

# AWS Certified DevOps Engineer Professional

By Stéphane Maarek



COURSE →



EXTRA PRACTICE EXAMS

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

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

# Table of Contents

- [Domain 1 – SDLC Automation](#)
- [Domain 2 – Configuration Management and IaC](#)
- [Domain 3 – Resilient Cloud Solutions](#)
- [Domain 4 – Monitoring and Logging](#)
- [Domain 5 – Incident and Event Response](#)
- [Domain 6 – Security and Compliance](#)
- [Other Services](#)
- [Exam Preparation & Congratulations](#)

# AWS Certified DevOps Engineer Professional Course

## DOP-C02

# Please do not skip this lecture

- **ADVANCED, PROFESSIONAL-LEVEL COURSE**
  - Do the AWS Certified Developer course & certification at a pre-requisite
  - It'll be easier if you do the AWS Certified SysOps course & certification as well
- **EXPERIENCE PREVAILS**
  - The AWS DevOps exam is hard and tests you on real-world experience (min 2 years)
  - Some lectures are re-used from other courses ([DVA/SOA] tag for example)
- **TAKE YOUR TIME**
  - Practice as much as possible at work
  - Take notes for features or services you didn't know about
- Happy learning, and good luck for your exam!

# Domain I - SDLC Automation

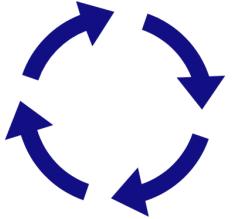
# CICD – Introduction

- We have learned how to:
  - Create AWS resources, manually (fundamentals)
  - Interact with AWS programmatically (AWS CLI)
  - Deploy code to AWS using Elastic Beanstalk
- All these manual steps make it very likely for us to do mistakes!
- We would like our code “in a repository” and have it deployed onto AWS
  - Automatically
  - The right way
  - Making sure it’s tested before being deployed
  - With possibility to go into different stages (dev, test, staging, prod)
  - With manual approval where needed
- To be a proper AWS developer... we need to learn AWS CICD

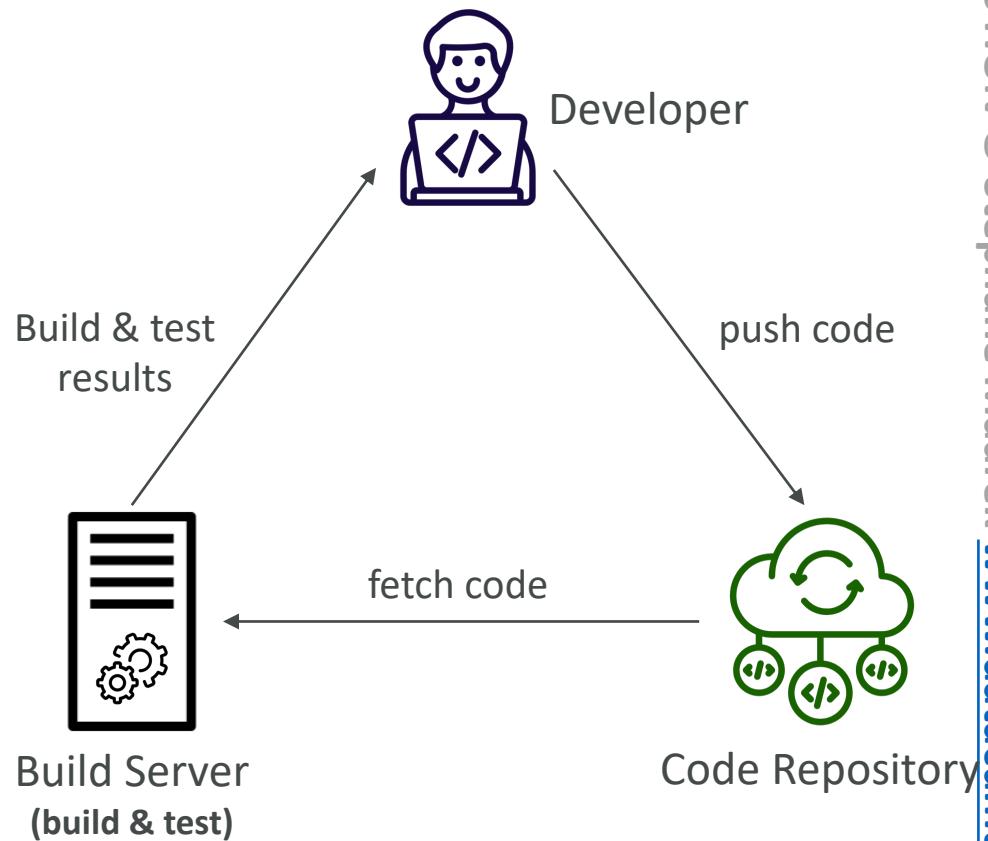
# CICD – Introduction

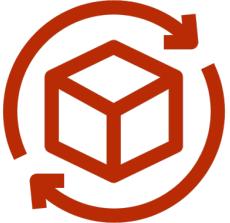
- This section is all about automating the deployment we've done so far while adding increased safety
- We'll learn about:
  - AWS CodeCommit – storing our code
  - AWS CodePipeline – automating our pipeline from code to Elastic Beanstalk
  - AWS CodeBuild – building and testing our code
  - AWS CodeDeploy – deploying the code to EC2 instances (not Elastic Beanstalk)
  - AWS CodeStar – manage software development activities in one place
  - AWS CodeArtifact – store, publish, and share software packages
  - AWS CodeGuru – automated code reviews using Machine Learning

# Continuous Integration (CI)



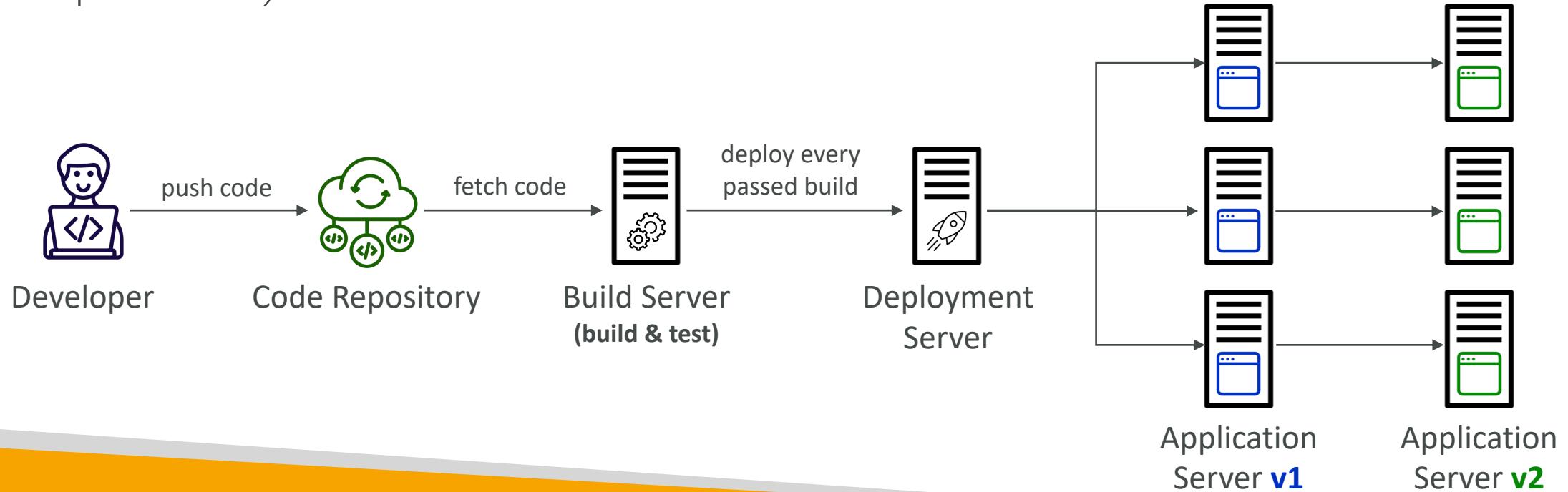
- Developers push the code to a code repository often (e.g., GitHub, CodeCommit, Bitbucket...)
  - A testing / build server checks the code as soon as it's pushed (CodeBuild, Jenkins CI...)
  - The developer gets feedback about the tests and checks that have passed / failed
- 
- Find bugs early, then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



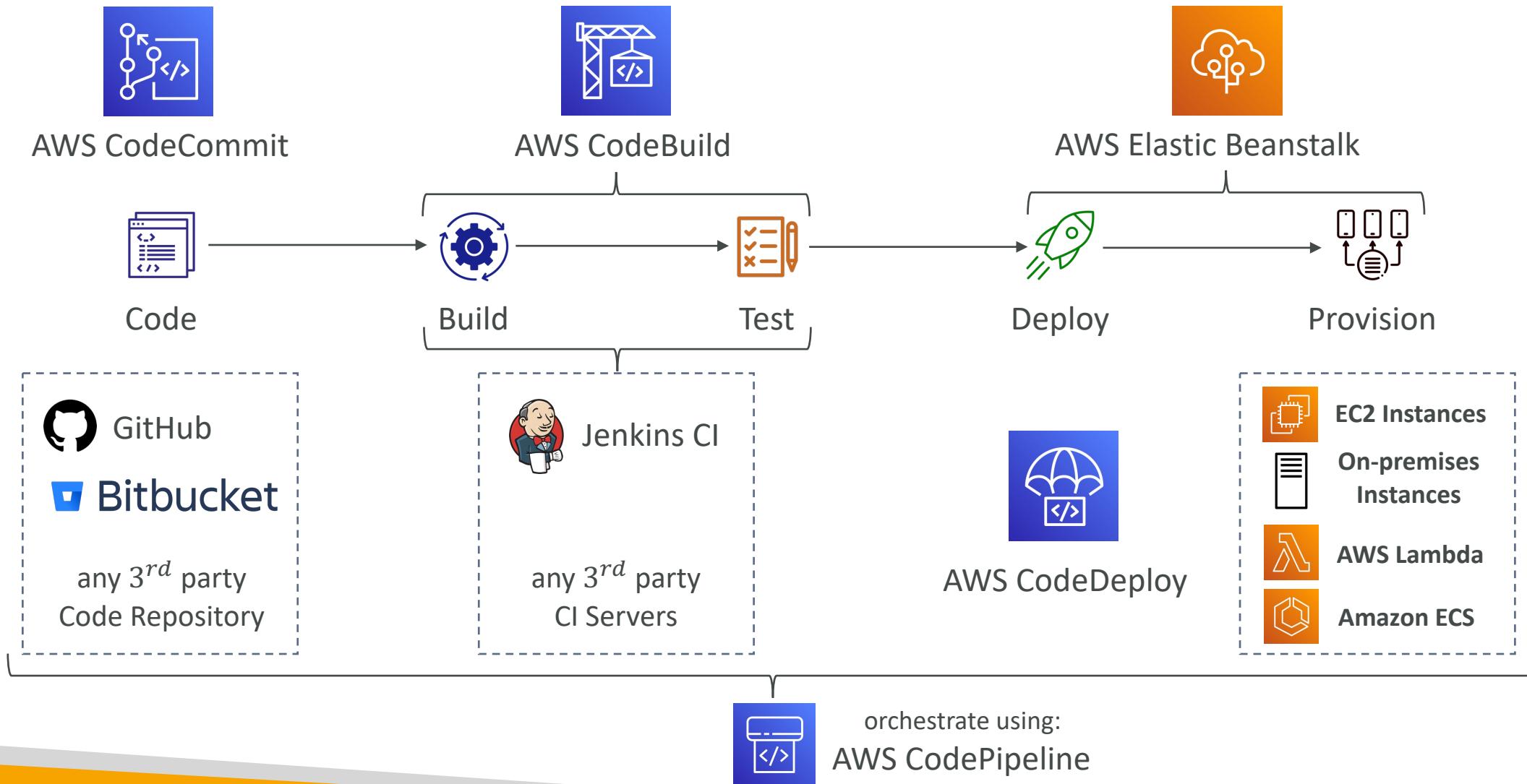


# Continuous Delivery (CD)

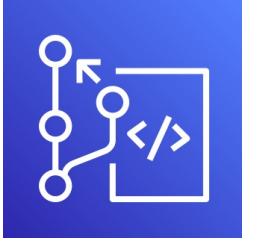
- Ensures that the software can be released reliably whenever needed
- Ensures deployments happen often and are quick
- Shift away from “one release every 3 months” to “5 releases a day”
- That usually means automated deployment (e.g., CodeDeploy, Jenkins CD, Spinnaker...)



# Technology Stack for CICD



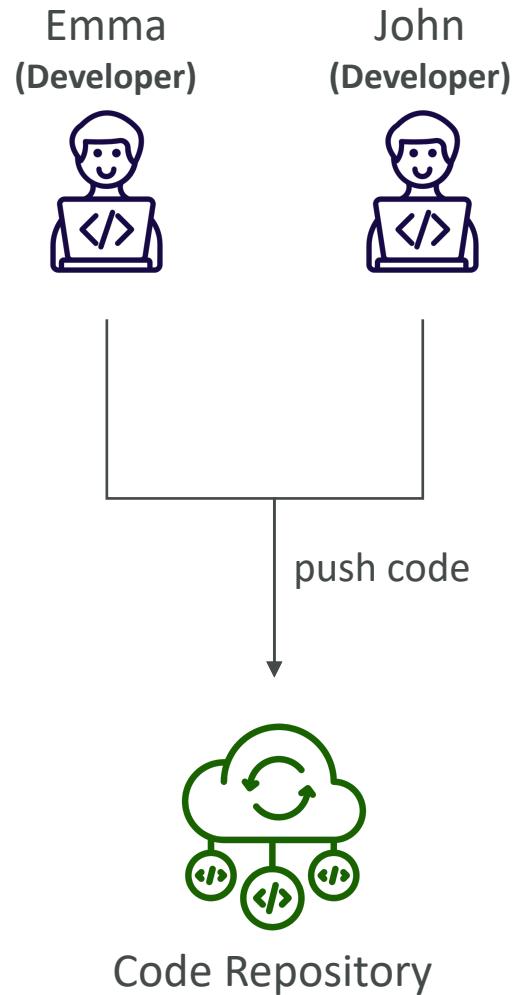
# AWS CodeCommit



- **Version control** is the ability to understand the various changes that happened to the code over time (and possibly roll back)
- All these are enabled by using a version control system such as **Git**
- A Git repository can be synchronized on your computer, but it usually is uploaded on a central online repository
- Benefits are:
  - Collaborate with other developers
  - Make sure the code is backed-up somewhere
  - Make sure it's fully viewable and auditable

# AWS CodeCommit

- Git repositories can be expensive
- The industry includes GitHub, GitLab, Bitbucket...
- And **AWS CodeCommit**:
  - Private Git repositories
  - No size limit on repositories (scale seamlessly)
  - Fully managed, highly available
  - Code only in AWS Cloud account => increased security and compliance
  - Security (encrypted, access control...)
  - Integrated with Jenkins, AWS CodeBuild, and other CI tools



# CodeCommit – Security

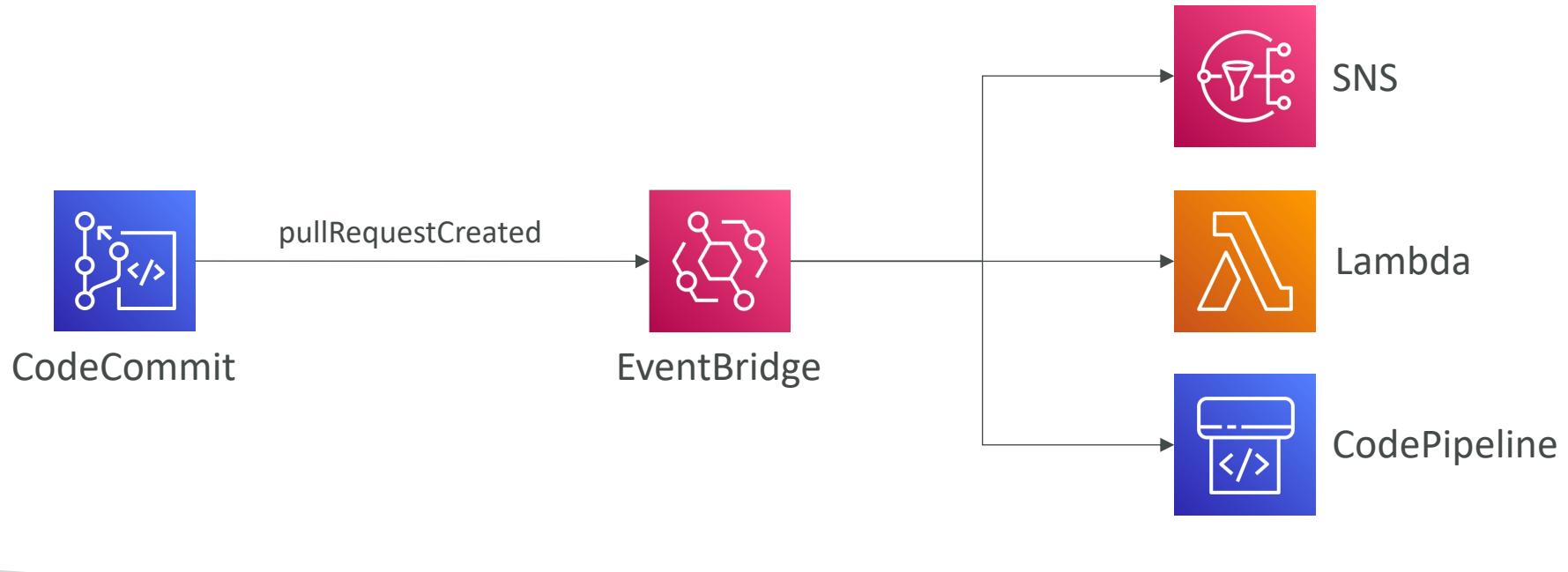
- Interactions are done using Git (standard)
- **Authentication**
  - SSH Keys – AWS Users can configure SSH keys in their IAM Console
  - HTTPS – with AWS CLI Credential helper or Git Credentials for IAM user
- **Authorization**
  - IAM policies to manage users/roles permissions to repositories
- **Encryption**
  - Repositories are automatically encrypted at rest using AWS KMS
  - Encrypted in transit (can only use HTTPS or SSH – both secure)
- **Cross-account Access**
  - Do NOT share your SSH keys or your AWS credentials
  - Use an IAM Role in your AWS account and use AWS STS (AssumeRole API)

# CodeCommit vs. GitHub

	CodeCommit	GitHub
Support Code Review (Pull Requests)	✓	✓
Integration with AWS CodeBuild	✓	✓
Authentication (SSH & HTTPS)	✓	✓
Security	IAM Users & Roles	GitHub Users
Hosting	Managed & hosted by AWS	<ul style="list-style-type: none"><li>- Hosted by GitHub</li><li>- GitHub Enterprise: self hosted on your servers</li></ul>
UI	Minimal	Fully Featured

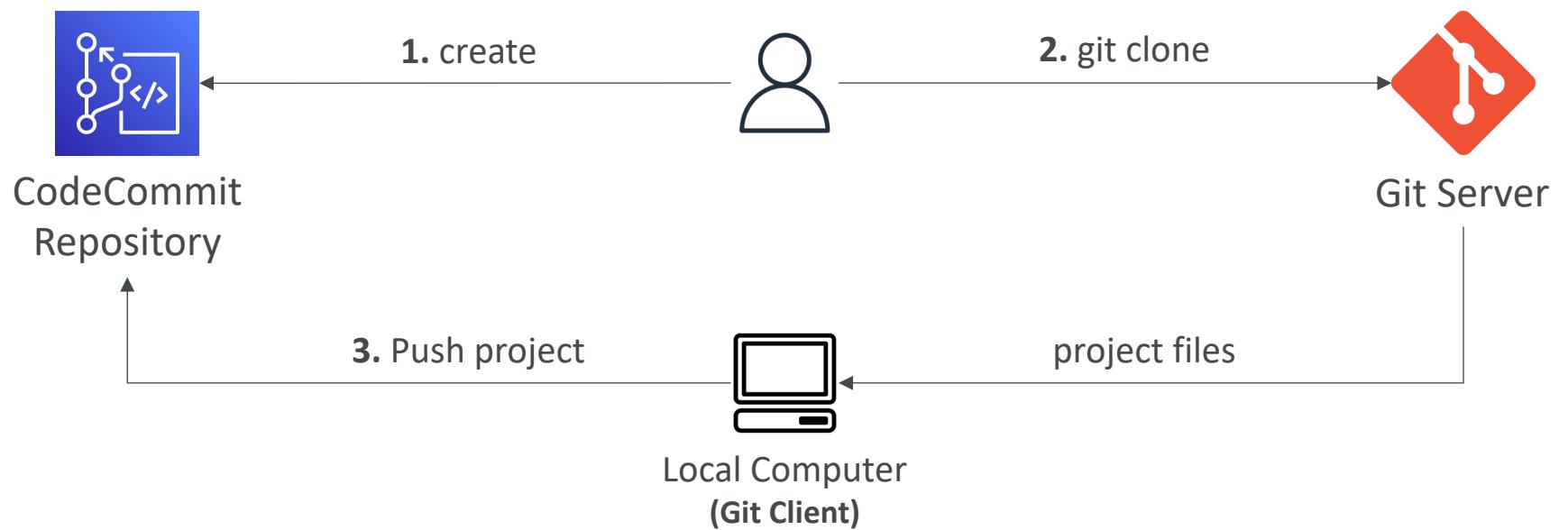
# CodeCommit – Monitoring with EventBridge

- You can monitor CodeCommit events in EventBridge (near real-time)
- pullRequestCreated, pullRequestStatusChanged, referenceCreated, commentOnCommitCreated...



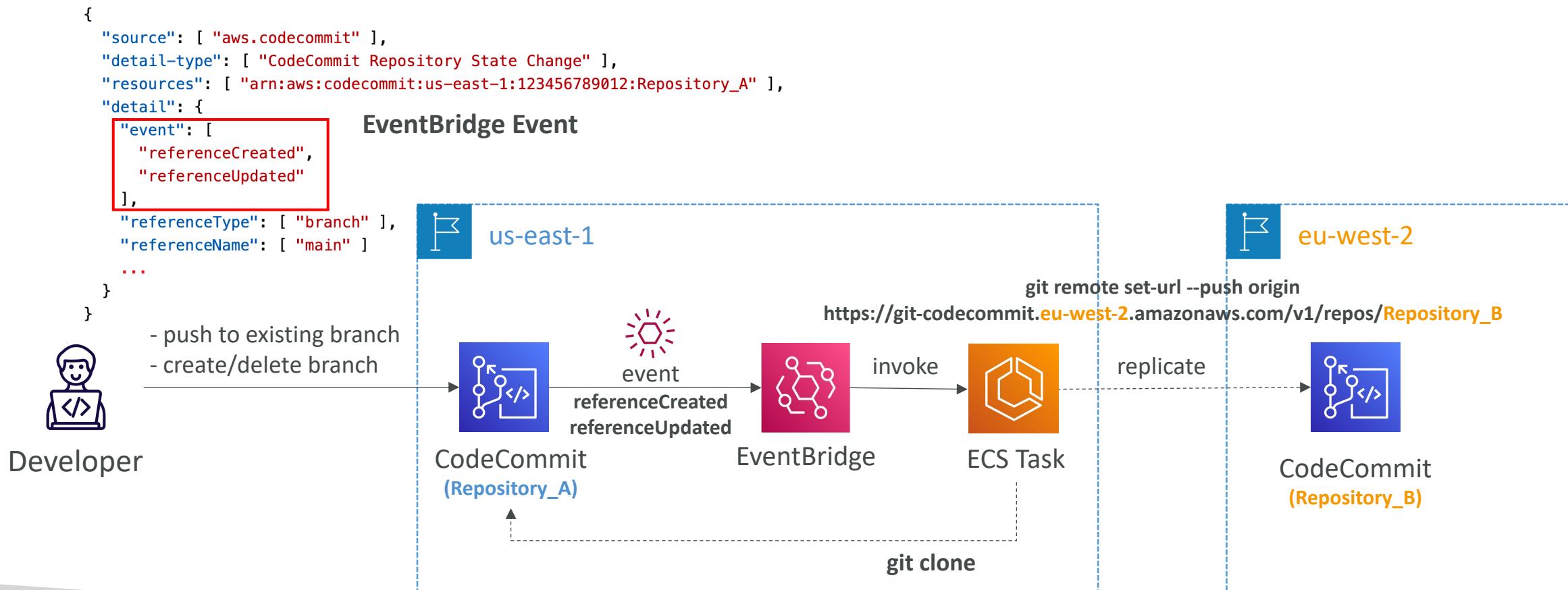
# Migrate Git Repository to CodeCommit

- You can migrate a project hosted on another Git repository (e.g., Github, GitLab...) to CodeCommit repository



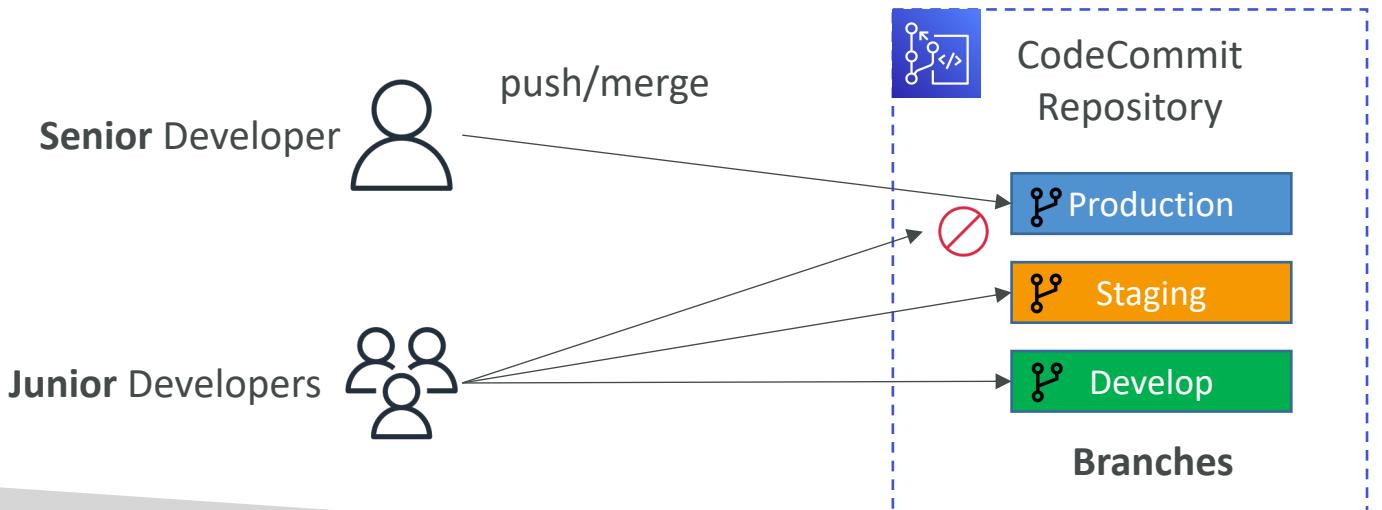
# CodeCommit – Cross-Region Replication

- Use case: achieve lower latency pulls for global developers, backups...



# CodeCommit – Branch Security

- By default, a user who has push permissions to a CodeCommit repository can contribute to any branch
- Use IAM policies to restrict users to push or merge code to a specific branch
- Example: only senior developers can push to production branch
- Note: Resource Policy is not supported yet

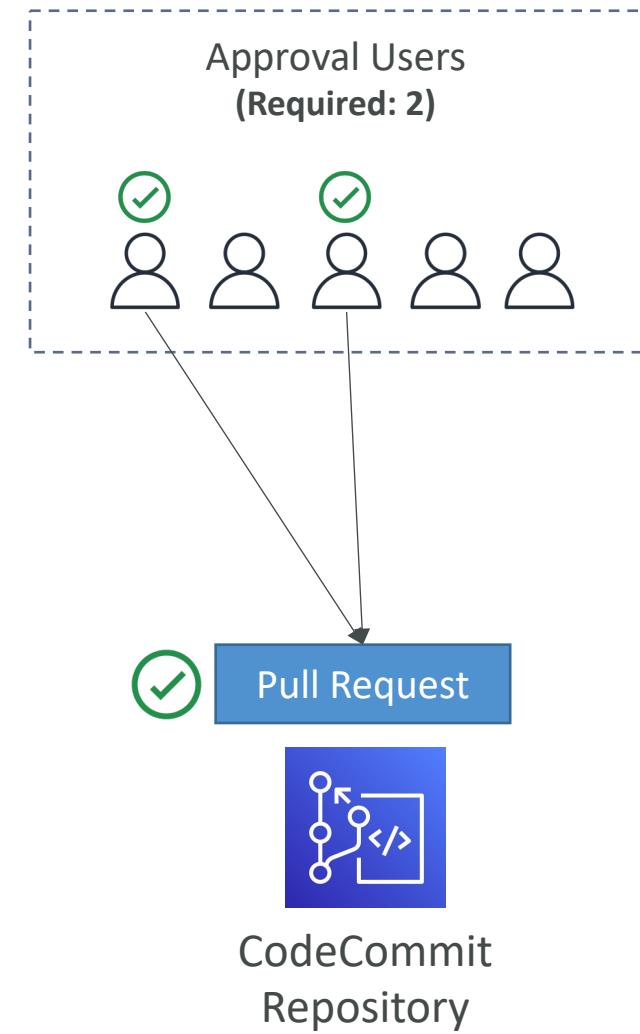


```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "codecommit:GitPush",
        "codecommit>DeleteBranch",
        "codecommit:PutFile",
        "codecommit:MergeBranchesByFastForward",
        "codecommit:MergeBranchesBySquash",
        "codecommit:MergeBranchesByThreeWay",
        "codecommit:MergePullRequestByFastForward",
        "codecommit:MergePullRequestBySquash",
        "codecommit:MergePullRequestByThreeWay"
      ],
      "Resource": "arn:aws:codecommit:us-east-2:123456789012:DemoRepo",
      "Condition": {
        "StringEqualsIfExists": [
          "codecommit:References": [
            "refs/heads/main",
            "refs/heads/prod"
          ]
        ],
        "Null": {
          "codecommit:References": "false"
        }
      }
    }
  ]
}
```

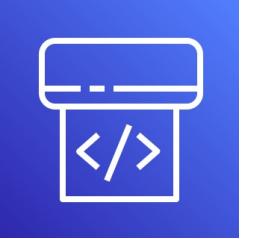
**IAM Policy**

# CodeCommit – Pull Request Approval Rules

- Helps ensure the quality of your code by requiring user(s) to approve the open PRs before the code can be merged
- Specify a pool of users to approve and number of users who must approve the PR
- Specify IAM Principal ARN (IAM users, federated users, IAM Roles, IAM Groups)
- Approval Rule Templates
  - Automatically apply Approval Rules to PRs in specific repositories
  - Example: define different rules for dev and prod branches

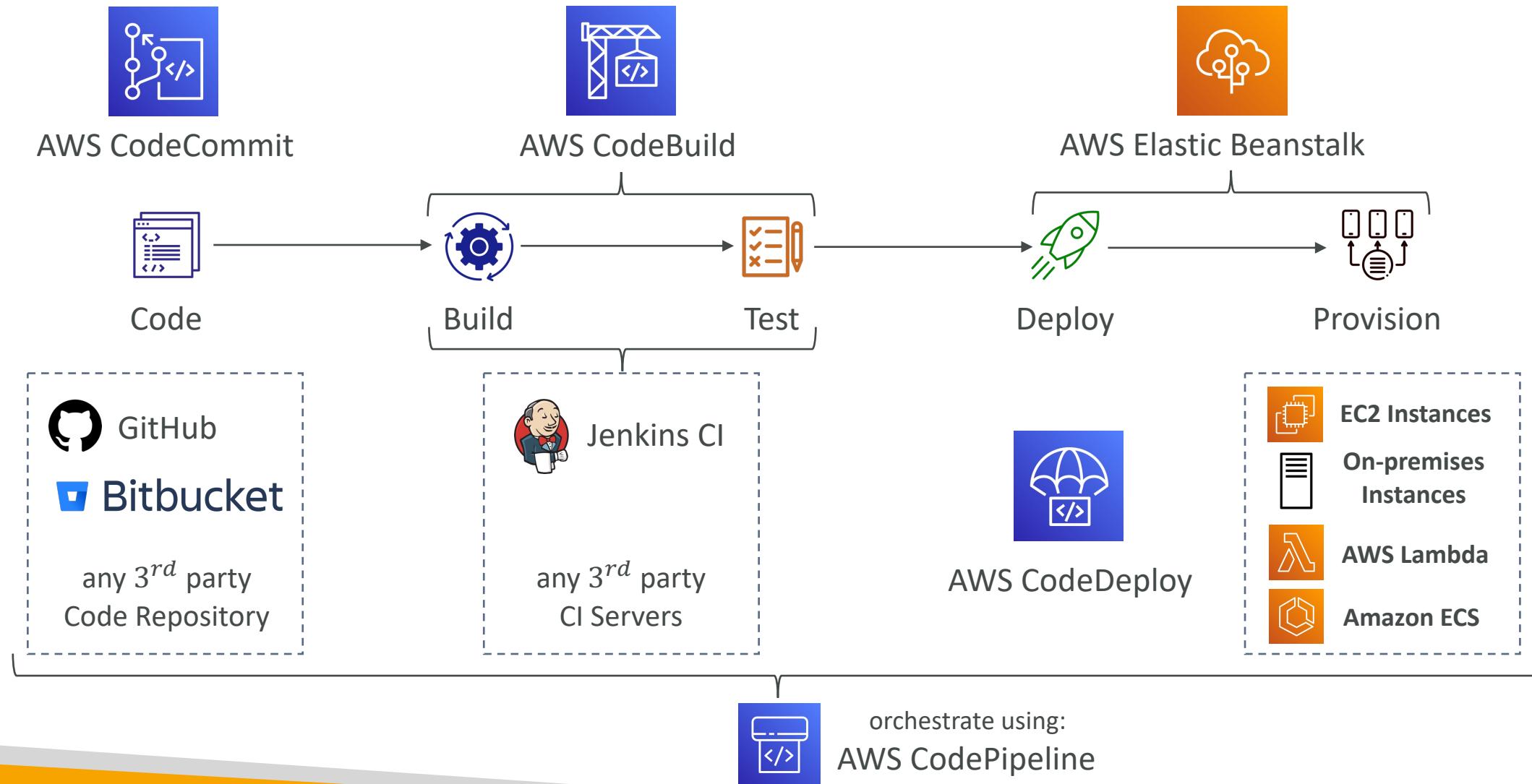


# AWS CodePipeline



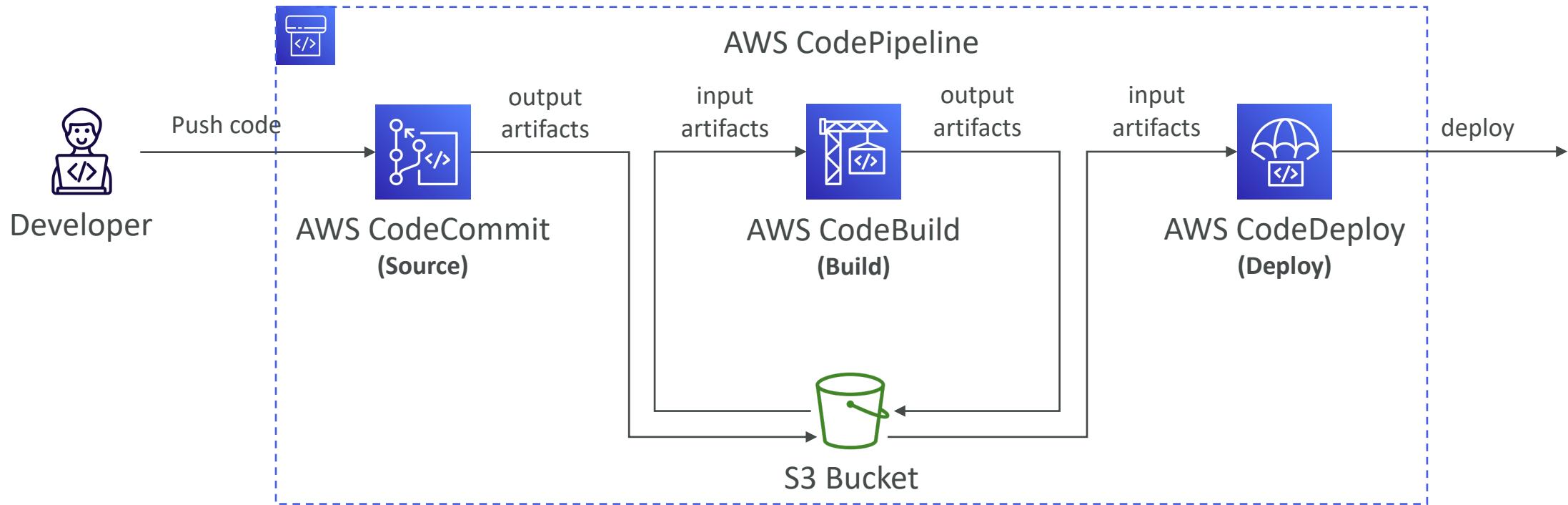
- Visual Workflow to orchestrate your CICD
- **Source** – CodeCommit, ECR, S3, Bitbucket, GitHub
- **Build** – CodeBuild, Jenkins, CloudBees, TeamCity
- **Test** – CodeBuild, AWS Device Farm, 3<sup>rd</sup> party tools...
- **Deploy** – CodeDeploy, Elastic Beanstalk, CloudFormation, ECS, S3...
- **Invoke** – Lambda, Step Functions
- Consists of stages:
  - Each stage can have sequential actions and/or parallel actions
  - Example: Build → Test → Deploy → Load Testing → ...
  - Manual approval can be defined at any stage

# Technology Stack for CICD



# CodePipeline – Artifacts

- Each pipeline stage can create artifacts
- Artifacts stored in an S3 bucket and passed on to the next stage

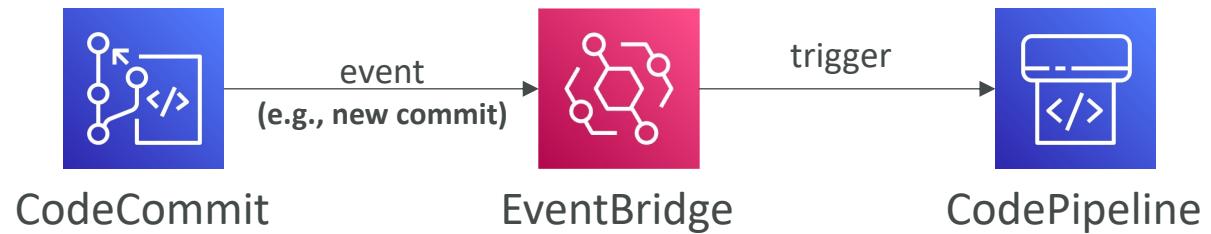


# CodePipeline – Troubleshooting

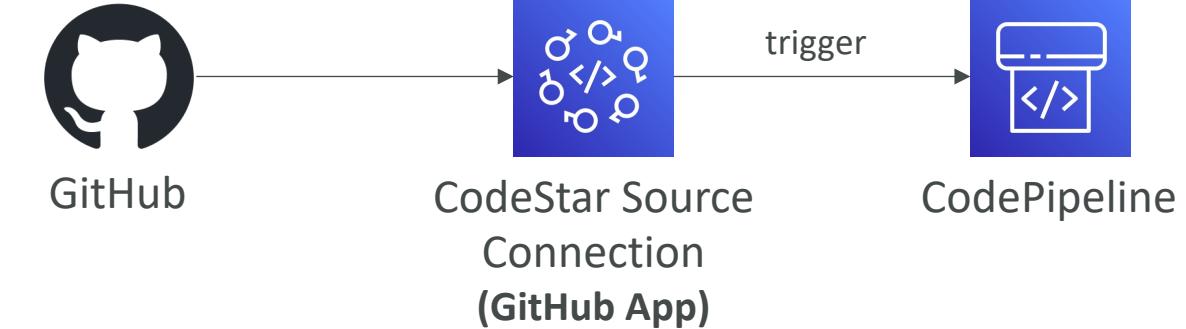
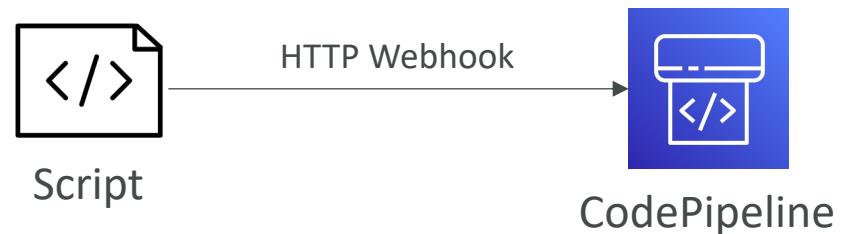
- For CodePipeline Pipeline/Action/Stage Execution State Changes
- Use **CloudWatch Events (Amazon EventBridge)**. Example:
  - You can create events for failed pipelines
  - You can create events for cancelled stages
- If CodePipeline fails a stage, your pipeline stops, and you can get information in the console
- If pipeline can't perform an action, make sure the "IAM Service Role" attached does have enough IAM permissions (IAM Policy)
- AWS CloudTrail can be used to audit AWS API calls

# CodePipeline – Events vs. Webhooks vs. Polling

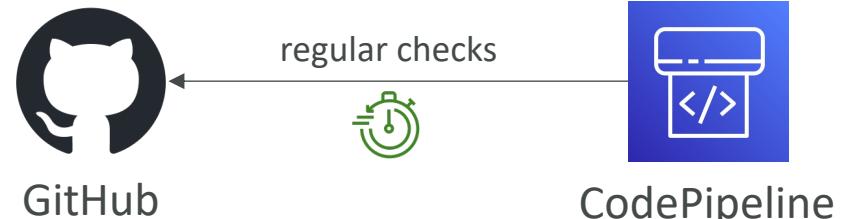
## Events



## Webhooks



## Polling



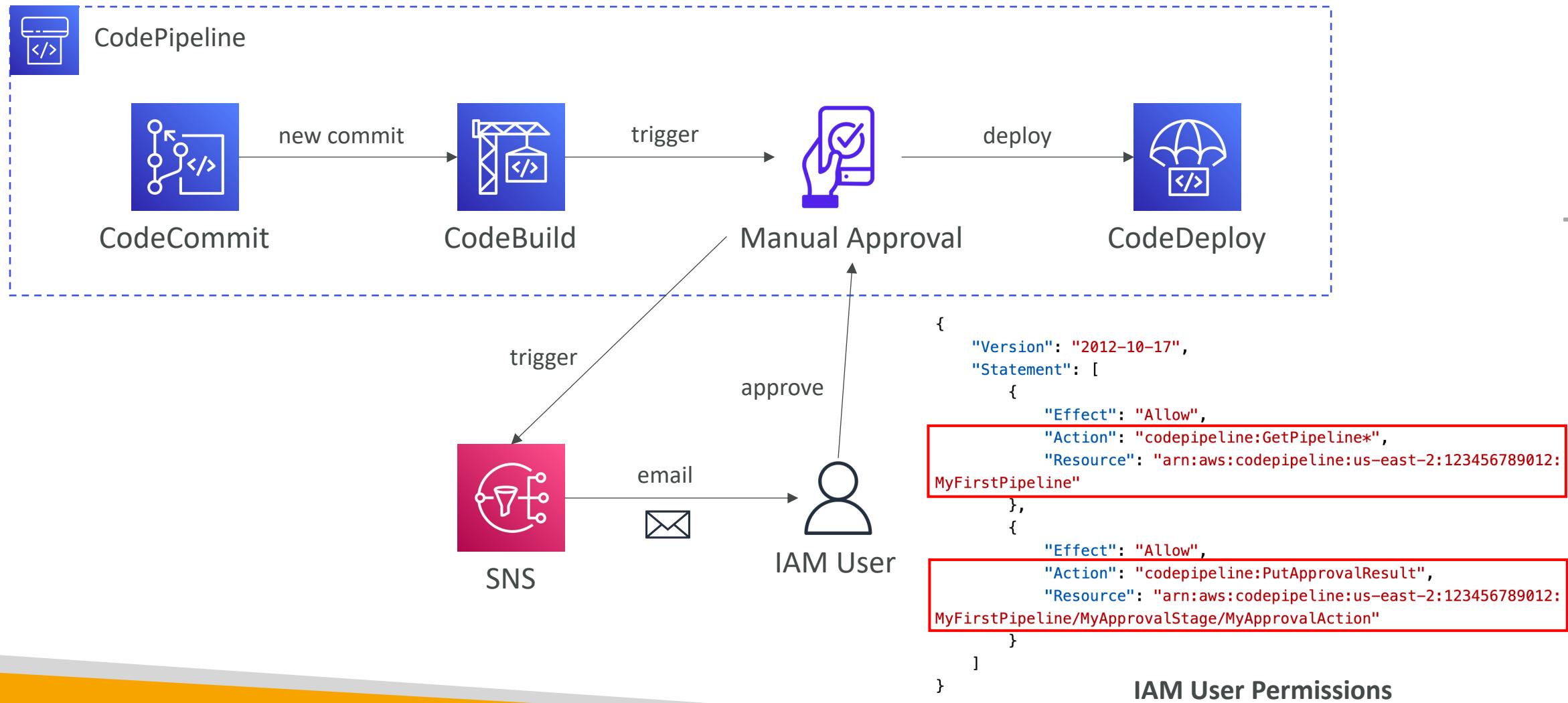
**Note:** Events are the default and recommended

# CodePipeline – Action Types Constraints for Artifacts

- Owner
  - AWS – for AWS services
  - 3<sup>rd</sup> Party – GitHub or Alexa Skills Kit
  - Custom – Jenkins
- Action Type
  - Source – S3, ECR, GitHub...
  - Build – CodeBuild, Jenkins
  - Test – CodeBuild, Device Farm, Jenkins
  - Approval – Manual
  - Invoke – Lambda, Step Functions
  - Deploy – S3, CloudFormation, CodeDeploy, Elastic Beanstalk, OpsWorks, ECS, Service Catalog...

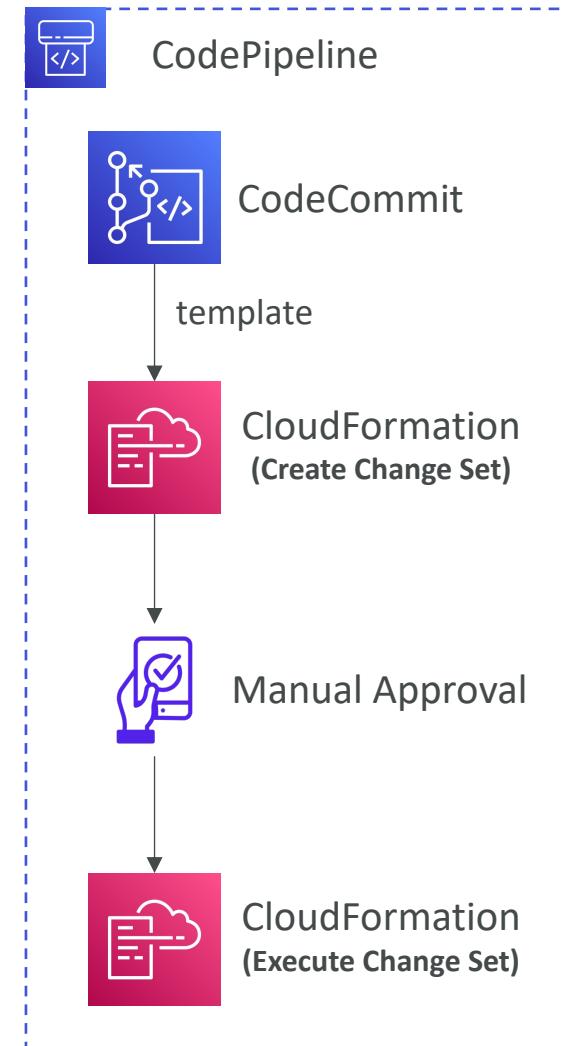
Owner	Action Type	Provider	Valid Number of Input Artifacts	Valid Number of Output Artifacts
AWS	Source	S3	0	1
AWS	Source	CodeCommit	0	1
AWS	Source	ECR	0	1
3 <sup>rd</sup> Party	Source	Github	0	1
AWS	Build	Codebuild	1 to 5	0 to 5
AWS	Test	CodeBuild	1 to 5	0 to 5
AWS	Test	Device Farm	1	0
AWS	Approval	Manual	0	0
AWS	Deploy	S3	1	0
AWS	Deploy	CloudFormation	0 to 10	0 to 1
AWS	Deploy	CodeDeploy	1	0
AWS	Deploy	Elastic Beanstalk	1	0
AWS	Deploy	OpsWorks Stacks	1	0
AWS	Deploy	ECS	1	0
AWS	Deploy	Service Catalog	1	0
AWS	Invoke	Lambda	0 to 5	0 to 5
AWS	Invoke	Step Functions	0 to 1	0 to 1
3 <sup>rd</sup> Party	Deploy	Alexa Skills Kit	1 to 2	0
Custom	Build	Jenkins	0 to 5	0 to 5
Custom	Test	Jenkins	0 to 5	0 to 5
Custom	Any Suggested Category	Specified in Custom Action	0 to 5	

# CodePipeline – Manual Approval Stage



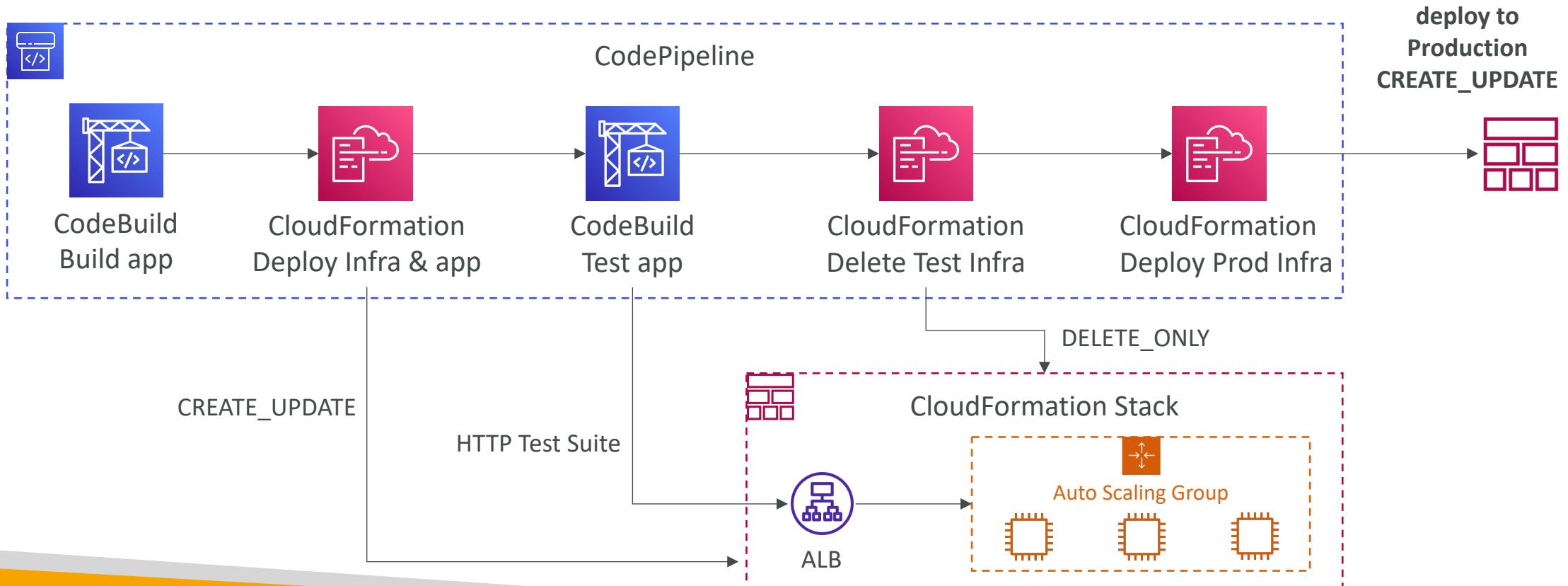
# CodePipeline – CloudFormation as a Target

- CloudFormation Deploy Action can be used to deploy AWS resources
- Example: deploy Lambda functions using CDK or SAM (alternative to CodeDeploy)
- Works with CloudFormation StackSets to deploy across multiple AWS accounts and AWS Regions
- Configure different settings:
  - Stack name, Change Set name, template, parameters, IAM Role, Action Mode...



# CodePipeline – CloudFormation Integration

- **CREATE\_UPDATE** – create or update an existing stack
- **DELETE\_ONLY** – delete a stack if it exists



# CodePipeline – CloudFormation as a Target

- Action Modes

- Create or Replace a Change Set, Execute a Change Set
- Create or Update a Stack, Delete a Stack, Replace a Failed Stack

- Template Parameter Overrides

- Specify a JSON object to override parameter values
- Retrieves the parameter value from CodePipeline Input Artifact
- All parameter names must be present in the template
- **Static** – use template configuration file (recommended)
- **Dynamic** – use parameter overrides

```
{  
  "ParameterName": {  
    "Fn::GetParam": [  
      "ArtifactName",  
      "config-file-name.json",  
      "ParamName"  
    ]  
  }  
}
```

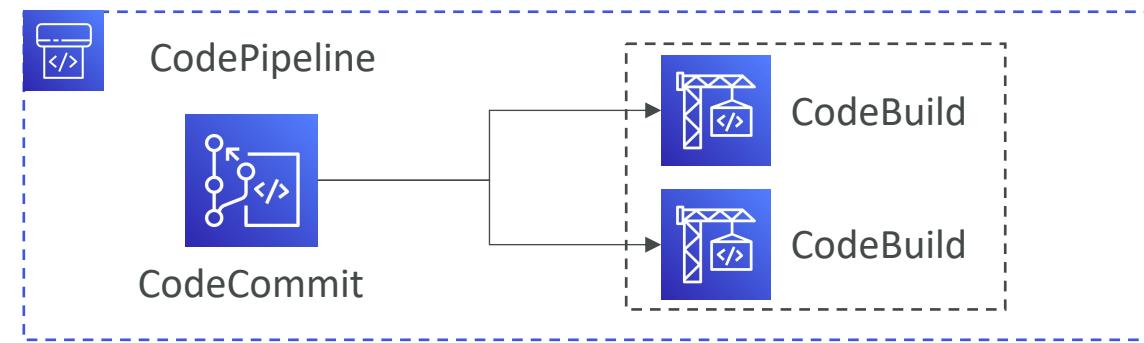
Retrieves **ParamName** parameter value from  
CodePipeline input artifact **config-file-name.json**

# CodePipeline – Best Practices

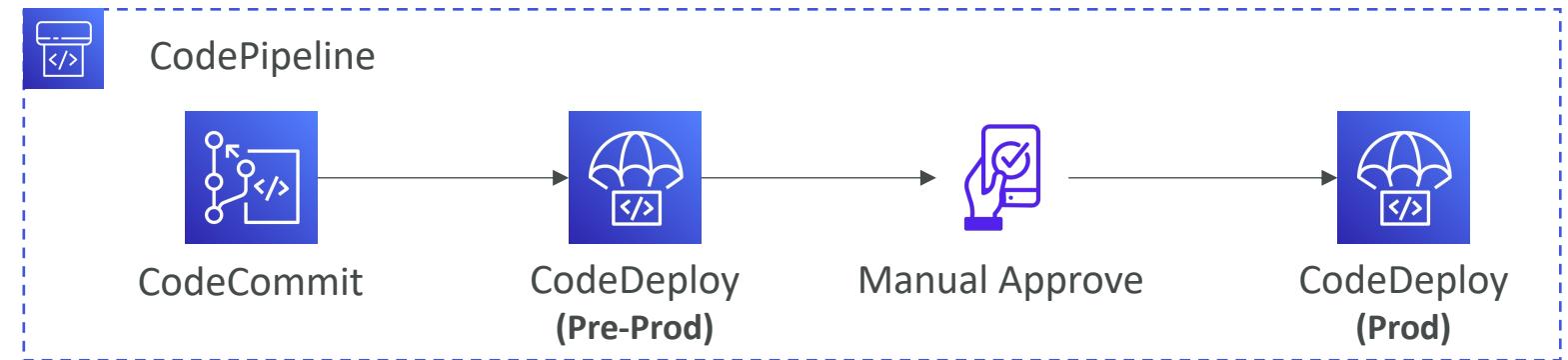
One CodePipeline, One CodeDeploy,  
Parallel deploy to multiple  
Deployment Groups



Parallel Actions using in  
a Stage using *RunOrder*

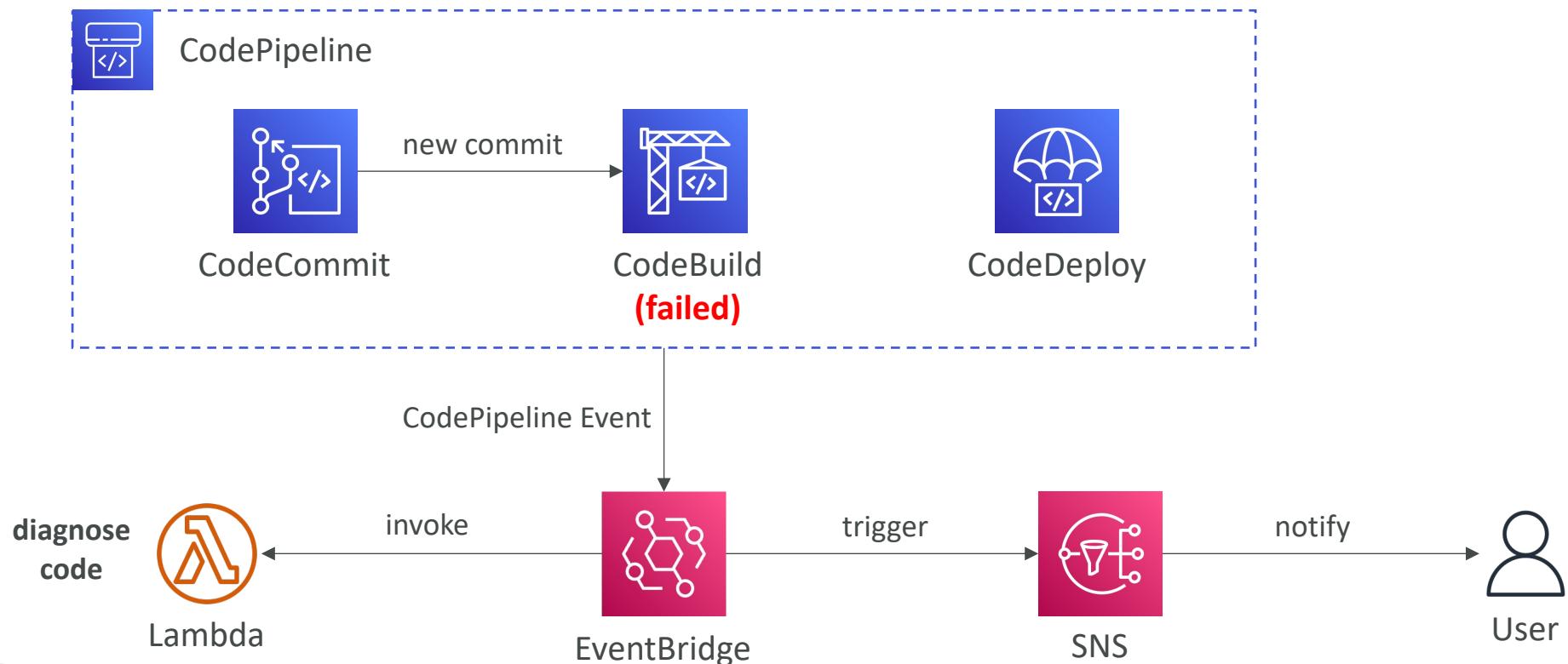


Deploy to Pre-Prod before  
Deploying to Prod



# CodePipeline & EventBridge

- EventBridge – detect and react to changes in execution states (e.g., intercept failures at certain stages)

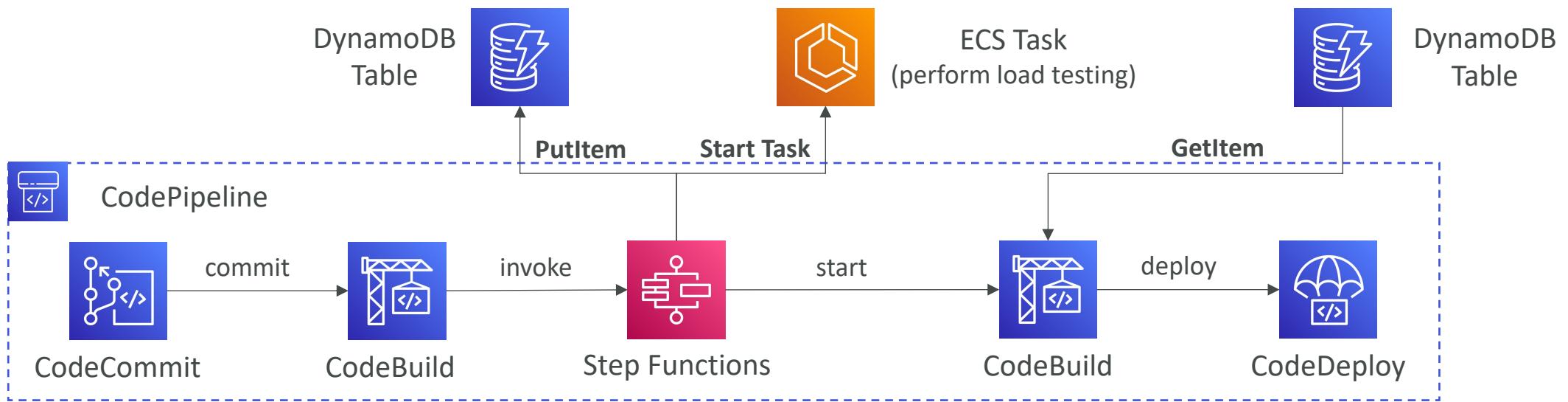


# CodePipeline – Invoke Action

- Lambda – invokes a Lambda function within a Pipeline



- Step Functions – starts a State Machine within a Pipeline

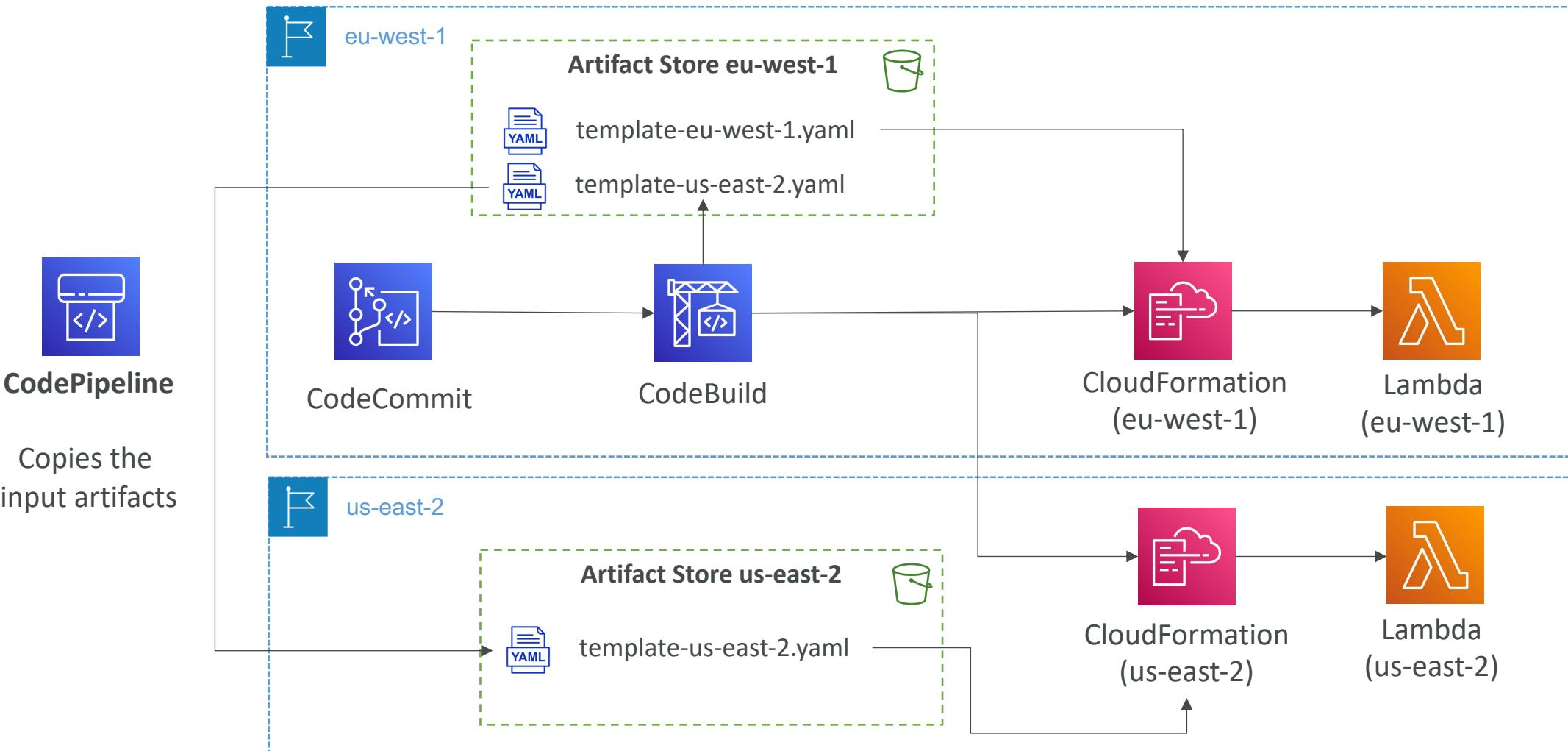


# CodePipeline – Multi Region

- Actions in your pipeline can be in different regions
  - Example: deploy a Lambda function through CloudFormation into multiple regions
- S3 Artifact Stores must be defined in each region where you have actions
  - CodePipeline must have read/write access into every artifact buckets
  - If you use the console default artifact buckets are configured, else you must create them
- CodePipeline handles the copying of input artifacts from one AWS Region to the other Regions when performing cross-region actions
  - In your cross-region actions, only reference the name of the input artifacts

▼ Artifact store		
Region	Type	Location
eu-west-1	S3	<a href="#">codepipeline-eu-west-1-460886087071</a>
eu-central-1	S3	<a href="#">codepipeline-eu-central-1-625328630265</a>

# CodePipeline – CloudFormation Multi Region



# AWS CodeBuild



- A fully managed continuous integration (CI) service
- Continuous scaling (no servers to manage or provision – no build queue)
- Compile source code, run tests, produce software packages...
- Alternative to other build tools (e.g., Jenkins)
- Charged per minute for compute resources (time it takes to complete the builds)
- Leverages Docker under the hood for reproducible builds
- Use prepackaged Docker images or create your own custom Docker image
- Security:
  - Integration with KMS for encryption of build artifacts
  - IAM for CodeBuild permissions, and VPC for network security
  - AWS CloudTrail for API calls logging

# AWS CodeBuild

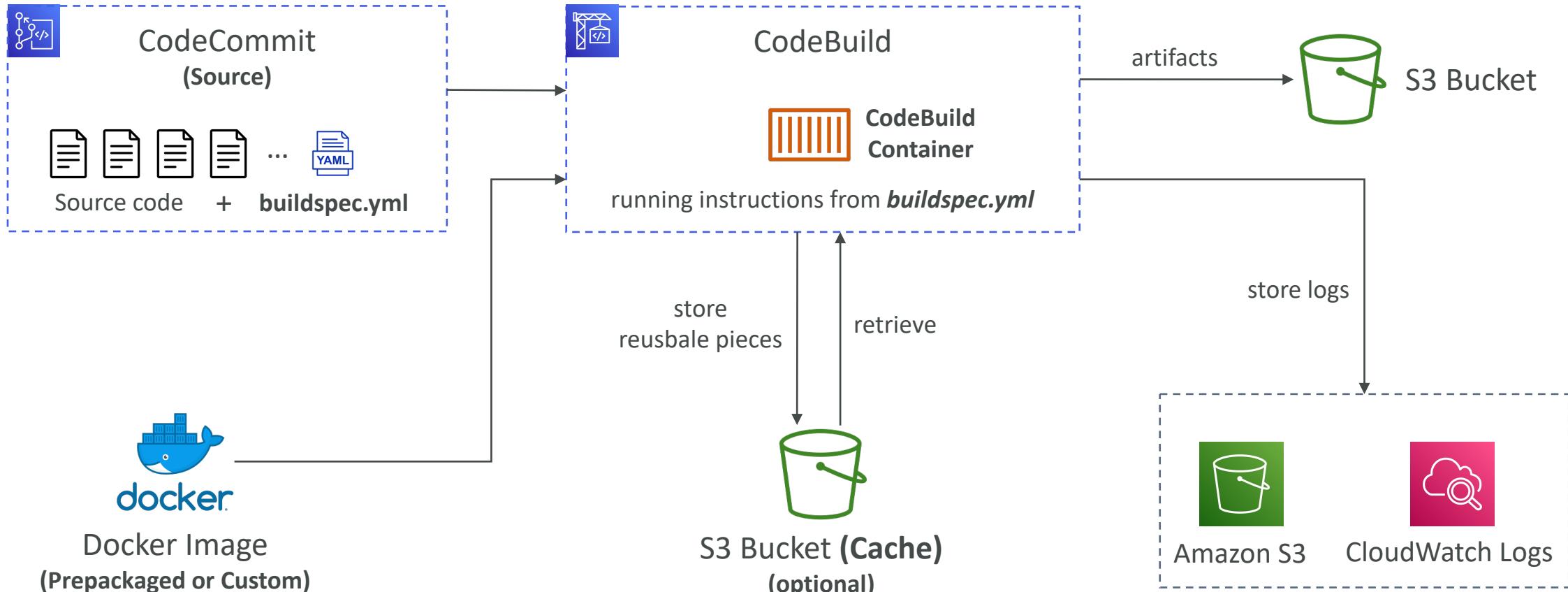


- Source – CodeCommit, S3, Bitbucket, GitHub
- Build instructions: Code file **buildspec.yml** or insert manually in Console
- Output logs can be stored in Amazon S3 & CloudWatch Logs
- Use CloudWatch Metrics to monitor build statistics
- Use EventBridge to detect failed builds and trigger notifications
- Use CloudWatch Alarms to notify if you need “thresholds” for failures
- Build Projects can be defined within CodePipeline or CodeBuild

# CodeBuild – Supported Environments

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

# CodeBuild – How it Works



# CodeBuild – buildspec.yml

- **buildspec.yml** file must be at the **root** of your code
- **env** – define environment variables
  - **variables** – plaintext variables
  - **parameter-store** – variables stored in SSM Parameter Store
  - **secrets-manager** – variables stored in AWS Secrets Manager
- **phases** – specify commands to run:
  - **install** – install dependencies you may need for your build
  - **pre\_build** – final commands to execute before build
  - **Build** – actual **build commands**
  - **post\_build** – finishing touches (e.g., zip output)
- **artifacts** – what to upload to S3 (encrypted with KMS)
- **cache** – files to cache (usually dependencies) to S3 for future build speedup

```
version: 0.2

env:
  variables:
    JAVA_HOME: "/usr/lib/jvm/java-8-openjdk-amd64"
  parameter-store:
    LOGIN_PASSWORD: /CodeBuild/dockerLoginPassword

phases:
  install:
    commands:
      - echo "Entered the install phase..."
      - apt-get update -y
      - apt-get install -y maven
  pre_build:
    commands:
      - echo "Entered the pre_build phase..."
      - docker login -u User -p $LOGIN_PASSWORD
  build:
    commands:
      - echo "Entered the build phase..."
      - echo "Build started on `date`"
      - mvn install
  post_build:
    commands:
      - echo "Entered the post_build phase..."
      - echo "Build completed on `date`"

artifacts:
  files:
    - target/messageUtil-1.0.jar

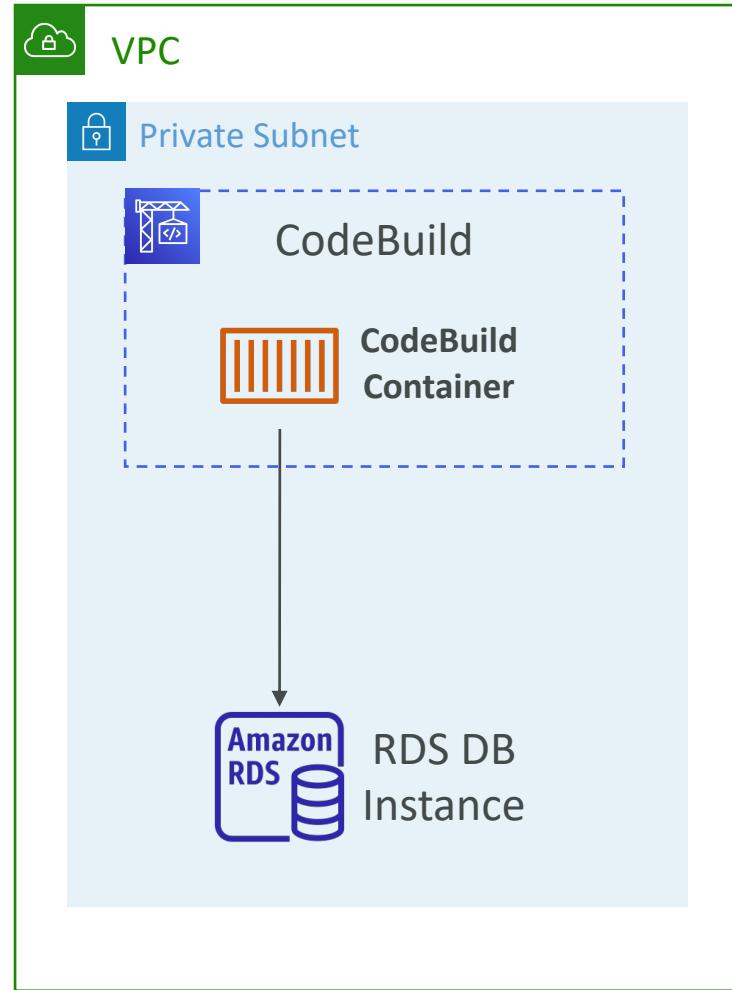
cache:
  paths:
    - "/root/.m2/**/*"
```

# CodeBuild – Local Build

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

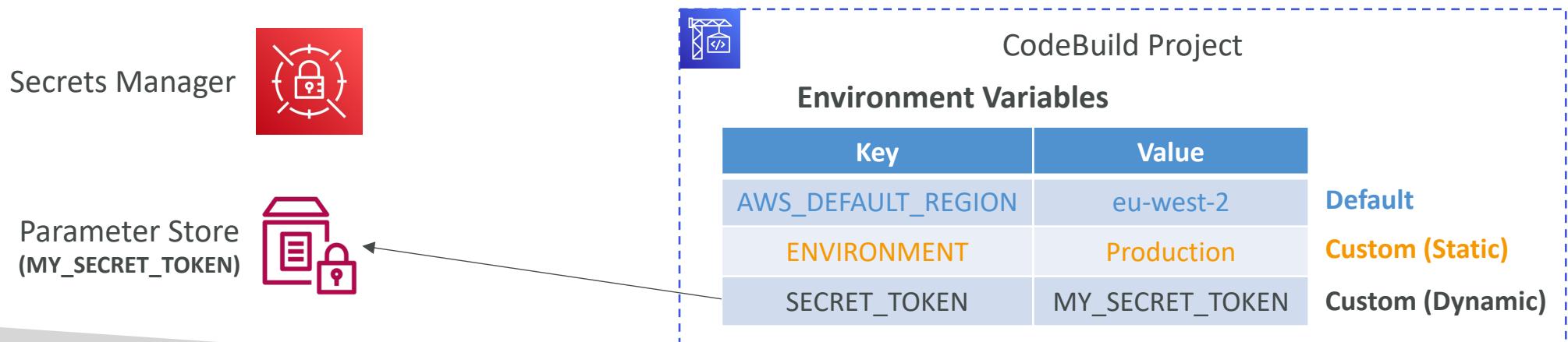
# CodeBuild – Inside VPC

- By default, your CodeBuild containers are launched outside your VPC
  - It cannot access resources in a VPC
- You can specify a VPC configuration:
  - VPC ID
  - Subnet IDs
  - Security Group IDs
- Then your build can access resources in your VPC (e.g., RDS, ElastiCache, EC2, ALB...)
- Use cases: integration tests, data query, internal load balancers...



