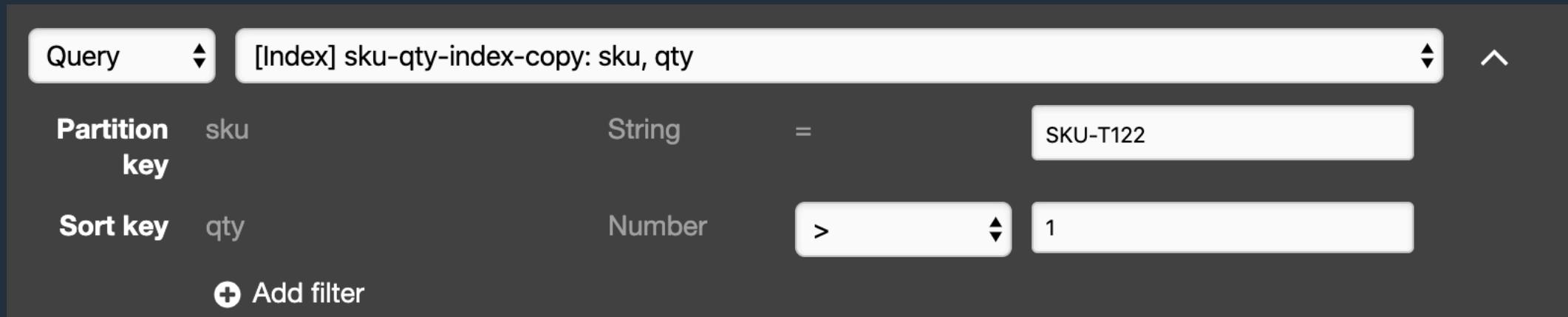
## Amazon DynamoDB – Global Secondary Index (GSI)

- In this example, we can query the index for orders of "SKU-T122" where the quantity is greater than 1:

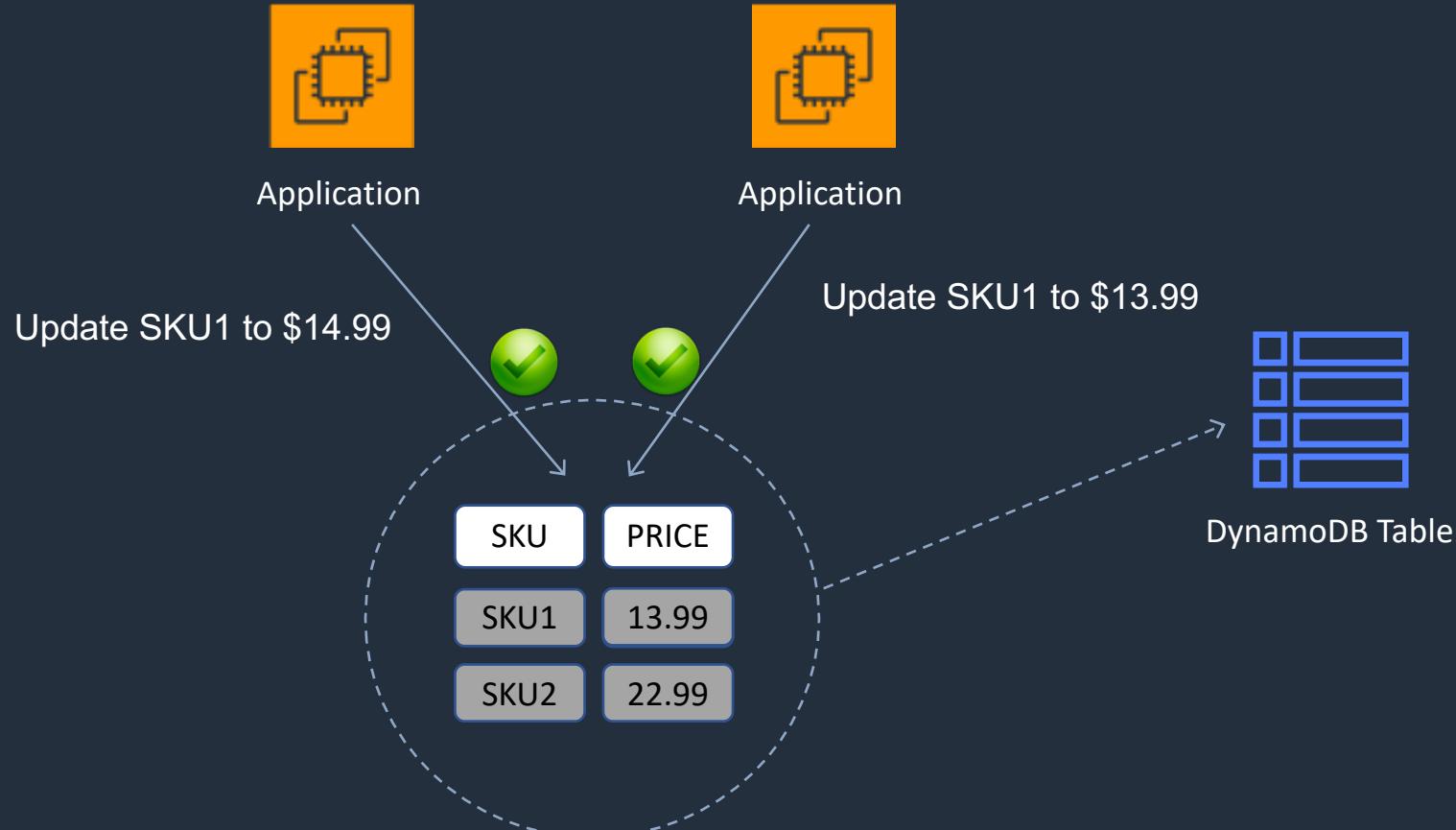
Query [Index] sku-qty-index-copy: sku, qty

<b>Partition key</b>	sku	String	=	SKU-T122
<b>Sort key</b>	qty	Number	>	1

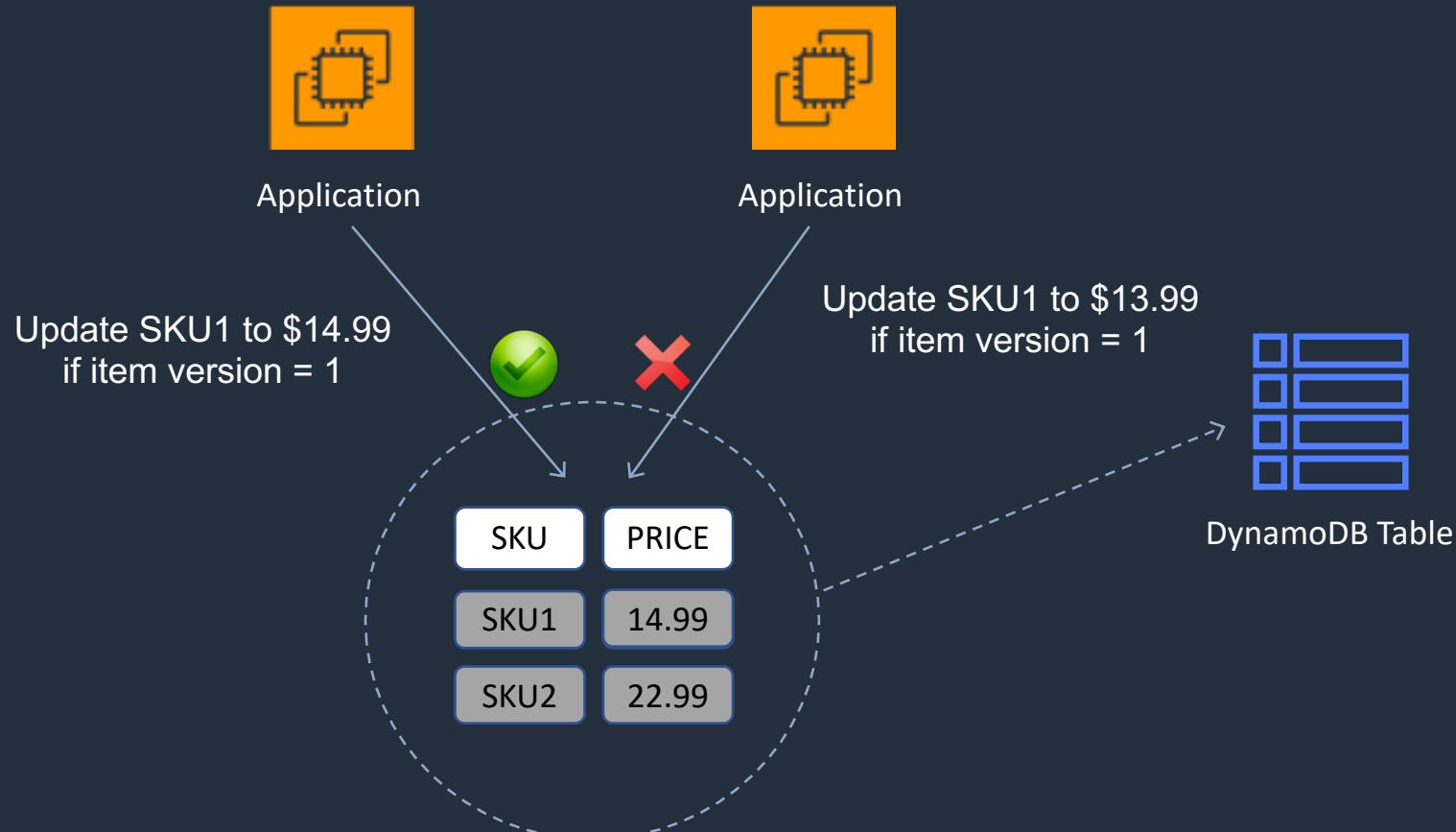
+ Add filter



# Amazon DynamoDB – Optimistic Locking



# Amazon DynamoDB – Optimistic Locking



## Amazon DynamoDB – Optimistic Locking

- Optimistic locking is a strategy to ensure that the client-side item that you are updating (or deleting) is the same as the item in Amazon DynamoDB.
- Protects database writes from being overwritten by the writes of others, and vice versa.

## Amazon DynamoDB – Conditional Updates

- To manipulate data in an Amazon DynamoDB table, you use the PutItem, UpdateItem, and DeleteItem operations.
- You can optionally specify a condition expression to determine which items should be modified.
- If the condition expression evaluates to true, the operation succeeds; otherwise, the operation fails.

## Amazon DynamoDB – Conditional Updates

- This example CLI command allows the write to proceed only if the item in question does not already have the same key:

```
aws dynamodb put-item --table-name ProductCatalog --item file://item.json \
--condition-expression "attribute_not_exists(Id)"
```

- This example CLI command uses attribute\_not\_exists to delete a product only if it does not have a Price attribute:

```
aws dynamodb delete-item --table-name ProductCatalog --key '{"Id": {"N": "456"}}' \
--condition-expression "attribute_not_exists(Price)"
```

## Amazon DynamoDB – Conditional Updates

- This example CLI command only deletes an item if the ProductCategory is either "Sporting Goods" or "Gardening Supplies" and the price is between 500 and 600.

```
aws dynamodb delete-item \  
  --table-name ProductCatalog \  
  --key '{"Id":{"N":"456"}}' \  
  --condition-expression "(ProductCategory IN (:cat1, :cat2)) and (Price between :lo and :hi)" \  
  --expression-attribute-values file://values.json
```

# Amazon DynamoDB – Time To Live (TTL)

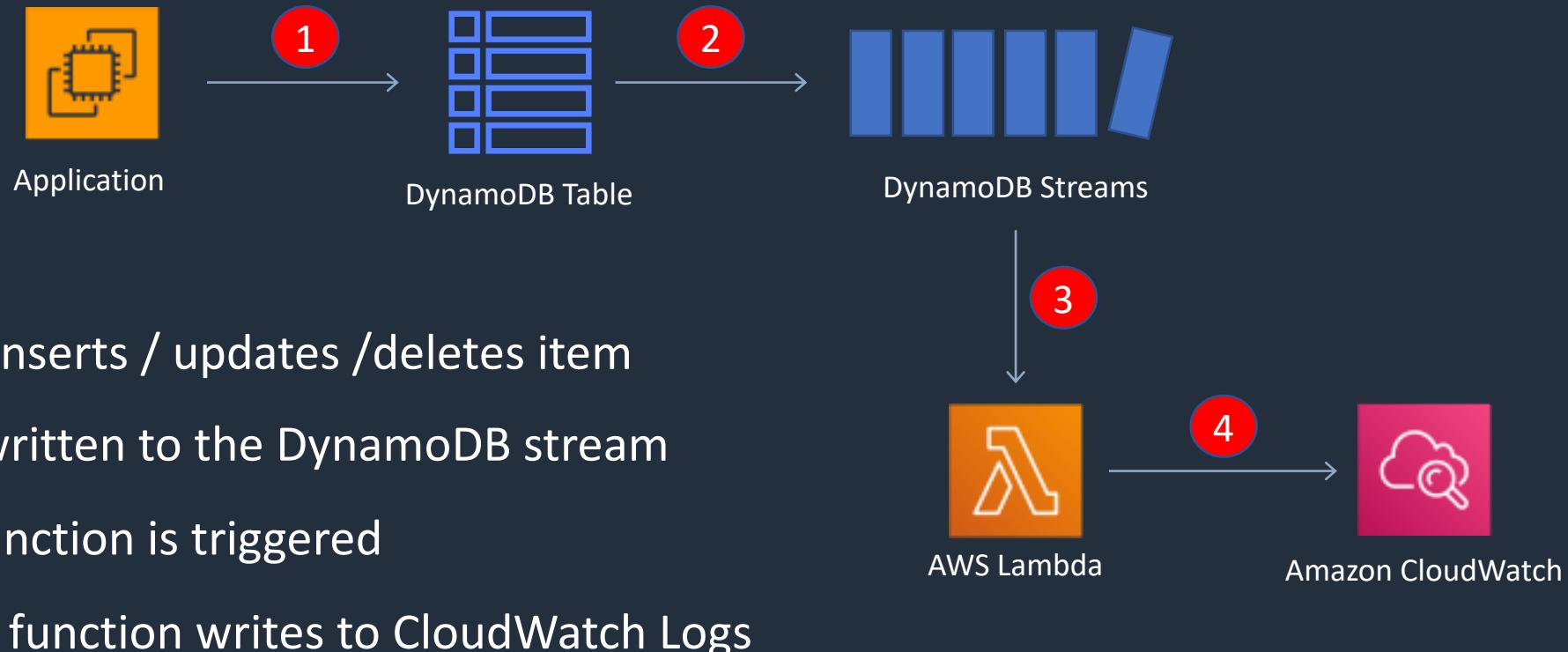
sessionid	data	date	expiry
23040210248	https://ama..	2020-04..	158415112
84324350122	https://ama..	2020-04..	158416214
923424325040	https://ama..	2020-04...	158418985

The item will be deleted when the expiry date (in Epoch format) expires

## Amazon DynamoDB – Time To Live (TTL)

- Time to Live (TTL) for Amazon DynamoDB lets you define when items in a table expire so that they can be automatically deleted from the database.
- With TTL enabled on a table, you can set a timestamp for deletion on a per-item basis, allowing you to limit storage usage to only those records that are relevant.
- TTL is useful if you have continuously accumulating data that loses relevance after a specific time period (for example, session data, event logs, usage patterns, and other temporary data).
- No extra cost and does not use WCU / RCU.
- Helps reduce storage and manage the table size over time.
- Enabled per row (you define a TTL column and add the expiry date / time there).
- DynamoDB typically deletes expired items within 48 hours of expiration.
- Deleted items are also deleted from the LSI / GSI.

# Amazon DynamoDB Streams



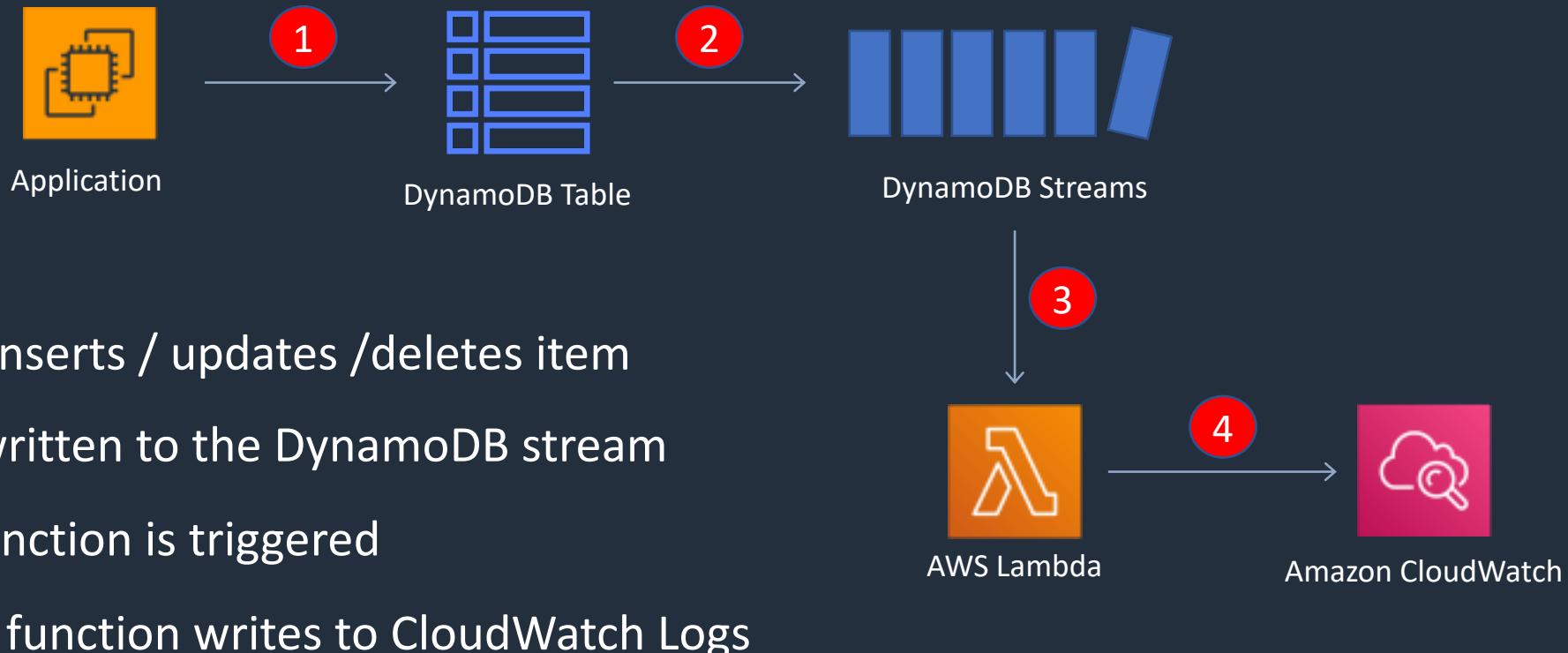
## Amazon DynamoDB Streams

- DynamoDB Streams captures a time-ordered sequence of item-level modifications in any DynamoDB table and stores this information in a log for up to 24 hours.
- Applications can access this log and view the data items as they appeared before and after they were modified, in near-real time.
- You can also use the CreateTable or UpdateTable API operations to enable or modify a stream.

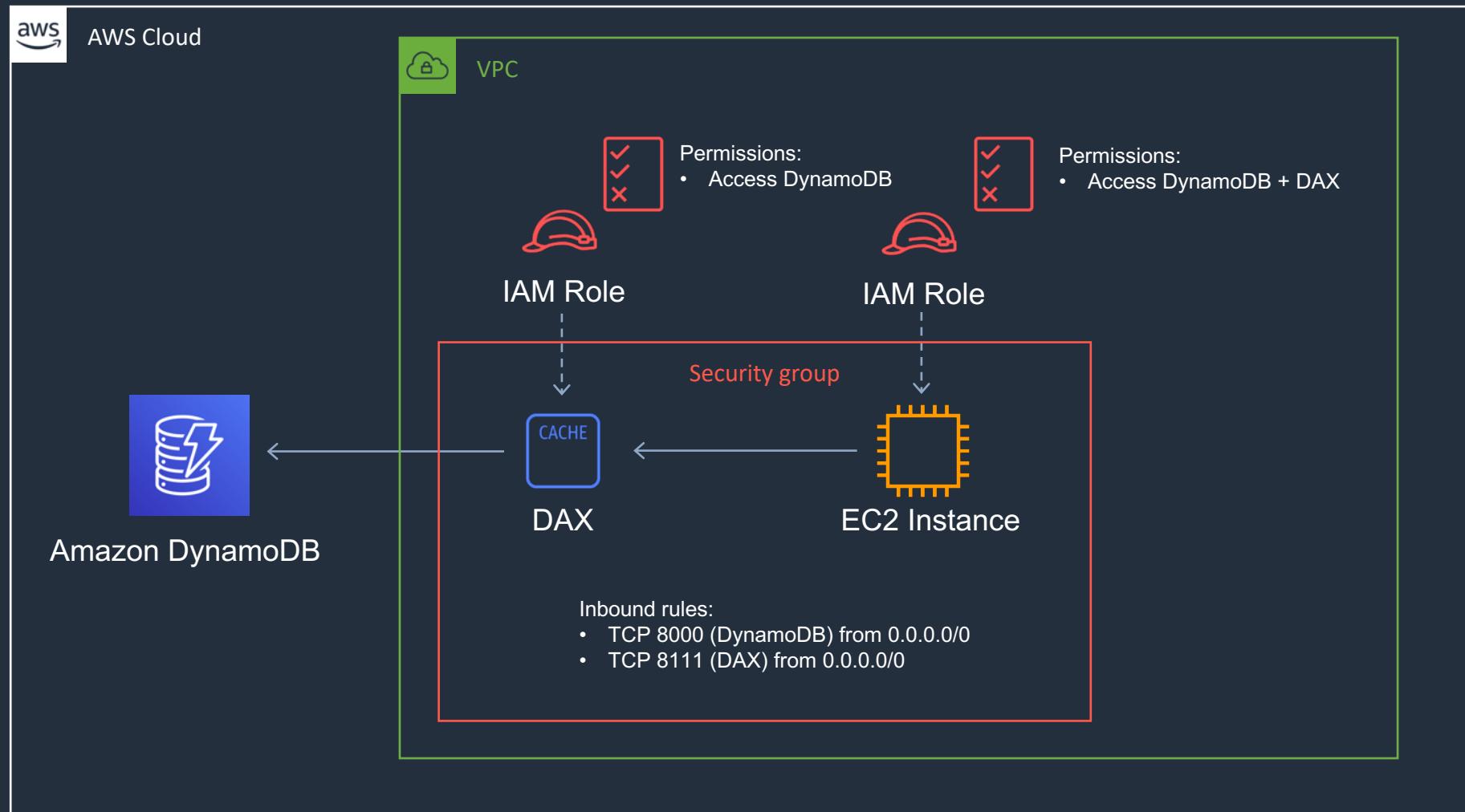
## Amazon DynamoDB Streams

- The **StreamSpecification** parameter determines how the stream is configured:
- **StreamEnabled** — Specifies whether a stream is enabled (true) or disabled (false) for the table.
- **StreamViewType** — Specifies the information that will be written to the stream whenever data in the table is modified:
  - KEYS\_ONLY — Only the key attributes of the modified item.
  - NEW\_IMAGE — The entire item, as it appears after it was modified.
  - OLD\_IMAGE — The entire item, as it appeared before it was modified.
  - NEW\_AND\_OLD\_IMAGES — Both the new and the old images of the item.

# Amazon DynamoDB Streams



# DynamoDB Accelerator (DAX)



## Amazon DynamoDB Accelerator (DAX)

- Amazon DynamoDB Accelerator (DAX) is a fully managed, highly available, in-memory cache for DynamoDB that delivers up to a 10x performance improvement.
- Improves performance from milliseconds to microseconds, even at millions of requests per second.
- DAX is a managed service that provides in-memory acceleration for DynamoDB tables.
- Provides managed cache invalidation, data population, and cluster management.
- DAX is used to improve READ performance (not writes).
- You do not need to modify application logic, since DAX is compatible with existing DynamoDB API calls.

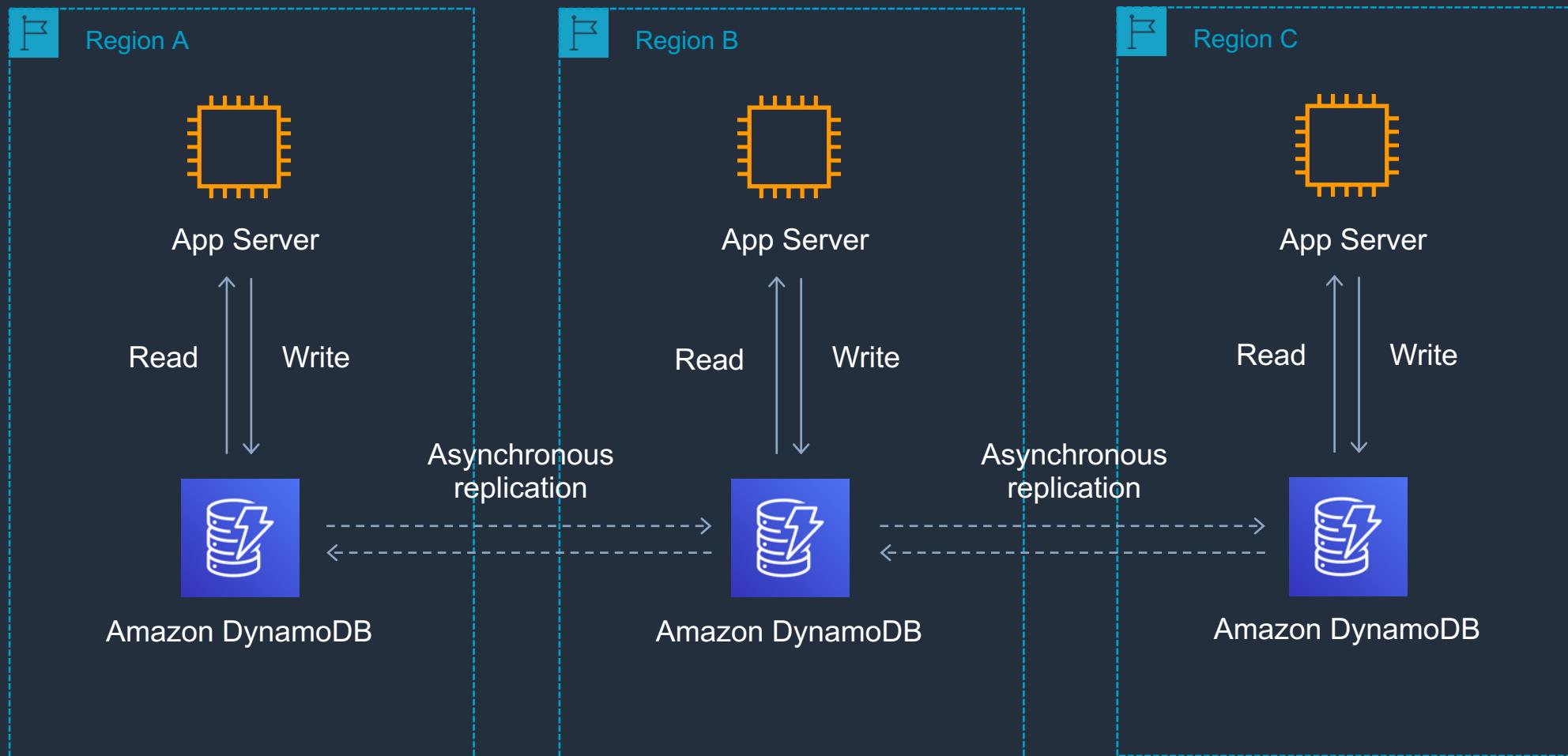
## Amazon DynamoDB Accelerator (DAX)

- You can enable DAX with just a few clicks in the AWS Management Console or using the AWS SDK.
- Just as with DynamoDB, you only pay for the capacity you provision.
- Provisioned through clusters and charged by the node (runs on EC2 instances).
- Pricing is per node-hour consumed and is dependent on the instance type you select.

## Amazon DynamoDB – DAX vs ElastiCache

- DAX is optimized for DynamoDB.
- DAX does not support lazy loading (uses write-through caching).
- With ElastiCache you have more management overhead (e.g. invalidation).
- With ElastiCache you need to modify application code to point to cache.
- ElastiCache supports more datastores.

# Amazon DynamoDB Global Tables



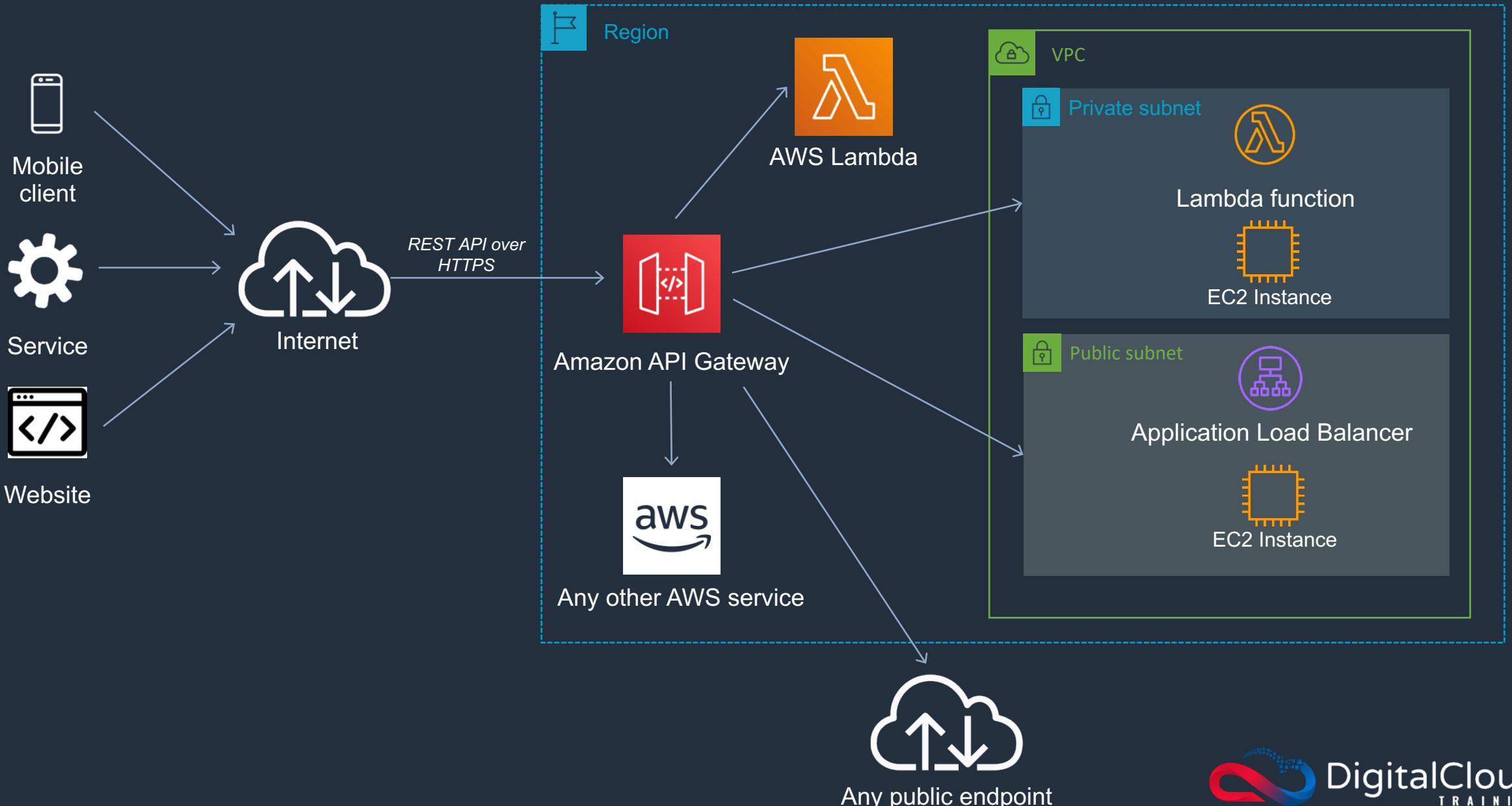
## Amazon DynamoDB Global Tables

- Amazon DynamoDB global tables provide a fully managed solution for deploying a multi-region, multi-master database.
- When you create a global table, you specify the AWS regions where you want the table to be available.
- DynamoDB performs all of the necessary tasks to create identical tables in these regions, and propagate ongoing data changes to all of them.
- DynamoDB global tables are ideal for massively scaled applications, with globally dispersed users.
- Global tables provide automatic multi-master replication to AWS regions world-wide, so you can deliver low-latency data access to your users no matter where they are located.

# SECTION 16

## Amazon API Gateway

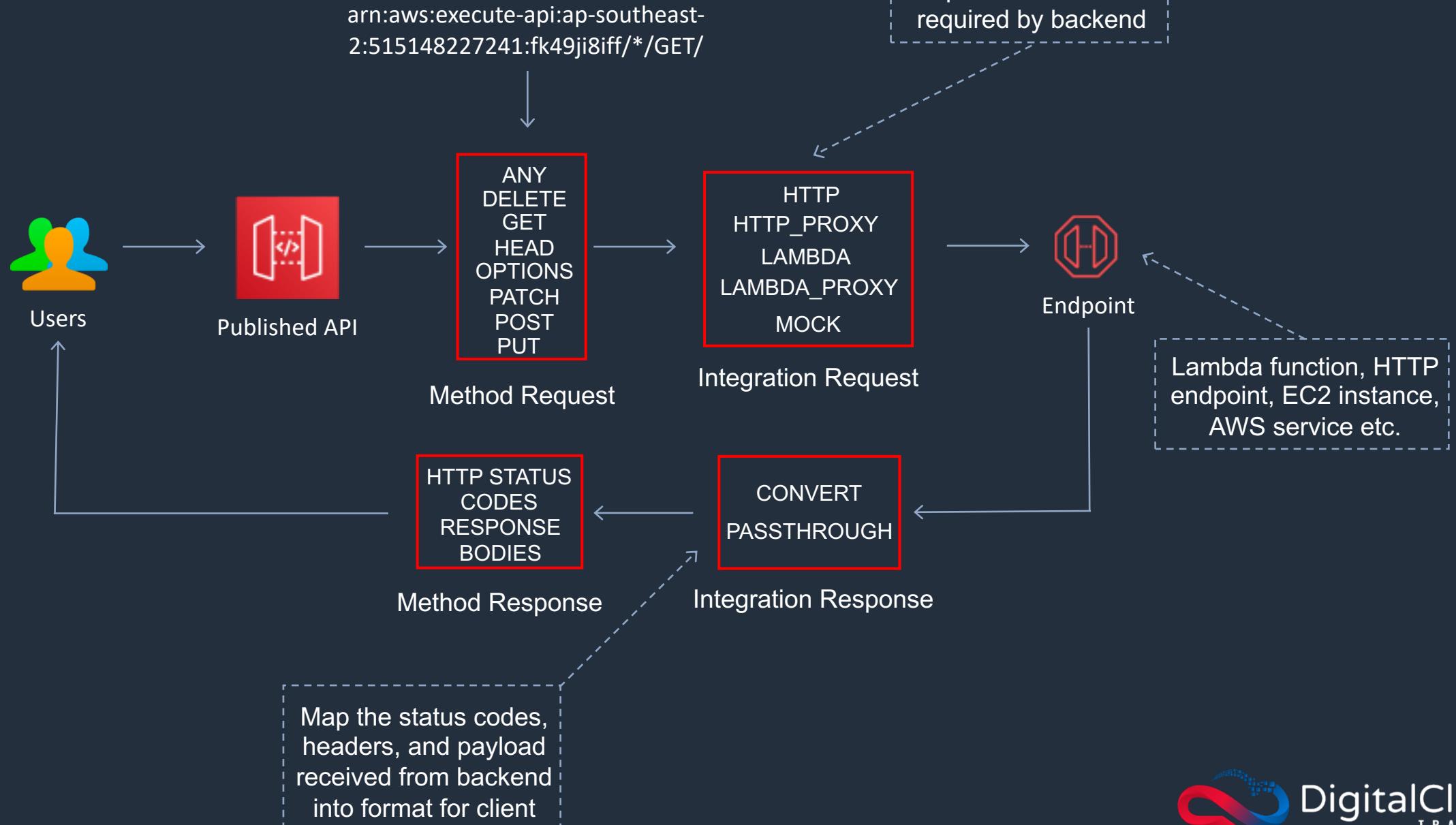
# API Gateway Overview



## Amazon API Gateway

- API Gateway is an AWS service that supports the following:
  - Creating, deploying, and managing a REST application programming interface (API) to expose backend HTTP endpoints, AWS Lambda functions, or other AWS services.
  - Creating, deploying, and managing a WebSocket API to expose AWS Lambda functions or other AWS services.
  - Invoking exposed API methods through the frontend HTTP and WebSocket endpoints.

# Amazon API Gateway – Structure of an API



## Amazon API Gateway

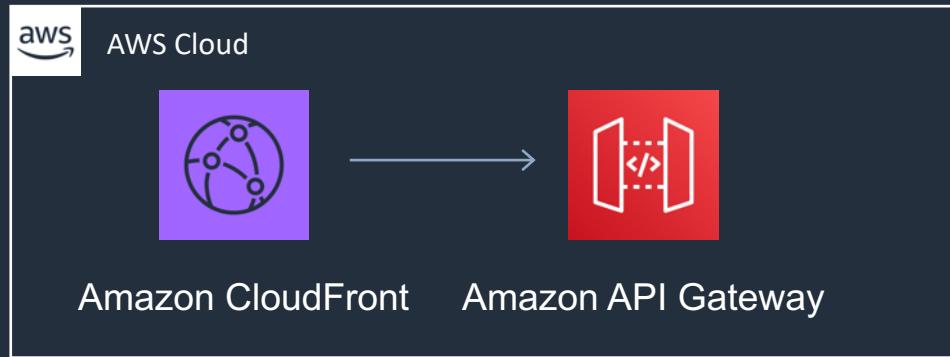
- API Gateway REST API:
  - A collection of HTTP resources and methods that are integrated with backend HTTP endpoints, Lambda functions, or other AWS services.
  - This collection can be deployed in one or more stages.
  - Typically, API resources are organized in a resource tree according to the application logic.
  - Each API resource can expose one or more API methods that have unique HTTP verbs supported by API Gateway.

## Amazon API Gateway

- API Gateway WebSocket API:
  - A collection of WebSocket routes and route keys that are integrated with backend HTTP endpoints, Lambda functions, or other AWS services.
  - The collection can be deployed in one or more stages.
  - API methods are invoked through frontend WebSocket connections that you can associate with a registered custom domain name.

# API Gateway Endpoint Types

Edge-optimized endpoint:



Key benefits:

- Reduced latency for requests from around the world

Regional endpoint:



Key benefits:

- Reduced latency for requests that originate in the same region
- Can also configure your own CDN and protect with WAF

Private endpoint:



Key benefits:

- Securely expose your REST APIs only to other services within your VPC or connect via Direct Connect

# Amazon API Gateway

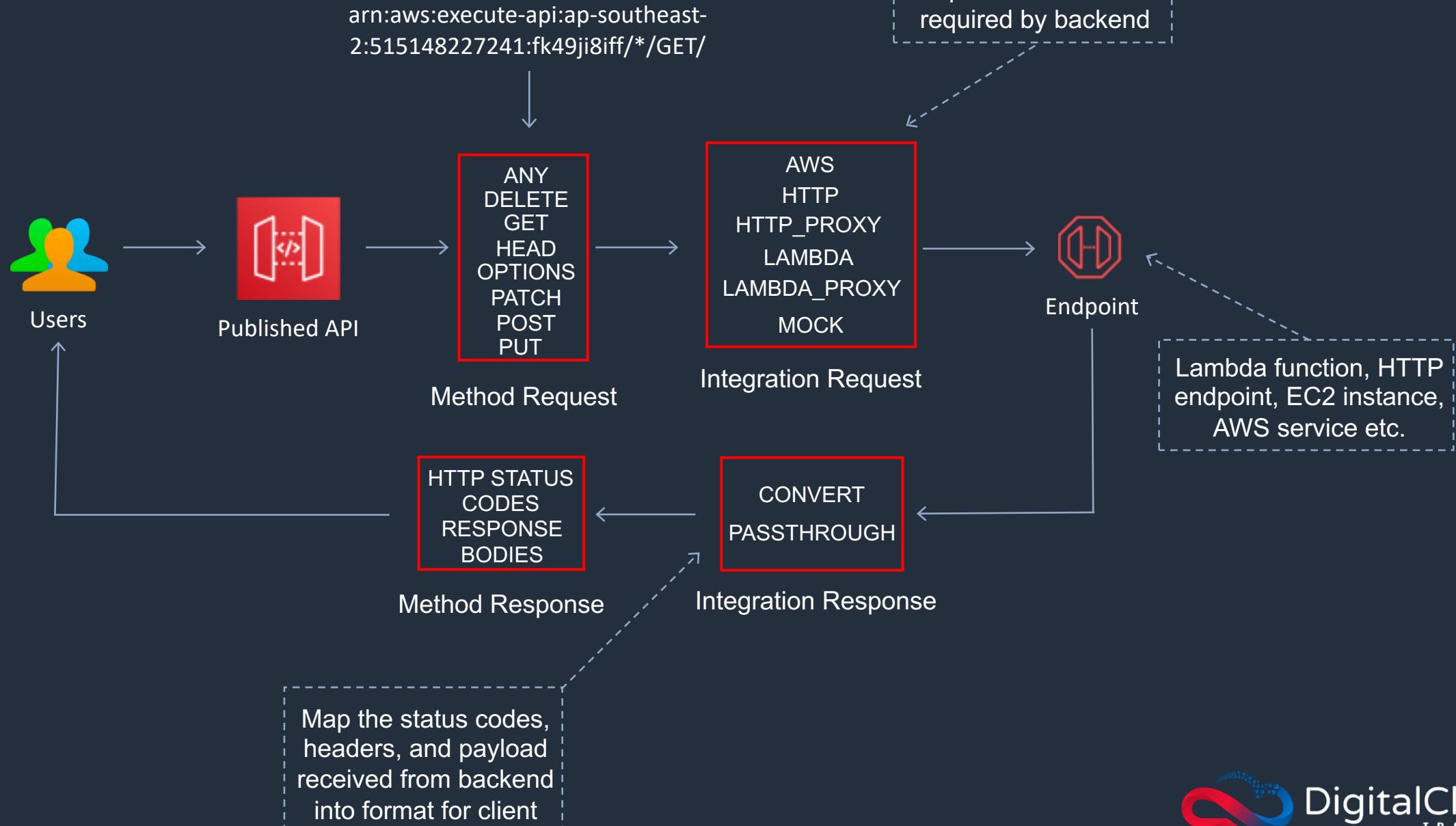
API Gateway Feature	Benefit
Support for RESTful APIs and WebSocket APIs	With API Gateway, you can create RESTful APIs using either HTTP APIs or REST APIs
Private integrations with AWS ELB & AWS Cloud Map	With API Gateway, you can route requests to private resources in your VPC. Using HTTP APIs, you can build APIs for services behind private ALBs, private NLBs, and IP-based services registered in AWS Cloud Map, such as ECS tasks.
Metering	Define plans that meter and restrict third-party developer access to APIs
Security	API Gateway provides multiple tools to authorize access to APIs and control service operation access
Resiliency	Manage traffic with throttling so that backend operations can withstand traffic spikes
Operations Monitoring	API Gateway provides a metrics dashboard to monitor calls to services
Lifecycle Management	Operate multiple API versions and multiple stages for each version simultaneously so that existing applications can continue to call previous versions after new API versions are published
AWS Authorization	Support for signature version 4 for REST APIs and WebSocket APIs, IAM access policies, and authorization with bearer tokens (e.g. JWT, SAML) using Lambda functions.

# Amazon API Gateway – Methods and Resources

- A resource represents a path in your API.
- Methods are created within resources
- A method represents a client-facing interface by which the client calls the API to access back-end resources.
- A method is created for a specific HTTP verb or you can use ANY.



# Amazon API Gateway – API Methods and Integrations



## Amazon API Gateway – Method Requests / Responses

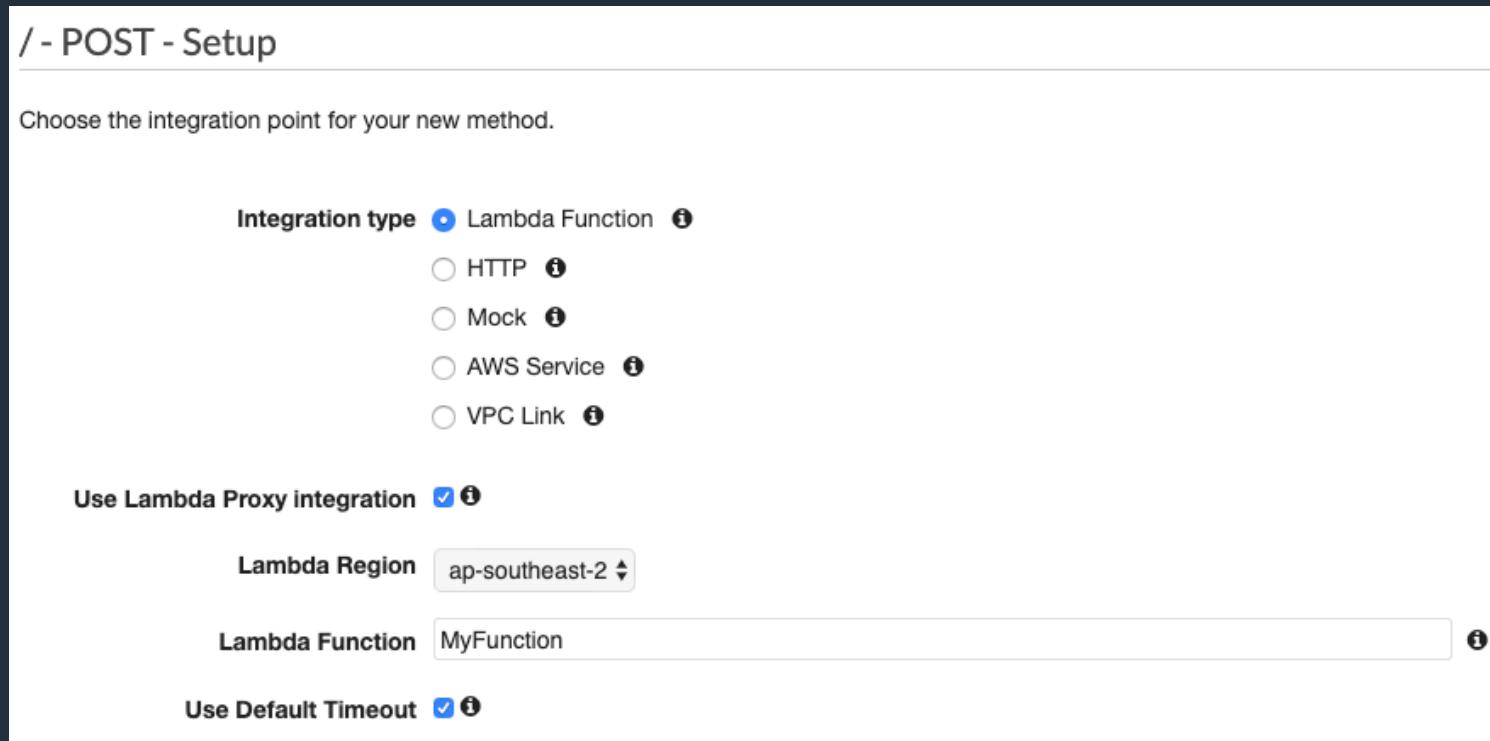
- A method represents a client-facing interface by which the client calls the API to access back-end resources.
- A Method resource is integrated with an Integration resource.
- Method request:
  - The public interface of a REST API method in API Gateway that defines the parameters and body that an app developer must send in requests to access the backend through the API
- Method response:
  - The public interface of a REST API that defines the status codes, headers, and body models that an app developer should expect in responses from the API.

## Amazon API Gateway – Integrations Requests / Responses

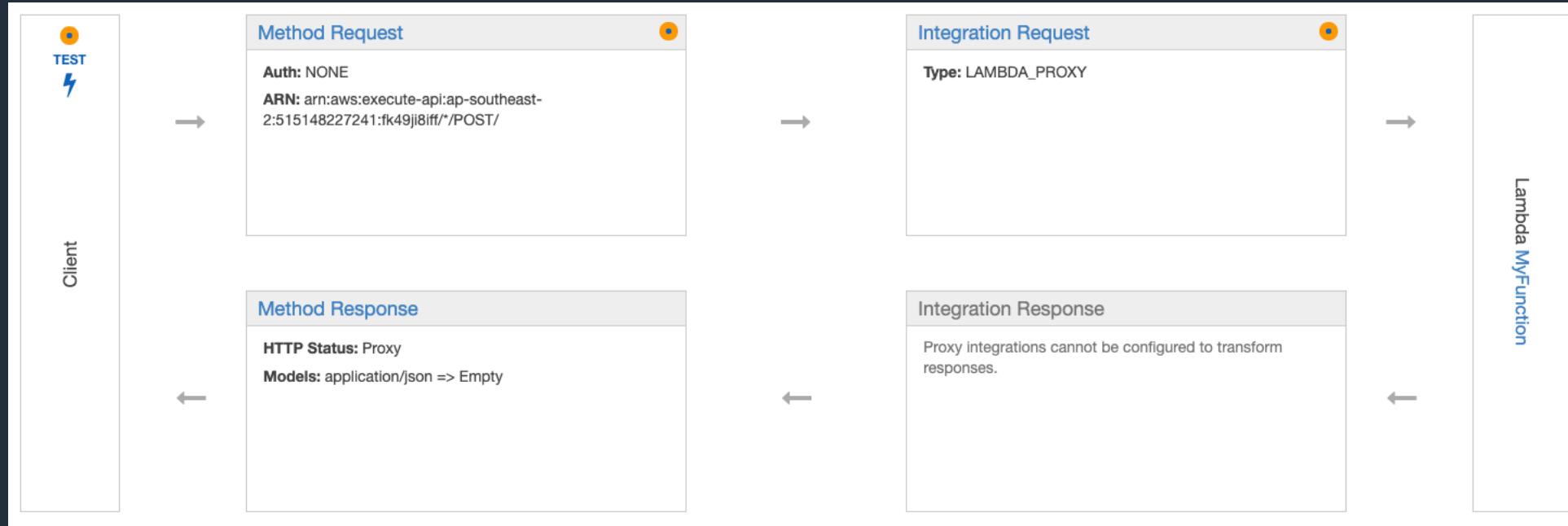
- API methods are integrated with backend endpoints using API integrations.
- Backend endpoints are known as “integration endpoints”.
- Integration request:
  - The internal interface of a WebSocket API route or REST API method in API Gateway, in which you map the body of a route request or the parameters and body of a method request to the formats required by the backend.
- Integration response:
  - The internal interface of a WebSocket API route or REST API method in API Gateway, in which you map the status codes, headers, and payload that are received from the backend to the response format that is returned to a client app.

## Amazon API Gateway – Lambda Proxy Example Configuration

- Set the integration's HTTP method to POST, the integration endpoint URI to the ARN of the Lambda function invocation action of a specific Lambda function, and the credential to an IAM role with permissions to allow API Gateway to call the Lambda function on your behalf:

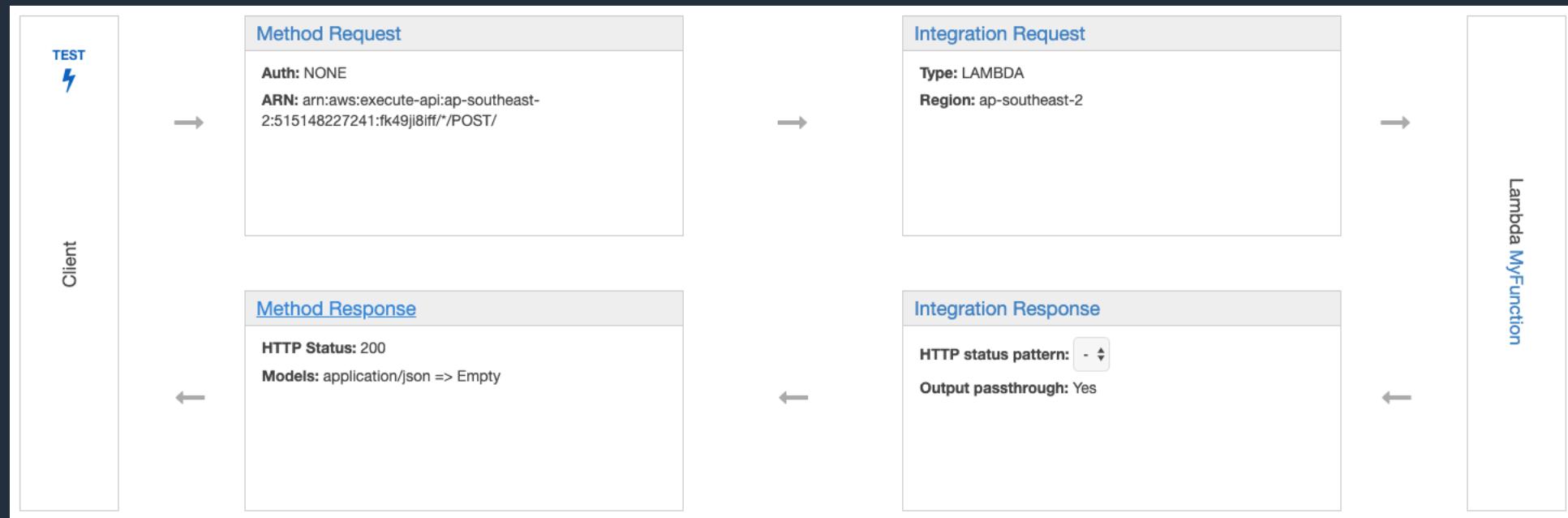


# Amazon API Gateway – Lambda Proxy Example Configuration



# Amazon API Gateway – Lambda Non-Proxy Example Configuration

- In Lambda non-proxy integration, in addition to the proxy integration setup steps, you also specify how the incoming request data is mapped to the integration request and how the resulting integration response data is mapped to the method response:

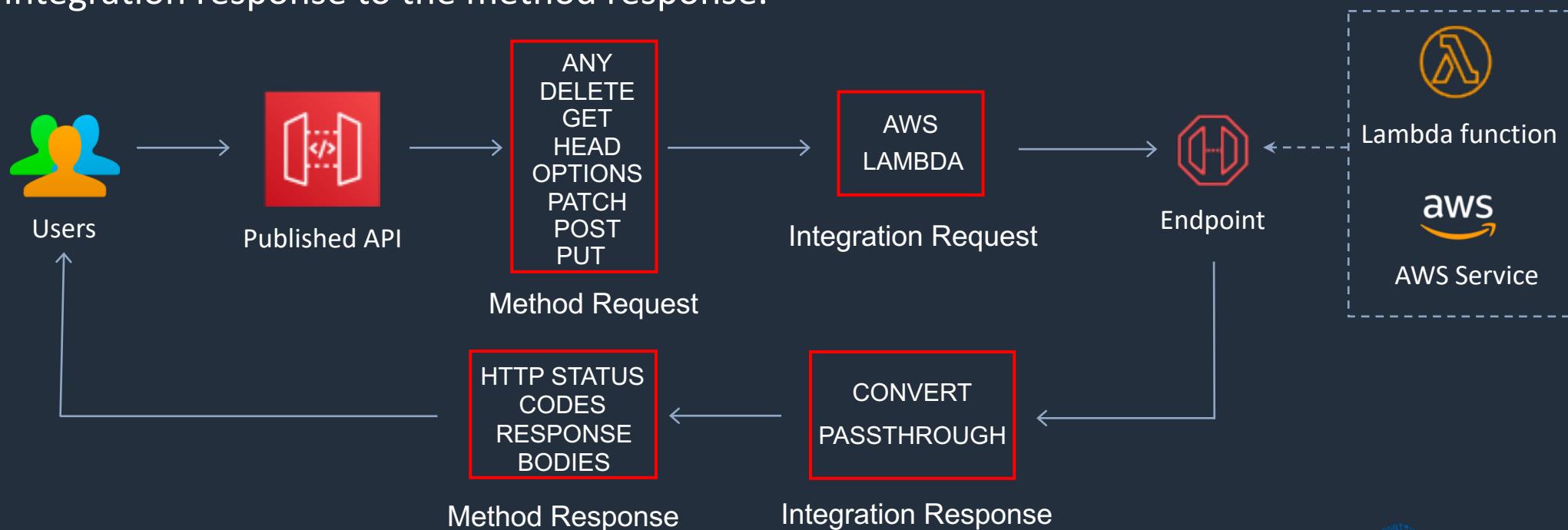


## Amazon API Gateway – Integration Types

- You choose an API integration type according to the types of integration endpoint you work with and how you want data to pass to and from the integration endpoint.
- For a Lambda function, you can have the Lambda proxy integration, or the Lambda custom integration.
- For an HTTP endpoint, you can have the HTTP proxy integration or the HTTP custom integration.
- For an AWS service action, you have the AWS integration of the non-proxy type only.
- API Gateway also supports the mock integration, where API Gateway serves as an integration endpoint to respond to a method request.

# Amazon API Gateway – Integration Types

- AWS:
  - This type of integration lets an API expose AWS service actions.
  - In AWS integration, you must configure both the integration request and integration response and set up necessary data mappings from the method request to the integration request, and from the integration response to the method response.

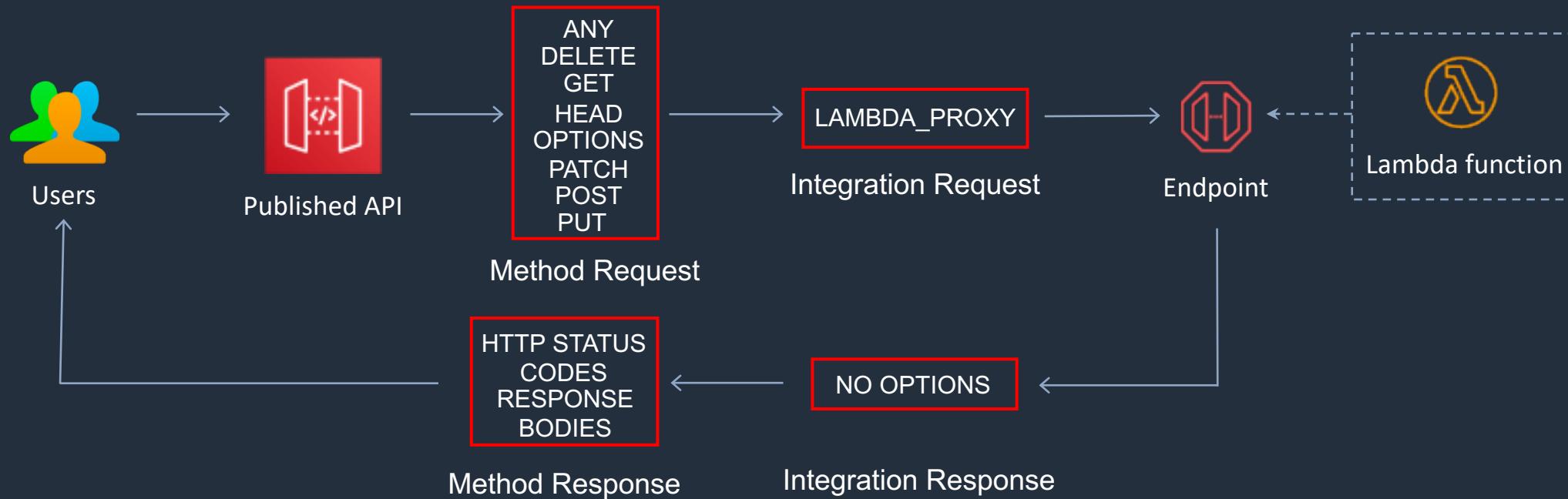


## Amazon API Gateway – Integration Types

- **AWS\_PROXY:**
  - This integration relies on direct interactions between the client and the integrated Lambda function.
  - With this type of integration, also known as the Lambda proxy integration, you do not set the integration request or the integration response.
  - API Gateway passes the incoming request from the client as the input to the backend Lambda function.
  - The integrated Lambda function takes the input of this format and parses the input from all available sources, including request headers, URL path variables, query string parameters, and applicable body.
  - The function returns the result following this output format.

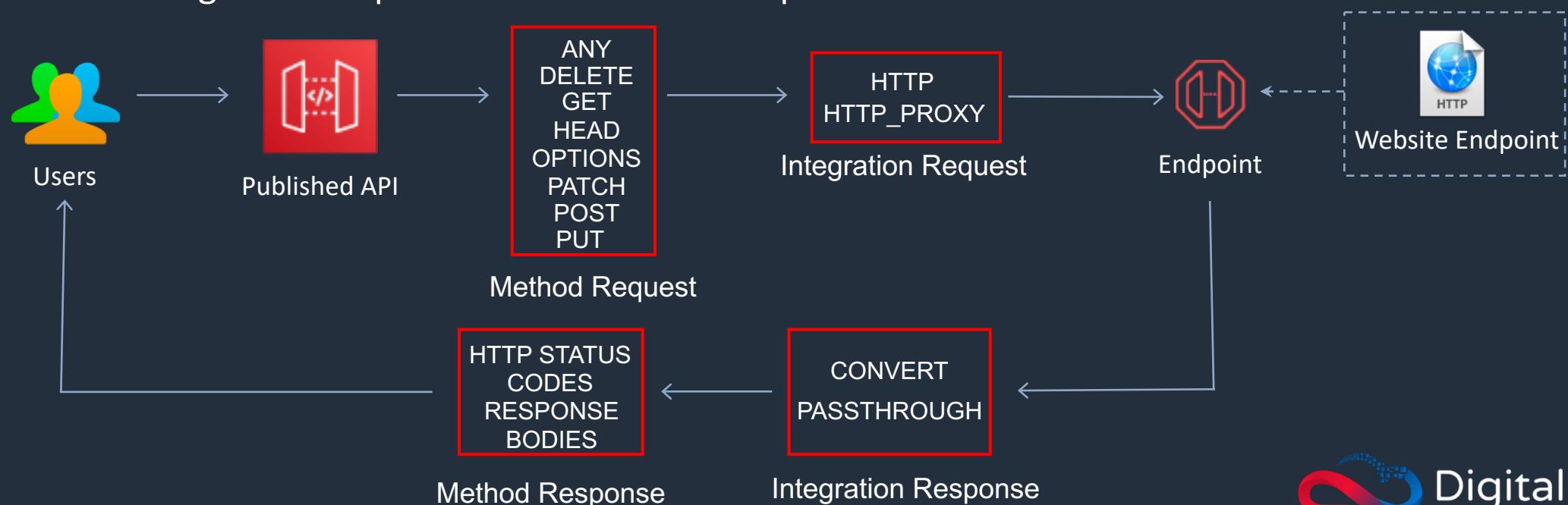
# Amazon API Gateway – Integration Types

## ➤ AWS\_PROXY:



# Amazon API Gateway – Integration Types

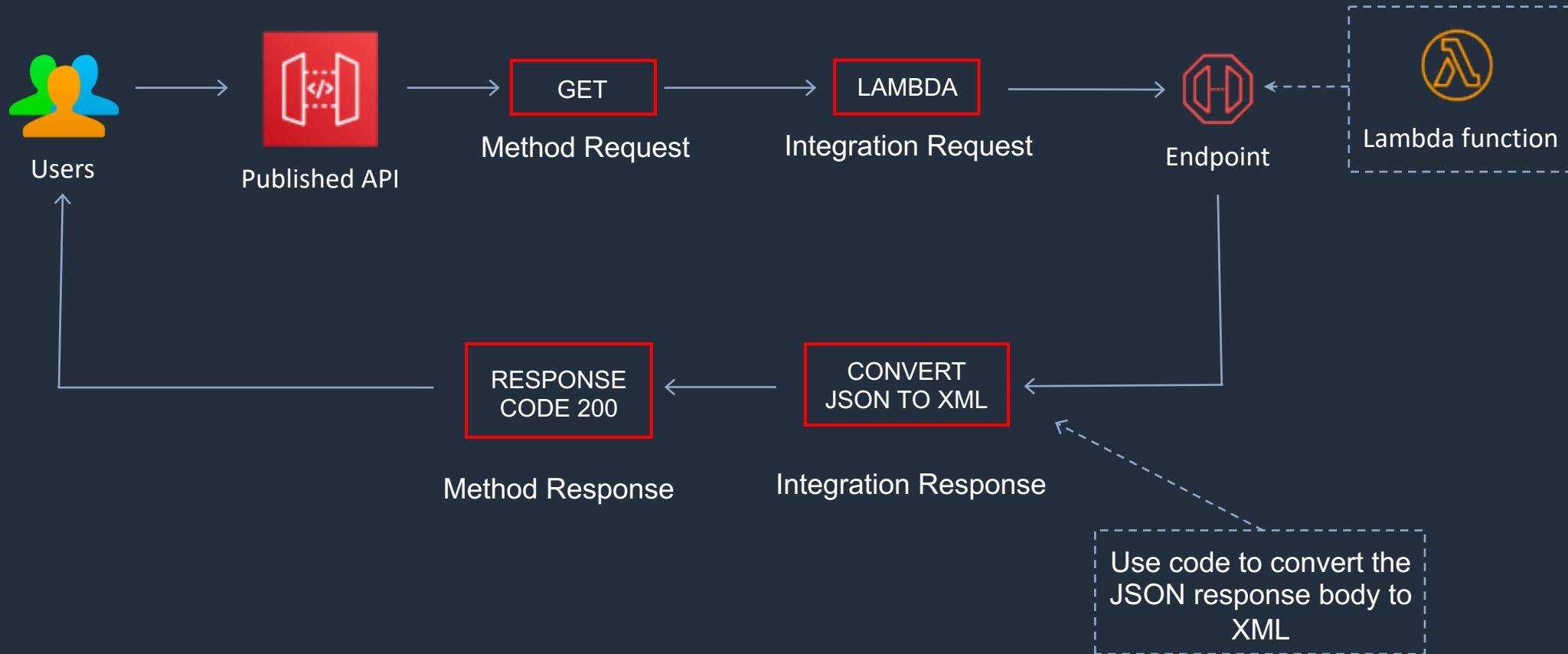
- **HTTP:**
  - This type of integration lets an API expose HTTP endpoints in the backend.
  - With the HTTP integration, also known as the HTTP custom integration, you must configure both the integration request and integration response.
  - You must set up necessary data mappings from the method request to the integration request, and from the integration response to the method response.



## Amazon API Gateway – Integration Types

- **MOCK:**
  - This type of integration lets API Gateway return a response without sending the request further to the backend.
  - This is useful for API testing because it can be used to test the integration set up without incurring charges for using the backend and to enable collaborative development of an API.
  - In collaborative development, a team can isolate their development effort by setting up simulations of API components owned by other teams by using the MOCK integrations.

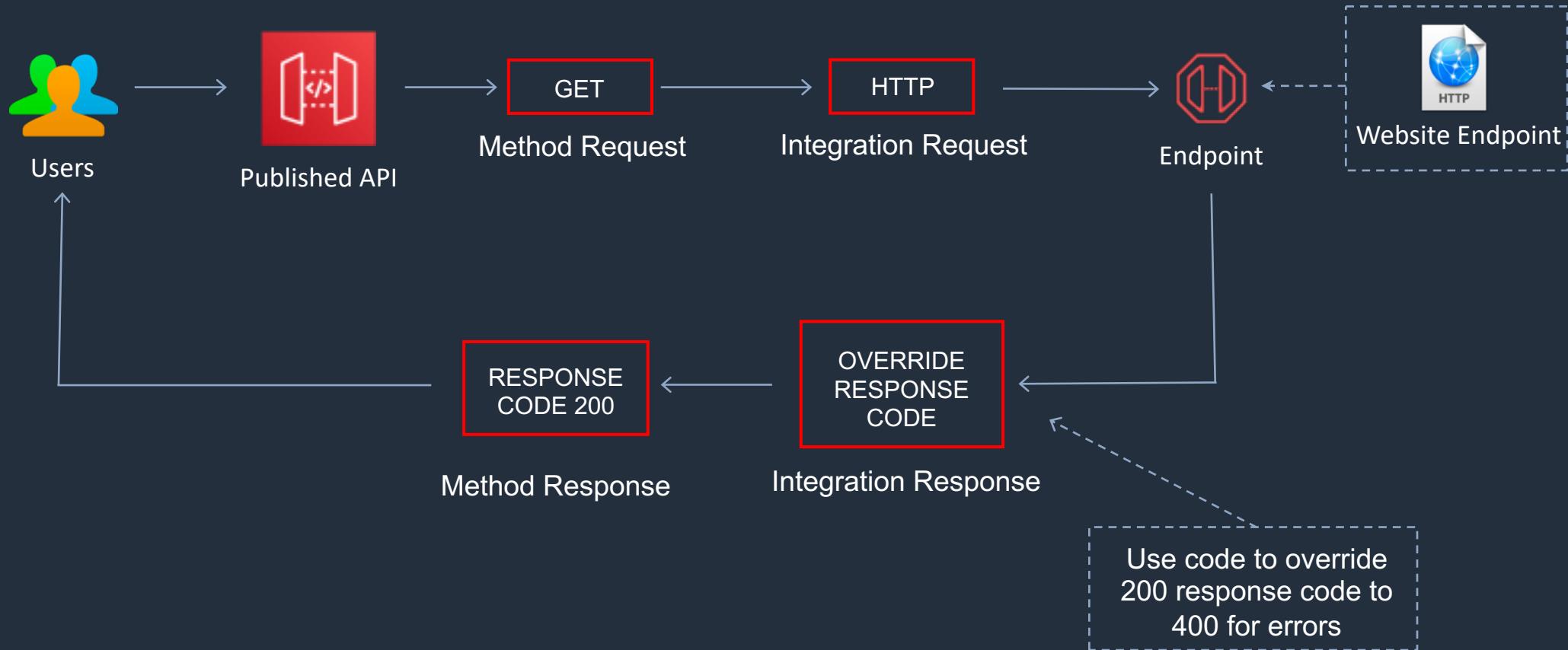
# Amazon API Gateway – Example with Mapping Template to Convert JSON to XML



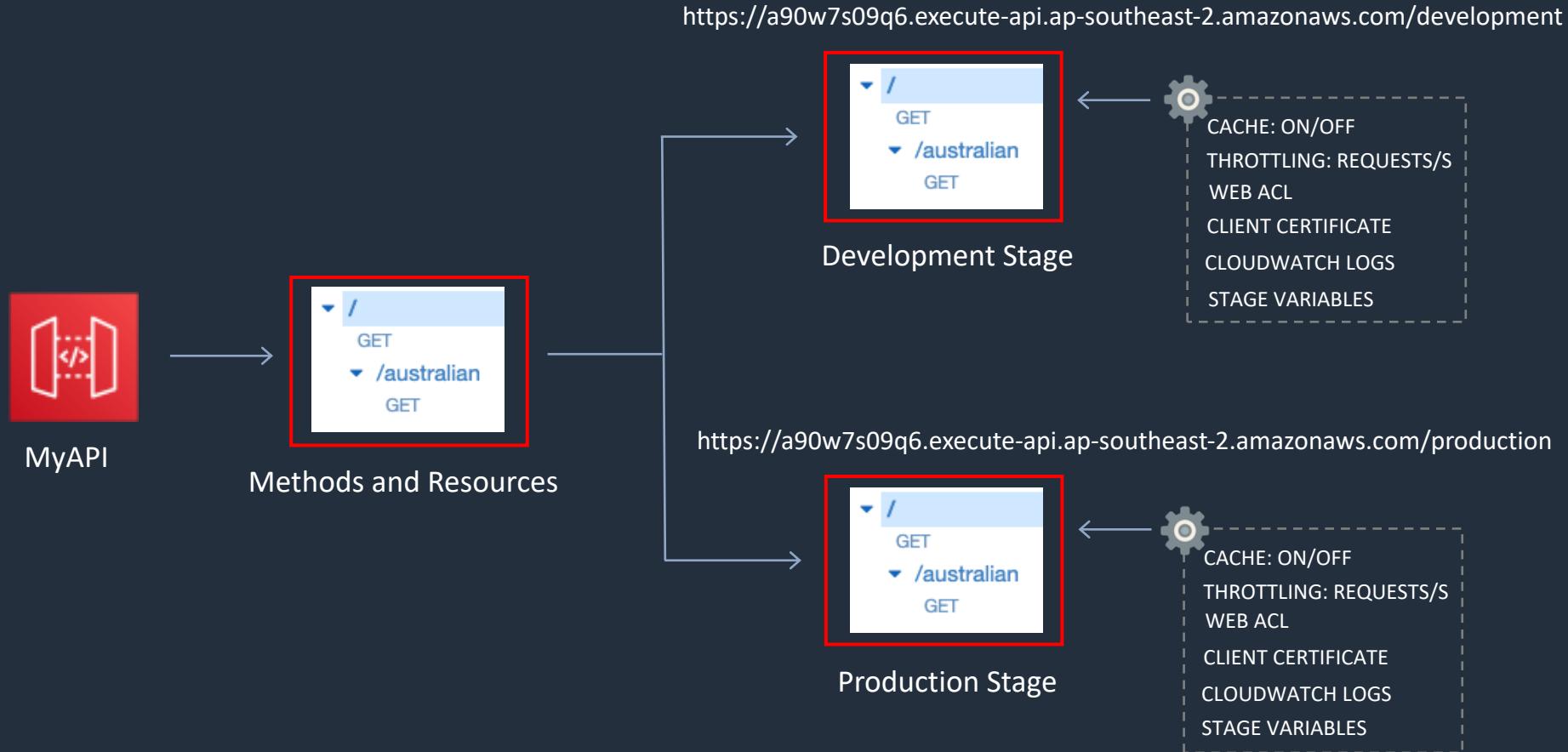
## Amazon API Gateway Mapping Templates

- In API Gateway, an API's method request can take a payload in a different format from the corresponding integration request payload, as required in the backend.
- Similarly, the backend may return an integration response payload different from the method response payload, as expected by the frontend.
- API Gateway lets you use mapping templates to map the payload from a method request to the corresponding integration request and from an integration response to the corresponding method response.
- A mapping template is a script expressed in Velocity Template Language (VTL) and applied to the payload using JSONPath expressions.

# Amazon API Gateway – Example with Mapping Template to Override Response Status Code



# Amazon API Gateway – Stages and Deployments



## Amazon API Gateway – Stages and Deployments

- Deployments are a snapshot of the APIs resources and methods.
- Deployments must be created and associated with a stage in order for anyone to access the API.
- A stage is a logical reference to a lifecycle state of your REST or WebSocket API (for example, ‘dev’, ‘prod’, ‘beta’, ‘v2’).
- API stages are identified by API ID and stage name.
- Stage variables are like environment variables for API Gateway.
- Stage variables can be used in:
- Lambda function ARN.
- HTTP endpoint.
- Parameter mapping templates.

## Amazon API Gateway – Stages and Deployments

- Use cases for stage variables:
  - Configure HTTP endpoints your stages talk to (dev, test, prod etc.).
  - Pass configuration parameters to AWS Lambda through mapping templates.
- Stage variables are passed to the “context” object in Lambda.
- Stage variables are used with Lambda aliases.
- You can create a stage variable to indicate the corresponding Lambda alias.
- You can create canary deployments for any stage and choose the % of traffic the canary channel receives.

# Amazon API Gateway – Open API 3 and Swagger

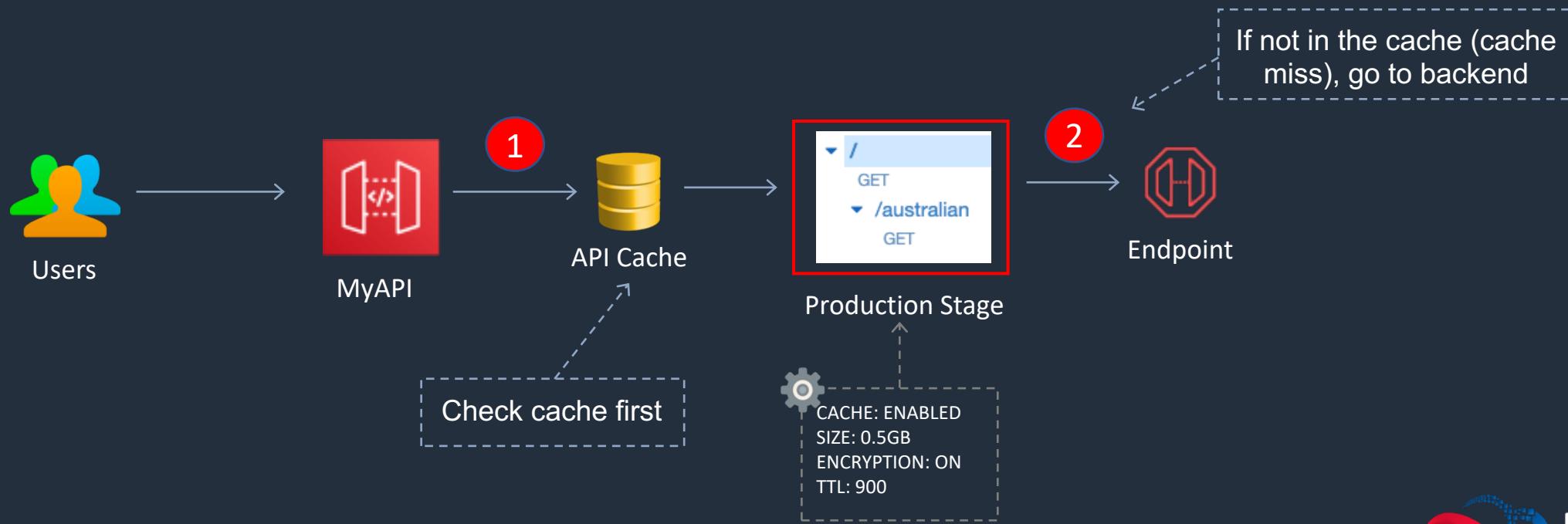
```
{  
  "swagger": "2.0",  
  "info": {  
    "version": "2020-03-15T07:45:21Z",  
    "title": "MyLambdaAPI"  
  },  
  "host": "a90w7s09q6.execute-api.ap-southeast-2.amazonaws.com",  
  "basePath": "/development",  
  "schemes": [  
    "https"  
  ],  
  "paths": {  
    "/": {  
      "get": {  
        "produces": [  
          "application/json"  
        ],  
        "responses": {  
          "200": {  
            "description": "200 response",  
            "schema": {  
              "$ref": "#/definitions/Empty"  
            }  
          }  
        }  
      }  
    }  
  }  
}
```

## Amazon API Gateway – Open API 3 and Swagger

- Can import existing Swagger / Open API 3.0 definitions (written in YAML or JSON) to API Gateway.
- This is a common way of defining REST APIs using API definition as code.
- Can also export current APIs as Swagger / Open API 3.0 definition.
- Uses the API Gateway Import API feature to import an API from an external definition.
- With the import API you can either create a new API by submitting a POST request that includes a Swagger definition in the payload and endpoint configuration or you can update an existing API by using a PUT request that contains a Swagger definition, or merge a definition with an existing API.
- You specify the options using a mode query parameter in the request URL.

## Amazon API Gateway – Caching

- You can add caching to API calls by provisioning an Amazon API Gateway cache and specifying its size in gigabytes.
- Caching allows you to cache the endpoint's response.
- Caching can reduce number of calls to the backend and improve latency of requests to the API.



## Amazon API Gateway – Caching

- API Gateway caches responses for a specific amount of time (time to live or TTL).
- The default TTL is 300 seconds (min 0, max 3600).
- Caches are defined per stage.
- You can encrypt caches.
- The cache capacity is between 0.5GB to 237GB.
- It is possible to override cache settings for specific methods.
- You are able to flush the entire cache (invalidate it) immediately if required.
- Clients can invalidate the cache with the header: Cache-Control: max-age=0.

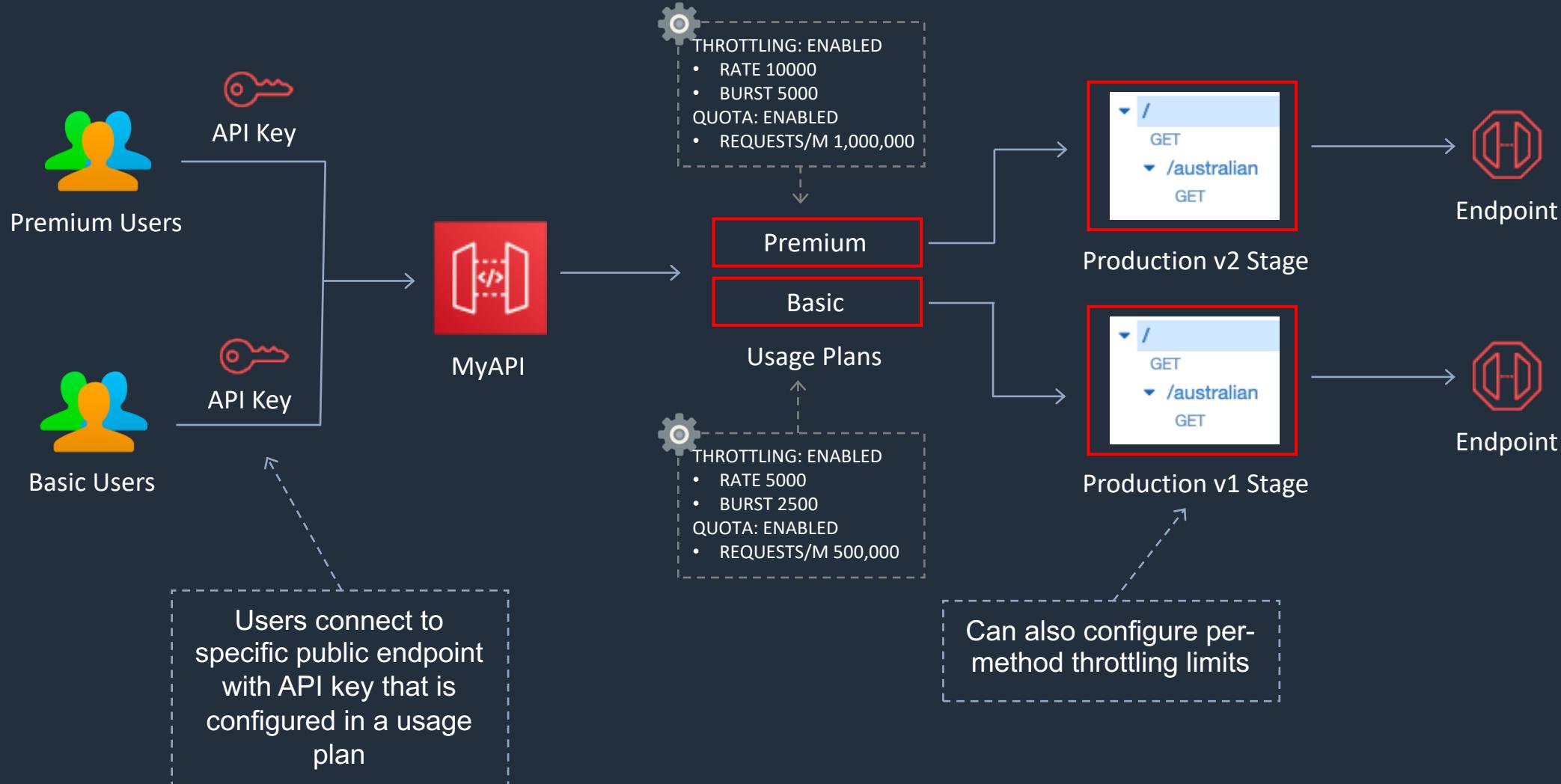
## Amazon API Gateway – Throttling

- API Gateway sets a limit on a steady-state rate and a burst of request submissions against all APIs in your account.
- Limits:
  - By default API Gateway limits the steady-state request rate to 10,000 requests per second.
  - The maximum concurrent requests is 5,000 requests across all APIs within an AWS account.
  - If you go over 10,000 requests per second or 5,000 concurrent requests you will receive a **429 Too Many Requests** error response.
- Upon catching such exceptions, the client can resubmit the failed requests in a way that is rate limiting, while complying with the API Gateway throttling limits.