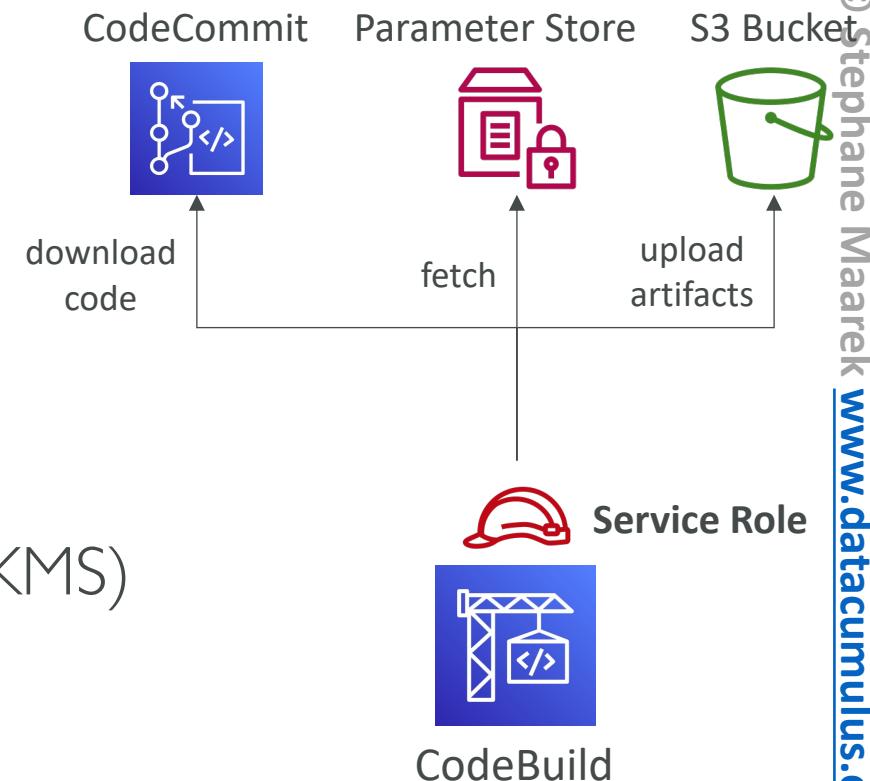
# CodeBuild – Environment Variables

- Default Environment Variables
  - Defined and provided by AWS
  - AWS\_DEFAULT\_REGION, CODEBUILD\_BUILD\_ARN, CODEBUILD\_BUILD\_ID, CODEBUILD\_BUILD\_IMAGE...
- Custom Environment Variables
  - Static – defined at build time (override using start-build API call)
  - Dynamic – using SSM Parameter Store and Secrets Manager

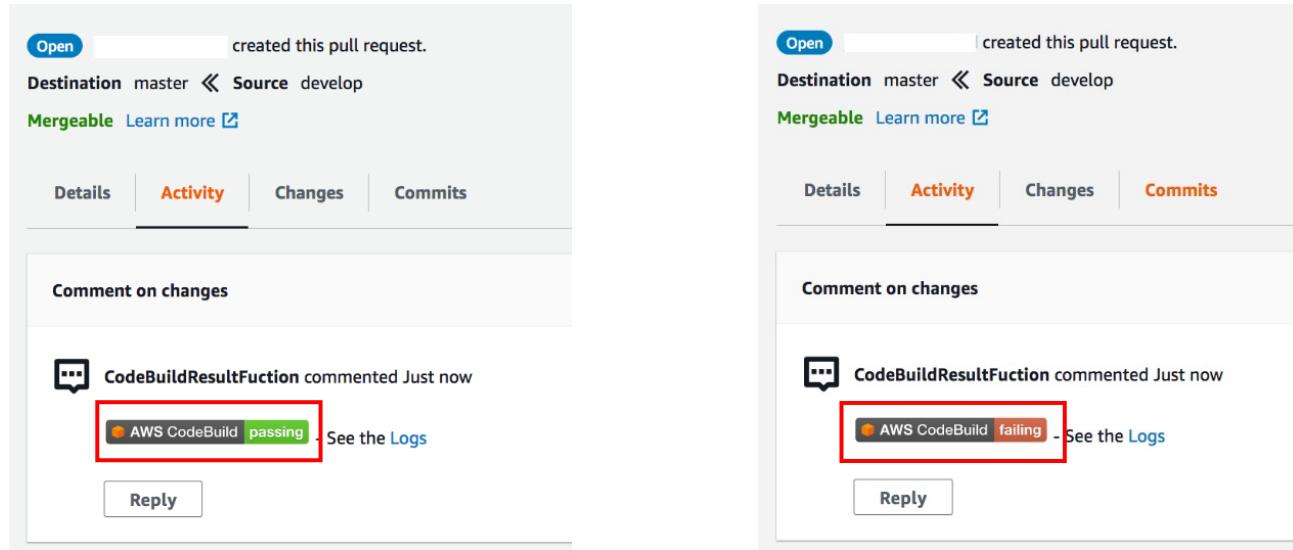


# CodeBuild – Security

- **CodeBuild Service Role** allows CodeBuild to access AWS resources on your behalf (assign the required permissions)
- Use cases:
  - Download code from CodeCommit repository
  - Fetch parameters from SSM Parameter Store
  - Upload build artifacts to S3 bucket
  - Fetch secrets from Secrets Manager
  - Store logs in CloudWatch Logs
- In-transit and at-rest data encryption (cache, logs...)
- Build output artifact encryption (requires access to KMS)

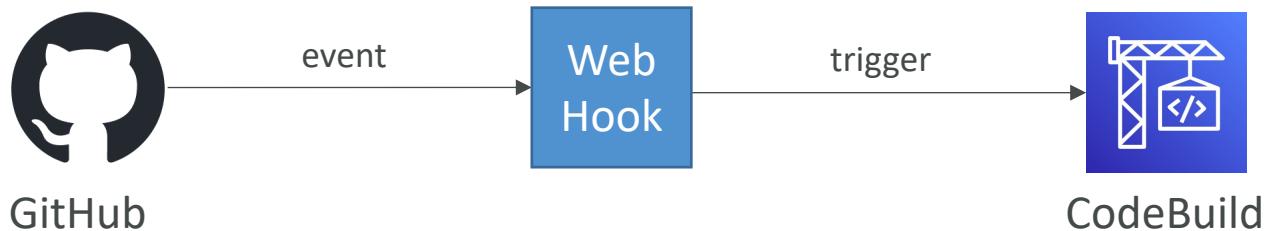
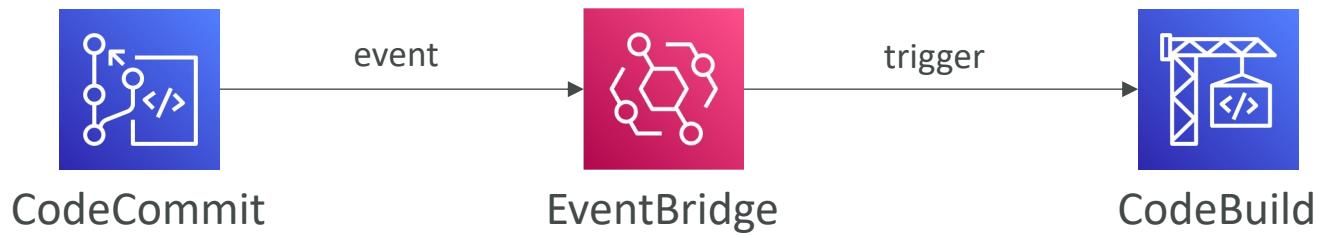


# CodeBuild – Build Badges



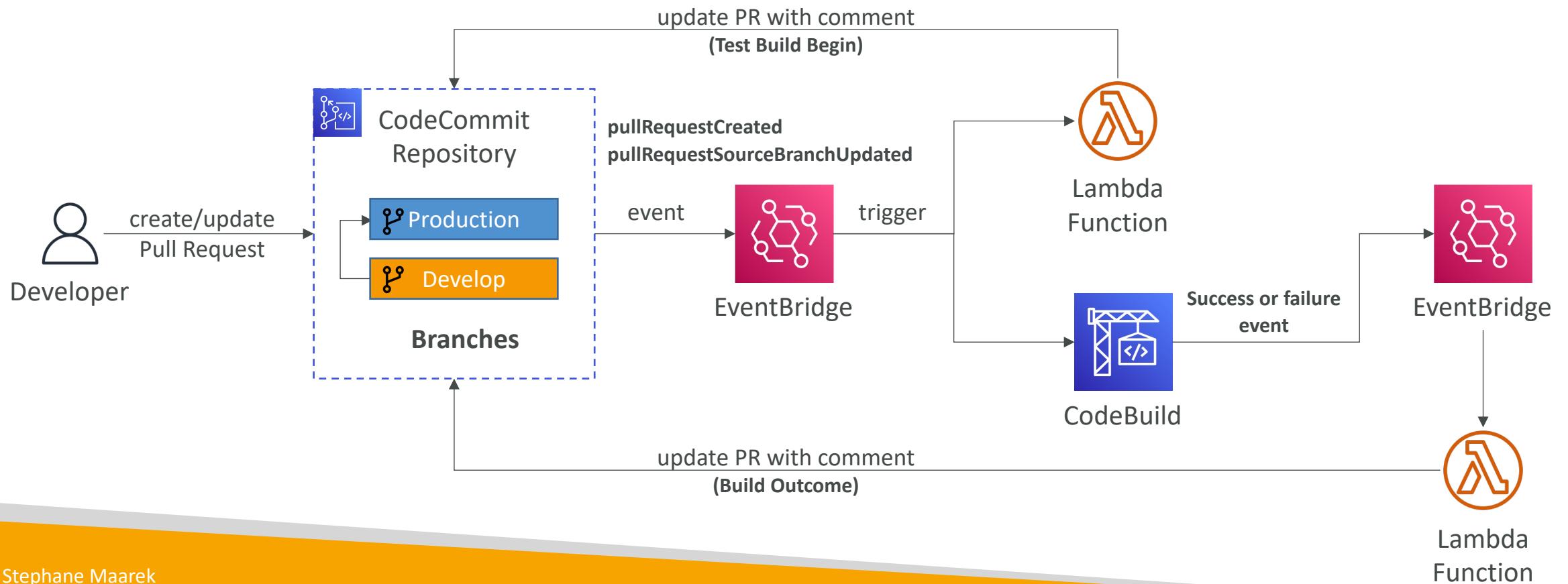
- Dynamically generated badge that displays the status of the latest build
- Can be accessed through a public URL for your CodeBuild project
- Supported for CodeCommit, GitHub, and BitBucket
- Note: Badges are available at the branch level

# CodeBuild – Triggers



# CodeBuild – Validate Pull Requests

- Validate proposed code changes in PRs before they get merged
- Ensure high level of code quality and avoid code conflicts



# CodeBuild – Validate Pull Requests

8: My awesome change

[Open](#) [ ] created this pull request.

Destination master << Source develop

Mergeable [Learn more](#)

[Close pull request](#) [Merge](#)

Details Activity Changes Commits

Comment on changes

CodeBuildResultFuction commented Just now

AWS CodeBuild failing - See the [Logs](#)

[Reply](#)

Comment on changes

PullRequestFunction commented 1 minute ago

Build started at 2018-12-22 22:30:40.033771

[Reply](#)

This screenshot shows a pull request interface with two main sections. The top section displays basic information: the title '8: My awesome change', a 'Mergeable' status, and a 'Merge' button. Below this are tabs for 'Details', 'Activity' (which is currently selected), 'Changes', and 'Commits'. The 'Activity' tab contains two comments. The first comment is from 'CodeBuildResultFuction' stating 'AWS CodeBuild failing - See the Logs'. The second comment is from 'PullRequestFunction' stating 'Build started at 2018-12-22 22:30:40.033771'. Both comments have a 'Reply' button below them. The bottom section is a large, empty white area labeled 'Comment on changes'.

Build Failed

8: My awesome change

[Open](#) [ ] created this pull request.

Destination master << Source develop

Mergeable [Learn more](#)

[Close pull request](#) [Merge](#)

Details Activity Changes Commits

Comment on changes

CodeBuildResultFuction commented Just now

AWS CodeBuild passing - See the [Logs](#)

[Reply](#)

Comment on changes

PullRequestFunction commented Just now

Build started at 2018-12-22 22:33:01.110539

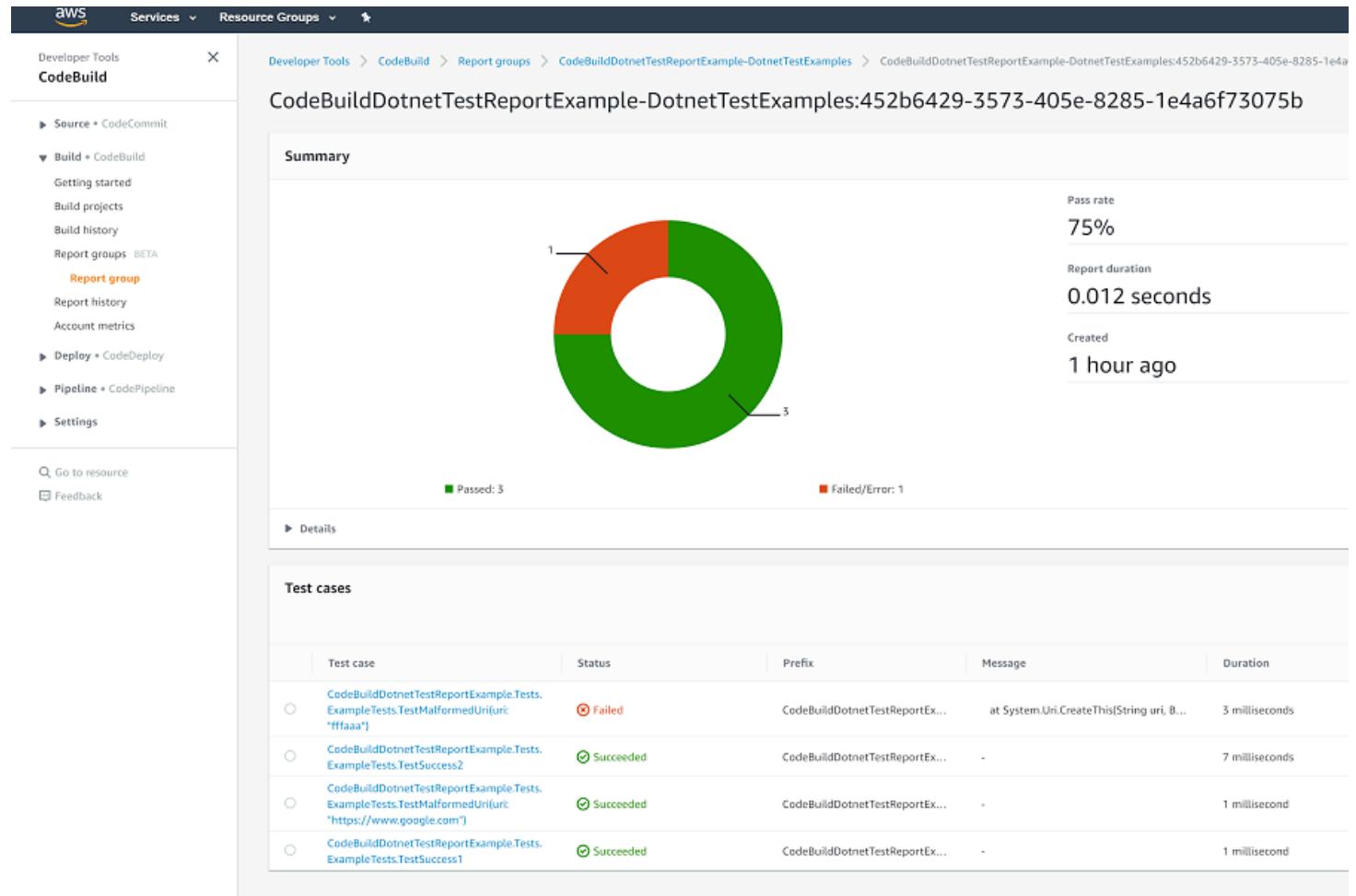
[Reply](#)

This screenshot shows a pull request interface with two main sections. The top section displays basic information: the title '8: My awesome change', a 'Mergeable' status, and a 'Merge' button. Below this are tabs for 'Details', 'Activity' (which is currently selected), 'Changes', and 'Commits'. The 'Activity' tab contains two comments. The first comment is from 'CodeBuildResultFuction' stating 'AWS CodeBuild passing - See the Logs'. The second comment is from 'PullRequestFunction' stating 'Build started at 2018-12-22 22:33:01.110539'. Both comments have a 'Reply' button below them. The bottom section is a large, empty white area labeled 'Comment on changes'.

Build Success

<https://aws.amazon.com/blogs/devops/validating-aws-codecommit-pull-requests-with-aws-codebuild-and-aws-lambda/>

# CodeBuild – Test Reports



# CodeBuild – Test Reports

- Contains details about tests that are run during builds
- **Unit tests, configuration tests, functional tests**
- Create your test cases with any test framework that can create report files in the following format:
  - JUnit XML, NUnit XML, NUnit3 XML
  - Cucumber JSON, TestNG XML, Visual Studio TRX
- Create a test report and add a **Report Group** name in *buildspec.yml* file with information about your tests

```
reports:          Report Group
  php-reports:
    files:
      - "reports/php/*.xml"
    file-format: "JUNITXML"

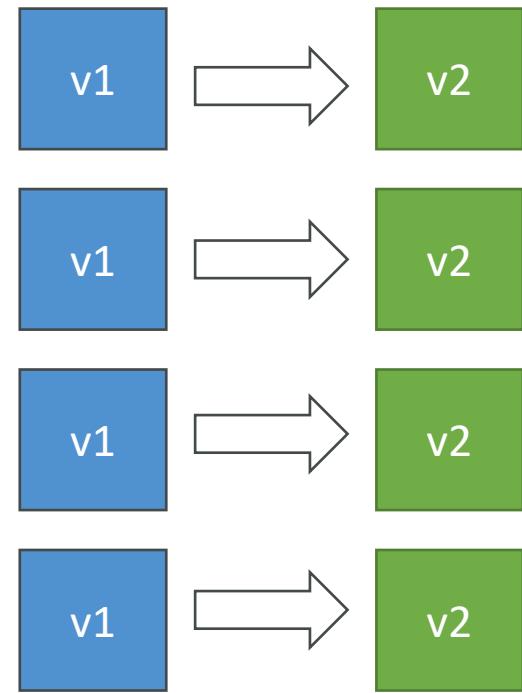
  nunit-reports:
    files:
      - "reports/nunit/*.xml"
    file-format: "NUNITXML"

buildspec.yml
```

# AWS CodeDeploy



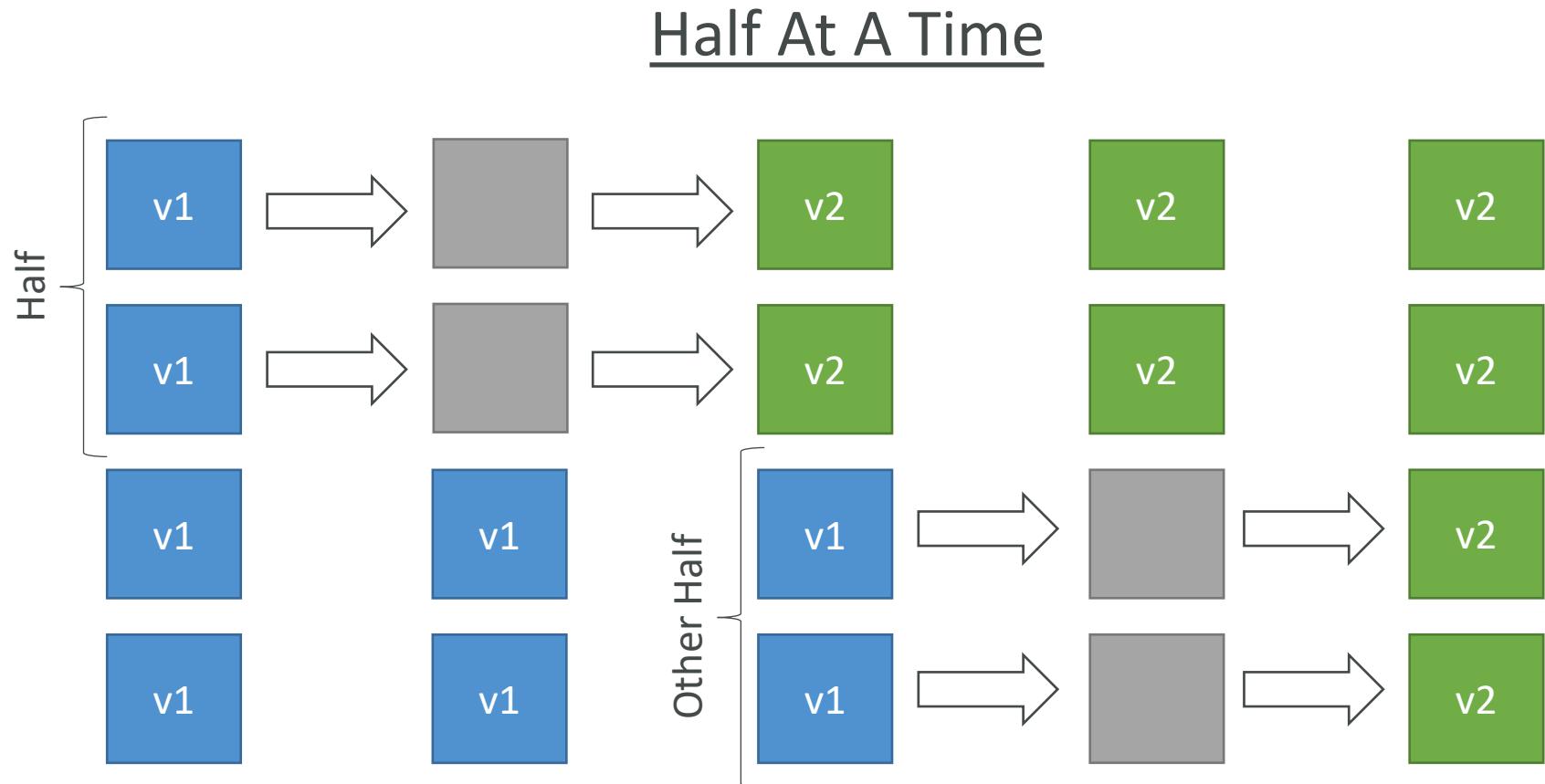
- Deployment service that automates application deployment
- Deploy new applications versions to EC2 Instances, On-premises servers, Lambda functions, ECS Services
- Automated Rollback capability in case of failed deployments, or trigger CloudWatch Alarm
- Gradual deployment control
- A file named **appspec.yml** defines how the deployment happens



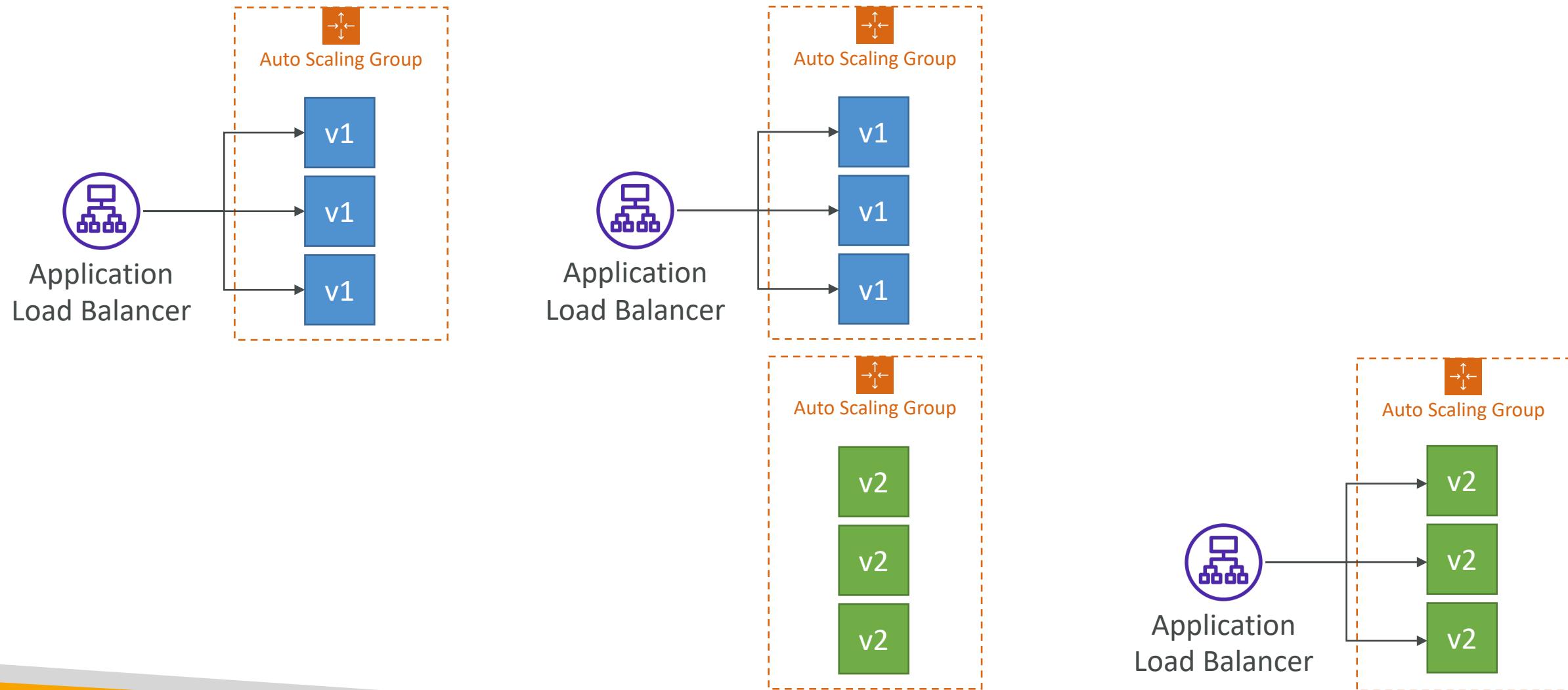
# CodeDeploy – EC2/On-premises Platform

- Can deploy to EC2 Instances & on-premises servers
- Perform in-place deployments or blue/green deployments
- Must run the **CodeDeploy Agent** on the target instances
- Define deployment speed
  - AllAtOnce: most downtime
  - HalfAtATime: reduced capacity by 50%
  - OneAtATime: slowest, lowest availability impact
  - Custom: define your %

# CodeDeploy – In-Place Deployment



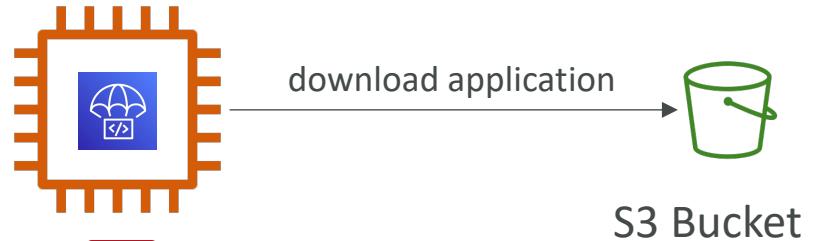
# CodeDeploy – Blue-Green Deployment



# CodeDeploy Agent

- The CodeDeploy Agent must be running on the EC2 instances as a prerequisites
- It can be installed and updated automatically if you're using Systems Manager
- The EC2 Instances must have sufficient permissions to access Amazon S3 to get deployment bundles

## EC2 Instance With CodeDeploy Agent

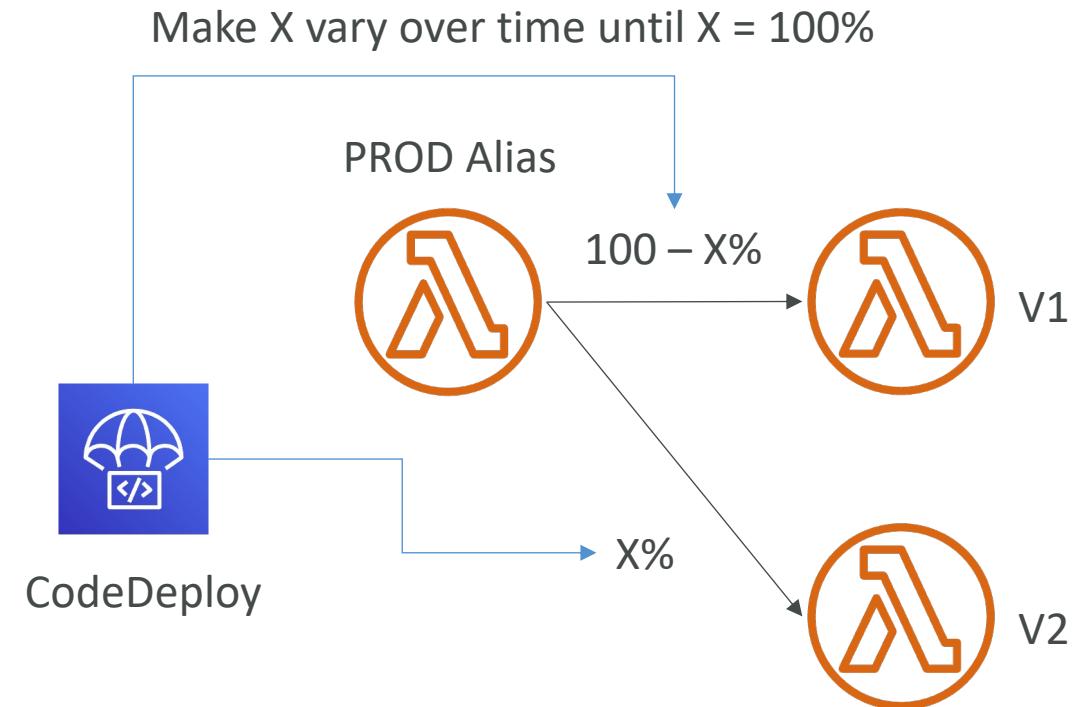


IAM Permissions

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Action": [  
        "s3:Get*",  
        "s3>List*"  
      ],  
      "Effect": "Allow",  
      "Resource": "*"  
    }  
  ]  
}
```

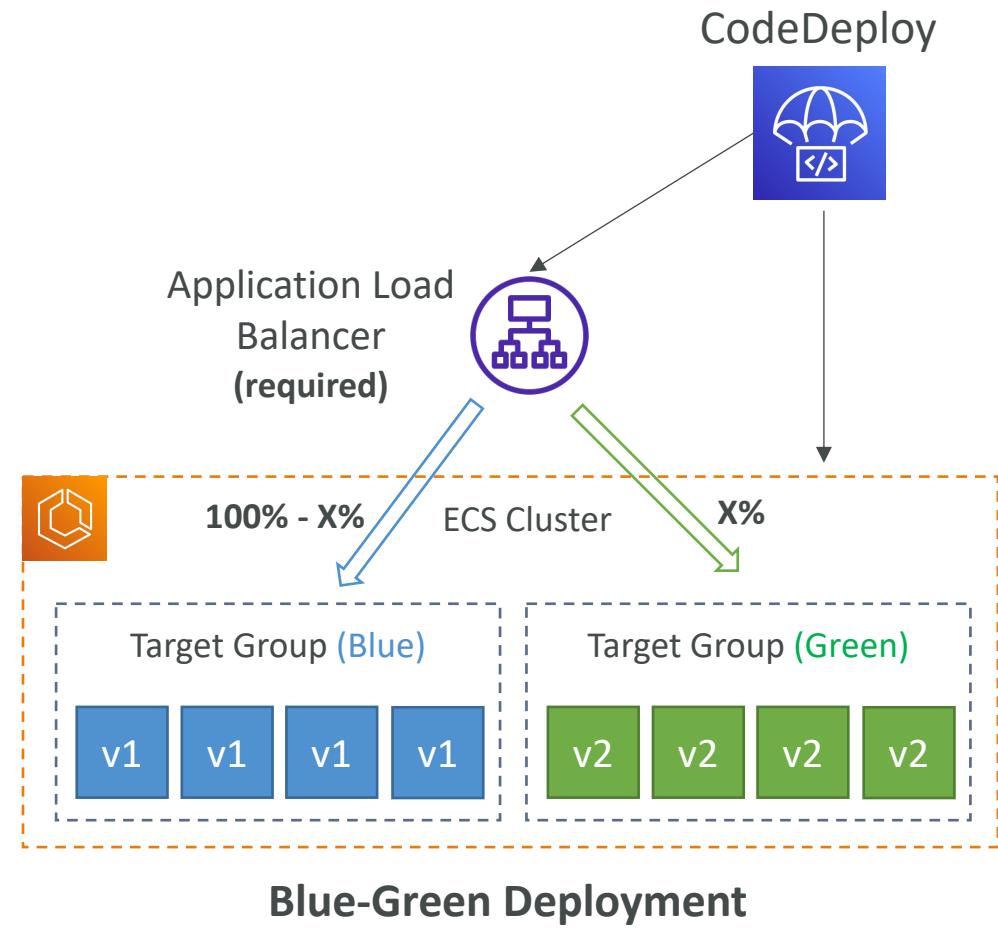
# CodeDeploy – Lambda Platform

- **CodeDeploy** can help you automate traffic shift for Lambda aliases
- Feature is integrated within the SAM framework
- **Linear**: grow traffic every N minutes until 100%
  - LambdaLinear10PercentEvery3Minutes
  - LambdaLinear10PercentEvery10Minutes
- **Canary**: try X percent then 100%
  - LambdaCanary10Percent5Minutes
  - LambdaCanary10Percent30Minutes
- **AllAtOnce**: immediate



# CodeDeploy – ECS Platform

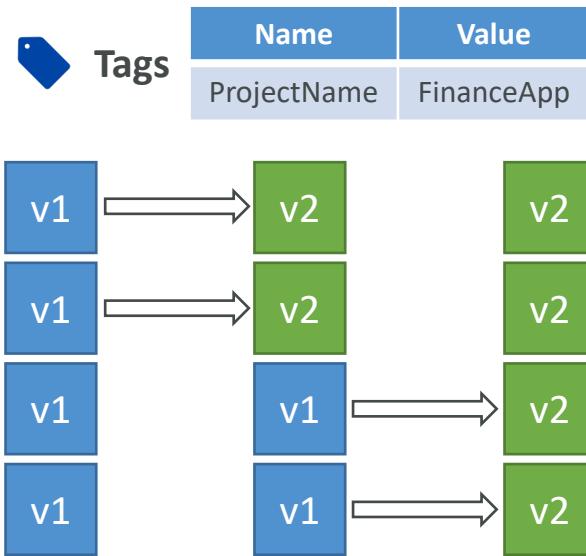
- CodeDeploy can help you automate the deployment of a new ECS Task Definition
- Only Blue/Green Deployments
- **Linear:** grow traffic every N minutes until 100%
  - ECSTaskDefinition10PercentEvery3Minutes
  - ECSTaskDefinition10PercentEvery10Minutes
- **Canary:** try X percent then 100%
  - ECSCanary10Percent5Minutes
  - ECSCanary10Percent30Minutes
- **AllAtOnce:** immediate



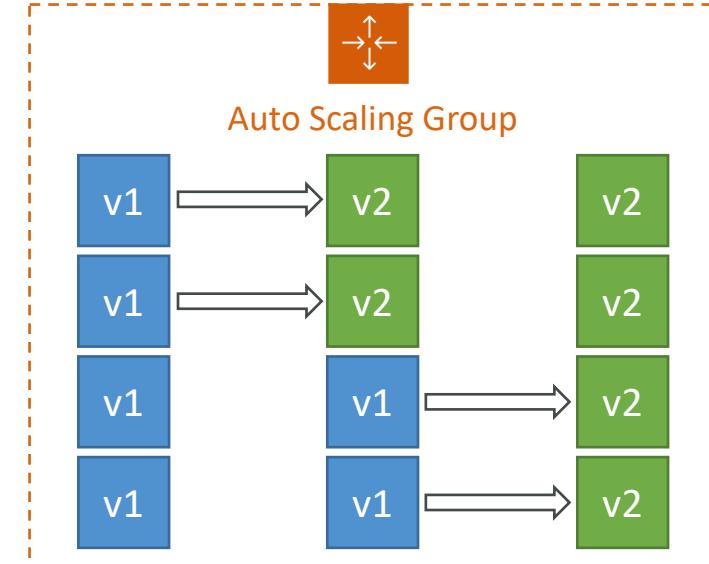
# CodeDeploy – In-place Deployments

- Use EC2 Tags or ASG to identify instances you want to deploy to
- With a Load Balancer: traffic is stopped before instance is updated, and started again after the instance is updated

In-Place Deployment: EC2 Tags



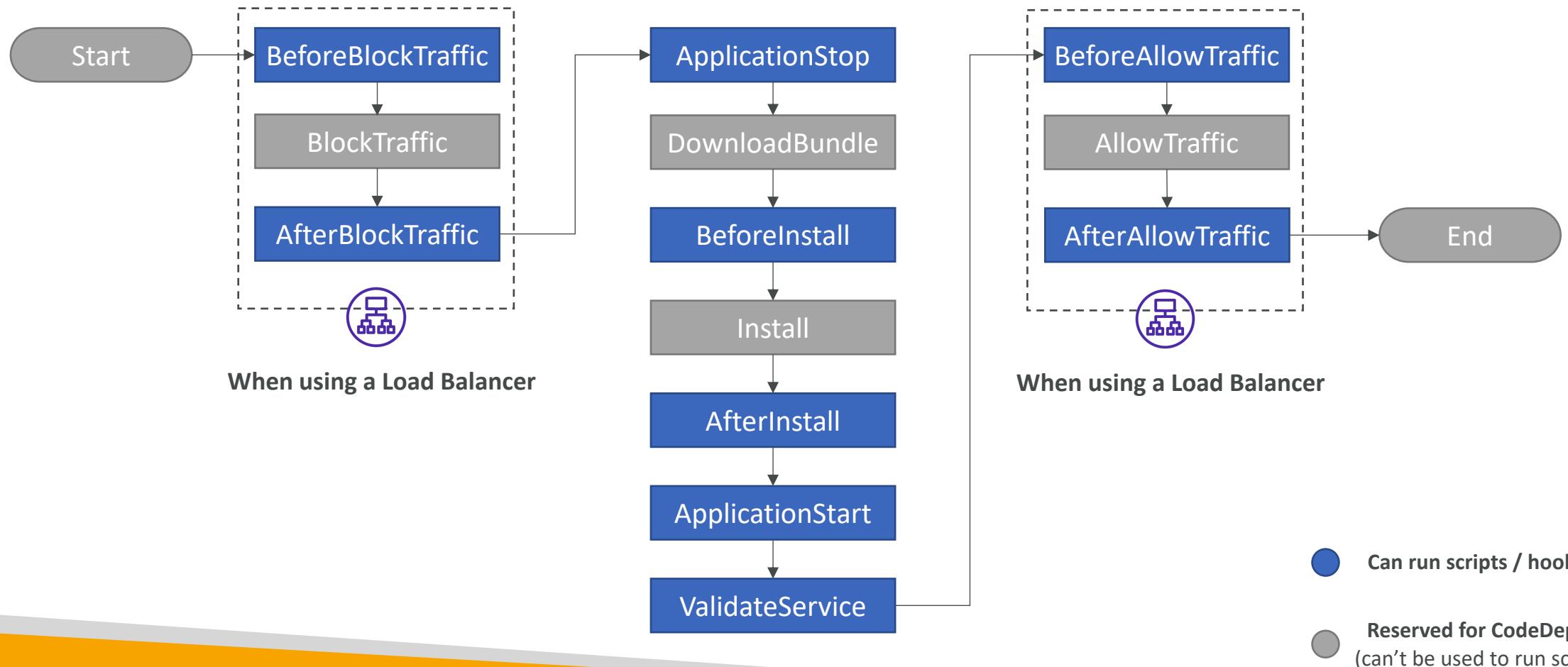
In-Place Deployment: ASG



**Note:** CodeDeploy will automatically deploy to newly launched EC2 instances in the ASG

# CodeDeploy – In-place Deployment Hooks

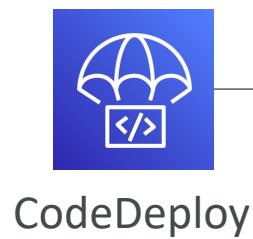
**Hooks** – one or more scripts to be run by CodeDeploy on each EC2 instance



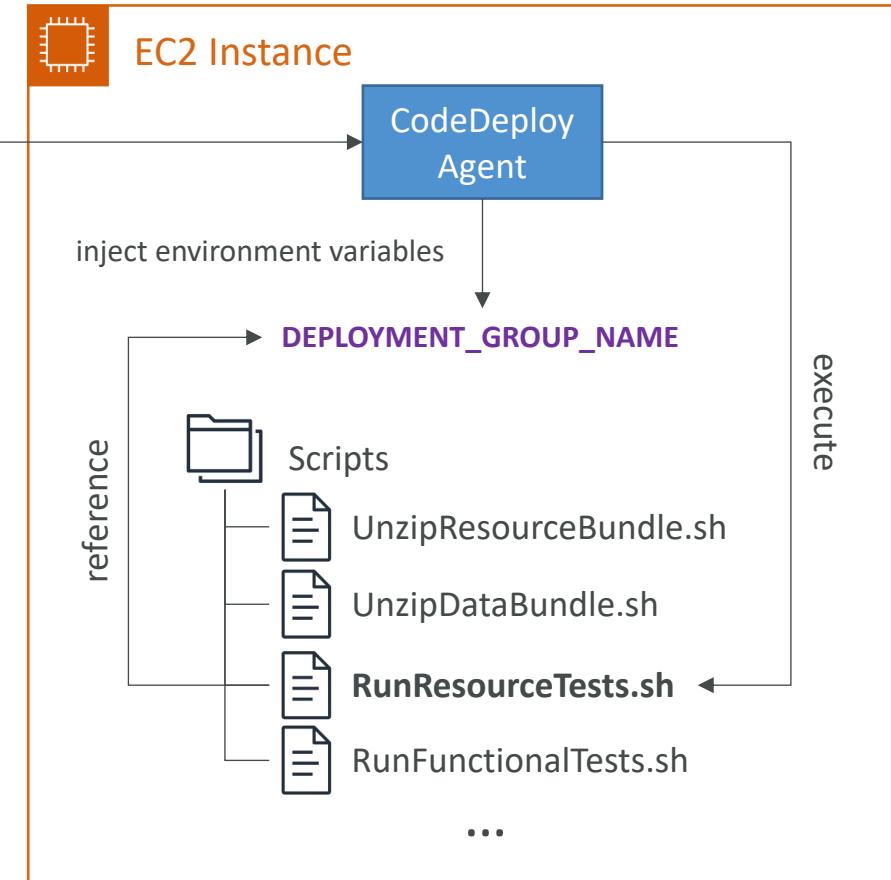
# CodeDeploy – EC2 Deployment Hooks

```
version: 0.0
os: linux
hooks:
  BeforeInstall:
    - location: Scripts/UnzipResourceBundle.sh
    - location: Scripts/UnzipDataBundle.sh
  AfterInstall:
    - location: Scripts/RunResourceTests.sh
      timeout: 180
  ApplicationStart:
    - location: Scripts/RunFunctionalTests.sh
      timeout: 3600
  ValidateService:
    - location: Scripts/MonitorService.sh
      timeout: 3600
      runas: codedeployuser
```

appspec.yml



run Hooks  
AfterInstall

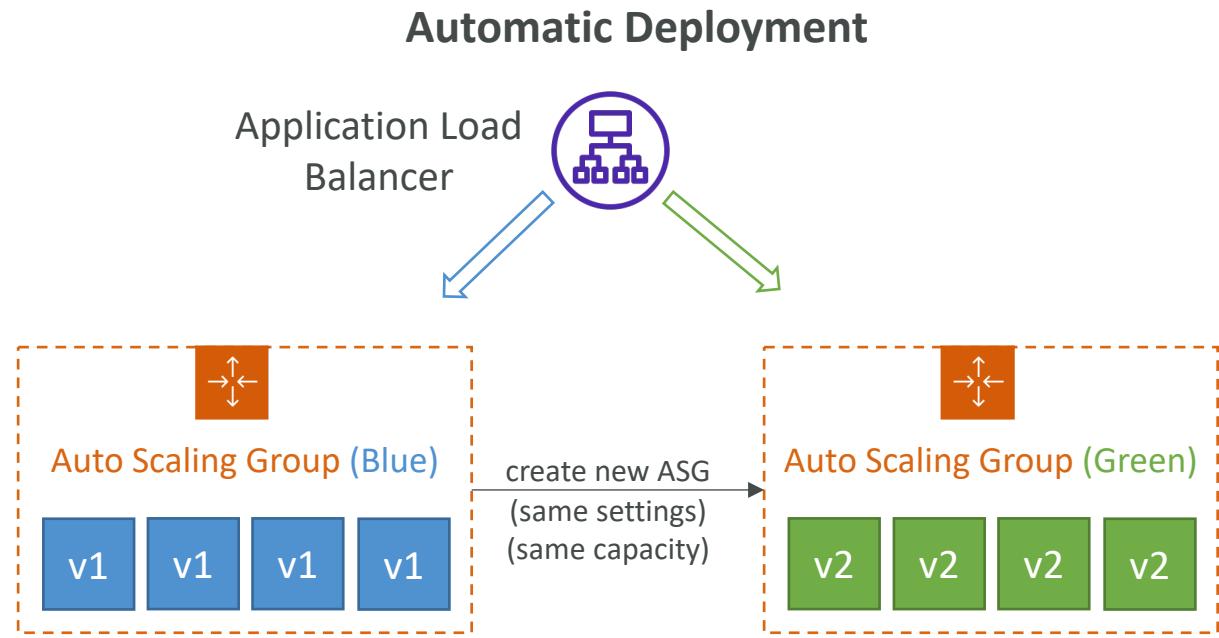
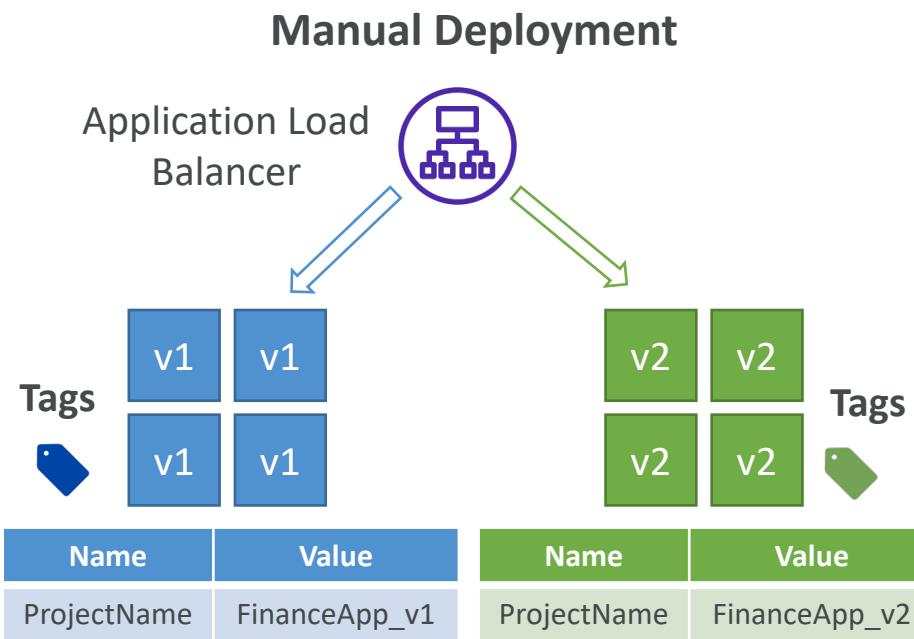


# CodeDeploy – Deployment Hooks Examples

- **BeforeInstall** – used for preinstall tasks, such as decrypting files and creating a backup of the current version
- **AfterInstall** – used for tasks such as configuring your application or changing file permissions
- **ApplicationStart** – used to start services that stopped during ApplicationStop
- **ValidateService** – used to verify the deployment was completed successfully
- **BeforeAllowTraffic** – run tasks on EC2 instances before registered to the load balancer
  - Example: perform health checks on the application and fail the deployment if the health checks are not successful

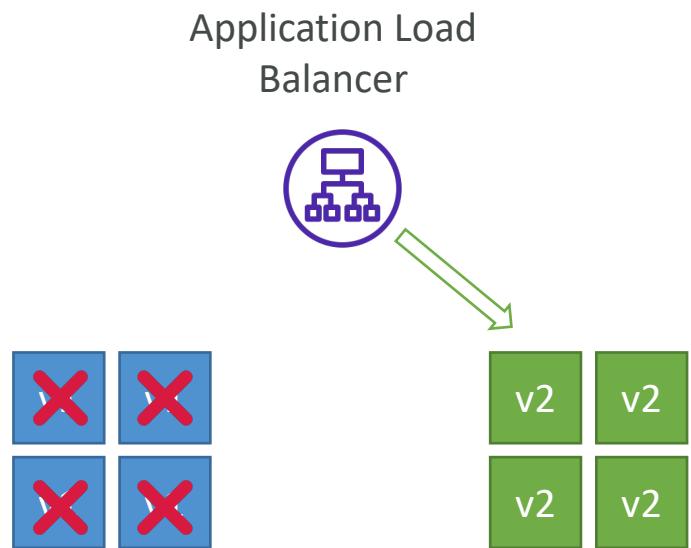
# CodeDeploy – Blue/Green Deployments

- Manually mode: provision EC2 Instances for Blue and Green and identify by Tags
- Automatic mode: new ASG is provisioned by CodeDeploy (settings are copied)
- Using a Load Balancer is necessary for Blue/Green



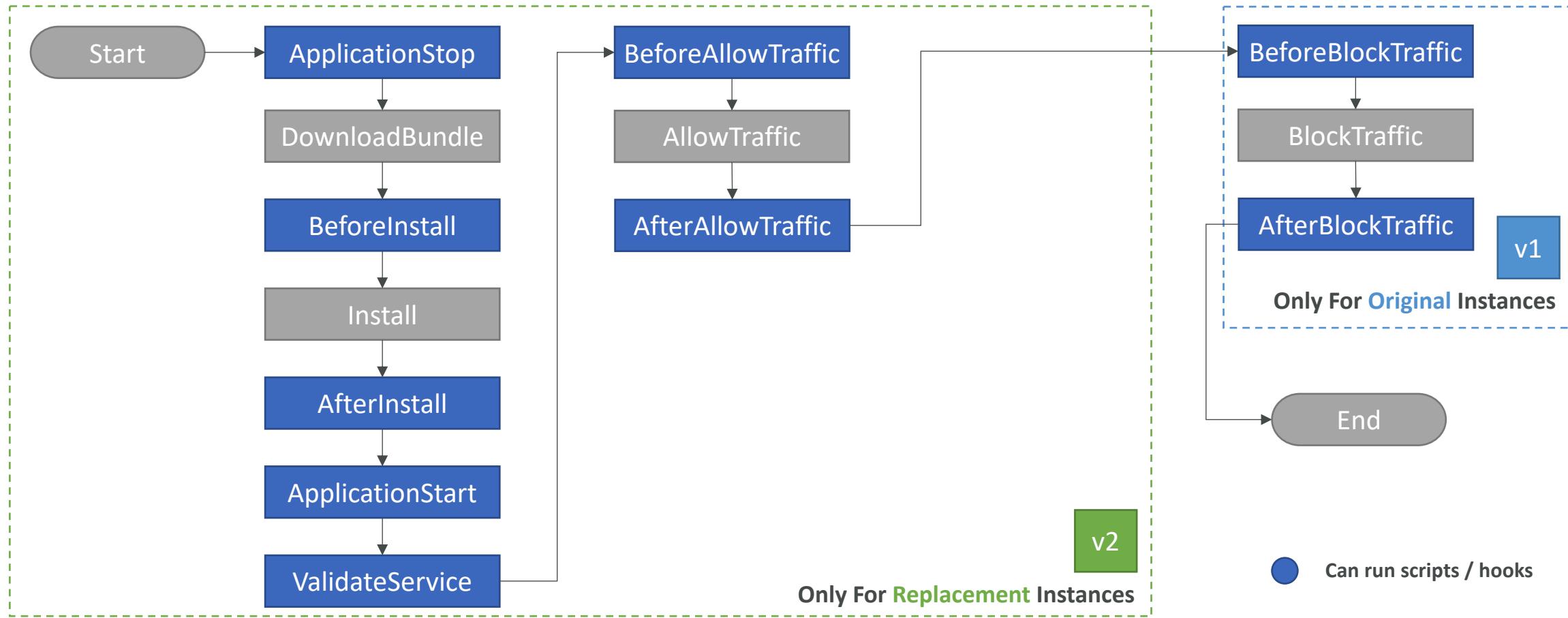
# CodeDeploy – Blue-Green Instance Termination

- **BlueInstanceTerminationOption** – specify whether to terminate the original (Blue) EC2 instances after a successfully Blue-Green deployment
- **Action Terminate** - Specify Wait Time, default 1 hour, max 2 days
- **Action Keep Alive** - Instances are kept running but deregistered from ELB and deployment group



# CodeDeploy – Blue/Green Deployment Hooks

**Hooks** – one or more scripts to be run by CodeDeploy on EC2 instances

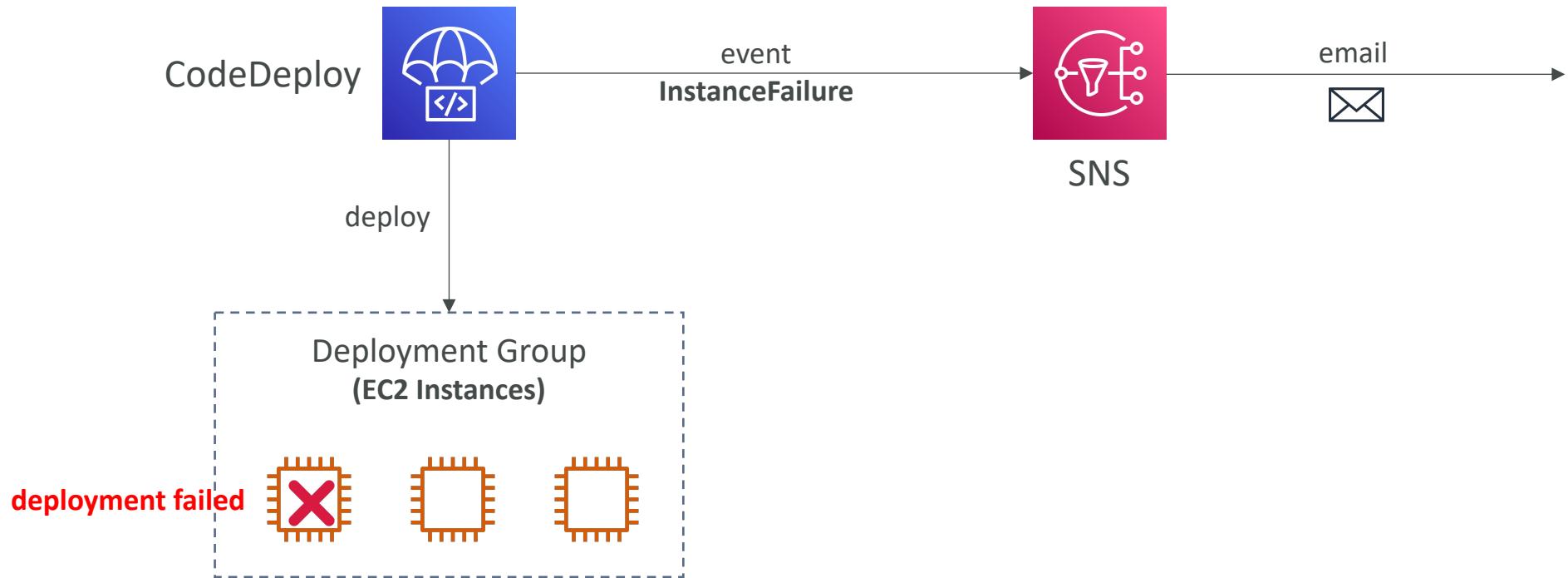


# CodeDeploy – Deployment Configurations

- Specifies the number of instances that must remain available at any time during the deployment
- Use Pre-defined Deployment Configurations
  - `CodeDeployDefault.AllAtOnce` – deploy to as many instances as possible at once
  - `CodeDeployDefault.HalfAtATime` – deploy to up to half of the instances at a time
  - `CodeDeployDefault.OneAtATime` – deploy to only one instance at a time
- Can create your own custom Deployment Configurations

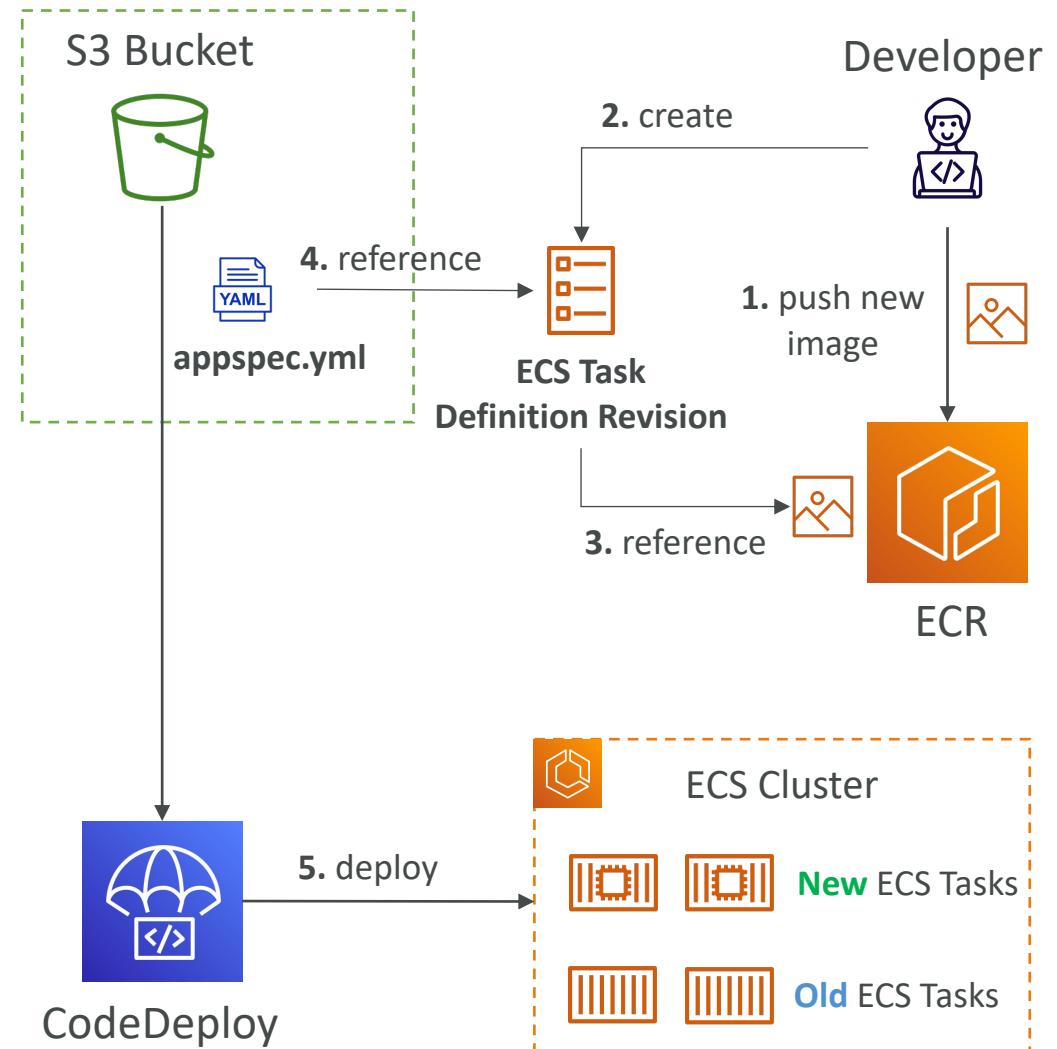
# CodeDeploy – Triggers

- Publish Deployment or EC2 instance events to SNS topic
- Examples: DeploymentSuccess, Deployment Failure, InstanceFailure

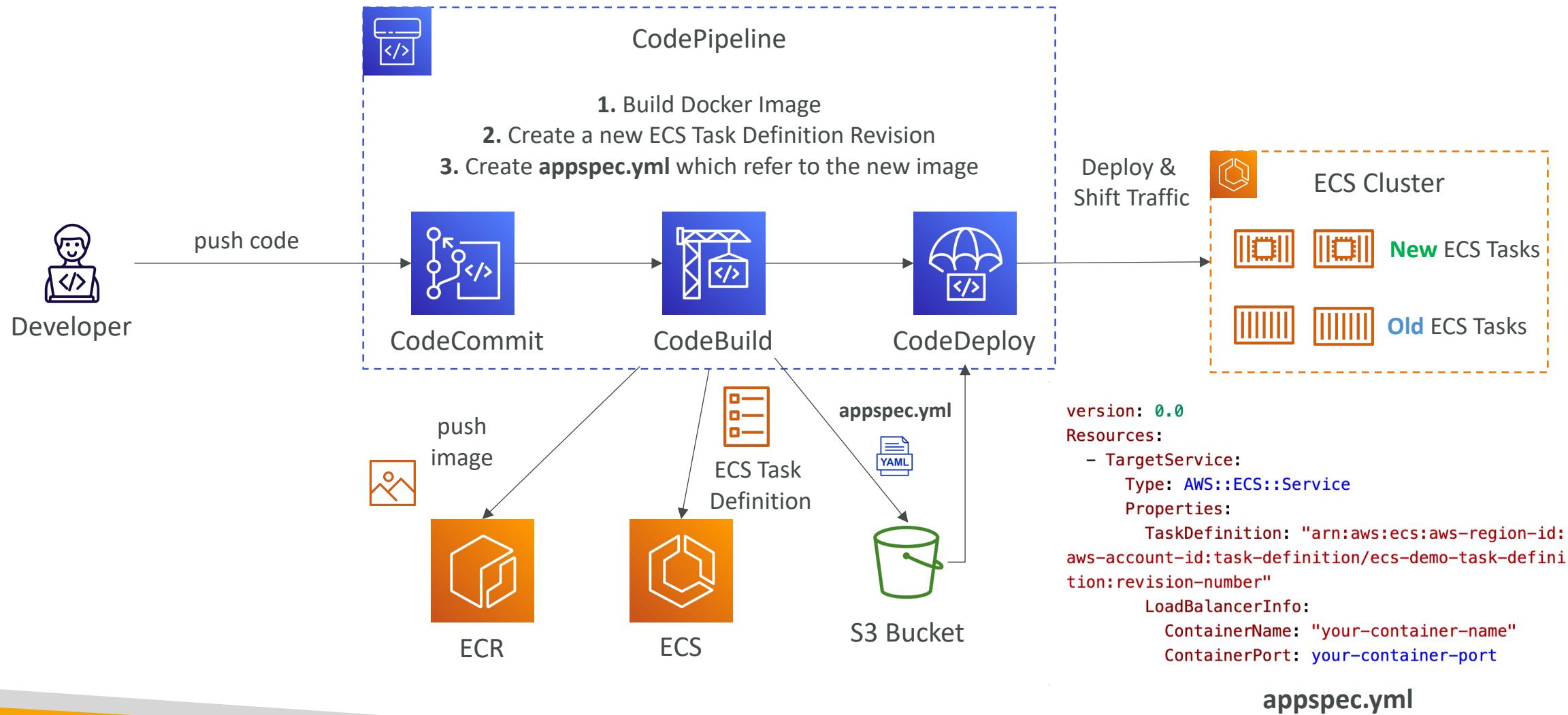


# CodeDeploy – Deployment to ECS

- Automatically handles a deployment of a new ECS Task Definition to an ECS Service
- Only supports Blue/Green and the service must be connected to a Load Balancer
- The ECS Task Definition and new Container Images must be already created
- A reference to the new ECS Task Revision (**TaskDefinition**) and load balancer information (**LoadBalancerInfo**) are specified in the **appspec.yml** file
- CodeDeploy Agent is NOT required

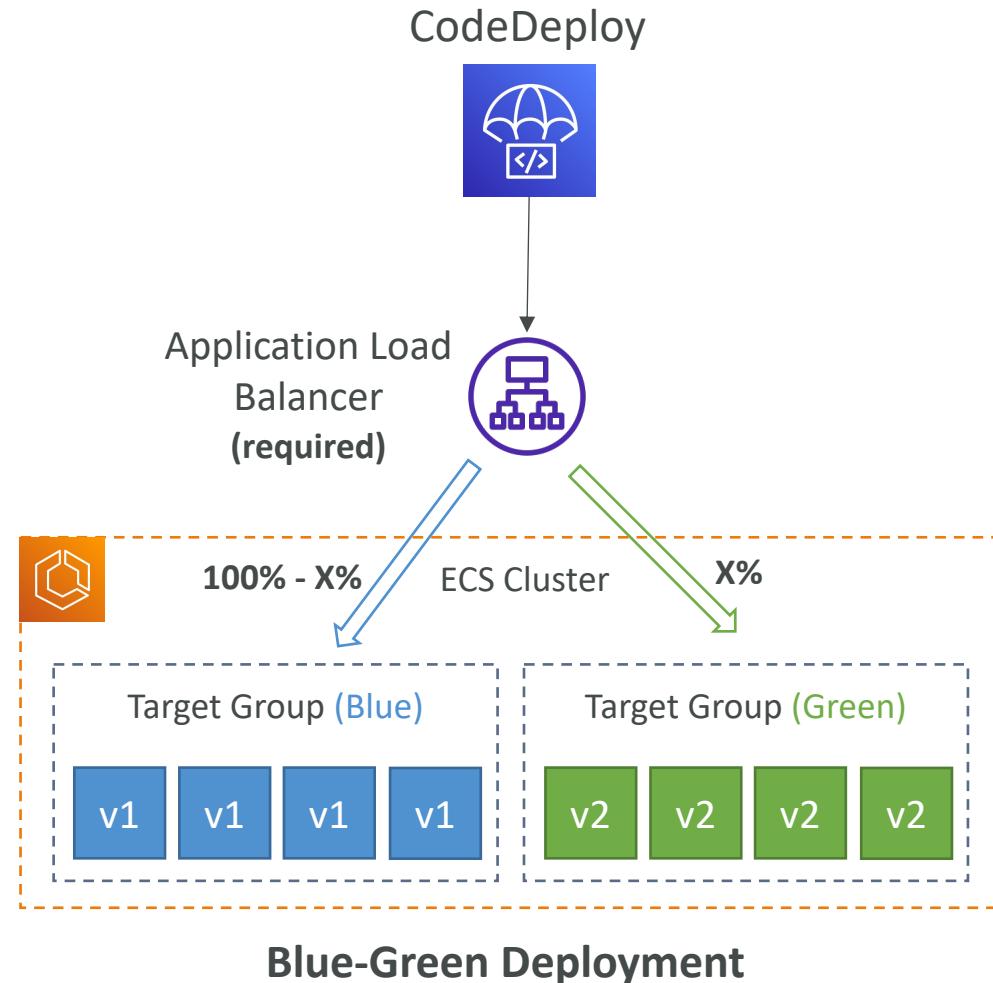


# CodeDeploy – Deployment to ECS

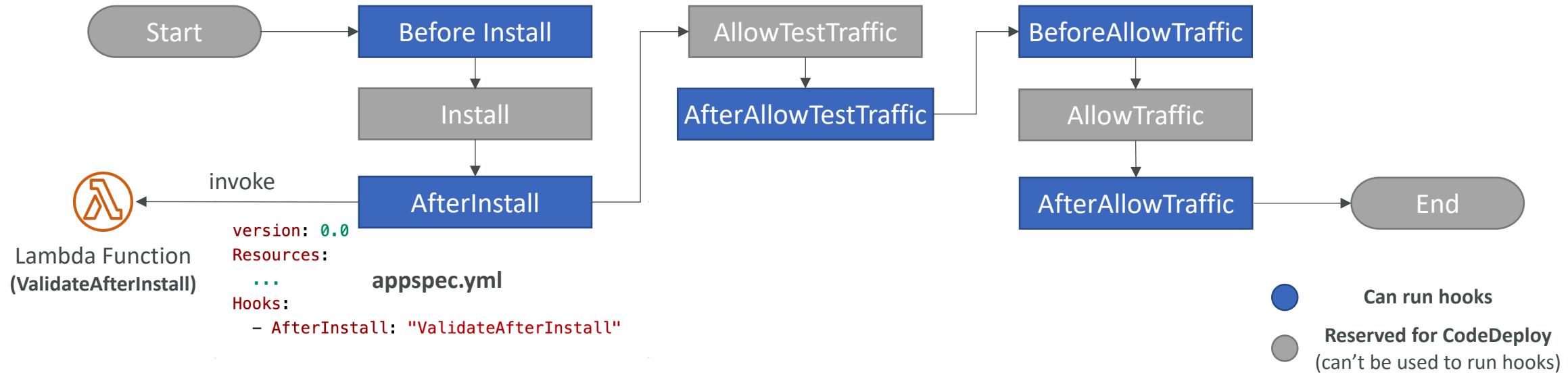


# CodeDeploy – Deployment to ECS

- You can shift traffic to the new Task Set using **Canary, Linear, or All At Once**
  - ECSLinear|0PercentEvery1Minutes
  - ECSLinear|0PercentEvery3Minutes
  - ECSCanary|0Percent5Minutes
  - ECSCanary|0Percent15Minutes
  - ECSAllAtOnce
- Can define a second ELB Test Listener to test the Replacement (Green) version before traffic is rebalanced



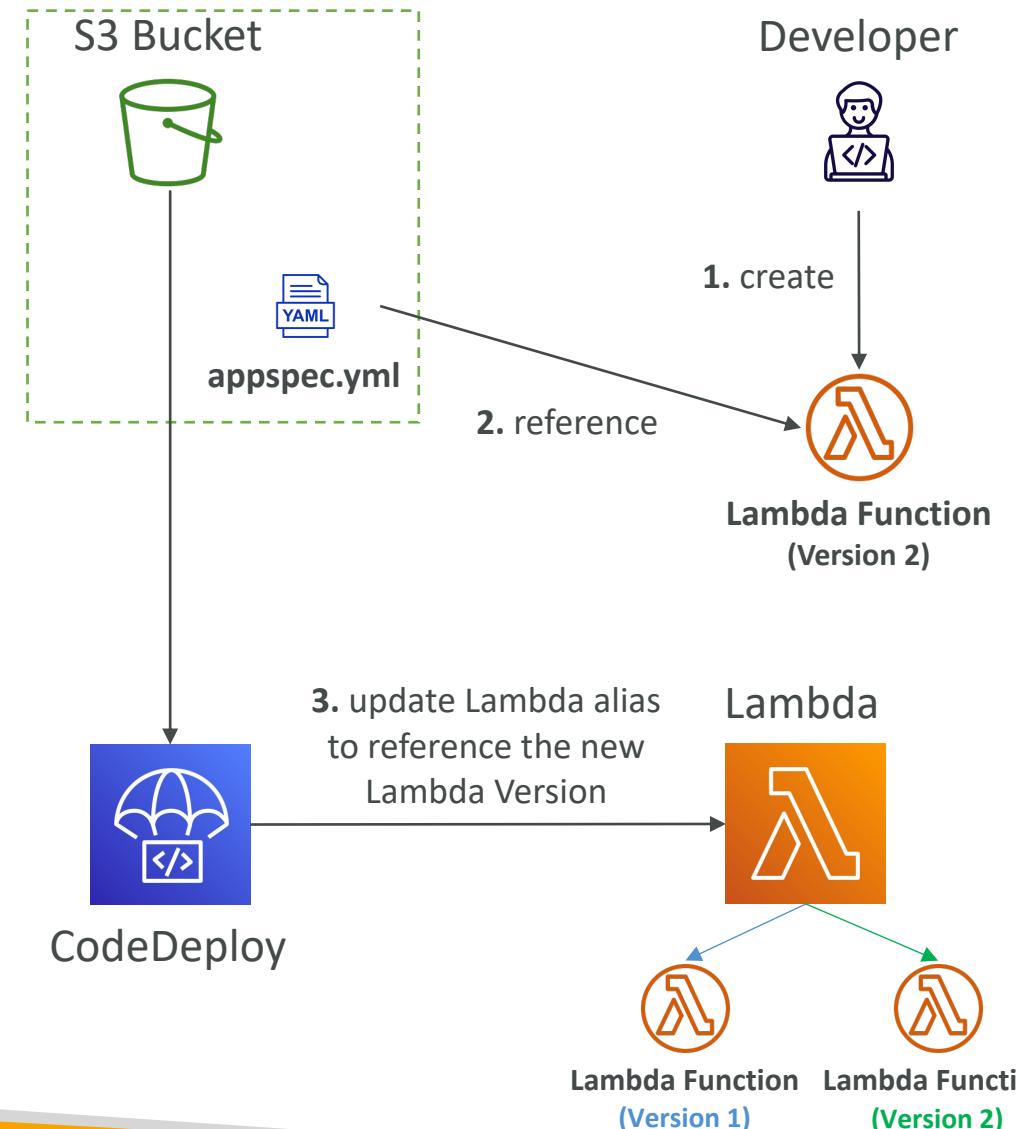
# CodeDeploy – ECS Deployment Hooks



- Hooks – a Lambda function to be executed once per deployment
- **AfterAllowTestTraffic** – run AWS Lambda function after the test ELB Listener serves traffic to the Replacement ECS Task Set
  - Example: perform health checks on the application and trigger a rollback if the health checks are not successful

# CodeDeploy – Deployment to Lambda

- You create a new Lambda function Version and want to deploy it
- Specify the Version info in the `appspec.yml` file
- CodeDeploy Updates a Lambda function with the new Lambda function Version
- CodeDeploy Agent is NOT required



# Lambda & CodeDeploy – AppSpec.yml

```
version: 0.0
```

```
Resources:
```

```
- myLambdaFunction:
```

```
  Type: AWS::Lambda::Function
```

```
  Properties:
```

```
    Name: myLambdaFunction
```

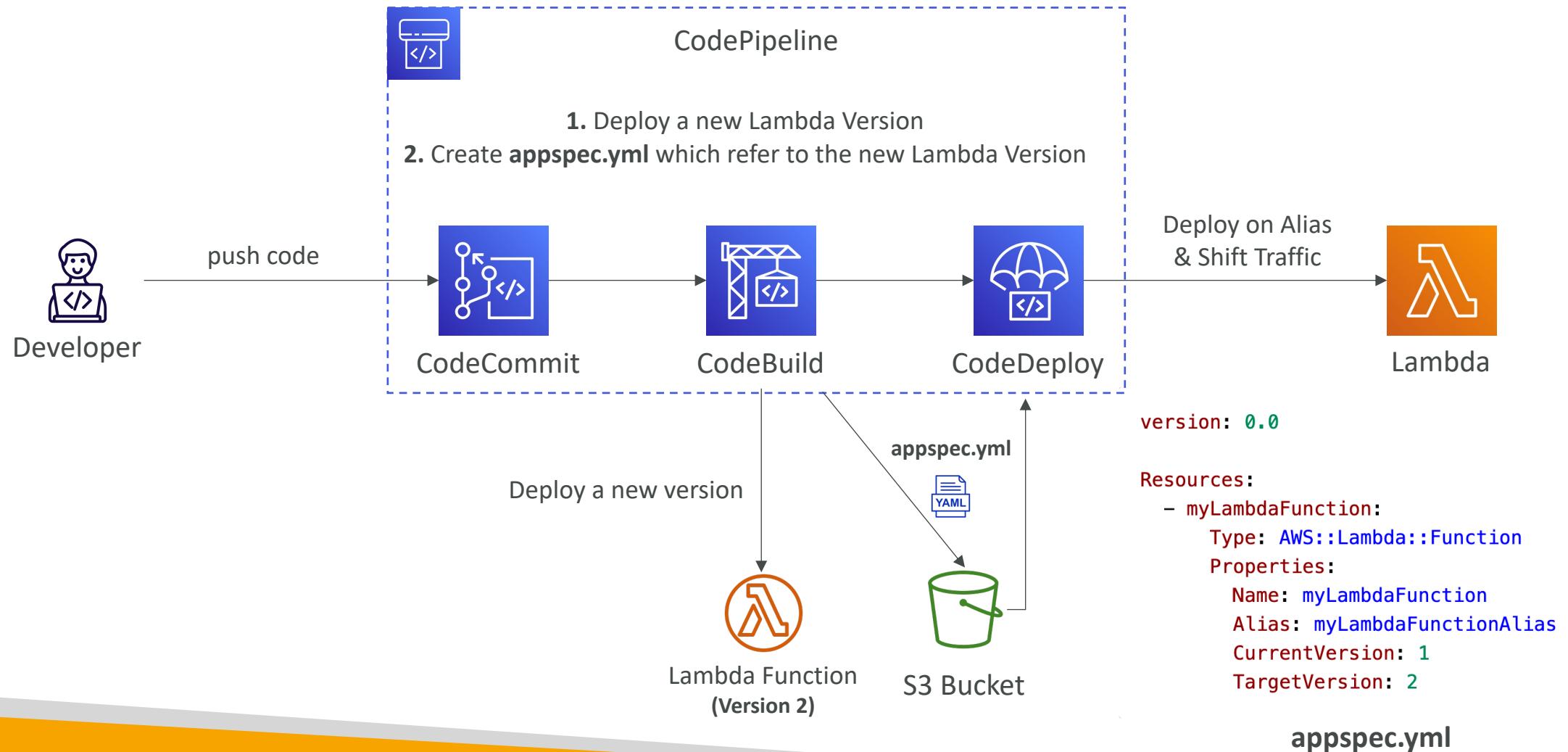
```
    Alias: myLambdaFunctionAlias
```

```
    CurrentVersion: 1
```

```
    TargetVersion: 2
```

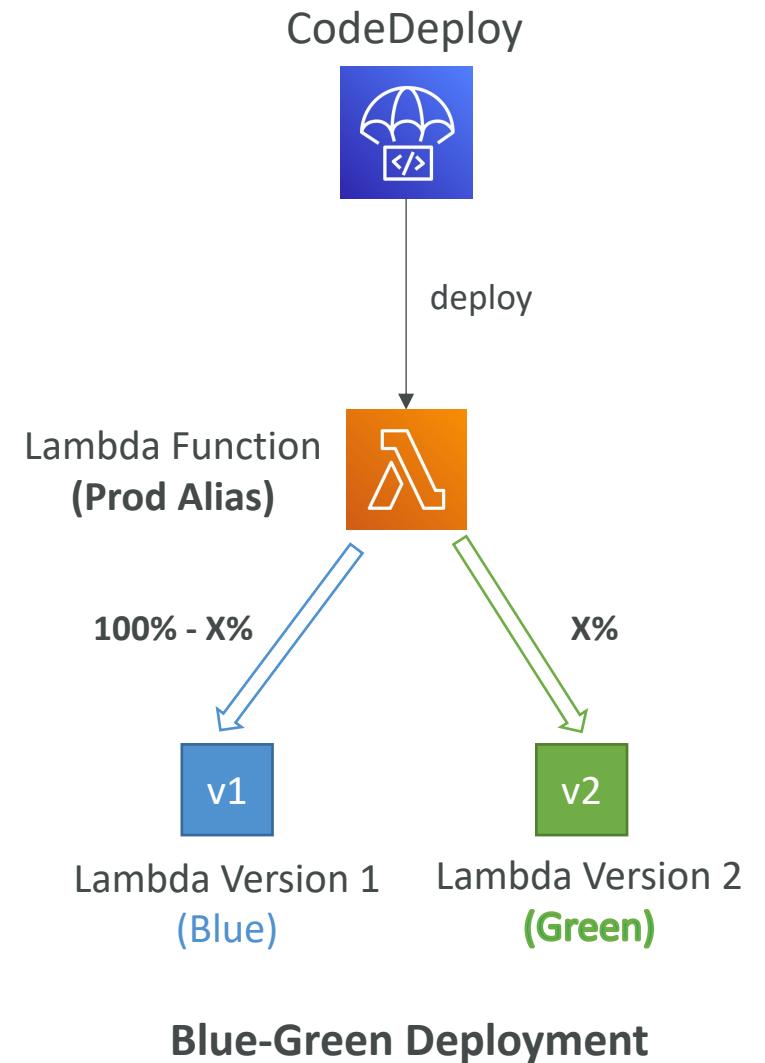
- **Name (required)** – the name of the Lambda function to deploy
- **Alias (required)** – the name of the alias to the Lambda function
- **CurrentVersion (required)** – the version of the Lambda function traffic currently points to
- **TargetVersion (required)** – the version of the Lambda function traffic is shifted to

# CodeDeploy – Deployment to Lambda

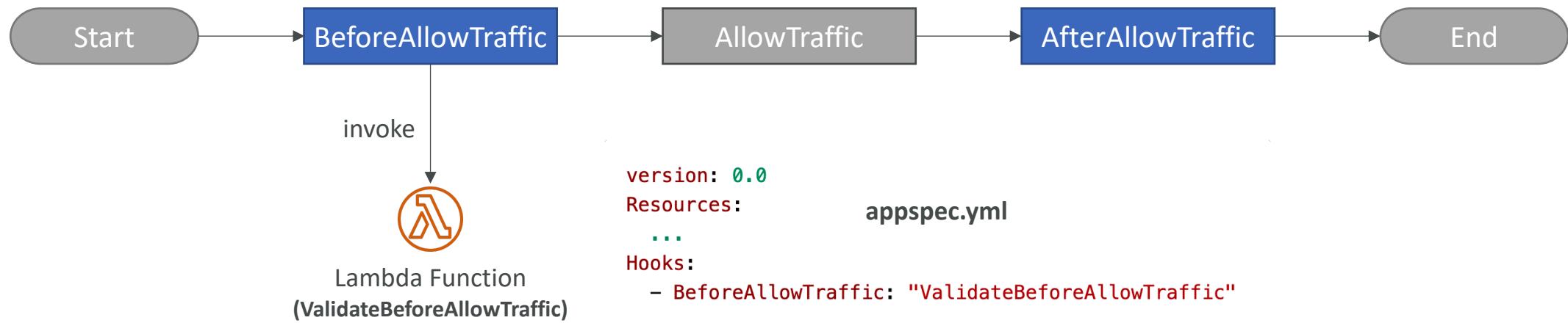


# CodeDeploy – Deployment to Lambda

- Only Blue-Green Deployment supported
- You can shift traffic to the new Task Set using **Canary, Linear, or All At Once**
  - LambdaLinear|0PercentEvery|Minute
  - LambdaLinear|0PercentEvery(2,3,10)Minutes
  - LambdaCanary|0Percent(5,10,15,30)Minutes
  - LambdaAllAtOnce



# CodeDeploy – Lambda Deployment Hooks (Blue-Green)



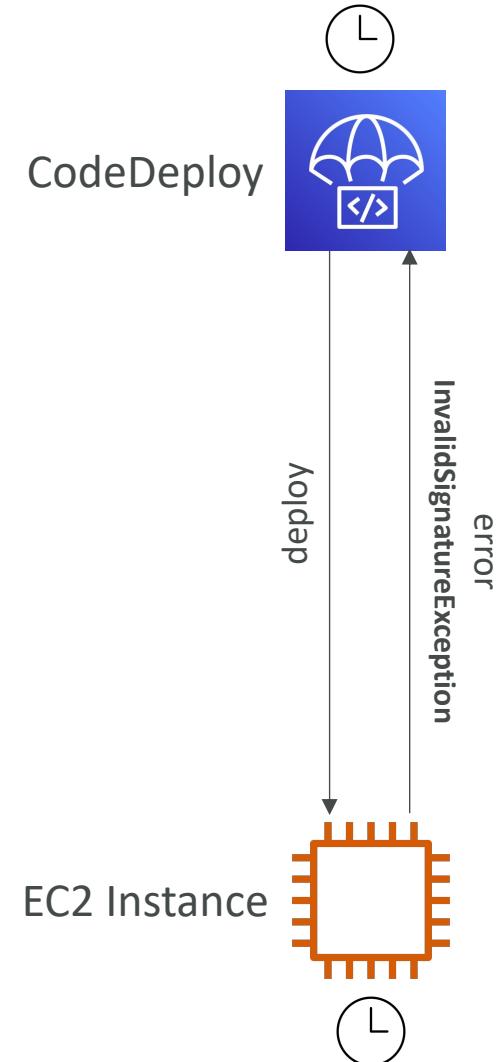
- Hooks – a Lambda function to be executed once per deployment
- BeforeAllowTraffic and AfterAllowTraffic hooks can be used to check the health of the Lambda function

# CodeDeploy – Redeploy & Rollbacks

- Rollback = redeploy a previously deployed revision of your application
- Deployments can be rolled back:
  - Automatically – rollback when a deployment fails or rollback when a CloudWatch Alarm thresholds are met
  - Manually
- Disable Rollbacks — do not perform rollbacks for this deployment
- If a roll back happens, CodeDeploy redeploys the last known good revision as a new deployment (not a restored version)

# CodeDeploy – Troubleshooting

28/01/2023 @ 5:01pm



- Deployment Error: “`InvalidSignatureException – Signature expired: [time] is now earlier than [time]`”
  - For CodeDeploy to perform its operations, it requires accurate time references
  - If the date and time on your EC2 instance are not set correctly, they might not match the signature date of your deployment request, which CodeDeploy rejects

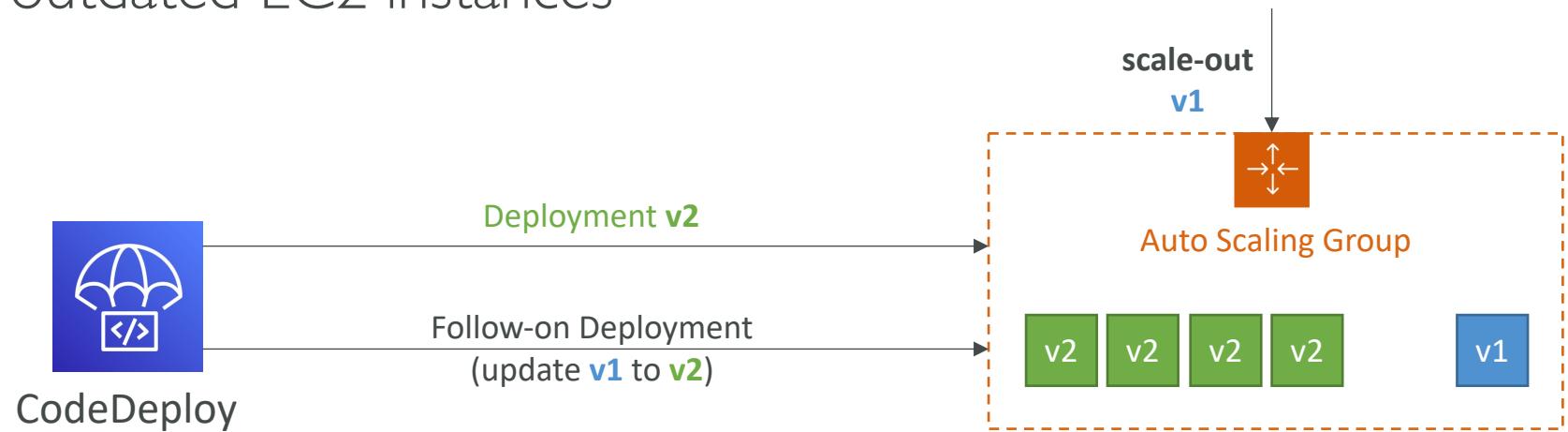
# CodeDeploy – Troubleshooting

- When the Deployment or all Lifecycle Events are skipped (EC2/On-Premises), you get one of the following errors:
  - “The overall deployment failed because too many individual instances failed deployment”
  - “Too few healthy instances are available for deployment”
  - “Some instances in your deployment group are experiencing problems. (Error code: HEALTH\_CONSTRAINTS)”
- Reasons:
  - CodeDeploy Agent might not be installed, running, or it can't reach CodeDeploy
  - CodeDeploy Service Role or IAM instance profile might not have the required permissions
  - You're using an HTTP Proxy, configure CodeDeploy Agent with `:proxy_uri:` parameter
  - Date and Time mismatch between CodeDeploy and Agent

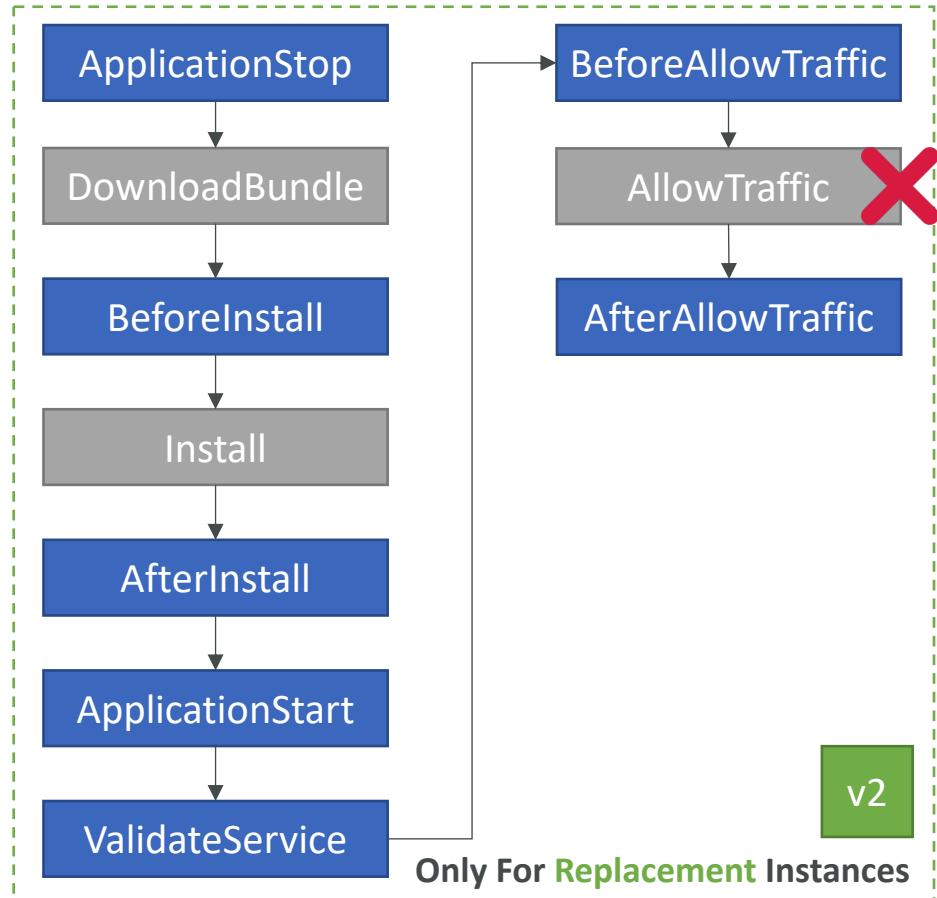


# CodeDeploy – Troubleshooting

- If a CodeDeploy deployment to ASG is underway and a scale-out event occurs, the new instances will be updated with the application revision that was most recently deployed (not the application revision that is currently being deployed)
- ASG will have EC2 instances hosting different versions of the application
- By default, CodeDeploy automatically starts a follow-on deployment to update any outdated EC2 instances



# CodeDeploy – Troubleshooting



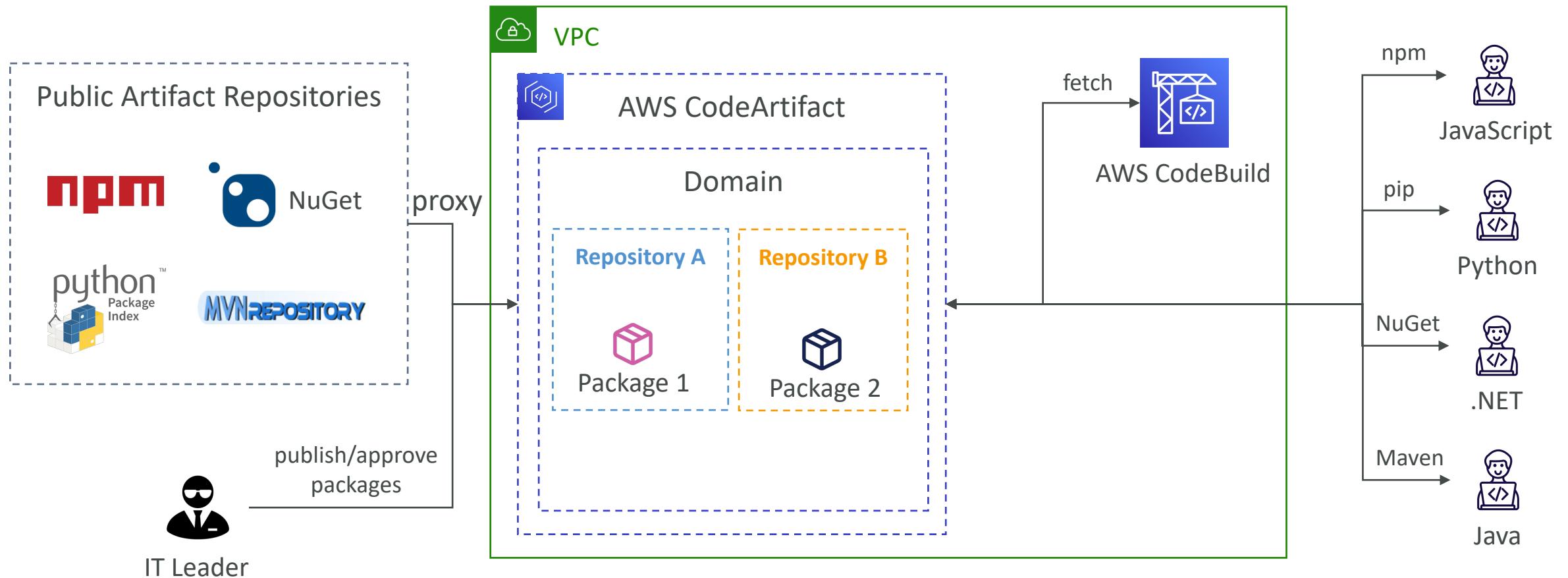
- **Issue:** failed AllowTraffic lifecycle event in Blue/Green Deployments with no error reported in the Deployment Logs
  - **Reason:** incorrectly configured health checks in ELB
  - **Resolution:** review and correct any errors in ELB health checks configuration

# AWS CodeArtifact



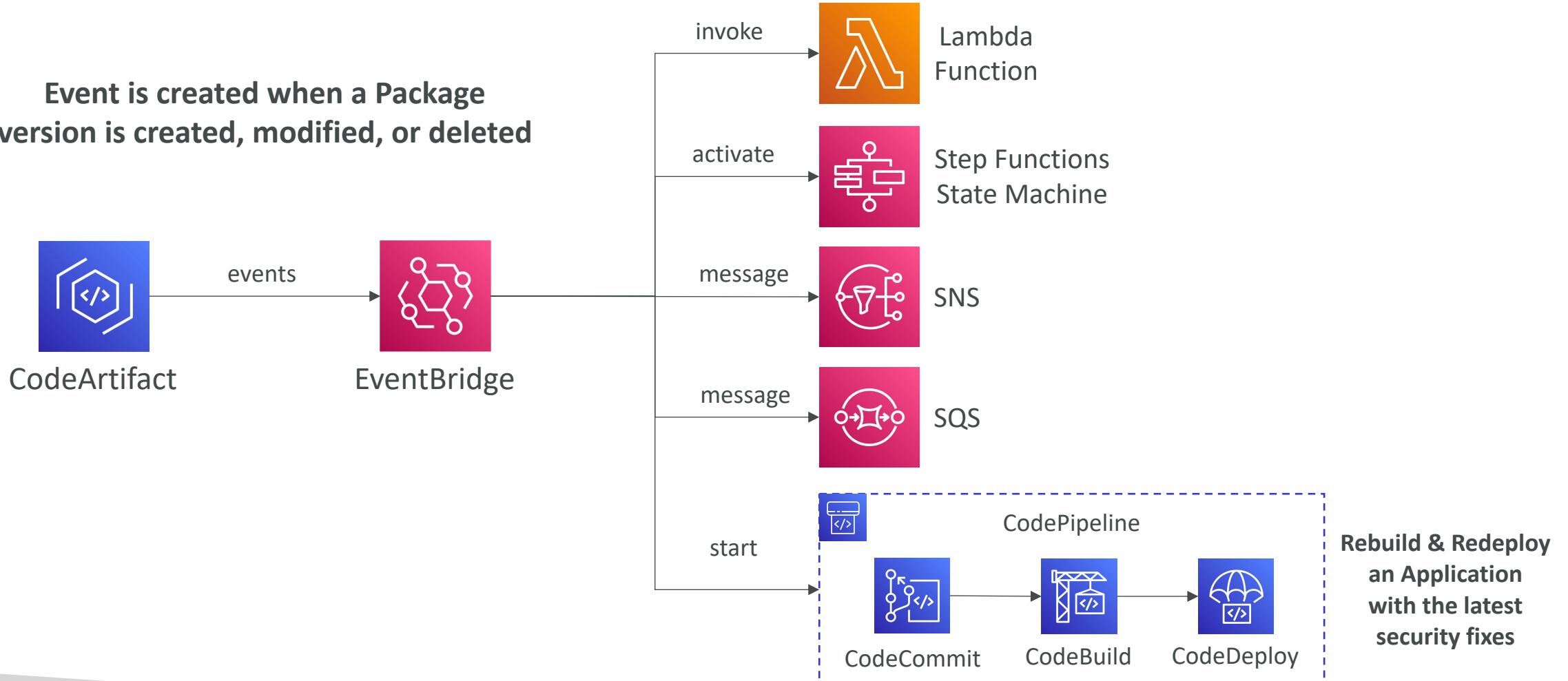
- Software packages depend on each other to be built (also called code dependencies), and new ones are created
- Storing and retrieving these dependencies is called **artifact management**
- Traditionally you need to setup your own artifact management system
- **CodeArtifact** is a secure, scalable, and cost-effective **artifact management** for software development
- Works with common dependency management tools such as Maven, Gradle, npm, yarn, twine, pip, and NuGet
- Developers and CodeBuild can then retrieve dependencies straight from CodeArtifact

# AWS CodeArtifact



# CodeArtifact – EventBridge Integration

**Event is created when a Package version is created, modified, or deleted**



# CodeArtifact – Resource Policy

- Can be used to authorize another account to access CodeArtifact
- A given principal can either read all the packages in a repository or none of them

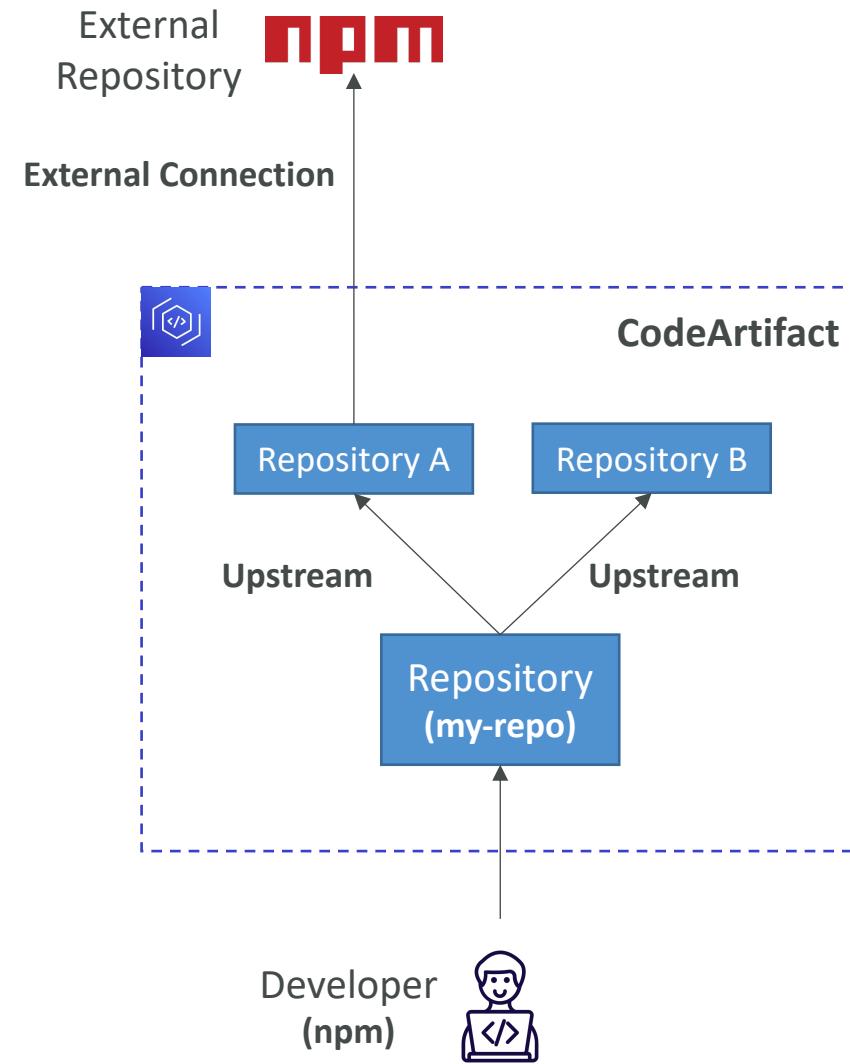


```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": [  
        "codeartifact:DescribePackageVersion",  
        "codeartifact:DescribeRepository",  
        "codeartifact:GetPackageVersionReadme",  
        "codeartifact:GetRepositoryEndpoint",  
        "codeartifact>ListPackages",  
        "codeartifact>ListPackageVersions",  
        "codeartifact>ListPackageVersionAssets",  
        "codeartifact>ListPackageVersionDependencies",  
        "codeartifact:ReadFromRepository"  
      ],  
      "Principal": {  
        "AWS": [  
          "arn:aws:iam:123456789012:root",  
          "arn:aws:iam:222333344555:user/bob"  
        ]  
      },  
      "Resource": "*"  
    }  
  ]  
}
```

Repository Resource Policy

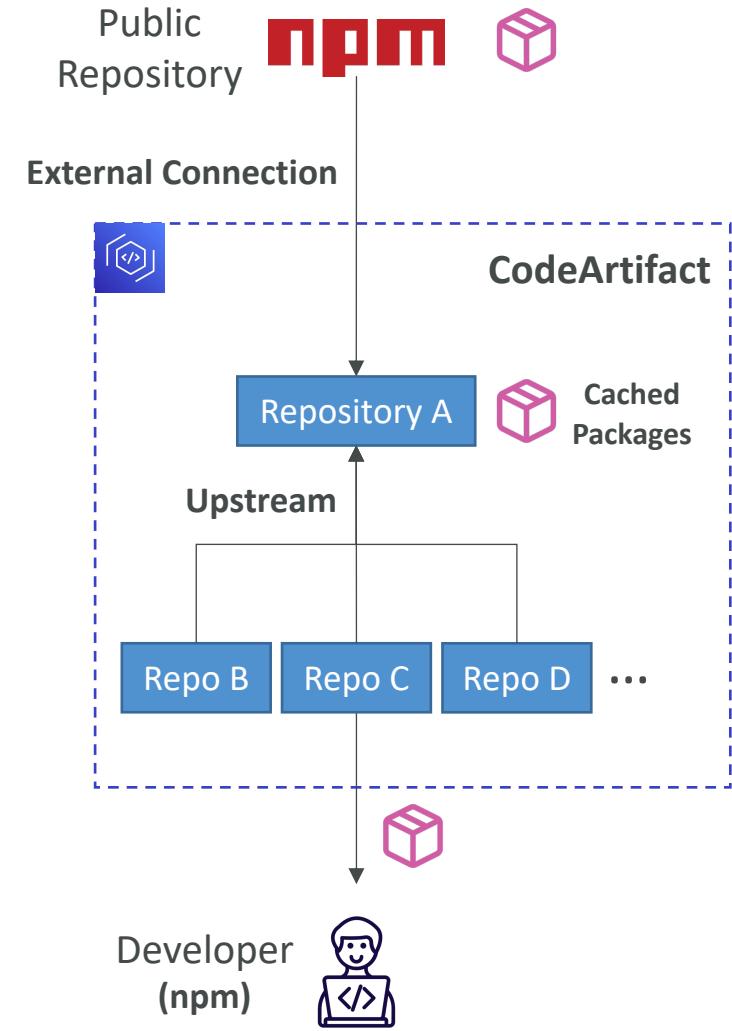
# CodeArtifact – Upstream Repositories

- A CodeArtifact repository can have other CodeArtifact repositories as **Upstream Repositories**
- Allows a package manager client to access the packages that are contained in more than one repository using a single repository endpoint
- Up to 10 Upstream Repositories
- Only one external connection



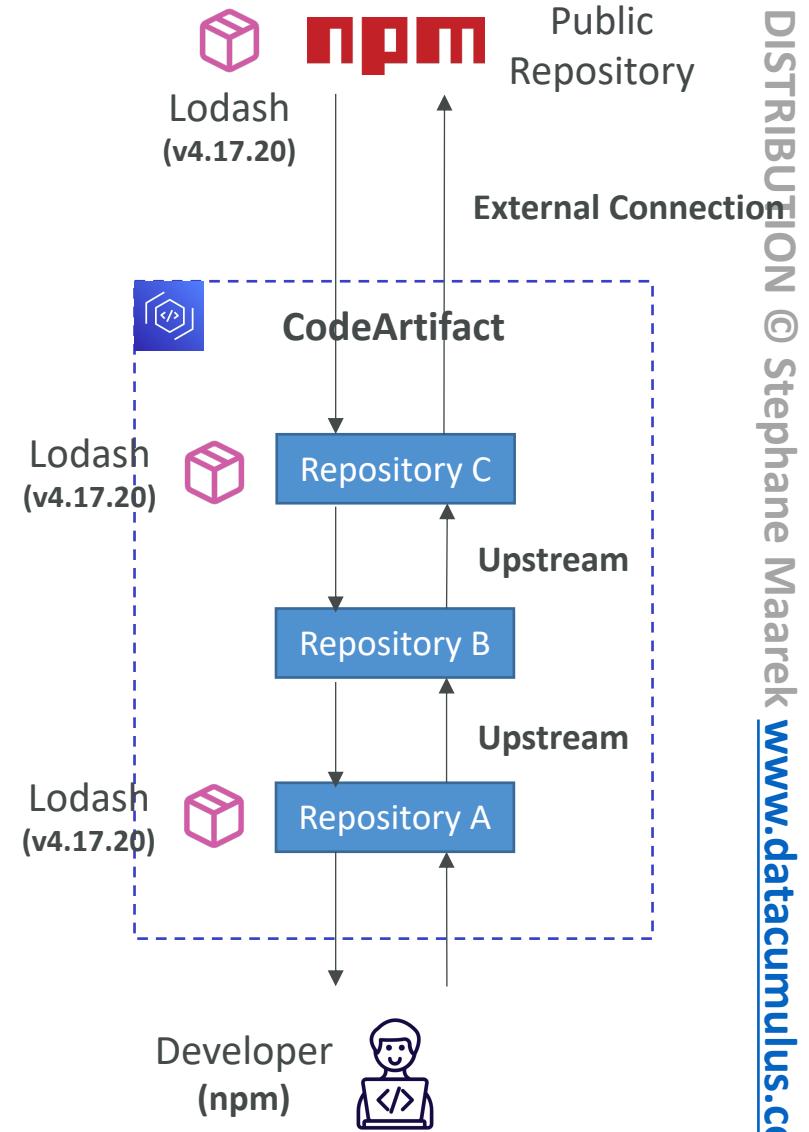
# CodeArtifact – External Connection

- An External Connection is a connection between a CodeArtifact Repository and an external/public repository (e.g., Maven, npm, PyPI, NuGet...)
  - Allows you to fetch packages that are not already present in your CodeArtifact Repository
  - A repository has a maximum of 1 external connection
  - Create many repositories for many external connections
- 
- Example – Connect to [npmjs.com](https://npmjs.com)
    - Configure one CodeArtifact Repository in your domain with an external connection to [npmjs.com](https://npmjs.com)
    - Configure all the other repositories with an upstream to it
    - Packages fetched from [npmjs.com](https://npmjs.com) are cached in the Upstream Repository, rather than fetching and storing them in each Repository



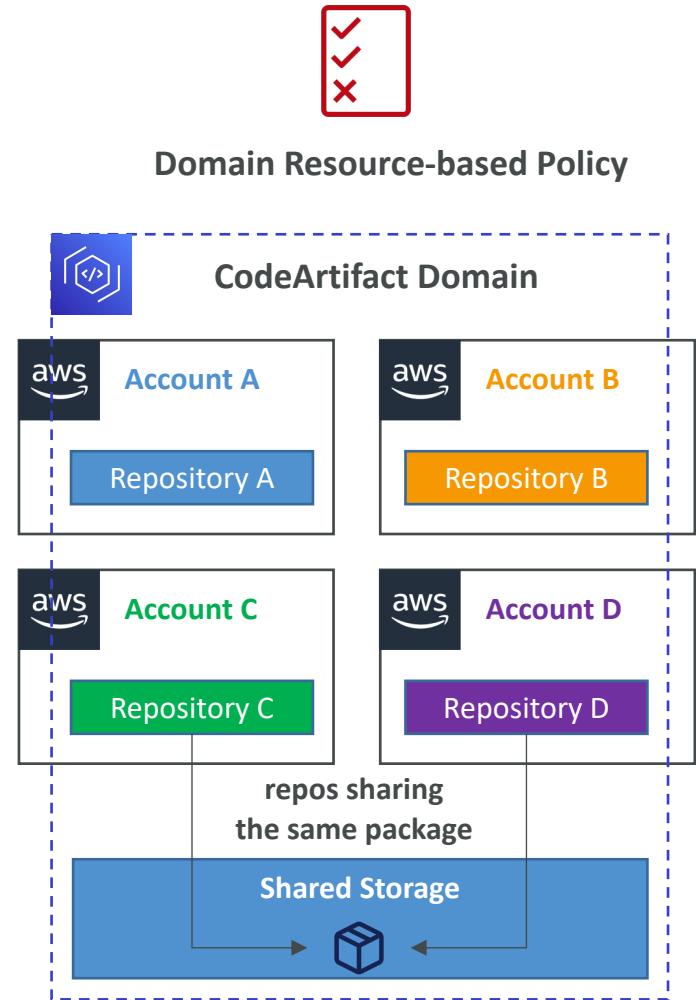
# CodeArtifact – Retention

- If a requested package version is found in an Upstream Repository, a reference to it is retained and is always available from the Downstream Repository
- The retained package version is not affected by changes to the Upstream Repository (deleting it, updating the package...)
- Intermediate repositories do not keep the package
- **Example – Fetching Package from npmjs.com**
  - Package Manager connected to Repository A requests the package **Lodash v4.17.20**
  - The package version is not present in any of the three repositories
  - The package version will be fetched from npmjs.com
  - When **Lodash 4.17.20** is fetched, it will be retained in:
    - Repository A – the most-downstream repository
    - Repository C – has the external connection to npmjs.com
    - The Package version will not be retained in Repository B as that is an intermediate Repository



# CodeArtifact – Domains

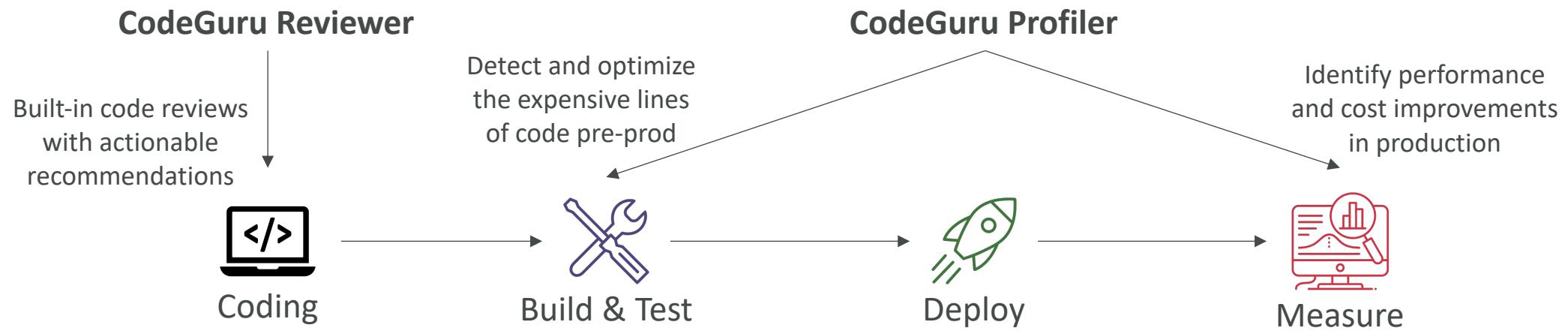
- **Deduplicated Storage** – asset only needs to be stored once in a domain, even if it's available in many repositories (only pay once for storage)
- **Fast Copying** – only metadata record are updated when you pull packages from an Upstream CodeArtifact Repository into a Downstream
- **Easy Sharing Across Repositories and Teams** – all the assets and metadata in a domain are encrypted with a single AWS KMS Key
- **Apply Policy Across Multiple Repositories** – domain administrator can apply policy across the domain such as:
  - Restricting which accounts have access to repositories in the domain
  - Who can configure connections to public repositories to use as sources of packages



# Amazon CodeGuru



- An ML-powered service for automated code reviews and application performance recommendations
- Provides two functionalities
  - **CodeGuru Reviewer:** automated code reviews for static code analysis (development)
  - **CodeGuru Profiler:** visibility/recommendations about application performance during runtime (production)



# Amazon CodeGuru Reviewer

- Identify critical issues, security vulnerabilities, and hard-to-find bugs
- Example: common coding best practices, resource leaks, security detection, input validation
- Uses Machine Learning and automated reasoning
- Hard-learned lessons across millions of code reviews on 1000s of open-source and Amazon repositories
- Supports Java and Python
- Integrates with GitHub, Bitbucket, and AWS CodeCommit

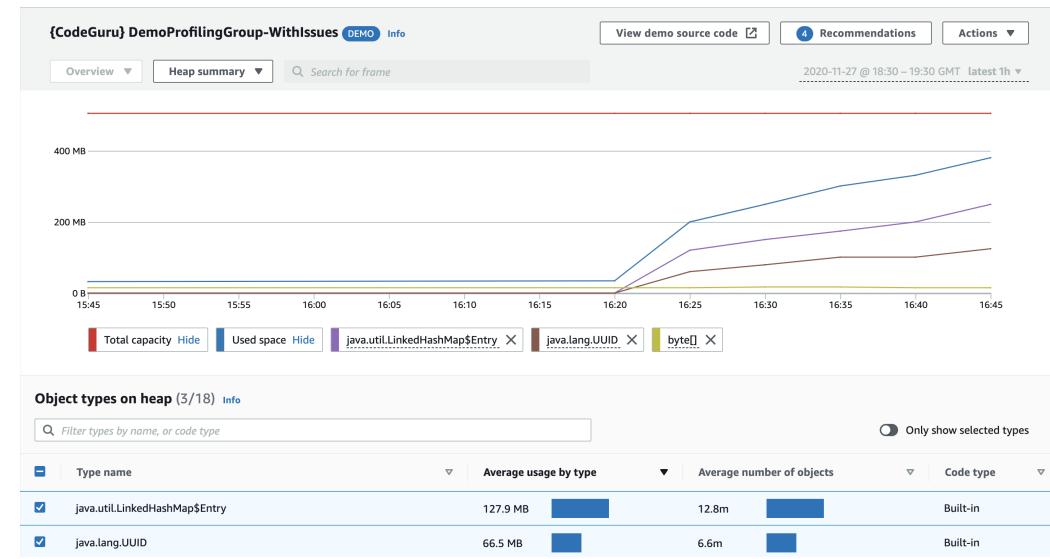
The screenshot shows the Amazon CodeGuru Reviewer interface. At the top, there's a navigation bar with 'CodeGuru' > 'Code reviews' > 'mw2tsa56o0000000'. Below it is a card for 'RepositoryAnalysis-amazon-codeguru-reviewer-sample-app-master-mw2tsa56o0000000'. The card displays details like Status (Completed), Recommendations (4), Metered lines of code (80), Time created (10 Nov 2020 08:08:47 AM GMT-0800), and Last updated (10 Nov 2020 08:11:44 AM GMT-0800). To the right, there's a sidebar with repository information: Type (RepositoryAnalysis), Provider (GitHub), Repository (amazon-codeguru-reviewer-sample-app), and Branch name (master). Below the main card, there's a section for 'Recommendations (4)' with a search bar. Three specific recommendations are listed:

- EventHandler.java Line: 79**: This code appears to be waiting for a resource before it runs. You could use the waiters feature to help improve efficiency. Consider using ObjectExists or ObjectNotExists. For more information, see <https://aws.amazon.com/blogs/developer/waiters-in-the-aws-sdk-for-java/>.  
Was this helpful?
- EventHandler.java Line: 100**: This code might not produce accurate results if the operation returns paginated results instead of all results. Consider adding another call to check for additional results.  
Was this helpful?
- EventHandler.java Line: 100**: This code uses an outdated API. [ListObjectsV2](#) is the revised List Objects API, and we recommend you use this revised API for new application developments.  
Was this helpful?

<https://aws.amazon.com/codeguru/features/>

# Amazon CodeGuru Profiler

- Helps understand the runtime behavior of your application
- Example: identify if your application is consuming excessive CPU capacity on a logging routine
- Features:
  - Identify and remove code inefficiencies
  - Improve application performance (e.g., reduce CPU utilization)
  - Decrease compute costs
  - Provides heap summary (identify which objects using up memory)
  - Anomaly Detection
- Support applications running on AWS or on-premise
- Minimal overhead on application



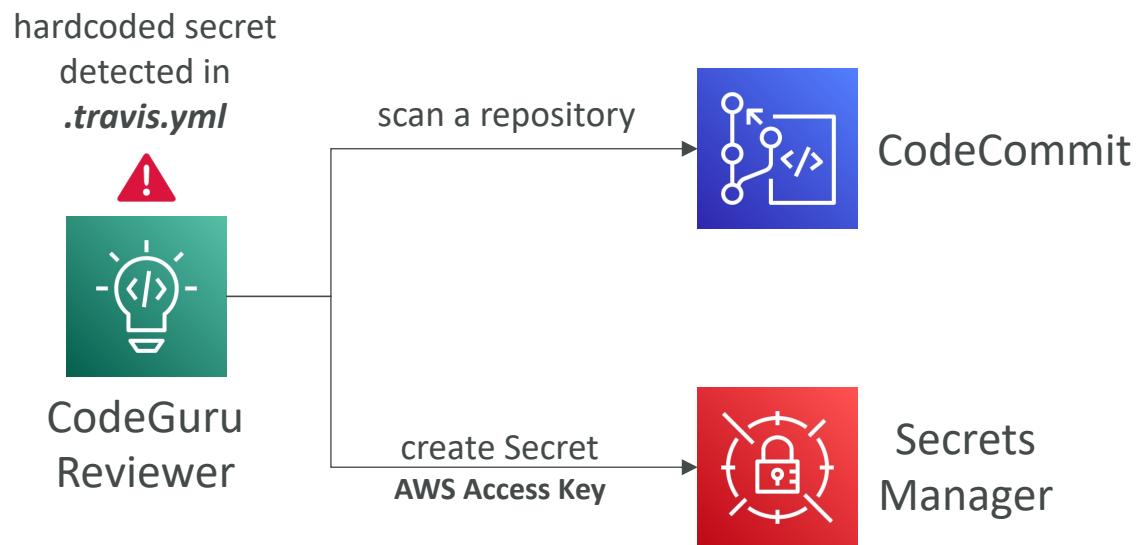
<https://aws.amazon.com/codeguru/features/>

# Amazon CodeGuru Reviewer - Extras

- **CodeGuru Reviewer Secrets Detector**

- Uses ML to identify hardcoded secrets embedded in your code (e.g., passwords, API keys, credentials, SSH keys...)
- Besides scanning code, it scans configuration and documentation files
- **Suggests remediation to automatically protect your secrets with Secrets Manager**

The screenshot shows the 'Recommendations (43)' section of the CodeGuru Reviewer interface. A search bar at the top has 'Type = Secrets' selected. Below it, a message states: 'It appears your code contains a hardcoded AWS Access Key ID. Hardcoded secrets or credentials can allow attackers to bypass authentication methods and perform malicious actions. We recommend revoking access to resources using this credential and storing future credentials in a management service such as AWS Secrets Manager.' A 'Protect your credential' button is highlighted with a yellow border. At the bottom, there's a link to a blog post: <https://aws.amazon.com/blogs/aws/codeguru-reviewer-secrets-detector-identify-hardcoded-secrets/>.



# Amazon CodeGuru Profiler - Extras

- Integrate and apply CodeGuru Profiler to Lambda functions either using:
- Function Decorator `@with_lambda_profiler`
  - Add `codeguru_profiler_agent` dependency to your Lambda function .zip file or use Lambda Layers

```
from codeguru_profiler_agent import with_lambda_profiler

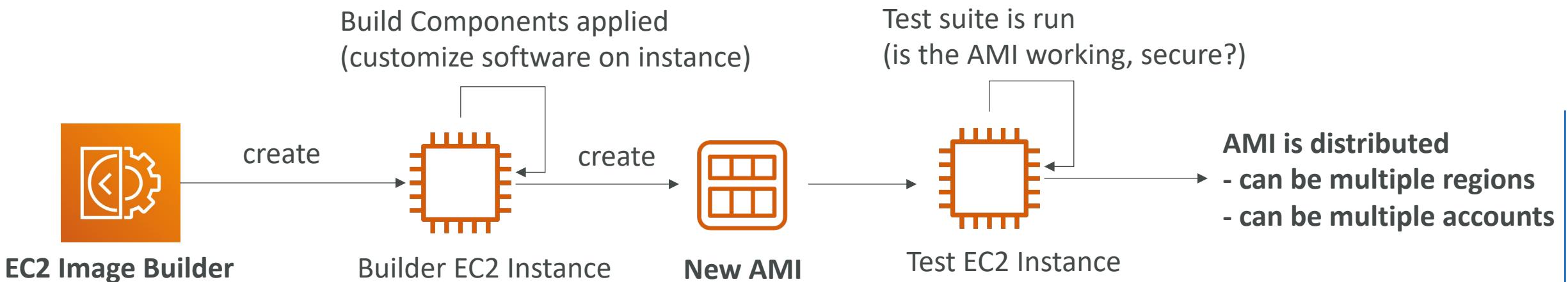
@with_lambda_profiler(profiling_group_name="MyGroupName")
def handler_name(event, context):
    return "Hello World"
```

- Enable Profiling in the Lambda function configuration

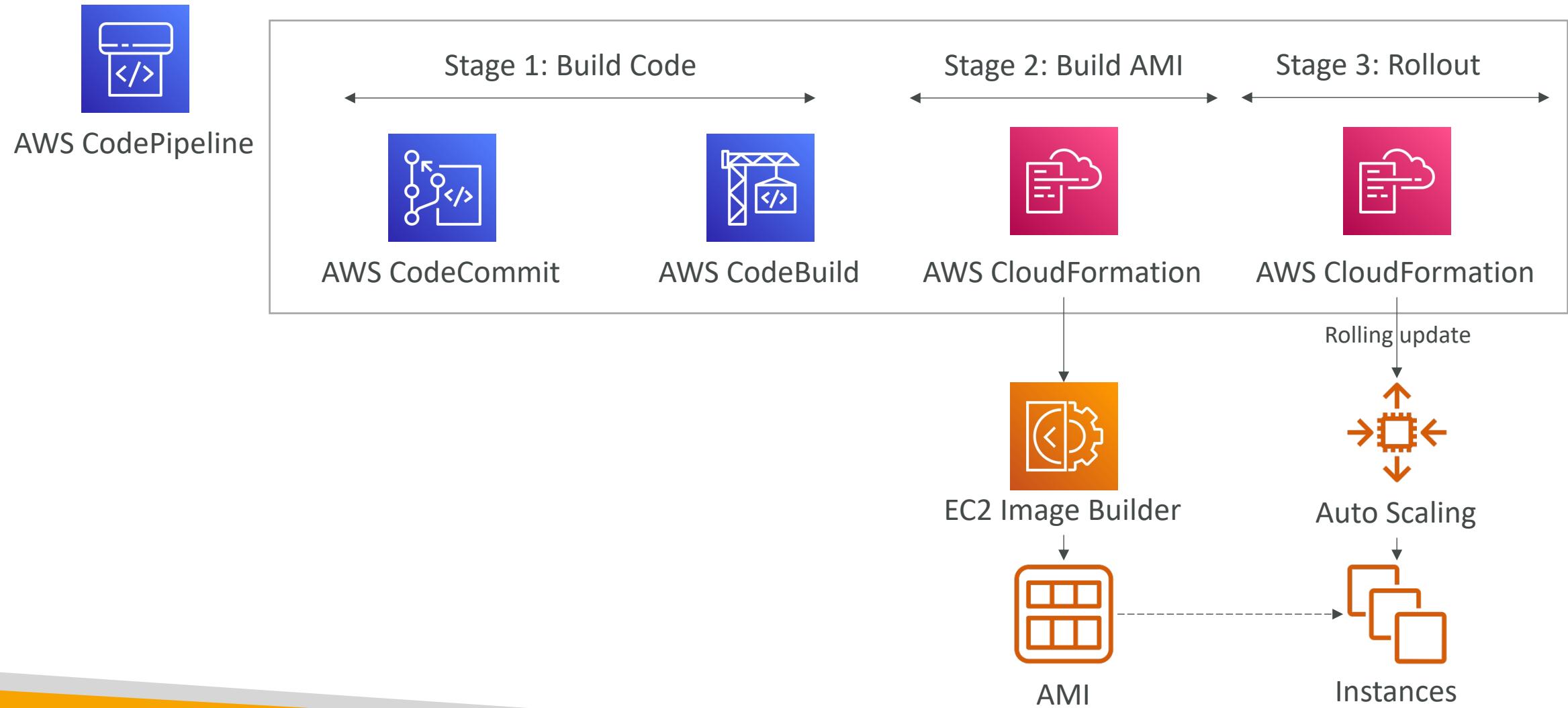
# EC2 Image Builder



- Used to automate the creation of Virtual Machines or container images
- => Automate the creation, maintain, validate and test EC2 AMIs
- Can be run on a schedule (weekly, whenever packages are updated, etc...)
- Free service (only pay for the underlying resources)
- Can publish AMI to multiple regions and multiple accounts

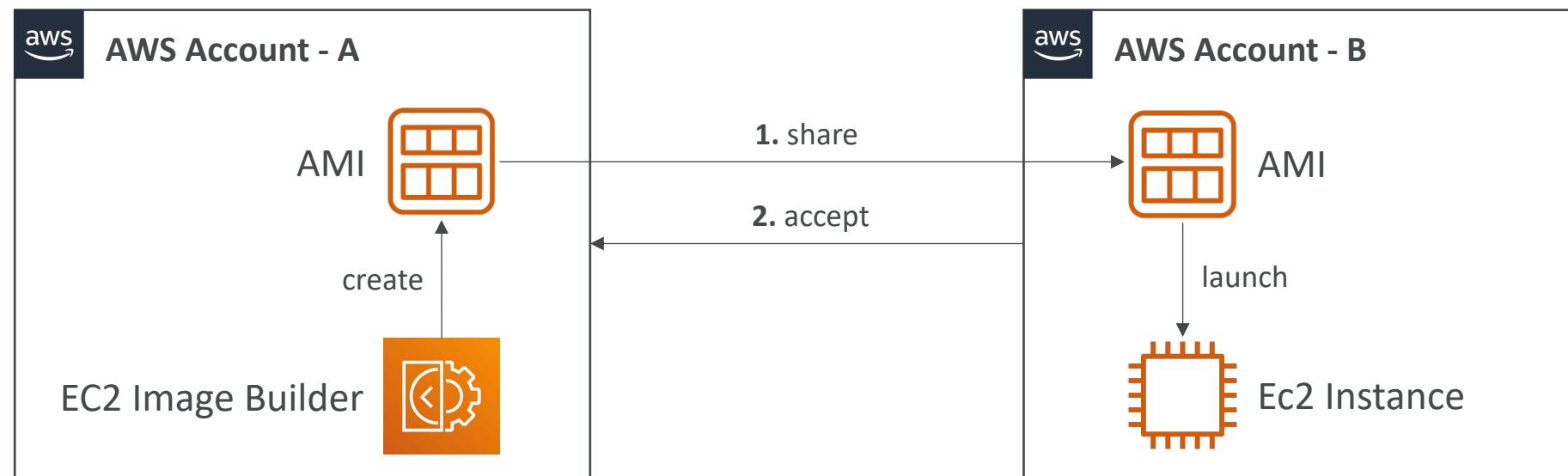


# EC2 Image Builder – CICD Architecture

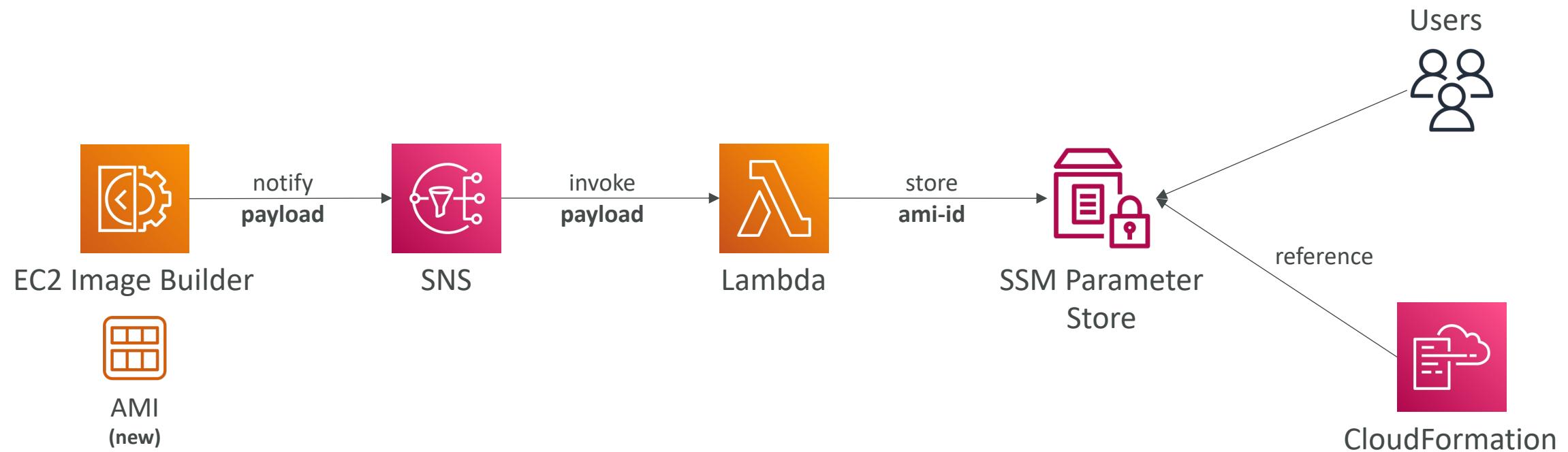


# EC2 Image Builder – Sharing using RAM

- Use AWS RAM to share **Images, Recipes, and Components** across AWS accounts or through AWS Organization



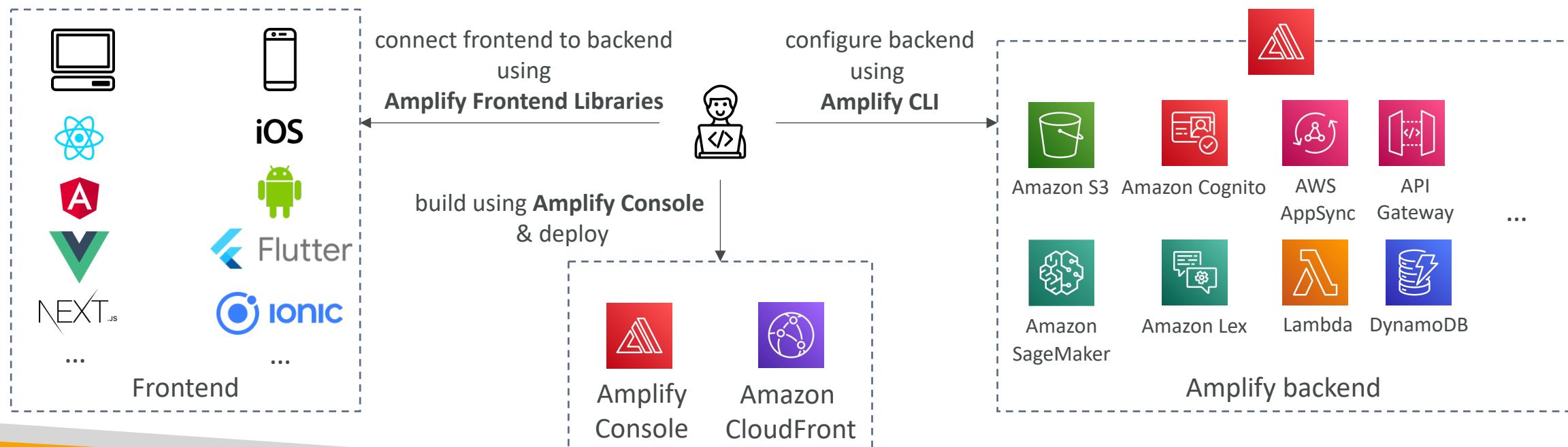
# EC2 Image Builder – Tracking Latest AMIs





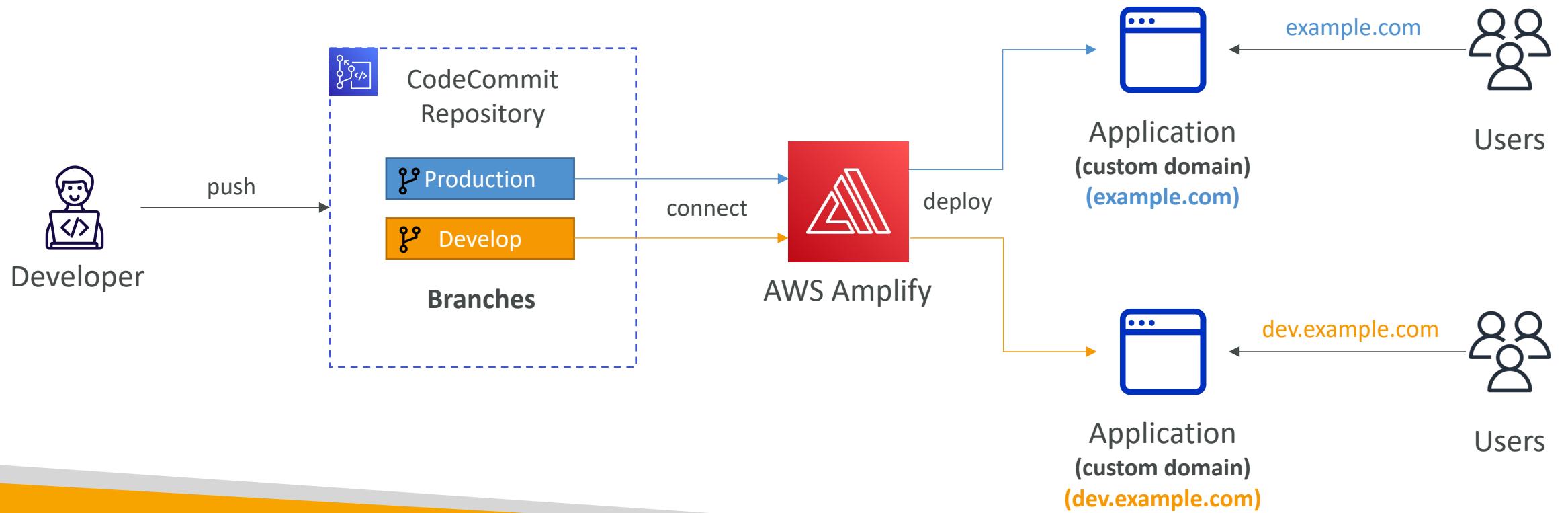
# AWS Amplify - web and mobile applications

- A set of tools and services that helps you develop and deploy scalable full stack web and mobile applications
- Authentication, Storage, API (REST, GraphQL), CI/CD, PubSub, Analytics, AI/ML Predictions, Monitoring...
- Connect your source code from GitHub, AWS CodeCommit, Bitbucket, GitLab, or upload directly



# AWS Amplify – Continuous Deployment

- Connect to CodeCommit and have one deployment per branch
- Connect your application to a custom domain (e.g., Route 53)



# Domain 2 - Configuration Management and IaC

# AWS CloudFormation



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

# CloudFormation – Template Example

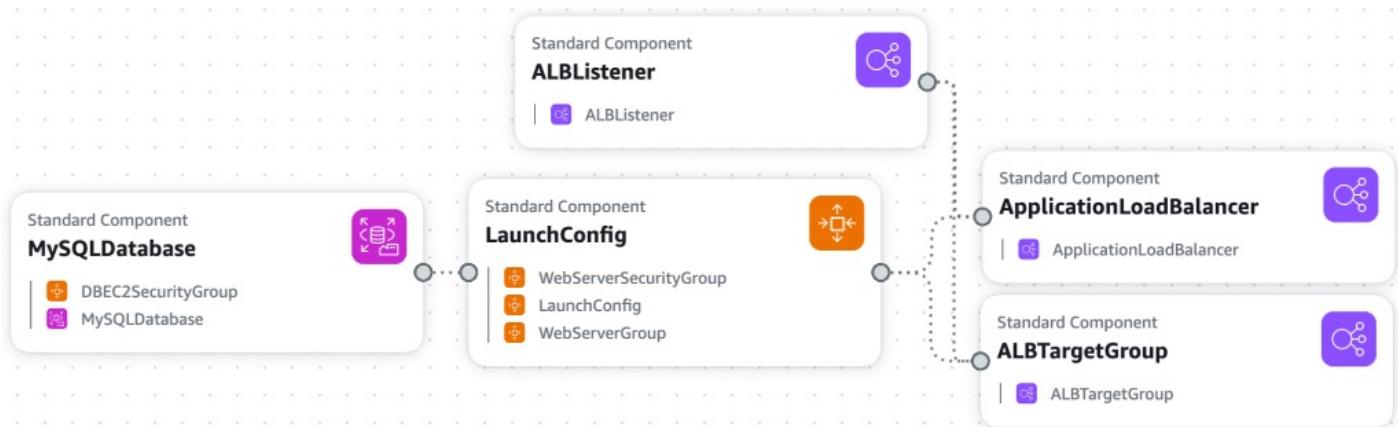
```