## Amazon API Gateway – Throttling

- Amazon API Gateway provides two basic types of throttling-related settings:
  - **Server-side** throttling limits are applied across all clients. These limit settings exist to prevent your API—and your account—from being overwhelmed by too many requests.
  - **Per-client** throttling limits are applied to clients that use API keys associated with your usage policy as client identifier.
- API Gateway throttling-related settings are applied in the following order:
  1. Per-client per-method throttling limits that you set for an API stage in a usage plan
  2. Per-client throttling limits that you set in a usage plan
  3. Default per-method limits and individual per-method limits that you set in API stage settings
  4. Account-level throttling

# Amazon API Gateway – Usage Plans and API Keys



## Amazon API Gateway – Usage Plans and API Keys

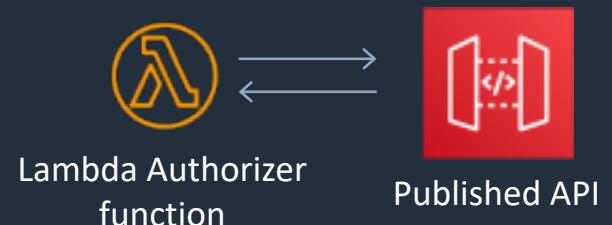
- A usage plan specifies who can access one or more deployed API stages and methods — and also how much and how fast they can access them.
- You can use a usage plan to configure throttling and quota limits, which are enforced on individual client API keys.
- The plan uses API keys to identify API clients and meters access to the associated API stages for each key.
- It also lets you configure throttling limits and quota limits that are enforced on individual client API keys.
- You can use API keys together with usage plans or Lambda authorizers to control access to your APIs.

# Access Control - IAM, Authorizers and User Pools

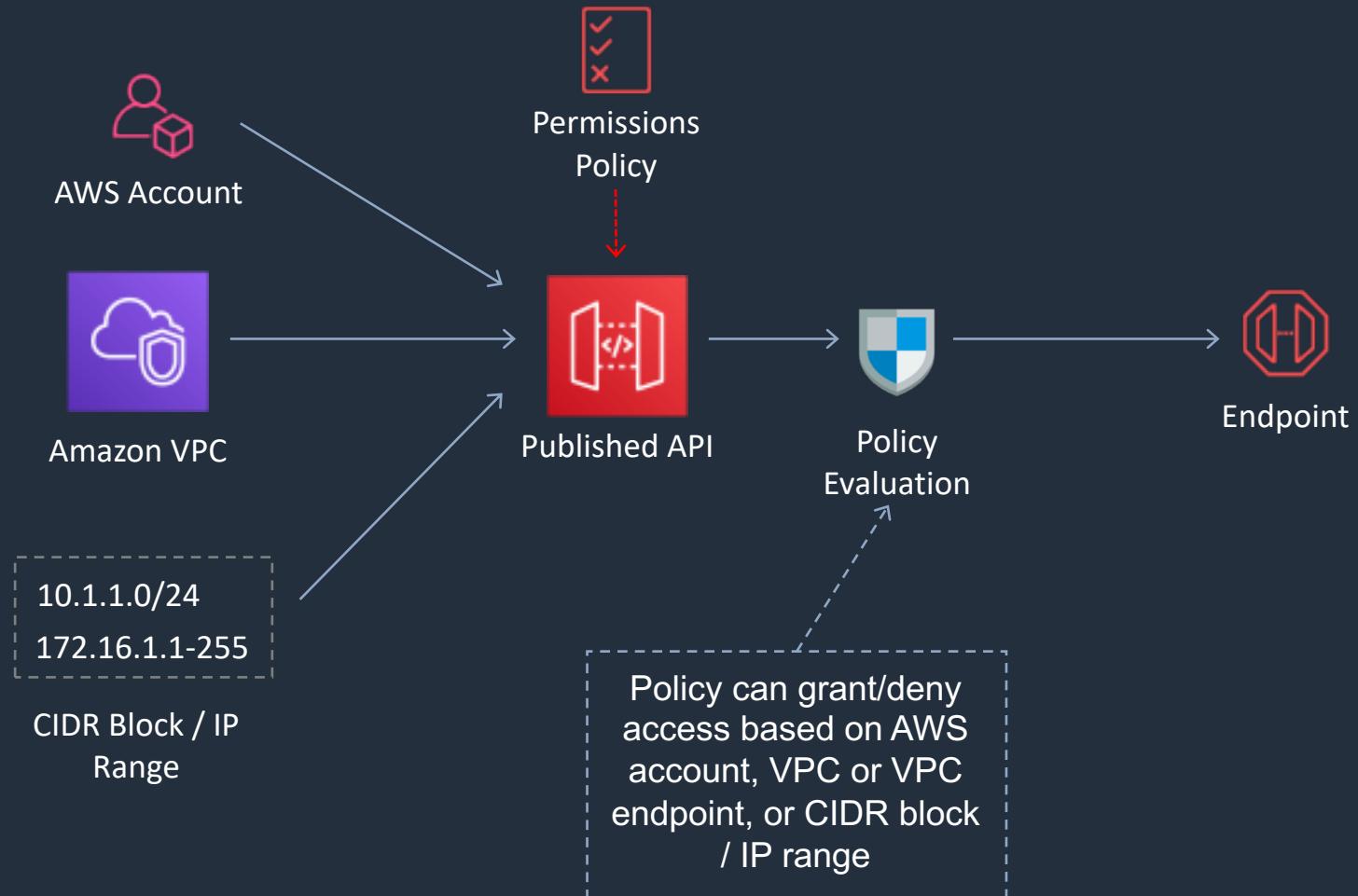
- There are several mechanisms for controlling and managing access to an API.
- These mechanisms include:
  - Resource-based policies.
  - Standard IAM Roles and Policies (identity-based policies).
  - IAM Tags.
  - Endpoint policies for interface VPC endpoints.
  - Lambda authorizers.
  - Amazon Cognito user pools.



IAM Policy



# Access Control - API Gateway Resource Policies



## Access Control - API Gateway Resource-Based Policies

- Amazon API Gateway resource policies are JSON policy documents that you attach to an API to control whether a specified principal (typically an IAM user or role) can invoke the API.
- You can use API Gateway resource policies to allow your API to be securely invoked by:
  - Users from a specified AWS account.
  - Specified source IP address ranges or CIDR blocks.
  - Specified virtual private clouds (VPCs) or VPC endpoints (in any account).
  - You can use resource policies for all API endpoint types in API Gateway: private, edge-optimized, and Regional.

# Access Control – IAM Permissions

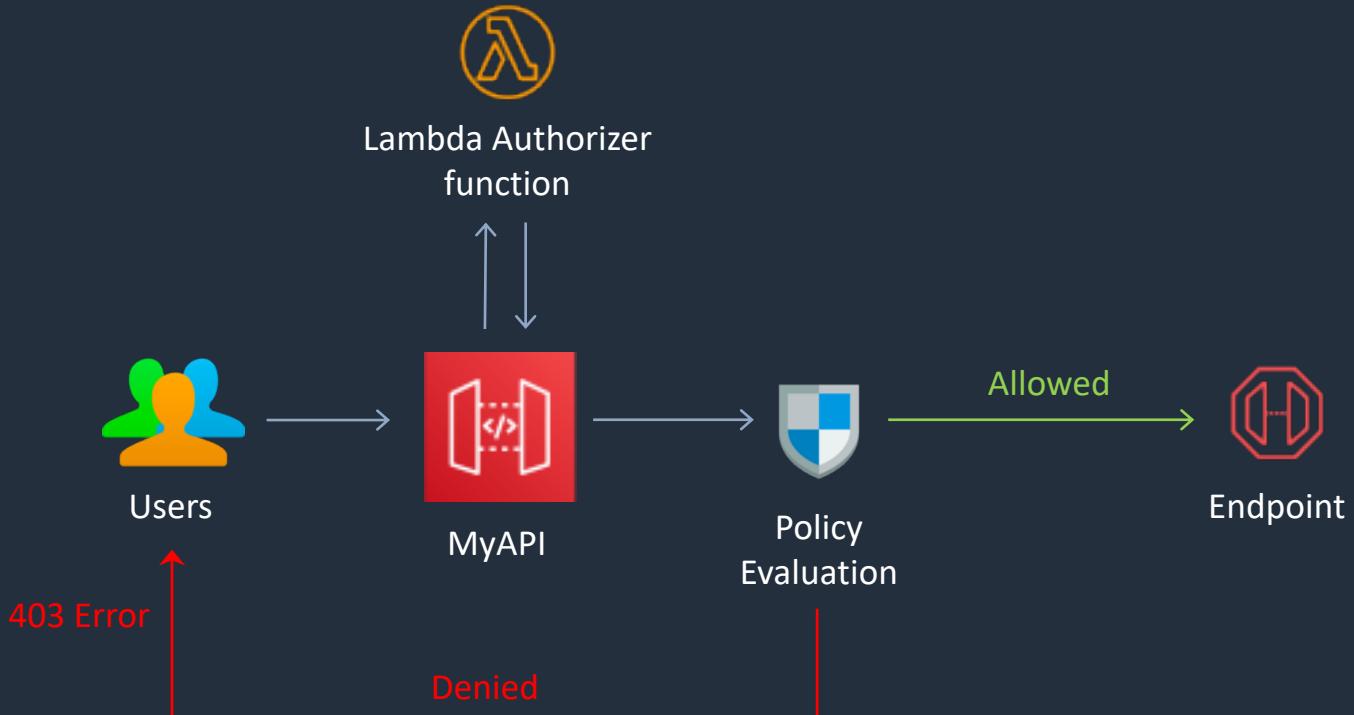
```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Permission",  
            "Action": [  
                "execute-api:Execution-operation"  
            ],  
            "Resource": [  
                "arn:aws:execute-api:region:account-id:api-id/stage/METHOD_HTTP_VERB/Resource-path"  
            ]  
        }  
    ]  
}
```

The diagram illustrates the structure of an IAM policy JSON document with three annotations:

- A dashed box labeled "Allow or deny" covers the "Effect" field.
- A dashed box labeled "Supported operations" covers the "Action" field.
- A dashed box labeled "ARN of deployed API" covers the "Resource" field.

# Amazon API Gateway – Lambda Authorizers

1. Client calls API method, passing bearer token or request parameters.
2. API Gateway calls the Lambda authorizer.
3. Lambda function authenticates the user using:
  - Call OAuth provider to get token.
  - Call SAML provider to get assertion.
  - Generate IAM Policy based on request parameters.
4. If successful, Lambda grants access and returns IAM policy and a principal identifier.
5. API Gateway evaluates the policy:
  - If access is allowed, execute the method.
  - If access is denied, return status code (e.g. 403).



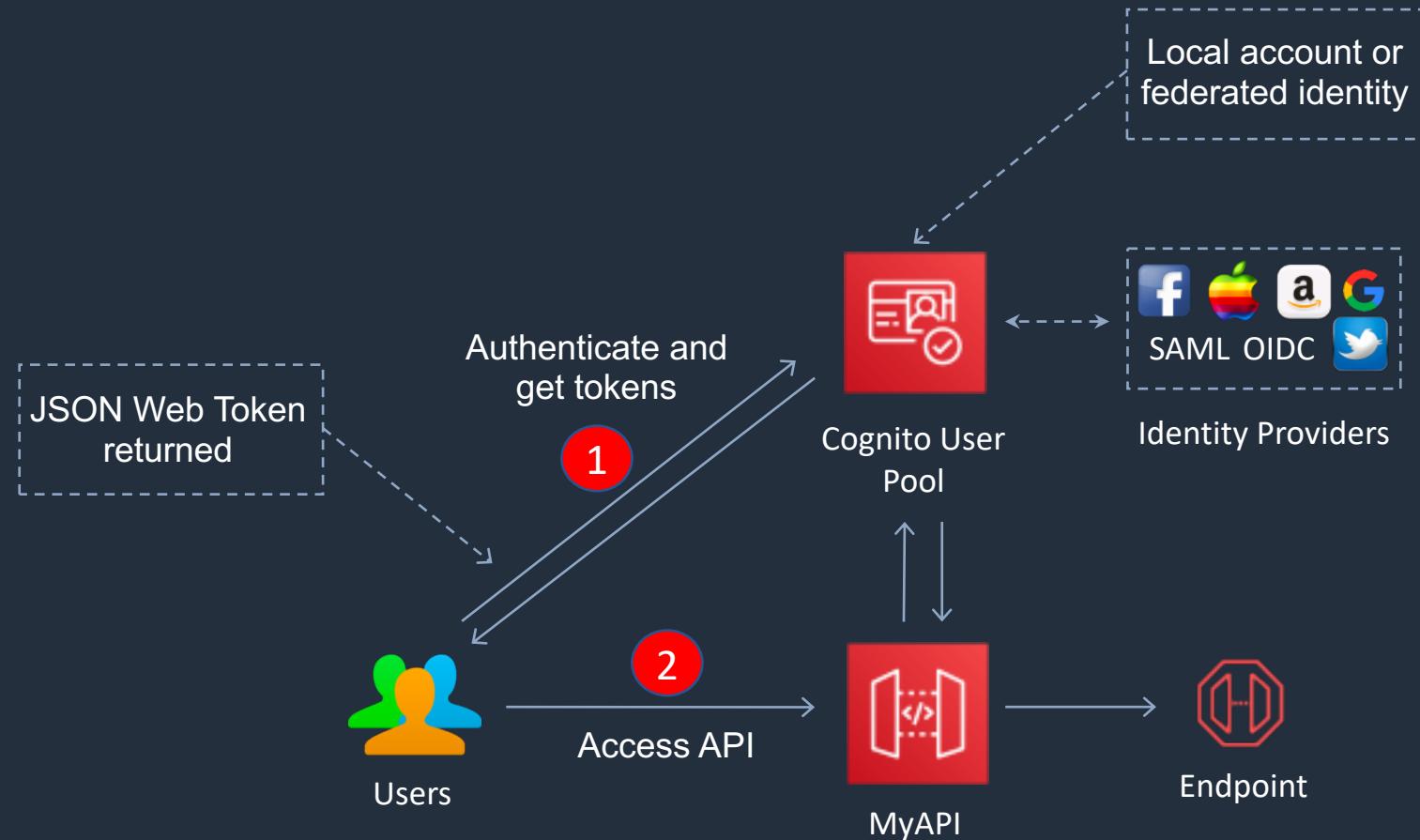
## Access Control – Lambda Authorizers

- A Lambda authorizer (formerly known as a custom authorizer) is an API Gateway feature that uses a Lambda function to control access to your API.
- A Lambda authorizer is useful if you want to implement a custom authorization scheme that uses a bearer token authentication strategy such as OAuth or SAML, or that uses request parameters to determine the caller's identity.
- When a client makes a request to one of your API's methods, API Gateway calls your Lambda authorizer, which takes the caller's identity as input and returns an IAM policy as output.

## Access Control – Lambda Authorizers

- There are two types of Lambda authorizers:
  - A token-based Lambda authorizer (also called a TOKEN authorizer) receives the caller's identity in a bearer token, such as a JSON Web Token (JWT) or an OAuth token.
  - A request parameter-based Lambda authorizer (also called a REQUEST authorizer) receives the caller's identity in a combination of headers, query string parameters, stageVariables, and \$context variables.
- For WebSocket APIs, only request parameter-based authorizers are supported.

# Access Control – Cognito User Pool Authorizer



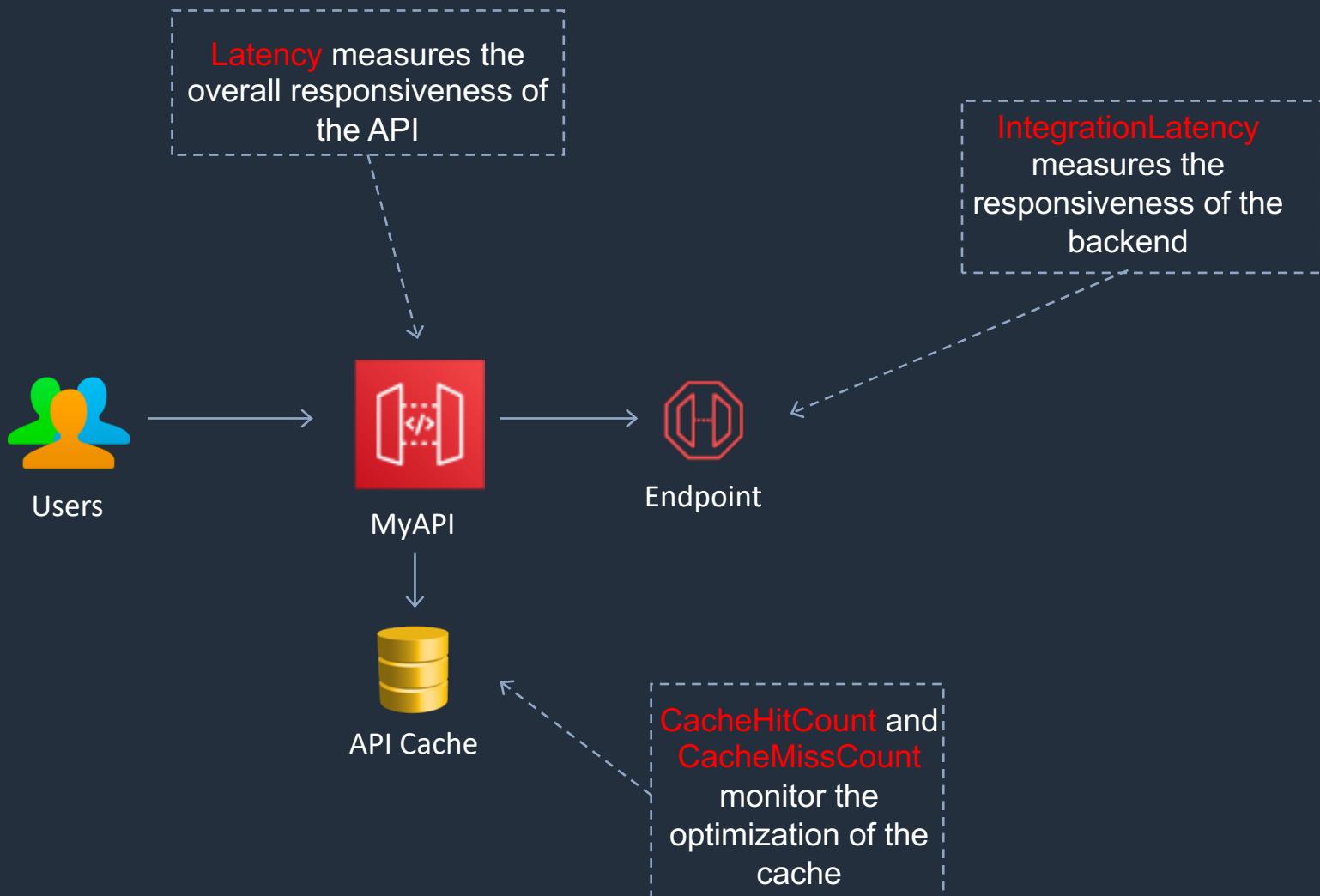
## Access Control – Cognito User Pool Authorizer

- A user pool is a user directory in Amazon Cognito.
- With a user pool, users can sign in to a web or mobile app through Amazon Cognito.
- Users can also sign in through social identity providers like Google, Facebook, Amazon, or Apple, and through SAML identity providers.
- Whether your users sign in directly or through a third party, all members of the user pool have a directory profile that you can access through a Software Development Kit (SDK).

## Access Control – Cognito User Pool Authorizer

- User pools provide:
  - Sign-up and sign-in services.
  - A built-in, customizable web UI to sign in users.
  - Social sign-in with Facebook, Google, Login with Amazon, and Sign in with Apple, as well as sign-in with SAML identity providers from your user pool.
  - User directory management and user profiles.
  - Security features such as multi-factor authentication (MFA),

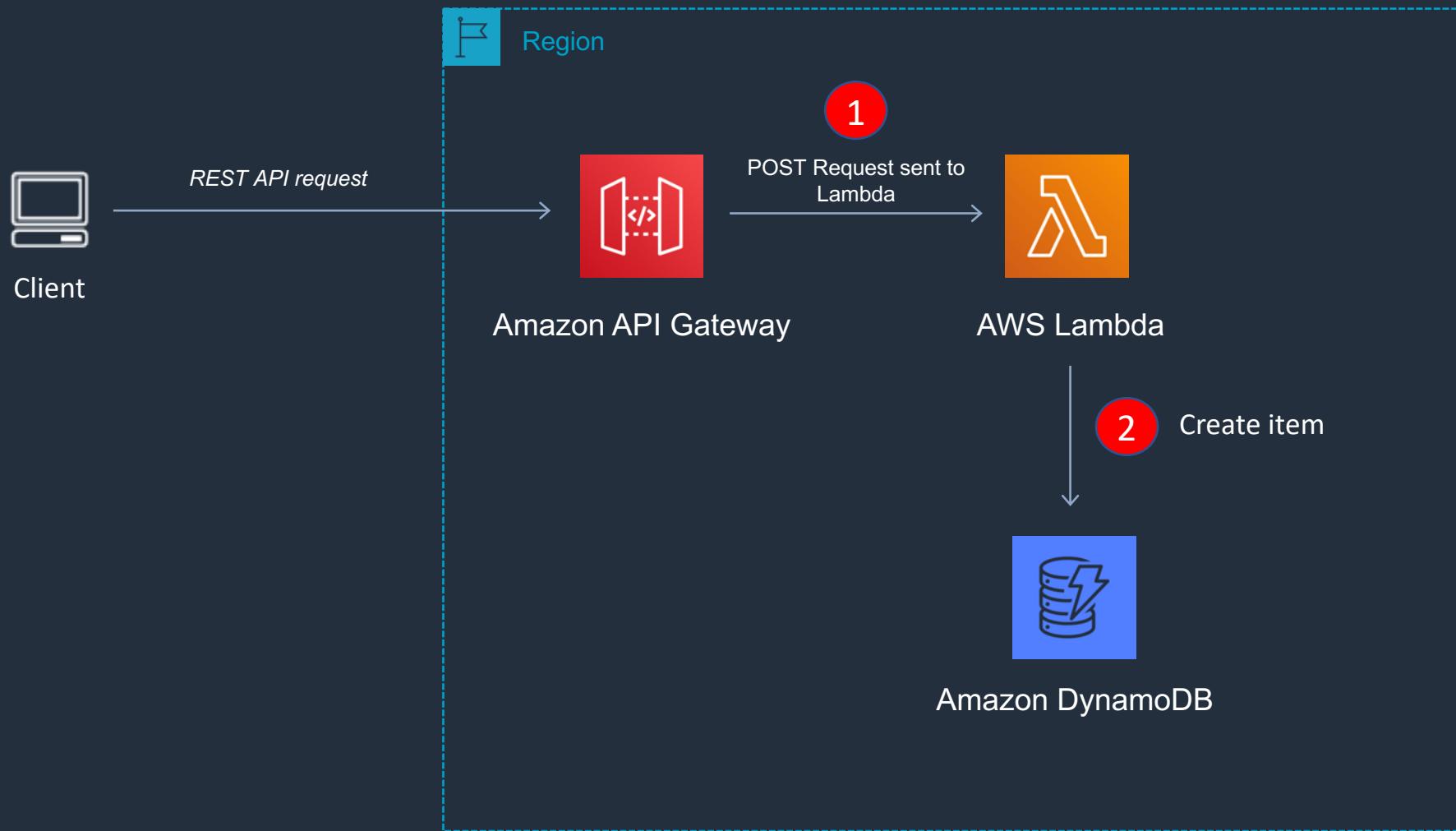
# Amazon API Gateway – Monitoring, Logging and Auditing



## Amazon API Gateway – Monitoring, Logging and Auditing

- The Amazon API Gateway logs (near real time) back-end performance metrics such as API calls, latency, and error rates to CloudWatch.
- You can monitor through the API Gateway dashboard (REST API) allowing you to visually monitor calls to the services.
- API Gateway also meters utilization by third-party developers and the data is available in the API Gateway console and through APIs.
- Amazon API Gateway is integrated with AWS CloudTrail to give a full auditable history of the changes to your REST APIs.
- All API calls made to the Amazon API Gateway APIs to create, modify, delete, or deploy REST APIs are logged to CloudTrail.

# Microservice with Lambda, API Gateway and DynamoDB



# SECTION 17

## AWS Serverless Application Model (SAM)

# AWS Serverless Application Model (SAM)



- Provides a **shorthand syntax** to express functions, APIs, databases, and event source mappings.
- Can be used to create Lambda functions, API endpoints, DynamoDB tables, and other resources.

```
AWSTemplateFormatVersion: '2010-09-09'  
Transform: AWS::Serverless-2016-10-31  
Resources:  
  HelloWorldFunction:  
    Type: AWS::Serverless::Function  
    Properties:  
      CodeUri: hello-world/  
      Handler: app.lambdaHandler  
      Runtime: nodejs12.x
```

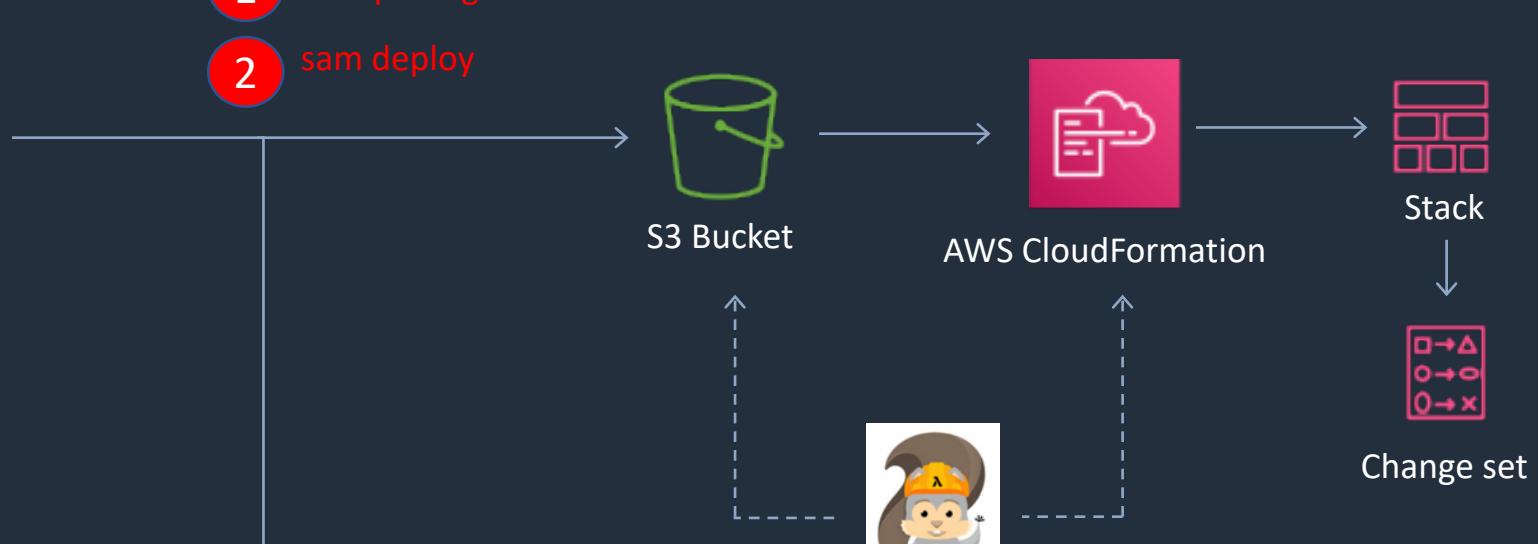
SAM Template (YAML)



Application Code

Commands:

- 1 sam package
- 2 sam deploy



SAM uploads source files to bucket  
and initiates CloudFormation

## AWS Serverless Application Model (AWS SAM)

- A SAM template file is a YAML configuration that represents the architecture of a serverless application.
- You use the template to declare all of the AWS resources that comprise your serverless application in one place.
- AWS SAM templates are an extension of AWS CloudFormation templates, so any resource that you can declare in an AWS CloudFormation template can also be declared in an AWS SAM template.

# AWS Serverless Application Model (AWS SAM)

Commands to run:

```
sam package -t template.yaml --s3-bucket dctlabs --  
output-template-file packaged-template.yaml
```

```
sam deploy --template-file packaged-template.yaml --  
stack-name my-cf-stack
```

Alternatively, run:

```
aws cloudformation package --template-file  
template.yaml --s3-bucket dctlabs --output-template-  
file packaged-template.yaml
```

```
aws cloudformation deploy --template-file packaged-  
template.yaml --stack-name my-cf-stack
```

## AWS Serverless Application Model (AWS SAM)

- The “Transform” header indicates it’s a SAM template:

**Transform: 'AWS::Serverless-2016-10-31'**

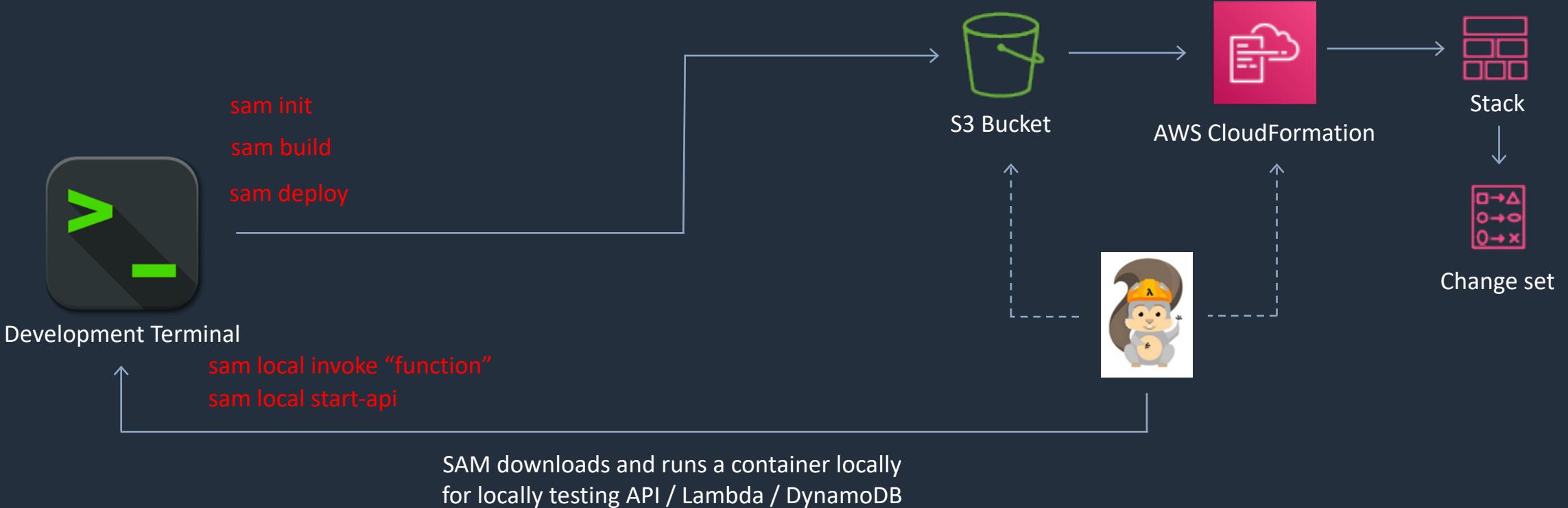
- There are several resources types:

- **AWS::Serverless::Function** (AWS Lambda)
- **AWS::Serverless::Api** (API Gateway)
- **AWS::Serverless::SimpleTable** (DynamoDB)
- **AWS::Serverless::Application** (AWS Serverless Application Repository)
- **AWS::Serverless::HttpApi** (API Gateway HTTP API)
- **AWS::Serverless::LayerVersion** (Lambda layers)

## AWS Serverless Application Model (AWS SAM)

- Installing AWS SAM
  - Need permissions to call AWS services.
  - Install Docker if you need to test your application locally.
  - Instructions available for Linux, Windows, and macOS

# AWS Serverless Application Model (AWS SAM)



# AWS Serverless Application Model (AWS SAM)

Commands to run:

```
sam package -t template.yaml --s3-bucket dctlabs --  
output-template-file packaged-template.yaml
```

```
sam deploy --template-file packaged-template.yaml --  
stack-name my-cf-stack
```

# AWS Serverless Application Model (AWS SAM)

```
AwSTemplateFormatVersion: '2010-09-09'  
Transform: AWS::Serverless-2016-10-31  
Resources:  
  HelloWorldFunction:  
    Type: AWS::Serverless::Function  
    Properties:  
      CodeUri: hello-world/  
      Handler: app.lambdaHandler  
      Runtime: nodejs12.x
```

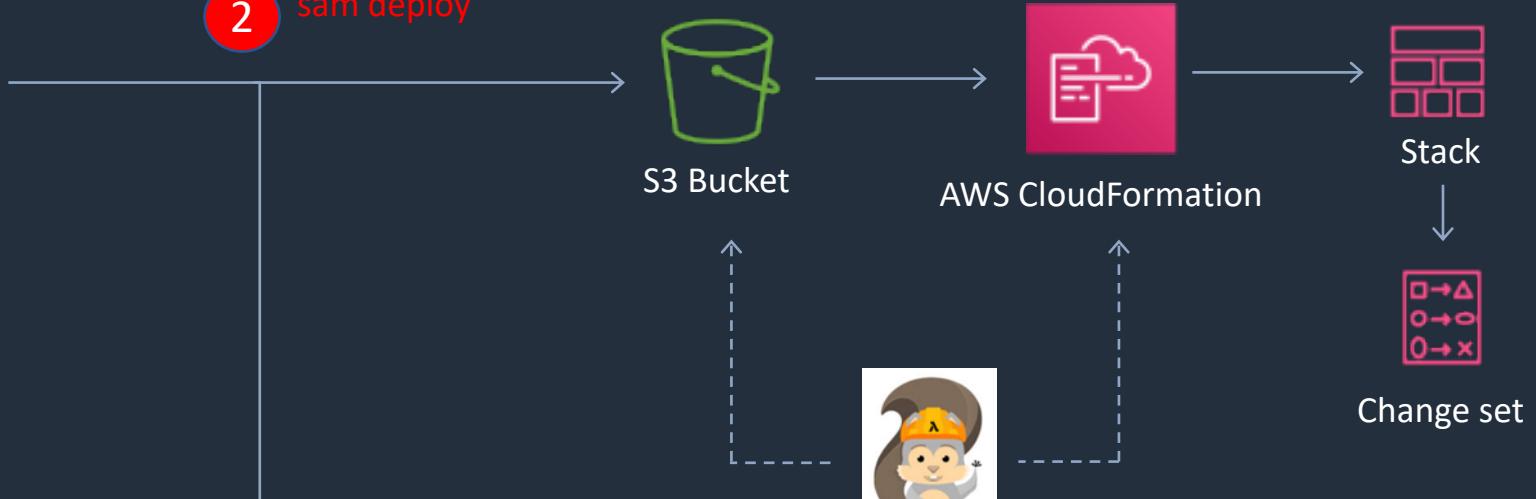
SAM Template (YAML)



Application Code

Commands:

- 1 sam package
- 2 sam deploy

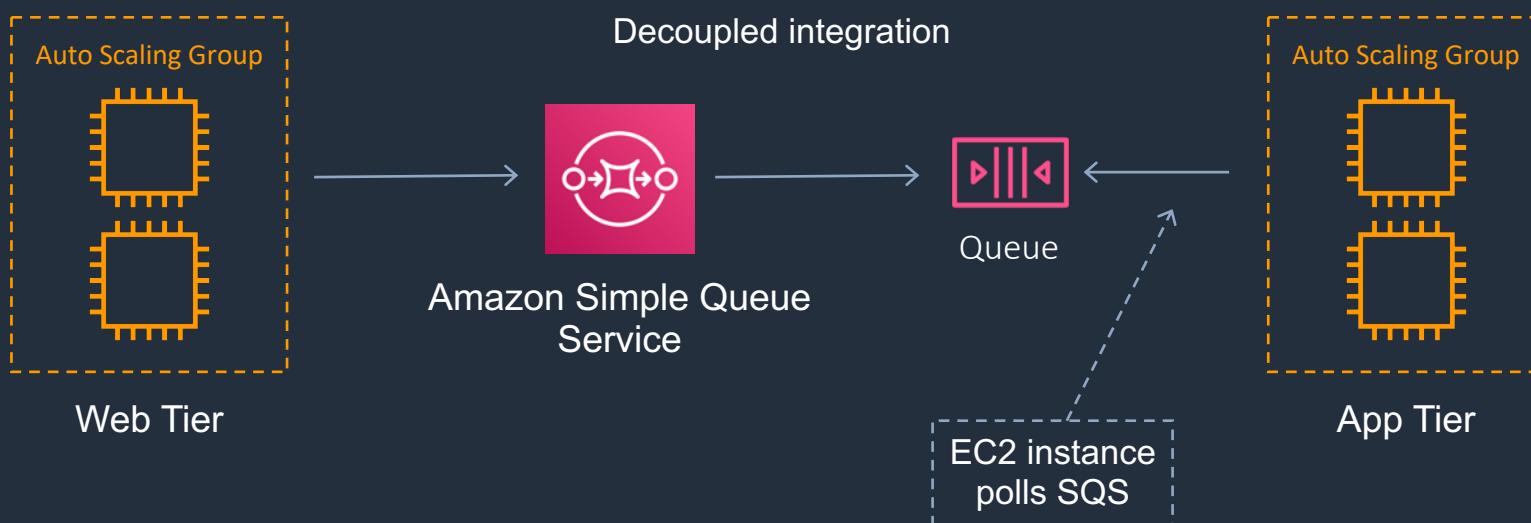


SAM uploads source files to bucket  
and initiates CloudFormation

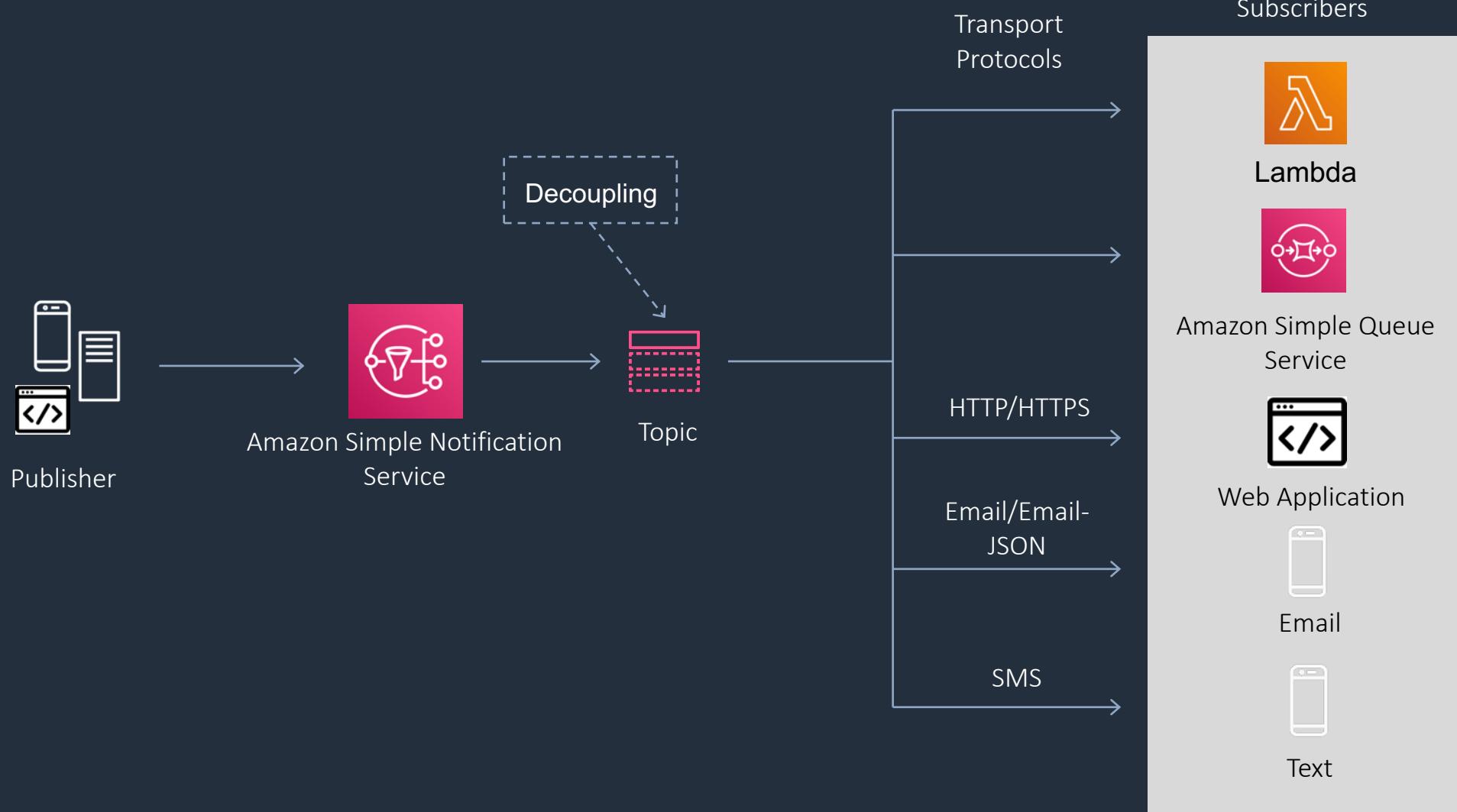
# SECTION 18

## AWS Application Integration

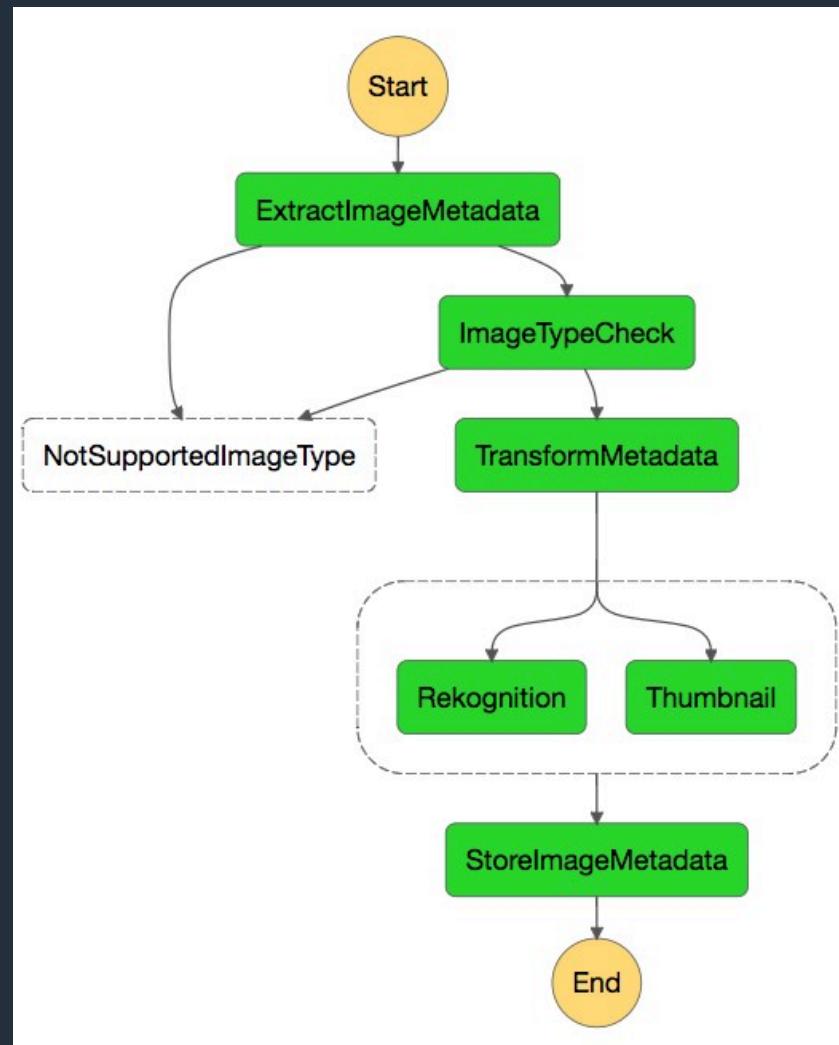
# Decoupling with a Message Queue



# Decoupling with a Notification Service



# Serverless Workflow Services



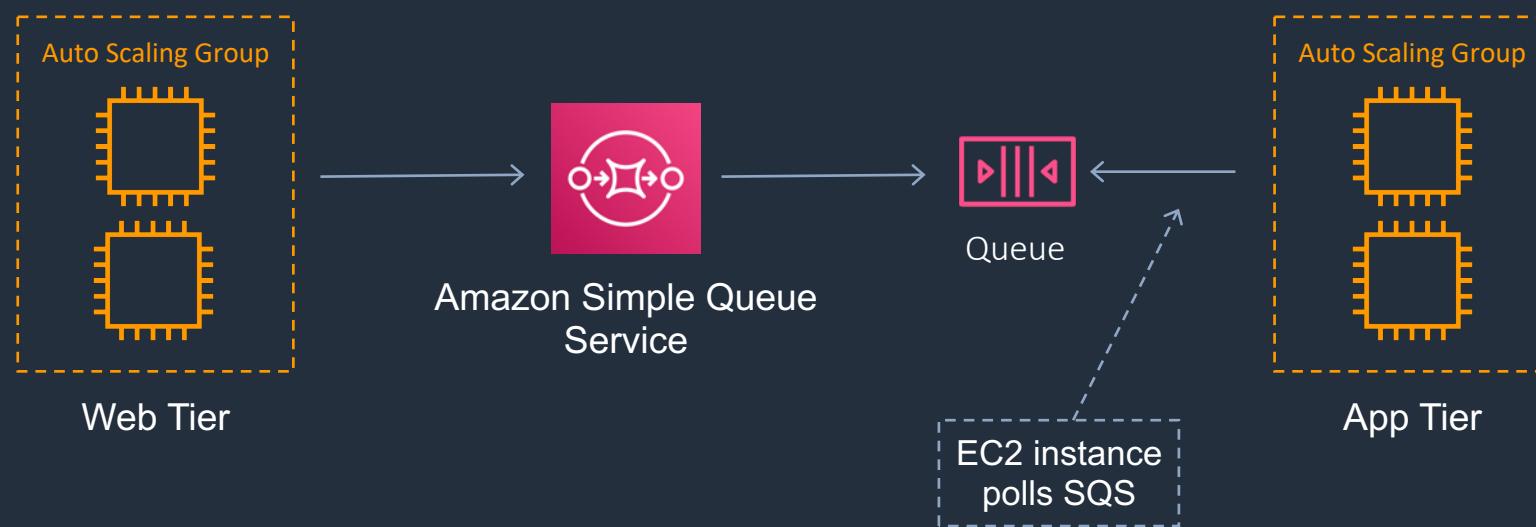
AWS Step Functions State Machine

# Application Integration Services

Service	What it does	Example use cases
Simple Queue Service	Messaging queue; store and forward patterns	Building distributed / decoupled applications
Simple Notification Service	Set up, operate, and send notifications from the cloud	Send email notification when CloudWatch alarm is triggered
Step Functions	Out-of-the-box coordination of AWS service components with visual workflow	Order processing workflow
Simple Workflow Service	Need to support external processes or specialized execution logic	Human-enabled workflows like an order fulfilment system or for procedural requests  Note: AWS recommends that for new applications customers consider Step Functions instead of SWF
Amazon Kinesis	Collect, process, and analyze streaming data.	Collect data from IoT devices for later processing

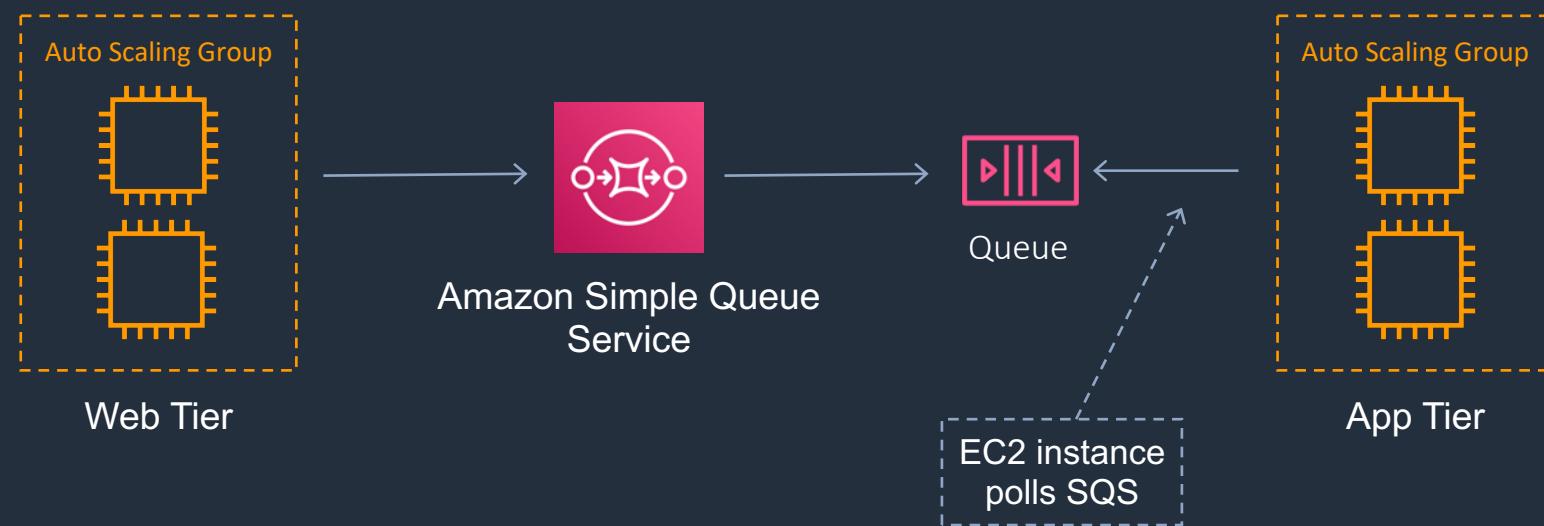
# Amazon Simple Queue Service (SQS)

- Amazon SQS is a fully managed message queuing service
- Amazon SQS queues messages received from one application component ready for consumption by another component.
- A queue is a temporary repository for messages that are awaiting processing.
- The queue acts as a buffer between the component producing and saving data, and the component receiving the data for processing.



# Amazon Simple Queue Service (SQS)

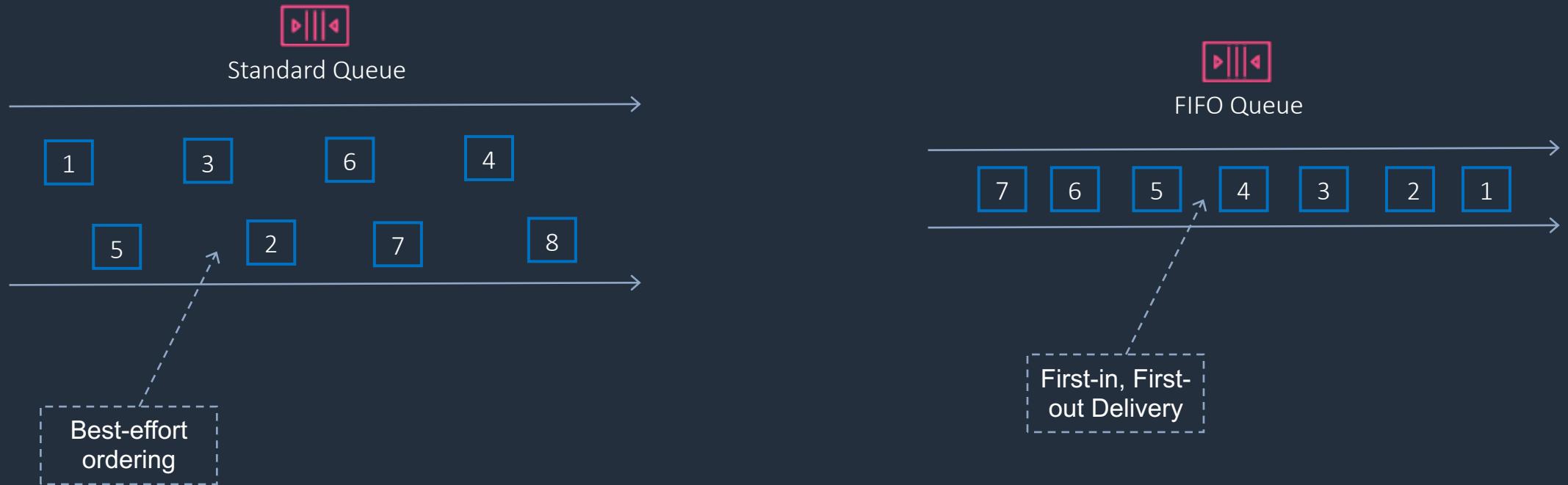
- The queue resolves issues that arise if the producer is producing work faster than the consumer can process it, or if the producer or consumer are only intermittently connected to the network.
- This is known as decoupling / loose coupling.
- Helps enable elasticity for your application.



## Amazon Simple Queue Service (SQS)

- Amazon SQS is pull-based, not push-based (like SNS).
- Messages are up to 256KB in size.
- Messages can be kept in the queue from 1 minute to 14 days.
- Default retention period is 4 days.
- Amazon SQS guarantees that your messages will be processed at least once.

# Amazon SQS – Queue Types



# Amazon SQS – Queue Types

Standard Queue	FIFO Queue
<b>Unlimited Throughput:</b> Standard queues support a nearly unlimited number of transactions per second (TPS) per API action.	<b>High Throughput:</b> FIFO queues support up to 300 messages per second (300 send, receive, or delete operations per second). When you batch 10 messages per operation (maximum), FIFO queues can support up to 3,000 messages per second
<b>Best-Effort Ordering:</b> Occasionally, messages might be delivered in an order different from which they were sent	<b>First-In-First-out Delivery:</b> The order in which messages are sent and received is strictly preserved
<b>At-Least-Once Delivery:</b> A message is delivered at least once, but occasionally more than one copy of a message is delivered	<b>Exactly-Once Processing:</b> A message is delivered once and remains available until a consumer processes and deletes it. Duplicates are not introduced into the queue

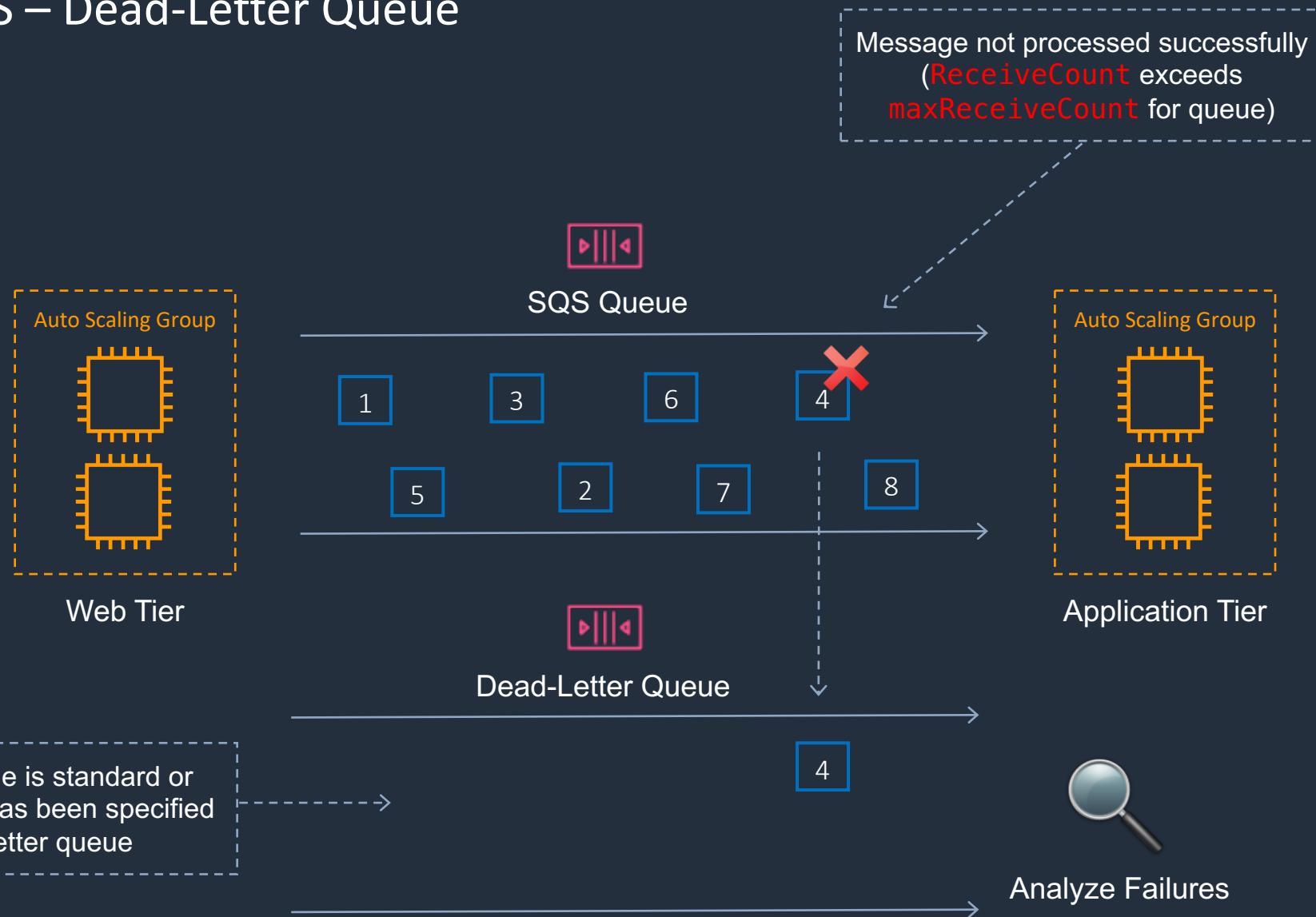
## Amazon SQS – Queue Types

- Deduplication with FIFO queues:
  - Provide a **MessageDeduplicationId** with the message.
  - The de-duplication interval is 5 minutes.
  - Content based duplication – the **MessageDeduplicationId** is generated as the SHA-256 with the message body.
- Sequencing with FIFO queues:
  - To ensure strict ordering between messages, specify a **MessageGroupId**.
  - Messages with a different Group ID may be received out of order.
  - Messages with the same Group ID are delivered to one consumer at a time.

## Amazon SQS – Queue Types

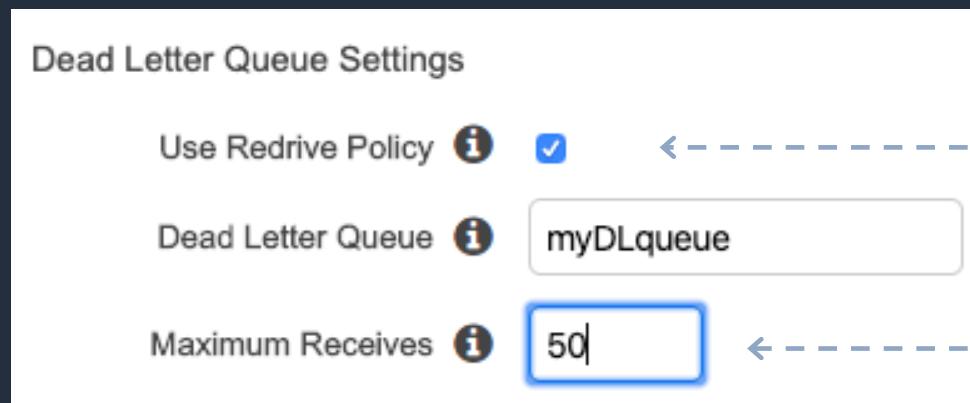
- FIFO queues require the Message Group ID and Message Deduplication ID parameters to be added to messages.
- Message Group ID:
  - The tag that specifies that a message belongs to a specific message group. Messages that belong to the same message group are guaranteed to be processed in a FIFO manner.
- Message Deduplication ID:
  - The token used for deduplication of messages within the deduplication interval.

# Amazon SQS – Dead-Letter Queue



## Amazon SQS – Dead-Letter Queue

- The main task of a dead-letter queue is handling message failure.
- A dead-letter queue lets you set aside and isolate messages that can't be processed correctly to determine why their processing didn't succeed.
- It is not a queue type, it is a standard or FIFO queue that has been specified as a dead-letter queue in the configuration of another standard or FIFO queue.



Enable Redrive Policy

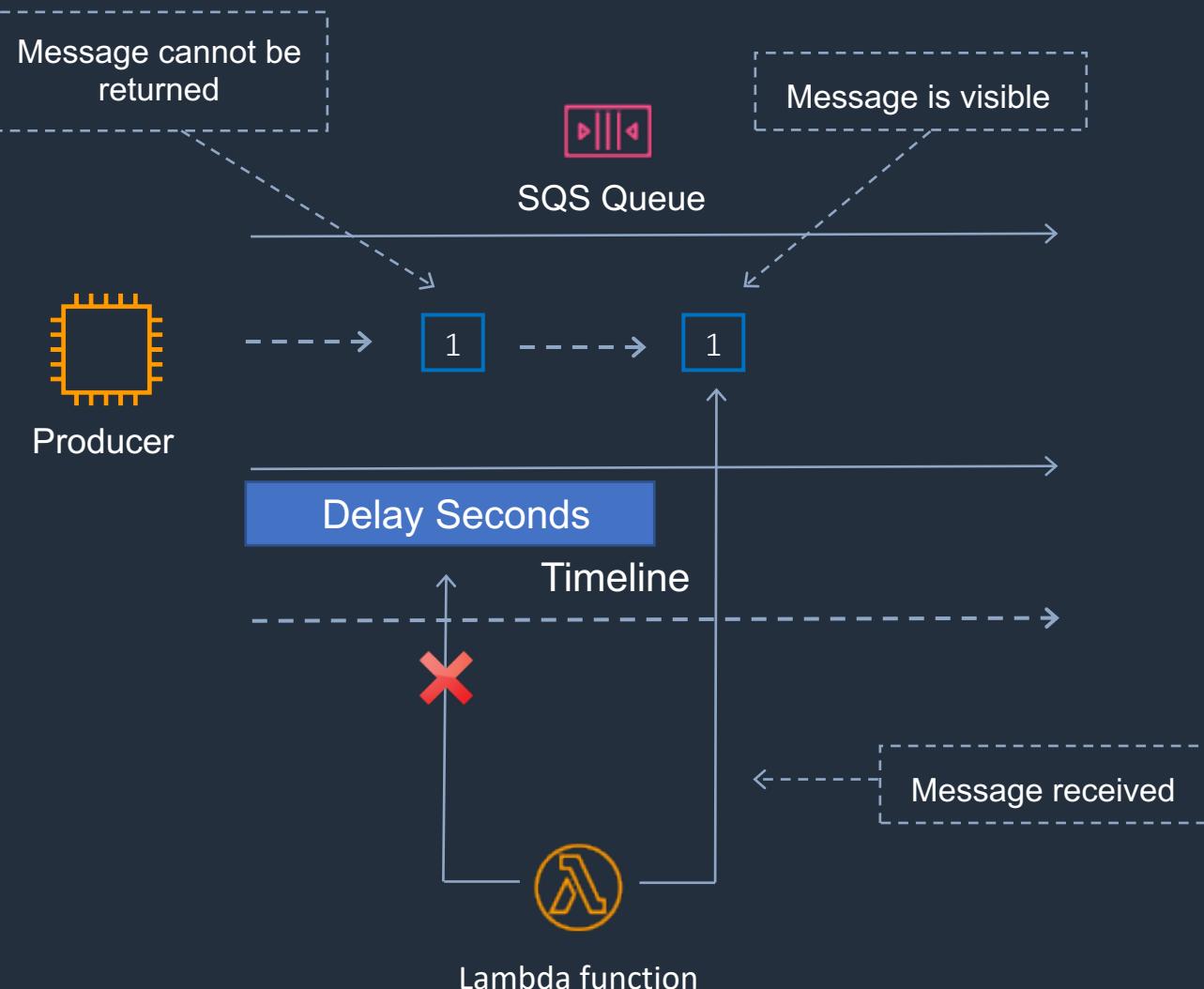
Specify the queue to use as a dead-letter queue

Specify the maximum receives before a message is sent to the dead-letter queue

## Amazon SQS – Dead-Letter Queue

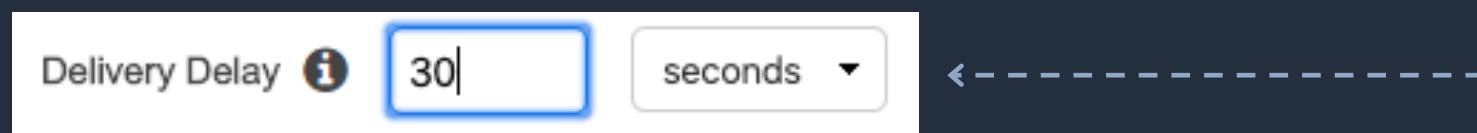
- Messages are moved to the dead-letter queue when the `ReceiveCount` for a message exceeds the `maxReceiveCount` for a queue.
- Dead-letter queues should not be used with standard queues when your application will keep retrying transmission.
- Dead-letter queues will break the order of messages in FIFO queues.

# Amazon SQS – Delay Queue



## Amazon SQS – Delay Queue

- Postpones delivery of new messages to a queue for a number of seconds.
- Messages sent to the Delay Queue remain invisible to consumers for the duration of the delay period.
- Default delay is 0 seconds, maximum is 900 seconds (15 minutes).
- For standard SQS queues, changing this setting doesn't affect the delay of messages already in the queue, only new messages.
- For FIFO queues, this affects the delay of messages already in the queue.

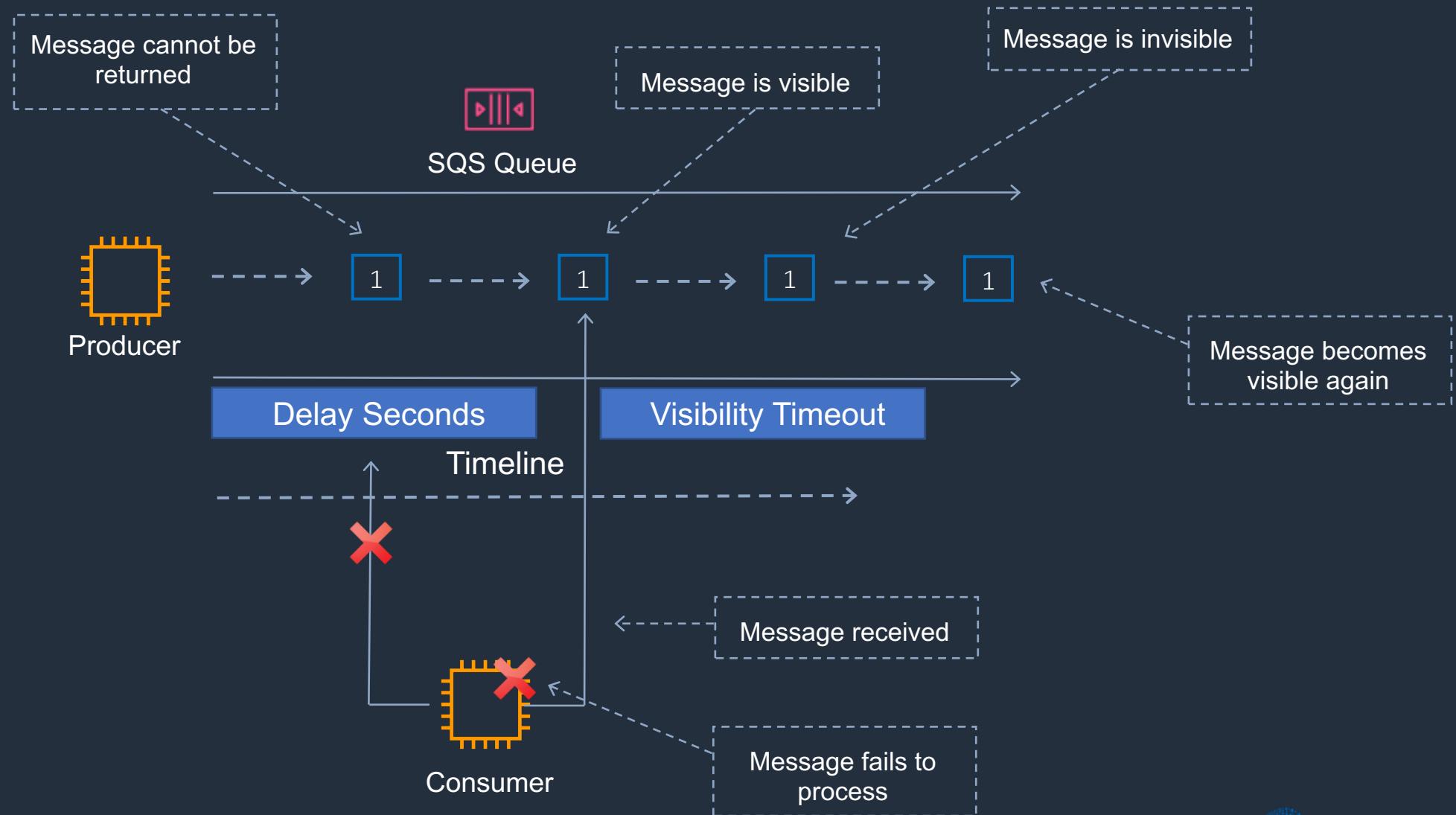


Specify time to delay first delivery of messages

## Amazon SQS – Delay Queue

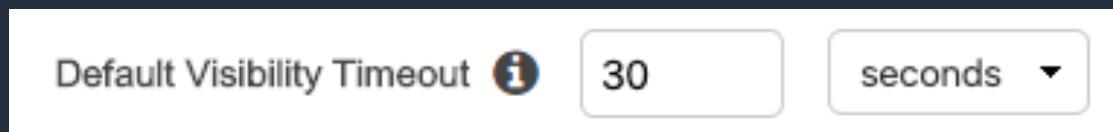
- When to use a delay queue:
  - Large distributed applications which may need to introduce a delay in processing.
  - You need to apply a delay to an entire queue of messages.
  - For example adding a delay of a few seconds to allow updates to sales or stock control databases before sending a notification to a customer confirming an online transaction.

# Amazon SQS – Visibility Timeout



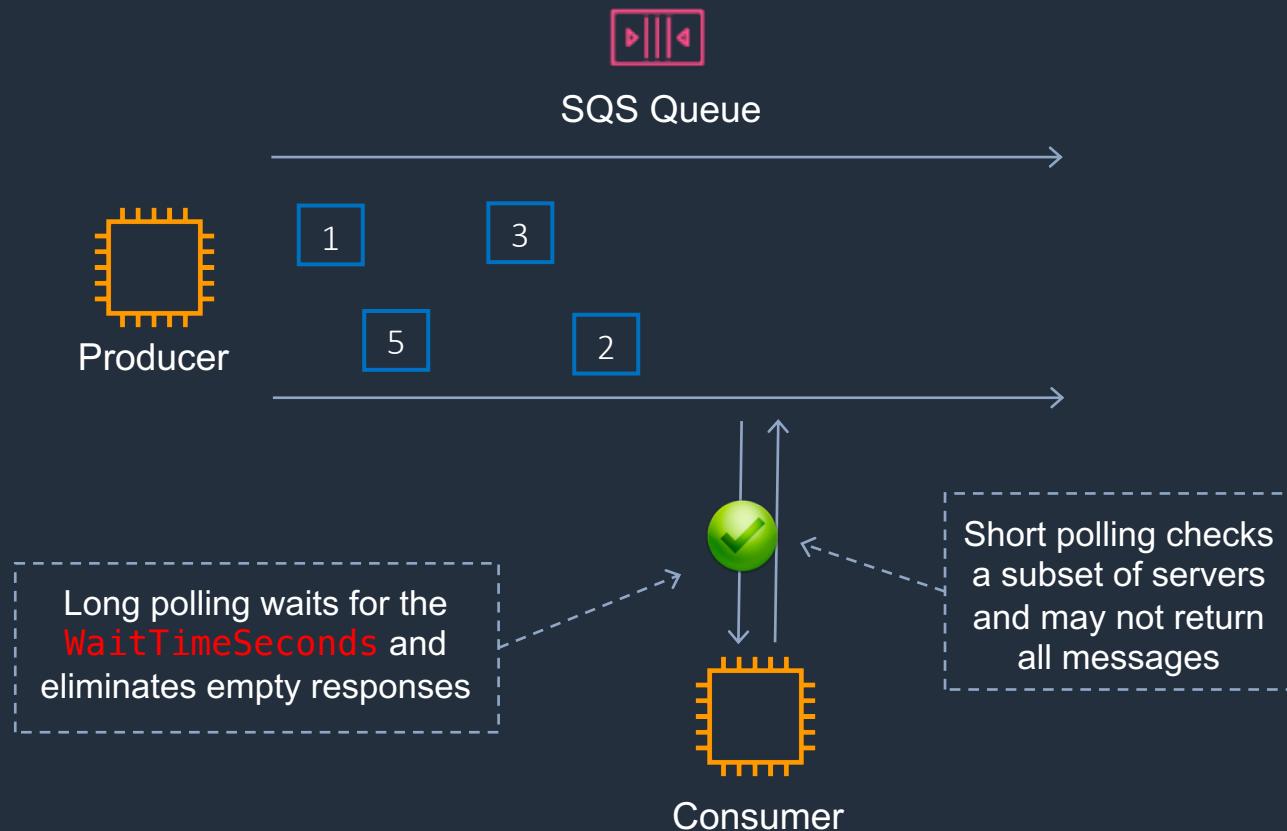
## Amazon SQS – Visibility Timeout

- The amount of time a message is invisible in the queue after a reader picks it up.
- Provided the job is processed before the visibility timeout expires, the message will then be deleted from the queue.
- If the job is not processed within the visibility timeout, the message will become visible again and another reader will process it.
- This could result in the same message being delivered twice.
- Default visibility timeout is 30 seconds.
- Increase it if your task takes >30 seconds.
- Maximum is 12 hours.



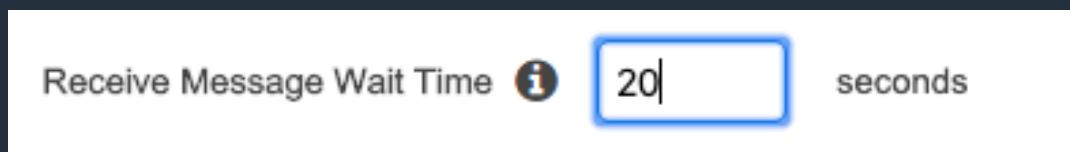
The length of time (in seconds) that a message received from a queue will be invisible to other receiving components.

# Amazon SQS – Long Polling vs Short Polling



## Amazon SQS – Long Polling

- Amazon SQS long polling is a way to retrieve messages from SQS queues.
- While the regular short polling returns immediately (even if the message queue is empty), long polling doesn't return a response until a message arrives in the message queue or the long poll times out.
- Long polling can lower costs
- Long polling can be enabled at the queue level or at the API level using **WaitTimeSeconds**.
- Long polling is in effect when the Receive Message Wait Time is a value greater than 0 seconds and up to 20 seconds.



←-----

The maximum amount of time that a long polling receive call will wait for a message to become available before returning an empty response.

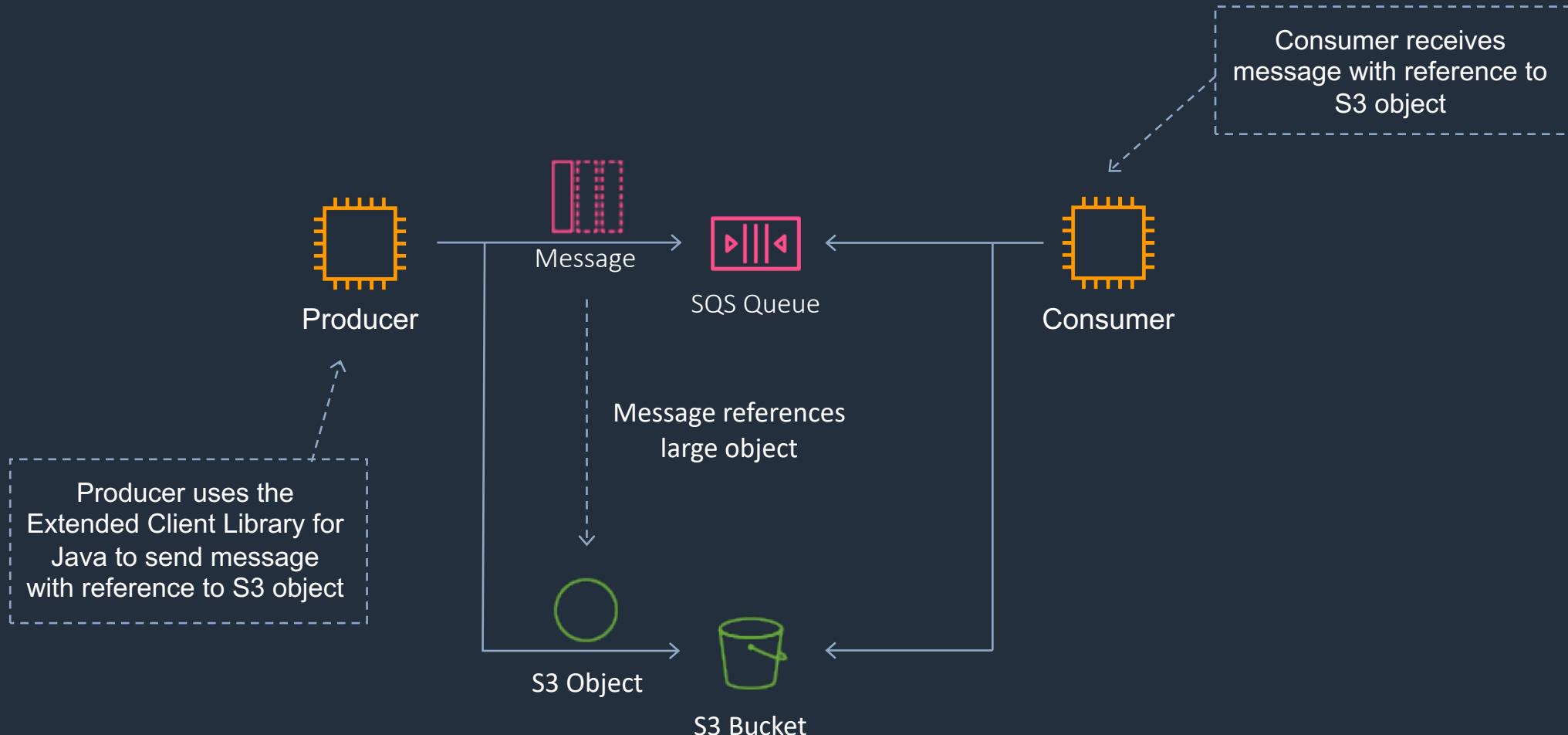
## Amazon SQS – API

- **ChangeMessageVisibility**
  - Changes the visibility timeout of a specified message in a queue to a new value.
  - Default is 30 seconds; minimum is 0 seconds; maximum is 12 hours.
- **GetQueueAttributes** and **SetQueueAttributes**
  - Gets/sets attributes for the specified queue.
  - Lots of possible values and recommend reviewing them in the AWS API reference.
  - Key attributes for the exam:
  - **DelaySeconds** – Configures a delay queue. 0/s to 900/s (default 0/s).
  - **ReceiveMessageWaitTimeSeconds** – Sets short/long polling. 0/s to 20/s (default 0s).
  - **VisibilityTimeout** - The visibility timeout for the queue. 0/ to 43,200/s (12 hours)  
(default 30/s)

## Amazon SQS – API

- **ReceiveMessage**
  - Retrieves one or more messages (up to 10), from the specified queue.
  - Using the **WaitTimeSeconds** parameter enables long-poll support.
- **SendMessage**
  - **DelaySeconds** parameter delays a message.
  - **MessageDuplicationId** parameter adds a deduplication ID (FIFO only).
  - **MessageGroupId** parameter adds a tag for a message group (FIFO only).