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

```



## Application Composer



# Benefits of AWS CloudFormation (1/2)

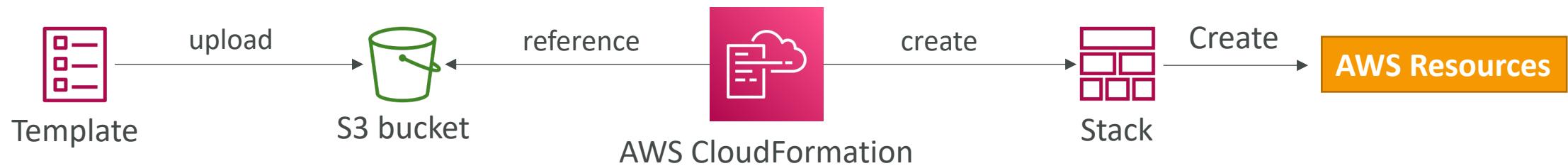
- **Infrastructure as code**
  - No resources are manually created, which is excellent for control
  - The code can be version controlled for example using Git
  - Changes to the infrastructure are reviewed through code
- **Cost**
  - Each resources within the stack is tagged with an identifier so you can easily see how much a stack costs you
  - You can estimate the costs of your resources using the CloudFormation template
  - Savings strategy: In Dev, you could automation deletion of templates at 5 PM and recreated at 8 AM, safely

# Benefits of AWS CloudFormation (2/2)

- **Productivity**
  - Ability to destroy and re-create an infrastructure on the cloud on the fly
  - Automated generation of Diagram for your templates!
  - Declarative programming (no need to figure out ordering and orchestration)
- **Separation of concern: create many stacks for many apps, and many layers.** Ex:
  - VPC stacks
  - Network stacks
  - App stacks
- **Don't re-invent the wheel**
  - Leverage existing templates on the web!
  - Leverage the documentation

# How CloudFormation Works

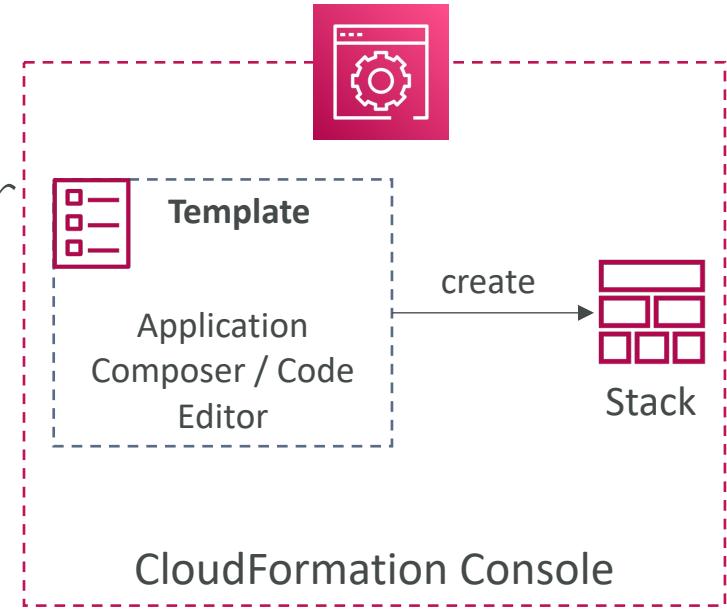
- Templates must be uploaded in S3 and then referenced in CloudFormation
- To update a template, we can't edit previous ones. We have to re-upload a new version of the template to AWS
- Stacks are identified by a name
- Deleting a stack deletes every single artifact that was created by CloudFormation.



# Deploying CloudFormation Templates

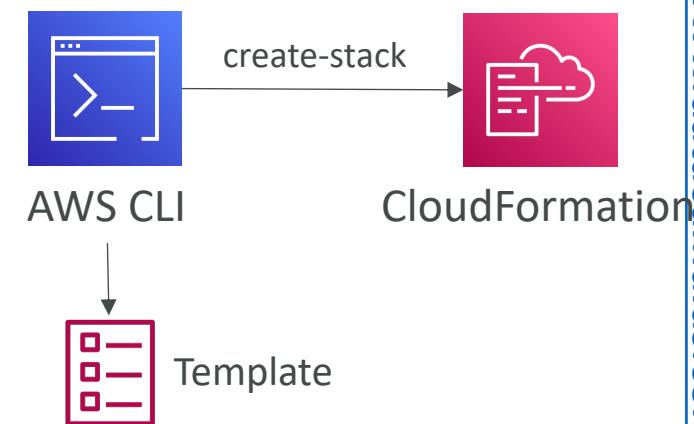
- **Manual way**

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



- **Automated way**

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

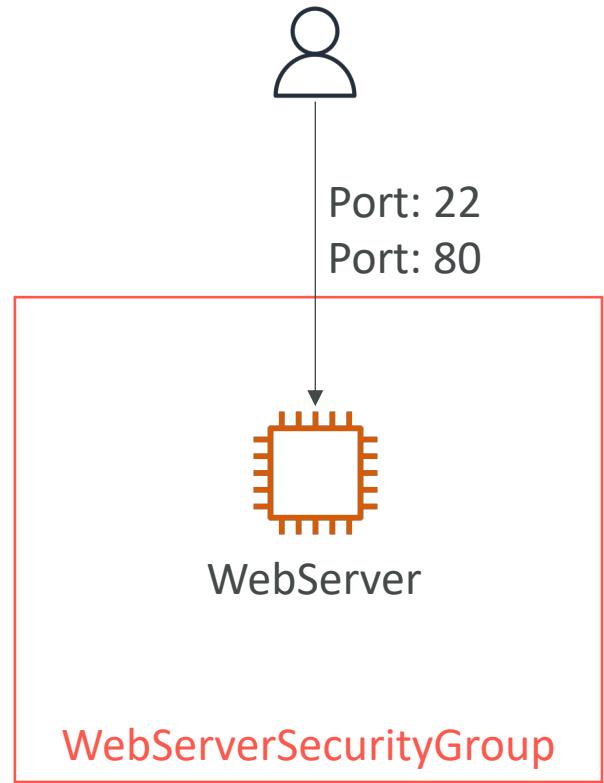


# CloudFormation – Building Blocks

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

# Introductory Example

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



# YAML Crash Course

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

- YAML and JSON are the languages you can use for CloudFormation
- JSON is horrible for CF
- YAML is great in so many ways
- Let's learn a bit about it!
  - Key value Pairs
  - Nested objects
  - Support Arrays
  - Multi line strings
  - Can include comments!

# CloudFormation – Resources

- Resources are the core of your CloudFormation template (**MANDATORY**)
- They represent the different AWS Components that will be created and configured
- Resources are declared and can reference each other
- AWS figures out creation, updates and deletes of resources for us
- There are over 700 types of resources (!)
- Resource types identifiers are of the form:

***service-provider::service-name::data-type-name***

# How do I find Resources documentation?

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

# Analysis of CloudFormation Template

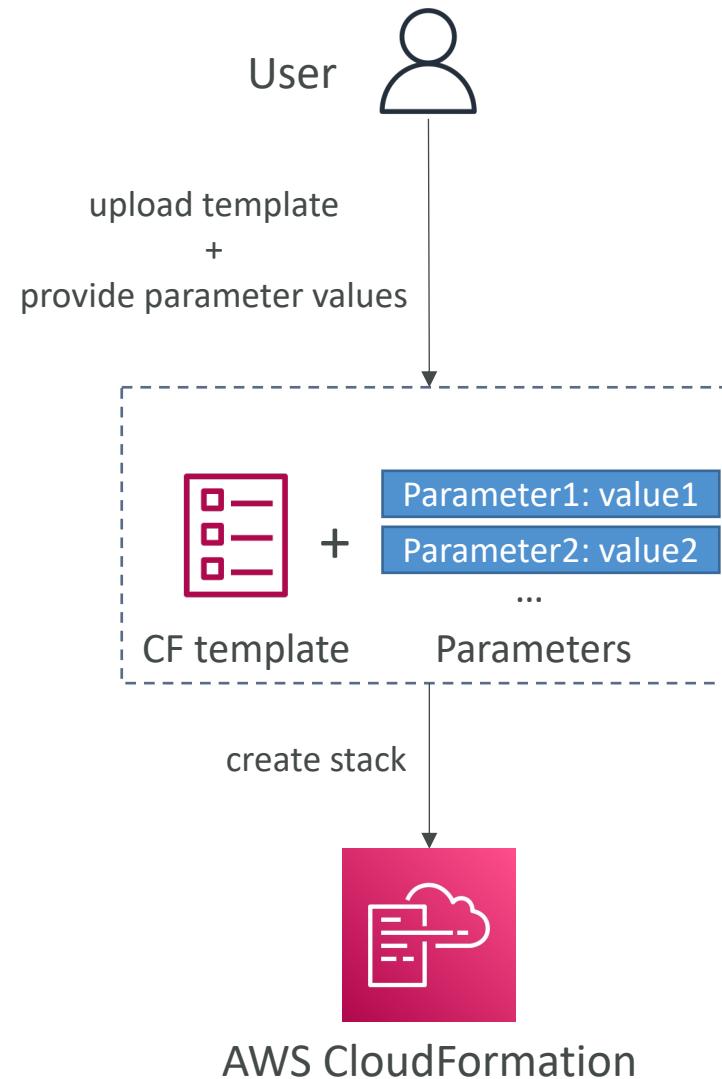
- Going back to the example of the introductory lecture, let's learn why it was written this way.
- Relevant documentation can be found here:
  - <https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-ec2-instance.html>
  - <https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-ec2-securitygroup.html>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

# CloudFormation – Resources FAQ

- Can I create a dynamic number of resources?
  - Yes, you can by using CloudFormation Macros and Transform
  - It is not in the scope of this course
- Is every AWS Service supported?
  - Almost. Only a select few niches are not there yet
  - You can work around that using CloudFormation Custom Resources

# CloudFormation – Parameters

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



# When should you use a Parameter?

**Parameters:**

**SecurityGroupDescription:**

**Description:** Security Group Description

**Type:** String

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

# CloudFormation – Parameters Settings

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

# CloudFormation – Parameters Example

## AllowedValues

### Parameters:

#### InstanceType:

Description: Choose an EC2 instance type

Type: String

#### AllowedValues:

- t2.micro
- t2.small
- t2.medium

Default: t2.micro

### Resources:

#### MyEC2Instance:

Type: AWS::EC2::Instance

#### Properties:

InstanceType: !Ref InstanceType  
ImageId: ami-0c02fb55956c7d316

## NoEcho

### Parameters:

#### DBPassword:

Description: The database admin password

Type: String

NoEcho: true

### Resources:

#### MyDBInstance:

Type: AWS::RDS::DBInstance

#### Properties:

DBInstanceClass: db.t2.micro  
AllocatedStorage: 20  
Engine: mysql  
MasterUsername: admin  
MasterUserPassword: !Ref DBPassword  
DBInstanceIdentifier: mydbinstance

# How to Reference a Parameter?

**Resources:**

**DBSubnet1:**

Type: AWS::EC2::Subnet

**Properties:**

VpcId: !Ref MyVPC

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

# CloudFormation – Pseudo Parameters

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

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

# CloudFormation – Mappings

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

**Mappings:**

**Mapping01:**

**Key01:**

**Name:** Value01

**Key02:**

**Name:** Value02

**Key03:**

**Name:** Value03

**RegionMap:**

**us-east-1:**

HVM64: ami-0ff8a91507f77f867

HVMG2: ami-0a584ac55a7631c0c

**us-west-1:**

HVM64: ami-0bdb828fd58c52235

HVMG2: ami-066ee5fd4a9ef77f1

**eu-west-1:**

HVM64: ami-047bb4163c506cd98

HVMG2: ami-0a7c483d527806435

# Accessing Mapping Values (Fn::FindInMap)

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

Mappings:

RegionMap:

us-east-1:

HVM64: ami-0ff8a91507f77f867  
HVMG2: ami-0a584ac55a7631c0c

us-west-1:

HVM64: ami-0bdb828fd58c52235  
HVMG2: ami-066ee5fd4a9ef77f1

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

Resources:

MyEC2Instance:

Type: AWS::EC2::Instance

Properties:

ImageId: `!FindInMap [RegionMap, !Ref "AWS::Region", HVM64]`

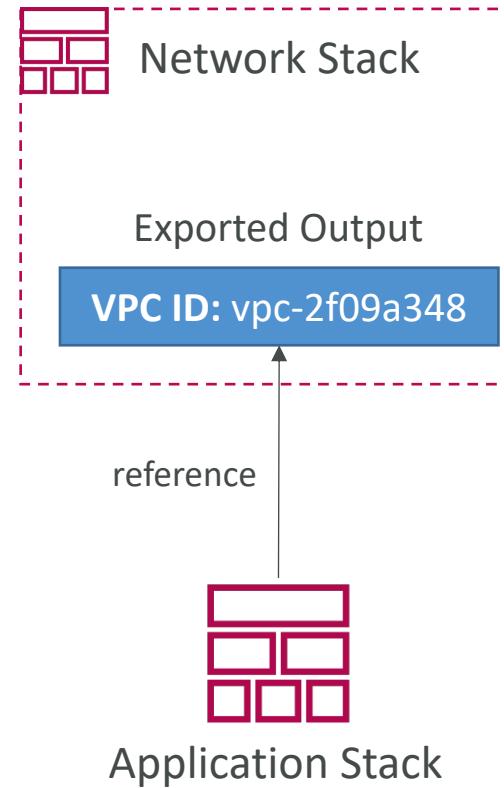
InstanceType: t2.micro

# When would you use Mappings vs. Parameters?

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

# CloudFormation – Outputs

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



# CloudFormation – Outputs

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

## Outputs:

### StackSSHSecurityGroup:

Description: The SSH Security Group for our Company

Value: !Ref MyCompanyWideSSHSecurityGroup

### Export:

Name: SSHSecurityGroup

# CloudFormation – Outputs Cross-Stack Reference

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

**Resources:**

**MySecureInstance:**

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

**Properties:**

**ImageId:** ami-0742b4e673072066f

**InstanceType:** t2.micro

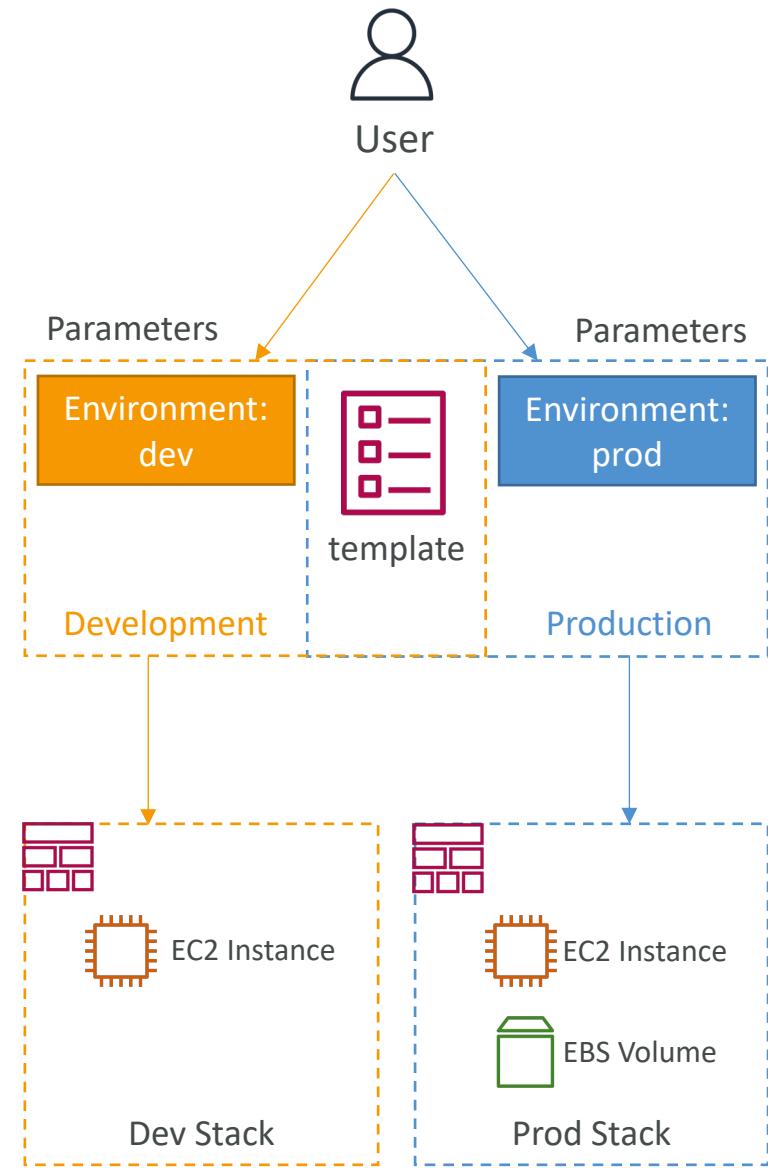
**AvailabilityZone:** us-east-1a

**SecurityGroups:**

- !ImportValue SSHSecurityGroup

# CloudFormation – Conditions

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



# How to define a Condition

## Conditions:

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

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

# How to use a Condition

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

**Resources:**

**MountPoint:**

**Type:** AWS::EC2::VolumeAttachment

**Condition:** CreateProdResources

# CloudFormation – Intrinsic Functions

Blue = must know

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

# Intrinsic Functions – Fn::Ref

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

**Resources:**

**DBSubnet1:**

**Type: AWS::EC2::Subnet**

**Properties:**

**VpcId: !Ref MyVPC**

# Intrinsic Functions – Fn::GetAtt

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

**Resources:**

**EC2Instance:**

Type: AWS::EC2::Instance

**Properties:**

ImageId: ami-0742b4e673072066f

InstanceType: t2.micro

**EBSVolume:**

Type: AWS::EC2::Volume

Condition: CreateProdResources

**Properties:**

Size: 100

AvailabilityZone: !GetAtt EC2Instance.AvailabilityZone

# Intrinsic Functions – Fn::FindInMap

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

```
Mappings:  
RegionMap:  
    us-east-1:  
        HVM64: ami-0ff8a91507f77f867  
        HVMG2: ami-0a584ac55a7631c0c  
    us-west-1:  
        HVM64: ami-0bdb828fd58c52235  
        HVMG2: ami-066ee5fd4a9ef77f1  
  
Resources:  
MyEC2Instance:  
    Type: AWS::EC2::Instance  
    Properties:  
        ImageId: !FindInMap [RegionMap, !Ref "AWS::Region", HVM64]  
        InstanceType: t2.micro
```

# Intrinsic Functions – Fn::ImportValue

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

**Resources:**

**MySecureInstance:**

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

**Properties:**

**ImageId:** ami-0742b4e673072066f

**InstanceType:** t2.micro

**AvailabilityZone:** us-east-1a

**SecurityGroups:**

- !ImportValue SSHSecurityGroup

# Intrinsic Functions – Fn::Base64

- Convert String to it's Base64 representation

```
!Base64 "ValueToEncode"
```

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

Resources:

WebServer:

Type: AWS::EC2::Instance

Properties:

...

UserData:

```
Fn::Base64: |
#!/bin/bash
dnf update -y
dnf install -y httpd
```

# Intrinsic Functions – Condition Functions

## Conditions:

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

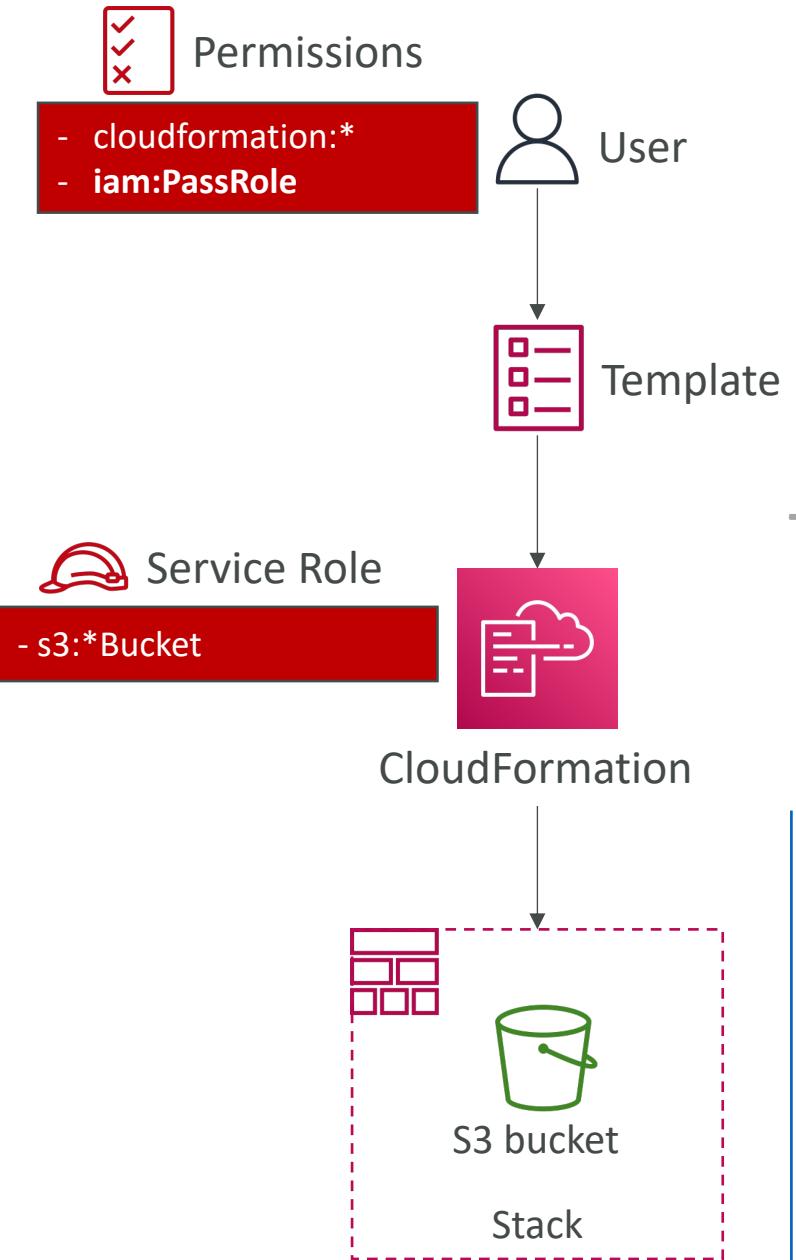
- The logical ID is for you to choose. It's how you name condition
- The intrinsic function (logical) can be any of the following:
  - Fn::And
  - Fn::Equals
  - Fn::If
  - Fn::Not
  - Fn::Or

# CloudFormation – Rollbacks

- Stack Creation Fails:
  - Default: everything rolls back (gets deleted). We can look at the log
  - Option to disable rollback and troubleshoot what happened
- Stack Update Fails:
  - The stack automatically rolls back to the previous known working state
  - Ability to see in the log what happened and error messages
- Rollback Failure? Fix resources manually then issue **ContinueUpdateRollback API** from Console
  - Or from the CLI using continue-update-rollback API call

# CloudFormation – Service Role

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



# CloudFormation Capabilities

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

# CloudFormation – DeletionPolicy Delete

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

Resources:

MyEC2Instance:

Type: AWS::EC2::Instance

Properties:

ImageId: ami-12345678

InstanceType: t2.micro

KeyName: my-key-pair

SecurityGroupIds:

- sg-12345678

**DeletionPolicy: Delete**

Resources:

MyS3Bucket:

Type: AWS::S3::Bucket

**DeletionPolicy: Delete**



# CloudFormation – DeletionPolicy Retain

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

## Resources:

**MyDynamoDBTable:**

Type: AWS::DynamoDB::Table

### Properties:

TableName: MyTable

### AttributeDefinitions:

- AttributeName: ID

AttributeType: S

### KeySchema:

- AttributeName: ID

KeyType: HASH

### ProvisionedThroughput:

ReadCapacityUnits: 5

WriteCapacityUnits: 5

**DeletionPolicy: Retain**

# CloudFormation – DeletionPolicy Snapshot

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

**Resources:**

**MyDBInstance:**

Type: AWS::RDS::DBInstance

**Properties:**

DBInstanceClass: db.t2.micro

AllocatedStorage: 20

Engine: mysql

MasterUsername: admin

MasterUserPassword: "ExamplePassword"

**DeletionPolicy: Snapshot**

# CloudFormation – Stack Policies

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

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

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

# CloudFormation – Termination Protection

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

# CloudFormation – Custom Resources

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

# How to define a Custom Resource?

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



## Resources:

MyCustomResourceUsingLambda:

Type: Custom::MyLambdaResource

## Properties:

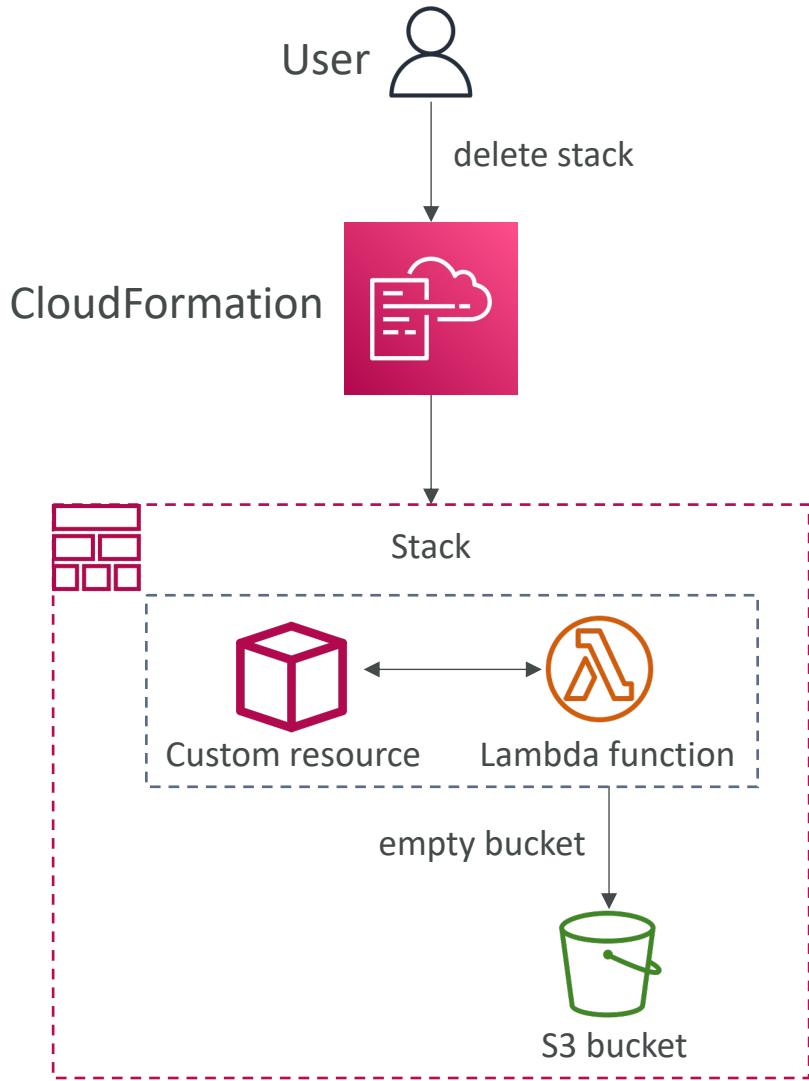
ServiceToken: arn:aws:lambda:REGION:ACCOUNT\_ID:function:FUNCTION\_NAME

# Input values (optional)

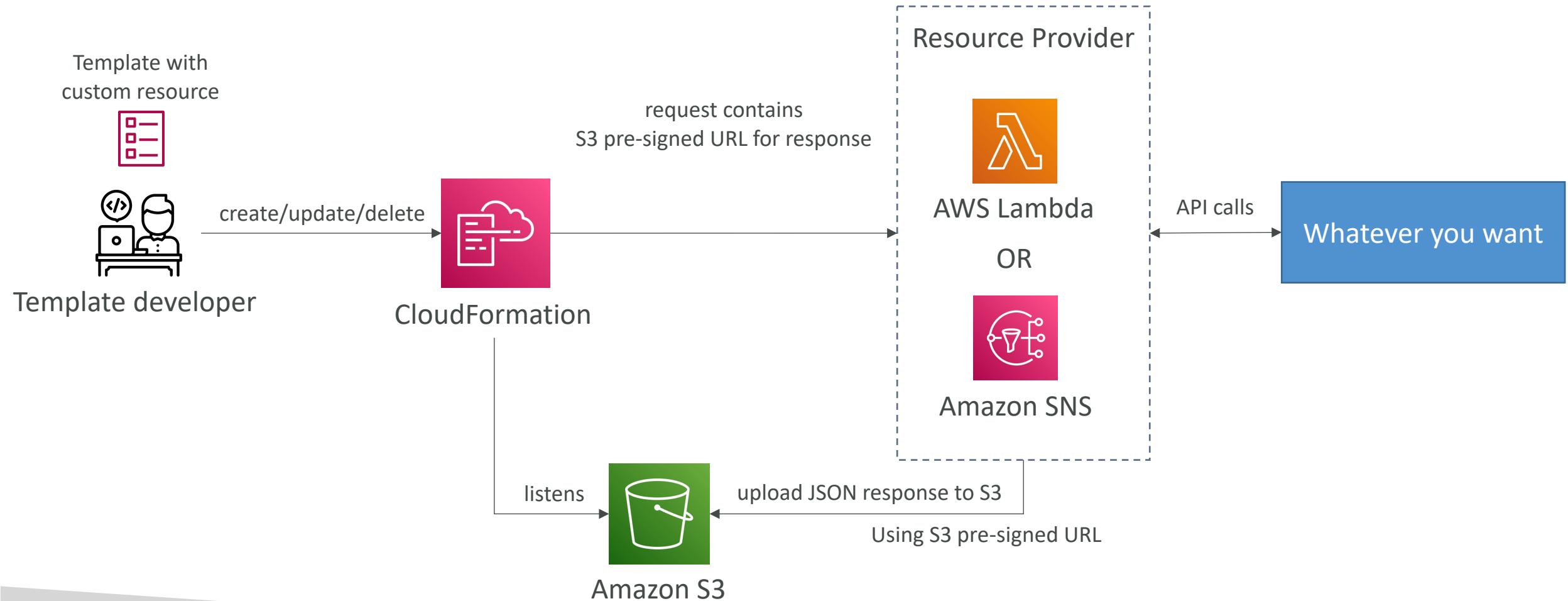
ExampleProperty: "ExampleValue"

# Use Case – Delete content from an S3 bucket

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

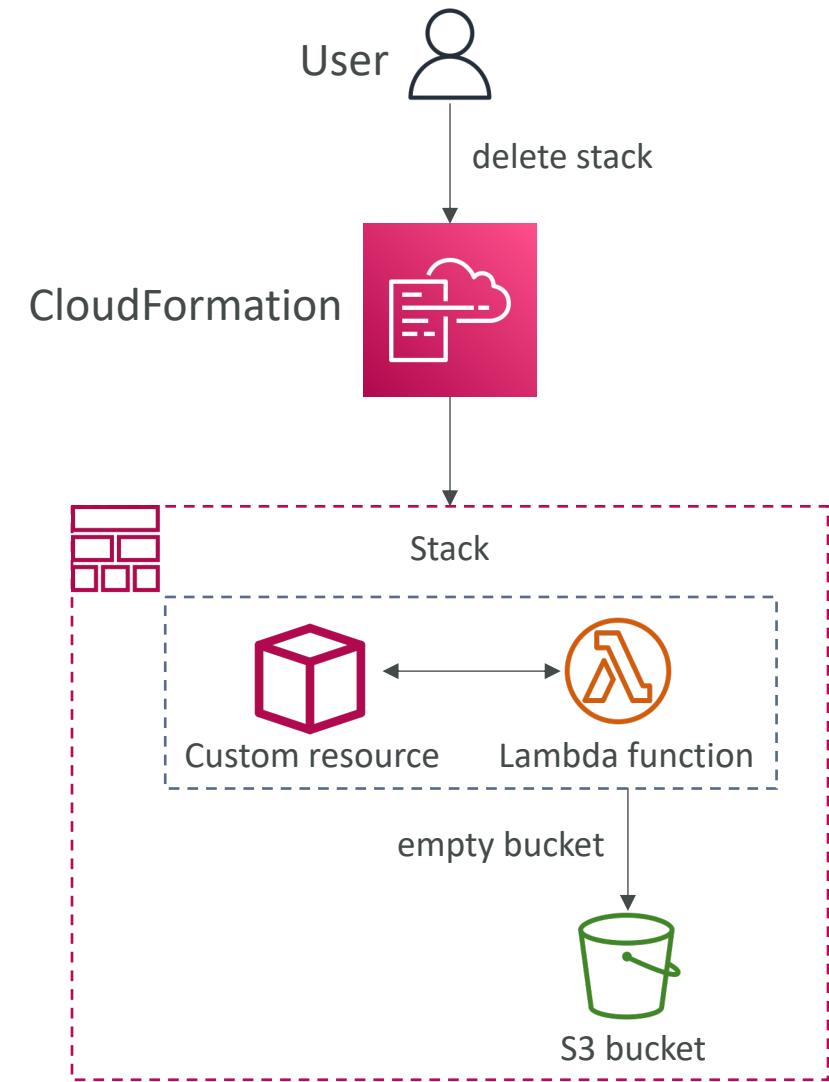


# Custom Resources – How does it work?



# Hands-On: Lambda-backed Custom Resource

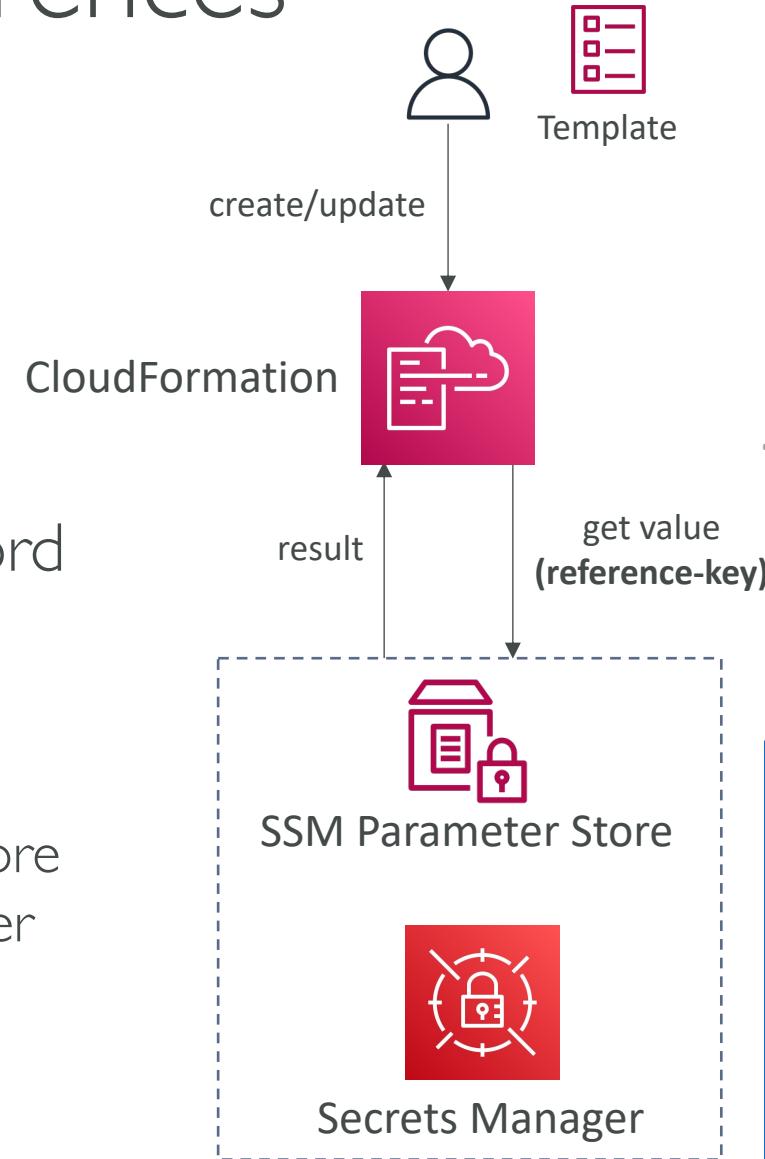
- You can't delete a non-empty S3 bucket
- To delete a non-empty S3 bucket, you must first delete all the objects inside it
- We'll create a custom resource with AWS Lambda that will be used to empty an S3 bucket before deleting it
- Let's create our first Custom Resource!



# CloudFormation – Dynamic References

- Reference external values stored in **Systems Manager Parameter Store** and **Secrets Manager** within CloudFormation templates
- CloudFormation retrieves the value of the specified reference during **create/update/delete operations**
- For example: retrieve RDS DB Instance master password from Secrets Manager
- Supports
  - **ssm** – for plaintext values stored in SSM Parameter Store
  - **ssm-secure** – for secure strings stored in SSM Parameter Store
  - **secretsmanager** – for secret values stored in Secrets Manager

`'{{resolve:service-name:reference-key}}'`



# CloudFormation – Dynamic References

## SSM

```
{resolve:ssm:parameter-name:version}}
```

```
Resources:  
S3Bucket:  
  Type: AWS::S3::Bucket  
Properties:  
  AccessControl: '{{resolve:ssm:S3AccessControl:2}}'
```

## SSM Secure

```
{resolve:ssm-secure:parameter-name:version}}
```

```
Resources:  
IAMUser:  
  Type: AWS::IAM::User  
Properties:  
  UserName: john  
  LoginProfile:  
    Password: '{{resolve:ssm-secure:IAMUserPassword:10}}'
```

## Secrets Manager

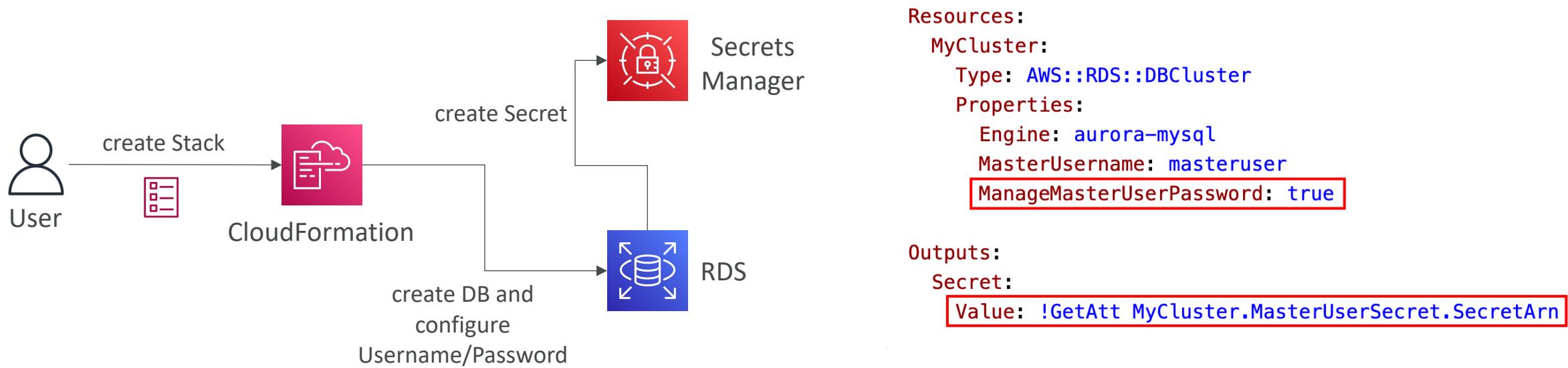
```
{resolve:secretsmanager:secret-id:secret-string:json-key:version-stage:version-id}}
```

```
Resources:  
DBInstance:  
  Type: AWS::RDS::DBInstance  
Properties:  
  DBName: MyRDSInstance  
  MasterUsername: '{{resolve:secretsmanager:MyRDSSecret:SecretString:username}}'  
  MasterUserPassword: '{{resolve:secretsmanager:MyRDSSecret:SecretString:password}}'
```

# CloudFormation, Secrets Manager & RDS

## Option I – ManageMasterUserPassword

- `ManageMasterUserPassword` – creates admin secret implicitly
- RDS, Aurora will manage the secret in Secrets Manager and its rotation



# CloudFormation, Secrets Manager & RDS

## Option 2 – Dynamic Reference

Resources:

```
MyDatabaseSecret:  
  Type: AWS::SecretsManager::Secret  
  Properties:  
    Name: MyDatabaseSecret  
    GenerateSecretString:  
      SecretStringTemplate: '{"username": "admin"}'  
      GenerateStringKey: "password"  
      PasswordLength: 16  
      ExcludeCharacters: '@/\''
```

1. secret is generated

```
MyDBInstance:  
  Type: AWS::RDS::DBInstance  
  Properties:  
    DBName: mydatabase  
    AllocatedStorage: 20  
    DBInstanceClass: db.t2.micro  
    Engine: mysql
```

```
  MasterUsername: '{{resolve:secretsmanager:MyDatabaseSecret:SecretString:username}}'  
  MasterUserPassword: '{{resolve:secretsmanager:MyDatabaseSecret:SecretString:password}}'
```

2. Reference secret in  
RDS DB instance

```
SecretRDSAttachment:  
  Type: AWS::SecretsManager::SecretTargetAttachment  
  Properties:  
    SecretId: !Ref MyDatabaseSecret  
    TargetId: !Ref MyDBInstance  
    TargetType: AWS::RDS::DBInstance
```

3. link the secret to  
RDS DB instance (for rotation)

# User Data in EC2 for CloudFormation

- We can have user data at EC2 instance launch through the console
- Let's learn how to write the same EC2 user-data script in our CloudFormation template
- The important thing to pass is the entire script through the function [Fn::Base64](#)
- Good to know, user data script log is in `/var/log/cloud-init-output.log`
- Let's see how to do this in CloudFormation!

# The Problems with EC2 User Data

- What if we want to have a very large instance configuration?
- What if we want to evolve the state of the EC2 instance without terminating it and creating a new one?
- How do we make EC2 user-data more readable?
- How do we know or signal that our EC2 user-data script completed successfully?
- Enter **CloudFormation Helper Scripts!**
  - Python scripts, that come directly on Amazon Linux AMIs, or can be installed using [yum](#) or [dnf](#) on non-Amazon Linux AMIs
  - `cfn-init`, `cfn-signal`, `cfn-get-metadata`, `cfn-hup`

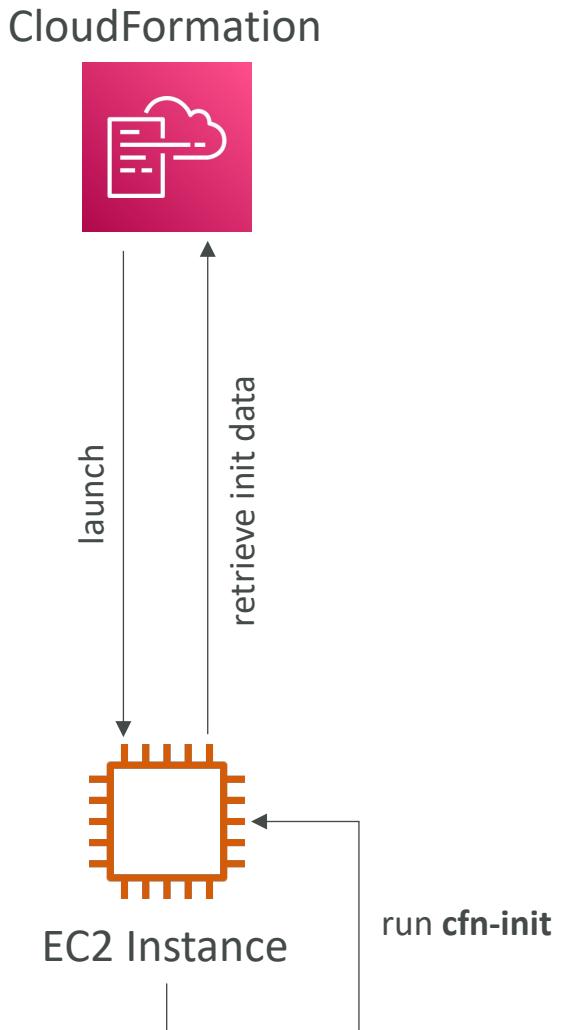
# AWS::CloudFormation::Init

- A `config` contains the following and is executed in that order
  - `Packages`: used to download and install pre-packaged apps and components on Linux/Windows (ex. MySQL, PHP, etc...)
  - `Groups`: define user groups
  - `Users`: define users, and which group they belong to
  - `Sources`: download files and archives and place them on the EC2 instance
  - `Files`: create files on the EC2 instance, using inline or can be pulled from a URL
  - `Commands`: run a series of commands
  - `Services`: launch a list of sysvinit

```
Resources:  
  EC2Instance:  
    Type: AWS::EC2::Instance  
    Properties:  
      ...  
      Metadata:  
        AWS::CloudFormation::Init:  
          config:  
            packages:  
              ...  
            groups:  
              ...  
            users:  
              ...  
            sources:  
              ...  
            files:  
              ...  
            commands:  
              ...  
            services:  
              ...
```

# CloudFormation – cfn-init

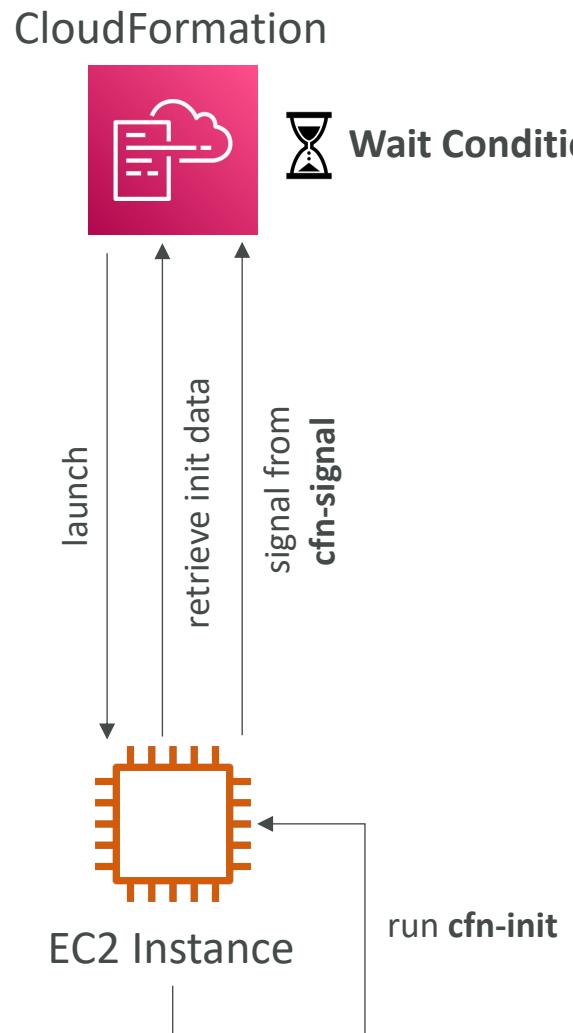
- Used to retrieve and interpret the resource metadata, installing packages, creating files and starting services
- With the **cfn-init** script, it helps make complex EC2 configurations readable
- The EC2 instance will query the CloudFormation service to get init data
- **AWS::CloudFormation::Init** must be in the Metadata of a resource
- Logs go to `/var/log/cfn-init.log`



# CloudFormation – cfn-signal & Wait Conditions

- We still don't know how to tell CloudFormation that the EC2 instance got properly configured after a `cfn-init`
- For this, we can use the `cfn-signal` script!
  - We run `cfn-signal` right after `cfn-init`
  - Tell CloudFormation service that the resource creation success/fail to keep on going or fail
- We need to define `WaitCondition`:
  - Block the template until it receives a signal from `cfn-signal`
  - We attach a `CreationPolicy` (also works on EC2, ASG)
  - We can define a `Count > 1` (in case you need more than 1 signal)

```
CreationPolicy:  
ResourceSignal:  
Timeout: PT5M  
Count: 1
```

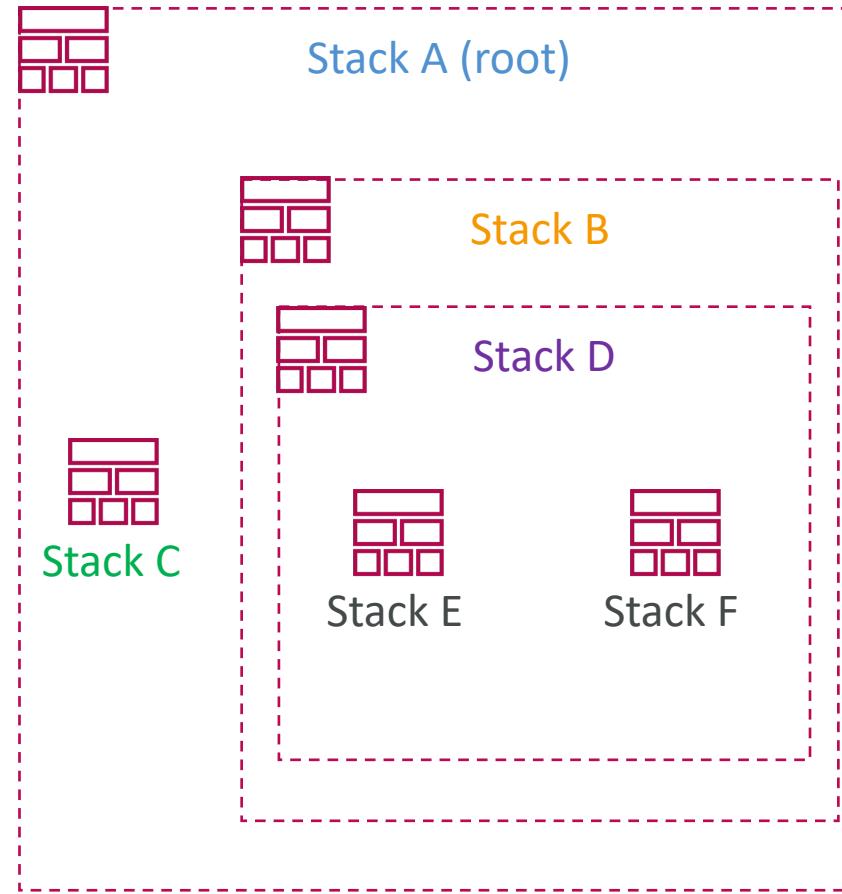


# Wait Condition Didn't Receive the Required Number of Signals from an Amazon EC2 Instance

- Ensure that the AMI you're using has the AWS CloudFormation helper scripts installed. If the AMI doesn't include the helper scripts, you can also download them to your instance
- Verify that the `cfn-init` & `cfn-signal` command was successfully run on the instance. You can view logs, such as `/var/log/cloud-init.log` or `/var/log/cfn-init.log`, to help you debug the instance launch
- You can retrieve the logs by logging in to your instance, but you must disable rollback on failure or else AWS CloudFormation deletes the instance after your stack fails to create
- Verify that the instance has a connection to the Internet. If the instance is in a VPC, the instance should be able to connect to the Internet through a NAT device if it's in a private subnet or through an Internet gateway if it's in a public subnet
- For example, run: `curl -I https://aws.amazon.com`

# CloudFormation – Nested Stacks

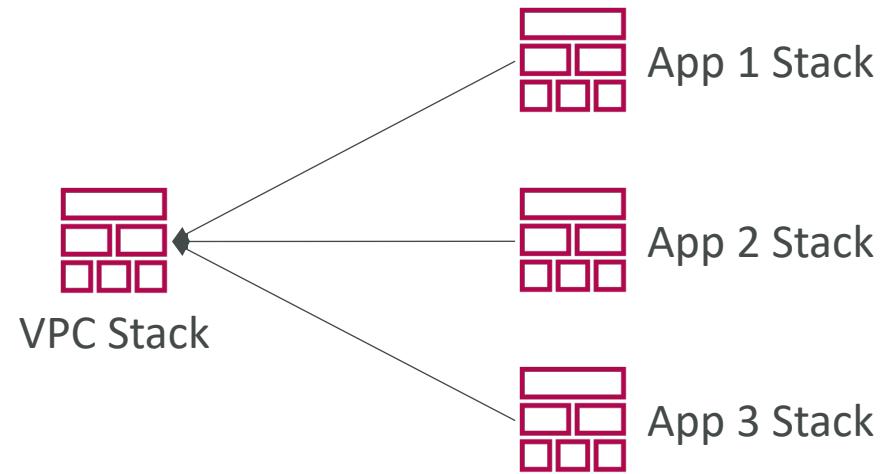
- Nested stacks are stacks as part of other stacks
- They allow you to isolate repeated patterns / common components in separate stacks and call them from other stacks
- Example:
  - Load Balancer configuration that is re-used
  - Security Group that is re-used
- Nested stacks are considered best practice
- To update a nested stack, always update the parent (root stack)
- Nested stacks can have nested stacks themselves!



# Cross Stacks vs. Nested Stacks

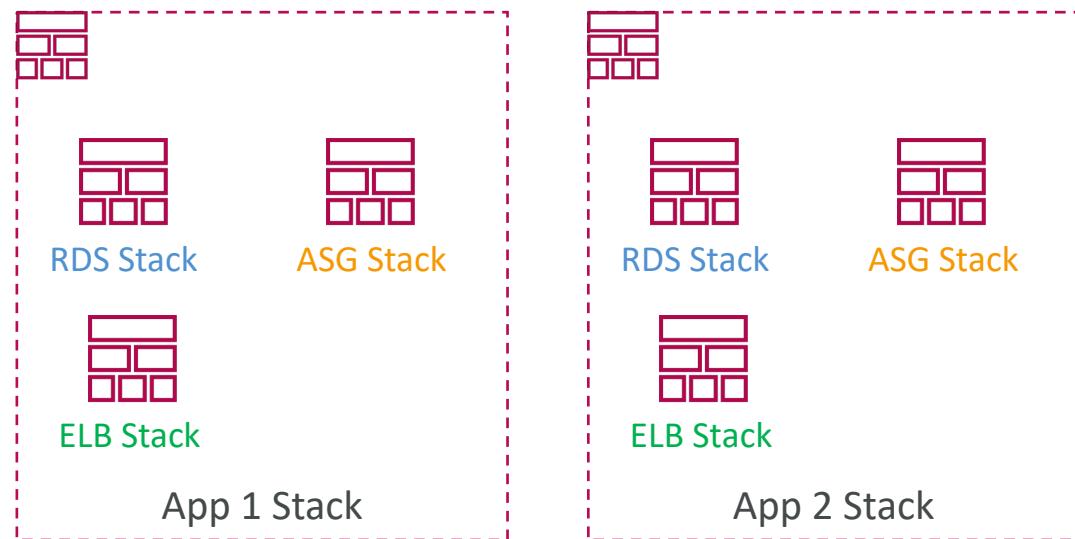
- **Cross Stacks**

- Helpful when stacks have different lifecycles
- Use Outputs Export and Fn::ImportValue
- When you need to pass export values to many stacks (VPC Id...)



- **Nested Stacks**

- Helpful when components must be re-used
- Example: re-use how to properly configure an Application Load Balancer
- The nested stack only is important to the higher-level stack (it's not shared)



# CloudFormation – DependsOn

- Specify that the creation of a specific resource follows another
- When added to a resource, that resource is created only after the creation of the resource specified in the **DependsOn** attribute
- Applied automatically when using **!Ref** and **!GetAtt**
- Use with any resource

**Resources:**

**EC2Instance:**

Type: AWS::EC2::Instance

DependsOn: DBInstance

**Properties:**

ImageId: ami-0a3c3a20c09d6f377

**DBInstance:**

Type: AWS::RDS::DBInstance

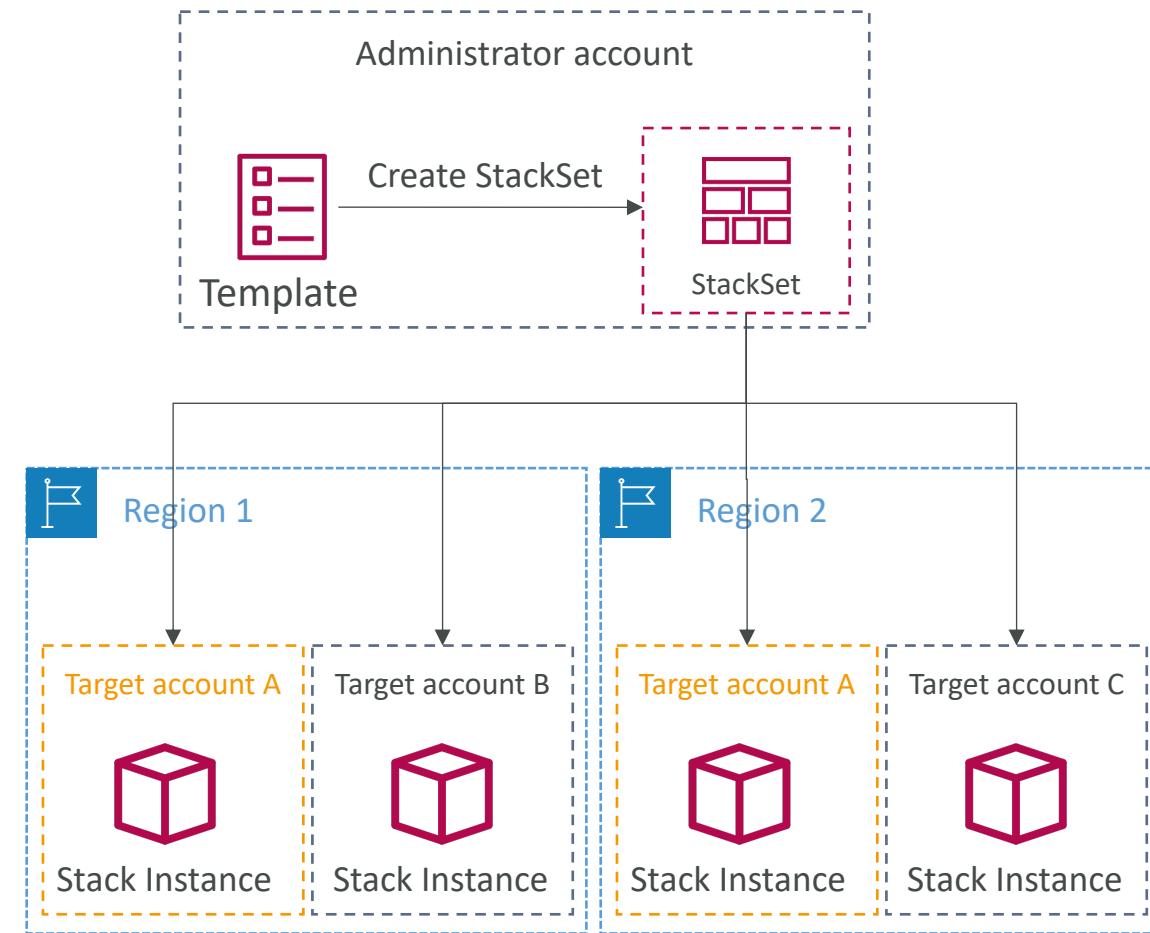
**Properties:**

Engine: MySQL

DBInstanceClass: db.t2.small

# CloudFormation – StackSets

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



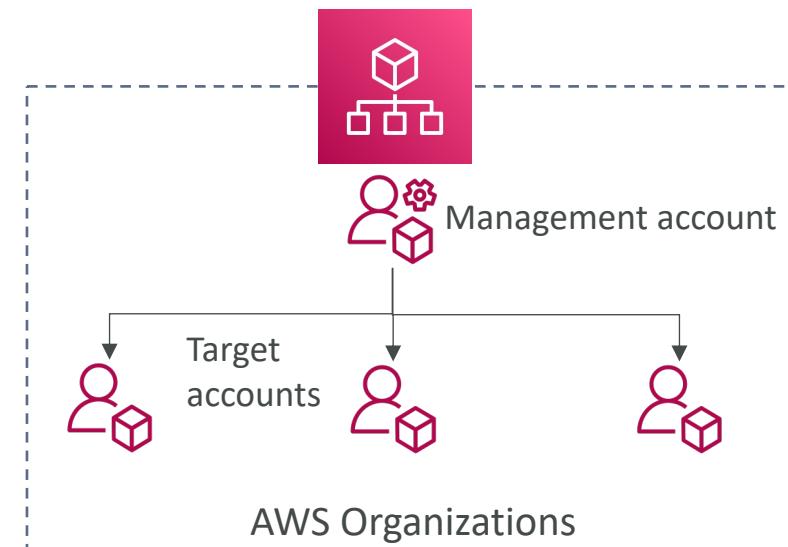
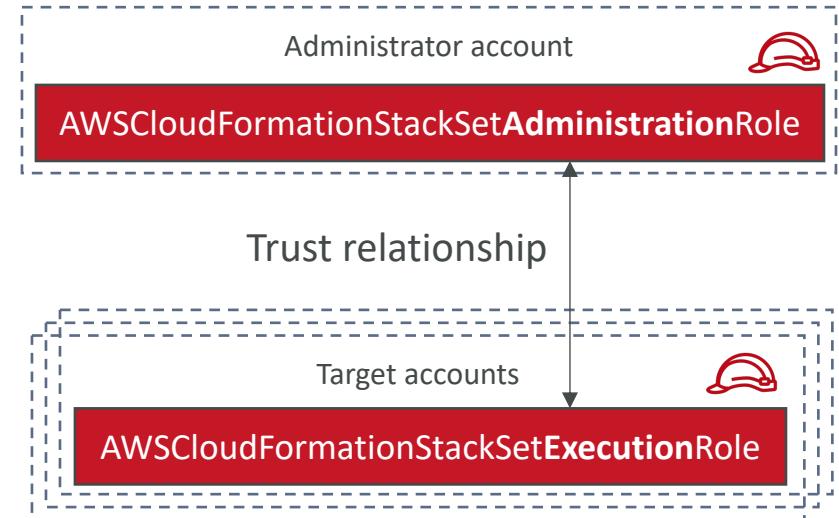
# CloudFormation – StackSets Permission Models

- **Self-managed Permissions**

- Create the IAM roles (with established trusted relationship) in both administrator and target accounts
- Deploy to any target account in which you have permissions to create IAM role

- **Service-managed Permissions**

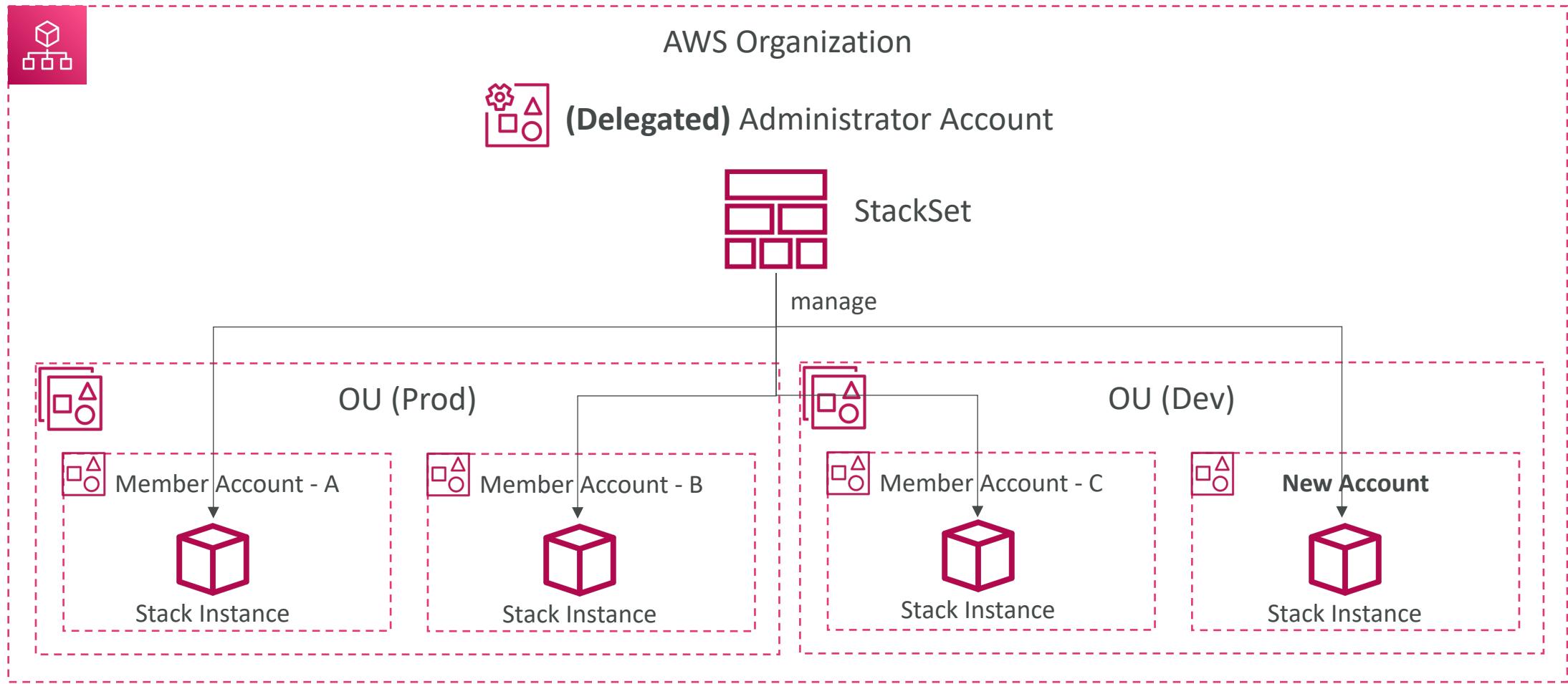
- Deploy to accounts managed by AWS Organizations
- StackSets create the IAM roles on your behalf (**enable trusted access** with AWS Organizations)
- Must **enable all features** in AWS Organizations
- Ability to deploy to accounts added to your organization in the future (Automatic Deployments)



# StackSets with AWS Organizations

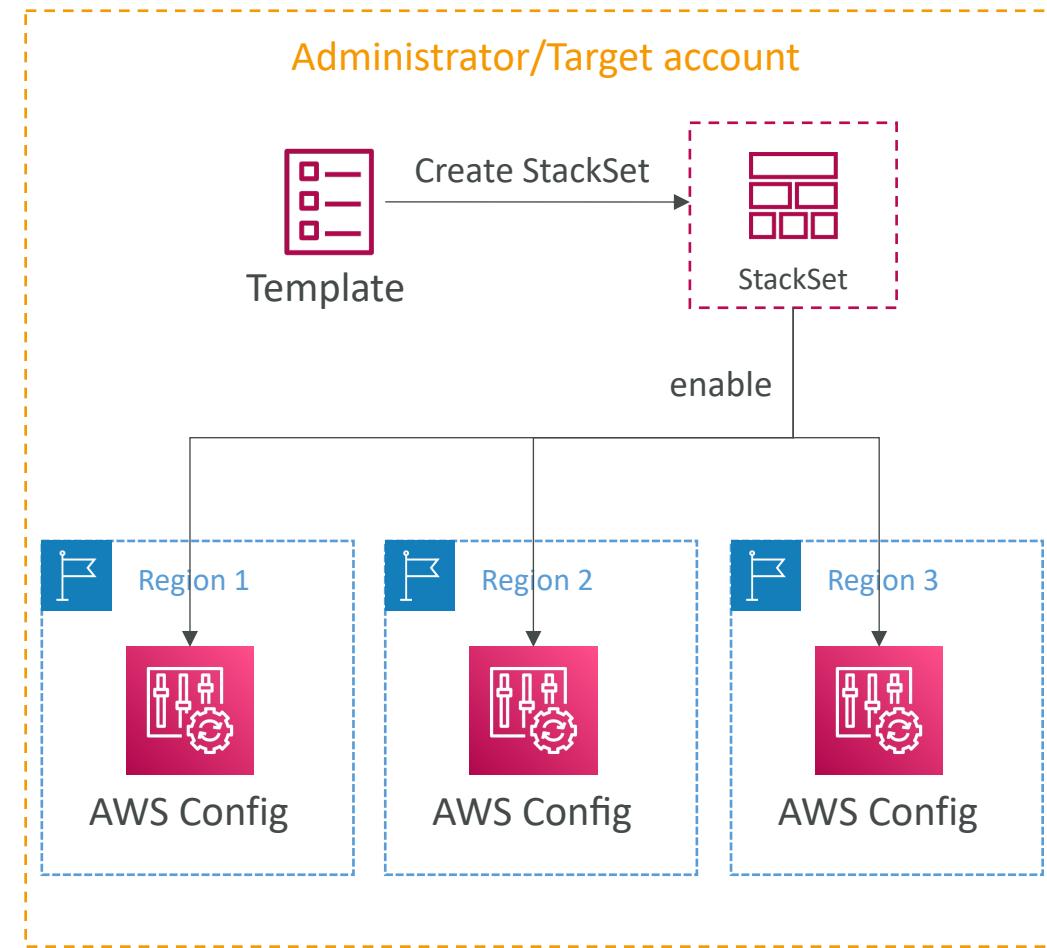
- Ability to automatically deploy Stack instances to new Accounts in an Organization
- Can delegate StackSets administration to member accounts in AWS Organization
- Trusted access with AWS Organizations must be enabled before delegated administrators can deploy to accounts managed by Organizations

# StackSets with AWS Organizations



# Hands-On: StackSets

- We'll use StackSets to enable AWS Config across AWS regions with a single click
- Let's see how this works!



# CloudFormation – Troubleshooting

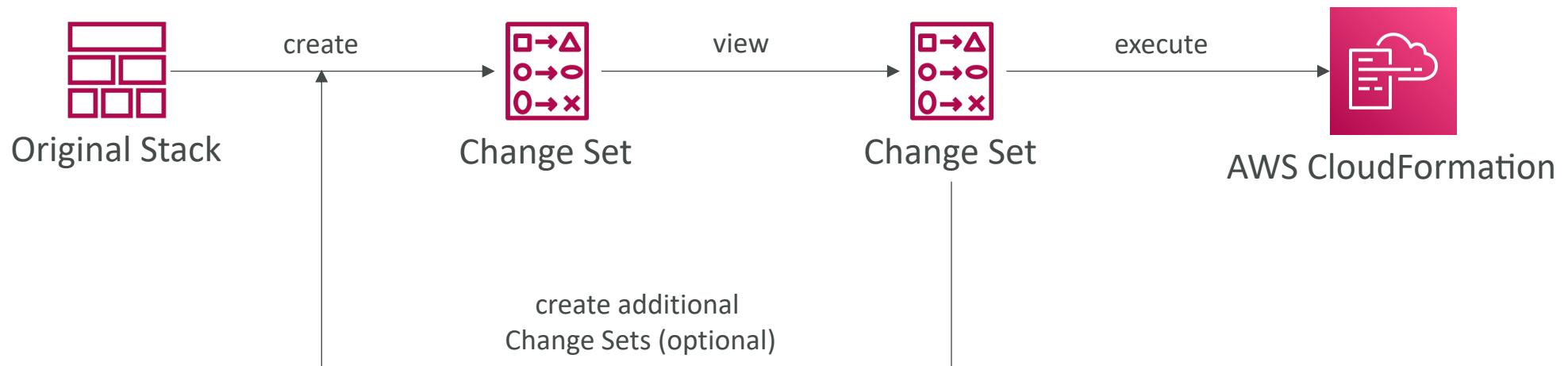
- **DELETE\_FAILED**
  - Some resources must be emptied before deleting, such as S3 buckets
  - Use Custom Resources with Lambda functions to automate some actions
  - Security Groups cannot be deleted until all EC2 instances in the group are gone
  - Think about using `DeletionPolicy=Retain` to skip deletions
- **UPDATE\_ROLLBACK\_FAILED**
  - Can be caused by resources changed outside of CloudFormation, insufficient permissions, Auto Scaling Group that doesn't receive enough signals...
  - Manually fix the error and then `ContinueUpdateRollback`

# CloudFormation – StackSet Troubleshooting

- A stack operation failed, and the stack instance status is **OUTDATED**.
  - Insufficient permissions in a target account for creating resources that are specified in your template.
  - The template could be trying to create global resources that must be unique but aren't, such as S3 buckets
  - The administrator account does not have a trust relationship with the target account
  - Reached a limit or a quota in the target account (too many resources)

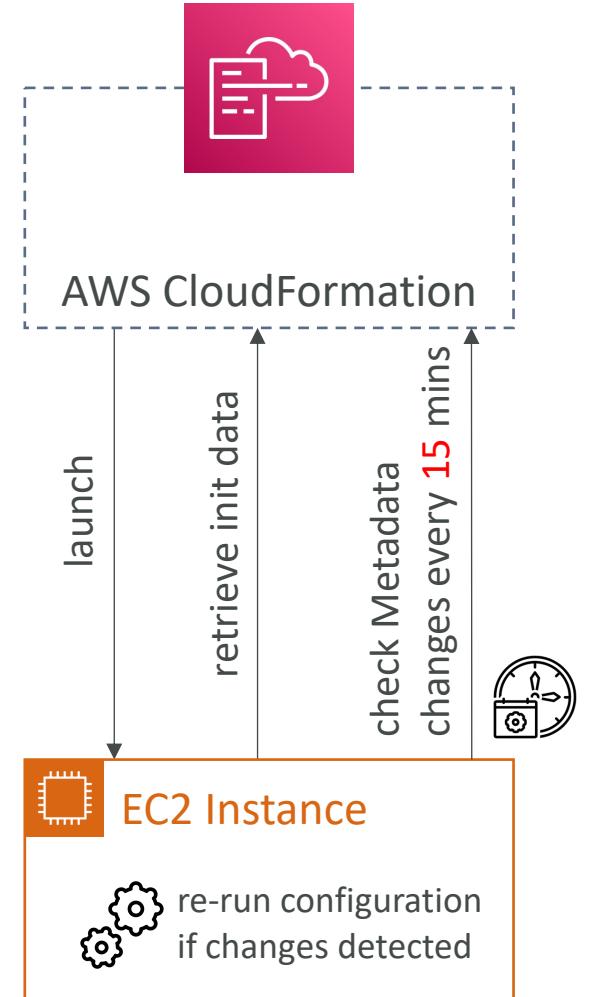
# CloudFormation – ChangeSets

- When you update a stack, you need to know what changes will happen before applying them for greater confidence
- ChangeSets won't say if the update will be successful
- For Nested Stacks, you see the changes across all stacks



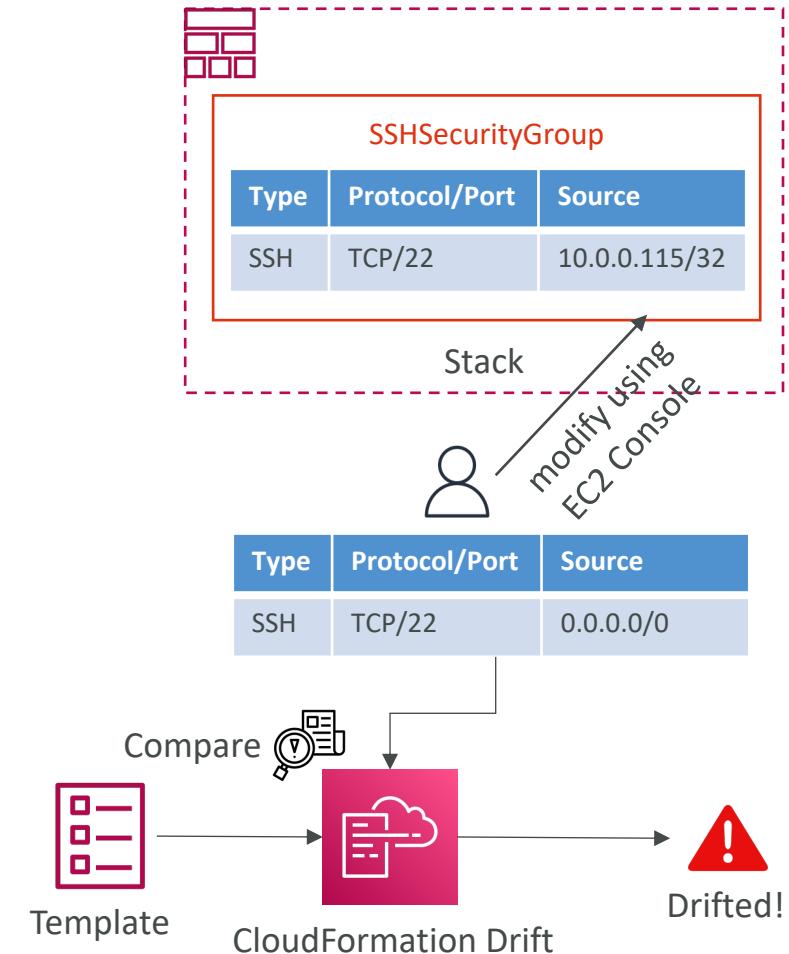
# cfn-hup

- Can be used to tell your EC2 instance to look for Metadata changes every 15 minutes and apply the Metadata configuration again
- It's very powerful but you really need to try it out to understand how it works
- It relies on a `cfn-hup` configuration, see `/etc/cfn/cfn-hup.conf` and `/etc/cfn/hooks.d/cfn-auto-reloader.conf`



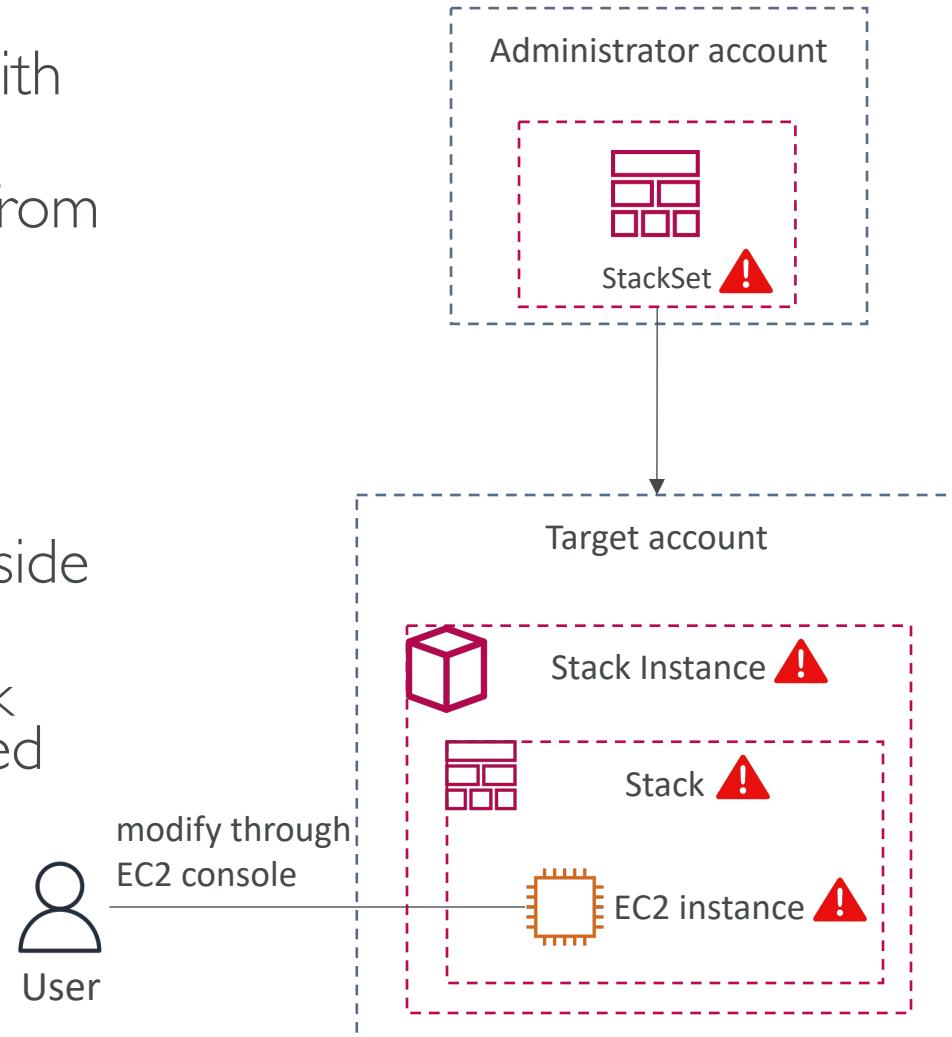
# CloudFormation – Drift

- CloudFormation allows you to create infrastructure
  - But it doesn't protect you against manual configuration changes
  - How do we know if our resources have drifted?
- 
- We can use CloudFormation Drift!
  - Detect drift on an entire stack or on individual resources within a stack



# StackSet Drift Detection

- Performs drift detection on the stack associated with each stack instance in the StackSet
- If the current state of a resource in a stack varies from the expected state:
  - The stack considered drifted
  - And the stack instance that the stack associated with considered drifted
  - And the StackSet is considered drifted
- Drift detection identifies unmanaged changes (outside CloudFormation)
- Changes made through CloudFormation to a stack directly (not at the StackSet level), aren't considered drifted
- You can stop drift detection on a StackSet



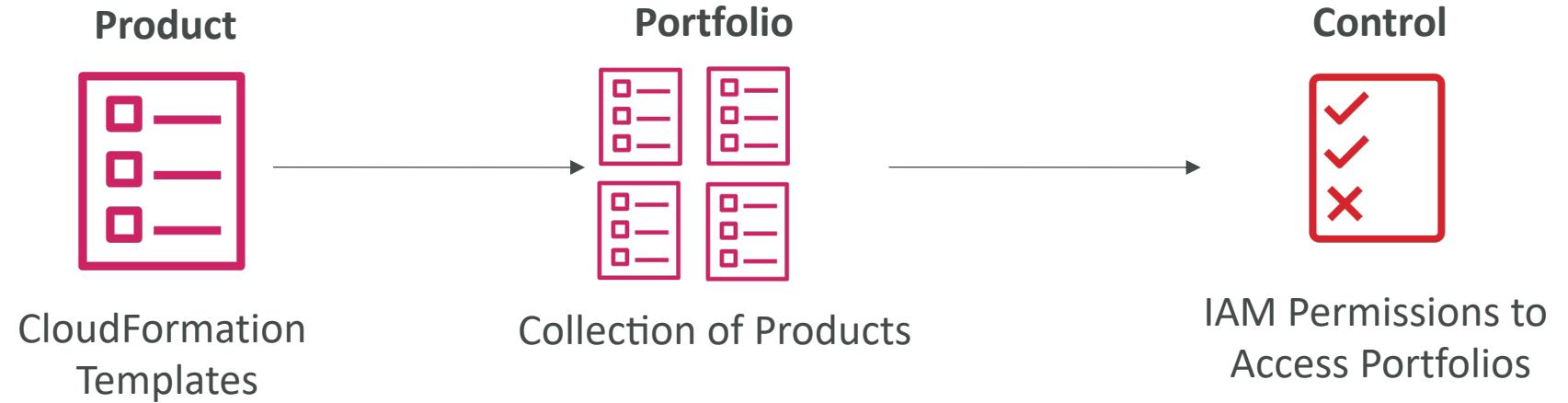
# AWS Service Catalog



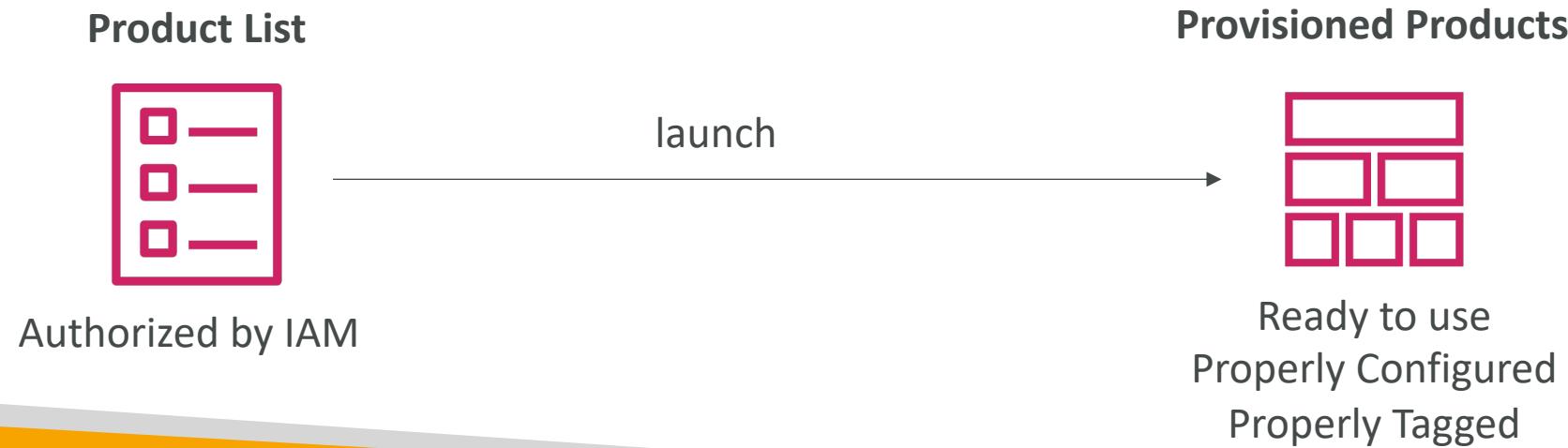
- Users that are new to AWS have too many options, and may create stacks that are not compliant / in line with the rest of the organization
- Some users just want a quick **self-service portal** to launch a set of authorized products pre-defined by admins
- Includes: virtual machines, databases, storage options, etc...
- Enter AWS Service Catalog!

# Service Catalog diagram

## ADMIN TASKS



## USER TASKS



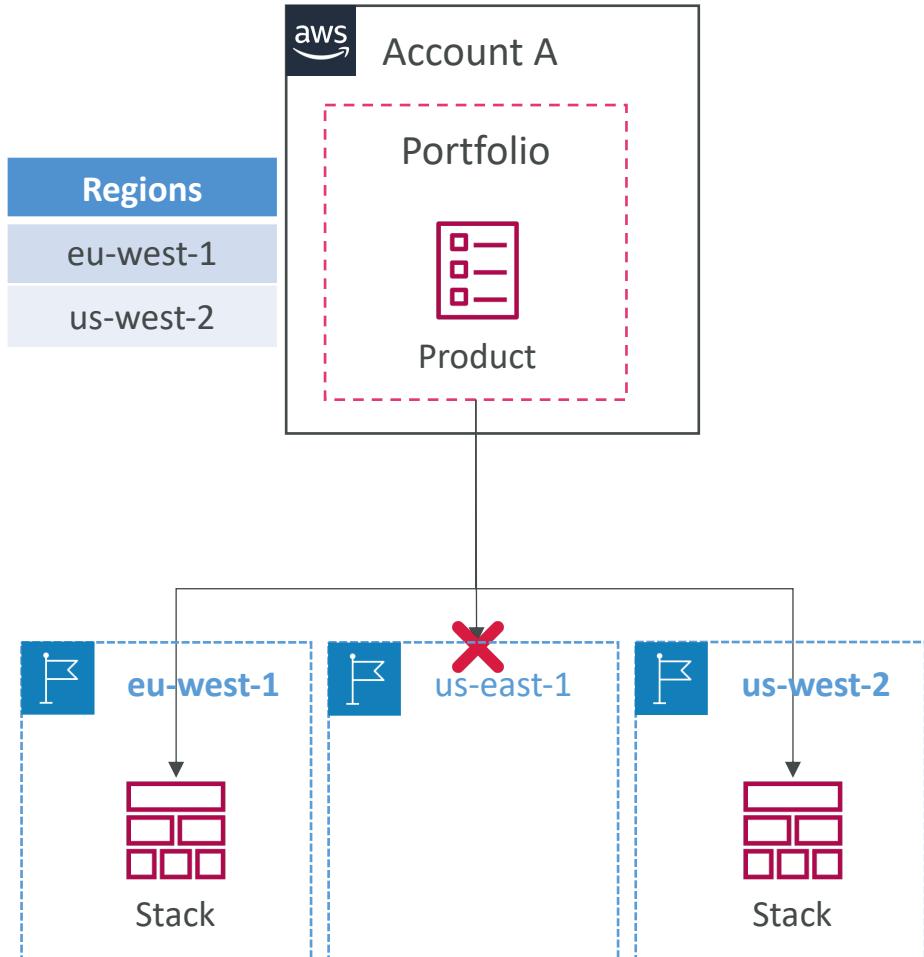
# AWS Service Catalog



- Create and manage catalogs of IT services that are approved on AWS
- The “products” are CloudFormation templates
- Ex: Virtual machine images, Servers, Software, Databases, Regions, IP address ranges
- **CloudFormation helps ensure consistency, and standardization by Admins**
- They are assigned to Portfolios (teams)
- Teams are presented a self-service portal where they can launch the products
- All the deployed products are centrally managed deployed services
- **Helps with governance, compliance, and consistency**
- Can give user access to launching products without requiring deep AWS knowledge
- Integrations with “self-service portals” such as ServiceNow

# Service Catalog – Stack Set Constraints

- Allows you to configure Product deployment options using CloudFormation StackSets
- Accounts – identify AWS accounts where you want to create Products
- Regions – identify AWS regions where you want to deploy to (with Order)
- Permissions – IAM StackSet Administrator Role to manage target AWS accounts

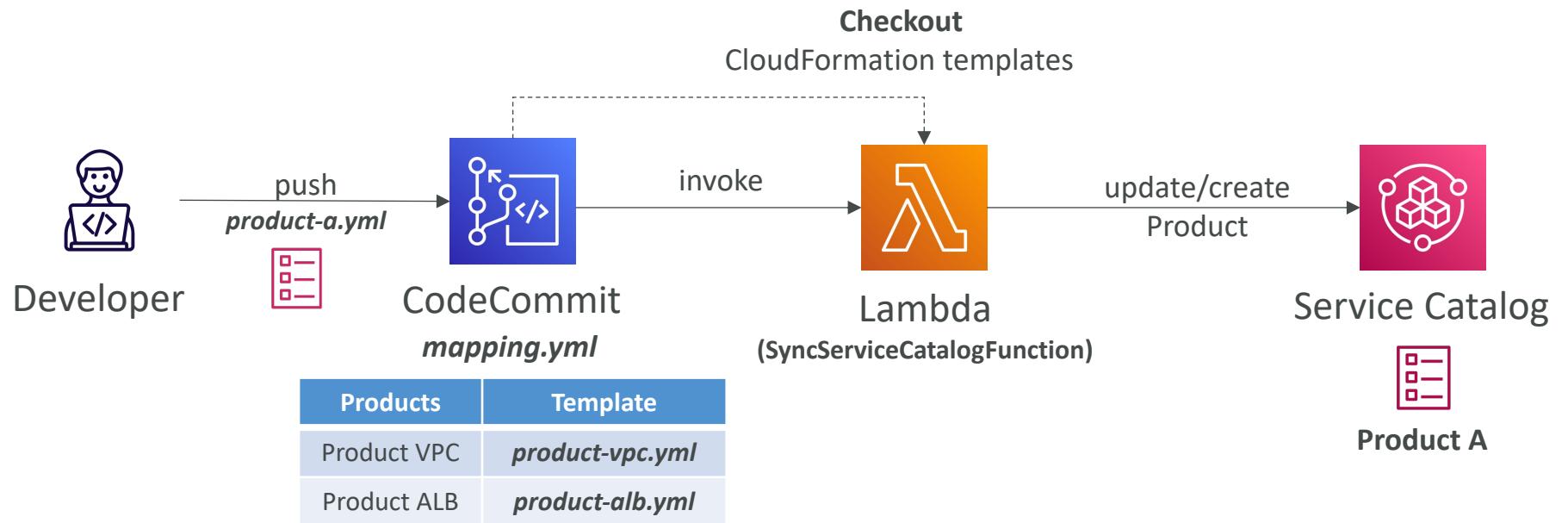


# Service Catalog – Launch Constraints

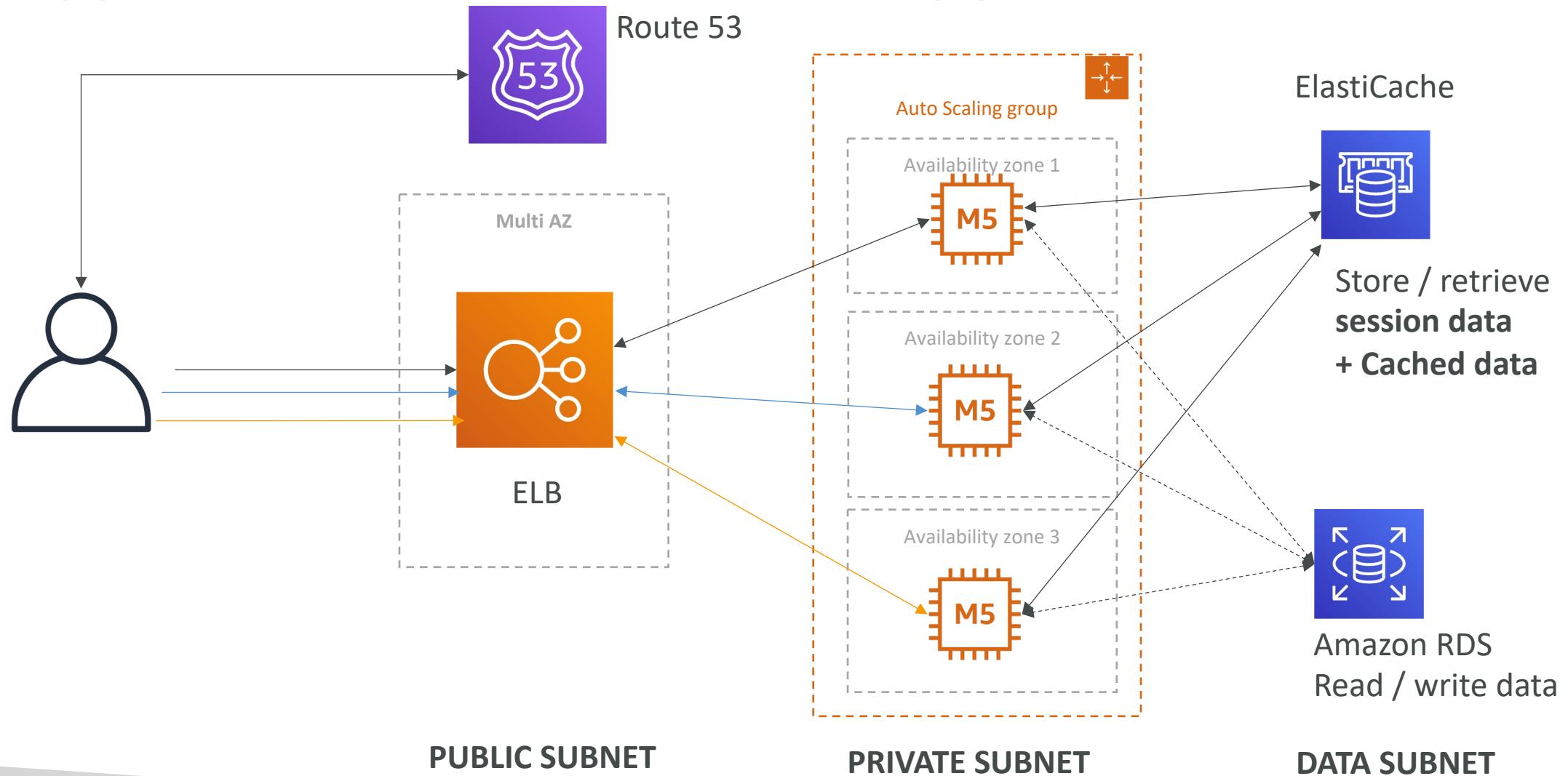
- IAM Role assigned to a Product which allows a user to launch, update, or terminate a product with minimal IAM permissions
- Example: end user has access only to Service Catalog, all other permissions required are attached to the Launch Constraint IAM Role
- IAM Role must have the following permissions:
  - CloudFormation (Full Access)
  - AWS Services in the CloudFormation template
  - S3 Bucket which contains the CloudFormation template (Read Access)

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": [  
        "cloudformation>CreateStack",  
        "cloudformation>DeleteStack",  
        "cloudformation>DescribeStackEvents",  
        "cloudformation>DescribeStacks",  
        "cloudformation>GetTemplateSummary",  
        "cloudformation>SetStackPolicy",  
        "cloudformation>ValidateTemplate",  
        "cloudformation>UpdateStack",  
        "ec2:*",  
        "servicecatalog:*",  
        "sns:*"  
      ],  
      "Resource": "*"  
    },  
    {  
      "Effect": "Allow",  
      "Action": [  
        "s3:GetObject"  
      ],  
      "Resource": "*",  
      "Condition": {  
        "StringEquals": {  
          "s3:ExistingObjectTag/servicecatalog:  
provisioning": "true"  
        }  
      }  
    }  
  ]  
}
```

# Service Catalog – Continuous Delivery Pipeline (Syncing with CodeCommit)



# Typical architecture: Web App 3-tier



# Developer problems on AWS

- Managing infrastructure
  - Deploying Code
  - Configuring all the databases, load balancers, etc
  - Scaling concerns
- 
- Most web apps have the same architecture (ALB + ASG)
  - All the developers want is for their code to run!
  - Possibly, consistently across different applications and environments

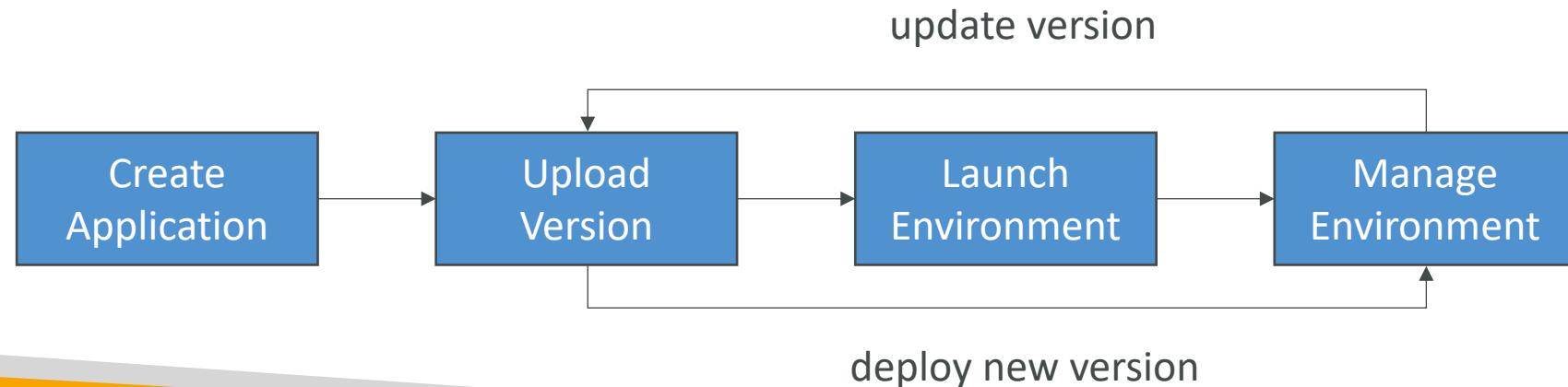
# Elastic Beanstalk – Overview



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

# Elastic Beanstalk – Components

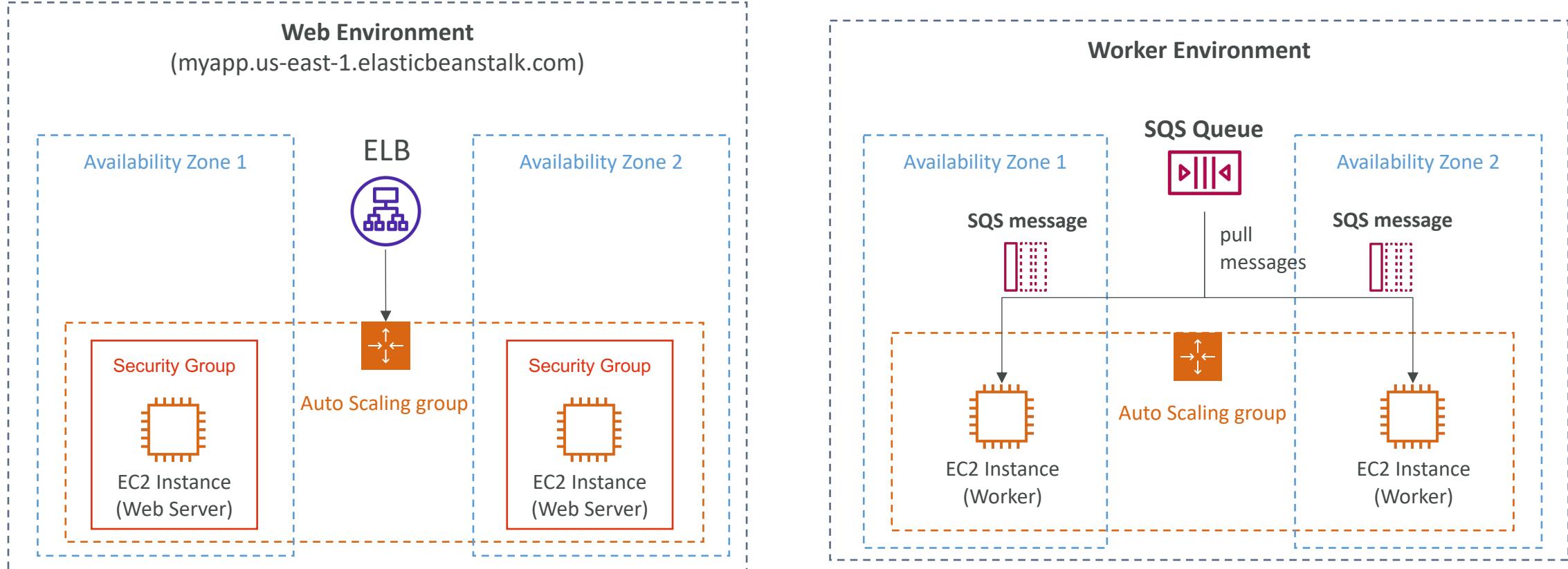
- **Application:** collection of Elastic Beanstalk components (environments, versions, configurations...)
- **Application Version:** an iteration of your application code
- **Environment**
  - Collection of AWS resources running an application version (only one application version at a time)
  - **Tiers:**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



# Elastic Beanstalk – Supported Platforms

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

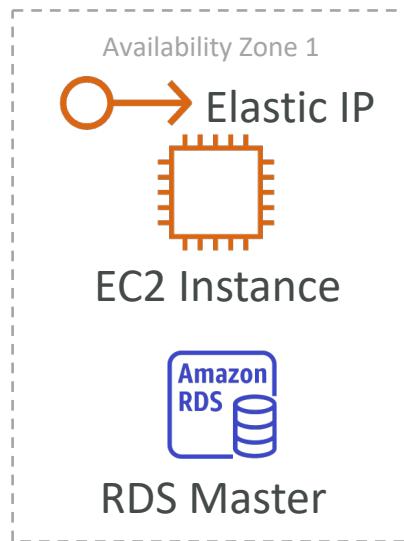
# Web Server Tier vs. Worker Tier



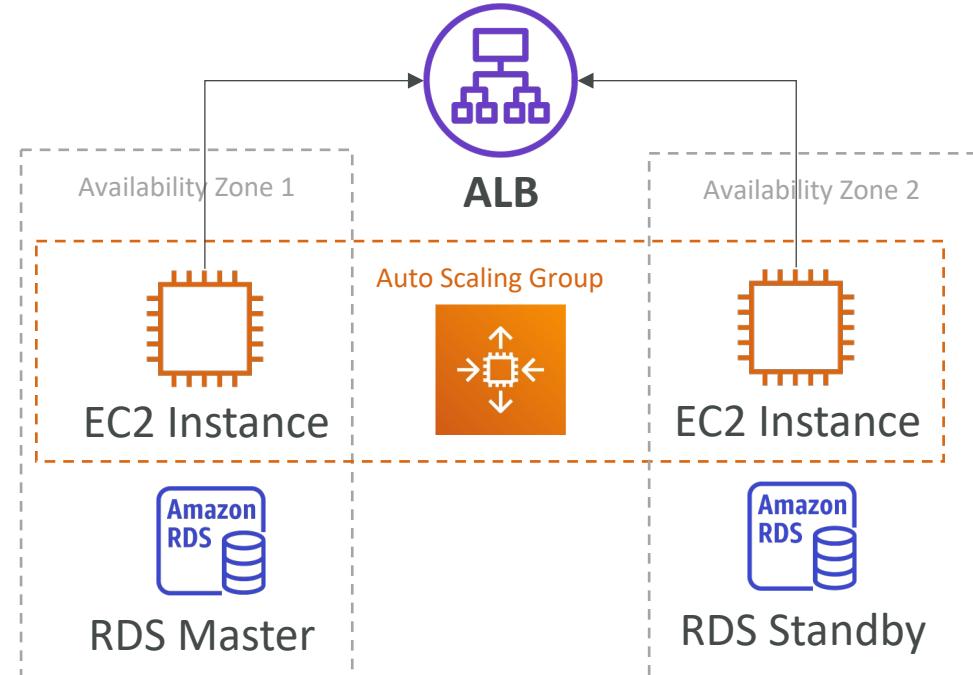
- Scale based on the number of SQS messages
- Can push messages to SQS queue from another Web Server Tier

# Elastic Beanstalk Deployment Modes

**Single Instance**  
Great for dev



**High Availability with Load Balancer**  
Great for prod



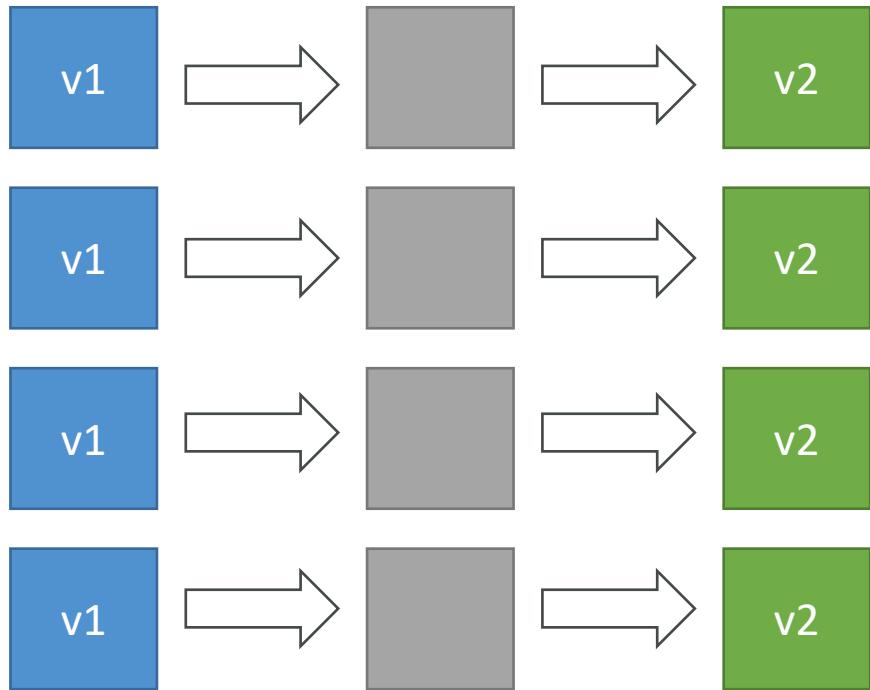
# Beanstalk Deployment Options for Updates

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

# Elastic Beanstalk Deployment

## All at once

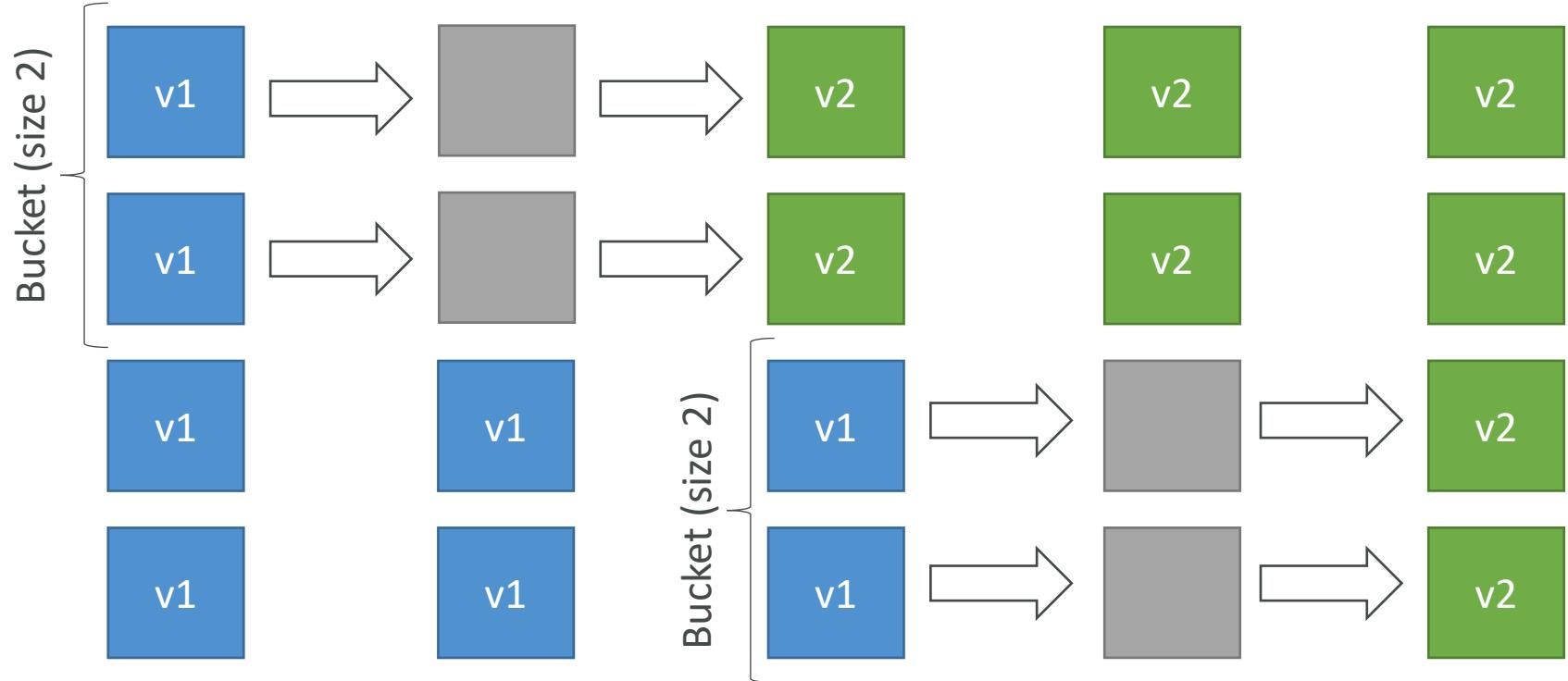
- Fastest deployment
- Application has downtime
- Great for quick iterations in development environment
- No additional cost



# Elastic Beanstalk Deployment

## Rolling

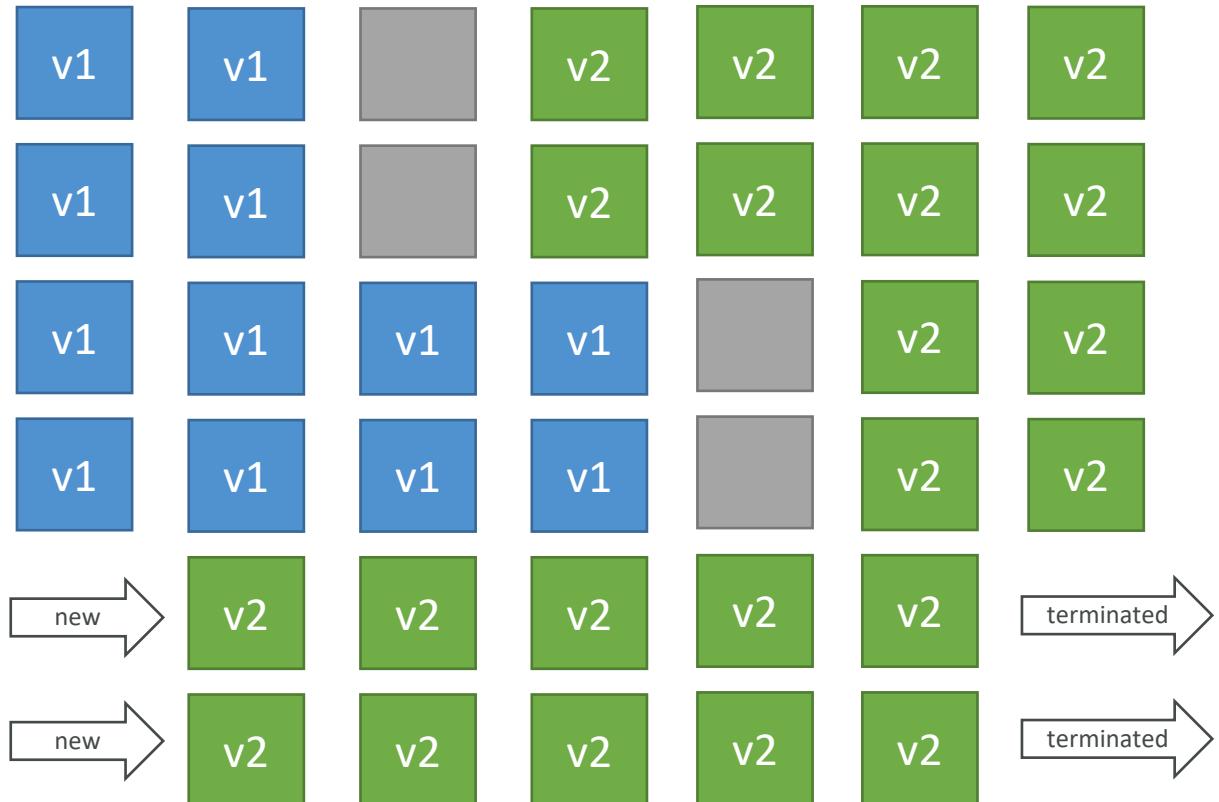
- Application is running below capacity
- Can set the bucket size
- Application is running both versions simultaneously
- No additional cost
- Long deployment



# Elastic Beanstalk Deployment

## Rolling with additional batches

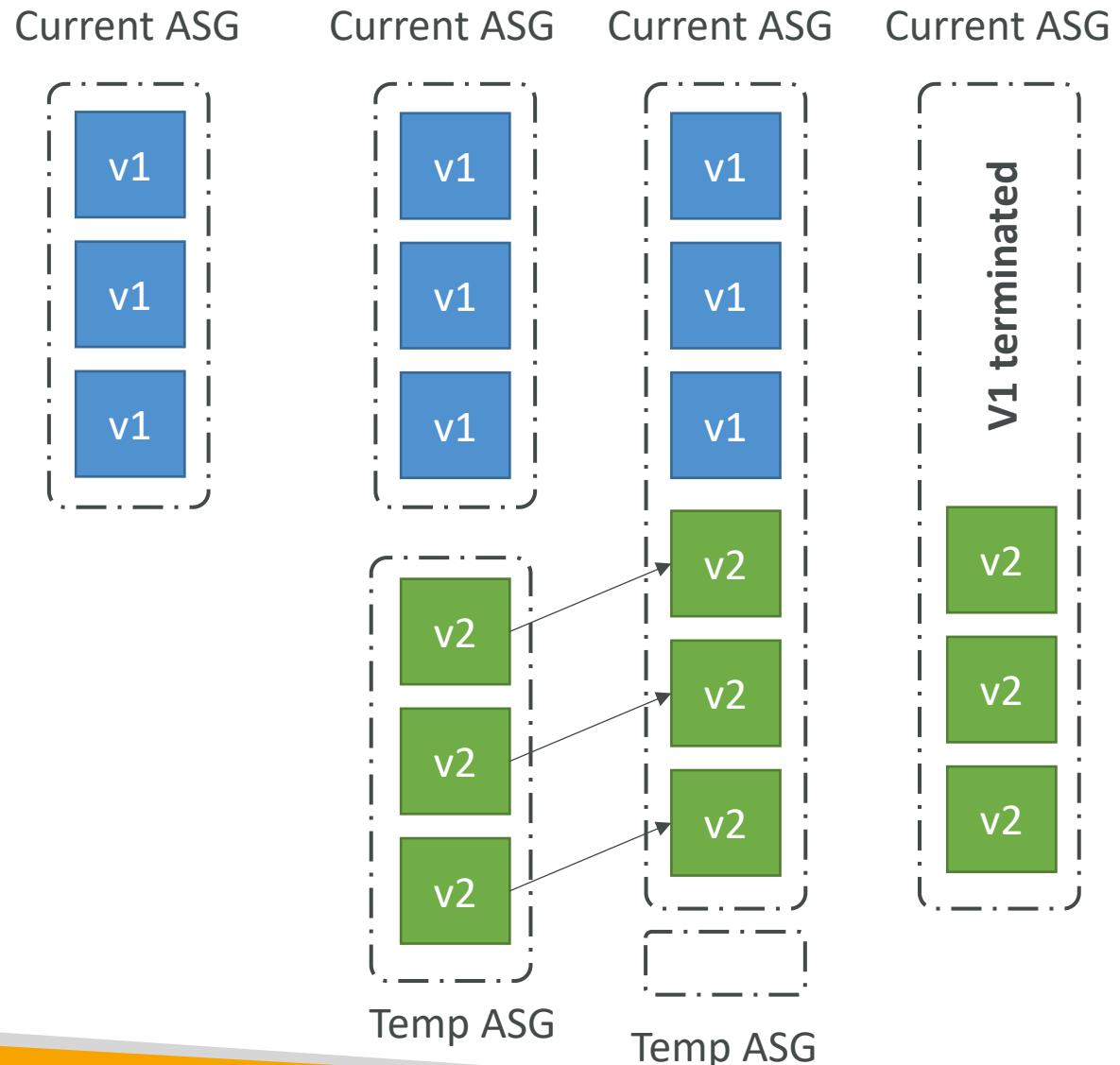
- Application is running at capacity
- Can set the bucket size
- Application is running both versions simultaneously
- Small additional cost
- Additional batch is removed at the end of the deployment
- Longer deployment
- Good for prod



# Elastic Beanstalk Deployment

## Immutable

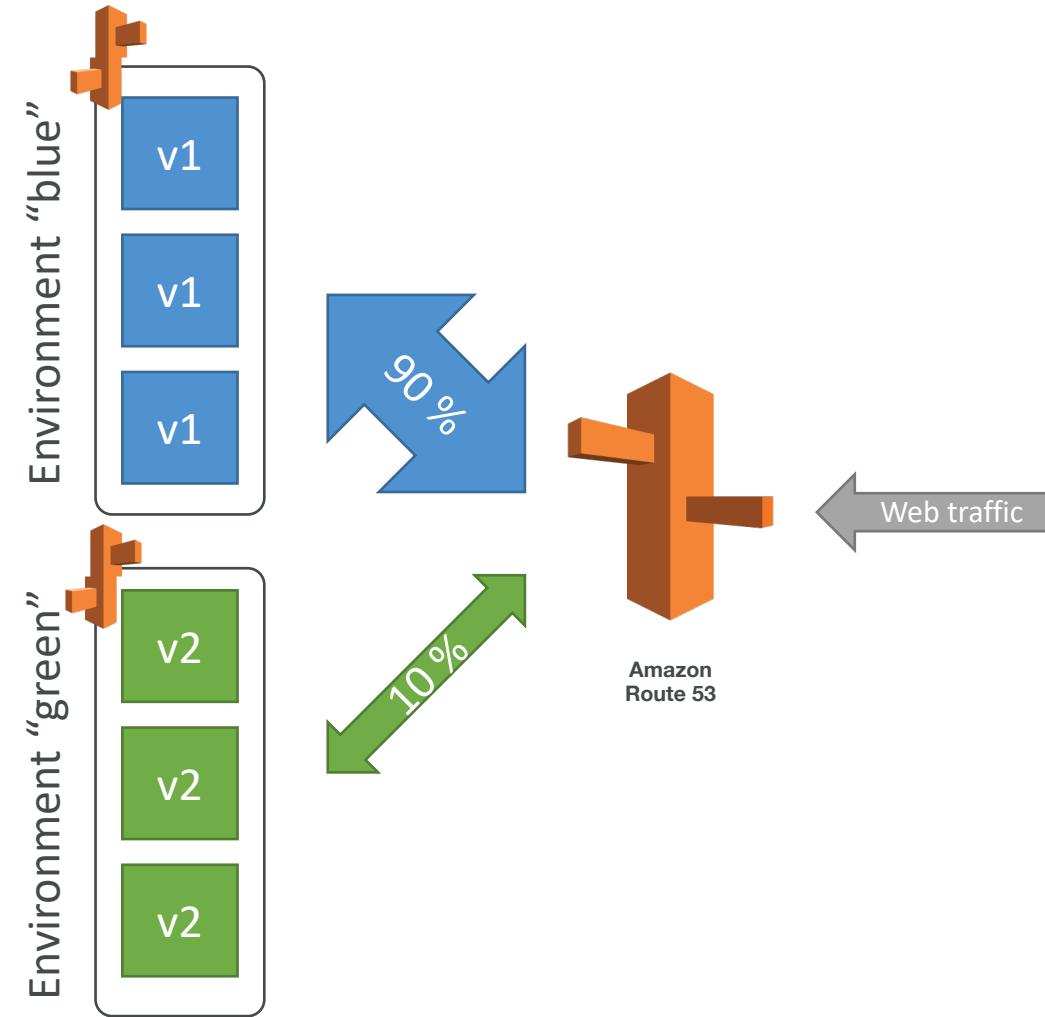
- Zero downtime
- New Code is deployed to new instances on a temporary ASG
- High cost, double capacity
- Longest deployment
- Quick rollback in case of failures  
(just terminate new ASG)
- Great for prod



# Elastic Beanstalk Deployment

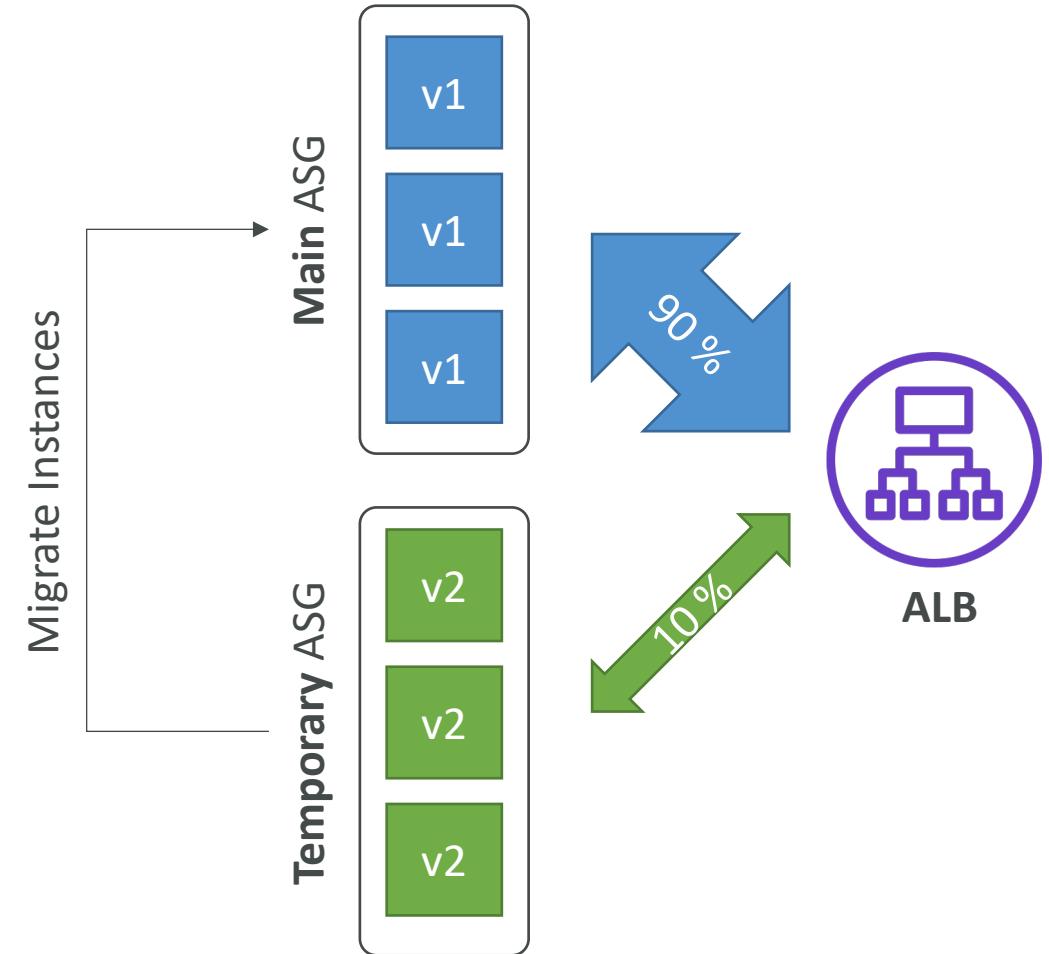
## Blue / Green

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



# Elastic Beanstalk - Traffic Splitting

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



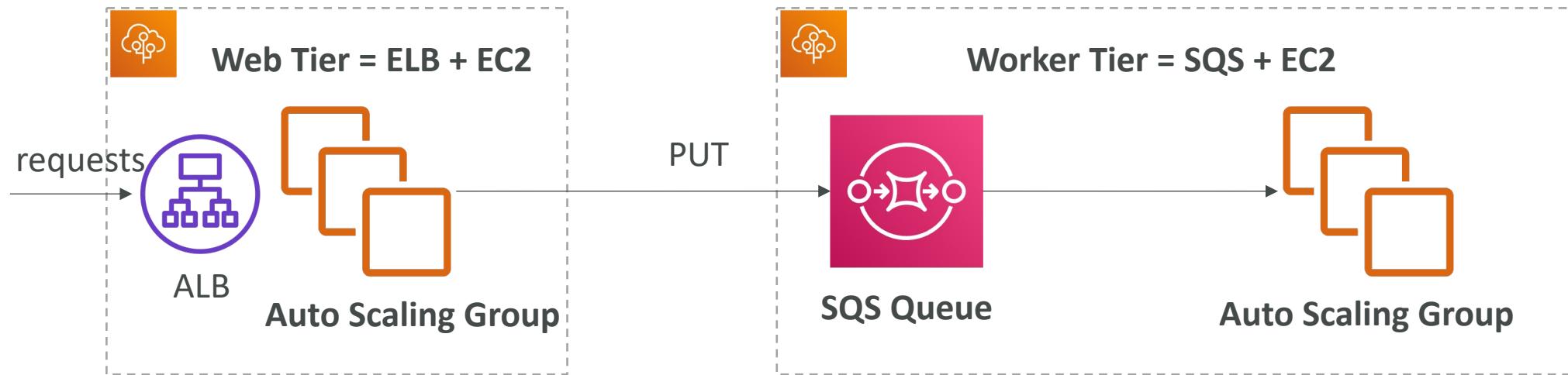
# Elastic Beanstalk Deployment Summary from AWS Doc

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

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

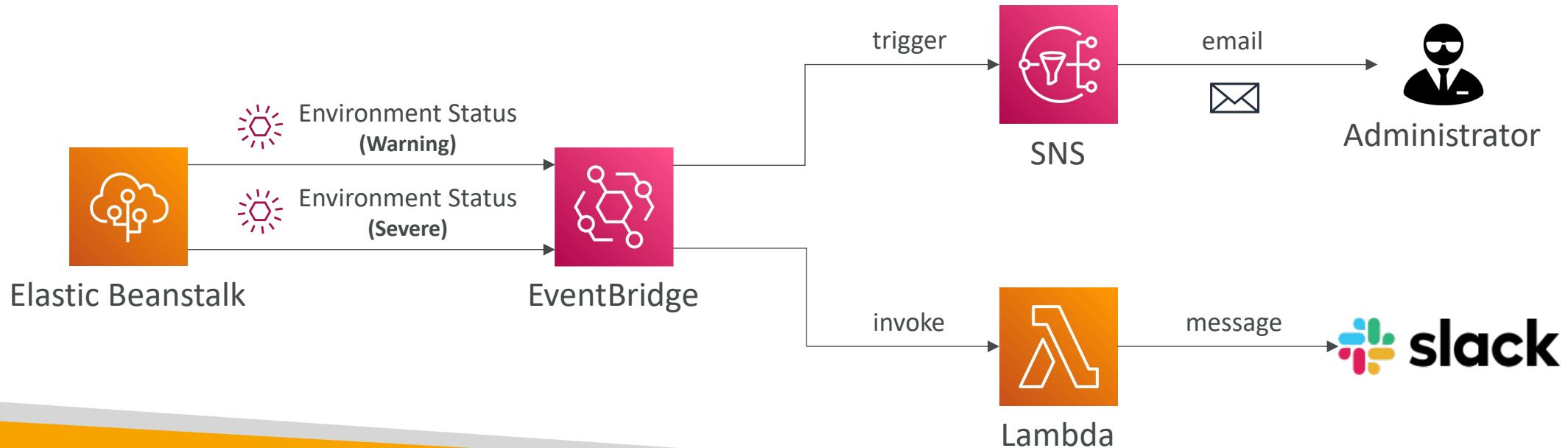
# Web Server vs Worker Environment

- If your application performs tasks that are long to complete, offload these tasks to a dedicated **worker environment**
- Decoupling your application into two tiers is common
- Example: processing a video, generating a zip file, etc
- You can define periodic tasks in a file `cron.yaml`



# Elastic Beanstalk – Notifications

- Create Rules in EventBridge to act to the following events:
  - Environment Operations Status – create, update, terminate (start, success, fail)
  - Other Resources Status – ASG, ELB, EC2 Instance (created, deleted)
  - Managed Updates Status – started, failed
  - Environment Health Status



# AWS SAM

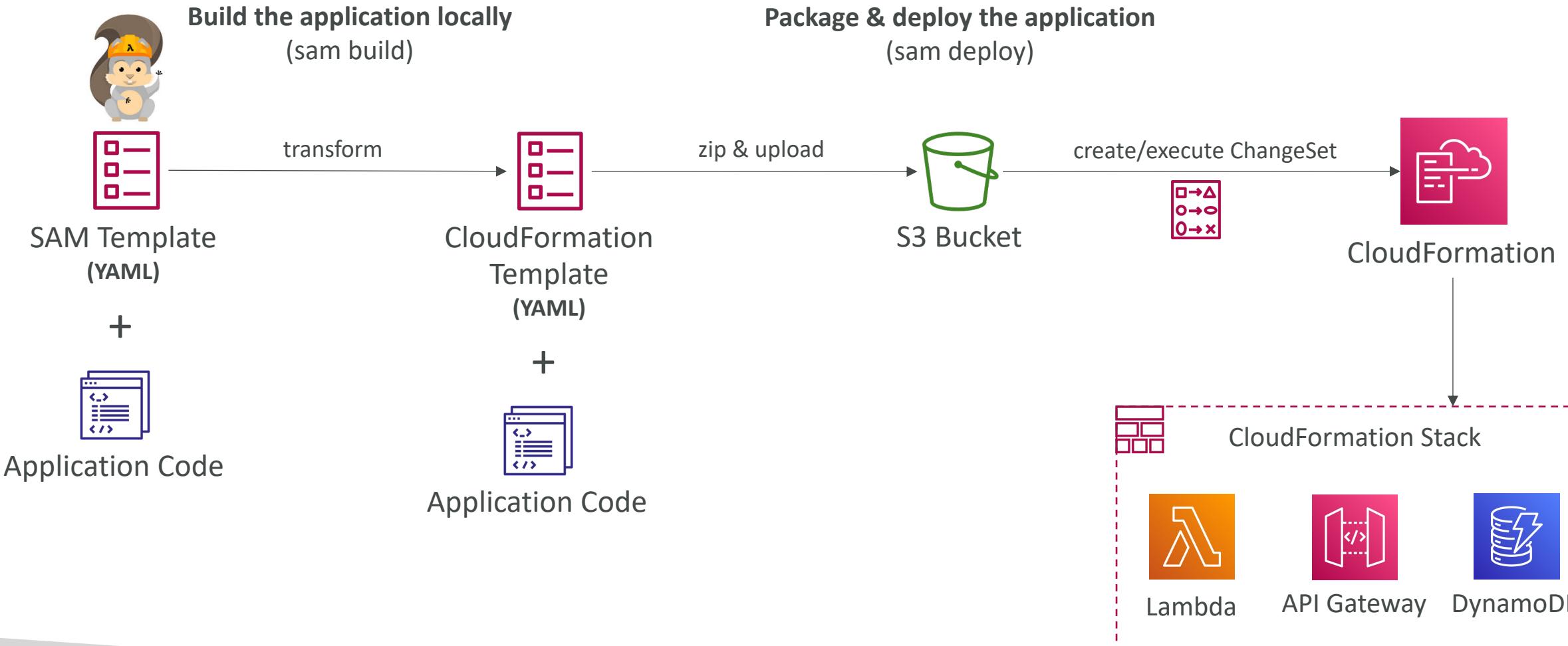


- SAM = Serverless Application Model
- Framework for developing and deploying serverless applications
- All the configuration is YAML code
- Generate complex CloudFormation from simple SAM YAML file
- Supports anything from CloudFormation: Outputs, Mappings, Parameters, Resources...
- SAM can use CodeDeploy to deploy Lambda functions
- SAM can help you to run Lambda, API Gateway, DynamoDB locally

# AWS SAM – Recipe

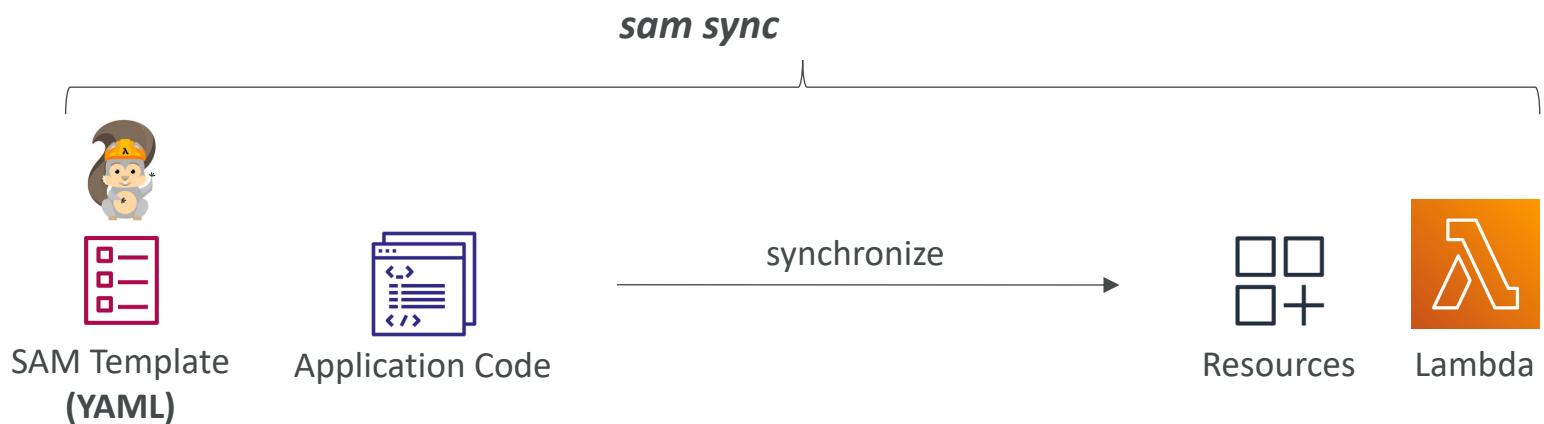
- Transform Header indicates it's SAM template:
  - Transform: 'AWS::Serverless-2016-10-31'
- Write Code
  - AWS::Serverless::Function
  - AWS::Serverless::Api
  - AWS::Serverless::SimpleTable
- Package & Deploy: `sam deploy` (optionally preceded by “`sam package`”)
- Quickly sync local changes to AWS Lambda (SAM Accelerate): `sam sync --watch`

# Deep Dive into SAM Deployment



# SAM Accelerate (sam sync)

- SAM Accelerate is a set of features to reduce latency while deploying resources to AWS
- *sam sync*
  - Synchronizes your project declared in SAM templates to AWS
  - Synchronizes code changes to AWS without updating infrastructure (uses service APIs & **bypass CloudFormation**)

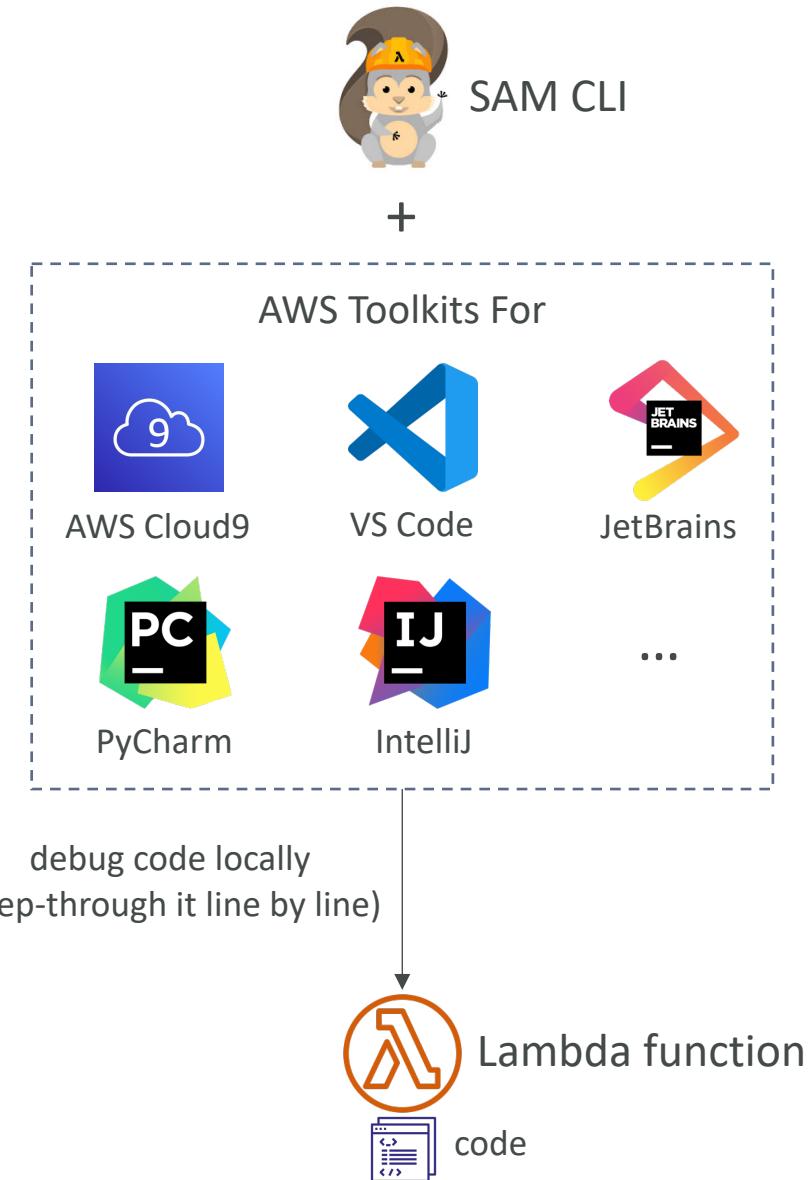


# SAM Accelerate (sam sync) – Examples

- **sam sync (no options)**
  - Synchronize code and infrastructure
- **sam sync --code**
  - Synchronize code changes without updating infrastructure (bypass CloudFormation, update in seconds)
- **sam sync --code --resource AWS::Serverless::Function**
  - Synchronize only all Lambda functions and their dependencies
- **sam sync --code --resource-id HelloWorldLambdaFunction**
  - Synchronize only a specific resource by its ID
- **sam sync --watch**
  - Monitor for file changes and automatically synchronize when changes are detected
  - If changes include configuration, it uses **sam sync**
  - If changes are code only, it uses **sam sync --code**

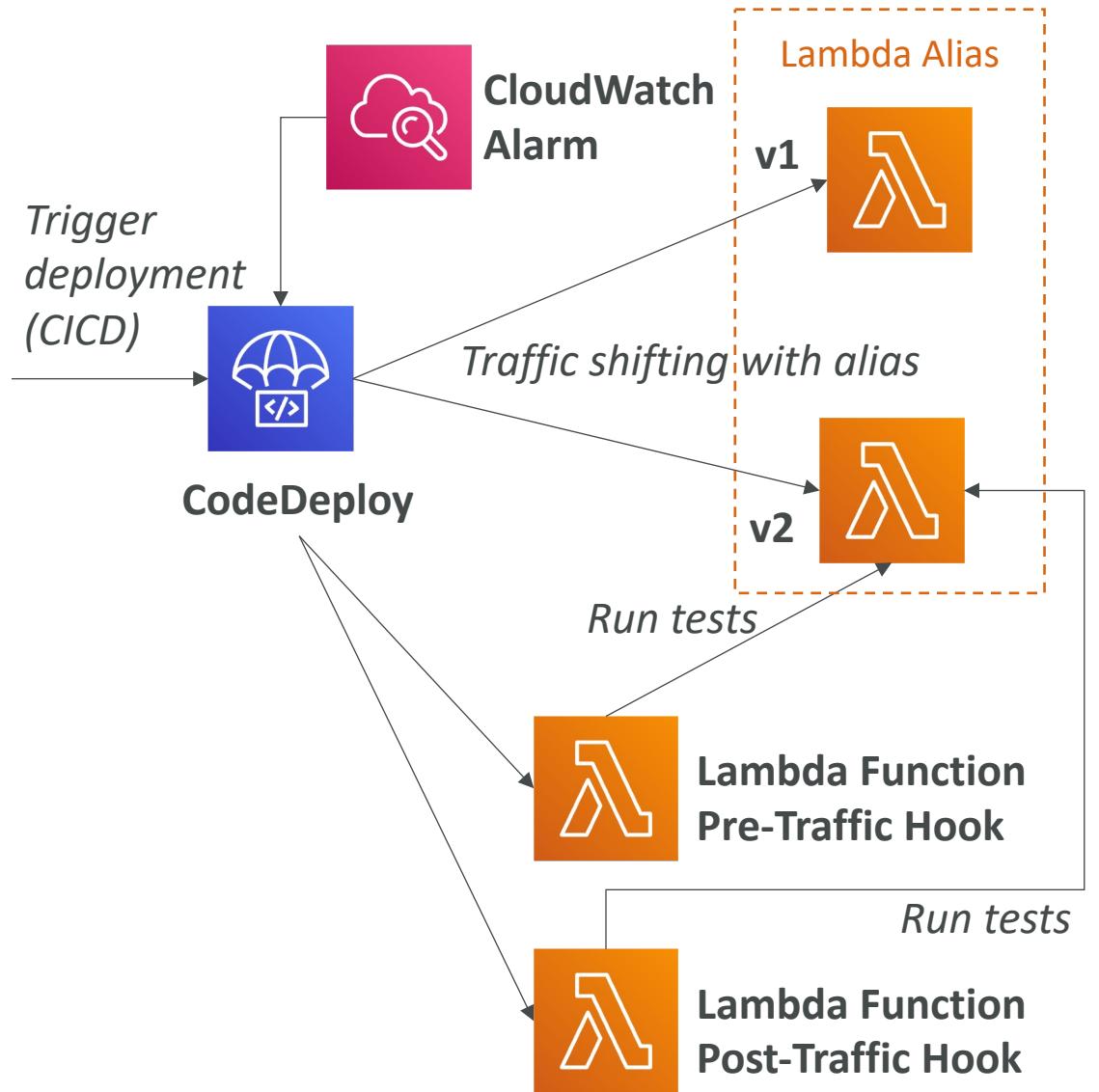
# SAM – CLI Debugging

- Locally build, test, and debug your serverless applications that are defined using AWS SAM templates
- Provides a lambda-like execution environment locally
- SAM CLI + AWS Toolkits => step-through and debug your code
- Supported IDEs: AWS Cloud9, Visual Studio Code, JetBrains, PyCharm, IntelliJ, ...
- **AWS Toolkits:** IDE plugins which allows you to build, test, debug, deploy, and invoke Lambda functions built using AWS SAM



# SAM and CodeDeploy

- SAM framework natively uses CodeDeploy to update Lambda functions
- Traffic Shifting feature
- Pre and Post traffic hooks features to validate deployment (before the traffic shift starts and after it ends)
- Easy & automated rollback using CloudWatch Alarms



# SAM and CodeDeploy

- AutoPublishAlias
  - Detects when new code is being deployed
  - Creates and publishes an updated version of that function with the latest code
  - Points the alias to the updated version of the Lambda function
- DeploymentPreference
  - Canary, Linear, AllAtOnce
- Alarms
  - Alarms that can trigger a rollback
- Hooks
  - Pre and post traffic shifting Lambda functions to test your deployment

## Resources:

### MyLambdaFunction:

Type: AWS::Serverless::Function

#### Properties:

Handler: index.handler

Runtime: nodejs12.x

CodeUri: s3://bucket/code.zip

AutoPublishAlias: live

## DeploymentPreference:

Type: Canary10Percent10Minutes

#### Alarms:

# A list of alarms that you want to monitor

- !Ref AliasErrorMetricGreaterThanZeroAlarm

- !Ref LatestVersionErrorMetricGreaterThanZeroAlarm

#### Hooks:

# Validation Lambda functions that are run before & after traffic shifting

PreTraffic: !Ref PreTrafficLambdaFunction

PostTraffic: !Ref PostTrafficLambdaFunction

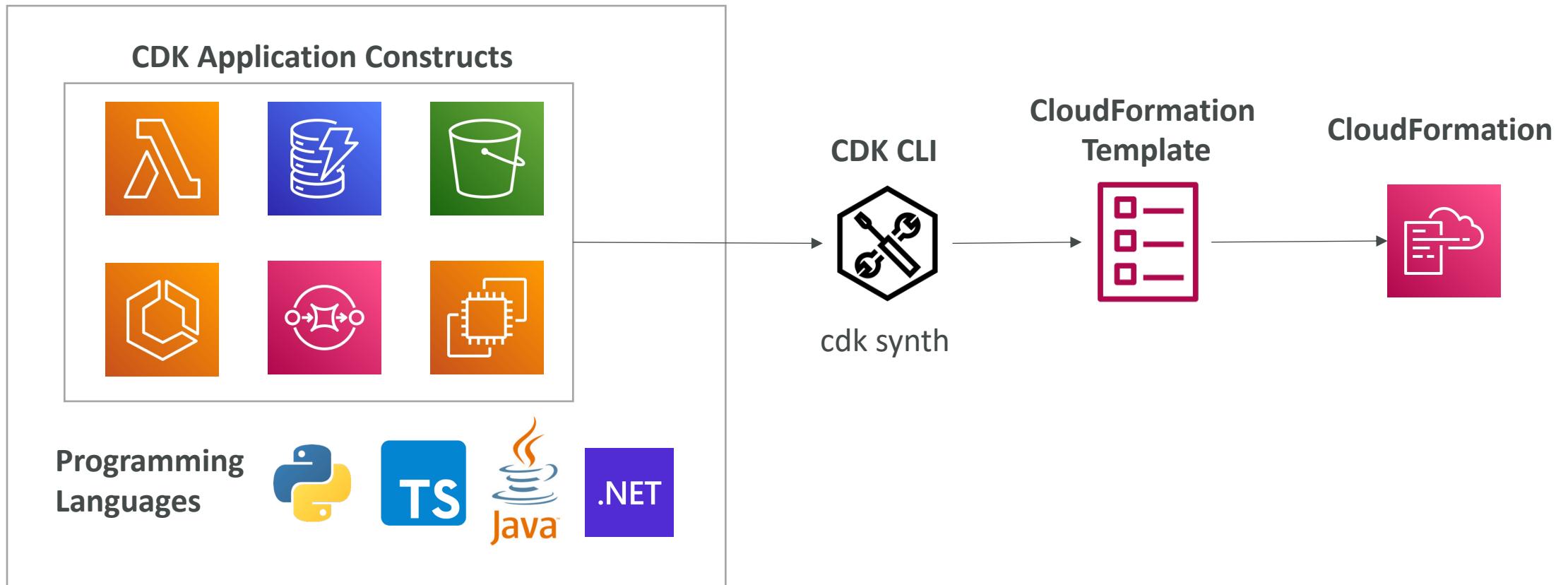
# AWS Cloud Development Kit (CDK)