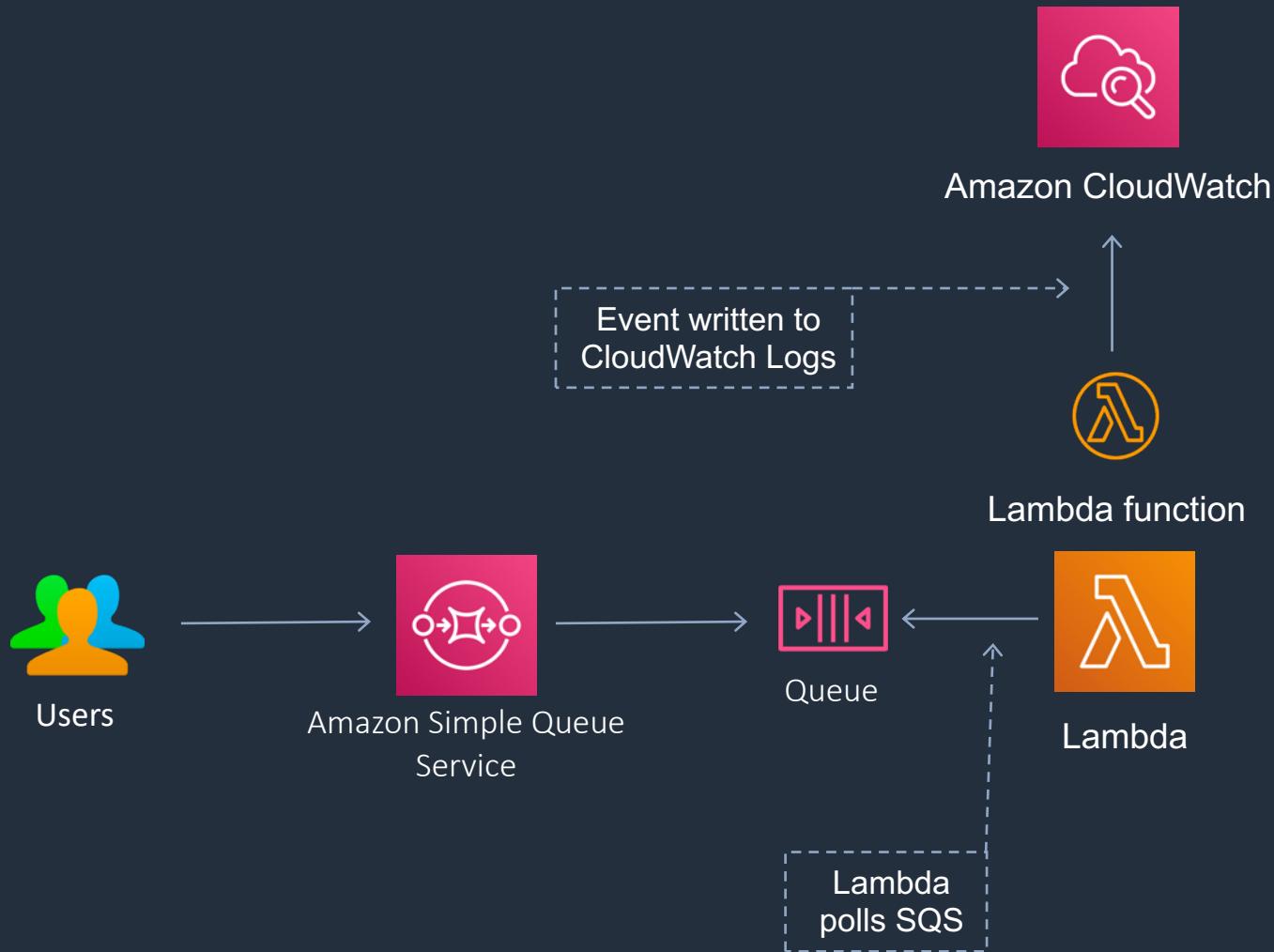
# Amazon SQS – Extended Client Library



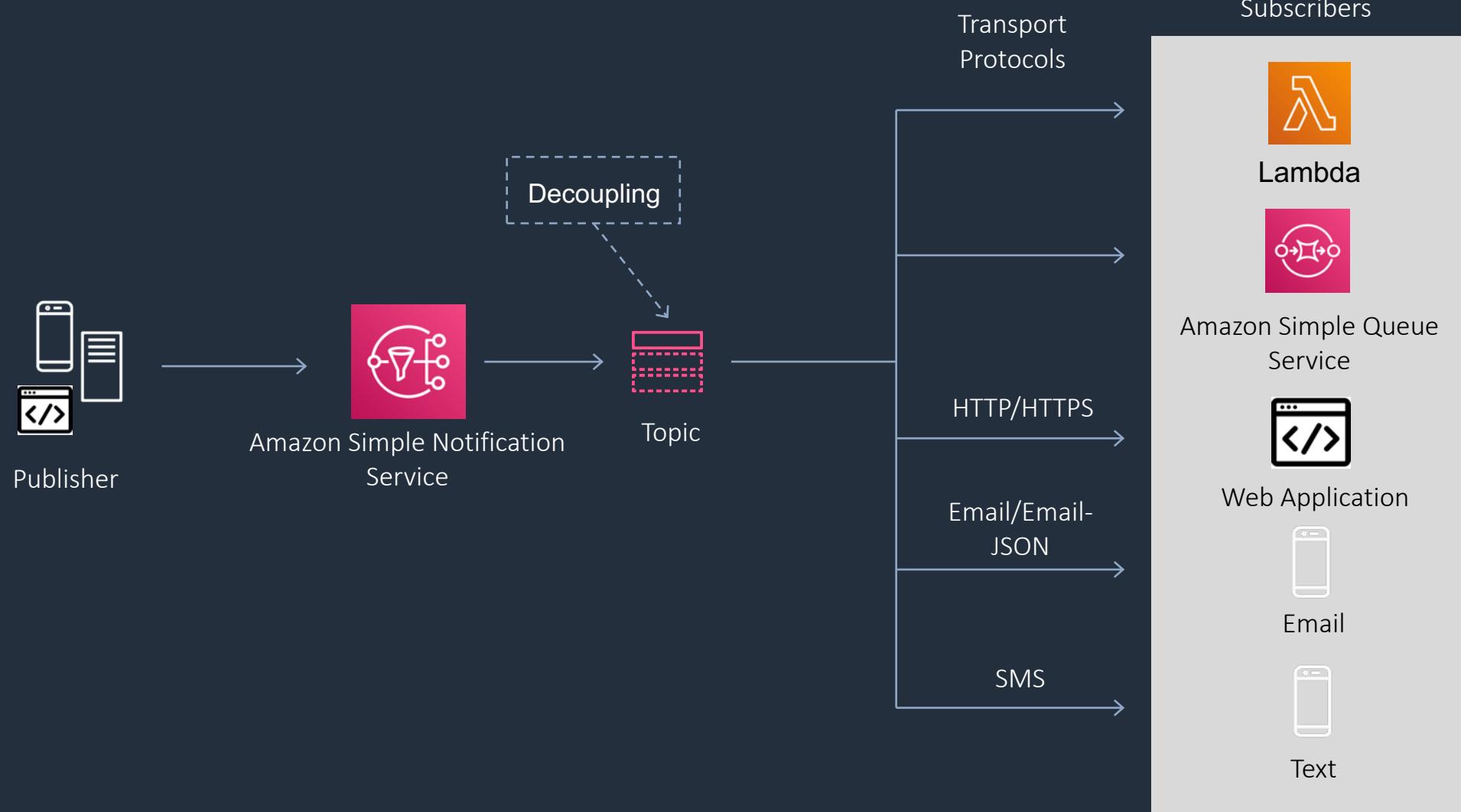
## Amazon SQS – Extended Client Library

- Maximum messages size in SQS is 256 KB.
- You can use Amazon S3 and the Amazon SQS Extended Client Library for Java to manage Amazon SQS messages.
- Useful for storing and consuming messages up to 2 GB in size.
- You can use the Amazon SQS Extended Client Library for Java library to do the following:
  - Specify whether messages are always stored in Amazon S3 or only when the size of a message exceeds 256 KB.
  - Send a message that references a single message object stored in an Amazon S3 bucket.
  - Get the corresponding message object from an Amazon S3 bucket.
  - Delete the corresponding message object from an Amazon S3 bucket.

# Trigger AWS Lambda from Amazon SQS Queue



# Amazon Simple Notification Service (SNS)



## Amazon Simple Notification Service (SNS) Overview

- Amazon Simple Notification Service (SNS) is a highly available, durable, secure, fully managed pub/sub messaging service.
- Amazon SNS provides topics for high-throughput, push-based, many-to-many messaging.
- Publisher systems can fan out messages to a large number of subscriber endpoints:
- Endpoints include:
  - Amazon SQS queues.
  - AWS Lambda functions.
  - HTTP/S webhooks.
  - Mobile push.
  - SMS.
  - Email.

## Amazon Simple Notification Service (SNS) Overview

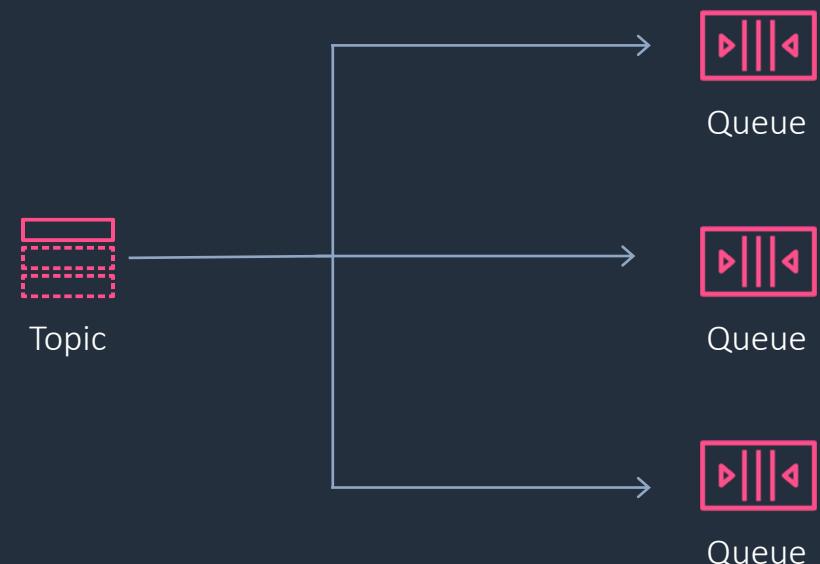
- Multiple recipients can be grouped using Topics.
- A topic is an “access point” for allowing recipients to dynamically subscribe for identical copies of the same notification.
- One topic can support deliveries to multiple endpoint types.
- All messages are stored redundantly across multiple availability zones.
- Instantaneous, push-based delivery.
- Simple APIs and easy integration with applications.
- Flexible message delivery over multiple transport protocols.

# Amazon Simple Notification Service (SNS) Overview

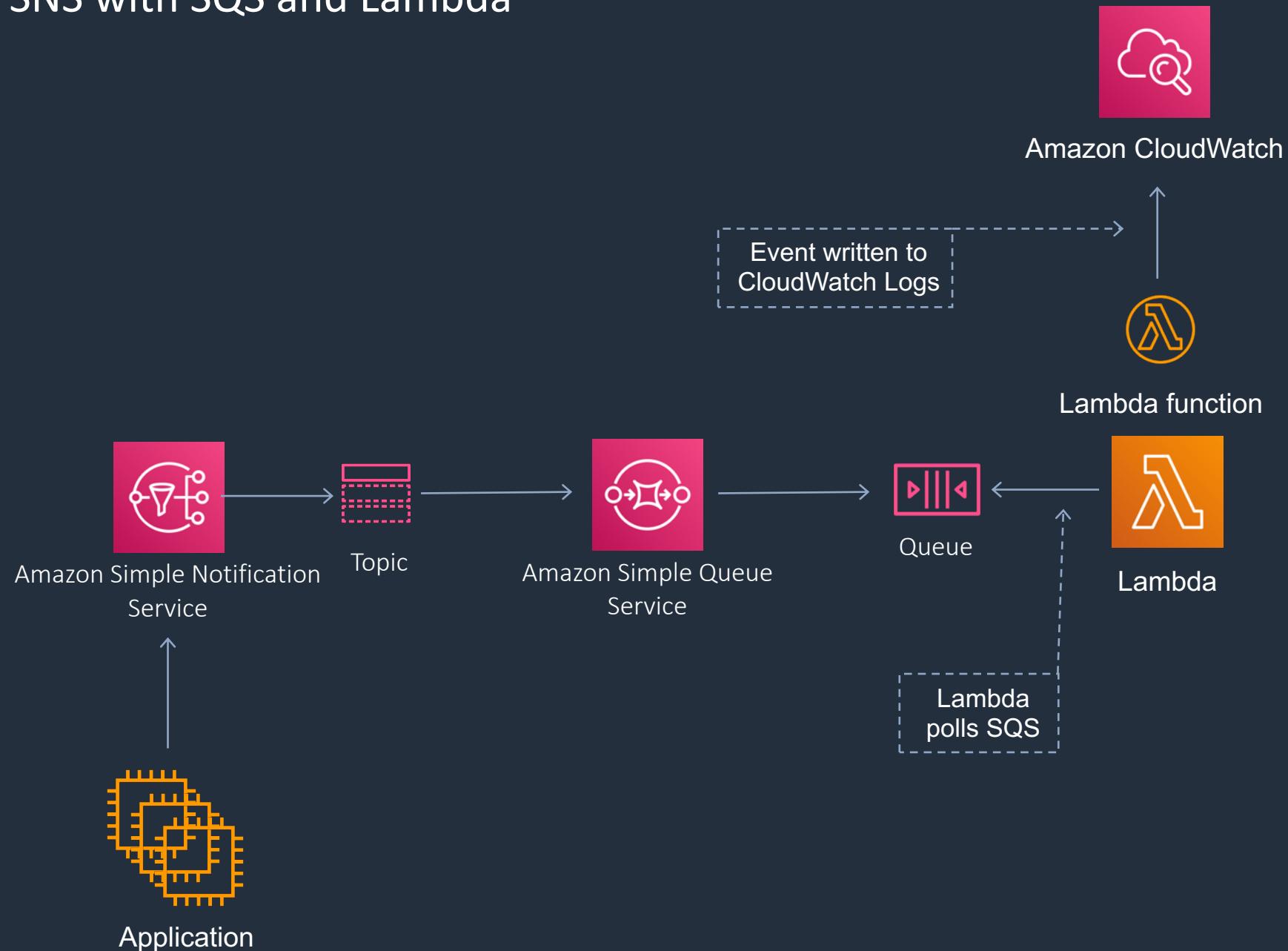
- Sending messages to many receivers:
  - The event producer sends one message to one SNS topic.
  - Many event receivers (subscribers) listen for notifications.
  - Each subscriber will get all the messages.
  - Subscribers can filter messages.
  - Up to 10,000,000 subscriptions per topic.
  - Limit of 100,000 topics.

## Amazon SNS + Amazon SQS Fan-Out

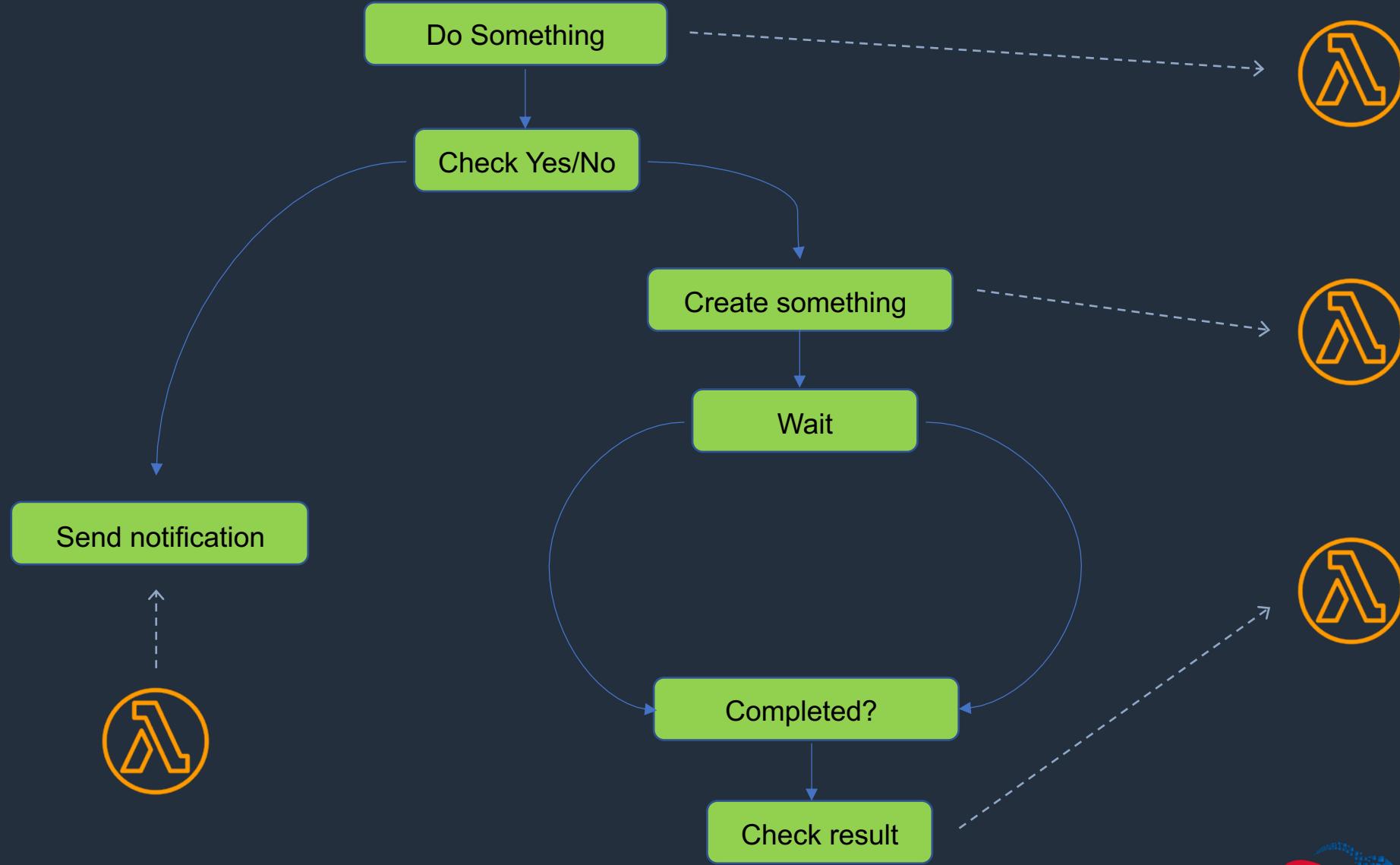
- You can subscribe one or more Amazon SQS queues to an Amazon SNS topic from a list of topics available for the selected queue.
- Amazon SQS manages the subscription and any necessary permissions.
- When you publish a message to a topic, Amazon SNS sends the message to every subscribed queue.



# Amazon SNS with SQS and Lambda



# AWS Step Functions



## AWS Step Functions - Overview

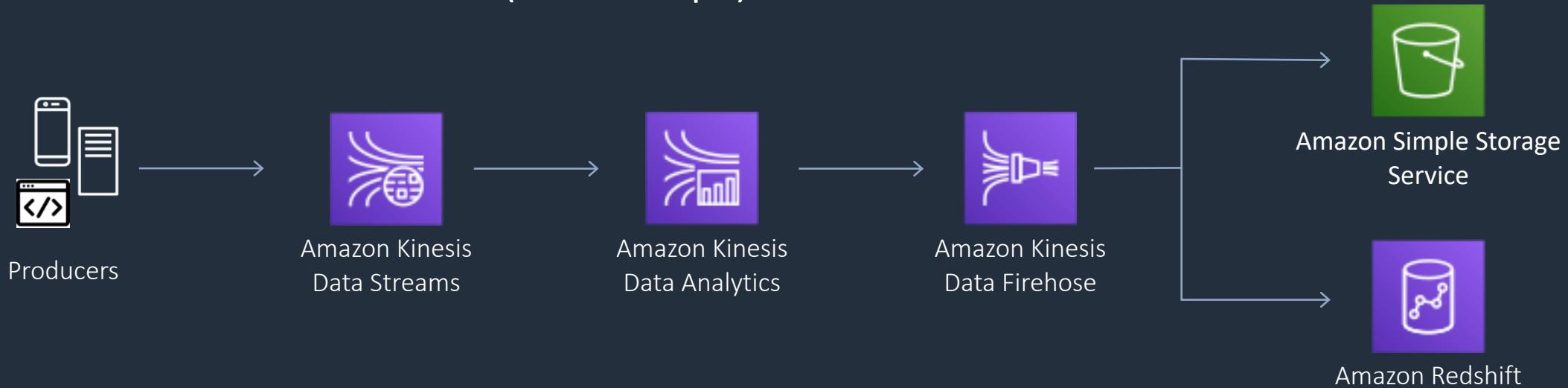
- AWS Step Functions makes it easy to coordinate the components of distributed applications as a series of steps in a visual workflow.
- You can quickly build and run state machines to execute the steps of your application in a reliable and scalable fashion.
- How it works:
  - Define the steps of your workflow in the JSON-based Amazon States Language. The visual console automatically graphs each step in the order of execution.
  - Start an execution to visualize and verify the steps of your application are operating as intended. The console highlights the real-time status of each step and provides a detailed history of every execution.
  - AWS Step Functions operates and scales the steps of your application and underlying compute for you to help ensure your application executes reliably under increasing demand.

## AWS Step Functions - Overview

- Uses the Amazon States Language declarative JSON.
- Apps can interact and update the stream via Step Function API.
- Visual interface describes flow and real-time status.
- Detailed logs of each step execution.
- Uses AWS Lambda functions.
- You can create sequential, branching or parallel steps.
- Step Functions automatically triggers and tracks each step.
- Logs the state of each step so if something goes wrong you can track what went wrong and where.

# Amazon Kinesis - Overview

- Amazon Kinesis makes it easy to collect, process, and analyze real-time, streaming data so you can get timely insights and react quickly to new information.
- Collection of services for processing streams of various data.
- The Kinesis family includes Kinesis Data Streams, Kinesis Data Firehose, Kinesis Data Analytics and Kinesis Video Streams (out of scope).

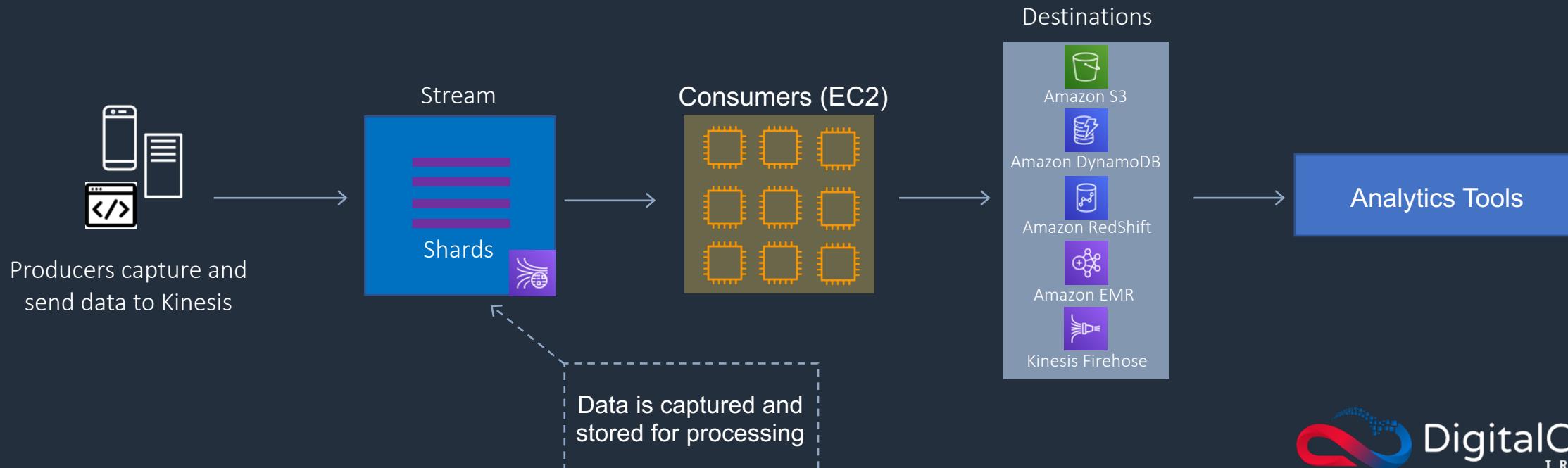


## Amazon Kinesis - Overview

- Examples of streaming data use cases include:
  - Purchases from online stores.
  - Stock prices.
  - Game data (statistics and results as the gamer plays).
  - Social network data.
  - Geospatial data (think uber.com).
  - IoT sensor data.

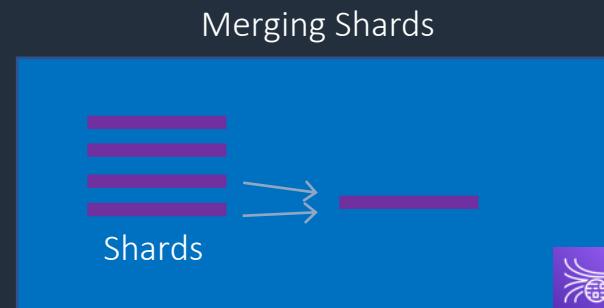
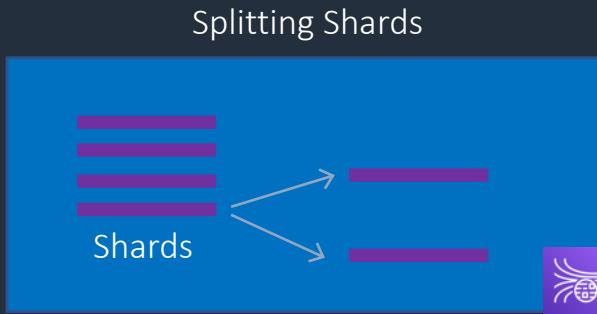
# Amazon Kinesis Data Streams

- Producers send data to Kinesis, data is stored in Shards for 24 hours (by default, up to 7 days).
- Consumers then take the data and process it - data can then be saved into another AWS service.
- One shard provides a capacity of 1MB/sec data input and 2MB/sec data output.
- One shard can support up to 1000 PUT records per second.



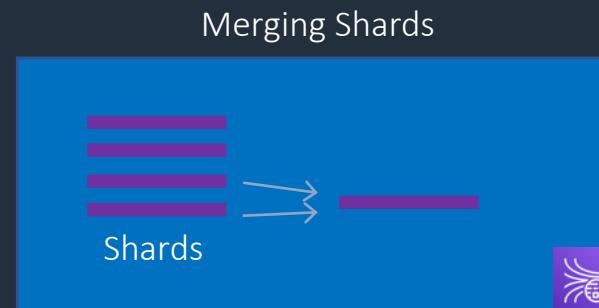
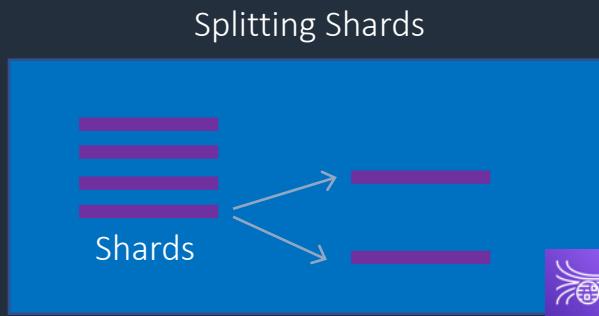
# Amazon Kinesis Data Streams

- The total capacity of the stream is the sum of the capacities of its shards.
- Kinesis Data Streams supports resharding, which lets you adjust the number of shards in your stream to adapt to changes in the rate of data flow through the stream.
- There are two types of resharding operations: shard split and shard merge.
  - In a shard split, you divide a single shard into two shards.
  - In a shard merge, you combine two shards into a single shard.



# Amazon Kinesis Data Streams

- Splitting increases the number of shards in your stream and therefore increases the data capacity of the stream.
- Splitting increases the cost of your stream (you pay per-shard).
- Merging reduces the number of shards in your stream and therefore decreases the data capacity—and cost—of the stream.



# Amazon Kinesis Data Firehose

- Producers send data to Firehose.
- There are no Shards, completely automated (scalability is elastic).
- Firehose data is sent to another AWS service for storing, data can be optionally processed using AWS Lambda.
- Fully managed service.
- Near real-time (60 seconds latency).

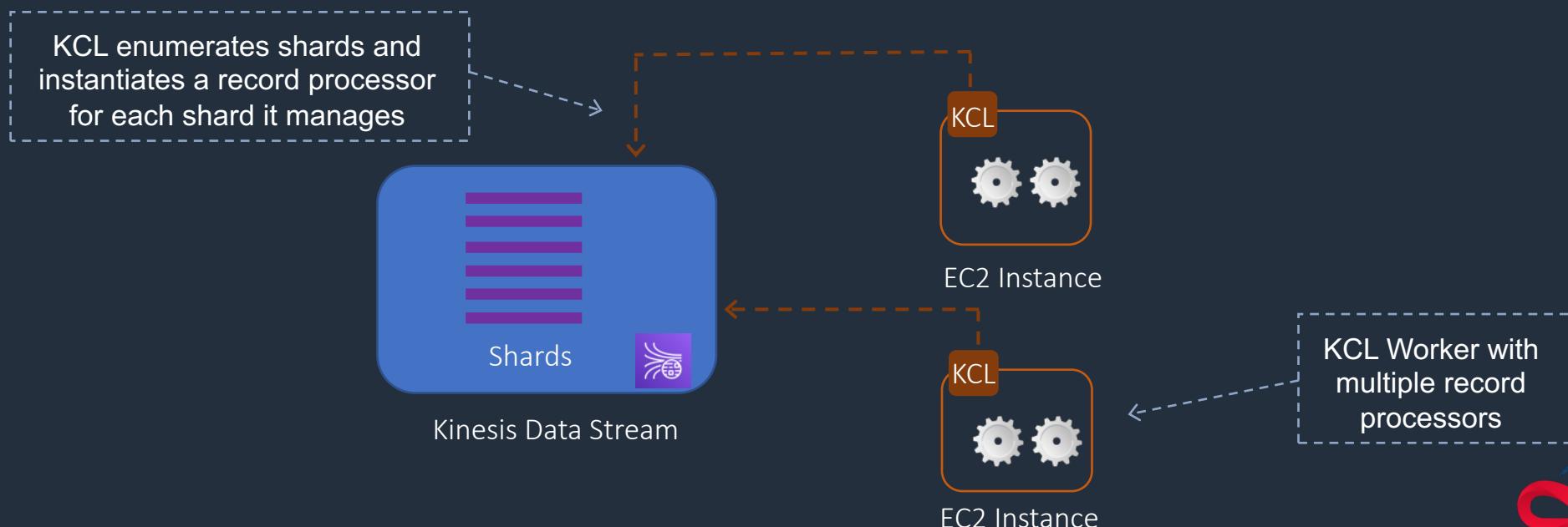


# Amazon Kinesis vs SQS and SNS

Amazon Kinesis	Amazon SQS	Amazon SNS
Consumers pull data	Consumers pull data	Push data to many subscribers
As many consumers as you need	Data is deleted after being consumed	Publisher / subscriber model
Routes related records to same record processor	Can have as many workers (consumers) as you need	Integrates with SQS for fan-out architecture pattern
Multiple applications can access stream concurrently	No ordering guarantee (except with FIFO queues)	Up to 10,000,000 subscribers
Ordering at the shard level	Provides messaging semantics	Up to 10,000,000 topics
Can consume records in correct order at later time	Individual message delay	Data is not persisted
Must provision throughput	Dynamically scales	No need to provision throughput

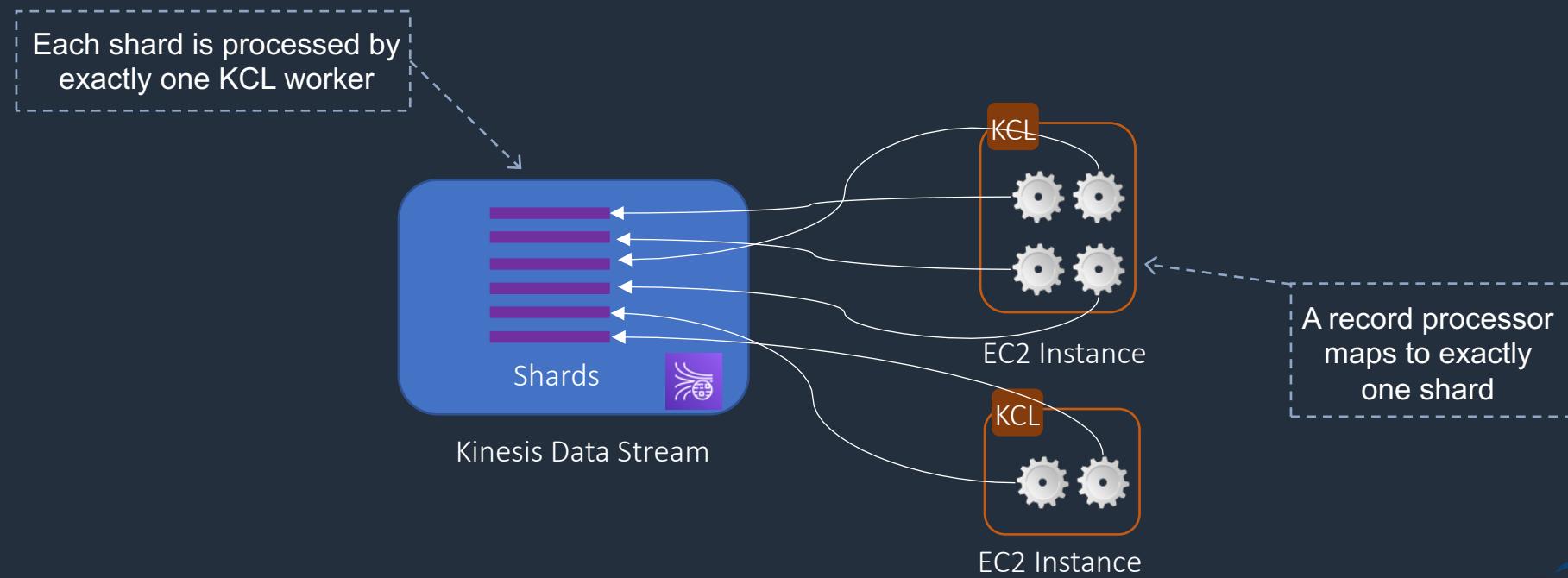
## Amazon Kinesis Client Library (KCL)

- The Kinesis Client Library (KCL) helps you consume and process data from a Kinesis data stream.
- The KCL is different from the Kinesis Data Streams API that is available in the AWS SDKs.
  - The Kinesis Data Streams API helps you manage many aspects of Kinesis Data Streams (including creating streams, resharding, and putting and getting records).
  - The KCL provides a layer of abstraction specifically for processing data in a consumer role.



## Amazon Kinesis Client Library (KCL)

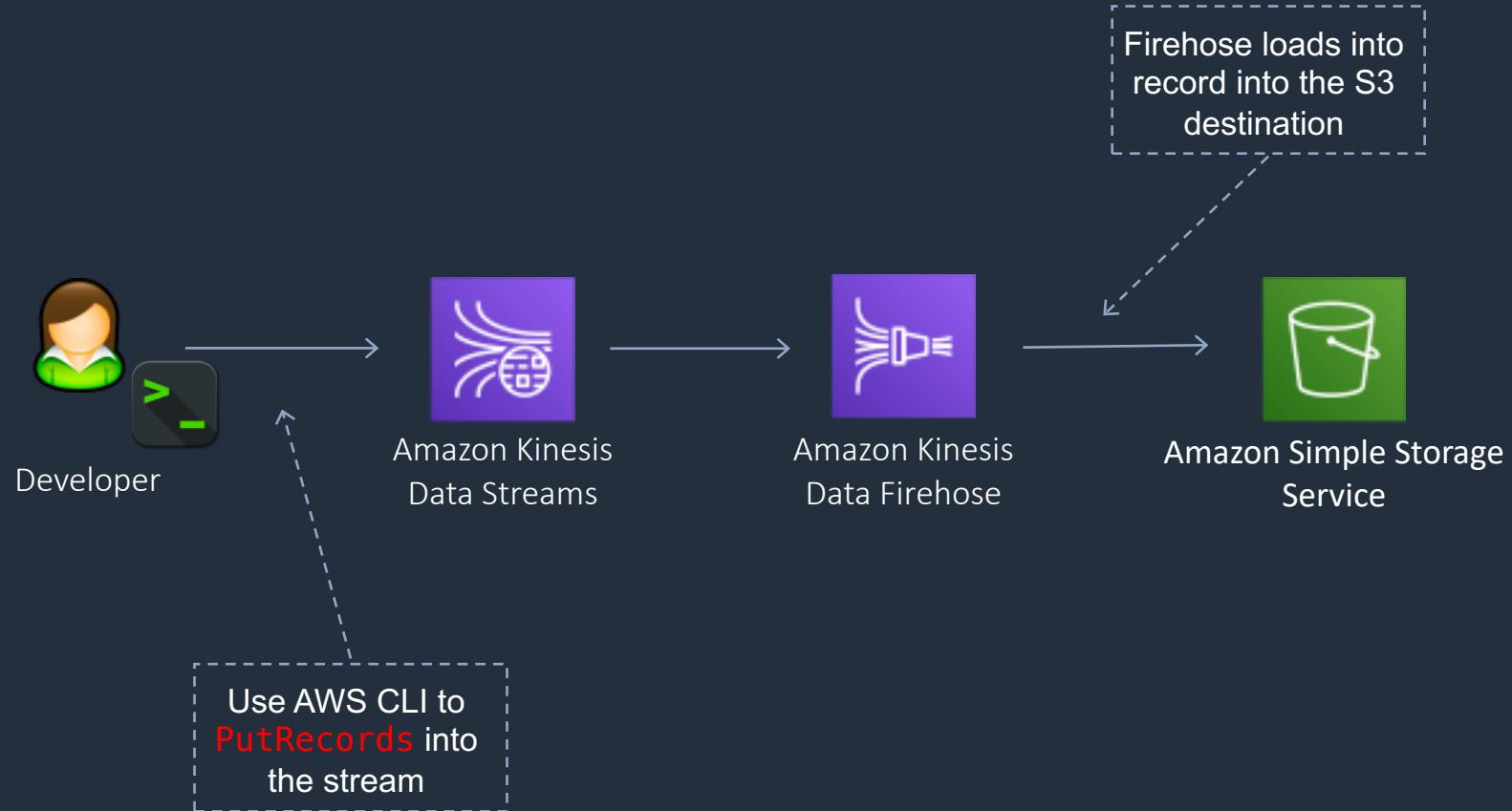
- Each shard is processed by exactly one KCL worker and has exactly one corresponding record processor, so you never need multiple instances to process one shard.
- However, one worker can process any number of shards, so it's fine if the number of shards exceeds the number of instances.



## Amazon Kinesis Client Library (KCL)

- The KCL acts as an intermediary between your record processing logic and Kinesis Data Streams.
- When you start a KCL application, it calls the KCL to instantiate a worker. The KCL performs the following tasks:
  - Connects to the stream
  - Enumerates the shards
  - Coordinates shard associations with other workers (if any)
  - Instantiates a record processor for every shard it manages
  - Pulls data records from the stream
  - Pushes the records to the corresponding record processor
  - Checkpoints processed records
  - Balances shard-worker associations when the worker instance count changes
  - Balances shard-worker associations when shards are split or merged

# Amazon Kinesis Data Streams with Firehose and S3



# SECTION 19

## Monitoring, Logging and Auditing: CloudWatch, CloudTrail and X-Ray

# Amazon CloudWatch – Examples of Functionality



## Amazon CloudWatch Overview

- Amazon CloudWatch monitors AWS resources and applications in real-time.
- CloudWatch collects and tracks metrics.
- Metrics are data points that are published to CloudWatch.
- CloudWatch alarms monitor metrics and automatically initiate actions.
- CloudWatch Logs centralizes logs from systems, applications and AWS services.
- CloudWatch Events delivers a stream of system events that describe changes in AWS resources.

# Amazon CloudWatch – Key Terminology and Concepts

- Namespace:
  - A namespace is a container for CloudWatch metrics.
  - Metrics in different namespaces are isolated from each other, so that metrics from different applications are not mistakenly aggregated into the same statistics.

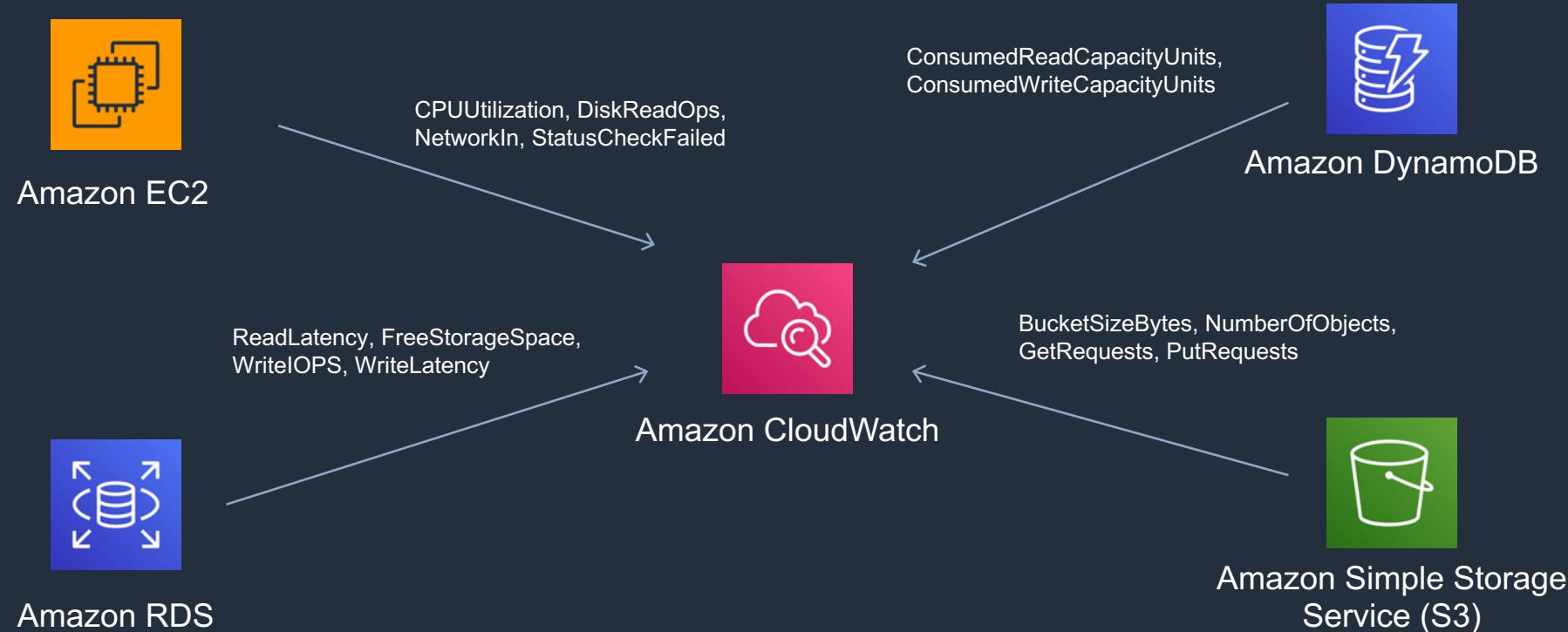
# Amazon CloudWatch – Namespaces

Service	Namespace
Amazon API Gateway	AWS/ApiGateway
Amazon CloudFront	AWS/CloudFront
AWS CloudHSM	AWS/CloudHSM
Amazon CloudWatch Logs	AWS/Logs
AWS CodeBuild	AWS/CodeBuild
Amazon Cognito	AWS/Cognito
Amazon DynamoDB	AWS/DynamoDB
Amazon EC2	AWS/EC2
AWS Elastic Beanstalk	AWS/ElasticBeanstalk

# Amazon CloudWatch – Key Terminology and Concepts

- Metrics:
  - Metrics are the fundamental concept in CloudWatch.
  - A metric represents a time-ordered set of data points that are published to CloudWatch.
  - AWS services send metrics to CloudWatch.
  - You can also send your own custom metrics to CloudWatch.
  - Metrics exist within a region.
  - Metrics cannot be deleted but automatically expire after 15 months.
  - Metrics are uniquely defined by a name, a namespace, and zero or more dimensions.