- Define your cloud infrastructure using a familiar language:
  - JavaScript/TypeScript, Python, Java, and .NET
- Contains high level components called **constructs**
- The code is “compiled” into a CloudFormation template (JSON/YAML)
- You can therefore deploy infrastructure and application runtime code together
  - Great for Lambda functions
  - Great for Docker containers in ECS / EKS

```
export class MyEcsConstructStack extends core.Stack {  
  constructor(scope: core.App, id: string, props?: core.StackProps)  
    super(scope, id, props);  
  
  const vpc = new ec2.Vpc(this, "MyVpc", {  
    maxAzs: 3 // Default is all AZs in region  
  });  
  
  const cluster = new ecs.Cluster(this, "MyCluster", {  
    vpc: vpc  
  });  
  
  // Create a Load-balanced Fargate service and make it public  
  new ecs_patterns.ApplicationLoadBalancedFargateService(this, "My  
    cluster", cluster, // Required  
    cpu: 512, // Default is 256  
    desiredCount: 6, // Default is 1  
    taskImageOptions: { image: ecs.ContainerImage.fromRegistry("am  
      memoryLimitMiB: 2048, // Default is 512  
      publicLoadBalancer: true // Default is false  
    });  
  }  
}
```

# CDK in a diagram



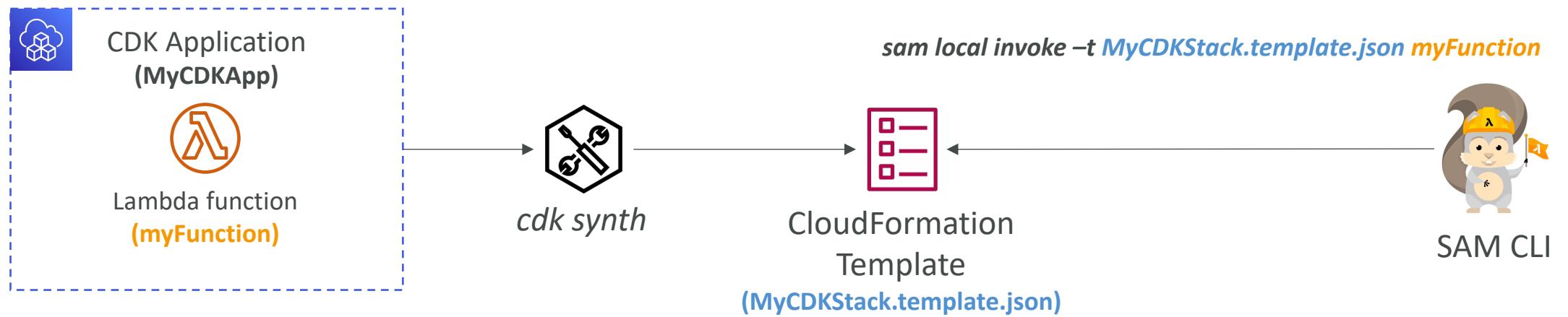
# CDK vs SAM



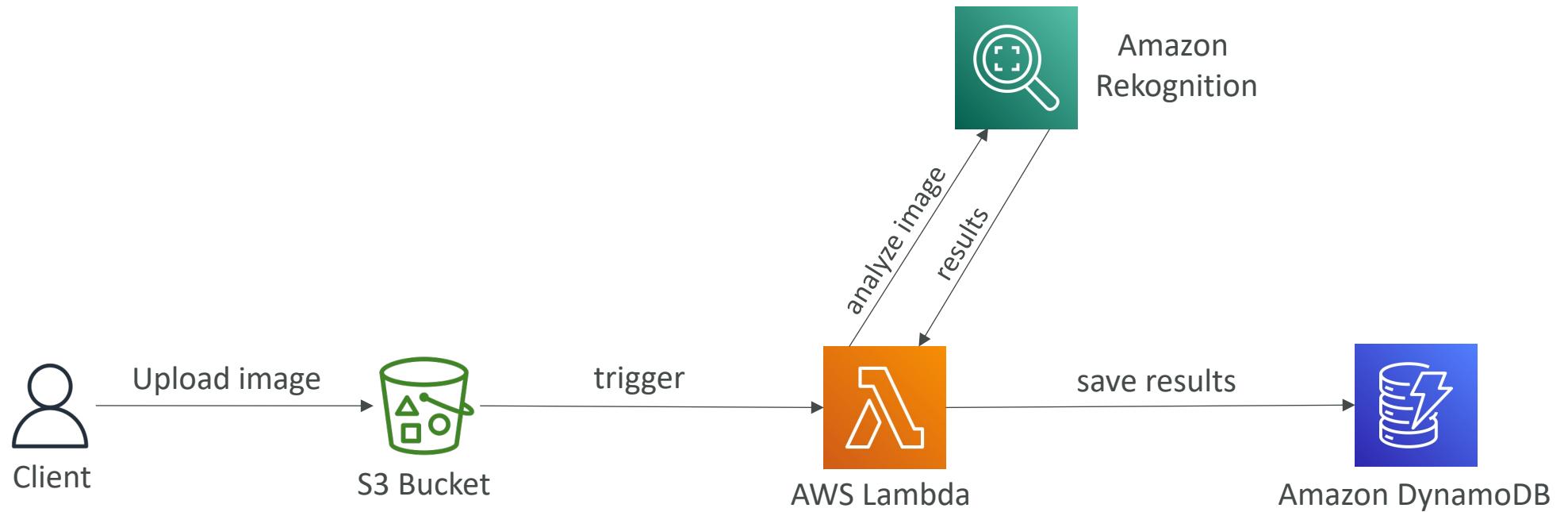
- SAM:
  - Serverless focused
  - Write your template declaratively in JSON or YAML
  - Great for quickly getting started with Lambda
  - Leverages CloudFormation
- CDK:
  - All AWS services
  - Write infra in a programming language JavaScript/TypeScript, Python, Java, and .NET
  - Leverages CloudFormation

# CDK + SAM

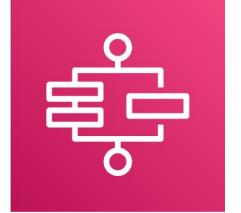
- You can use SAM CLI to locally test your CDK apps
- You must first run `cdk synth`



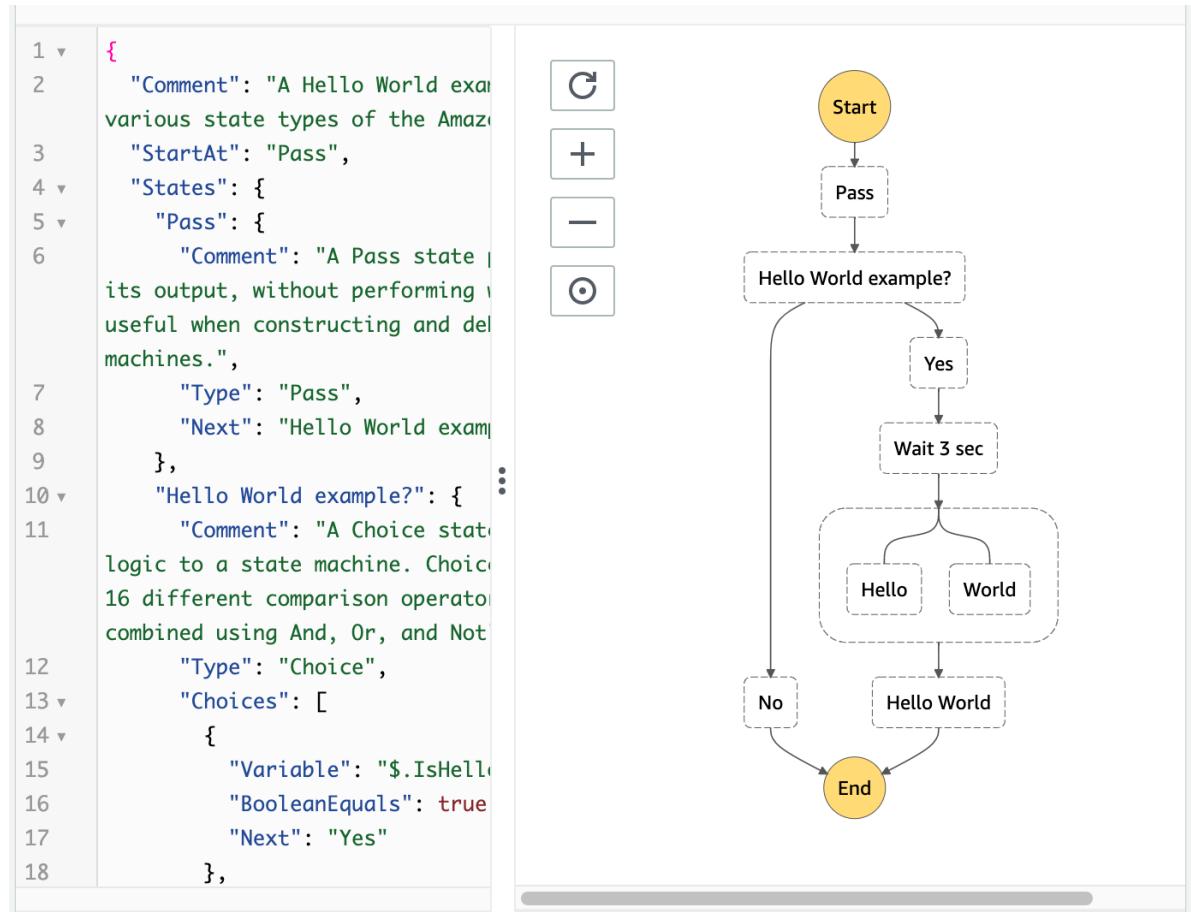
# CDK Hands-On



# AWS Step Functions

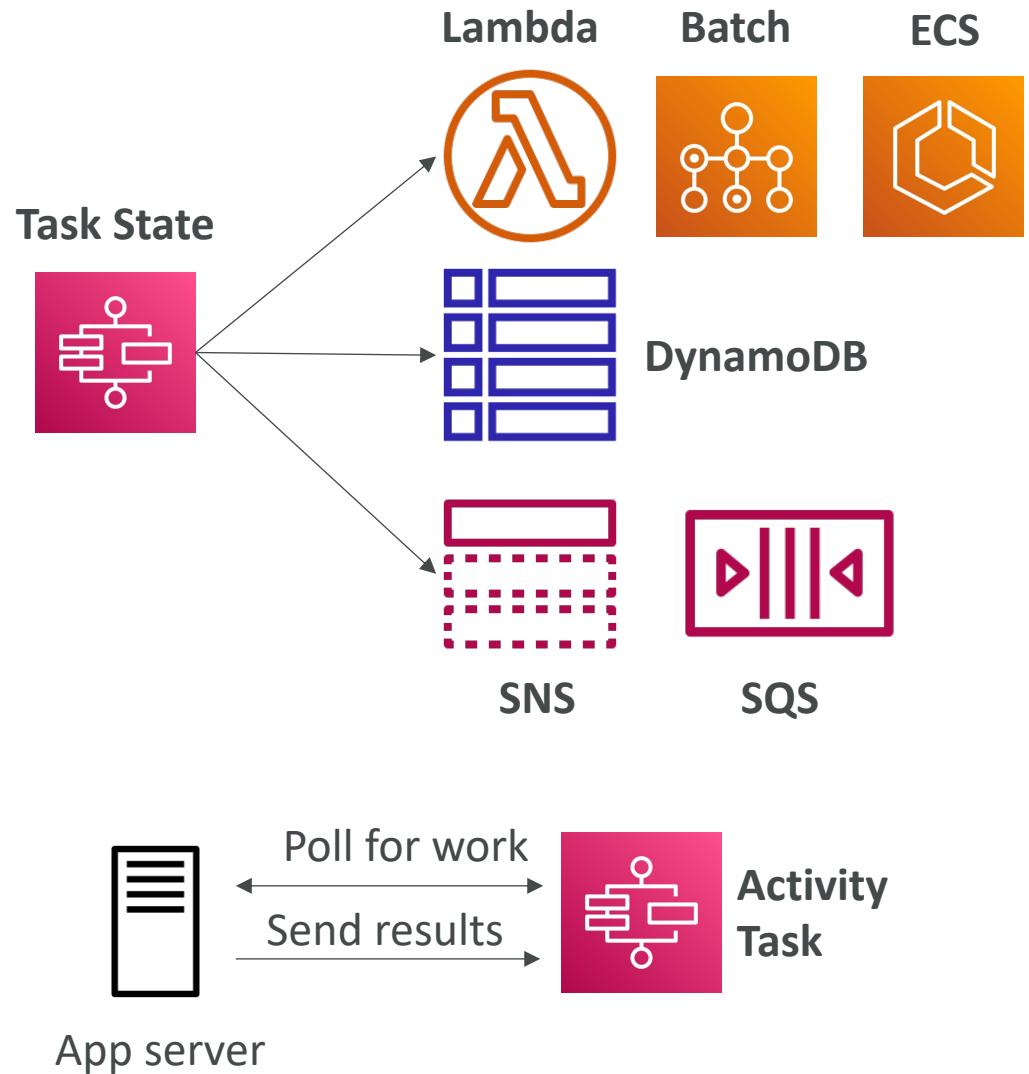


- Model your **workflows** as **state machines** (one per workflow)
  - Order fulfillment, Data processing
  - Web applications, Any workflow
- Written in **JSON**
- Visualization of the workflow and the execution of the workflow, as well as history
- Start workflow with SDK call, API Gateway, Event Bridge (CloudWatch Event)



# Step Function – Task States

- Do some work in your state machine
- Invoke one AWS service
  - Can invoke a Lambda function
  - Run an AWS Batch job
  - Run an ECS task and wait for it to complete
  - Insert an item from DynamoDB
  - Publish message to SNS, SQS
  - Launch another Step Function workflow...
- Run an one Activity
  - EC2, Amazon ECS, on-premises
  - Activities poll the Step functions for work
  - Activities send results back to Step Functions



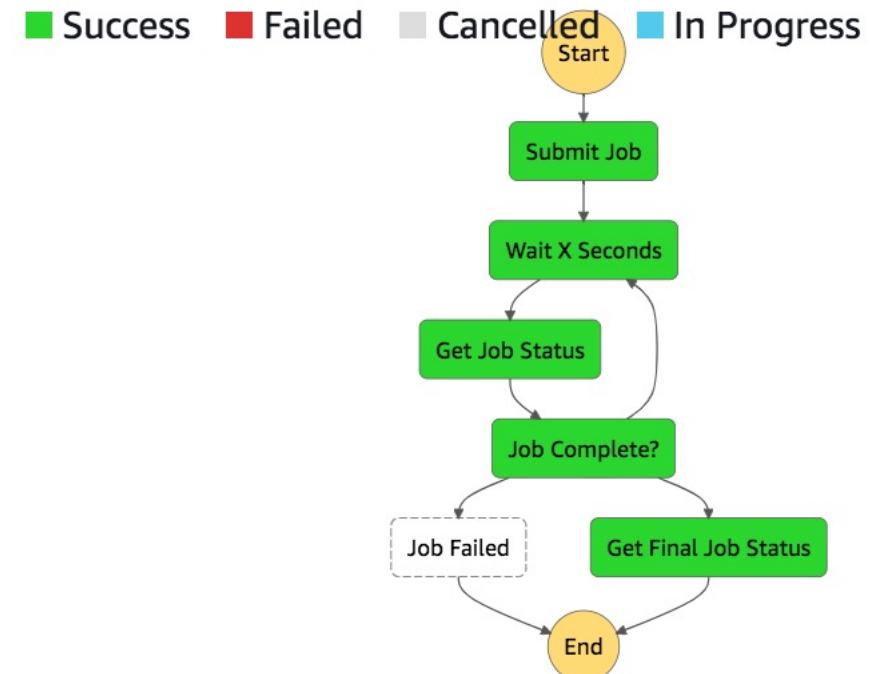
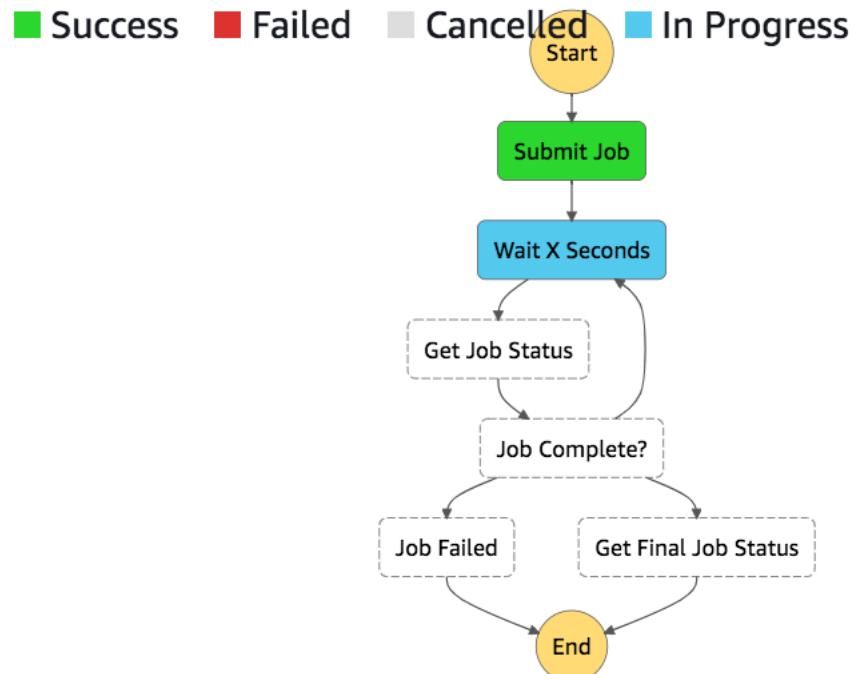
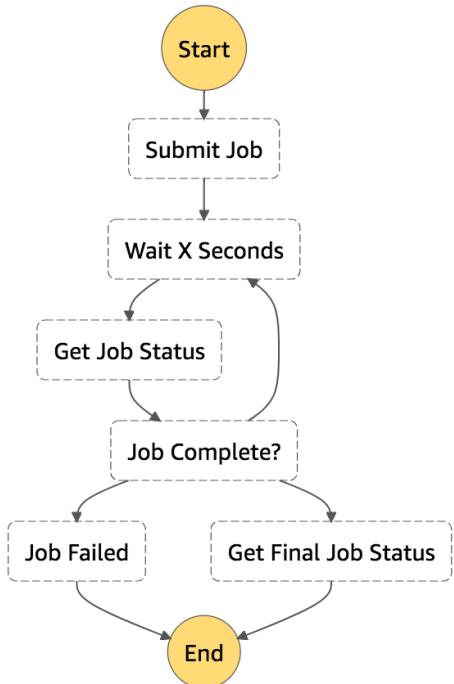
# Example – Invoke Lambda Function

```
"Invoke Lambda function": {  
    "Type": "Task",  
    "Resource": "arn:aws:states:::lambda:invoke",  
    "Parameters": {  
        "FunctionName":  
            "arn:aws:lambda:REGION:ACCOUNT_ID:function:FUNCTION_NAME",  
        "Payload": {  
            "Input.$": "$"  
        }  
    },  
    "Next": "NEXT_STATE",  
    "TimeoutSeconds": 300  
}
```

# Step Function - States

- **Choice State** - Test for a condition to send to a branch (or default branch)
- **Fail or Succeed State** - Stop execution with failure or success
- **Pass State** - Simply pass its input to its output or inject some fixed data, without performing work.
- **Wait State** - Provide a delay for a certain amount of time or until a specified time/date.
- **Map State** - Dynamically iterate steps.'
- **Parallel State** - *Begin parallel branches of execution.*

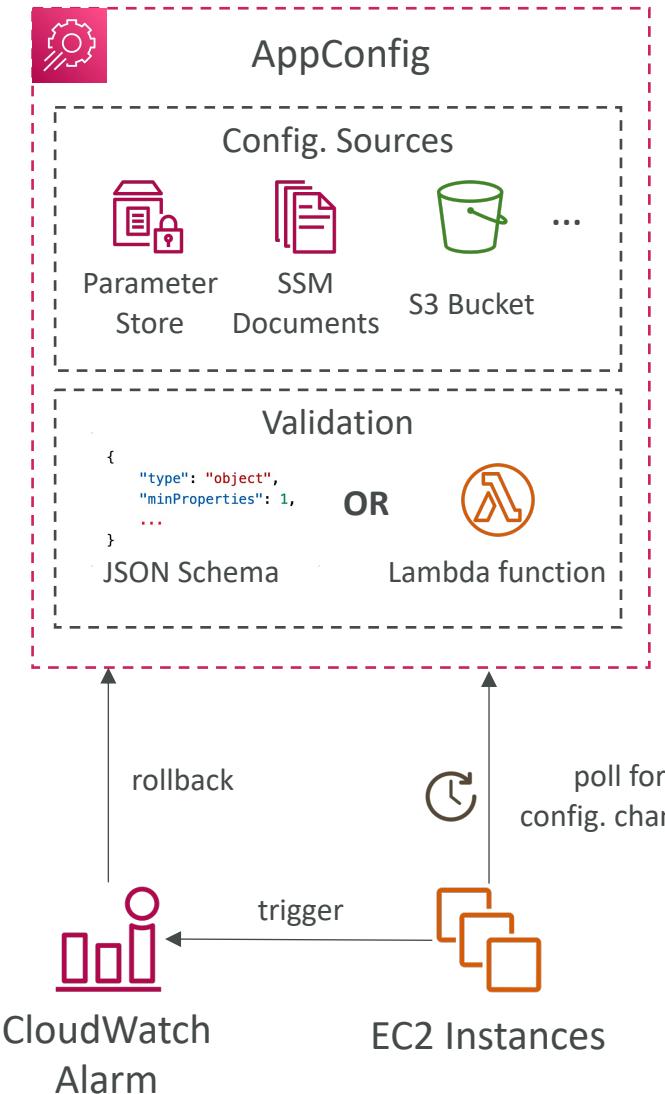
# Visual workflow in Step Functions



# AWS AppConfig



- Configure, validate, and deploy dynamic configurations
- Deploy dynamic configuration changes to your applications independently of any code deployments
  - You don't need to restart the application
- Feature flags, application tuning, allow/block listing...
- Use with apps on EC2 instances, Lambda, ECS, EKS...
- Gradually deploy the configuration changes and rollback if issues occur
- Validate configuration changes before deployment using:
  - **JSON Schema** (syntactic check) or
  - **Lambda Function** – run code to perform validation (semantic check)



# AWS Systems Manager Overview



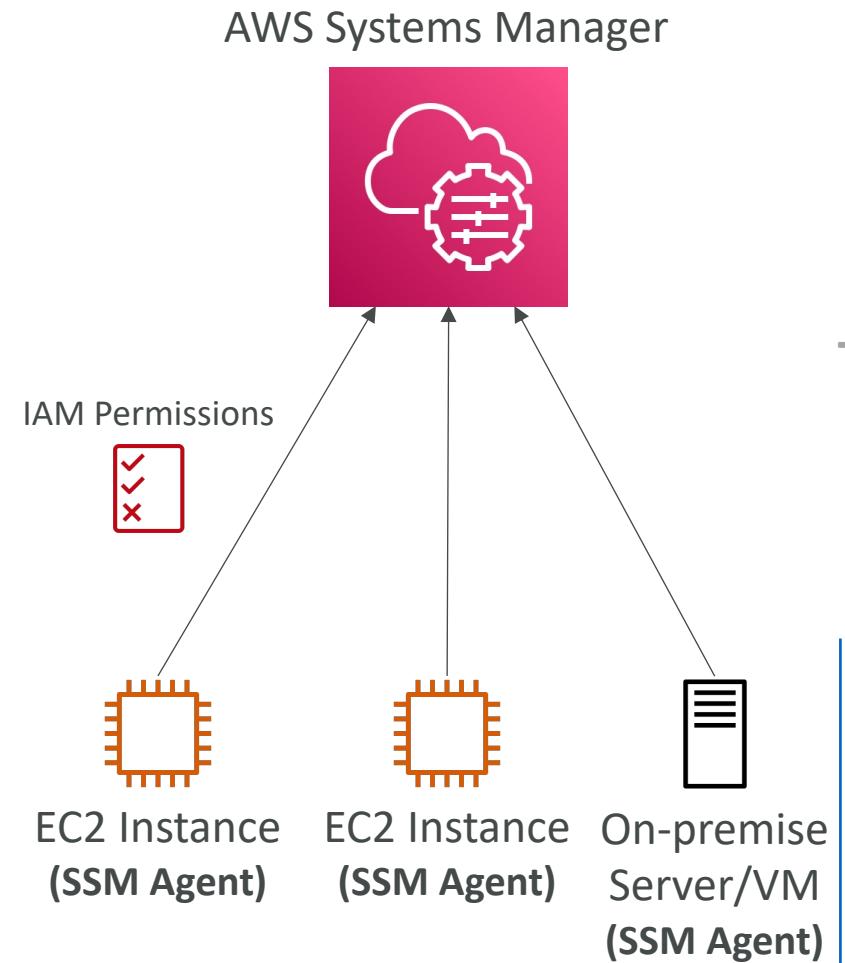
- Helps you manage your EC2 and On-Premises systems at scale
- Get operational insights about the state of your infrastructure
- Easily detect problems
- Patching automation for enhanced compliance
- Works for both Windows and Linux OS
- Integrated with CloudWatch metrics / dashboards
- Integrated with AWS Config
- Free service

# AWS Systems Manager Features

- Resource Groups
- Operations Management
  - Explorer
  - OpsCenter
  - CloudWatch Dashboard
  - PHD
  - Incident Manager
- Shared Resources
  - Documents
- Change Management
  - Change Manager
  - Automation
  - Change Calendar
  - Maintenance Windows
- Application Management
  - Application Manager
  - AppConfig
  - Parameter Store
- Node Management
  - Fleet Manager
  - Compliance
  - Inventory
  - Hybrid Activations
  - Session Manager
  - Run Command
  - State Manager
  - Patch Manager
  - Distributer

# How Systems Manager works

- We need to install the SSM agent onto the systems we control
- Installed by default on Amazon Linux 2 AMI & some Ubuntu AMI
- If an instance can't be controlled with SSM, it's probably an issue with the SSM agent!
- Make sure the EC2 instances have a proper IAM role to allow SSM actions



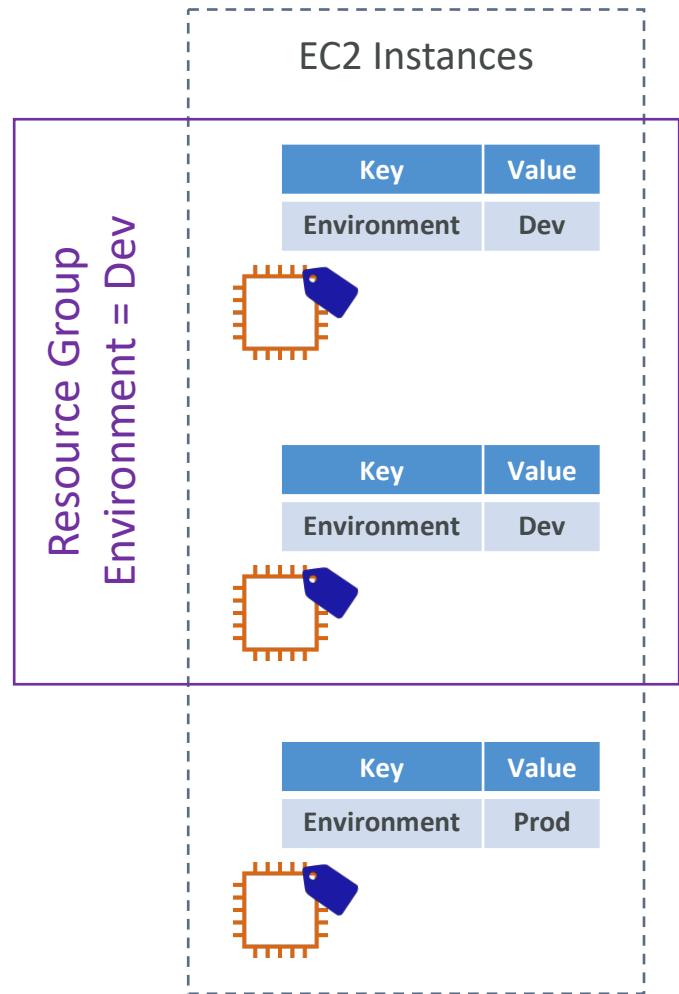


# AWS Tags

- You can add text key-value pairs called Tags to many AWS resources
- Commonly used in EC2
- Free naming, common tags are Name, Environment, Team ...
- They're used for
  - Resource grouping
  - Automation
  - Cost allocation
- Better to have too many tags than too few!

# Resource Groups

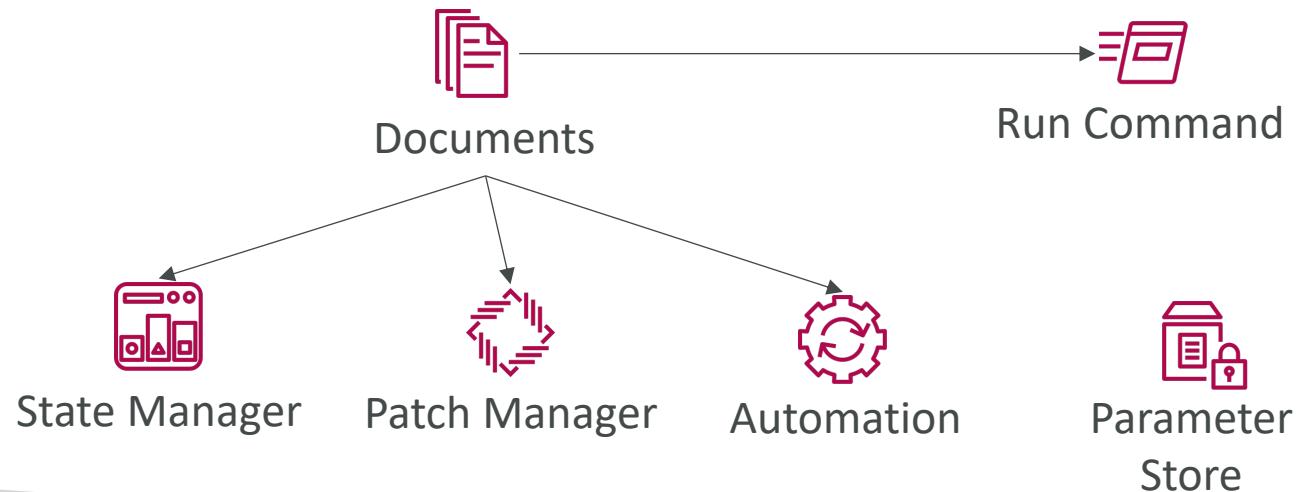
- Create, view or manage logical group of resources thanks to **tags**
- Allows creation of logical groups of resources such as
  - Applications
  - Different layers of an application stack
  - Production versus development environments
- Regional service
- Works with EC2, S3, DynamoDB, Lambda, etc...



# SSM – Documents

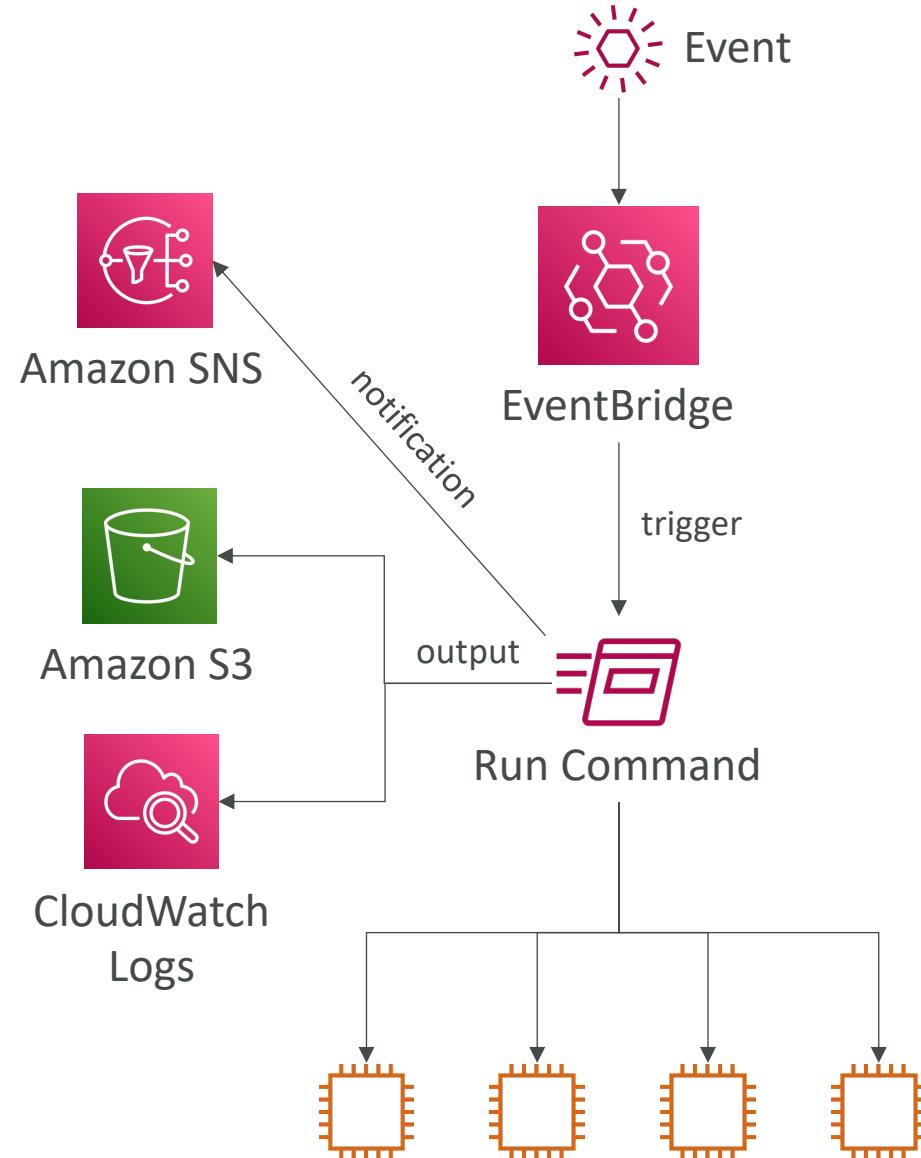
- Documents can be in JSON or YAML
- You define parameters
- You define actions
- Many documents already exist in AWS

```
---  
schemaVersion: '2.2'  
description: State Manager Bootstrap Example  
parameters: {}  
mainSteps:  
  - action: aws:runShellScript  
    name: configureServer  
    inputs:  
      runCommand:  
        - sudo yum install -y httpd  
        - sudo yum --enablerepo=epel install -y clamav
```



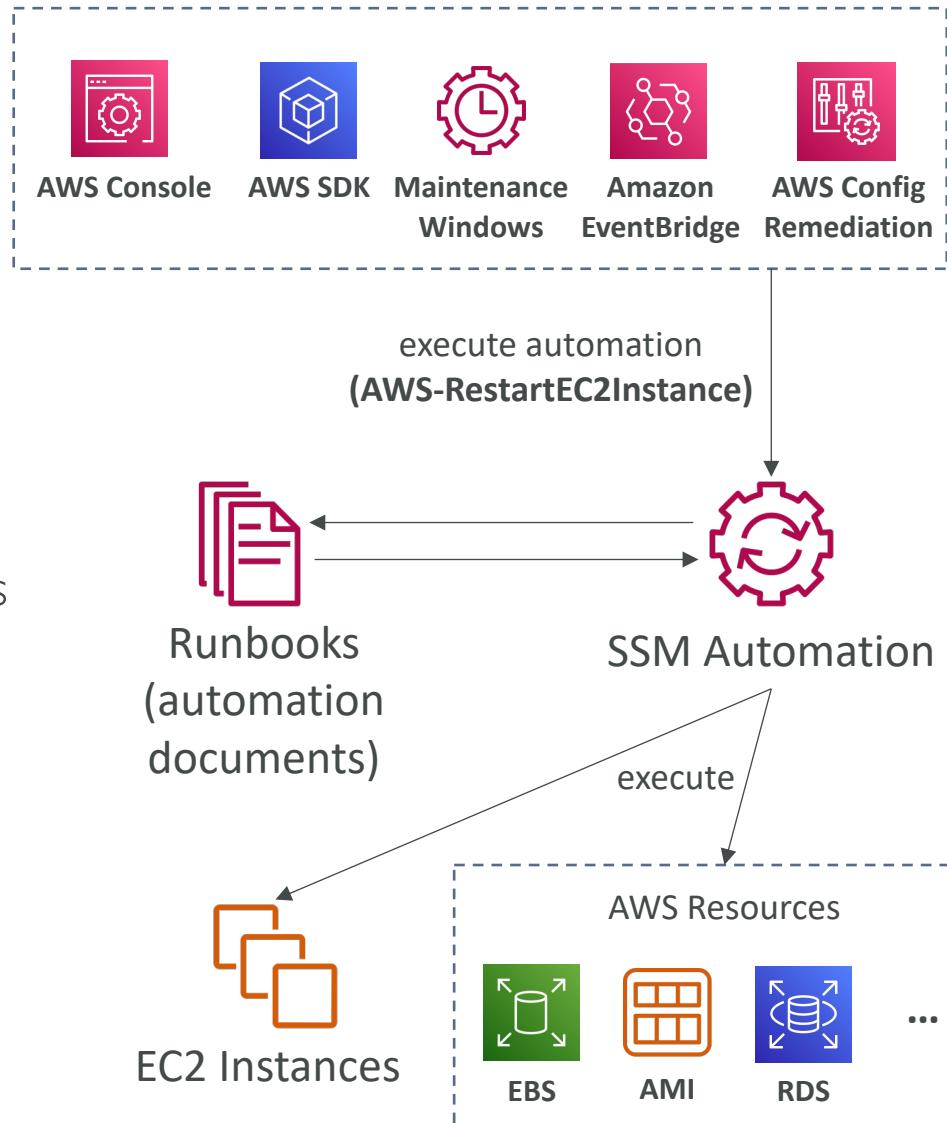
# SSM – Run Command

- Execute a document (= script) or just run a command
- Run command across multiple instances (using resource groups)
- Rate Control / Error Control
- Integrated with IAM & CloudTrail
- No need for SSH
- Command Output can be shown in the Console, sent to S3 bucket or CloudWatch Logs
- Send notifications to SNS about command statuses (In progress, Success, Failed...)
- Can be invoked using EventBridge



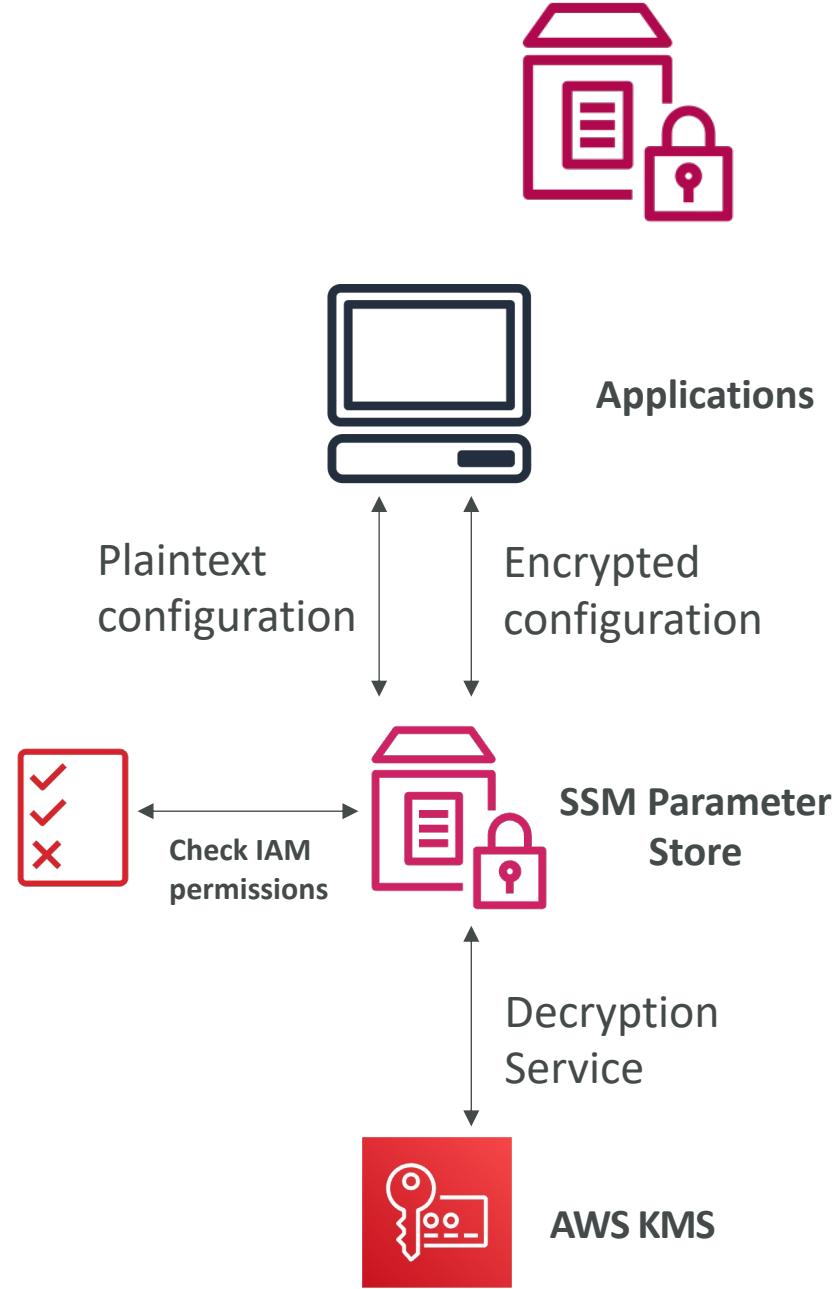
# SSM – Automation

- Simplifies common maintenance and deployment tasks of EC2 instances and other AWS resources
- Example: restart instances, create an AMI, EBS snapshot
- **Automation Runbook**
  - SSM Documents of type Automation
  - Defines actions preformed on your EC2 instances or AWS resources
  - Pre-defined runbooks (AWS) or create custom runbooks
- Can be triggered
  - Manually using AWS Console, AWS CLI or SDK
  - By Amazon EventBridge
  - On a schedule using Maintenance Windows
  - By AWS Config for rules remediations



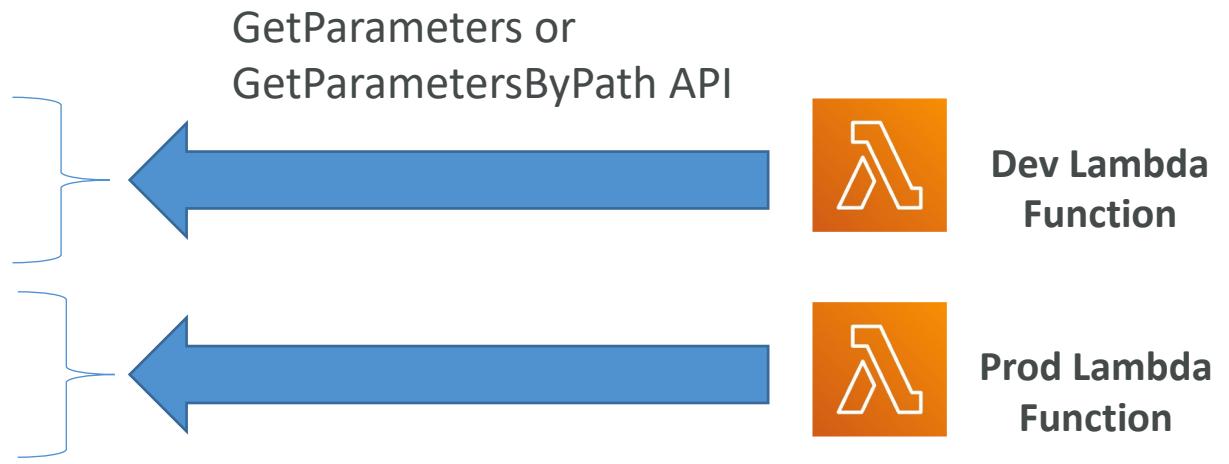
# SSM Parameter Store

- Secure storage for configuration and secrets
- Optional Seamless Encryption using KMS
- Serverless, scalable, durable, easy SDK
- Version tracking of configurations / secrets
- Security through IAM
- Notifications with Amazon EventBridge
- Integration with CloudFormation



# SSM Parameter Store Hierarchy

- /my-department/
  - my-app/
    - dev/
      - db-url
      - db-password
    - prod/
      - db-url
      - db-password
  - other-app/
  - /other-department/
  - /aws/reference/secretsmanager/secret\_ID\_in\_Secrets\_Manager
  - /aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86\_64-gp2 (public)



# Standard and advanced parameter tiers

	Standard	Advanced
Total number of parameters allowed (per AWS account and Region)	10,000	100,000
Maximum size of a parameter value	4 KB	8 KB
Parameter policies available	No	Yes
Cost	No additional charge	Charges apply
Storage Pricing	Free	\$0.05 per advanced parameter per month

# Parameters Policies (for advanced parameters)

- Allow to assign a TTL to a parameter (expiration date) to force updating or deleting sensitive data such as passwords
- Can assign multiple policies at a time

**Expiration (to delete a parameter)**

```
{  
  "Type": "Expiration",  
  "Version": "1.0",  
  "Attributes": {  
    "Timestamp": "2020-12-02T21:34:33.000Z"  
  }  
}
```

**ExpirationNotification (EventBridge)**

```
{  
  "Type": "ExpirationNotification",  
  "Version": "1.0",  
  "Attributes": {  
    "Before": "15",  
    "Unit": "Days"  
  }  
}
```

**NoChangeNotification (EventBridge)**

```
{  
  "Type": "NoChangeNotification",  
  "Version": "1.0",  
  "Attributes": {  
    "After": "20",  
    "Unit": "Days"  
  }  
}
```



# SSM – Patch Manager

- Automates the process of patching managed instances
- OS updates, applications updates, security updates...
- Supports both EC2 instances and on-premises servers
- Supports Linux, macOS, and Windows
- Patch on-demand or on a schedule using **Maintenance Windows**
- Scan instances and generate patch compliance report (missing patches)
- Patch compliance report can be sent to S3



# SSM – Patch Manager

- **Patch Baseline**

- Defines which patches should and shouldn't be installed on your instances
- Ability to create custom Patch Baselines (specify approved/rejected patches)
- Patches can be auto-approved within days of their release
- By default, install only critical patches and patches related to security

- **Patch Group**

- Associate a set of instances with a specific Patch Baseline
- Example: create Patch Groups for different environments (dev, test, prod)
- Instances should be defined with the tag key **Patch Group**
- An instance can only be in one Patch Group
- Patch Group can be registered with only one Patch Baseline

# SSM – Patch Manager Patch Baselines

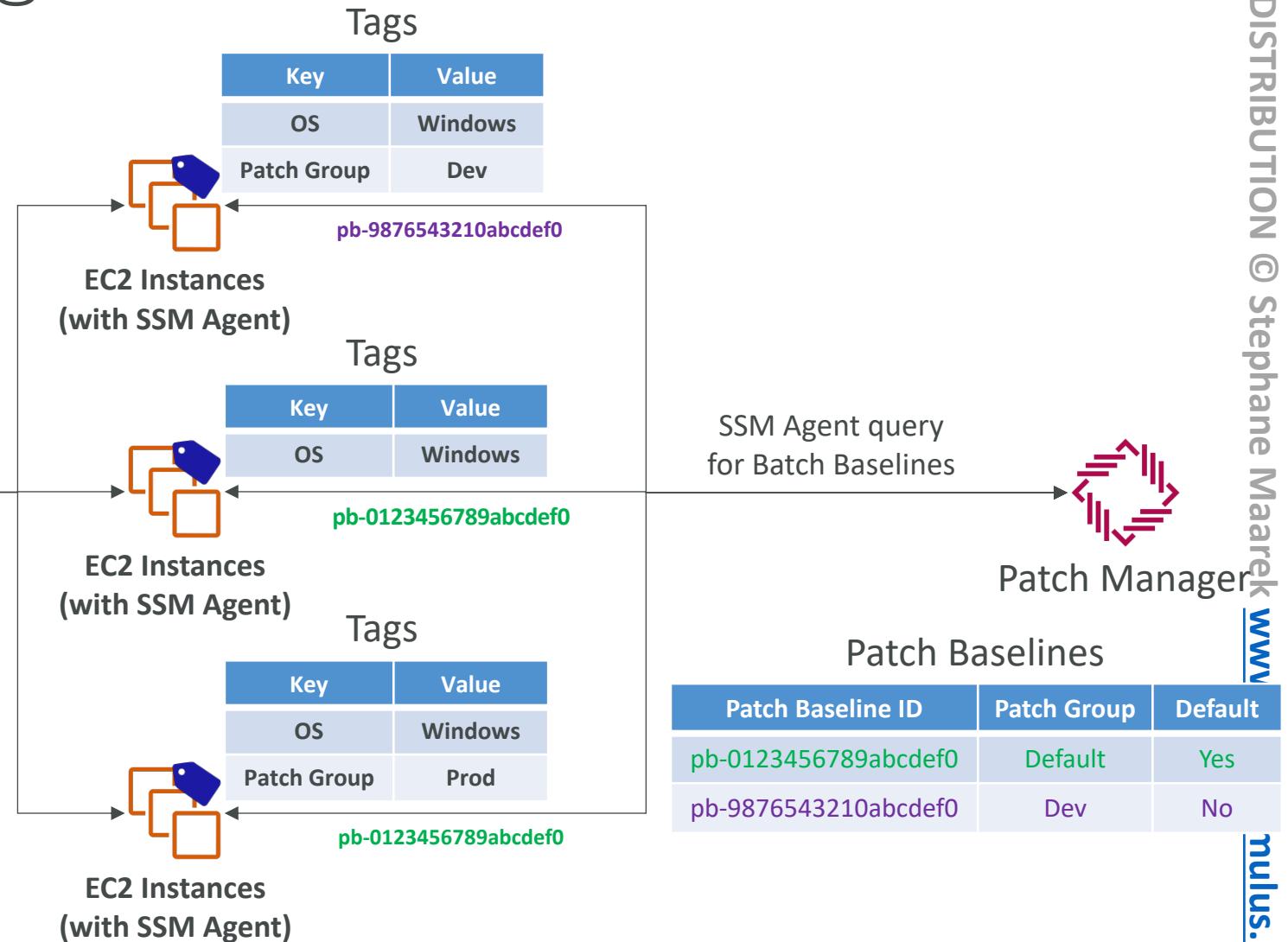
- Pre-Defined Patch Baseline

- Managed by AWS for different Operating Systems (can't be modified)
- **AWS-RunPatchBaseline (SSM Document)** – apply both operating system and application patches (Linux, macOS, Windows Server)

- Custom Patch Baseline

- Create your own Patch Baseline and choose which patches to auto-approve
- Operating System, allowed patches, rejected patches...
- Ability to specify custom and alternative patch repositories

# SSM – Patch Manager



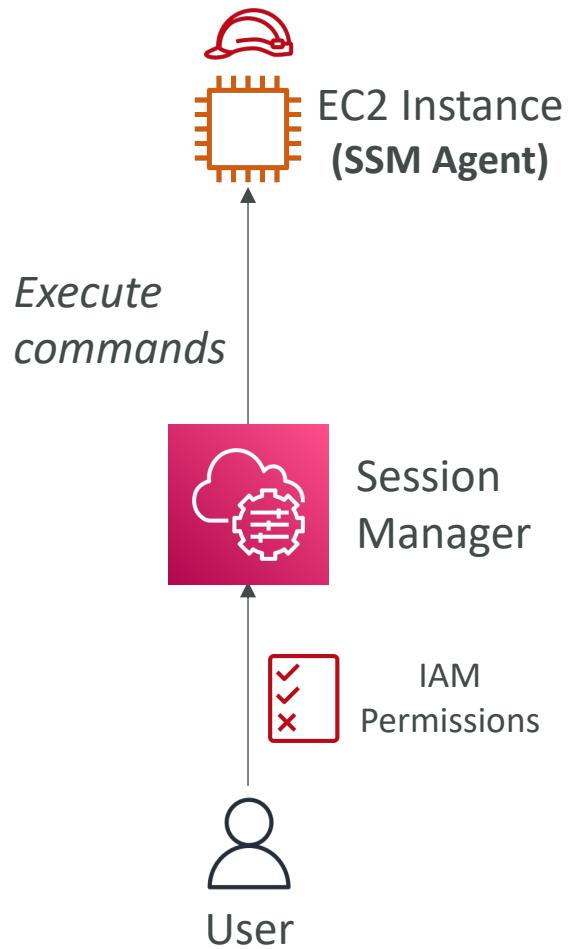


# SSM – Maintenance Windows

- Defines a schedule for when to perform actions on your instances
- Example: OS patching, updating drivers, installing software...
- Maintenance Window contains
  - Schedule
  - Duration
  - Set of registered instances
  - Set of registered tasks

# SSM – Session Manager

- Allows you to start a secure shell on your EC2 and on-premises servers
- Access through AWS Console, AWS CLI, or Session Manager SDK
- Does not need SSH access, bastion hosts, or SSH keys
- Supports Linux, macOS, and Windows
- Log connections to your instances and executed commands
- Session log data can be sent to S3 or CloudWatch Logs
- CloudTrail can intercept **StartSession** events



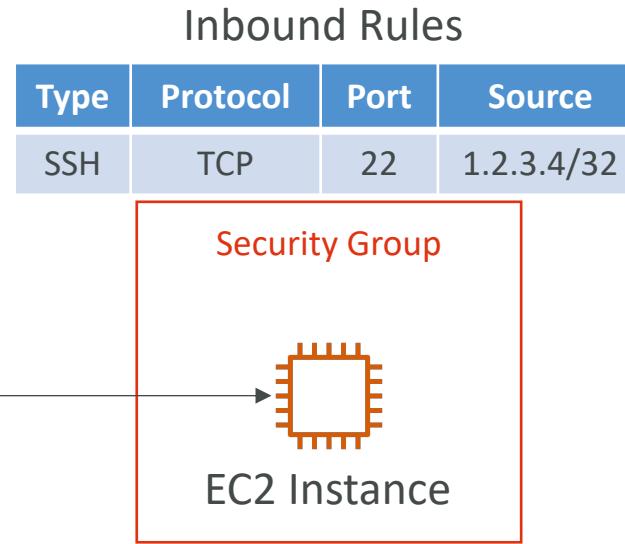
# SSM – Session Manager

- IAM Permissions
  - Control which users/groups can access Session Manager and which instances
  - Use tags to restrict access to only specific EC2 instances
  - Access SSM + write to S3 + write to CloudWatch
- Optionally, you can restrict commands a user can run in a session

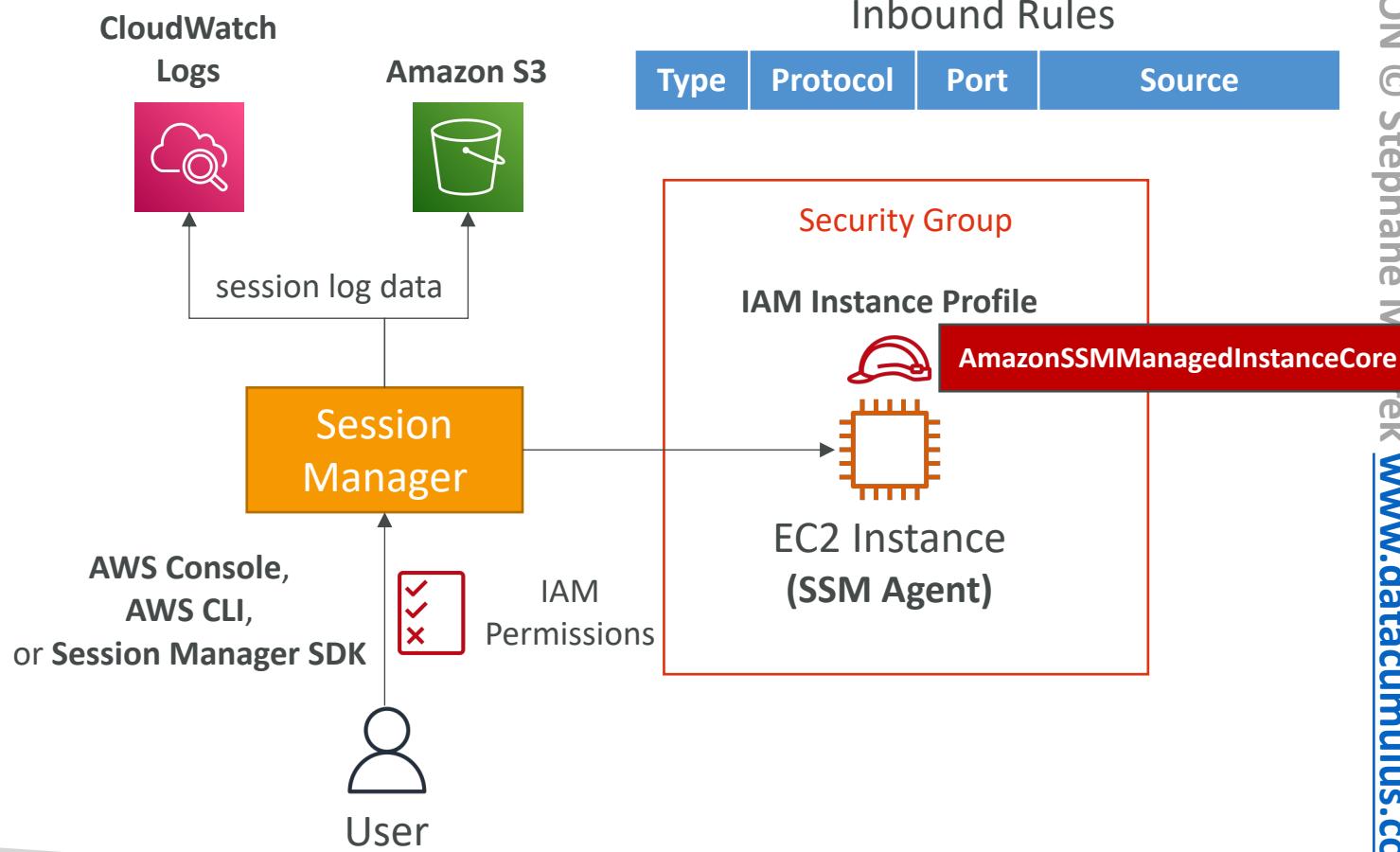
```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "ssm:StartSession",  
            "Resource": "arn:aws:ec2:us-east-1:123456789012:instance/*",  
            "Condition": {  
                "StringLike": {  
                    "ssm:resourceTag/Environment": ["Dev"]  
                }  
            }  
        }  
    ]  
}
```

# SSH vs. SSM Session Manager

# Connect using SSH

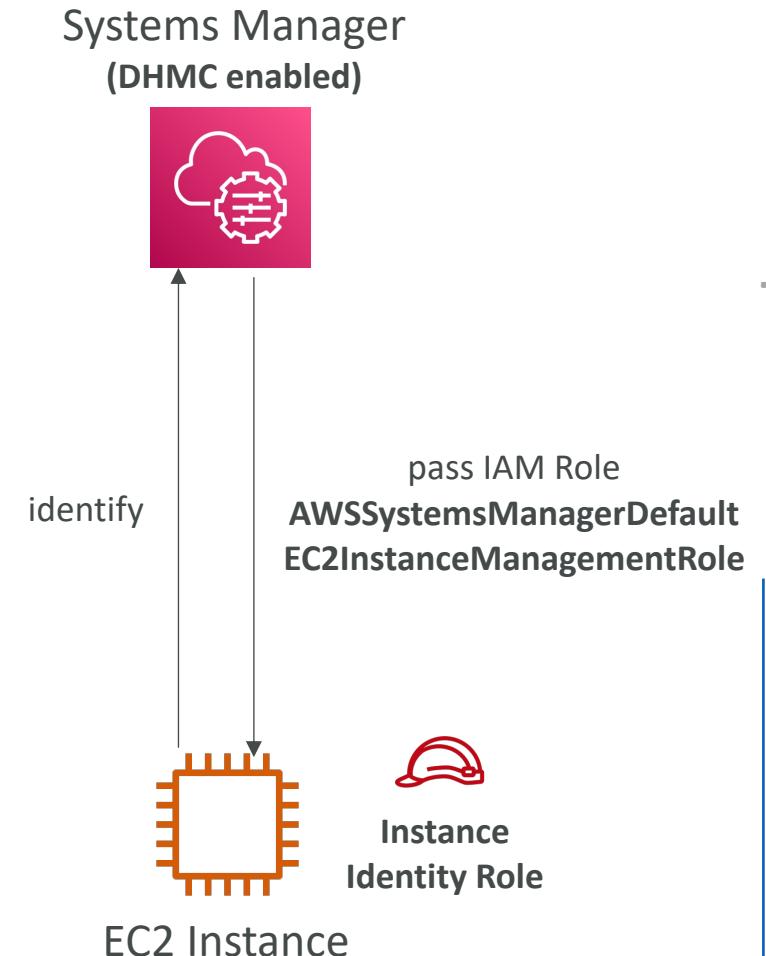


[Connect using SSM Session Manager](#)



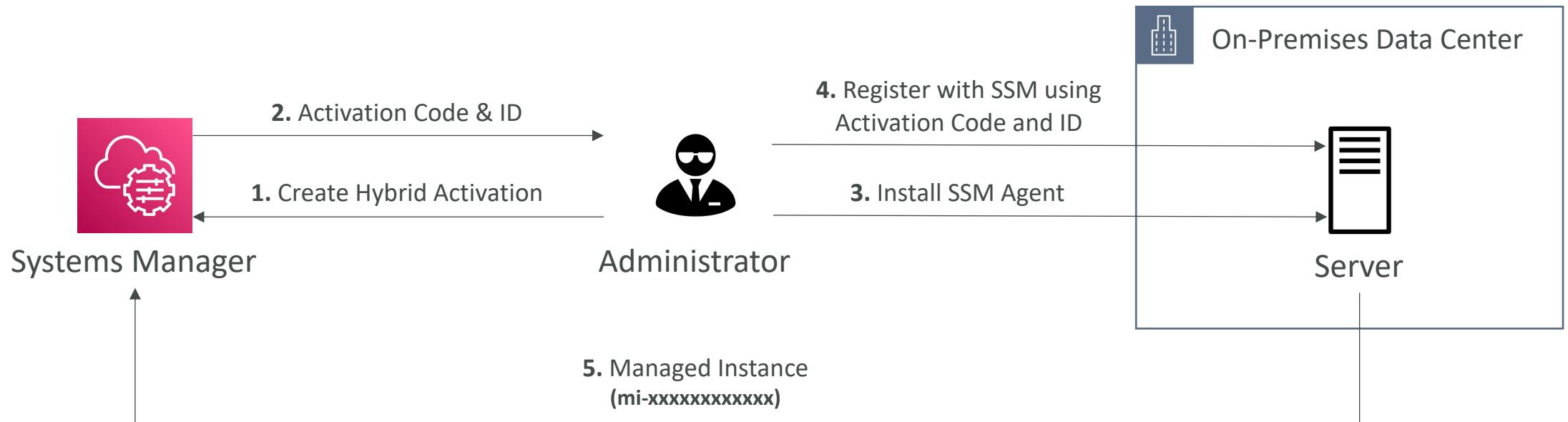
# Systems Manager Default Host Management Configuration

- When enabled, it automatically configures your EC2 instances as managed instances **without the use of EC2 Instance Profile**
- **Instance Identity Role** – a type of IAM Role with no permissions beyond identifying the EC2 instance to AWS Services (e.g., Systems Manager)
- EC2 instances must have **IMDSv2 enabled** and **SSM Agent installed** (**doesn't support IMDSv1**)
- Automatically enables Session Manager, Patch Manager, and Inventory
- Automatically keeps the SSM Agent up to date
- Must be enabled per AWS Region

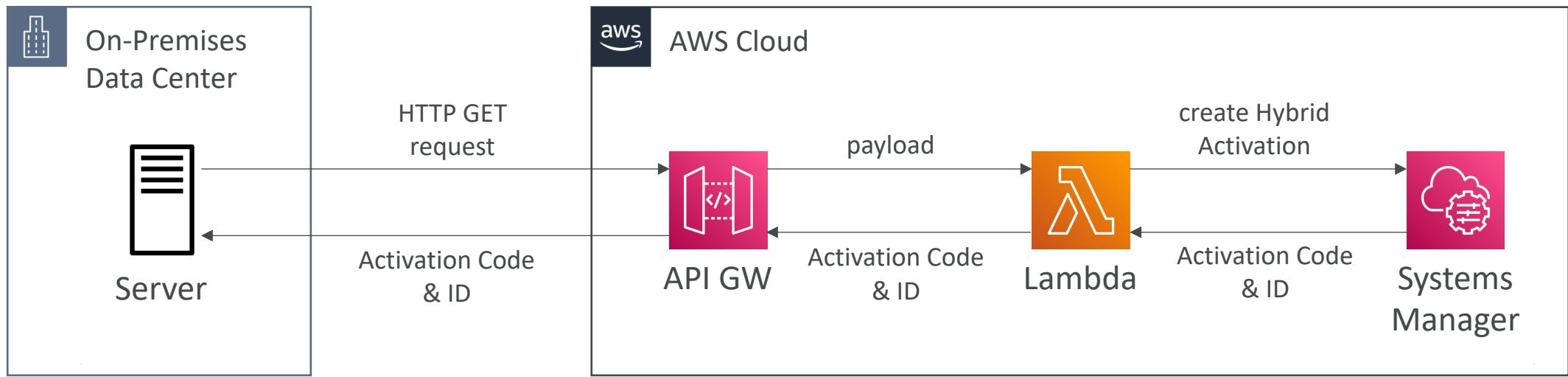


# Systems Manager - Hybrid Environments

- You can setup Systems Manager to manage on-premises servers, IoT devices, edge devices, and virtual machines (e.g., VMs in other cloud providers)
- In Systems Manager Console, EC2 instances use the prefix “i-” and hybrid managed nodes use the prefix “mi-”



# Automating SSM Hybrid Activations Creation



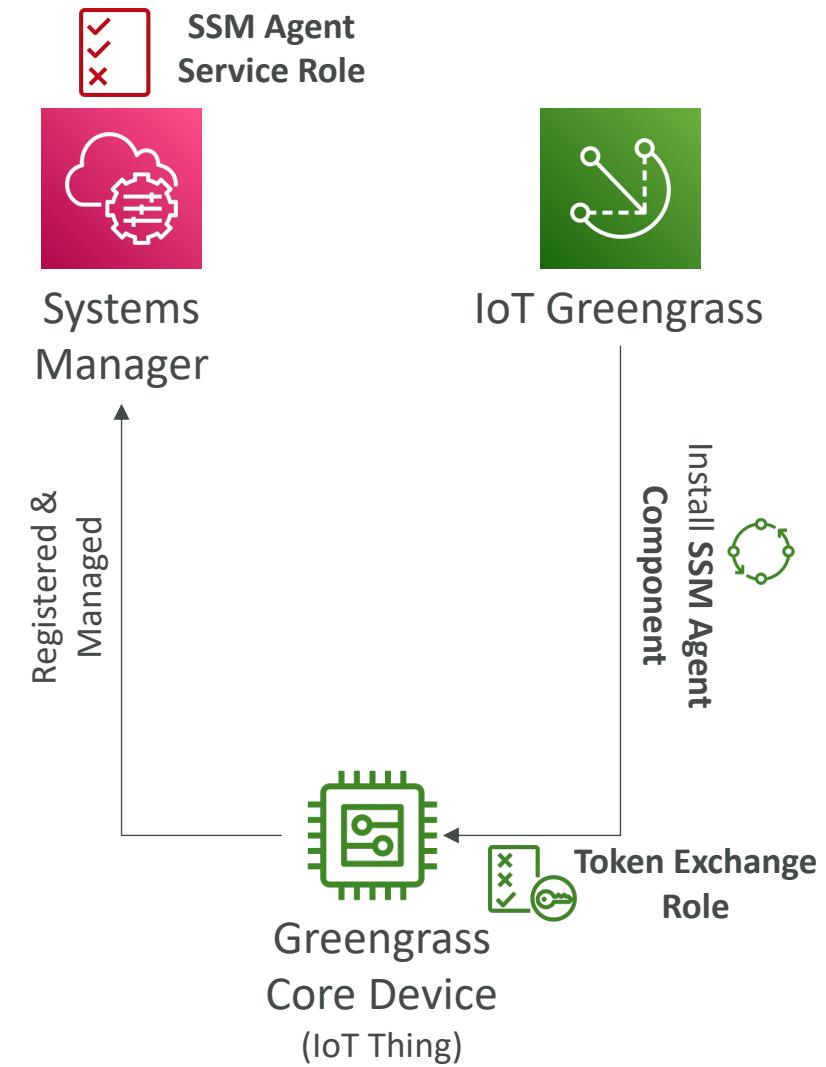
```
#!/bin/bash
sudo yum erase amazon-ssm-agent --assumeyes

credentials=$(curl -s https://xxxxxxxxxx.execute-api.us-east-1.amazonaws.com/lambdastage)
activationcode=$(echo $credentials | jq -r '.ActivationCode')
activationid=$(echo $credentials| jq -r '.ActivationId')

mkdir /tmp/ssm
curl https://s3.amazonaws.com/ec2-downloads-windows/SSMAgent/latest/linux_amd64/amazon-ssm-agent.rpm -o /tmp/ssm/amazon-ssm-agent.rpm
sudo dnf install -y /tmp/ssm/amazon-ssm-agent.rpm
sudo systemctl stop amazon-ssm-agent
sudo -E amazon-ssm-agent -register -code $activationcode -id $activationid -region us-east-1
sudo systemctl start amazon-ssm-agent
```