# Amazon CloudWatch - Metrics

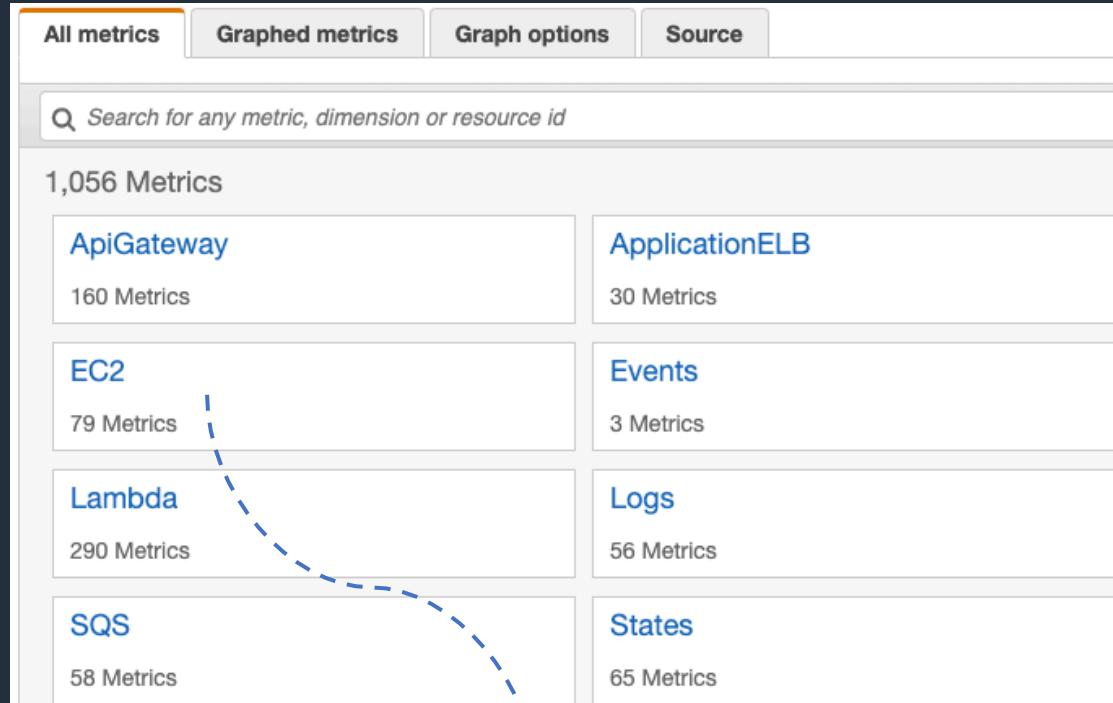


# Amazon CloudWatch – Key Terminology and Concepts

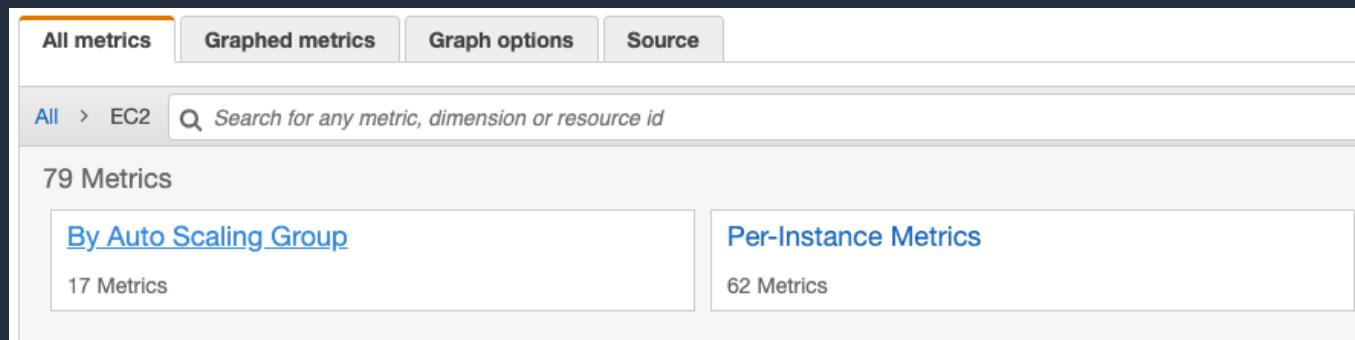
- Dimensions:
  - A dimension is a name/value pair that is part of the identity of a metric.
  - You can assign up to 10 dimensions to a metric.
  - Dimensions are categories for the characteristics of each metric.

# Amazon CloudWatch – Dimensions

These are namespaces



These are dimensions



# Amazon CloudWatch – Key Terminology and Concepts

- Statistics:
  - Statistics are metric data aggregations over specified periods of time.
  - CloudWatch provides statistics based on the metric data points provided by your custom data or provided by other AWS services to CloudWatch.

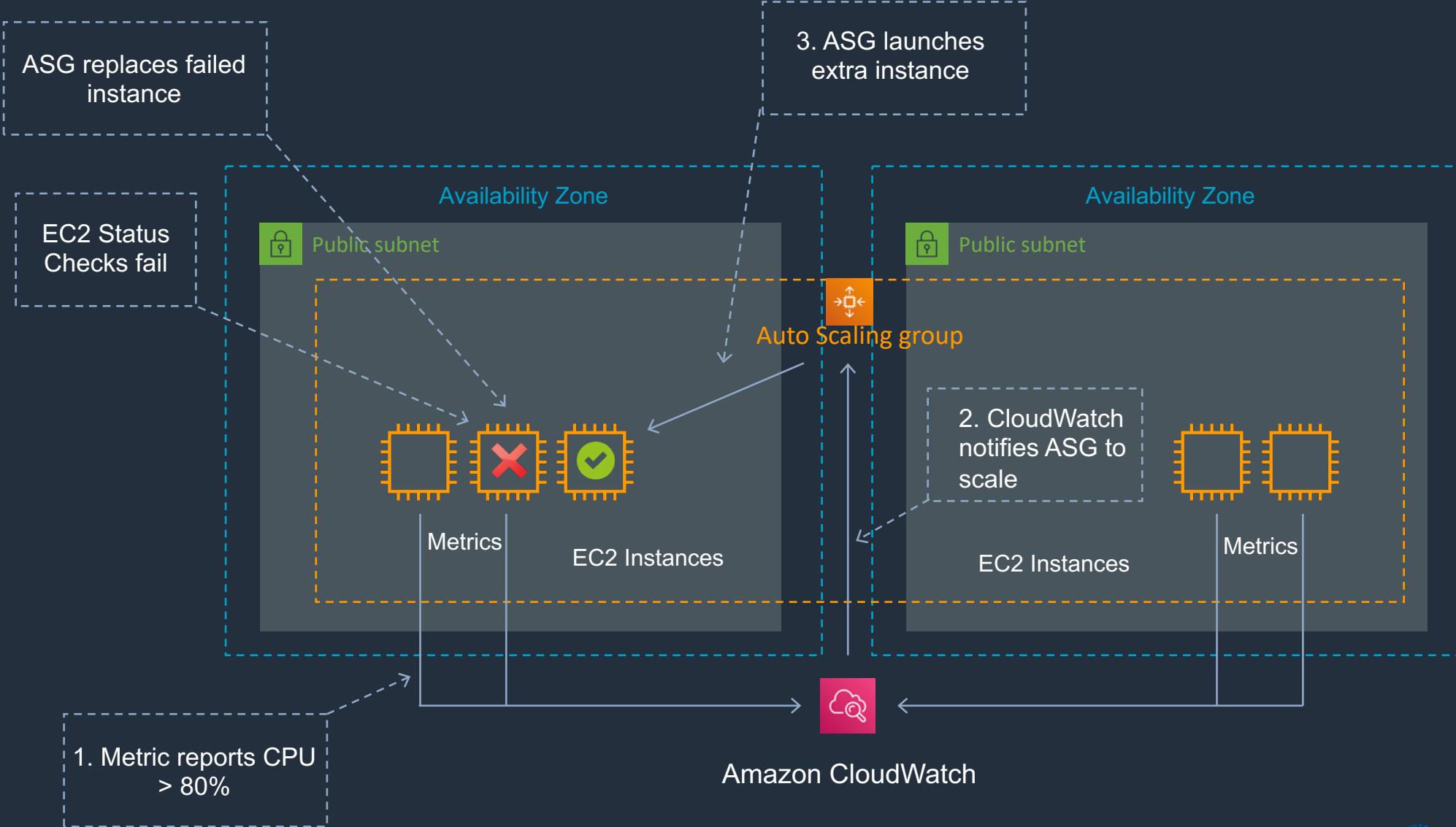
# Amazon CloudWatch – Statistics

Statistic	Description
Minimum	The lowest value observed during the specified period. You can use this value to determine low volumes of activity for your application.
Maximum	The highest value observed during the specified period. You can use this value to determine high volumes of activity for your application.
Sum	All values submitted for the matching metric added together. This statistic can be useful for determining the total volume of a metric.
Average	The value of Sum / SampleCount during the specified period. By comparing this statistic with the Minimum and Maximum, you can determine the full scope of a metric and how close the average use is to the Minimum and Maximum. This comparison helps you to know when to increase or decrease your resources as needed.
SampleCount	The count (number) of data points used for the statistical calculation.
pNN.NN	The value of the specified percentile. You can specify any percentile, using up to two decimal places (for example, p95.45). Percentile statistics are not available for metrics that include any negative values. For more information, see <a href="#">Percentiles</a> .

# Amazon CloudWatch – Key Terminology and Concepts

- Alarms:
  - You can use an alarm to automatically initiate actions on your behalf.
  - An alarm watches a single metric over a specified time period, and performs one or more specified actions, based on the value of the metric relative to a threshold over time.
  - The action is a notification sent to an Amazon SNS topic or an Auto Scaling policy.
  - You can also add alarms to dashboards.
  - Alarms invoke actions for sustained state changes only.
  - CloudWatch alarms do not invoke actions simply because they are in a particular state.
  - The state must have changed and been maintained for a specified number of periods.

# Auto Scaling with CloudWatch



# Amazon CloudWatch – Useful API Actions

- **GetMetricData**
  - Retrieve as many as 500 different metrics in a single request.
- **PutMetricData**
  - Publishes metric data points to Amazon CloudWatch.
  - CloudWatch associates the data points with the specified metric.
  - If the specified metric does not exist, CloudWatch creates the metric.
- **GetMetricStatistics**
  - Gets statistics for the specified metric.
  - CloudWatch aggregates data points based on the length of the period that you specify.
  - Maximum number of data points returned from a single call is 1,440.

## Amazon CloudWatch – Useful API Actions

- **PutMetricAlarm**
  - Creates or updates an alarm and associates it with the specified metric, metric math expression, or anomaly detection model.
  - Alarms based on anomaly detection models cannot have Auto Scaling actions.

## Amazon CloudWatch – Useful CLI Commands

Using dimensions with the CLI:

```
aws cloudwatch put-metric-data --metric-name Buffers --namespace MyNameSpace --unit Bytes  
--value 231434333 --dimensions InstanceId=1-23456789,InstanceType=m1.small
```

Retrieve statistics for the metric above:

```
aws cloudwatch get-metric-statistics --metric-name Buffers --namespace MyNameSpace --  
dimensions Name=InstanceId,Value=1-23456789 Name=InstanceType,Value=m1.small --start-time  
2016-10-15T04:00:00Z --end-time 2016-10-19T07:00:00Z --statistics Average --period 60
```

## Amazon CloudWatch – Useful CLI Commands

Publish a single metric:

```
aws cloudwatch put-metric-data --metric-name PageViewCount --namespace MyService --value  
2 --timestamp 2016-10-20T12:00:00.000Z
```

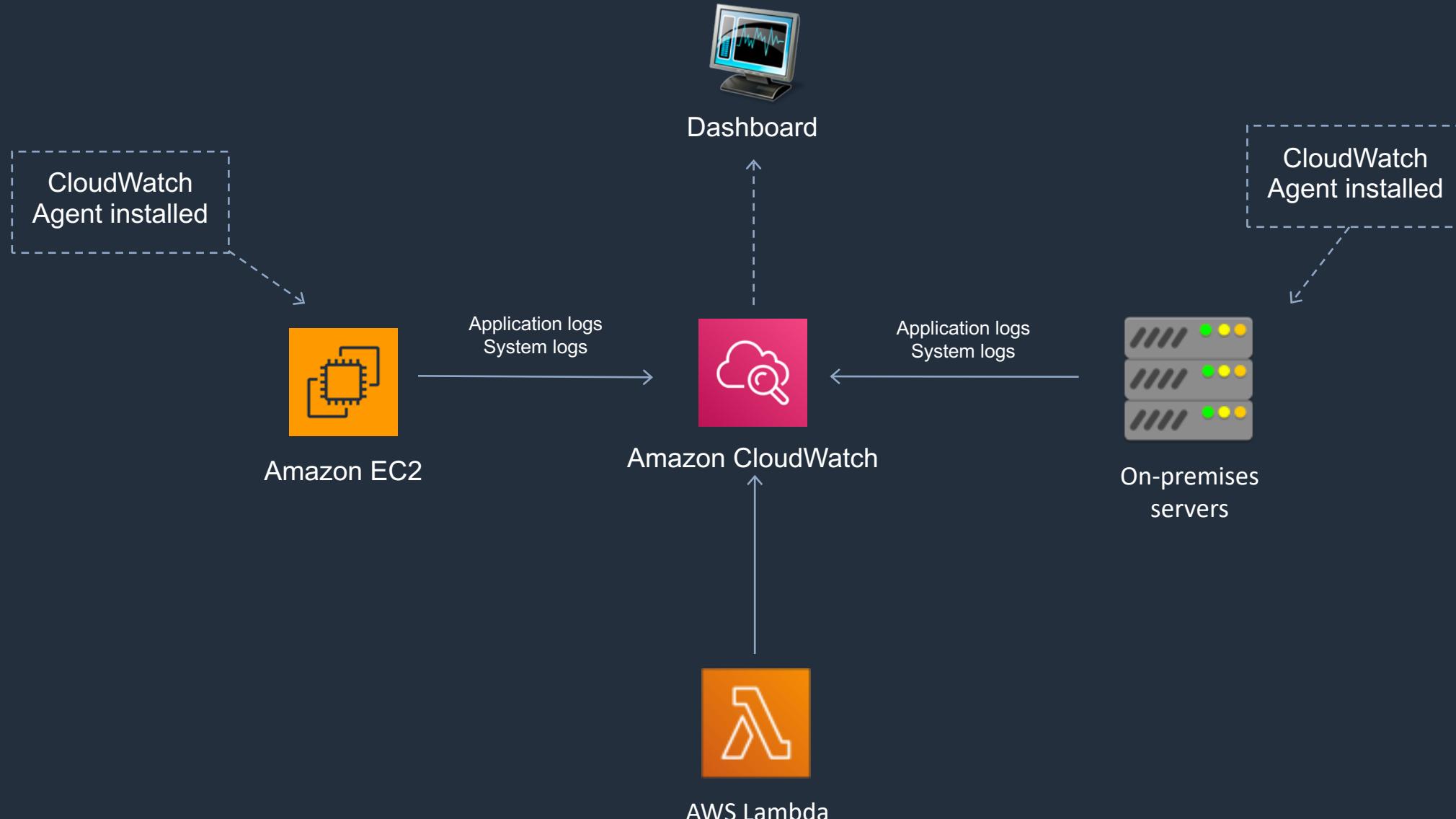
Retrieve statistics for the metric above:

```
aws cloudwatch get-metric-statistics --namespace MyService --metric-name PageViewCount  
--statistics "Sum" "Maximum" "Minimum" "Average" "SampleCount" --start-time 2016-10-  
20T12:00:00.000Z --end-time 2016-10-20T12:05:00.000Z --period 60
```

## Amazon CloudWatch – Custom Metrics and Resolution

- You can publish your own metrics to CloudWatch using the AWS CLI or an API.
- Each metric is one of the following:
  - Standard resolution, with data having a one-minute granularity.
  - High resolution, with data at a granularity of one second.
- Metrics produced by AWS services are standard resolution by default.
- When you publish a custom metric, you can define it as either standard resolution or high resolution.
- When you publish a high-resolution metric, CloudWatch stores it with a resolution of 1 second, and you can read and retrieve it with a period of 1 second, 5 seconds, 10 seconds, 30 seconds, or any multiple of 60 seconds.

# Amazon CloudWatch Logs



## Amazon CloudWatch Logs

- CloudWatch Logs enables you to centralize the logs from all of your systems, applications, and AWS services.
- Features:
  - Monitor logs from Amazon EC2 instances - monitors application and system logs and can trigger notifications.
  - Monitor CloudTrail Logged Events – alarms can be created in CloudWatch based on API activity captured by CloudTrail.
  - Log retention – by default, logs are retained indefinitely. Configurable per log group from 1 day to 10 years.

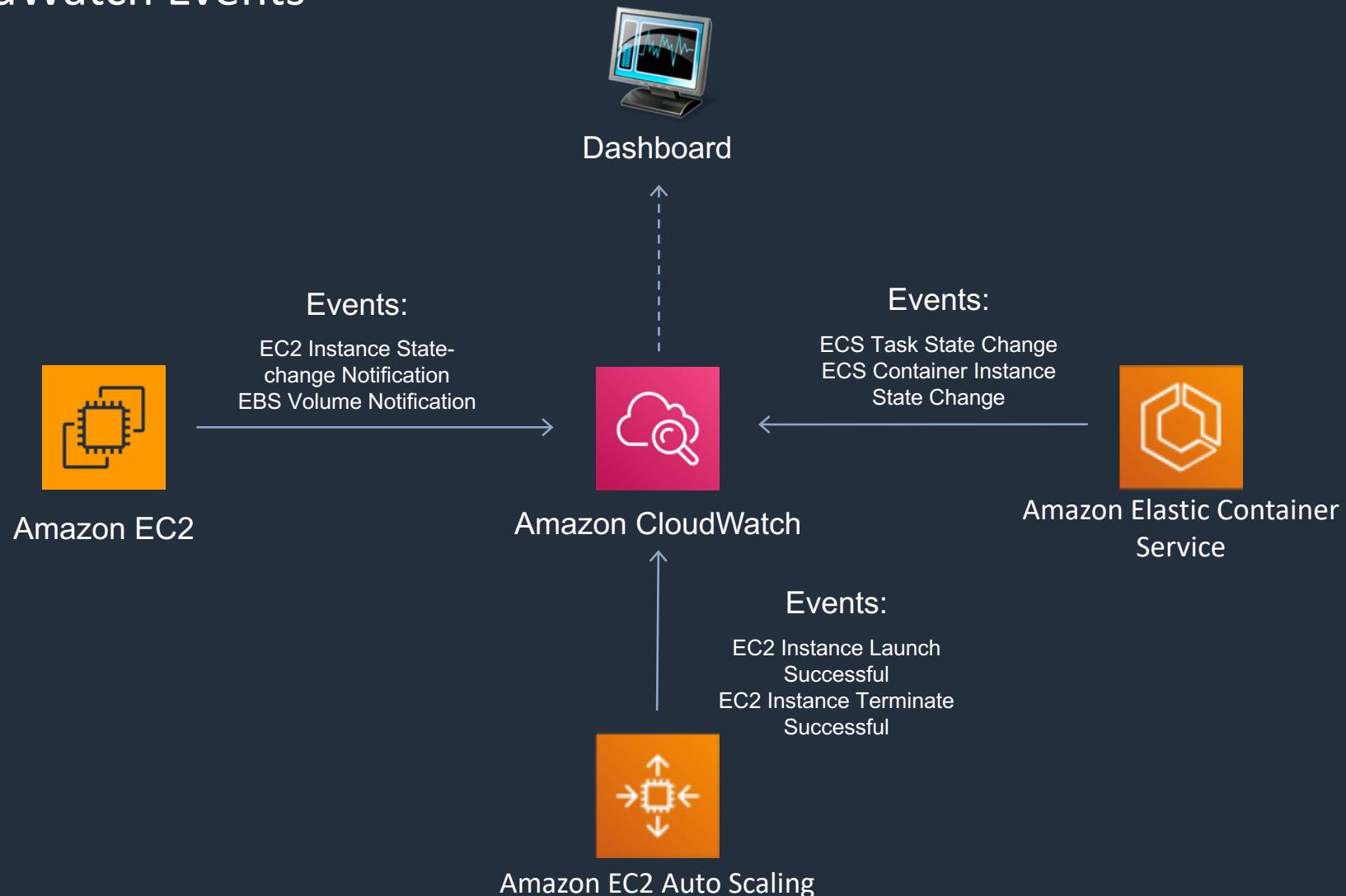
## Amazon CloudWatch Logs Agent

- The CloudWatch Logs agent provides an automated way to send log data to CloudWatch Logs from Amazon EC2 instances.
- There is now a unified CloudWatch agent that collects both logs and metrics.
- The unified CloudWatch agent includes metrics such as memory and disk utilization.

## Amazon CloudWatch Agent

- The unified CloudWatch agent enables you to do the following:
  - Collect more system-level metrics from Amazon EC2 instances across operating systems.  
The metrics can include in-guest metrics, in addition to the metrics for EC2 instances.
  - Collect system-level metrics from on-premises servers. These can include servers in a hybrid environment as well as servers not managed by AWS.
  - Retrieve custom metrics from your applications or services using the StatsD and collectd protocols.

# Amazon CloudWatch Events



## Amazon CloudWatch Events

- Amazon CloudWatch Events delivers a near real-time stream of system events that describe changes in AWS resources.
- Can use CloudWatch Events to schedule automated actions that self-trigger at certain times using cron or rate expressions
- Can match events and route them to one or more target functions or streams.

## Amazon CloudWatch Events

- Targets include:
  - Amazon EC2 instances
  - AWS Lambda functions
  - Streams in Amazon Kinesis Data Streams
  - Delivery streams in Amazon Kinesis Data Firehose
  - Log groups in Amazon CloudWatch Logs
  - Amazon ECS tasks
  - Systems Manager Run Command
  - Systems Manager Automation
  - AWS Batch jobs
  - Step Functions state machines
  - Pipelines in CodePipeline
  - CodeBuild projects
  - Amazon Inspector assessment templates
  - Amazon SNS topics
  - Amazon SQS queues

# Amazon CloudWatch Events

Specify event source:

Event Pattern i  Schedule i

**Build event pattern to match events by service**

Service Name: EC2

Event Type: EC2 Instance State-change Notification

Any state  Specific state(s)

Any instance  Specific instance Id(s)

**Event Pattern Preview**

```
{  
  "source": [  
    "aws.ec2"  
  ],  
  "detail-type": [  
    "EC2 Instance State-change Notification"  
  ]  
}
```

[Copy to clipboard](#) [Edit](#)

Specify event target:

**Targets**

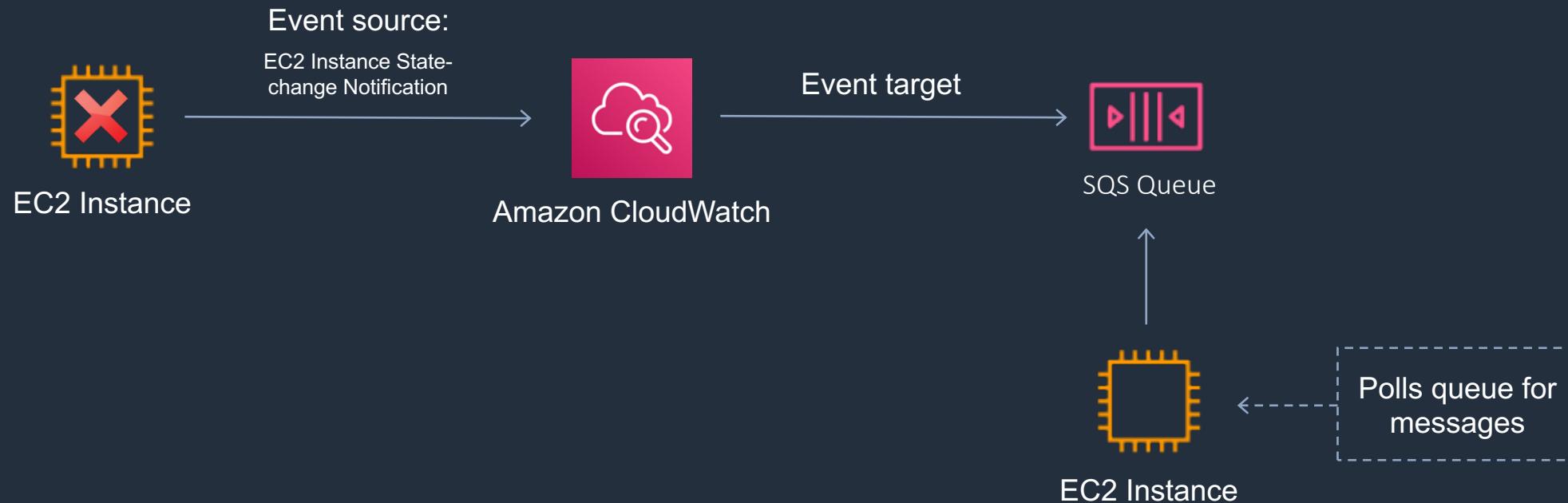
Select Target to invoke when an event matches your Event Pattern or when schedule is triggered.

**Lambda function**

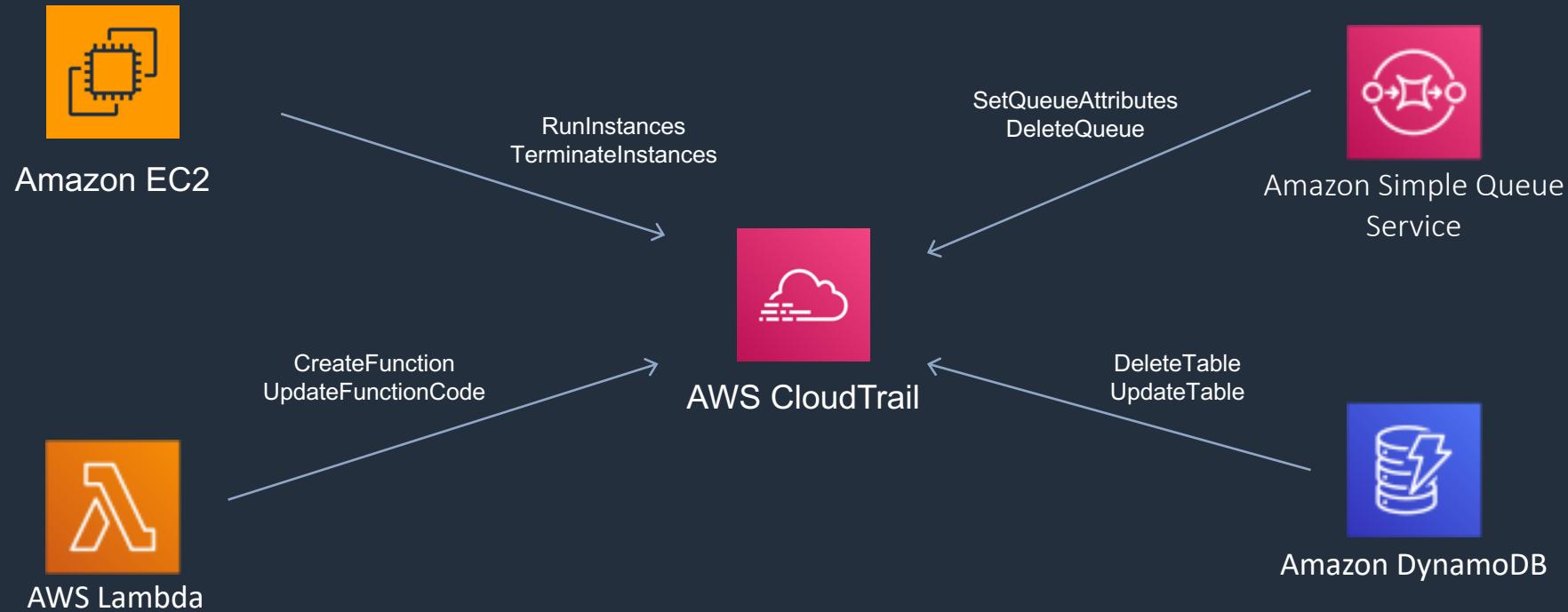
Function\* [Select function](#)

▶ Configure version/alias  
▶ Configure input

# Amazon CloudWatch Events Example



# Auditing with Amazon CloudTrail



## Amazon CloudTrail

- AWS CloudTrail is a web service that records activity made on your account.
- A CloudTrail trail can be created which delivers log files to an Amazon S3 bucket.
- Enables governance, compliance, and operational and risk auditing of your AWS account.
- Events include actions taken in the AWS Management Console, AWS Command Line Interface, and AWS SDKs and APIs.
- CloudTrail is enabled on your AWS account when you create it.
- You can create two types of trails for an AWS account:
  - A trail that applies to all regions - records events in all regions and delivers to an S3 bucket.
  - A trail that applies to a single region – records events in a single region and delivers to an S3 bucket. Additional single trails can use the same or different bucket.

## Amazon CloudTrail

- Management events provide information about management operations that are performed on resources in your AWS account. These are also known as control plane operations.
- Example management events include:
  - Configuring security (for example, IAM `AttachRolePolicy` API operations).
  - Registering devices (for example, Amazon EC2 `CreateDefaultVpc` API operations).
  - Configuring rules for routing data (for example, Amazon EC2 `CreateSubnet` API operations).
  - Setting up logging (for example, AWS CloudTrail `CreateTrail` API operations).

## Amazon CloudTrail

- Data events provide information about the resource operations performed on or in a resource. These are also known as data plane operations. Data events are often high-volume activities.

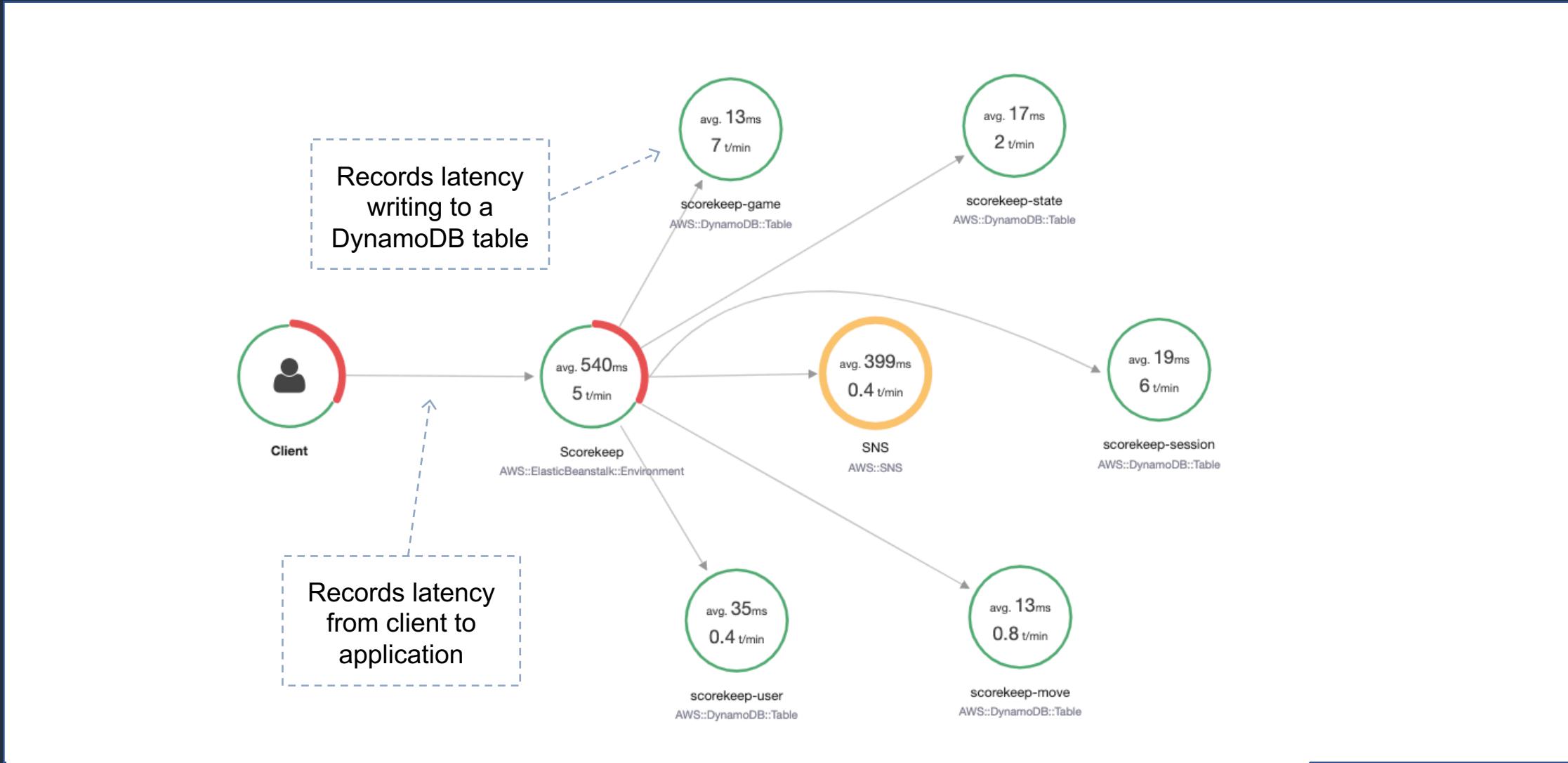
Example data events include:

- Amazon S3 object-level API activity (for example, `GetObject`, `DeleteObject`, and `PutObject` API operations).
- AWS Lambda function execution activity (the `Invoke` API).

# CloudWatch vs CloudTrail

CloudWatch	CloudTrail
<b>Performance monitoring (operations)</b>	<b>Auditing (security)</b>
<b>Log events across AWS services – think operations</b>	<b>Log API activity across AWS services – think activities</b>
<b>Higher-level comprehensive monitoring and eventing</b>	<b>More low-level granular</b>
<b>Log from multiple accounts</b>	<b>Log from multiple accounts</b>
<b>Logs stored indefinitely</b>	<b>Logs stored to S3 or CloudWatch indefinitely</b>
<b>Alarms history for 14 days</b>	<b>No native alarming; can use CloudWatch alarms</b>

# AWS X-Ray Overview



## AWS X-Ray Overview

- AWS X-Ray helps developers analyze and debug production, distributed applications, such as those built using a microservices architecture.
- AWS X-Ray supports applications running on:
  - Amazon EC2.
  - Amazon ECS.
  - AWS Lambda.
  - AWS Elastic Beanstalk.
- Need to integrate the X-Ray SDK with your application and install the X-Ray agent.

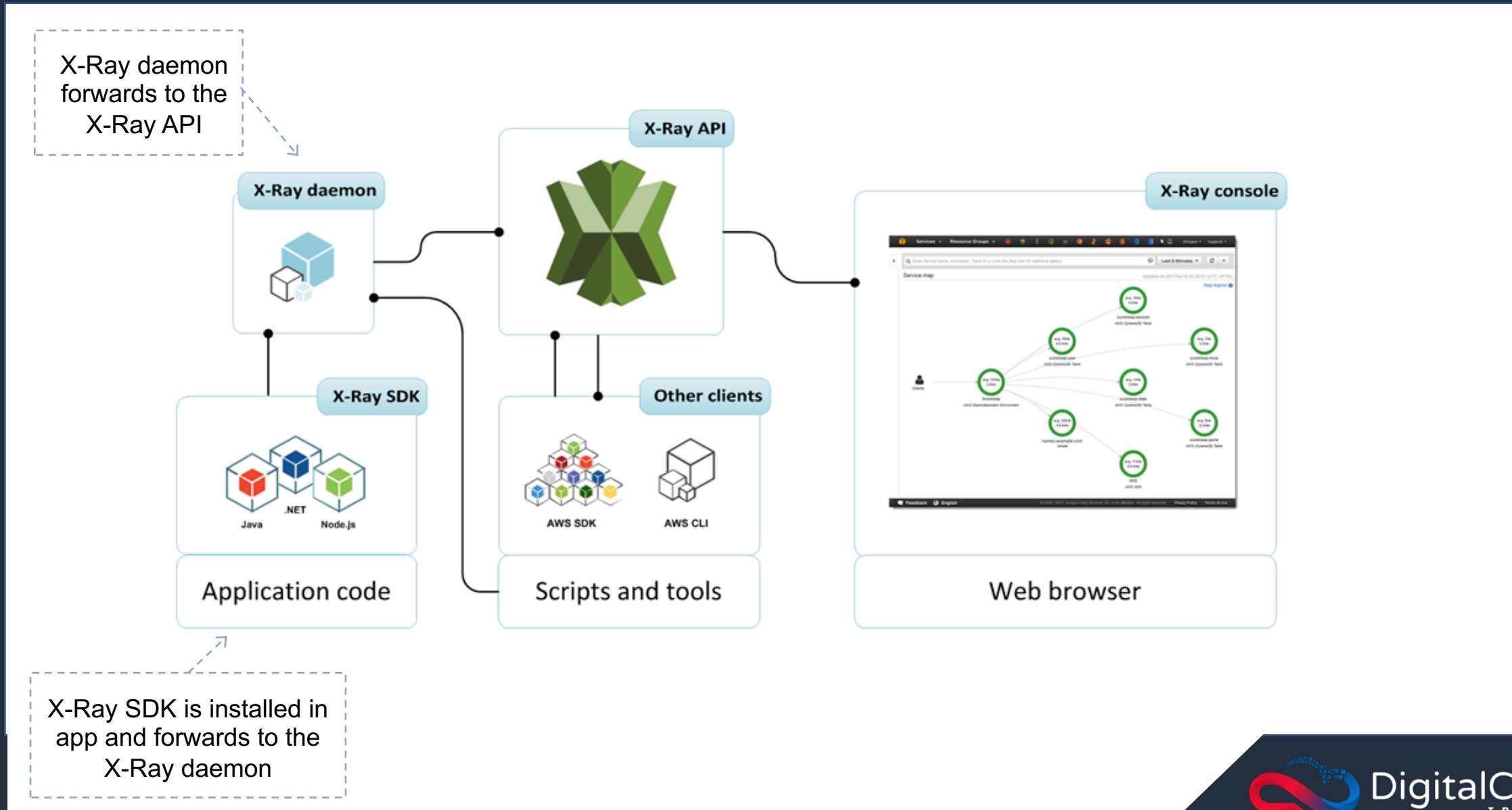
## AWS X-Ray Overview

- X-Ray on EC2 / On-premise:
  - Linux system must run the X-Ray daemon.
  - IAM instance role if EC2, other AWS credentials on on-premise instance.
- X-Ray on Lambda:
  - Make sure the X-Ray integration is ticked in the Lambda configuration (Lambda will run the daemon).
  - IAM role is the Lambda role.

## AWS X-Ray Overview

- X-Ray on Elastic Beanstalk:
  - Set configuration in the Elastic Beanstalk console.
  - Or use the Beanstalk extension (.ebextensions/xray-daemon.config)
- X-Ray on ECS/EKS/Fargate:
  - Create a Docker image that runs the daemon or use the official X-Ray Docker image.
  - Ensure port mappings and network settings are correct and IAM task roles are defined.

# AWS X-Ray Overview



## AWS X-Ray SDK

- The X-Ray SDK is installed in your application and forwards to the X-Ray daemon which forwards to the X-Ray API.
- You can then visualize what is happening in the X-Ray console.
- The X-Ray SDK provides:
  - Interceptors to add your code to trace incoming HTTP requests.
  - Client handlers to instrument AWS SDK client that your application uses to call other AWS services.
  - An HTTP client to use to instrument calls to other internal and external HTTP web services.

## AWS X-Ray SDK

- Code must be instrumented to use the AWS X-Ray SDK.
- The IAM role must be correct to send traces to X-Ray.

# AWS X-Ray Traces

- An X-Ray trace is a set of data points that share the same trace ID.

Trace list						
ID	AGE	METHOD	RESPONSE	RESPONSE TIME	URL	
...99829bcfe4879ae	4.9 min			1.9 sec		
...7ca85e421f2ad20	3.5 min	POST	200	89.0 ms	http://scorekeep-env.eb...	
...e4a825c60a6d07	1.4 min	GET	200	14.0 ms	http://scorekeep-env.eb...	
...f902af2a7b0f7452	1.2 min	GET	200	9.0 ms	http://scorekeep-env.eb...	
...f2592062a09955d	3.8 min	POST	200	211 ms	http://scorekeep-env.eb...	
...97820381d86a9e	3.5 min	POST	200	85.0 ms	http://scorekeep-env.eb...	
...3eb01960525202	15.4 sec	GET	200	10.0 ms	http://scorekeep-env.eb...	
...df2e71793a6a5a5	10.4 sec	GET	200	10.0 ms	http://scorekeep-env.eb...	
...d3cb3464967acf1	3.5 min	POST	200	132 ms	http://scorekeep-env.eb...	
...39dd015d584294	3.5 min	POST	200	112 ms	http://scorekeep-env.eb...	
...8ff6ab041fc62c46	3.8 min	PUT	200	79.0 ms	http://scorekeep-env.eb...	
...b7d0b3681e58e1	3.8 min	PUT	200	24.0 ms	http://scorekeep-env.eb...	
...f20535a7290b93d	0.4 sec	GET	200	9.0 ms	http://scorekeep-env.eb...	
...81f99716a0ed469	3.6 min	POST	200	137 ms	http://scorekeep-env.eb...	