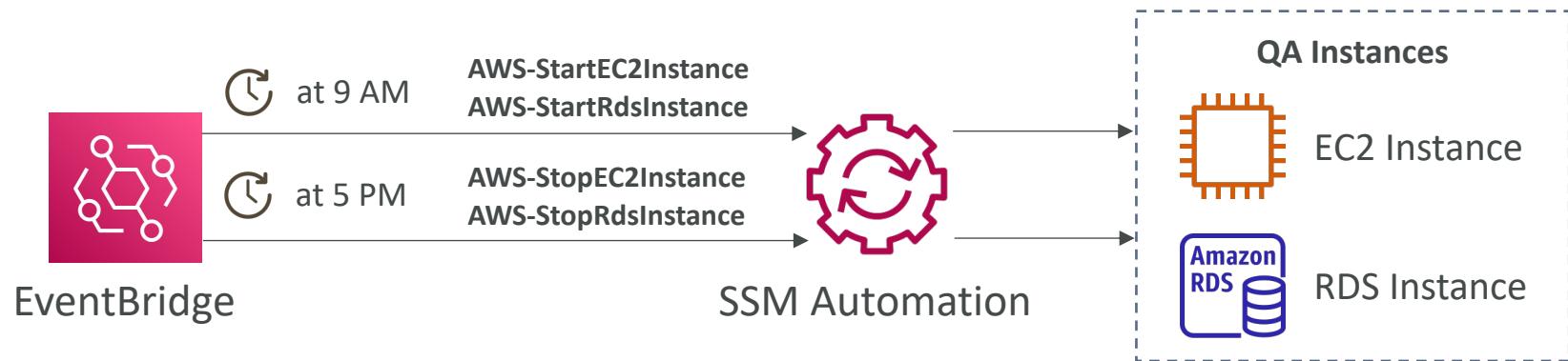
# IoT Greengrass Instance Activation

- Manage IoT Greengrass Core devices using SSM
- Install SSM Agent on Greengrass Core devices (registered as a managed node in SSM)
- SSM Agent can be installed manually or deployed as a Greengrass Component (pre-built software module that you deploy directly to Greengrass Core devices)
- You must add permissions to the Token Exchange Role (IAM Role for the IoT core device) to communicate with Systems Manager
- Supports all SSM Capabilities (Patch Manager, Session Manager, Run Command...)
- Use cases: easily update and maintain OS and software updates across a fleet of Greengrass Core devices

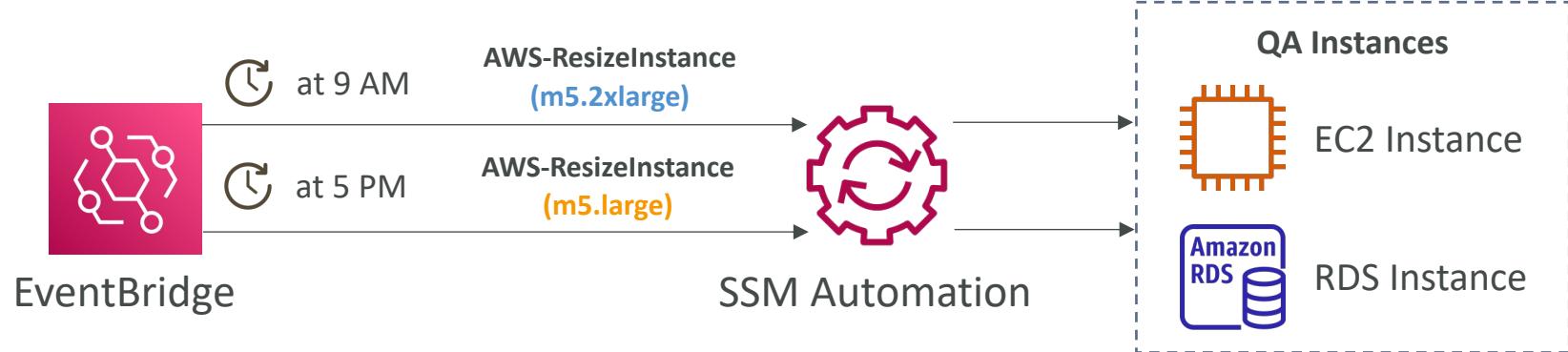


# Systems Manager – Automation – Use Cases

Reduce Costs by automatically  
**start and stop** EC2 instances and  
RDS DB Instances

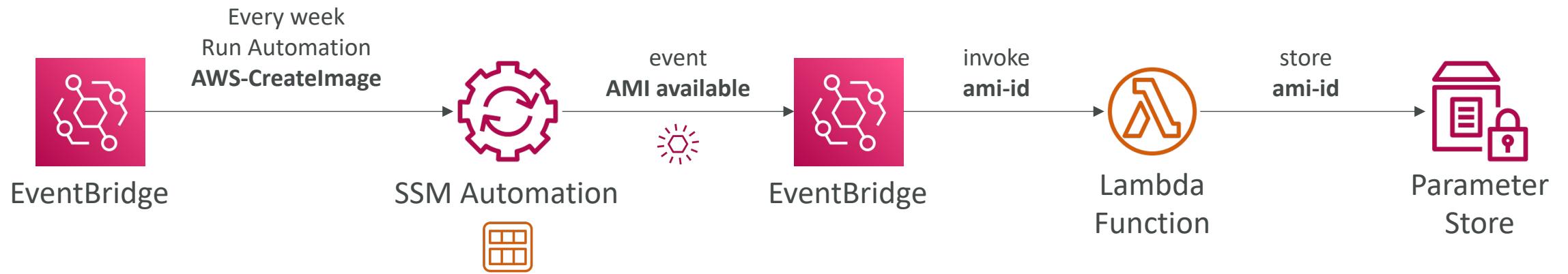


Reduce Costs by automatically  
**downsize** EC2 instances and RDS  
DB Instances

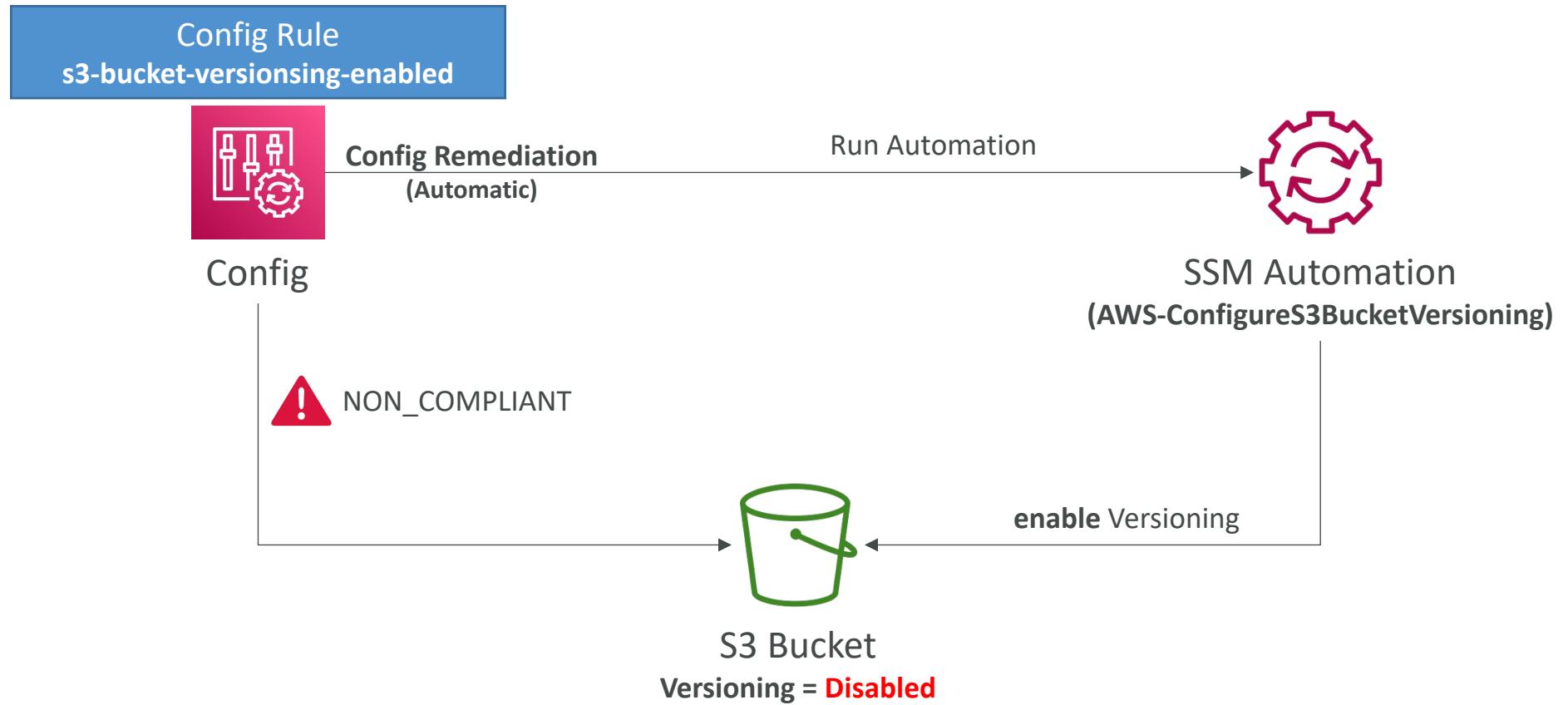


# Systems Manager – Automation – Use Cases

## Build a golden AMI



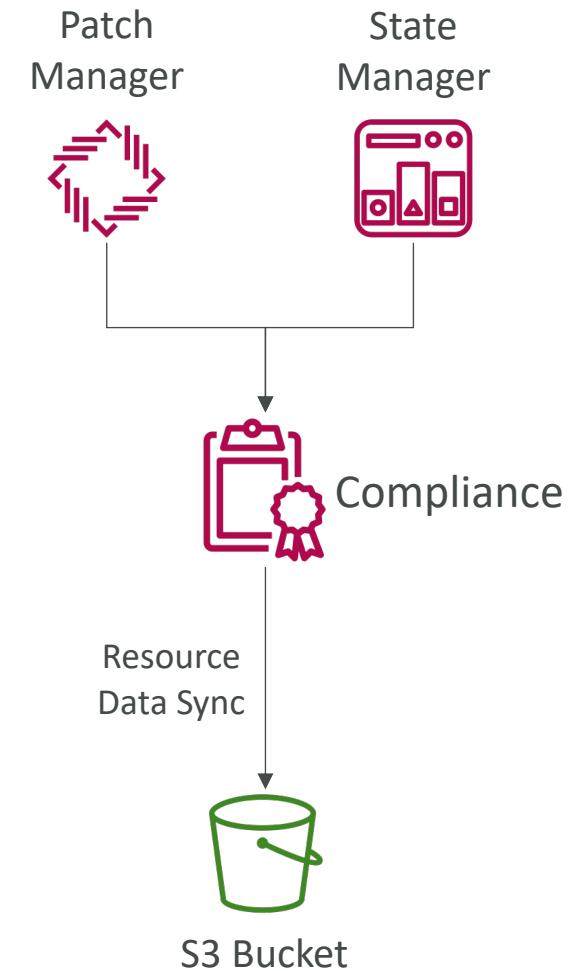
# Systems Manager – Automation – Use Cases



# Systems Manager – Compliance



- Scan your fleet of managed nodes for patch compliance and configuration inconsistencies
- Displays current data about:
  - Patches in Patch Manager
  - Associations in State Manager
- Can sync data to an S3 bucket using **Resource Data Sync**, and you can analyze using Athena and QuickSight
- Can collect and aggregate data from multiple accounts and regions
- Can send compliance data to Security Hub

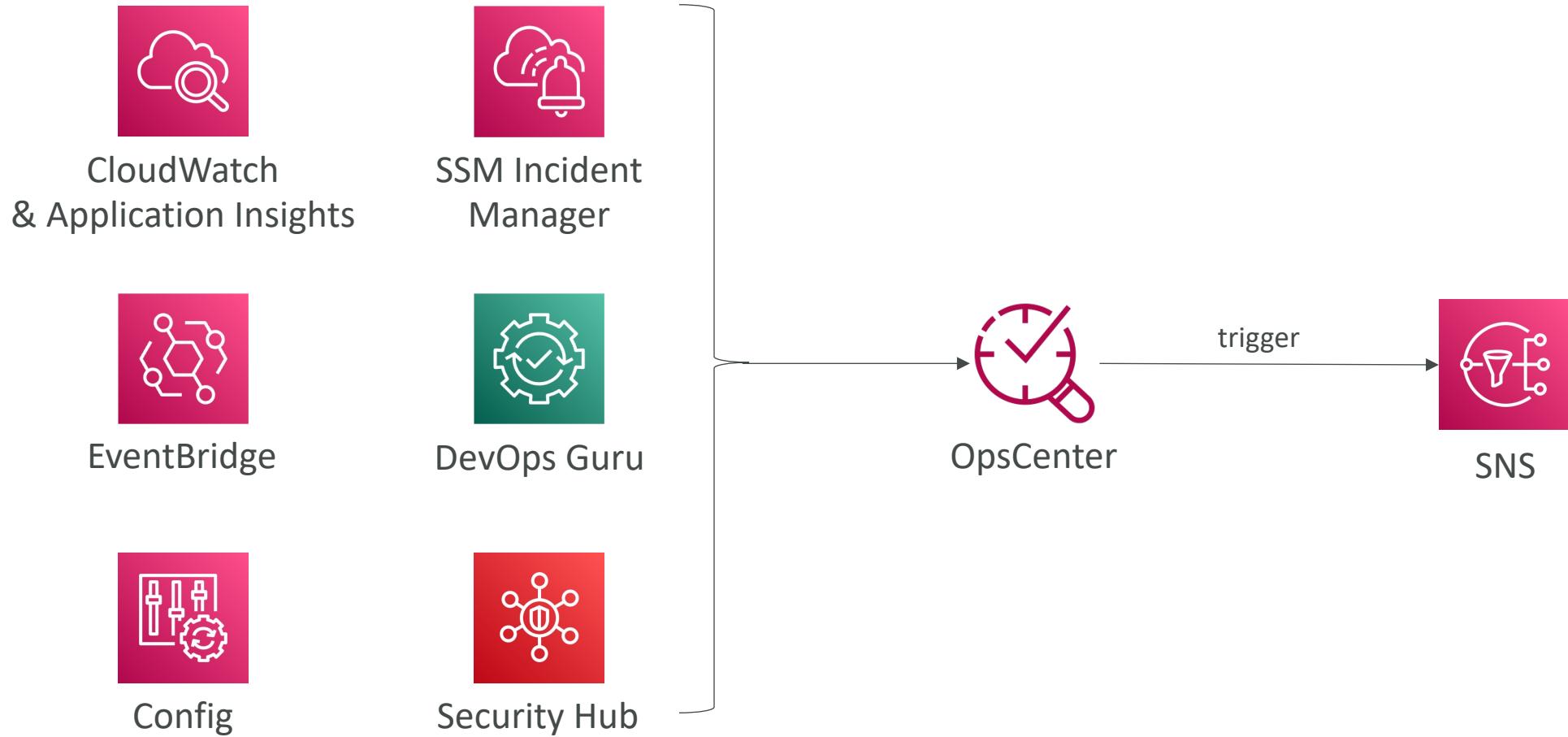




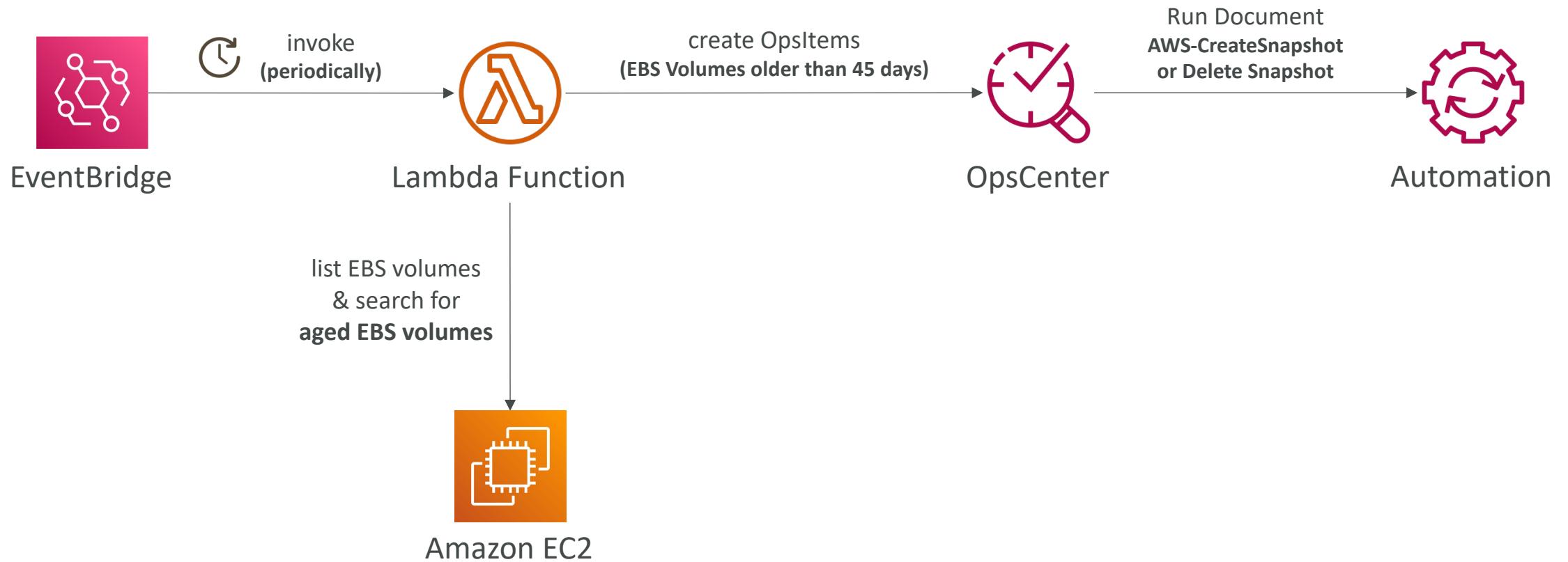
# Systems Manager – OpsCenter

- Allows you to view, investigate, and remediate issues in one place (no need to navigate across different AWS services)
- Security issues (Security Hub), performance issues (DynamoDB throttle), failures (ASG failed launch instance)...
- Reduce meantime to resolve issues
- **OpsItems**
  - Operational issue or interruption that needs investigation and remediation
  - Event, resource, AWS Config changes, CloudTrail logs, EventBridge...
  - **Provides recommended Runbooks to resolve the issue**
- Supports both EC2 instances and on-premises managed nodes

# Systems Manager – OpsCenter

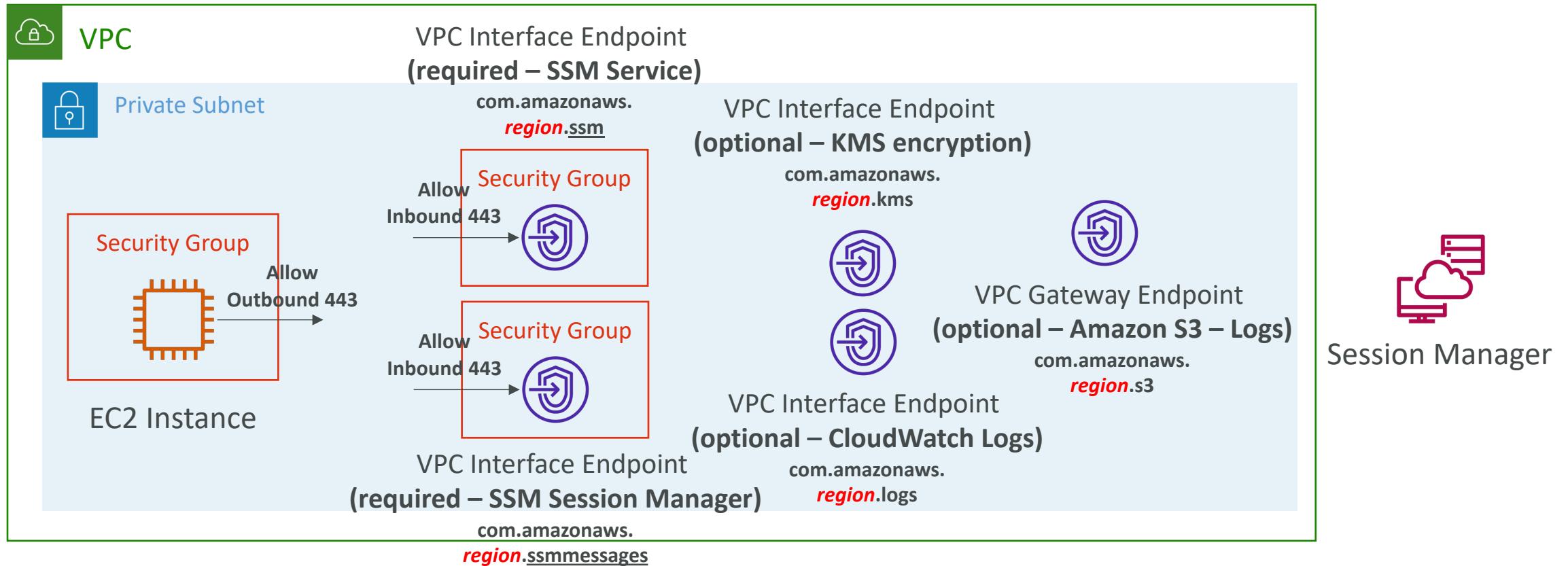


# Systems Manager – OpsCenter – Reduce Costs by Deleting Orphaned EBS Volumes



# VPC Endpoint – SSM Session Manager

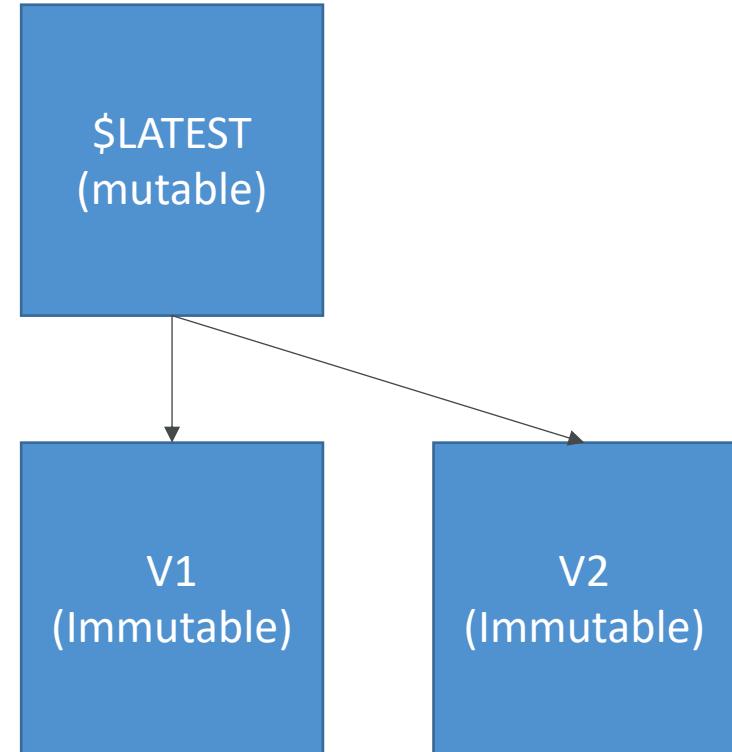
- Connect to EC2 instances in private subnets, without Internet access



# Domain 3 – Resilient Cloud Solutions

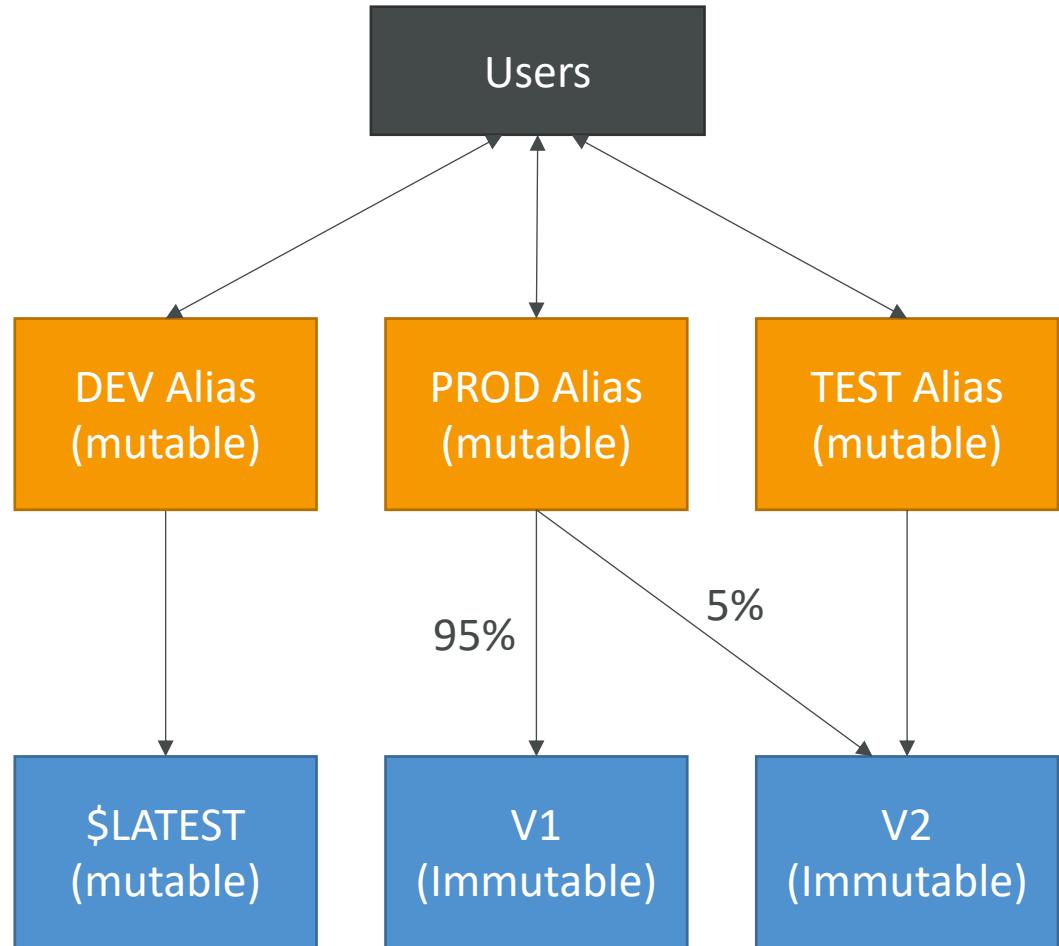
# AWS Lambda Versions

- When you work on a Lambda function, we work on **\$LATEST**
- When we're ready to publish a Lambda function, we create a version
- Versions are immutable
- Versions have increasing version numbers
- Versions get their own ARN (Amazon Resource Name)
- Version = code + configuration (nothing can be changed - immutable)
- Each version of the lambda function can be accessed



# AWS Lambda Aliases

- Aliases are "pointers" to Lambda function versions
- We can define a "dev", "test", "prod" aliases and have them point at different lambda versions
- Aliases are mutable
- Aliases enable Canary deployment by assigning weights to lambda functions
- Aliases enable stable configuration of our event triggers / destinations
- Aliases have their own ARNs
- Aliases cannot reference aliases

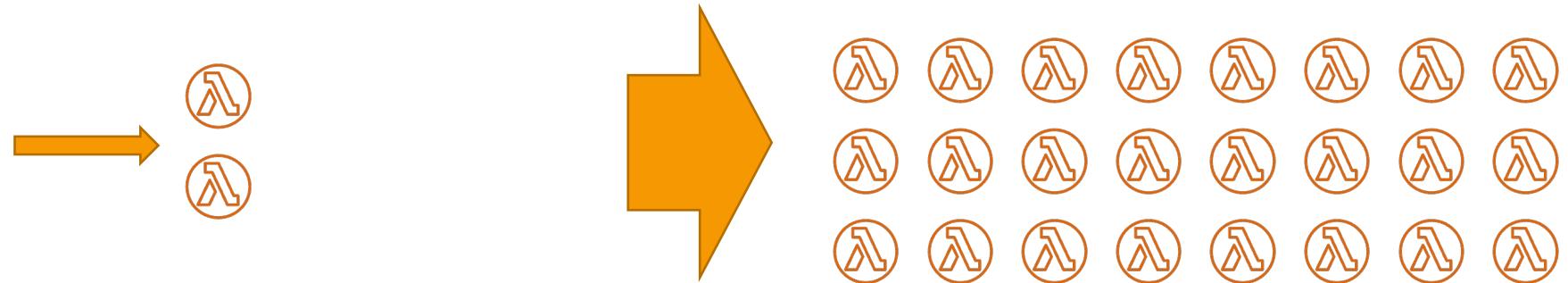


# Lambda Environment Variables

- Environment variable = key / value pair in “String” form
- Adjust the function behavior without updating code
- The environment variables are available to your code
- Lambda Service adds its own system environment variables as well
  
- Helpful to store secrets (encrypted by KMS)
- Secrets can be encrypted by the Lambda service key, or your own CMK

# Lambda Concurrency and Throttling

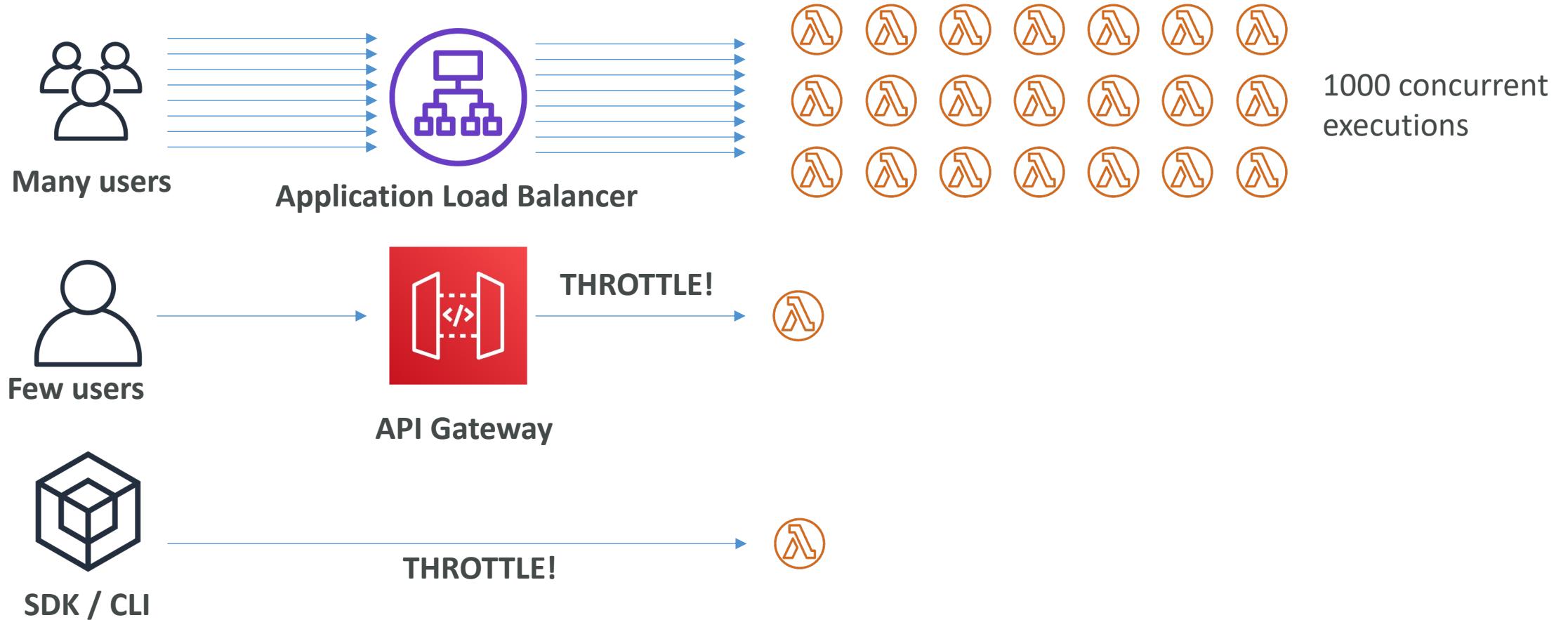
- Concurrency limit: up to 1000 concurrent executions



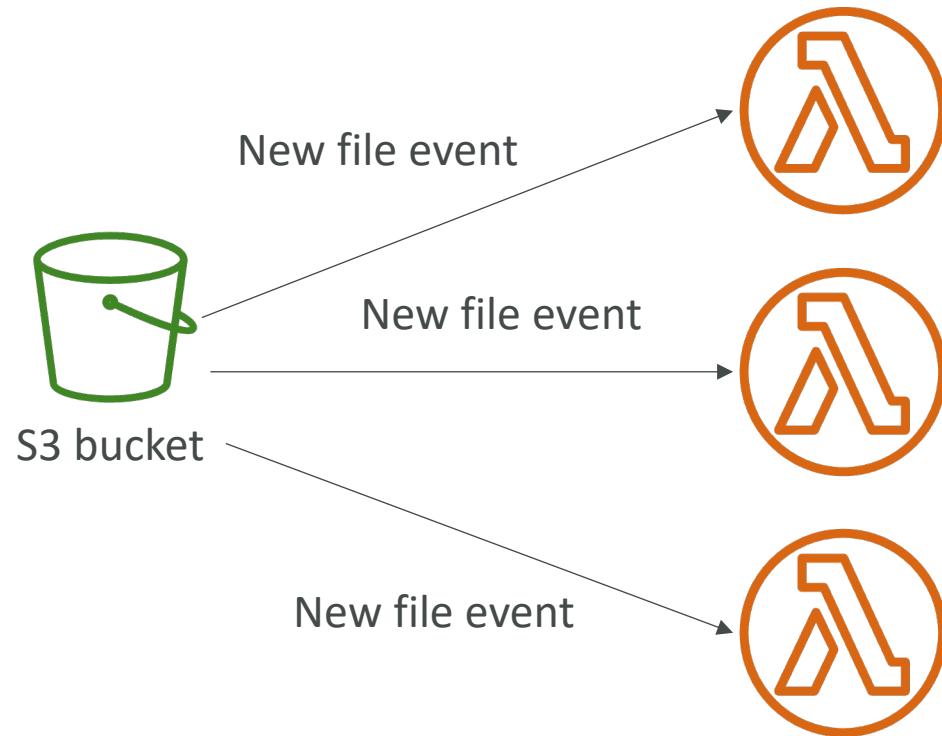
- Can set a “reserved concurrency” at the function level (=limit)
- Each invocation over the concurrency limit will trigger a “Throttle”
- Throttle behavior:
  - If synchronous invocation => return ThrottleError - 429
  - If asynchronous invocation => retry automatically and then go to DLQ
- If you need a higher limit, open a support ticket

# Lambda Concurrency Issue

- If you don't reserve (=limit) concurrency, the following can happen:



# Concurrency and Asynchronous Invocations



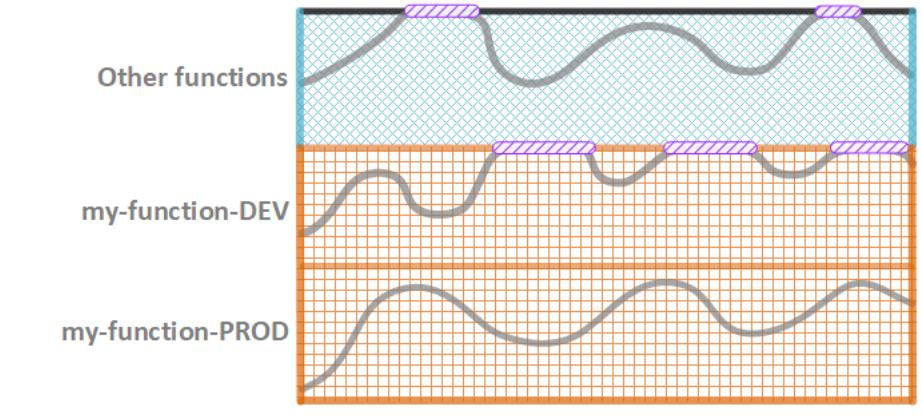
- If the function doesn't have enough concurrency available to process all events, additional requests are throttled.
- For throttling errors (429) and system errors (500-series), Lambda returns the event to the queue and attempts to run the function again for up to 6 hours.
- The retry interval increases exponentially from 1 second after the first attempt to a maximum of 5 minutes.

# Cold Starts & Provisioned Concurrency

- **Cold Start:**
  - New instance => code is loaded and code outside the handler run (init)
  - If the init is large (code, dependencies, SDK...) this process can take some time.
  - First request served by new instances has higher latency than the rest
- **Provisioned Concurrency:**
  - Concurrency is allocated before the function is invoked (in advance)
  - So the cold start never happens and all invocations have low latency
  - Application Auto Scaling can manage concurrency (schedule or target utilization)
- **Note:**
  - Note: cold starts in VPC have been dramatically reduced in Oct & Nov 2019
  - <https://aws.amazon.com/blogs/compute/announcing-improved-vpc-networking-for-aws-lambda-functions/>

# Reserved and Provisioned Concurrency

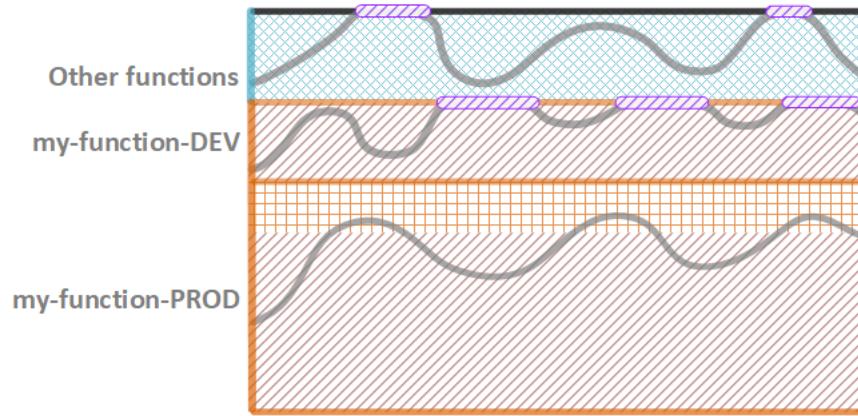
Reserved Concurrency



Legend

- Function concurrency
- Reserved concurrency
- Unreserved concurrency
- Throttling

Provisioned Concurrency with Reserved Concurrency



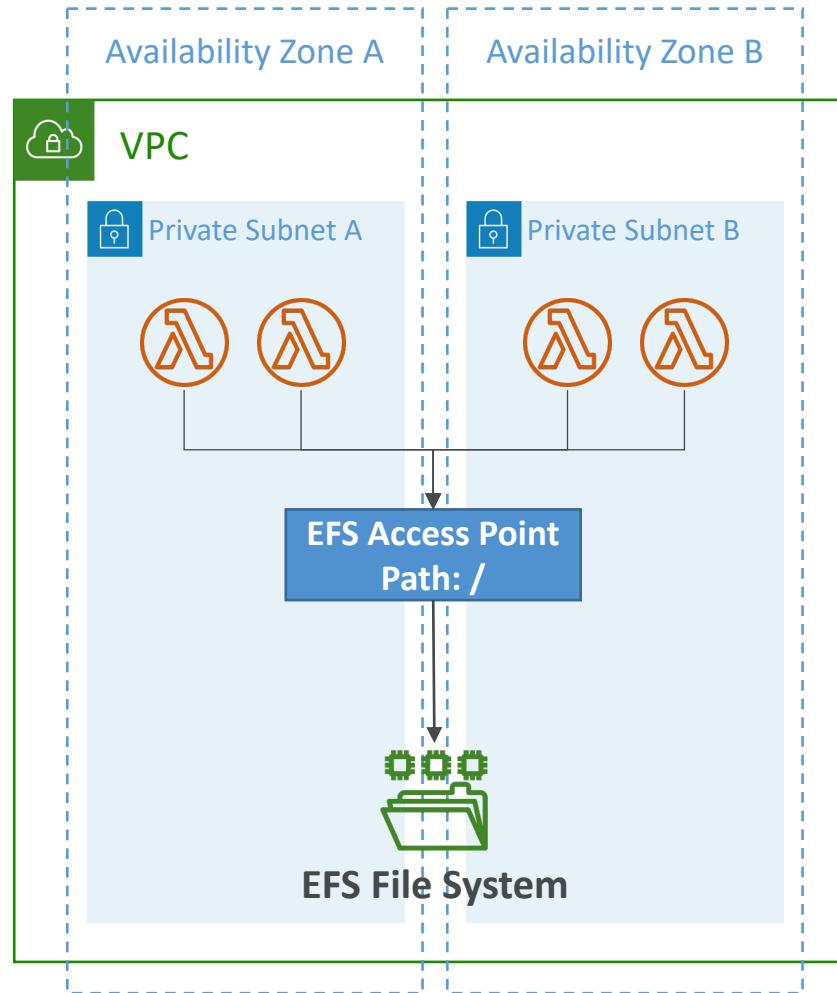
Legend

- Function concurrency
- Reserved concurrency
- Provisioned concurrency
- Unreserved concurrency
- Throttling

<https://docs.aws.amazon.com/lambda/latest/dg/configuration-concurrency.html>

# Lambda – File Systems Mounting

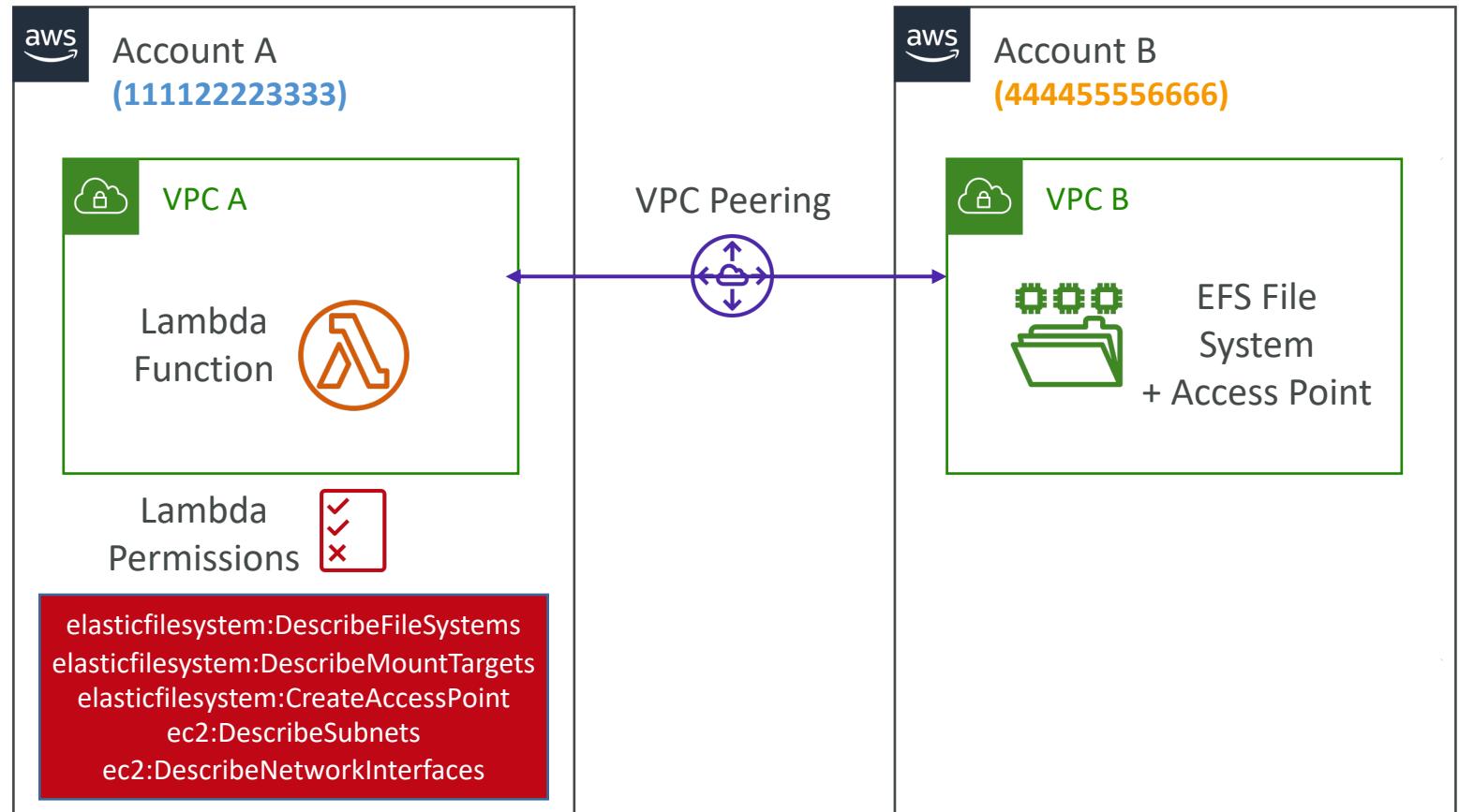
- Lambda functions can access EFS file systems if they are running in a VPC
- Configure Lambda to mount EFS file systems to local directory during initialization
- Must leverage EFS Access Points
- Limitations: watch out for the EFS connection limits (one function instance = one connection) and connection burst limits



# Lambda – Storage Options

	Ephemeral Storage /tmp	Lambda Layers	Amazon S3	Amazon EFS
<b>Max. Size</b>	10,240 MB	5 layers per function up to 250MB total	Elastic	Elastic
<b>Persistence</b>	Ephemeral	Durable	Durable	Durable
<b>Content</b>	Dynamic	Static	Dynamic	Dynamic
<b>Storage Type</b>	File System	Archive	Object	File System
<b>Operations supported</b>	any File System operation	Immutable	Atomic with Versioning	any File System operation
<b>Pricing</b>	Included in Lambda	Included in Lambda	Storage + Requests + Data Transfer	Storage + Data Transfer + Throughput
<b>Sharing/Permissions</b>	Function Only	IAM	IAM	IAM + NFS
<b>Relative Data Access Speed from Lambda</b>	Fastest	Fastest	Fast	Very Fast
<b>Shared Across All Invocations</b>	No	Yes	Yes	Yes

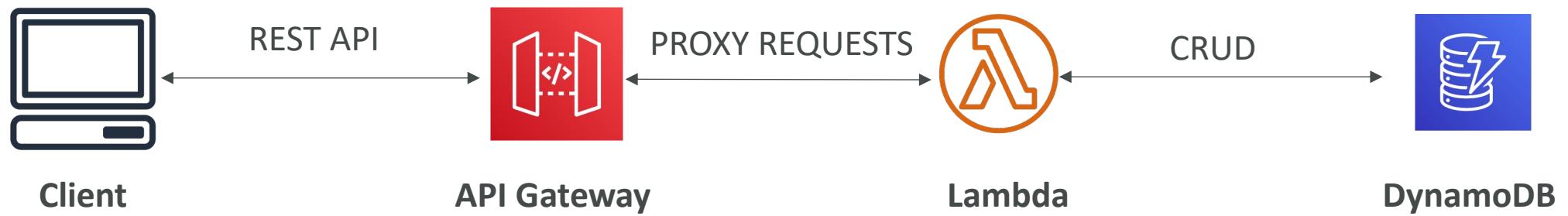
# Lambda – Cross-Account EFS Mounting



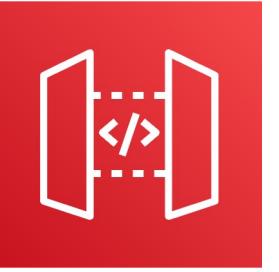
## EFS File System Policy

```
"Statement": [  
    {  
        "Effect": "Allow",  
        "Action": [  
            "elasticfilesystem:ClientMount",  
            "elasticfilesystem:ClientWrite"  
        ],  
        "Principal": {  
            "AWS": "arn:aws:iam::111122223333:root"  
        }  
    }  
]
```

# Example: Building a Serverless API



# AWS API Gateway



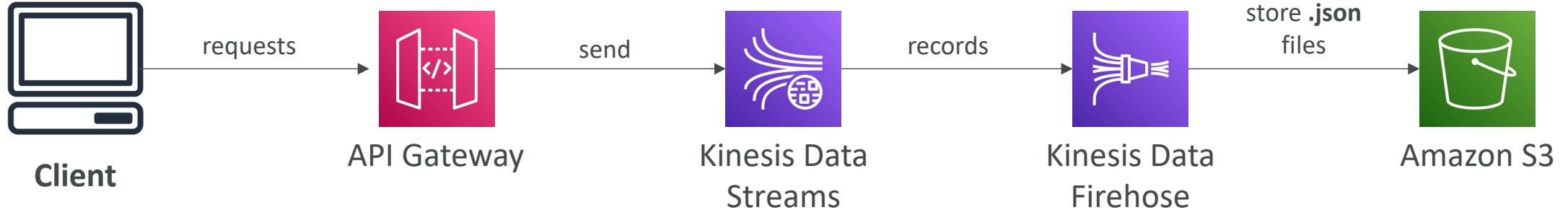
- AWS Lambda + API Gateway: No infrastructure to manage
- Support for the WebSocket Protocol
- Handle API versioning (v1, v2...)
- Handle different environments (dev, test, prod...)
- Handle security (Authentication and Authorization)
- Create API keys, handle request throttling
- Swagger / Open API import to quickly define APIs
- Transform and validate requests and responses
- Generate SDK and API specifications
- Cache API responses

# API Gateway – Integrations High Level

- Lambda Function
  - Invoke Lambda function
  - Easy way to expose REST API backed by AWS Lambda
- HTTP
  - Expose HTTP endpoints in the backend
  - Example: internal HTTP API on premise, Application Load Balancer...
  - Why? Add rate limiting, caching, user authentications, API keys, etc...
- AWS Service
  - Expose any AWS API through the API Gateway
  - Example: start an AWS Step Function workflow, post a message to SQS
  - Why? Add authentication, deploy publicly, rate control...

# API Gateway – AWS Service Integration

## Kinesis Data Streams example



# API Gateway - Endpoint Types

- **Edge-Optimized (default):** For global clients
  - Requests are routed through the CloudFront Edge locations (improves latency)
  - The API Gateway still lives in only one region
- **Regional:**
  - For clients within the same region
  - Could manually combine with CloudFront (more control over the caching strategies and the distribution)
- **Private:**
  - Can only be accessed from your VPC using an interface VPC endpoint (ENI)
  - Use a resource policy to define access

# API Gateway – Security

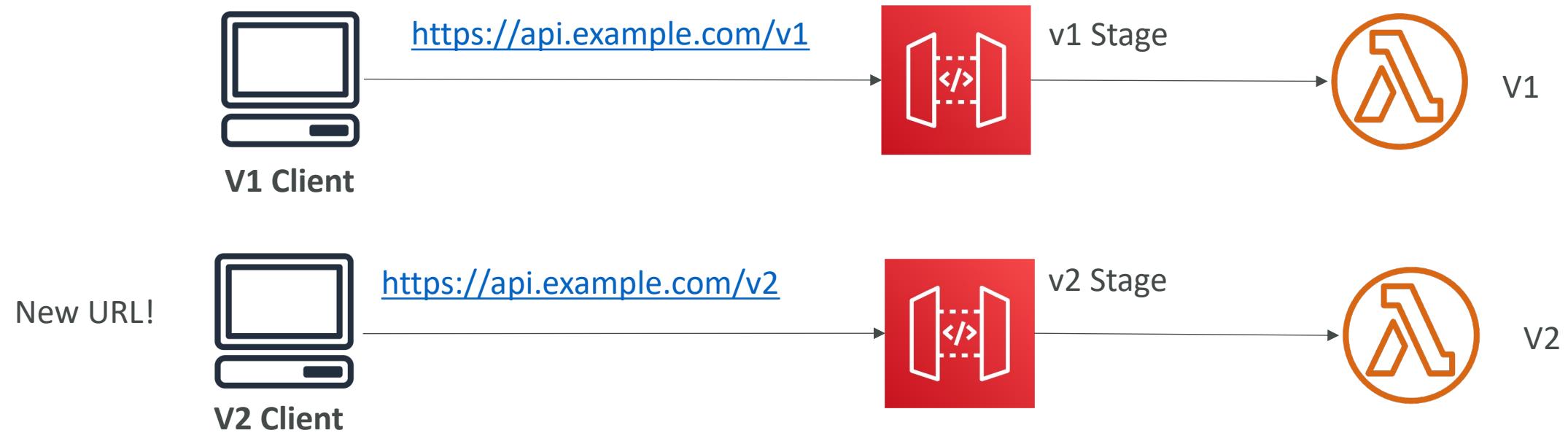
- User Authentication through
  - IAM Roles (useful for internal applications)
  - Cognito (identity for external users – example mobile users)
  - Custom Authorizer (your own logic)
- Custom Domain Name HTTPS security through integration with AWS Certificate Manager (ACM)
  - If using Edge-Optimized endpoint, then the certificate must be in **us-east-1**
  - If using Regional endpoint, the certificate must be in the API Gateway region
  - Must setup CNAME or A-alias record in Route 53

# API Gateway – Deployment Stages

- Making changes in the API Gateway does not mean they're effective
- You need to make a “deployment” for them to be in effect
- It's a common source of confusion
- Changes are deployed to “Stages” (as many as you want)
- Use the naming you like for stages (dev, test, prod)
- Each stage has its own configuration parameters
- Stages can be rolled back as a history of deployments is kept

# API Gateway – Stages v1 and v2

## API breaking change

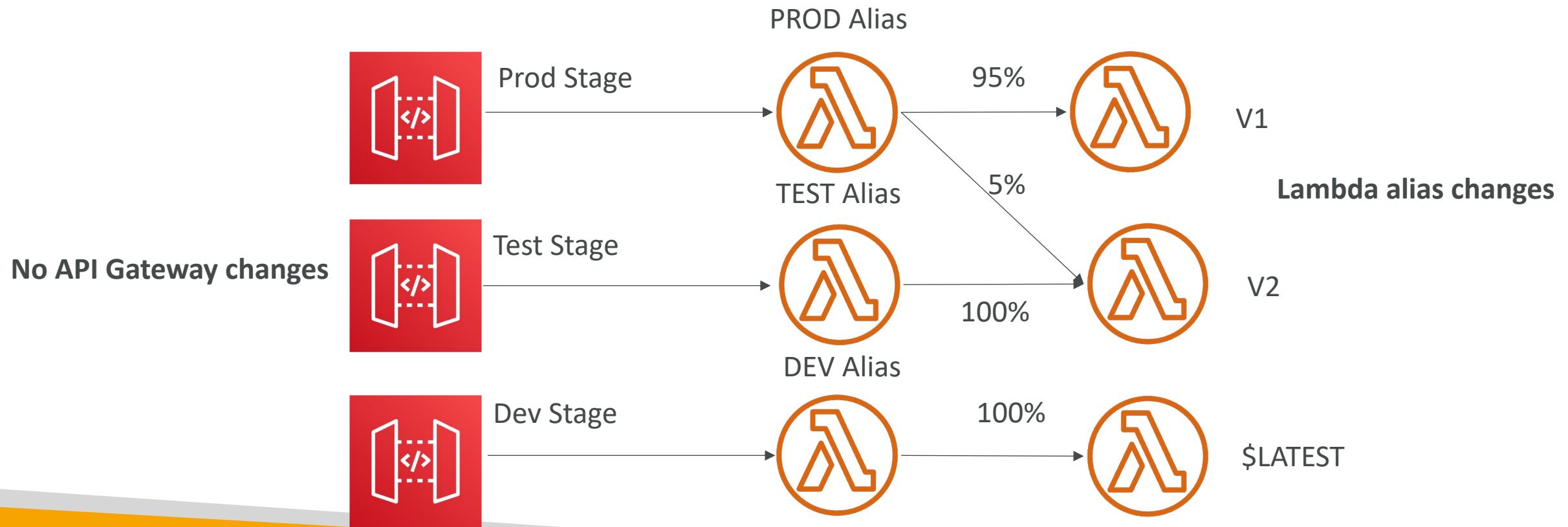


# API Gateway – Stage Variables

- Stage variables are like environment variables for API Gateway
- Use them to change often changing configuration values
- They can be used in:
  - Lambda function ARN
  - HTTP Endpoint
  - Parameter mapping templates
- Use cases:
  - Configure HTTP endpoints your stages talk to (dev, test, prod...)
  - Pass configuration parameters to AWS Lambda through mapping templates
- Stage variables are passed to the "context" object in AWS Lambda
- Format: \${stageVariables.variableName}

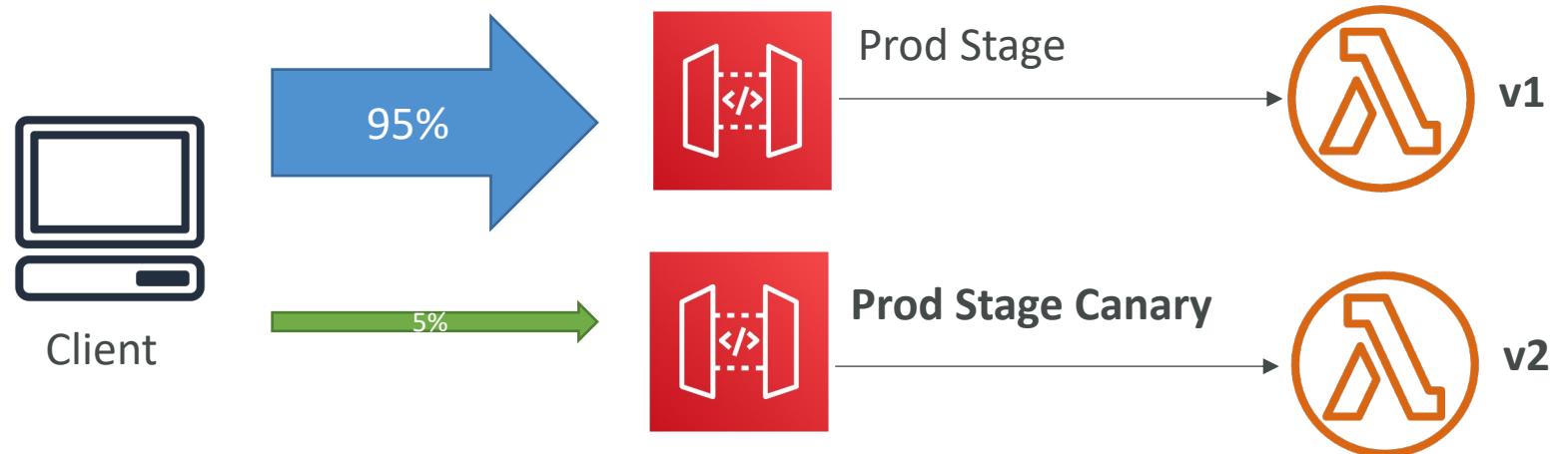
# API Gateway Stage Variables & Lambda Aliases

- We create a **stage variable** to indicate the corresponding Lambda alias
- Our API gateway will automatically invoke the right Lambda function!



# API Gateway – Canary Deployment

- Possibility to enable canary deployments for any stage (usually prod)
- Choose the % of traffic the canary channel receives



- Metrics & Logs are separate (for better monitoring)
- Possibility to override stage variables for canary
- This is blue / green deployment with AWS Lambda & API Gateway

# API Gateway - Open API spec

- Common way of defining REST APIs, using API definition as code
- Import existing OpenAPI 3.0 spec to API Gateway
  - Method
  - Method Request
  - Integration Request
  - Method Response
  - + AWS extensions for API gateway and setup every single option
- Can export current API as OpenAPI spec
- OpenAPI specs can be written in YAML or JSON
- Using OpenAPI we can generate SDK for our applications



# REST API – Request Validation

- You can configure API Gateway to perform basic validation of an API request before proceeding with the integration request
- When the validation fails, API Gateway immediately fails the request
  - Returns a 400-error response to the caller
- This reduces unnecessary calls to the backend
- Checks:
  - The required request parameters in the URI, query string, and headers of an incoming request are included and non-blank
  - The applicable request payload adheres to the configured JSON Schema request model of the method

# REST API – RequestValidation – OpenAPI

- Setup request validation by importing OpenAPI definitions file

```
{  
    "openapi": "3.0.0",  
    "info": {  
        "title": "ReqValidation Sample",  
        "version": "1.0.0"  
    },  
    "servers": [ ... ],  
    "x-amazon-apigateway-request-validation": {  
        "all": {  
            "validateRequestBody": true,  
            "validateRequestParameters": true  
        },  
        "params-only": {  
            "validateRequestBody": false,  
            "validateRequestParameters": true  
        }  
    }  
}
```

Defines the Validators

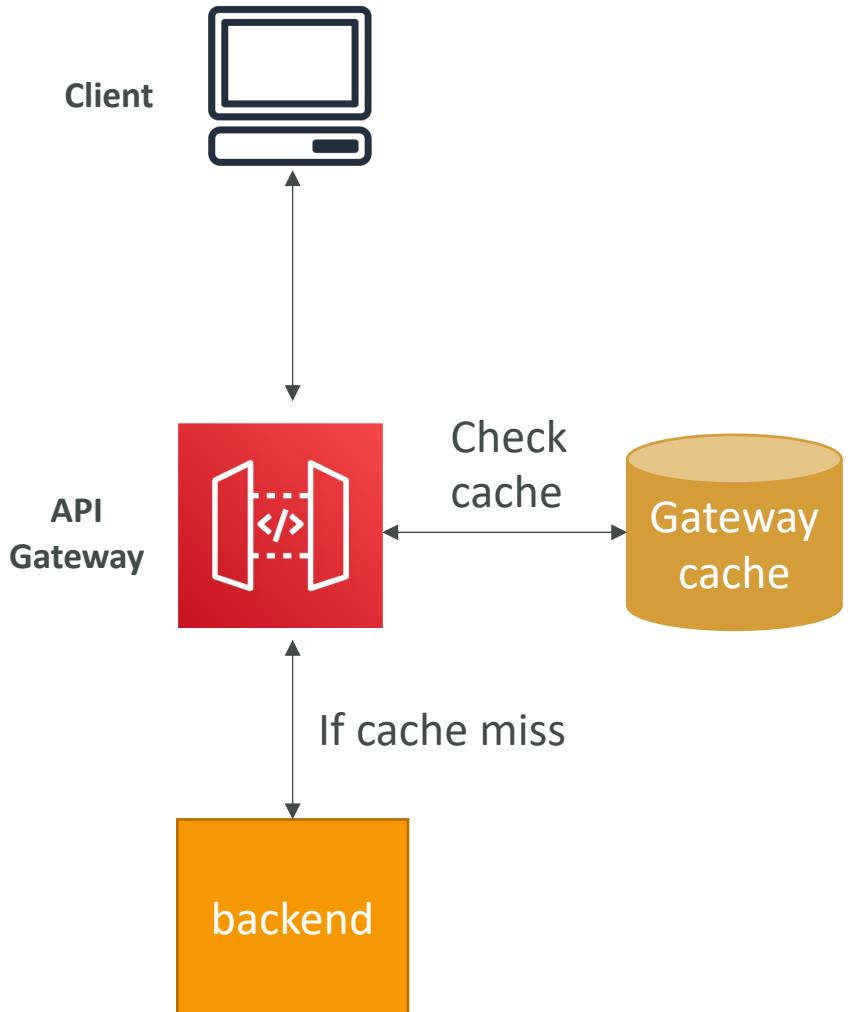
```
{  
    "openapi": "3.0.0",  
    "info": {  
        "title": "ReqValidation Sample",  
        "version": "1.0.0"  
    },  
    "servers": [ ... ],  
    "x-amazon-apigateway-request-validator": "params-only",  
    ...  
}  
  
{  
    "openapi": "3.0.0",  
    "info": {  
        "title": "ReqValidation Sample",  
        "version": "1.0.0"  
    },  
    "servers": [ ... ],  
    "paths": {  
        "/validation": {  
            "post": {  
                "x-amazon-apigateway-request-validator": "all"  
            }  
        }  
    },  
    ...  
}
```

Enable **params-only** Validator  
on all API methods

Enable **all** Validator on the  
**POST /validation** method  
(overrides the **params-only** validator  
inherited from the API)

# Caching API responses

- Caching reduces the number of calls made to the backend
- Default TTL (time to live) is 300 seconds (min: 0s, max: 3600s)
- Caches are defined per stage
- Possible to override cache settings per method
- Cache encryption option
- Cache capacity between 0.5GB to 237GB
- Cache is expensive, makes sense in production, may not make sense in dev / test



# API Gateway Cache Invalidation

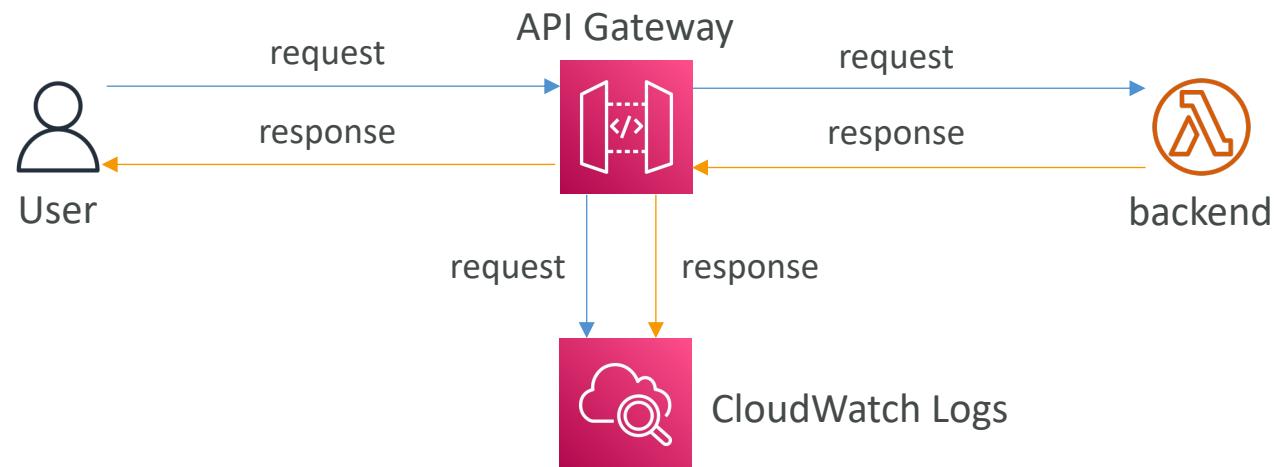
- Able to flush the entire cache (invalidate it) immediately
- Clients can invalidate the cache with **header: Cache-Control: max-age=0** (with proper IAM authorization)
- If you don't impose an InvalidateCache policy (or choose the Require authorization check box in the console), any client can invalidate the API cache