# AWS X-Ray Segments

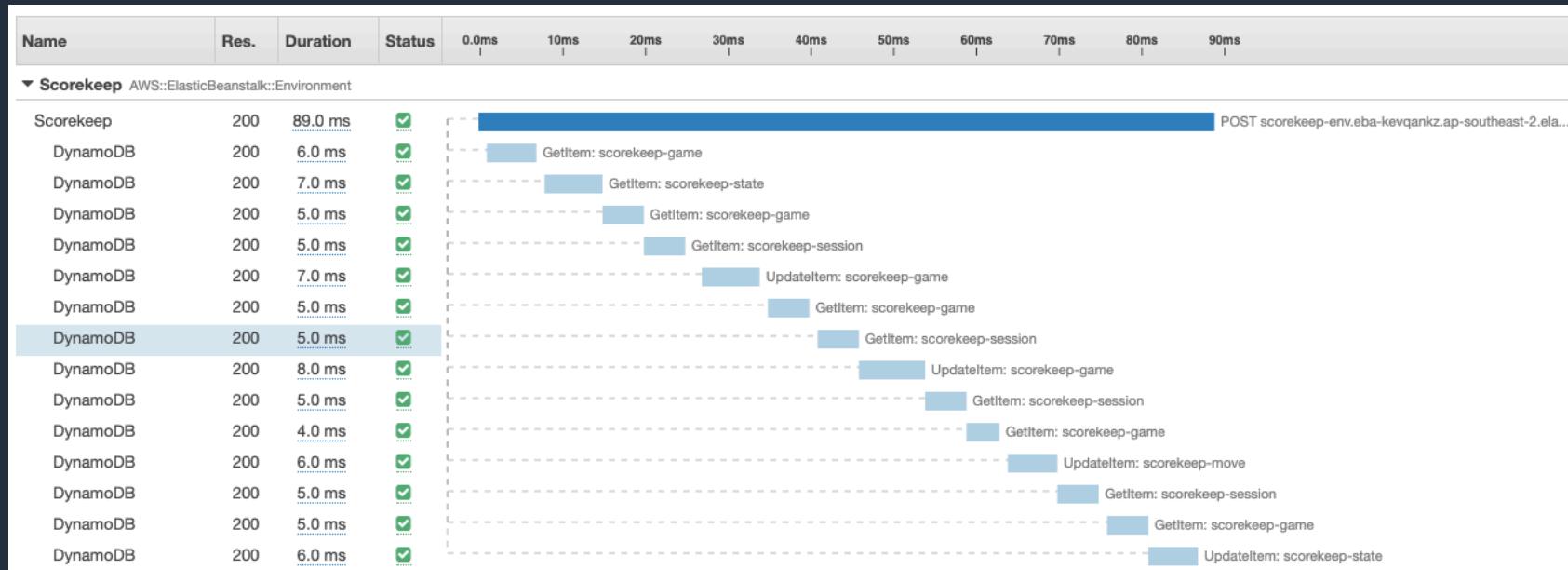
- An X-Ray segment encapsulates all the data points for a single component (for example, authorization service) of the distributed application.
- Segments include system-defined and user-defined data in the form of annotations and are composed of one or more sub-segments that represent remote calls made from the service.

Segment - Scorekeep	
<a href="#">Overview</a>	
<a href="#">Resources</a>	
<a href="#">Annotations</a>	
<a href="#">Metadata</a>	
<a href="#">Exceptions</a>	
Segment ID	1a2c8fa70d37cf85
Parent ID	
Name	Scorekeep
Origin	AWS::ElasticBeanstalk::Environment
<b>Time</b>	
Start time	2020-03-28 01:17:03.424 (UTC)
End time	2020-03-28 01:17:03.513 (UTC)
Duration	89.0 ms
In progress	false
<b>Errors &amp; Faults</b>	
Error	false
Fault	false
<b>Request &amp; Response</b>	
Request url	<a href="http://scorekeep-env.eba-kevqankz.ap-southeast-2.elasticbeanstalk.com/api/move/TAH81J0U/KBMFBE42/BC6GJL5N">http://scorekeep-env.eba-kevqankz.ap-southeast-2.elasticbeanstalk.com/api/move/TAH81J0U/KBMFBE42/BC6GJL5N</a>
Request method	POST
Request user_agent	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.149 Safari/537.36
Request client_ip	49.182.18.7
Request x_forwarded_for	true
Response status	200

## AWS X-Ray Subsegments

- Subsegments provide more granular timing information and details about downstream calls that your application made to fulfill the original request.
- A subsegment can contain additional details about a call to an AWS service, an external HTTP API, or an SQL database.
- You can even define arbitrary subsegments to instrument specific functions or lines of code in your application.
- For services that don't send their own segments, like Amazon DynamoDB, X-Ray uses subsegments to generate inferred segments and downstream nodes on the service map.
- This lets you see all of your downstream dependencies, even if they don't support tracing, or are external.

# AWS X-Ray Subsegments



Subsegment - DynamoDB

Overview	Resources	Annotations	Metadata	Exceptions
Subsegment ID 453489941bfe1def	Parent ID 1a2c8fa70d37cf85	Name DynamoDB		
<b>Time</b>				
Start time 2020-03-28 01:17:03.425 (UTC)	End time 2020-03-28 01:17:03.431 (UTC)	Duration 6.0 ms	In progress false	
<b>Errors &amp; Faults</b>				
Error false	Fault false			
<b>Request &amp; Response</b>				
Response status 200	Response content_length 397			

## AWS X-Ray Annotations

- An X-Ray annotation is system-defined or user-defined data associated with a segment.
- System-defined annotations include data added to the segment by AWS services, whereas user-defined annotations are metadata added to a segment by a developer.
- A segment can contain multiple annotations.
- These are key / value pairs used to index traces and use with filters.
- Use annotations to record information on segments or subsegments that you want indexed for search.
- Can call `putAnnotation` with a String key, and a Boolean, Number, or String value:

```
document.putAnnotation("mykey", "my value");
```

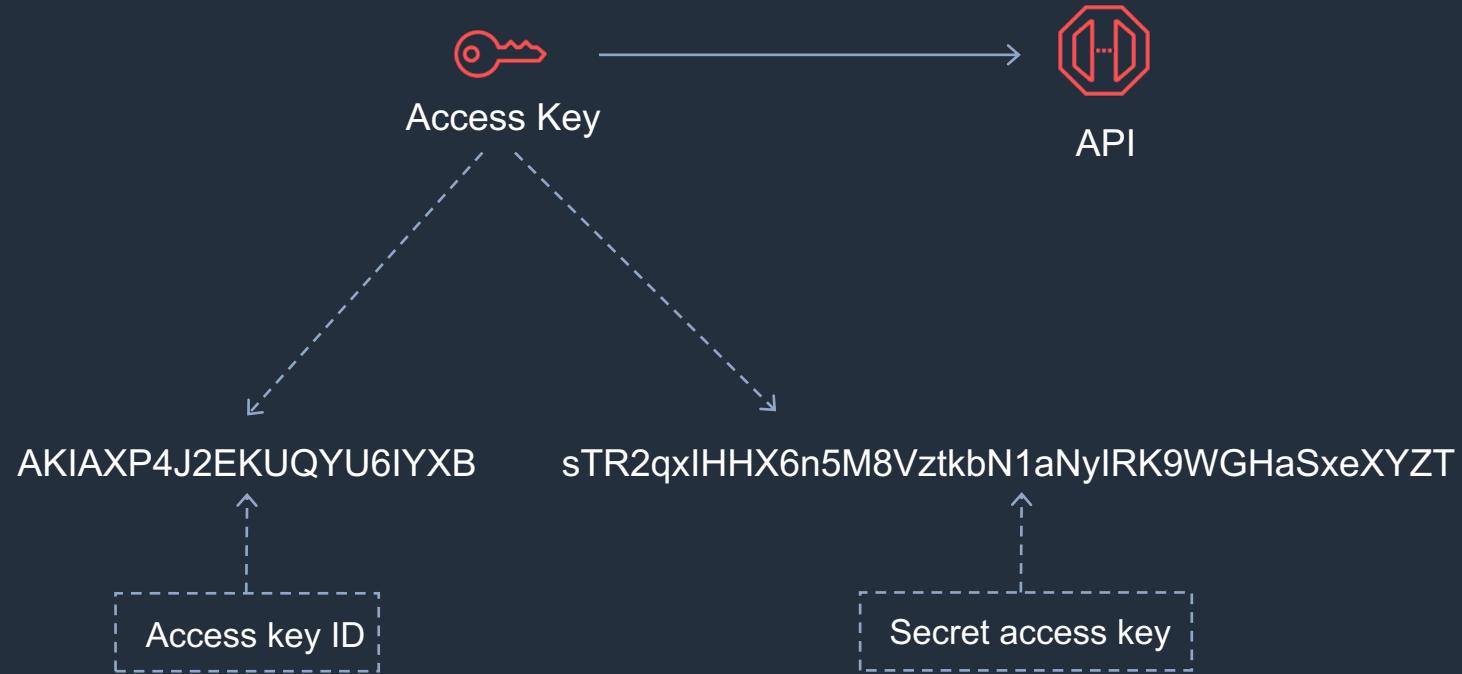
## AWS X-Ray Metadata

- Key / value pairs, not indexed and not used for searching.

# SECTION 20

# Advanced IAM

# IAM Access Keys



# IAM Access Keys

- Access keys are stored in:
  - Linux: `~/.aws/credentials`
  - Windows: `%UserProfile%\aws\credentials`
- Can include multiple profiles:

```
[default]
aws_access_key_id = AKIAXP4J2EKUQYU6IYXB
aws_secret_access_key = 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
[paul]
aws_access_key_id = AKIAXP4J2EKURIG7440I
aws_secret_access_key = VZ60RVGL1eqAI6TvaM6Q5rlIqNvgtZP5XyCo6SPN
```

## IAM Access Keys

- The “config” file is also stored in the same directory and stores the region selection:

```
[default]
region = ap-southeast-2
[profile paul]
region = ap-southeast-2
```

# AWS Managed and Customer Managed Policies

## AWS Managed Policies

- An AWS managed policy is a standalone policy that is created and administered by AWS.
- Standalone policy means that the policy has its own Amazon Resource Name (ARN) that includes the policy name.
- AWS managed policies are designed to provide permissions for many common use cases.
- You cannot change the permissions defined in AWS managed policies.
- The following list of policies are all AWS managed policies:

	Policy name ▾	Type	Used as	Description
○	▶ <a href="#">AmazonDynamoDBFullAccess</a>	AWS managed	Permissions policy (3)	Provides full access to Amazon DynamoDB via the AWS Management Console.
○	▶ <a href="#">AmazonDynamoDBFullAccesswithDat...</a>	AWS managed	None	Provides full access to Amazon DynamoDB including Export/Import using AWS Data P...
○	▶ <a href="#">AmazonDynamoDBReadOnlyAccess</a>	AWS managed	None	Provides read only access to Amazon DynamoDB via the AWS Management Console.
○	▶ <a href="#">AmazonEC2ContainerRegistryFullAcc...</a>	AWS managed	None	Provides administrative access to Amazon ECR resources
○	▶ <a href="#">AmazonEC2ContainerRegistryPower...</a>	AWS managed	None	Provides full access to Amazon EC2 Container Registry repositories, but does not allo...
○	▶ <a href="#">AmazonEC2ContainerRegistryReadOnly</a>	AWS managed	None	Provides read-only access to Amazon EC2 Container Registry repositories.
○	▶ <a href="#">AmazonEC2ContainerServiceAutoscal...</a>	AWS managed	Permissions policy (1)	Policy to enable Task Autoscaling for Amazon EC2 Container Service
○	▶ <a href="#">AmazonEC2ContainerServiceEventsR...</a>	AWS managed	None	Policy to enable CloudWatch Events for EC2 Container Service
○	▶ <a href="#">AmazonEC2ContainerServiceforEC2R...</a>	AWS managed	Permissions policy (1)	Default policy for the Amazon EC2 Role for Amazon EC2 Container Service.

## AWS Managed Policies

- Some AWS managed policies are designed for specific job functions.
- The job-specific AWS managed policies include:
  - Administrator
  - Billing
  - Database Administrator
  - Data Scientist
  - Developer Power User
  - Network Administrator
  - Security Auditor
  - Support User
  - System Administrator
  - View-Only User

# Customer Managed Policies

- You can create standalone policies that you administer in your own AWS account, which we refer to as customer managed policies.
- You can then attach the policies to multiple principal entities in your AWS account.
- When you attach a policy to a principal entity, you give the entity the permissions that are defined in the policy.

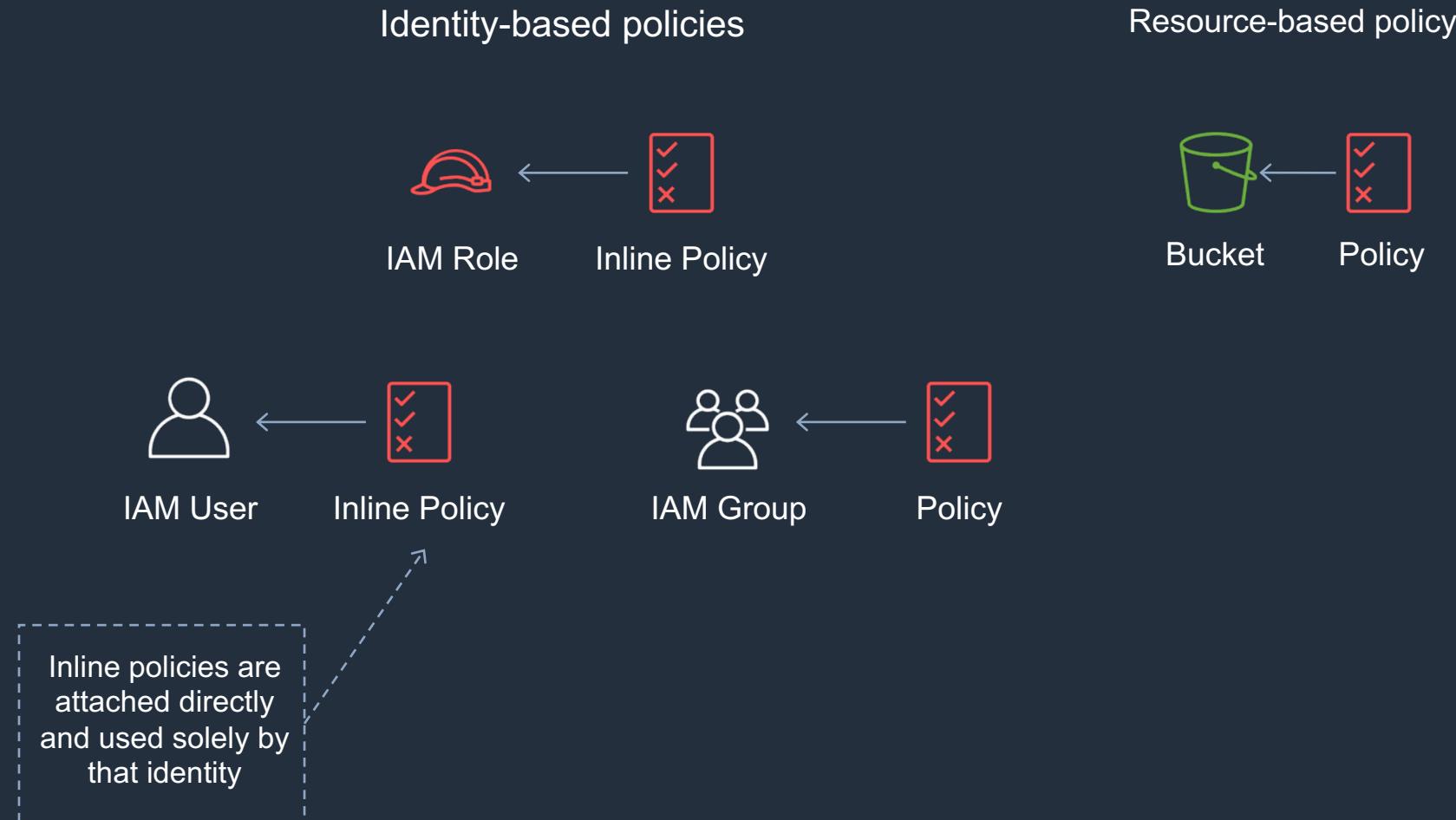
The screenshot shows the AWS IAM Policy editor interface. On the left, a tree view displays policy details:

- EC2 (All actions)**:
  - Service**: EC2
  - Actions**: Manual actions
    - \*
  - Resources**: All resources
  - Request conditions**: Restrict Source IP: 1.1.1.1

On the right, the corresponding JSON code is shown:

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Sid": "VisualEditor0",  
      "Effect": "Allow",  
      "Action": "ec2:*",  
      "Resource": "*",  
      "Condition": {  
        "IpAddress": {  
          "aws:SourceIp": "1.1.1.1"  
        }  
      }  
    }  
  ]  
}
```

# Identity-Based and Resource-Based Policies



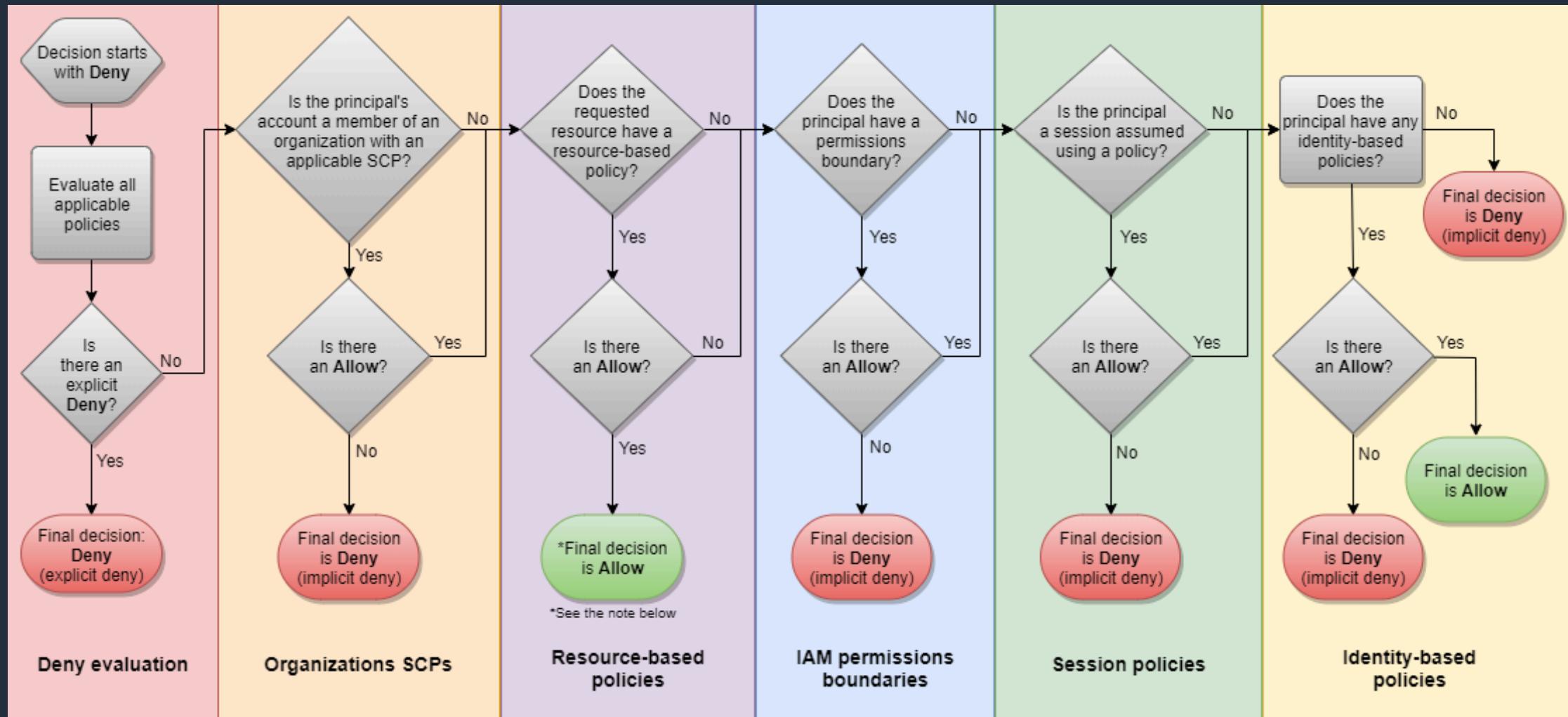
## IAM Policy Evaluation Logic

- **Identity-based policies** – Identity-based policies are attached to an IAM identity (user, group of users, or role) and grant permissions to IAM entities (users and roles).
- **Resource-based policies** – Resource-based policies grant permissions to the principal (account, user, role, or federated user) specified as the principal.
- **IAM permissions boundaries** – Permissions boundaries are an advanced feature that sets the maximum permissions that an identity-based policy can grant to an IAM entity (user or role).
- **AWS Organizations service control policies (SCPs)** – Organizations SCPs specify the maximum permissions for an organization or organizational unit (OU).
- **Session policies** – Session policies are advanced policies that you pass as parameters when you programmatically create a temporary session for a role or federated user.

## IAM Policy Evaluation Logic

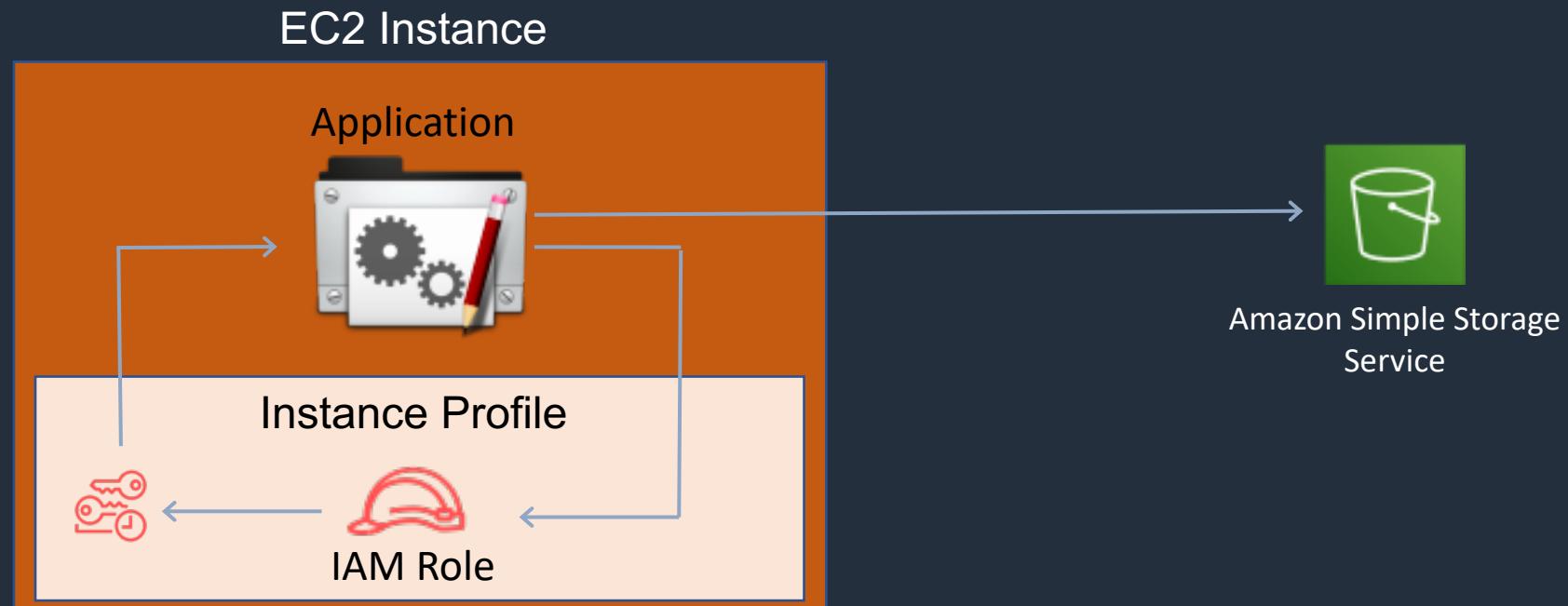
- By default, all requests are implicitly denied. (Alternatively, by default, the AWS account root user has full access.)
- An explicit allow in an identity-based or resource-based policy overrides this default.
- If a permissions boundary, Organizations SCP, or session policy is present, it might override the allow with an implicit deny.
- An explicit deny in any policy overrides any allows.

# IAM Policy Evaluation Logic



## IAM Instance Profiles

- An instance profile is a container for an IAM role that you can use to pass role information to an EC2 instance when the instance starts.
- An instance profile can contain only one IAM role, although a role can be included in multiple instance profiles.

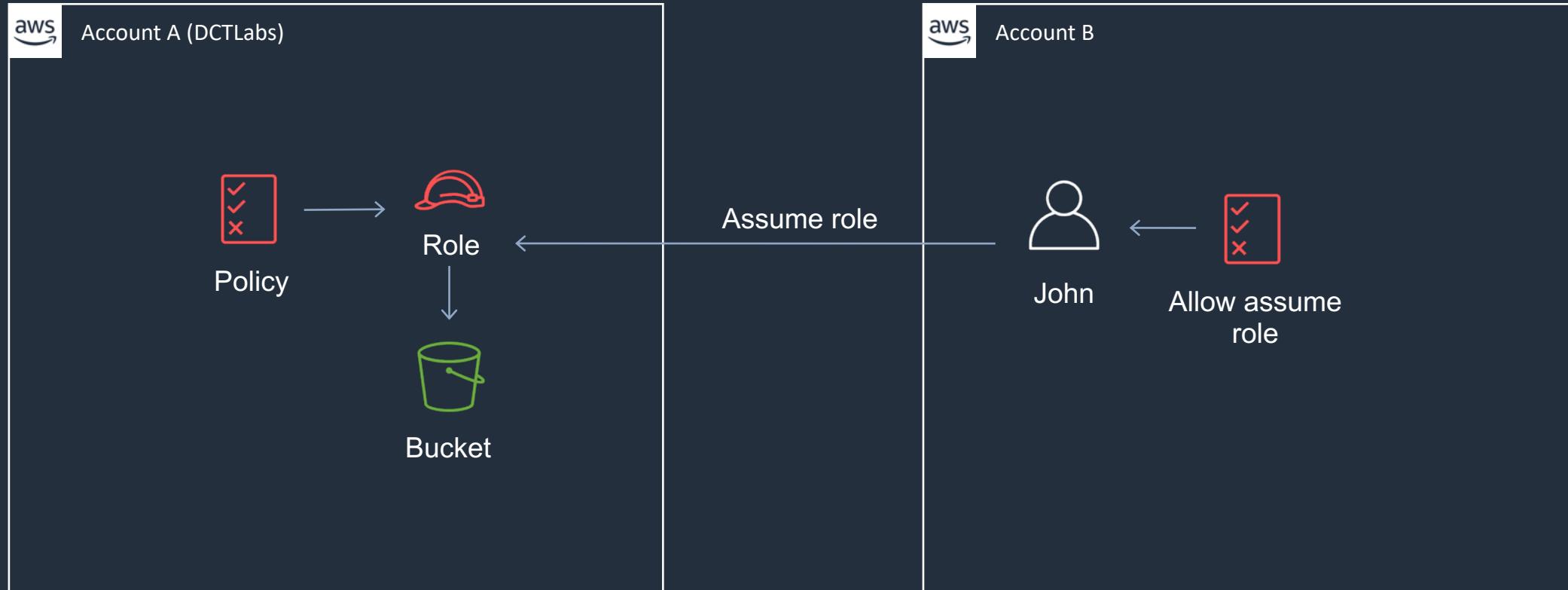


## IAM Instance Profiles

- You can use the following AWS CLI commands to work with instance profiles in an AWS account.
  - Create an instance profile: `aws iam create-instance-profile`
  - Add a role to an instance profile: `aws iam add-role-to-instance-profile`
  - List instance profiles: `aws iam list-instance-profiles`, `aws iam list-instance-profiles-for-role`
  - Get information about an instance profile: `aws iam get-instance-profile`
  - Remove a role from an instance profile: `aws iam remove-role-from-instance-profile`
  - Delete an instance profile: `aws iam delete-instance-profile`

# Cross-Account Access

**NOTE:** Hands-on lab provided for this pattern in section 7



## Cross-Account Access

- Useful for situations where an AWS customer has separate AWS account – for example for development and production resources.
- Cross Account Access makes it easier to work productively within a multi-account (or multi-role) AWS environment by making it easy to switch roles within the AWS Management Console.
- Can sign-in to the console using your IAM user name and then switch the console to manage another account without having to enter another user name and password.
- Lets users from one AWS account access resources in another.
- To make a request in a different account the resource in that account must have an attached resource-based policy with the permissions you need.
- Or you must assume a role (identity-based policy) within that account with the permissions you need.

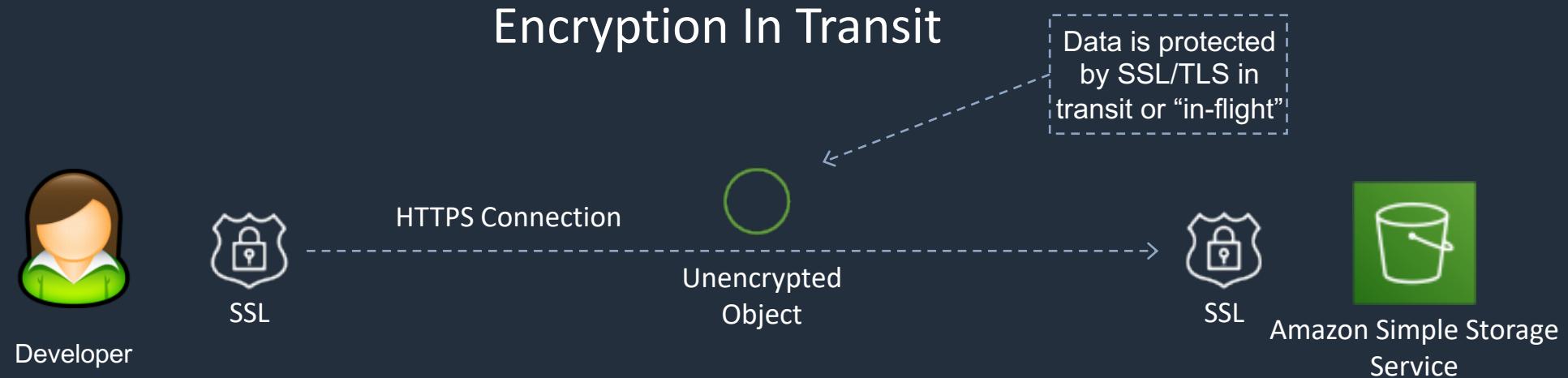
# IAM Best Practices

- Lock Away Your AWS Account Root User Access Keys
- Create Individual IAM Users
- Use Groups to Assign Permissions to IAM Users
- Grant Least Privilege
- Get Started Using Permissions with AWS Managed Policies
- Use Customer Managed Policies Instead of Inline Policies
- Use Access Levels to Review IAM Permissions
- Configure a Strong Password Policy for Your Users
- Enable MFA
- Use Roles for Applications That Run on Amazon EC2 Instances
- Use Roles to Delegate Permissions
- Do Not Share Access Keys
- Rotate Credentials Regularly
- Remove Unnecessary Credentials
- Use Policy Conditions for Extra Security
- Monitor Activity in Your AWS Account

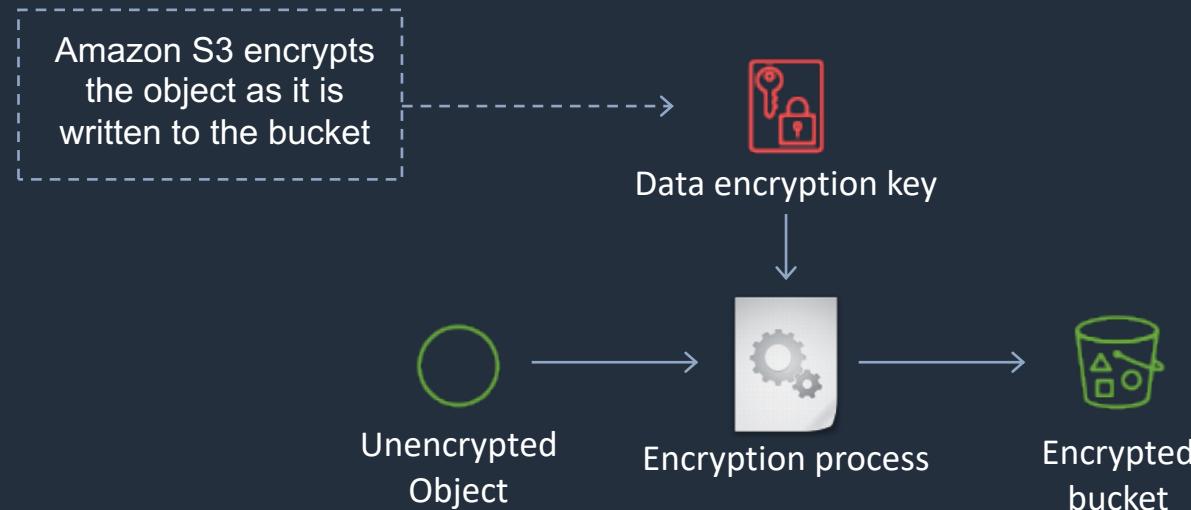
# SECTION 21

## AWS Security Services

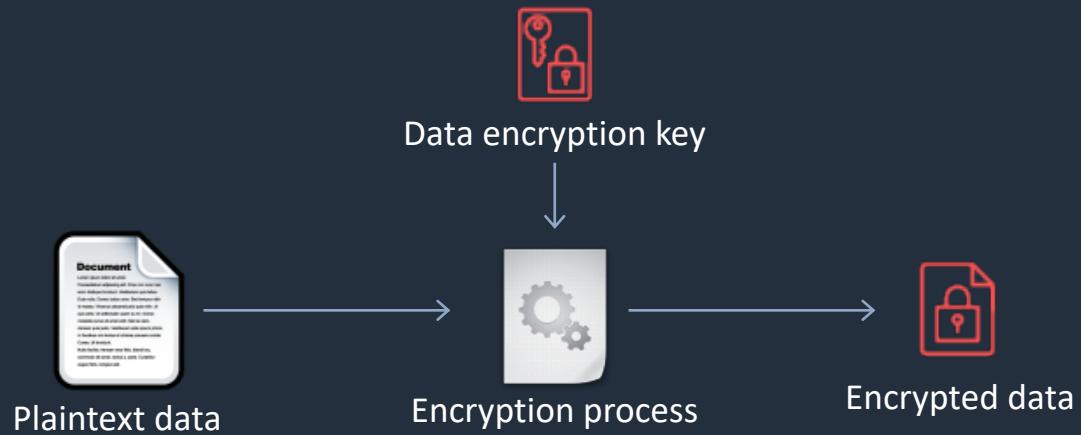
# Encryption – In Transit vs At Rest



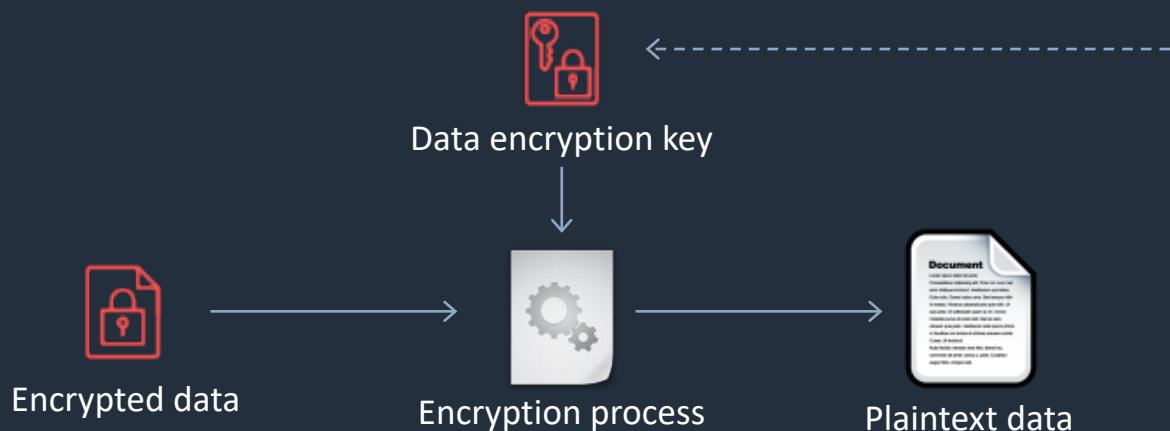
### Encryption At Rest



# Symmetric Encryption



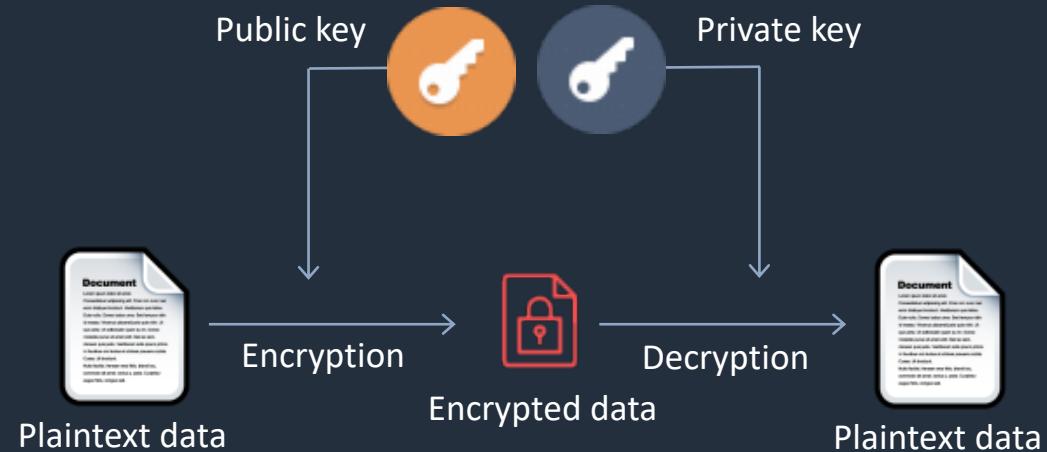
# Decryption



The same key is used for both encryption and decryption

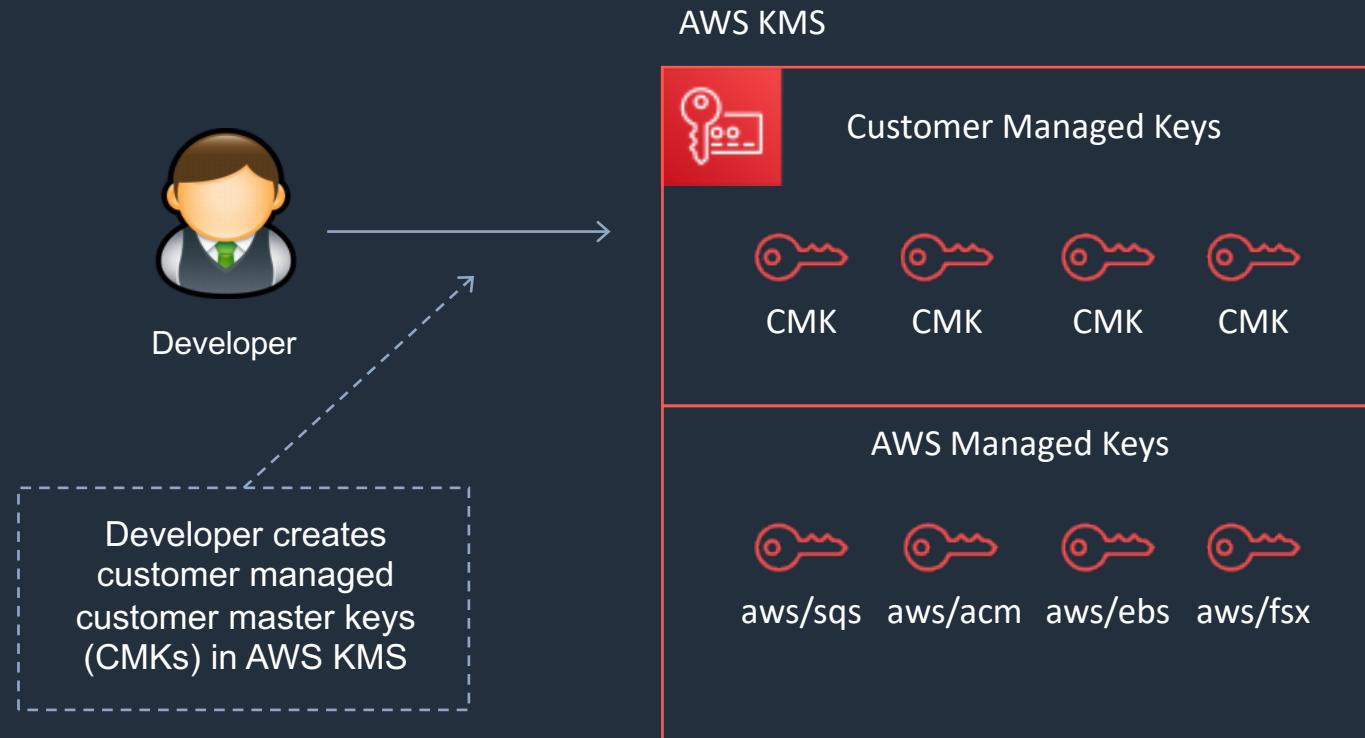
# Asymmetric Encryption

- Asymmetric encryption is also known as public key cryptography.
- Messages encrypted with the public key can only be decrypted with the private key.
- Messages encrypted with the private key can be decrypted with the public key.
- Examples include SSL/TLS and SSH.



# AWS Key Management Service (KMS)

- AWS KMS is a service for creating and controlling encryption keys.
- The customer master keys (CMKs) are protected by hardware security modules (HSMs).



## AWS Key Management Service (KMS)

- With AWS KMS you can also perform the following cryptographic functions using master keys:
  - Encrypt, decrypt, and re-encrypt data.
  - Generate data encryption keys that you can export from the service in plaintext or encrypted under a master key that doesn't leave the service.
  - Generate random numbers suitable for cryptographic applications.

## AWS KMS – Customer Master Keys (CMKs)

- Customer master keys are the primary resources in AWS KMS.
- The CMK includes metadata, such as the key ID, creation date, description, and key state.
- The CMK also contains the key material used to encrypt and decrypt data.
- AWS KMS supports symmetric and asymmetric CMKs.
- CMKs are created in AWS KMS. Symmetric CMKs and the private keys of asymmetric CMKs never leave AWS KMS unencrypted.
- By default, AWS KMS creates the key material for a CMK.
- A CMK can encrypt data up to 4KB in size.
- A CMK can generate, encrypt and decrypt Data Encryption Keys (DEKs).

AWS KMS



# AWS KMS – Customer Master Keys (CMKs)

Type of CMK	Can view	Can manage	Used only for my AWS account	Automatic rotation
Customer managed CMK	Yes	Yes	Yes	Optional. Every 365 days
AWS managed CMK	Yes	No	Yes	Required. Every 1095 days
AWS owned CMK	No	No	No	Varies

## AWS KMS – Customer Managed CMKs

- Customer managed CMKs are CMKs in your AWS account that you create, own, and manage.
- You have full control over these CMKs, including establishing and maintaining their key policies, IAM policies, and grants, enabling and disabling them, rotating their cryptographic material, adding tags, creating aliases that refer to the CMK, and scheduling the CMKs for deletion.
- Customer managed CMKs incur a monthly fee and a fee for use in excess of the free tier.

Customer managed keys (3)					
<input type="text"/> Filter keys by alias, key ID, or key type					
<input type="checkbox"/>	Alias	Key ID	Status	Key spec ⓘ	Key usage
<input type="checkbox"/>	MyEVTest	166aa513-c989-4ee4-8a2f-9d4d2f6b5766	Enabled	SYMMETRIC_DEFAULT	Encrypt and decrypt
<input type="checkbox"/>	MyTestCMK	56d9c8ac-c4d0-40d9-9246-28776cf385ad	Enabled	SYMMETRIC_DEFAULT	Encrypt and decrypt
<input type="checkbox"/>	MyNewKey	aafdf495-60be-4dbc-a231-dba98724722d	Enabled	SYMMETRIC_DEFAULT	Encrypt and decrypt

## AWS KMS – AWS Managed CMKs

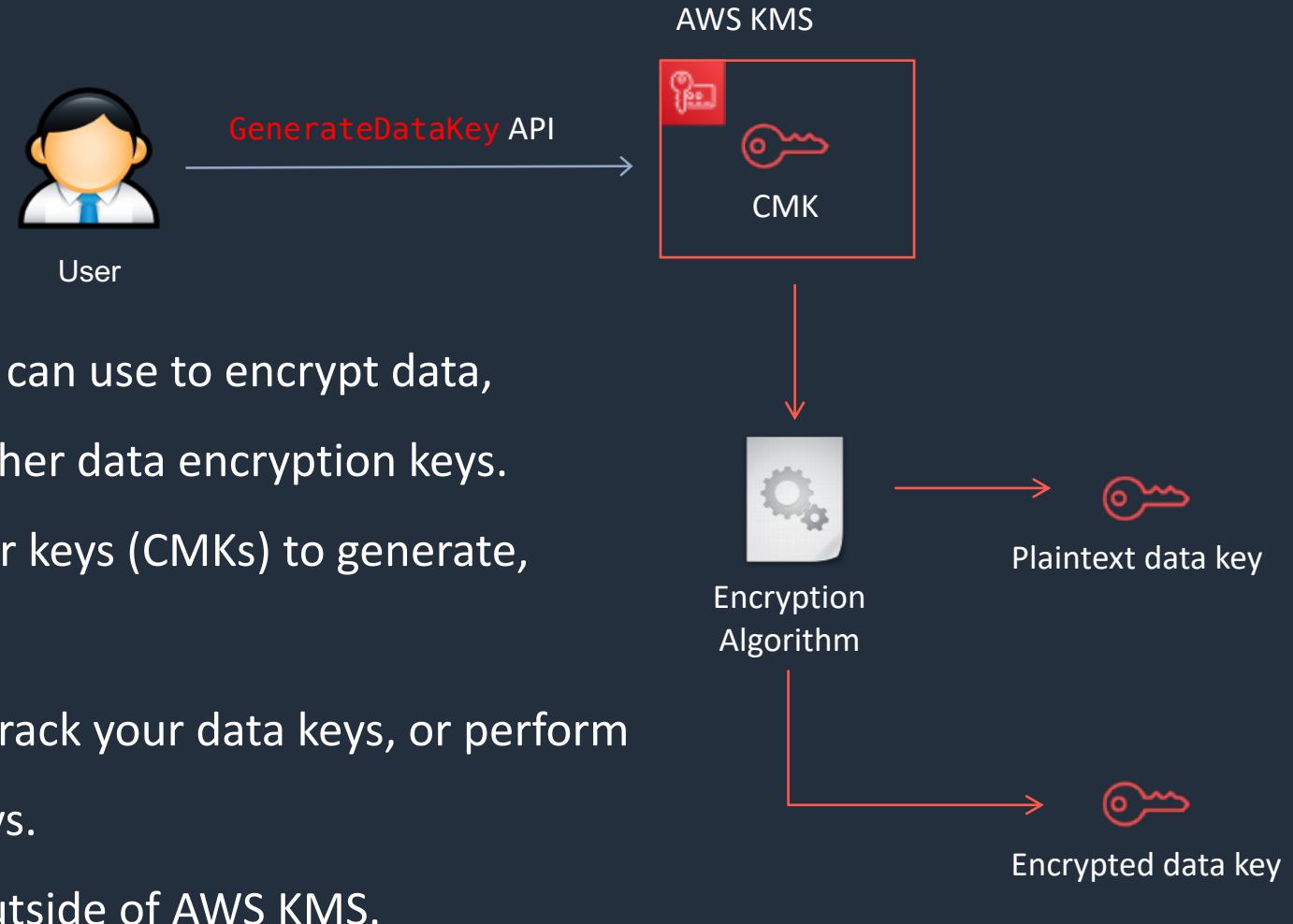
- AWS managed CMKs are CMKs in your account that are created, managed, and used on your behalf by an AWS service that is integrated with AWS KMS.
- You cannot manage these CMKs, rotate them, or change their key policies.
- You also cannot use AWS managed CMKs in cryptographic operations directly; the service that creates them uses them on your behalf.
- You do not pay a monthly fee for AWS managed CMKs. They can be subject to fees for use in excess of the free tier, but some AWS services cover these costs for you.

Alias	Key ID
aws/sqs	025b9386-b1f8-4fa9-84e2-ac3220b1de59
aws/acm	2d604e85-c2d4-42dc-ab0b-0b356f5fe26e
aws/codecommit	41fea9df-e447-4992-8af7-6ddec6d81175
aws/elasticfilesystem	460c4f05-fe98-4a35-b940-3e1992f04314
aws/glue	617516fe-bf19-4da4-a743-2b13c41973e1
aws/lambda	7f513d01-784b-41b9-9c51-51621db7b5e1
aws/ebs	b9baa4f6-3e87-4256-af6a-d181940df286
aws/lightsail	bc7ba666-8e17-444a-800c-d3d4be303a97
aws/fsx	cebc434c-ee2b-4a61-9b5a-f63be9fdb068
aws/kinesis	d99014b5-09d4-480d-9b2a-3a7d7e3e9c5b

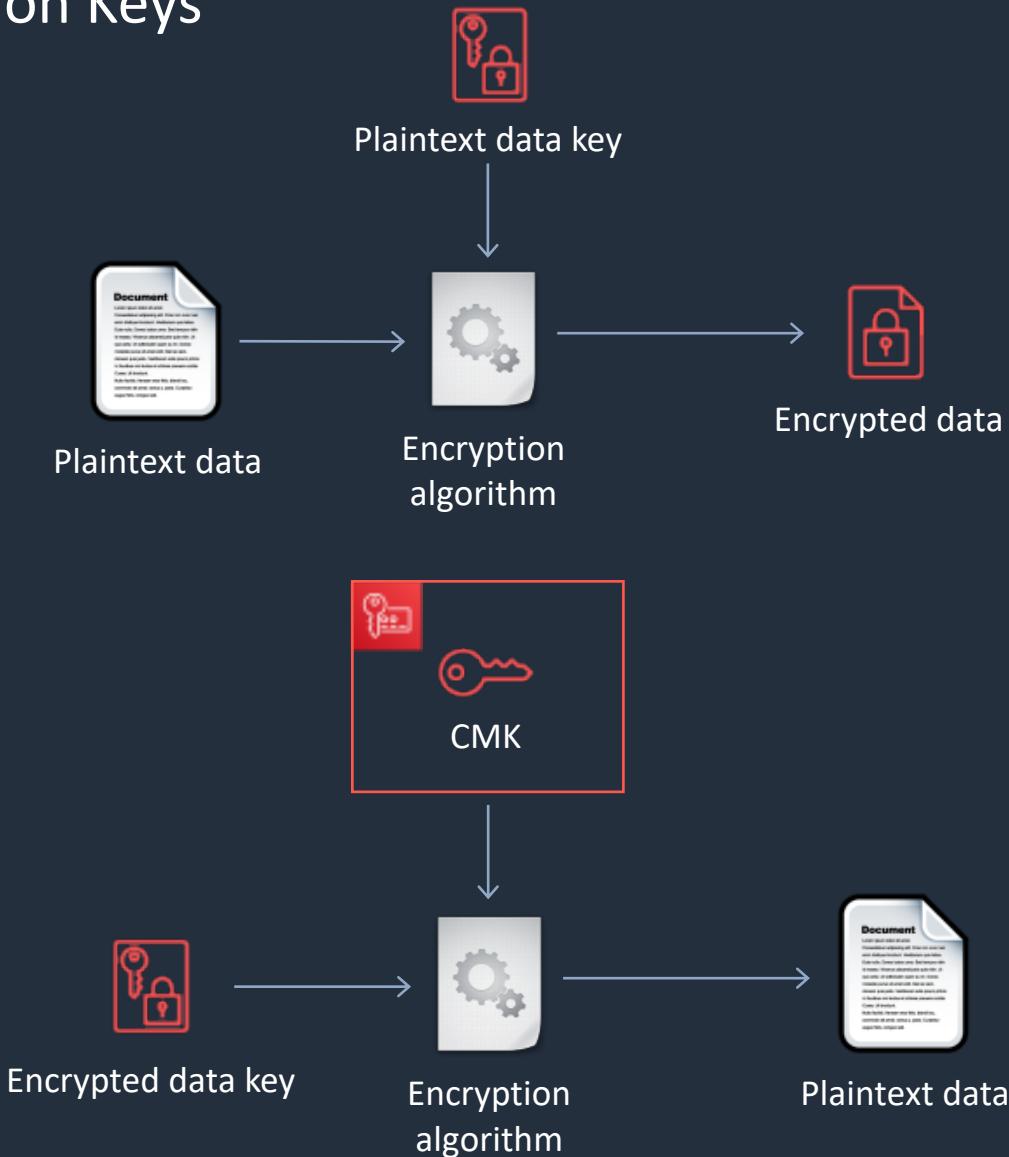
## AWS KMS – AWS Owned CMKs

- AWS owned CMKs are a collection of CMKs that an AWS service owns and manages for use in multiple AWS accounts.
- Although AWS owned CMKs are not in your AWS account, an AWS service can use its AWS owned CMKs to protect the resources in your account.
- You do not need to create or manage the AWS owned CMKs.
- However, you cannot view, use, track, or audit them.
- You are not charged a monthly fee or usage fee for AWS owned CMKs and they do not count against the AWS KMS quotas for your account.

# AWS KMS – Data Encryption Keys



# AWS KMS – Data Encryption Keys



## AWS KMS API and CLI

- **Encrypt** (aws kms encrypt):
  - Encrypts plaintext into ciphertext by using a customer master key (CMK).
  - You can encrypt small amounts of arbitrary data, such as a personal identifier or database password, or other sensitive information.
  - You can use the Encrypt operation to move encrypted data from one AWS region to another.
- **Decrypt** (aws kms decrypt):
- Decrypts ciphertext that was encrypted by an AWS KMS customer master key (CMK) using any of the following operations:
  - **Encrypt**
  - **GenerateDataKey**
  - **GenerateDataKeyValuePair**
  - **GenerateDataKeyWithoutPlaintext**
  - **GenerateDataKeyValuePairWithoutPlaintext**

## AWS KMS API and CLI

- **Re-encrypt** (aws kms re-encrypt):
  - Decrypts ciphertext and then re-encrypts it entirely within AWS KMS.
  - You can use this operation to change the customer master key (CMK) under which data is encrypted, such as when you manually rotate a CMK or change the CMK that protects a ciphertext.
  - You can also use it to re-encrypt ciphertext under the same CMK, such as to change the encryption context of a ciphertext.
- **Enable-key-rotation**:
  - Enables automatic rotation of the key material for the specified symmetric customer master key (CMK).
  - You cannot perform this operation on a CMK in a different AWS account.

## AWS KMS API and CLI

- **GenerateDataKey** (aws kms generate-data-key):
  - Generates a unique symmetric data key.
  - This operation returns a plaintext copy of the data key and a copy that is encrypted under a customer master key (CMK) that you specify.
  - You can use the plaintext key to encrypt your data outside of AWS KMS and store the encrypted data key with the encrypted data.
- **GenerateDataKeyWithoutPlaintext** (generate-data-key-without-plaintext):
  - Generates a unique symmetric data key.
  - This operation returns a data key that is encrypted under a customer master key (CMK) that you specify.
  - To request an asymmetric data key pair, use the **GenerateDataKeyValuePair** or **GenerateDataKeyValuePairWithoutPlaintext** operations.

# AWS CloudHSM

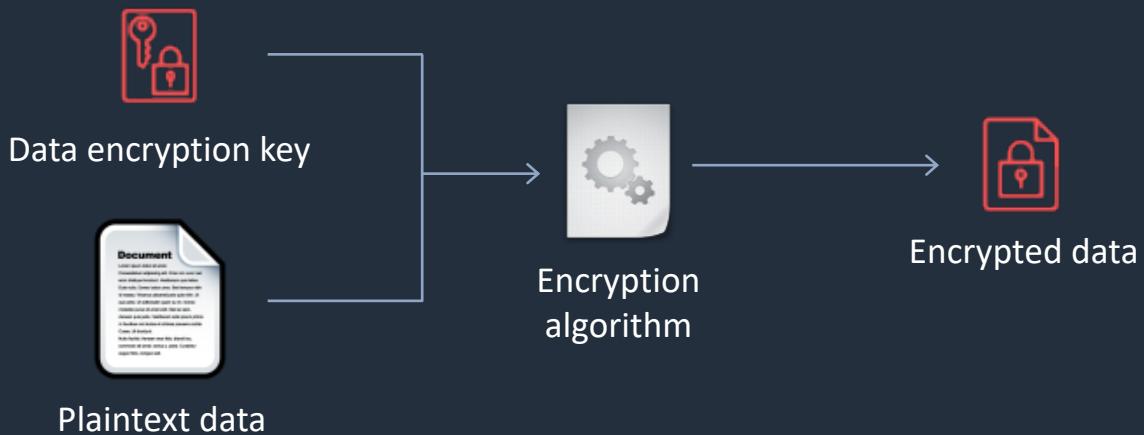
- AWS CloudHSM is a cloud-based hardware security module (HSM) that enables you to easily generate and use your own encryption keys on the AWS Cloud.
- You can manage your own encryption keys using FIPS 140-2 Level 3 validated HSMs.
- CloudHSM runs in your VPC.

CloudHSM		AWS KMS
<b>Tenancy</b>	Single-tenant HSM	Multi-tenant AWS service
<b>Availability</b>	Customer-managed durability and available	Highly available and durable key storage and management
<b>Root of Trust</b>	Customer managed root of trust	AWS managed root of trust
<b>FIPS 140-2</b>	Level 3	Level 2 / Level 3 in some areas
<b>3<sup>rd</sup> Party Support</b>	Broad 3 <sup>rd</sup> Party Support	Broad AWS service support

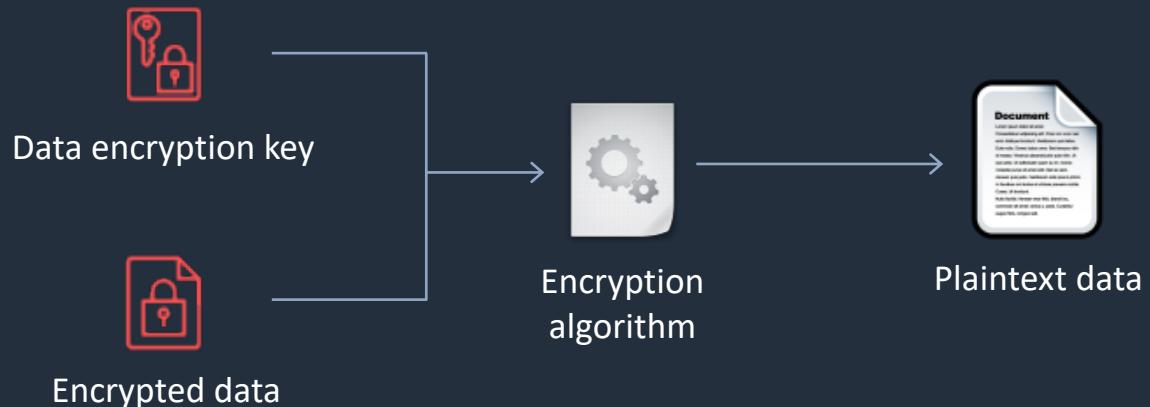
# AWS CloudHSM

- Benefits:
  - FIPS 140-2 level 3 validated HSMs.
  - You can configure AWS Key Management Service (KMS) to use your AWS CloudHSM cluster as a custom key store rather than the default KMS key store.
  - Managed service and automatically scales.
  - Retain control of your encryption keys - you control access (and AWS has no visibility of your encryption keys).

## Symmetric Key Encryption



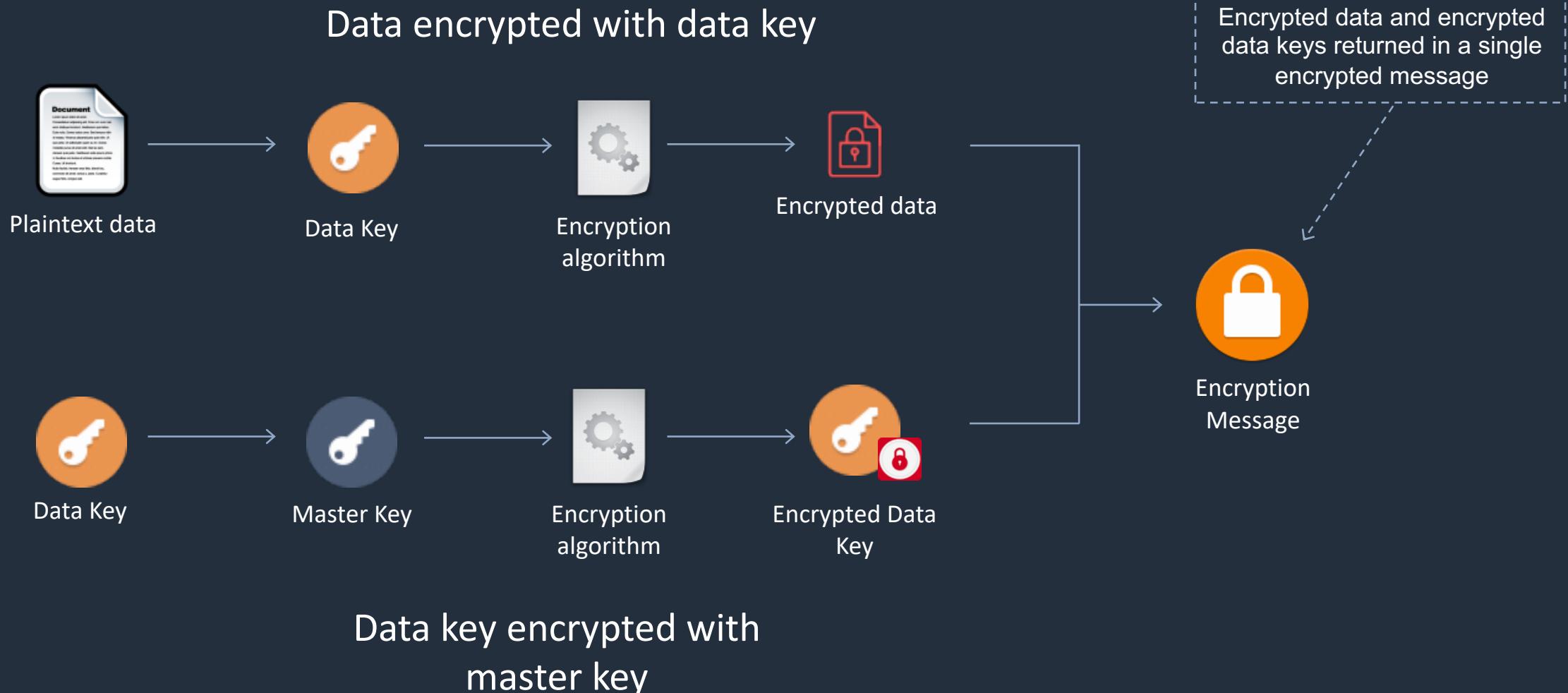
## Symmetric Key Decryption



## AWS Encryption SDK - Envelope Encryption

- With envelope encryption you encrypt the data key to protect it.
- To do this you need a master key.
- Benefits include:
  - Protection of data keys - can store the encrypted data key with the encrypted data.
  - Can reencrypt the data keys protecting data rather than the data (saving time).
  - Combine the strengths of multiple algorithms.
- Master keys must be protected (KMS, CloudHSM etc.).

# AWS Encryption SDK – Envelope Encryption



# S3 Encryption

## Server-side encryption with S3 managed keys (SSE-S3)

- S3 managed keys
- Unique object keys
- Master key
- AES 256



Encryption / decryption



## Server-side encryption with AWS KMS managed keys (SSE-KMS)



- KMS managed keys
- Customer master keys
- CMK can be customer generated



Encryption / decryption



## Server-side encryption with client provided keys (SSE-C)



Encryption / decryption



- Client managed keys
- AWS KMS CMK

## Client-side encryption



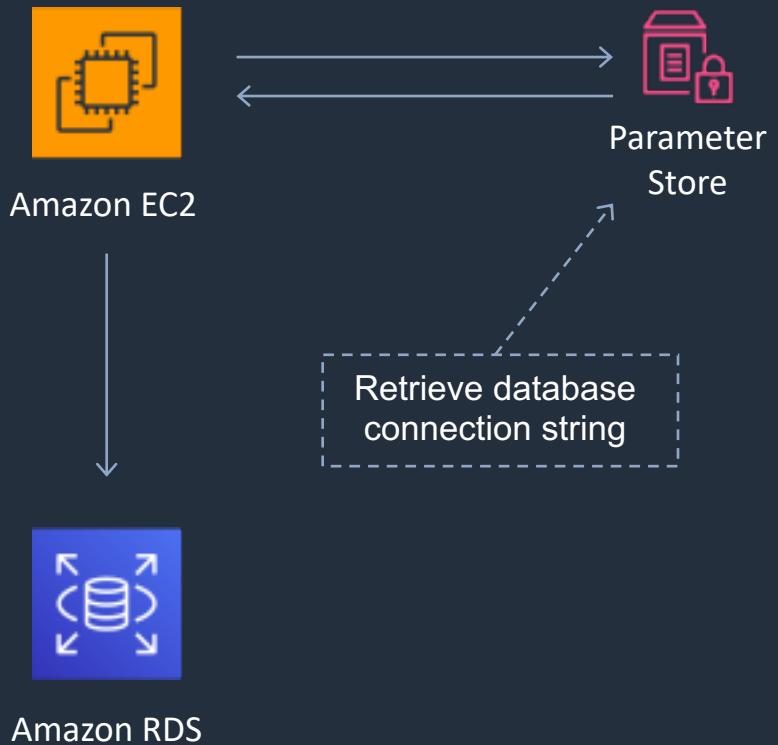
Encryption / decryption



- Client managed keys
- Not stored on AWS

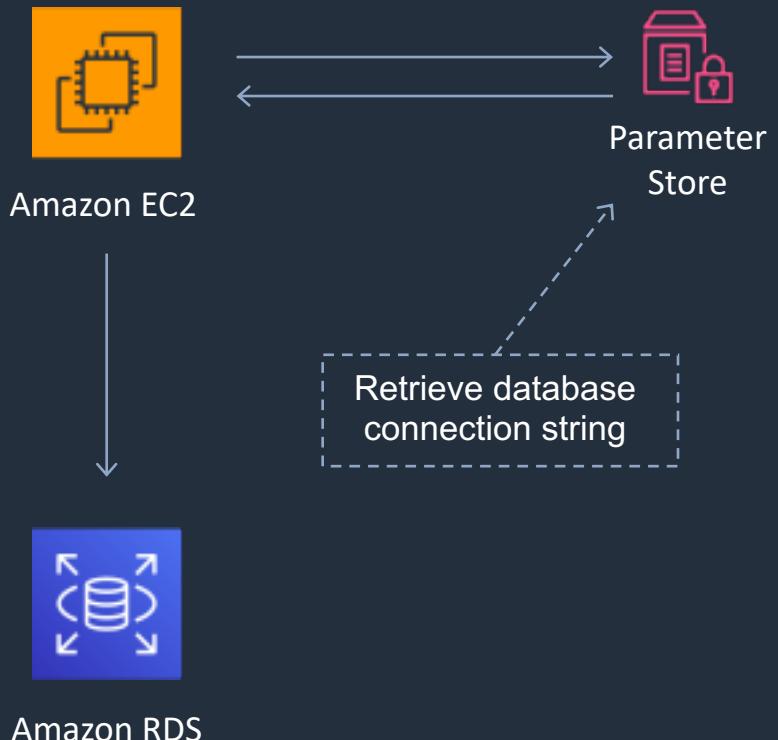
# AWS Systems Manager Parameter Store

- AWS Systems Manager Parameter Store provides secure, hierarchical storage for configuration data management and secrets management.
- It is highly scalable, available, and durable.
- You can store data such as passwords, database strings, and license codes as parameter values.
- You can store values as plaintext (unencrypted data) or ciphertext (encrypted data).
- You can then reference values by using the unique name that you specified when you created the parameter.



# AWS Systems Manager Parameter Store

- No native rotation of keys (difference with Secrets Manager which does it automatically).
- There are two tiers:
  - Standard – limit of 10,000 parameters, up to 4 KB, no additional charges.
  - Advanced – more than 10,000 parameters, up to 8 KB, charges apply.



# AWS Secrets Manager

- AWS Secrets Manager helps you meet your security and compliance requirements by enabling you to rotate secrets safely without the need for code deployments.
- Secrets Manager offers built-in integration for Amazon RDS, Amazon Redshift, and Amazon DocumentDB and rotates these database credentials on your behalf automatically.
- You can customize Lambda functions to extend Secrets Manager rotation to other secret types, such as API keys and OAuth tokens.

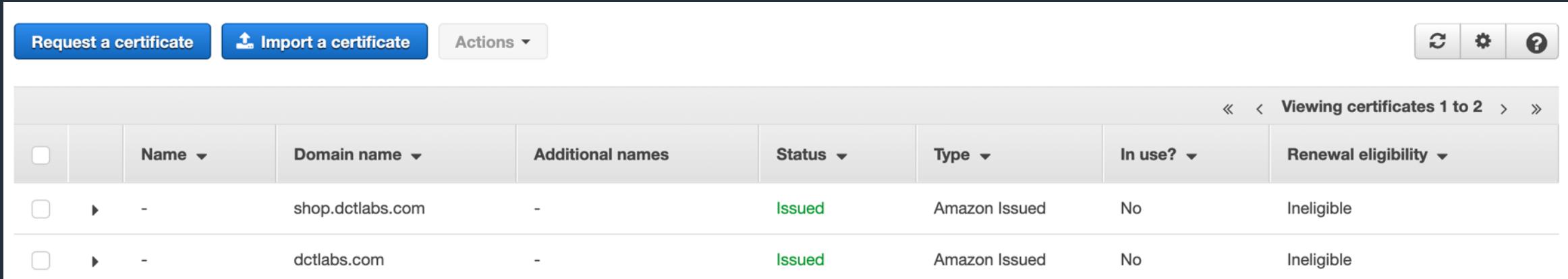


# AWS Secrets Manager vs SSM Parameter Store

	<b>Secrets Manager</b>	<b>SSM Parameter Store</b>
<b>Automatic Key Rotation</b>	Yes, built-in for some services, use Lambda for others	No native key rotation
<b>Key/Value Type</b>	Encrypted only	String, StringList, SecureString (encrypted)
<b>Change history</b>	No	Yes
<b>Price</b>	Charges apply per secret	Free for standard, charges for advanced

## AWS Certificate Manager (ACM)

- AWS Certificate Manager is used for creating and managing public SSL/TLS certificates.
- You can use public certificates provided by ACM (ACM certificates) or certificates that you import into ACM.
- Can also request private certificates from a private certificate authority (CA) created using AWS Certificate Manager Private Certificate Authority.
- ACM certificates can secure multiple domain names and multiple names within a domain.
- You can also use ACM to create wildcard SSL certificates that can protect an unlimited number of subdomains.



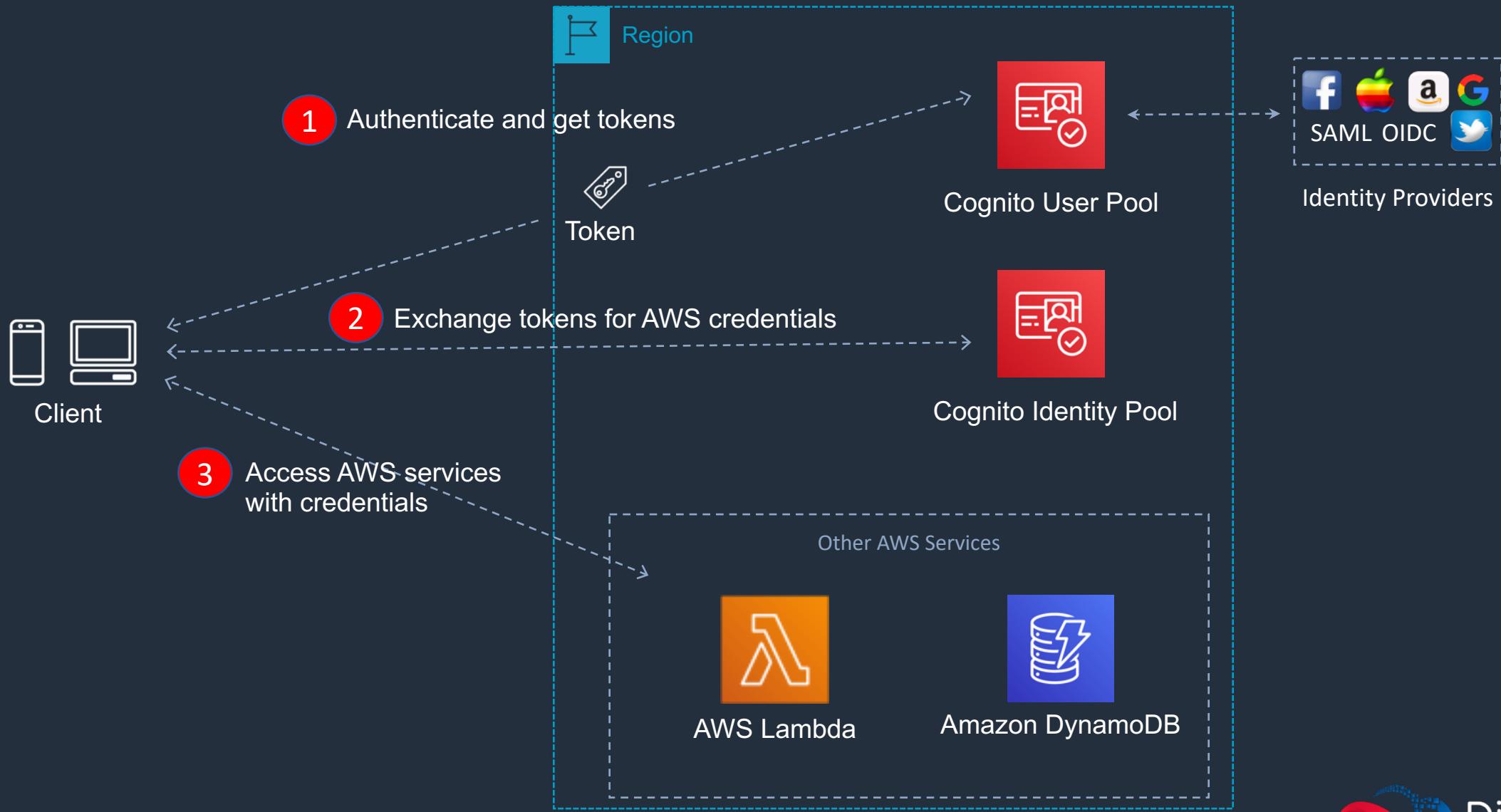
The screenshot shows the AWS Certificate Manager console interface. At the top, there are three buttons: 'Request a certificate' (blue), 'Import a certificate' (blue with an upload icon), and 'Actions' (dropdown). To the right are three small icons: a refresh symbol, a gear, and a question mark. Below the buttons is a header row with columns: 'Name ▾', 'Domain name ▾', 'Additional names', 'Status ▾', 'Type ▾', 'In use? ▾', and 'Renewal eligibility ▾'. The status column contains green text indicating 'Issued'. The type column shows 'Amazon Issued' for both entries. The renewal eligibility column shows 'Ineligible' for both. The table lists two certificates:

	Name ▾	Domain name ▾	Additional names	Status ▾	Type ▾	In use? ▾	Renewal eligibility ▾
<input type="checkbox"/>	▶ -	shop.dctlabs.com	-	Issued	Amazon Issued	No	Ineligible
<input type="checkbox"/>	▶ -	dctlabs.com	-	Issued	Amazon Issued	No	Ineligible

## Amazon Cognito

- Amazon Cognito lets you add user sign-up, sign-in, and access control to your web and mobile apps quickly and easily.
- Amazon Cognito provides authentication, authorization, and user management for your web and mobile apps.
- Your users can sign in directly with a user name and password, or through a third party such as Facebook, Amazon, or Google.

# Amazon Cognito



## Amazon Cognito – Web Identity Federation

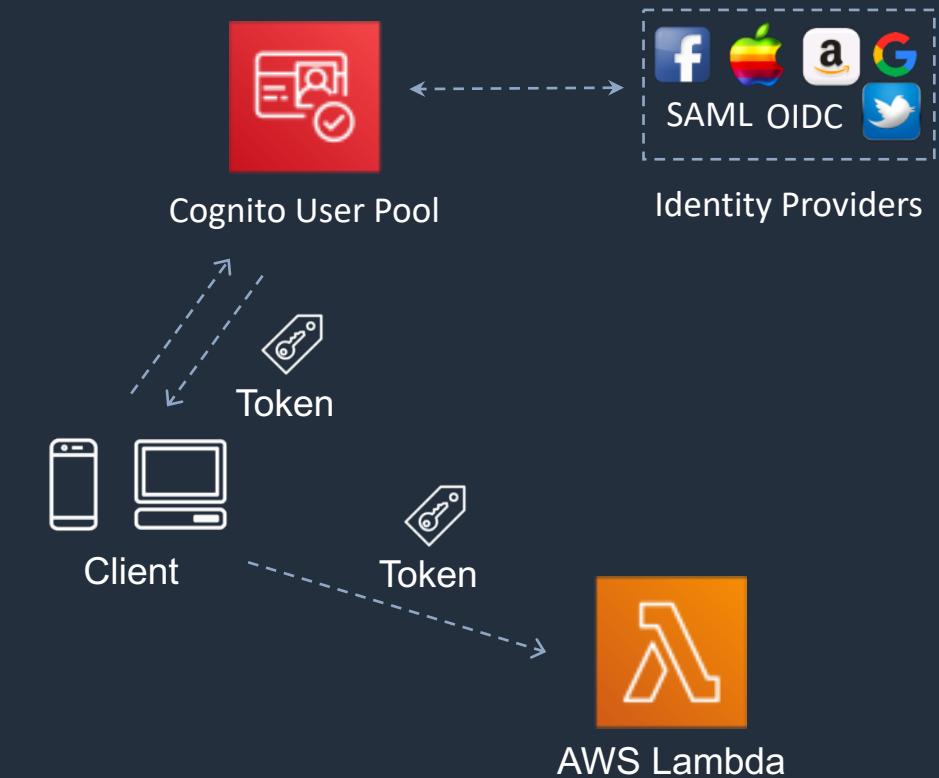
- AWS Cognito works with external identity providers that support SAML or OpenID Connect, social identity providers (such as Facebook, Twitter, Amazon).
- Federation allows users to authenticate with a Web Identity Provider (e.g. Google, Facebook, Amazon).
- The user authenticates first with the Web ID provider and receives an authentication token, which it then exchanges for temporary AWS credentials allowing them to assume an IAM role allowing access to the required resources.
- Cognito is an Identity Broker which handles interaction between your applications and the Web ID provider (you don't need to write your own code to do this).
- You can use Amazon, Facebook, Twitter, Digits, Google and any other OpenID Connect compatible identity provider.
- You can also integrate your own identity provider.

## Amazon Cognito – User Pools and Identity Pools

- The two main components of AWS Cognito are user pools and identity pools:
  - User pools are user directories that provide sign-up and sign-in options for your app users.
  - Identity pools enable you to grant your users access to other AWS services.
- You can use identity pools and user pools separately or together.
- Cognito Identity provides temporary security credentials to access your app's backend resources in AWS or any service behind Amazon API Gateway.
- Users can sign-up and sign-in using email, phone number, or user name.
- End users of an application can also sign in with SMS-based MFA.

# Amazon Cognito – User Pools

- Cognito User Pools are user directories used to manage sign-up and sign-in functionality for mobile and web applications.
- With a user pool, users can sign into your web or mobile app through Amazon Cognito.
- Users can also sign in through social identity providers like Facebook or Amazon, and through SAML identity providers.
- Cognito acts as an Identity Broker between the ID provider and AWS.
- Amazon Cognito issues JSON web tokens (JWT).

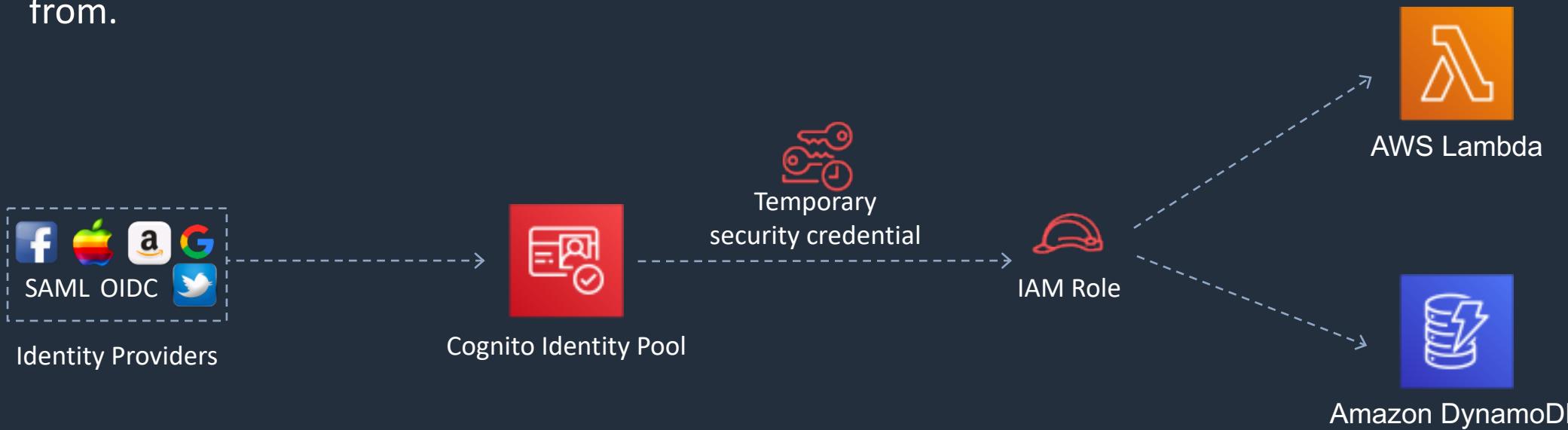


## Amazon Cognito – User Pools

- User pools provide:
  - Sign-up and sign-in services.
  - A built-in, customizable web UI to sign in users.
  - Social sign-in with Facebook, Google, and Login with Amazon, as well as sign-in with SAML identity providers from your user pool.
  - User directory management and user profiles.
  - Security features such as multi-factor authentication (MFA), checks for compromised credentials, account takeover protection, and phone and email verification.
  - Customized workflows and user migration through AWS Lambda triggers.

# Amazon Cognito – Identity Pools

- Identity Pools enable you to create unique identities for your users and authenticate them with identity providers.
- With an identity, you can obtain temporary, limited-privilege AWS credentials to access other AWS services.
- Cognito tracks the association between user identity and the various different devices they sign-in from.



## Amazon Cognito – Identity Pools

- Amazon Cognito identity pools support the following identity providers:
  - Public providers: Login with Amazon (Identity Pools), Facebook (Identity Pools), Google (Identity Pools).
  - Amazon Cognito User Pools.
  - Open ID Connect Providers (Identity Pools).
  - SAML Identity Providers (Identity Pools).
  - Developer Authenticated Identities (Identity Pools).
- **Exam tip:** To make it easier to remember the difference between User Pools and Identity Pools, think of Users Pools as being similar to IAM Users or Active Directory and Identity Pools as being similar to an IAM Role.

# THE END

Hope you enjoyed it!