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": [  
        "execute-api:InvalidateCache"  
      ],  
      "Resource": [  
        "arn:...:api-id/stage-name/GET/resource-path-specifier"  
      ]  
    }  
  ]  
}
```

# API Gateway – Logging & Tracing

- **CloudWatch Logs**

- Log contains information about request/response body
- Enable CloudWatch logging at the Stage level (with Log Level - ERROR, DEBUG, INFO)
- Can override settings on a per API basis



- **X-Ray**

- Enable tracing to get extra information about requests in API Gateway
- X-Ray API Gateway + AWS Lambda gives you the full picture

# API Gateway – CloudWatch Metrics



- Metrics are by stage, Possibility to enable detailed metrics
- **CacheHitCount & CacheMissCount:** efficiency of the cache
- **Count:** The total number API requests in a given period.
- **IntegrationLatency:** The time between when API Gateway relays a request to the backend and when it receives a response from the backend.
- **Latency:** The time between when API Gateway receives a request from a client and when it returns a response to the client. The latency includes the integration latency and other API Gateway overhead.
- **4XXError** (client-side) & **5XXError** (server-side)

# API Gateway Throttling

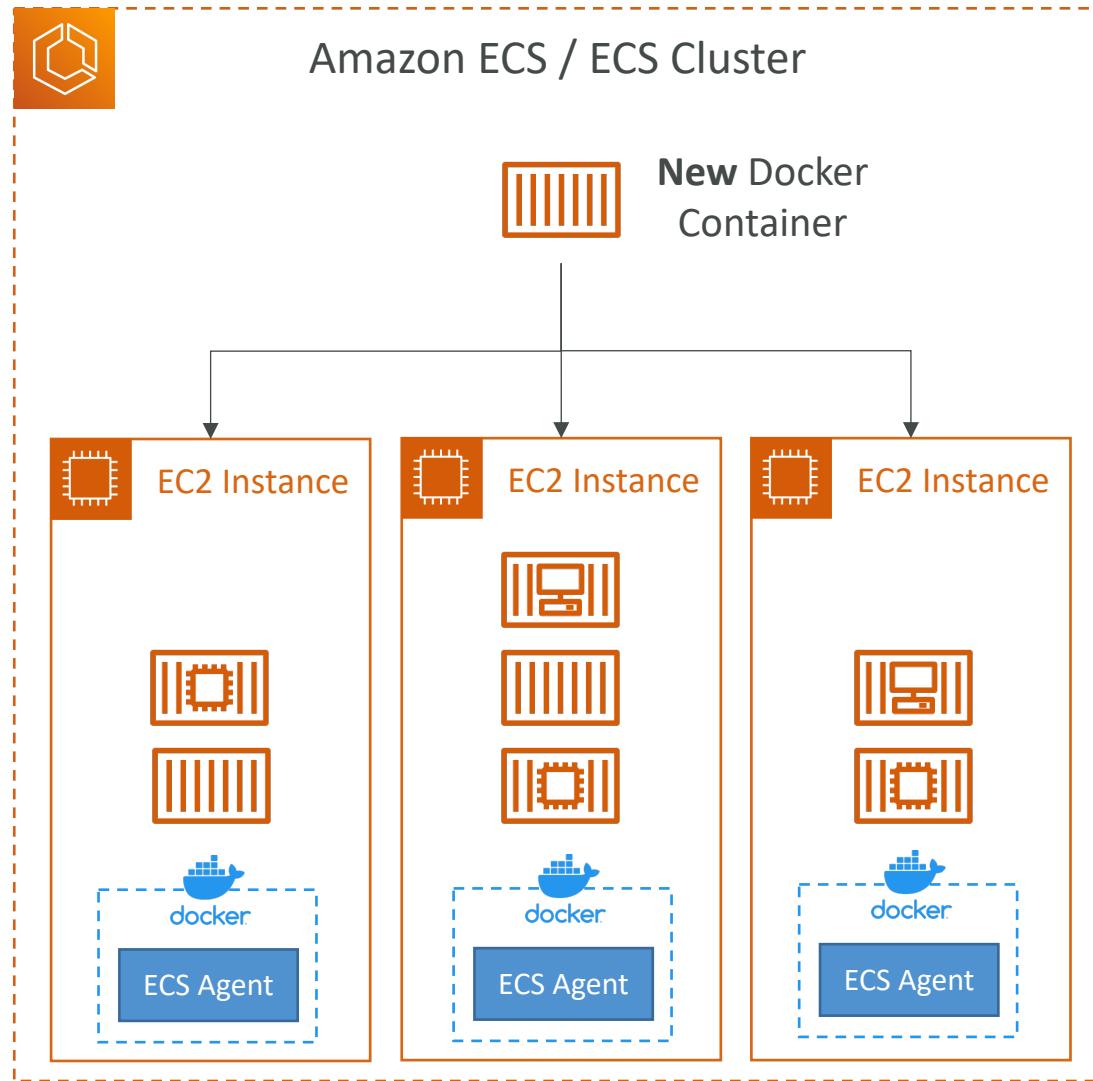
- Account Limit
    - API Gateway throttles requests at 10000 rps across all API
    - Soft limit that can be increased upon request
  - In case of throttling => 429 Too Many Requests (retryable error)
  - Can set Stage limit & Method limits to improve performance
  - Or you can define Usage Plans to throttle per customer
- 
- Just like Lambda Concurrency, one API that is overloaded, if not limited, can cause the other APIs to be throttled

# API Gateway - Errors

- 4xx means Client errors
  - 400: Bad Request
  - 403: Access Denied, WAF filtered
  - 429: Quota exceeded, Throttle
- 5xx means Server errors
  - 502: Bad Gateway Exception, usually for an incompatible output returned from a Lambda proxy integration backend and occasionally for out-of-order invocations due to heavy loads.
  - 503: Service Unavailable Exception
  - 504: Integration Failure – ex Endpoint Request Timed-out Exception  
**API Gateway requests time out after 29 second maximum**

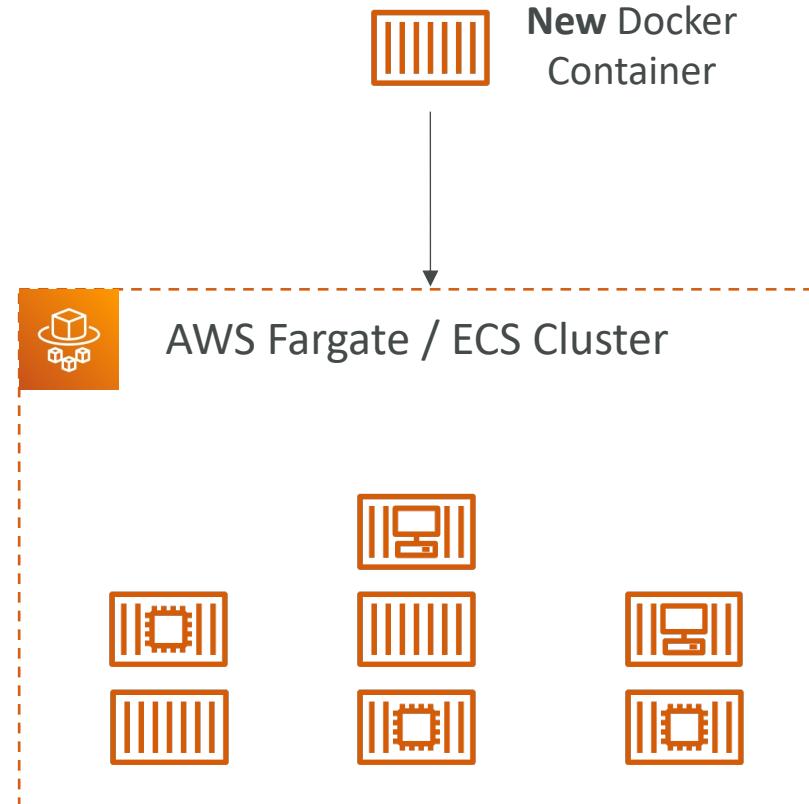
# Amazon ECS - EC2 Launch Type

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



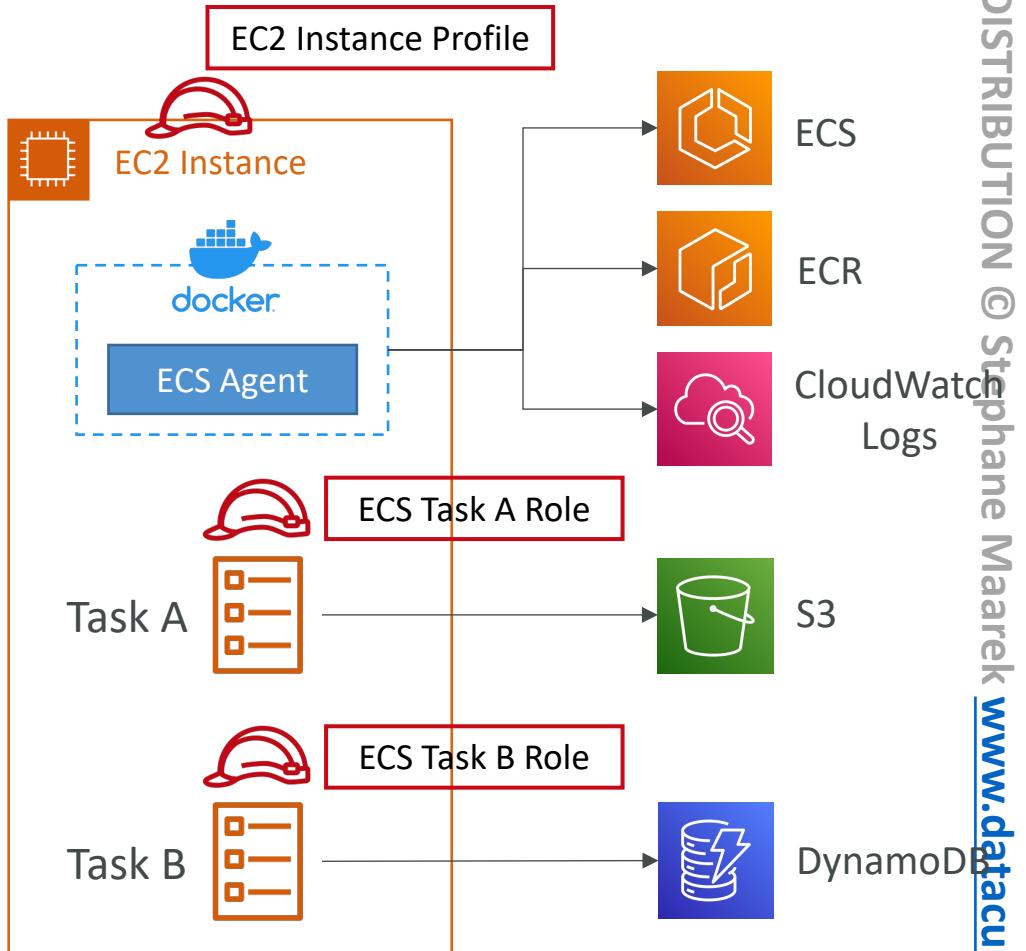
# Amazon ECS – Fargate Launch Type

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



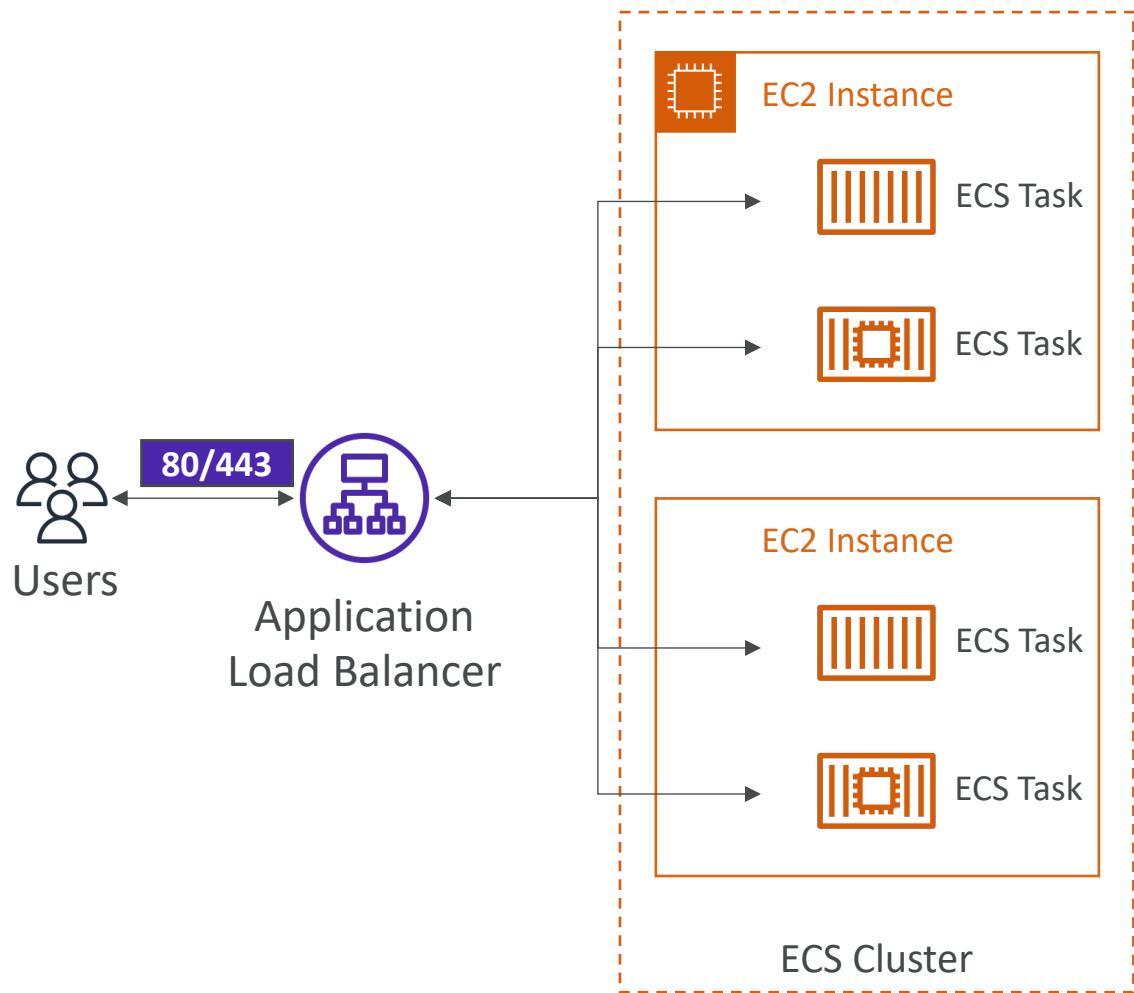
# Amazon ECS – IAM Roles for ECS

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



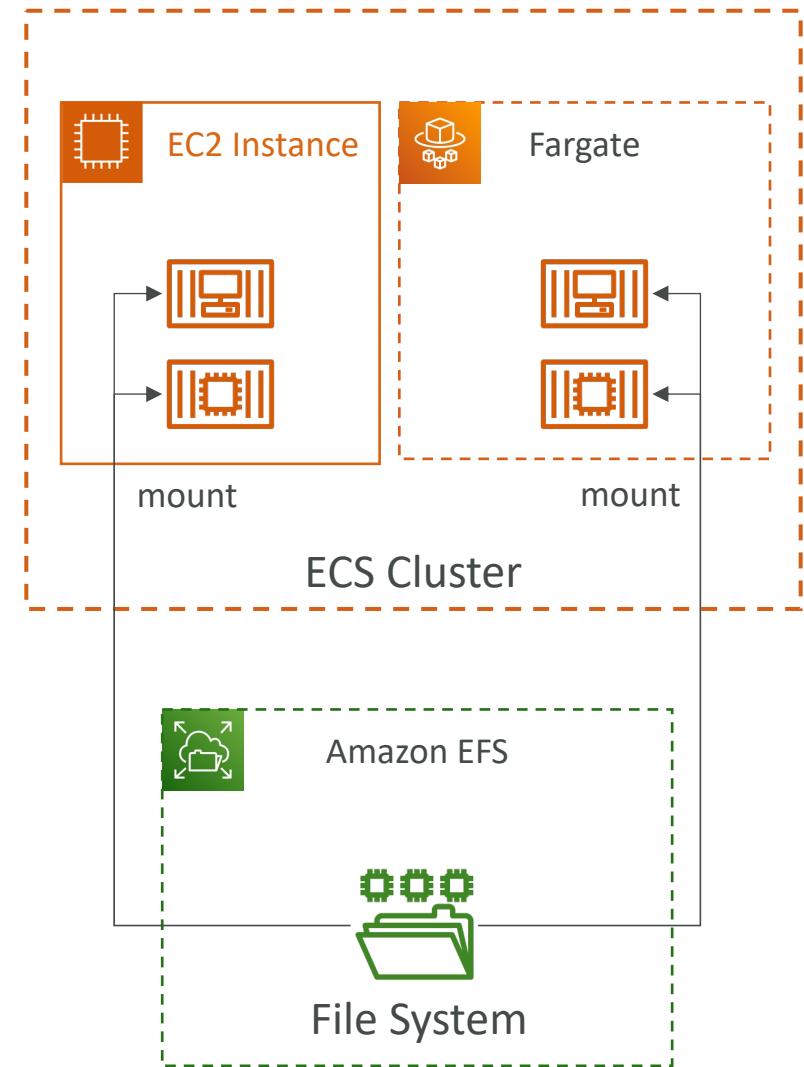
# Amazon ECS – Load Balancer Integrations

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

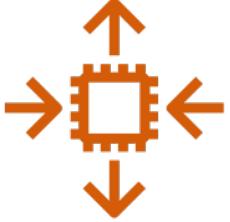


# Amazon ECS – Data Volumes (EFS)

- Mount EFS file systems onto ECS tasks
- Works for both **EC2** and **Fargate** launch types
- Tasks running in any AZ will share the same data in the EFS file system
- **Fargate + EFS = Serverless**
- Use cases: persistent multi-AZ shared storage for your containers
- Note:
  - Amazon S3 cannot be mounted as a file system



# ECS Service Auto Scaling

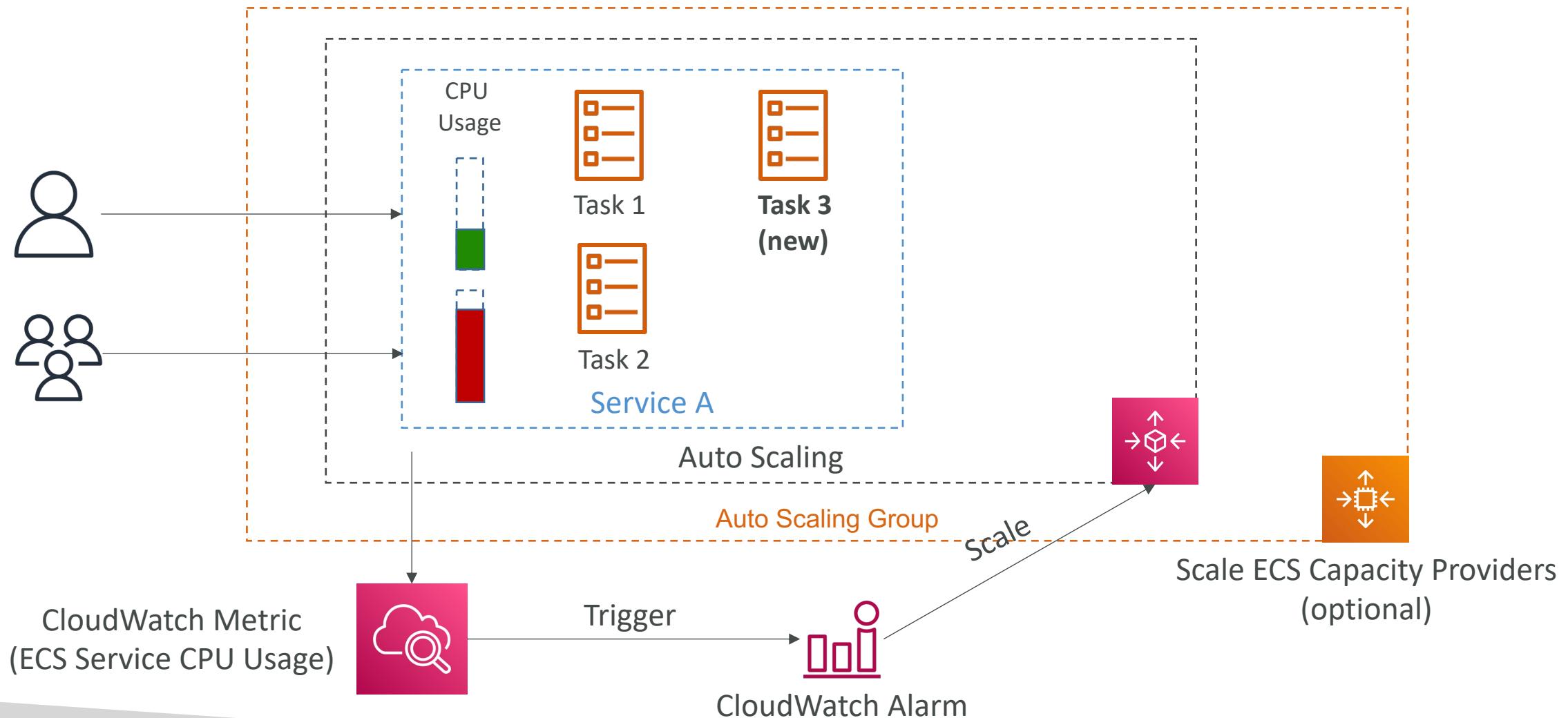


- Automatically increase/decrease the desired number of ECS tasks
- Amazon ECS Auto Scaling uses **AWS Application Auto Scaling**
  - ECS Service Average CPU Utilization
  - ECS Service Average Memory Utilization - Scale on RAM
  - ALB Request Count Per Target – metric coming from the ALB
- **Target Tracking** – scale based on target value for a specific CloudWatch metric
- **Step Scaling** – scale based on a specified CloudWatch Alarm
- **Scheduled Scaling** – scale based on a specified date/time (predictable changes)
- ECS Service Auto Scaling (task level) **≠** EC2 Auto Scaling (EC2 instance level)
- Fargate Auto Scaling is much easier to setup (because **Serverless**)

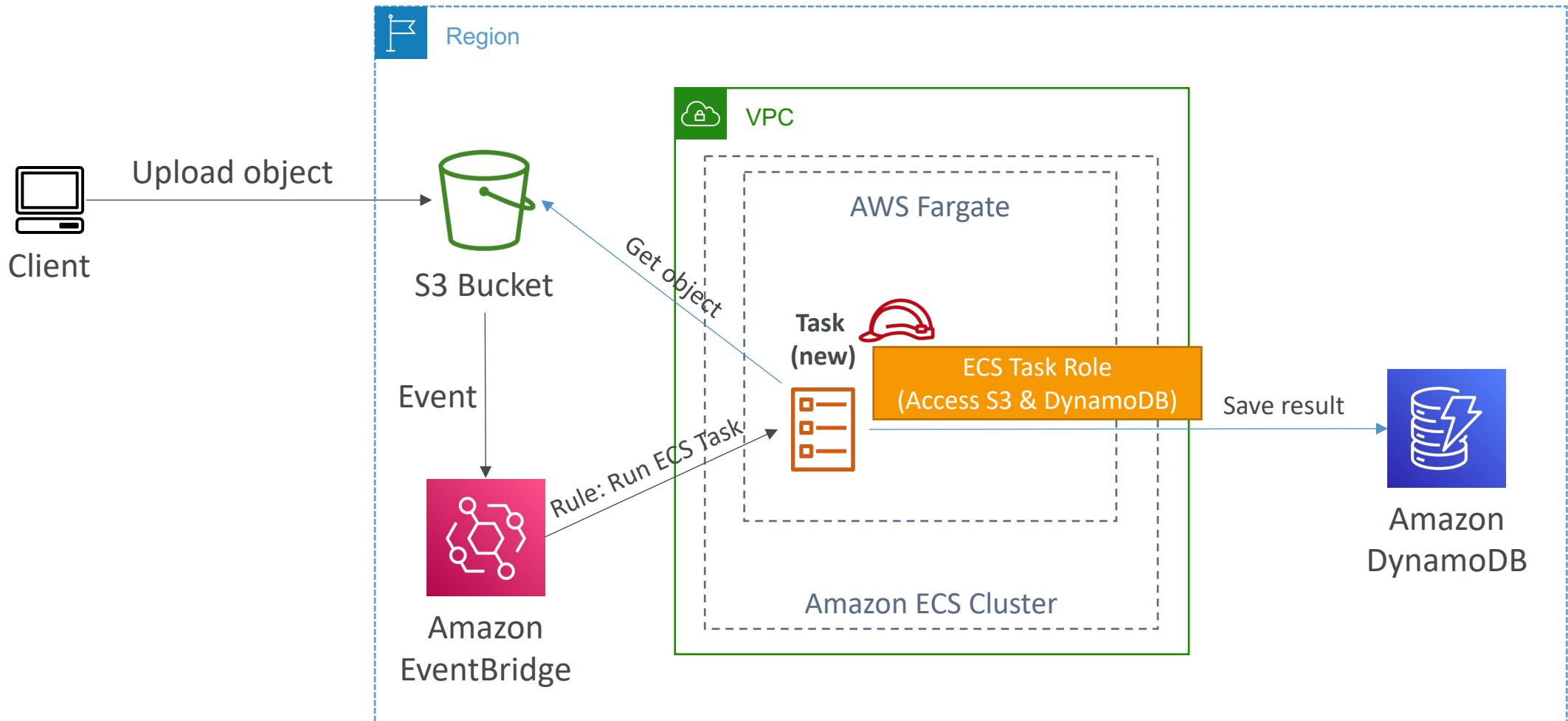
# EC2 Launch Type – Auto Scaling EC2 Instances

- Accommodate ECS Service Scaling by adding underlying EC2 Instances
- Auto Scaling Group Scaling
  - Scale your ASG based on CPU Utilization
  - Add EC2 instances over time
- ECS Cluster Capacity Provider
  - Used to automatically provision and scale the infrastructure for your ECS Tasks
  - Capacity Provider paired with an Auto Scaling Group
  - Add EC2 Instances when you're missing capacity (CPU, RAM...)

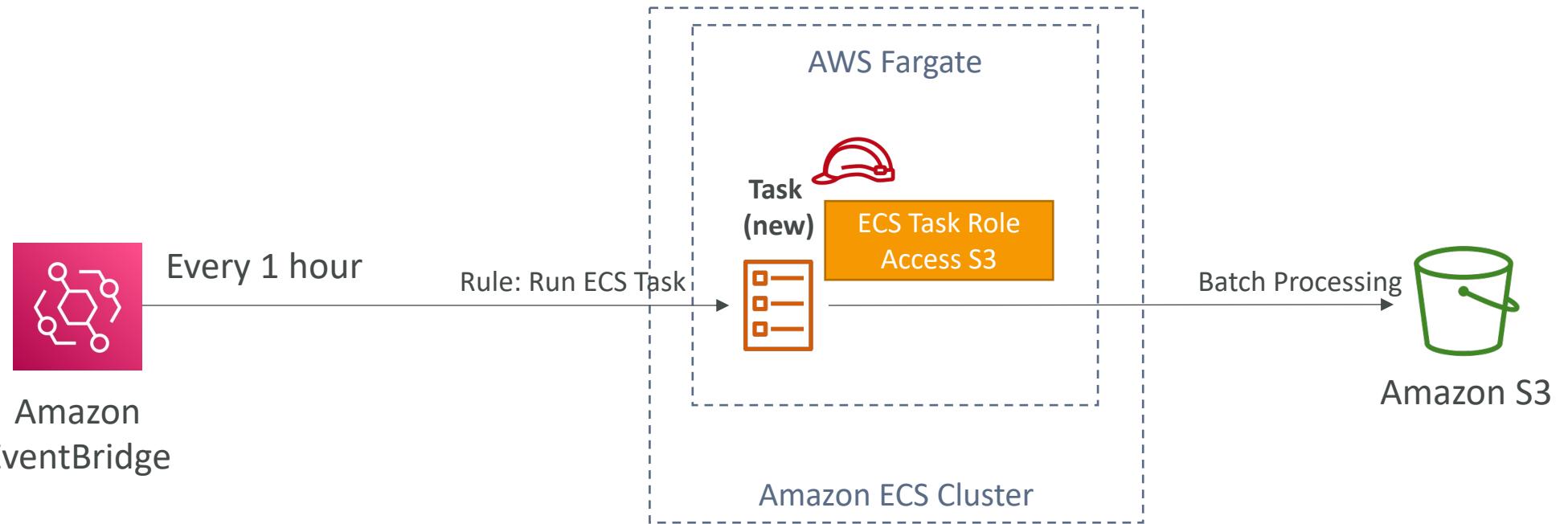
# ECS Scaling – Service CPU Usage Example



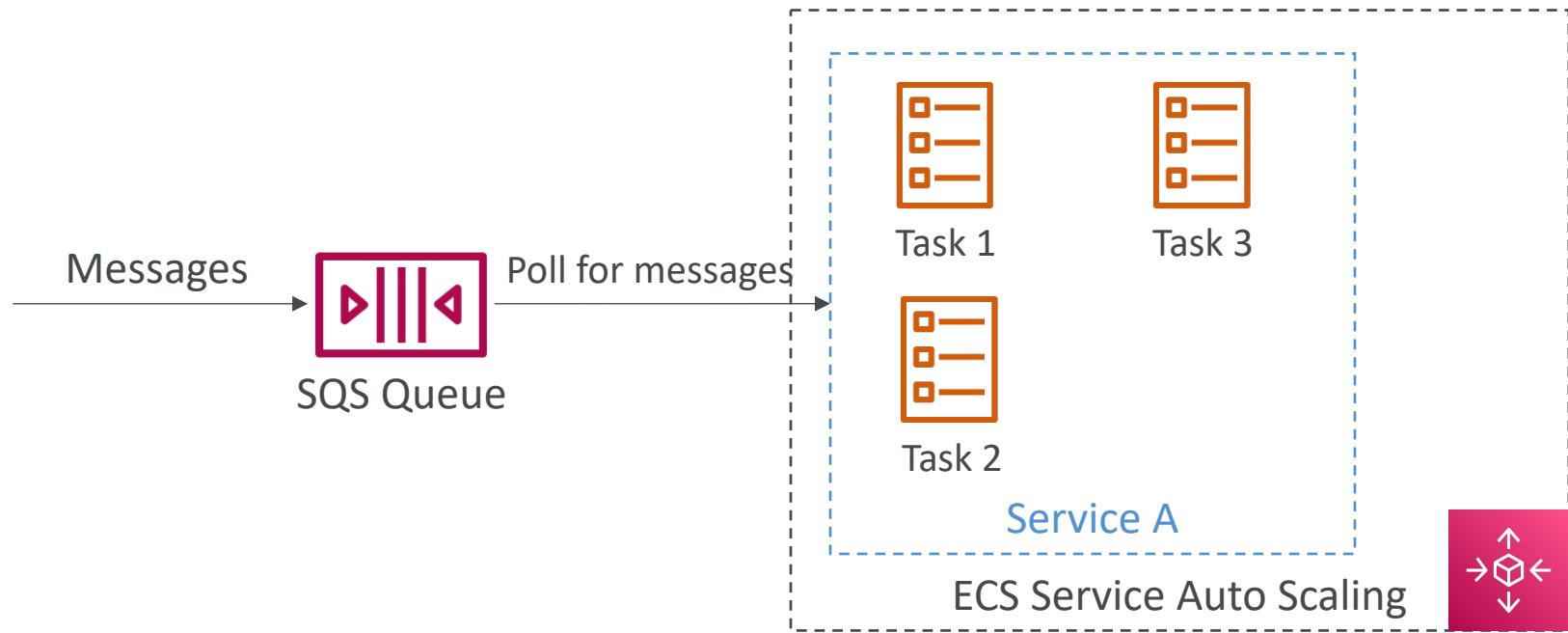
# ECS tasks invoked by Event Bridge



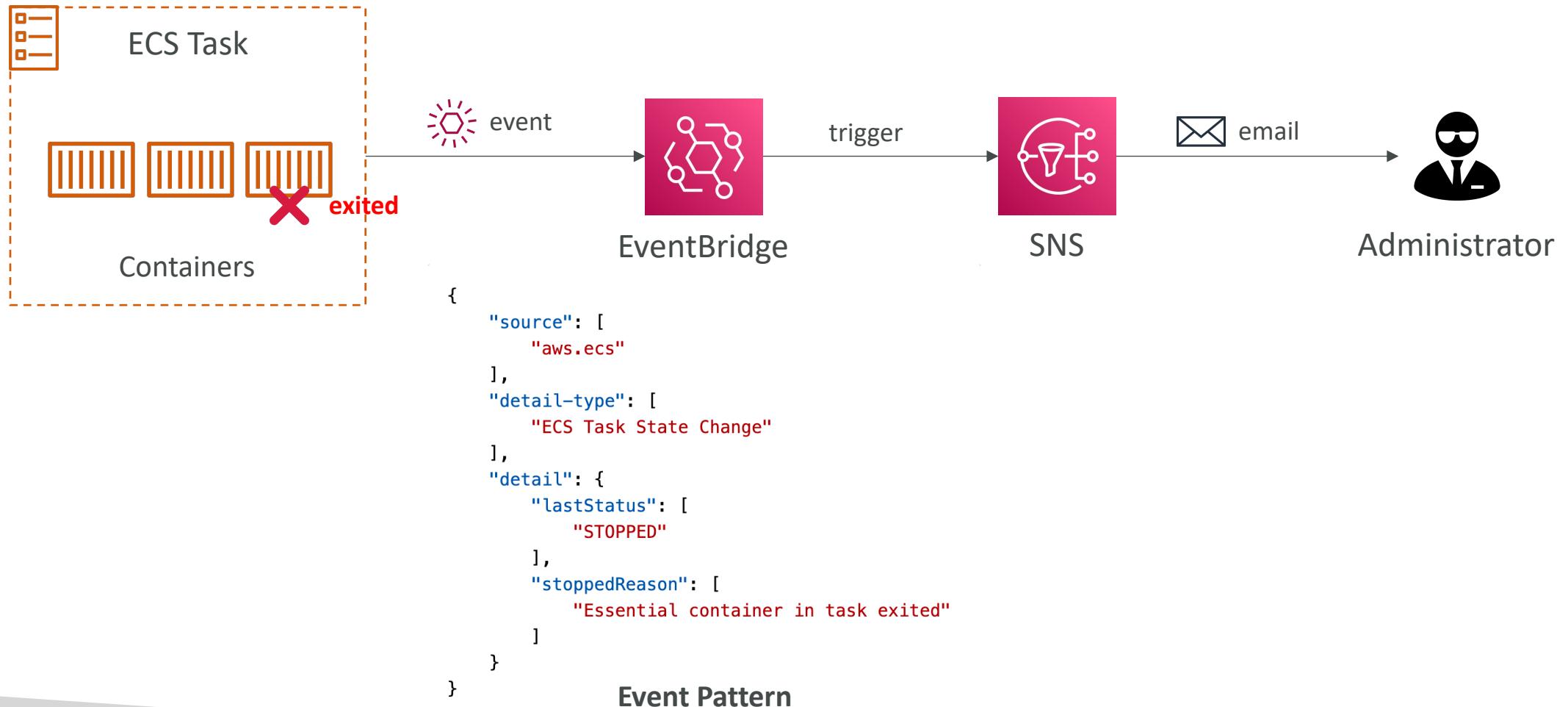
# ECS tasks invoked by Event Bridge Schedule



# ECS – SQS Queue Example

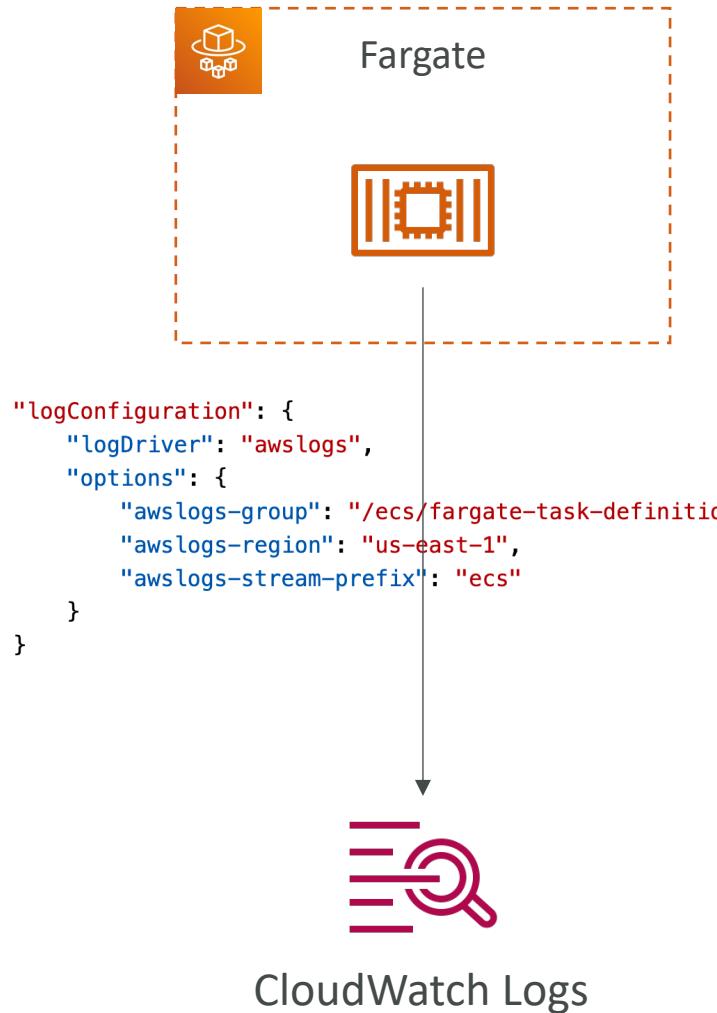


# ECS – Intercept Stopped Tasks using EventBridge



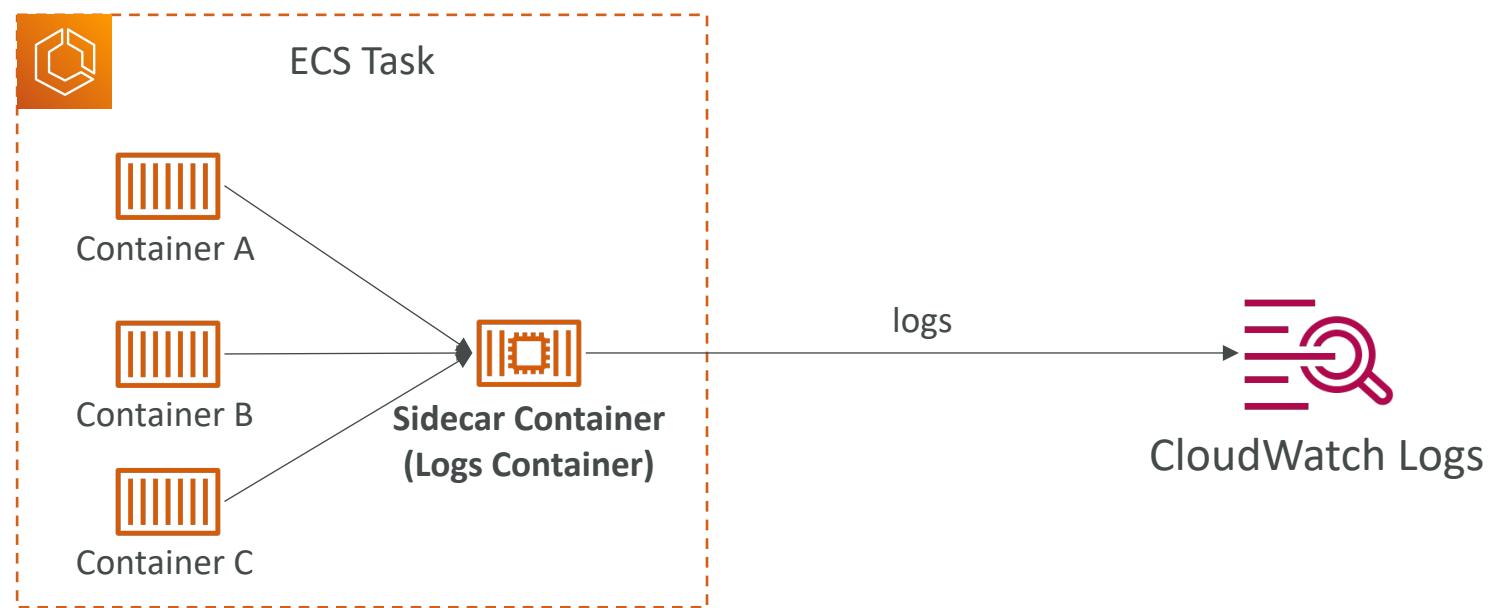
# ECS – Logging with “awslogs” driver

- Containers can send application logs directly to CloudWatch Logs
- You need to turn on **awslogs** log driver (for CW Logs)
- Configure **logConfiguration** parameters in your Task Definition
- **Fargate Launch Type**
  - Task Execution Role must have the required permissions
  - Supports **awslogs**, **splunk**, **awsfirelens** log drivers
- **EC2 Launch Type**
  - Prevents logs from taking up disk space on your container EC2 instances
  - Uses **CloudWatch Unified Agent** & **ECS Container Agent**
  - Enable logging using **ECS\_AVAILABLE\_LOGGING\_DRIVERS** in **/etc/ecs/ecs.config**
  - Container EC2 instances must have permissions



# ECS – Logging with Sidecar Container

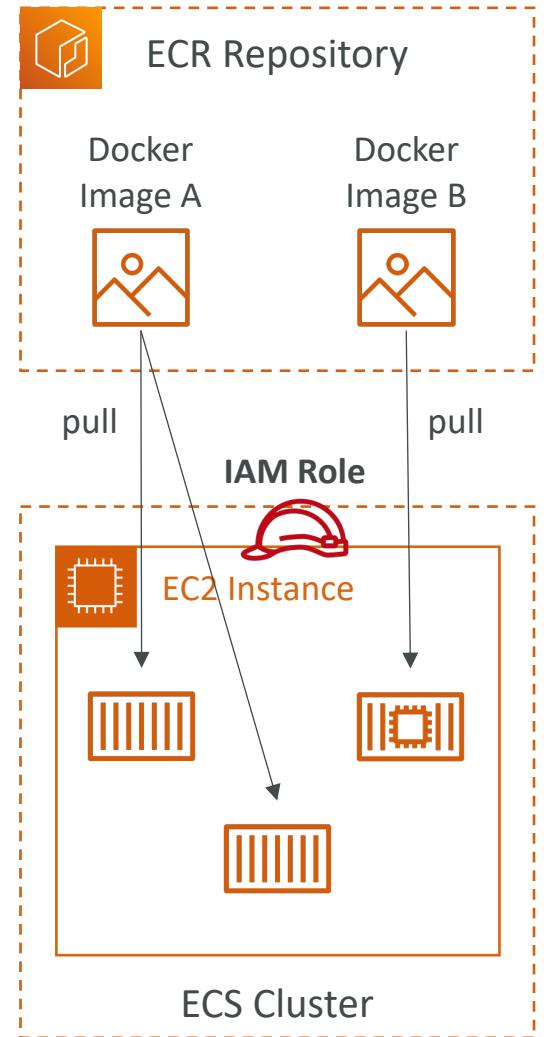
- Using a sidecar container which is responsible for collecting logs from all other containers and files on the file system and send the logs to a log storage (e.g., CloudWatch Logs)





# Amazon ECR

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



# Amazon ECR – Lifecycle Policies

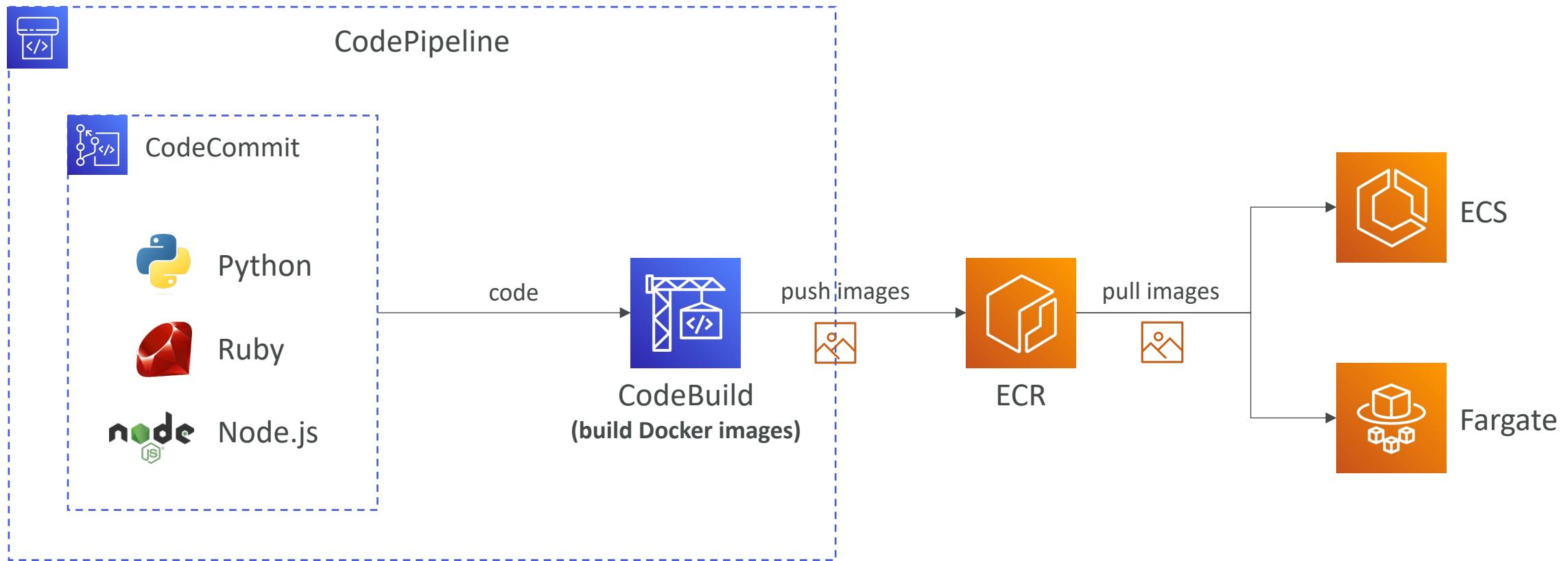
- Automatically remove old or unused images based on age or count
- Each Lifecycle Policy contains one or more rules
- All rules are evaluated at the same time, then applied based on priority
- Images are expired within 24 hours after they meet the criteria
- Helps you reduce unnecessary storage costs

**expire untagged images older than 14 days**

```
{  
  "rules": [  
    {  
      "rulePriority": 1,  
      "description": "Expire images older than 14 days",  
      "selection": {  
        "tagStatus": "untagged",  
        "countType": "sinceImagePushed",  
        "countUnit": "days",  
        "countNumber": 14  
      },  
      "action": {  
        "type": "expire"  
      }  
    }  
  ]  
}  
  
{  
  "rules": [  
    {  
      "rulePriority": 1,  
      "description": "Keep only one untagged image, expire all others",  
      "selection": {  
        "tagStatus": "untagged",  
        "countType": "imageCountMoreThan",  
        "countNumber": 1  
      },  
      "action": {  
        "type": "expire"  
      }  
    }  
  ]  
}
```

**keep only one untagged image and expires all others**

# Amazon ECR – Uniform Pipeline

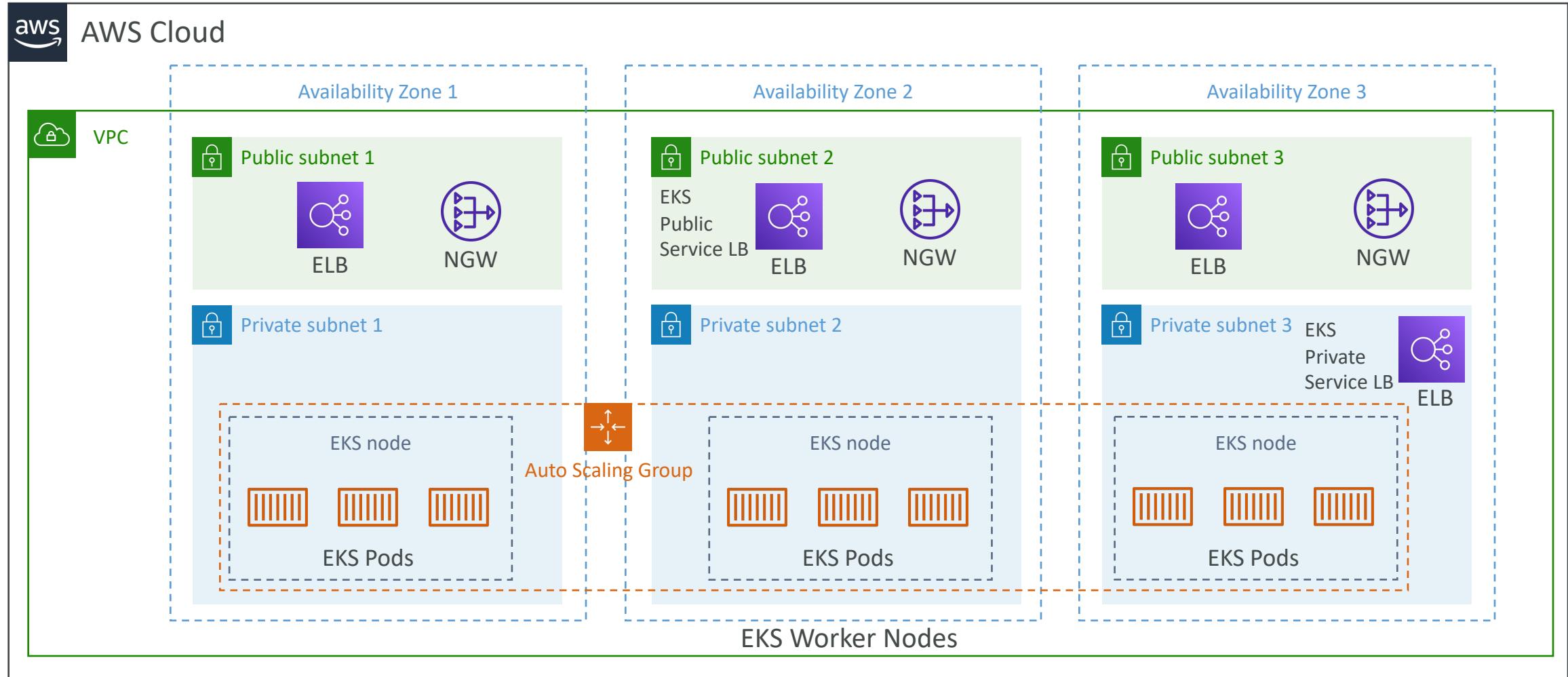


# Amazon EKS Overview



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

# Amazon EKS - Diagram

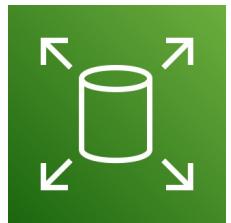


# Amazon EKS – Node Types

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

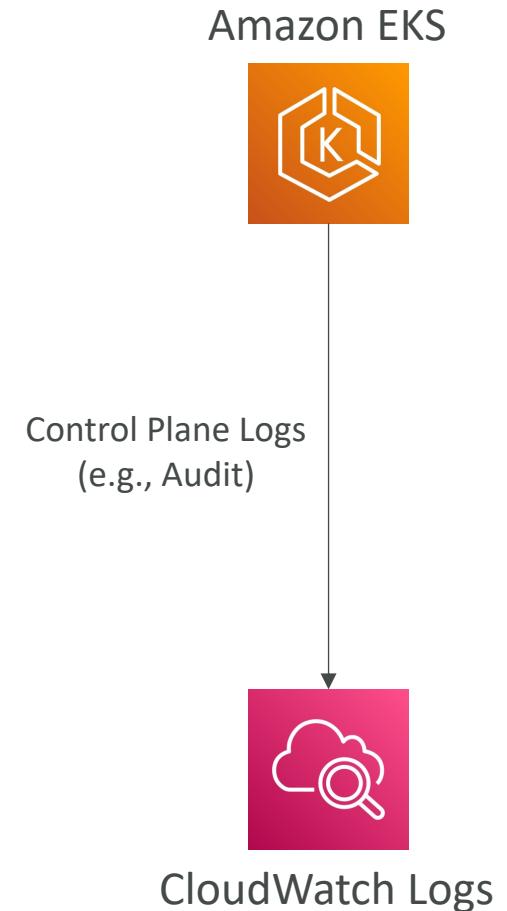
# Amazon EKS – Data Volumes

- Need to specify **StorageClass** manifest on your EKS cluster
- Leverages a **Container Storage Interface (CSI)** compliant driver
- Support for...
- Amazon EBS
- Amazon EFS (works with Fargate)
- Amazon FSx for Lustre
- Amazon FSx for NetApp ONTAP



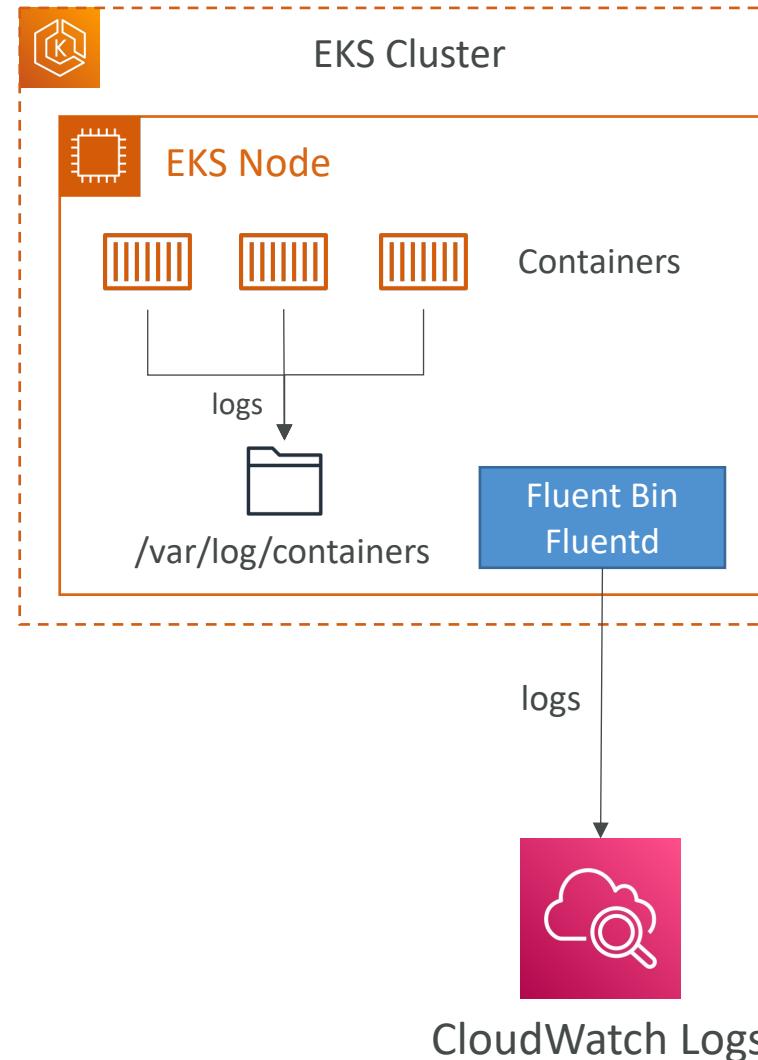
# Amazon EKS – Control Plane Logging

- Send EKS Control Plane audit and diagnostic logs to CloudWatch Logs
- EKS Control Plane Log Types
  - API Server (api)
  - Audit (audit)
  - Authenticator (authenticator)
  - Controller Manager (controllerManager)
  - Scheduler (scheduler)
- Ability to select the exact log types to send to CloudWatch Logs

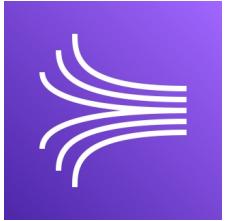


# Amazon EKS – Nodes & Containers Logging

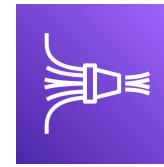
- You can capture node, pod, and containers logs and send them to CloudWatch Logs
- Use **CloudWatch Agent** to send metrics to CloudWatch
- Use the **Fluent Bit**, or **Fluentd** log drivers to send logs to CloudWatch Logs
- Container logs are stored on a Node directory `/var/log/containers`
- Use **CloudWatch Container Insights** to get a dashboarding monitoring solution for nodes, pods, tasks, and services



# Kinesis Overview

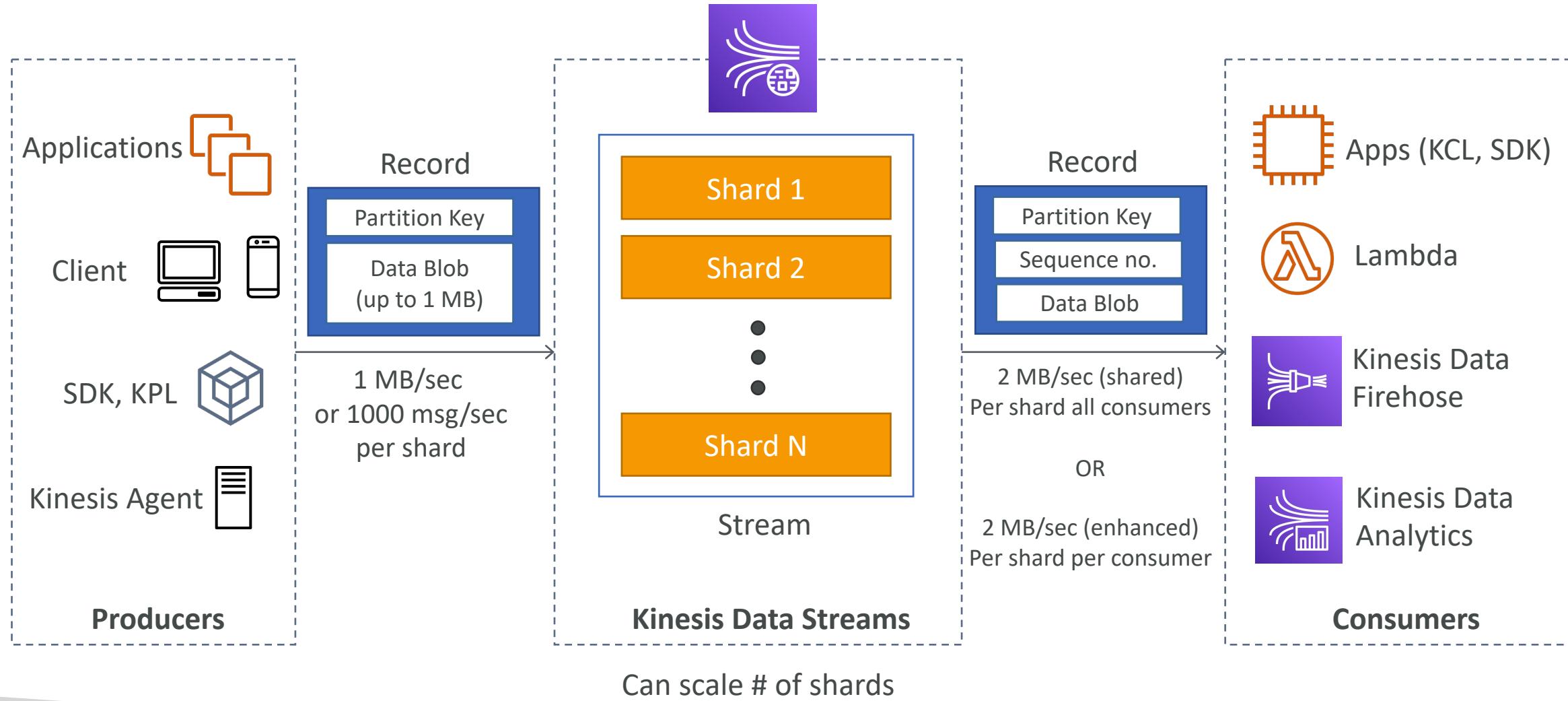


- Makes it easy to **collect, process, and analyze** streaming data in real-time
- Ingest real-time data such as: Application logs, Metrics, Website clickstreams, IoT telemetry data...



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

# Kinesis Data Streams





# Kinesis Data Streams

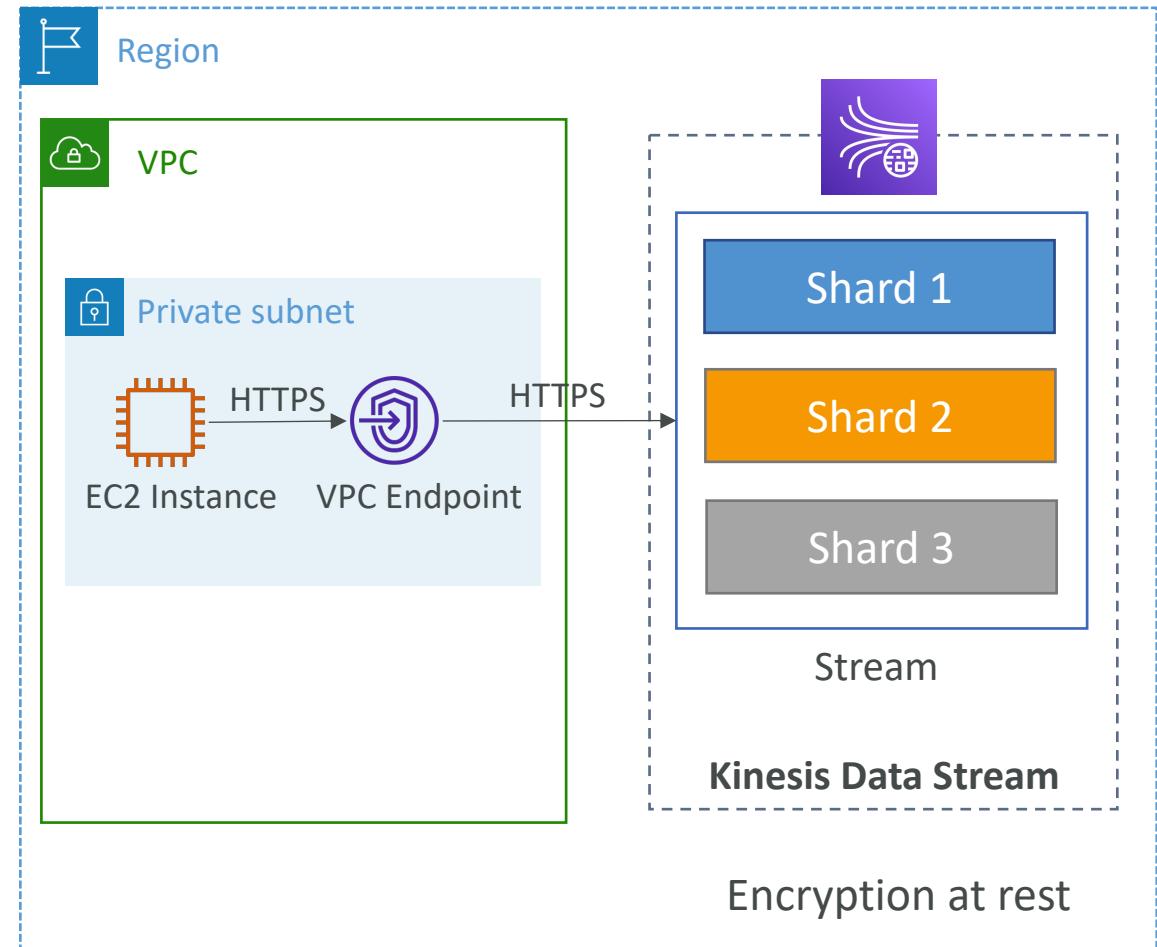
- Retention between 1 day to 365 days
- Ability to reprocess (replay) data
- Once data is inserted in Kinesis, it can't be deleted (immutability)
- Data that shares the same partition goes to the same shard (ordering)
- Producers: AWS SDK, Kinesis Producer Library (KPL), Kinesis Agent
- Consumers:
 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

# Kinesis Data Streams – Capacity Modes

- **Provisioned mode:**
  - You choose the number of shards provisioned, scale manually or using API
  - Each shard gets 1MB/s in (or 1000 records per second)
  - Each shard gets 2MB/s out (classic or enhanced fan-out consumer)
  - You pay per shard provisioned per hour
- **On-demand mode:**
  - No need to provision or manage the capacity
  - Default capacity provisioned (4 MB/s in or 4000 records per second)
  - Scales automatically based on observed throughput peak during the last 30 days
  - Pay per stream per hour & data in/out per GB

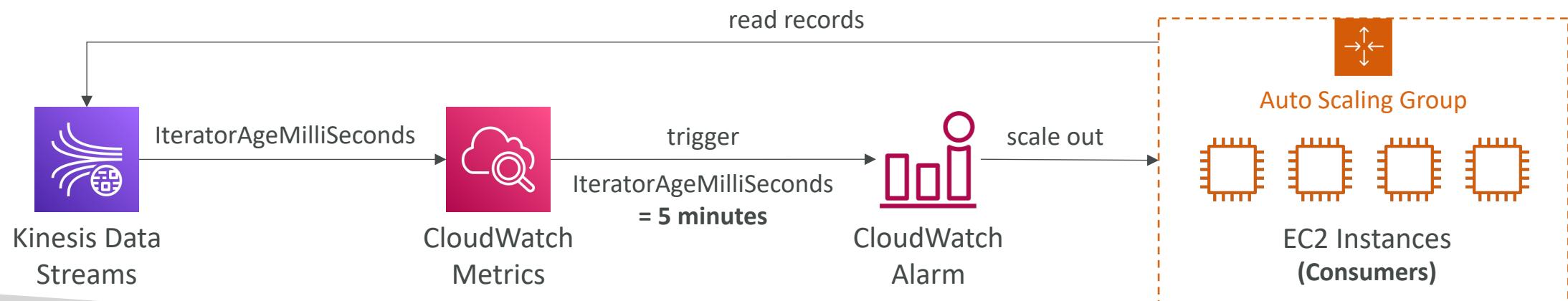
# Kinesis Data Streams Security

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

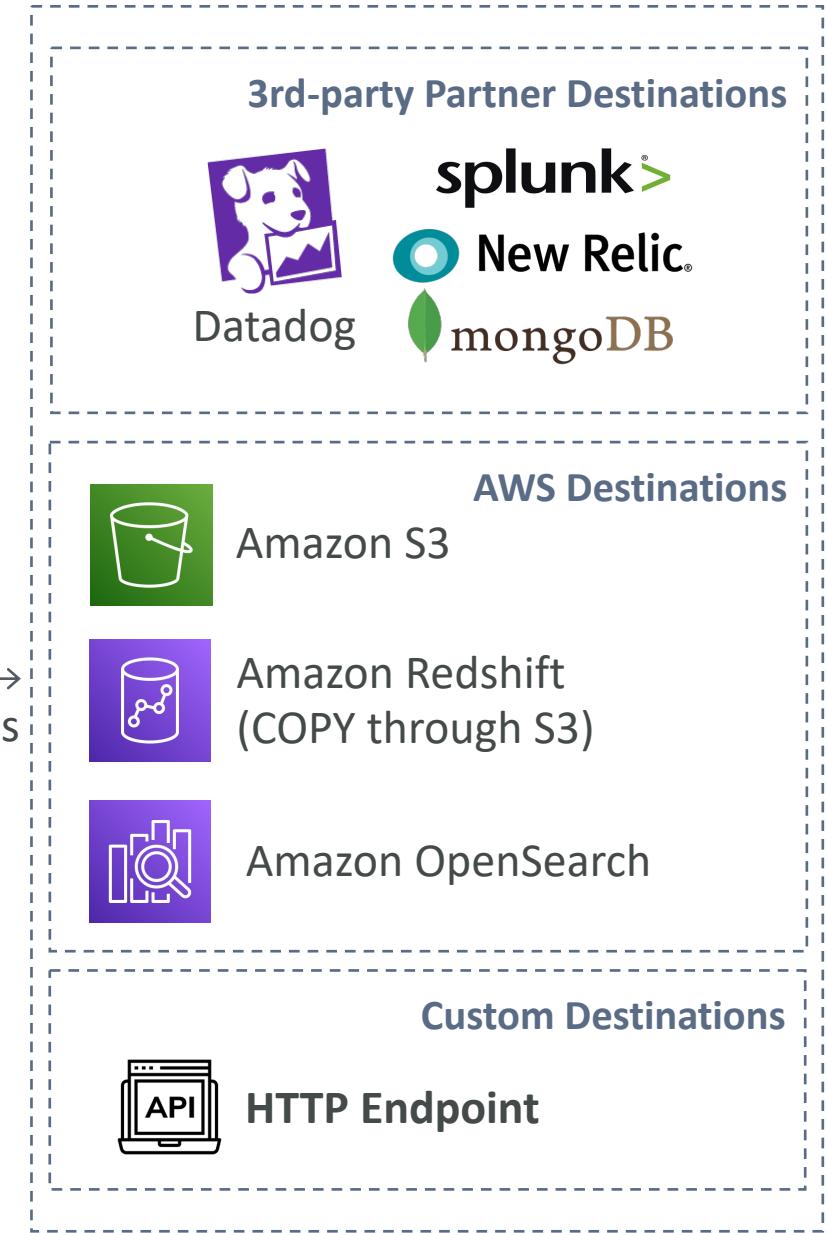
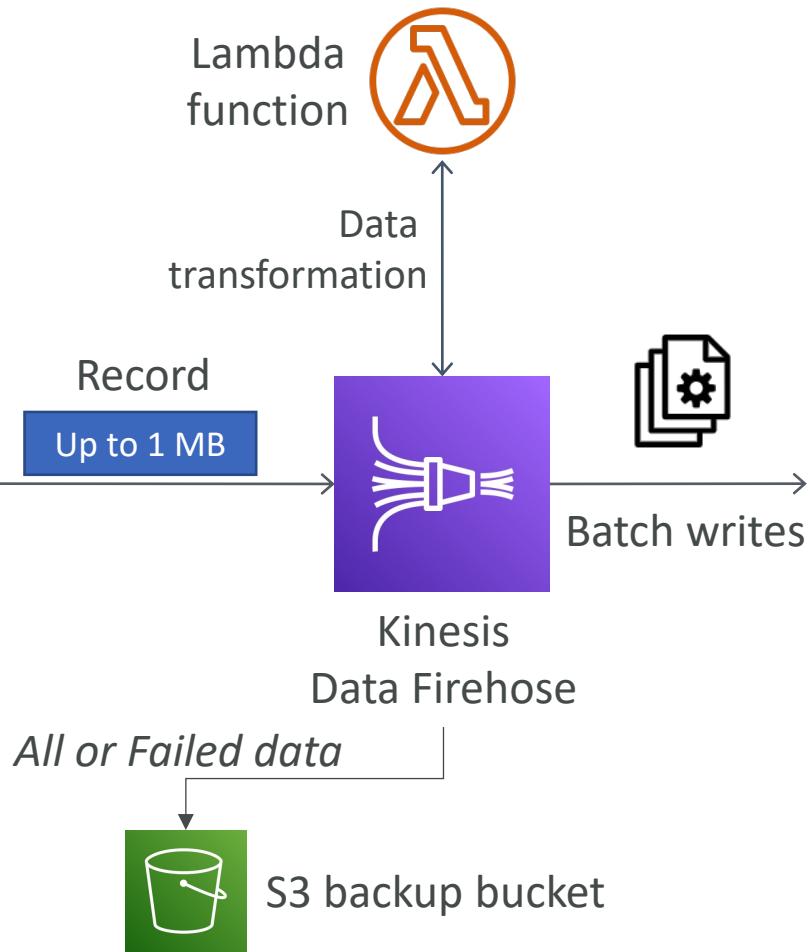
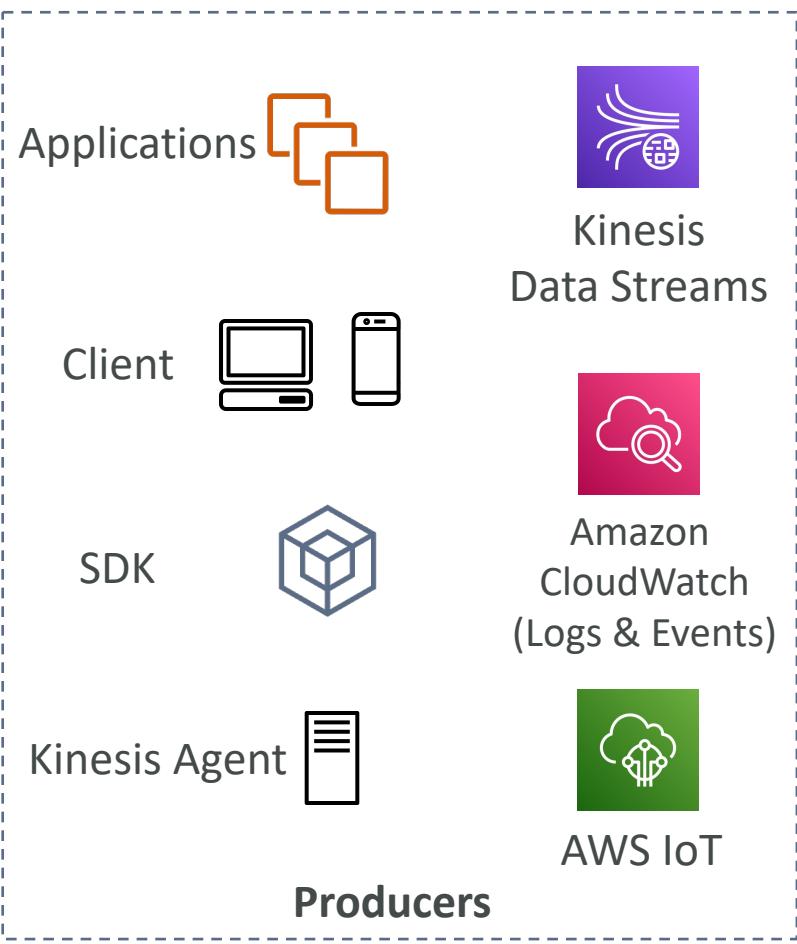


# Kinesis Data Streams – Scaling Consumers

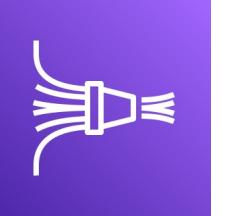
- `GetRecords.IteratorAgeMilliseconds` (CloudWatch Metric)
  - The difference between current time and when the last record of the `GetRecords` call was written to the stream
  - Used to track the progress of Kinesis Consumers (tracks the read position)
  - `IteratorAgeMilliseconds` = 0, then records being read are completely caught up with the Stream
  - `IteratorAgeMilliseconds > 0` means we're not processing the records fast enough



# Kinesis Data Firehose



# Kinesis Data Firehose



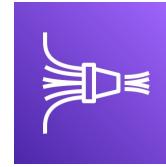
- Fully Managed Service, no administration, automatic scaling, serverless
  - AWS: Redshift / Amazon S3 / OpenSearch
  - 3rd party partner: Splunk / MongoDB / DataDog / NewRelic / ...
  - Custom: send to any HTTP endpoint
- Pay for data going through Firehose
- **Near Real Time**
  - Buffer interval: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

# Kinesis Data Streams vs Firehose



## Kinesis Data Streams

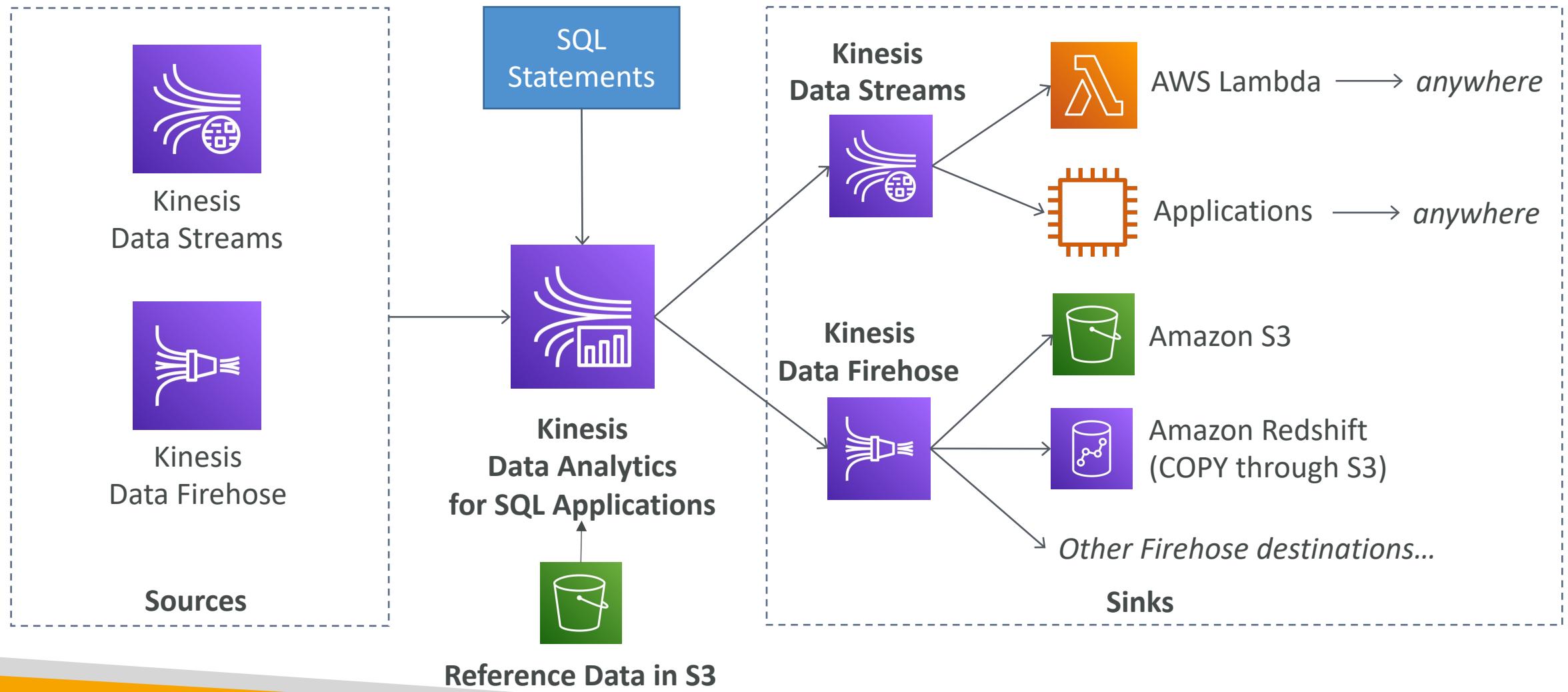
- Streaming service for ingest at scale
- Write custom code (producer / consumer)
- Real-time (~200 ms)
- Manage scaling (shard splitting / merging)
- Data storage for 1 to 365 days
- Supports replay capability



## Kinesis Data Firehose

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

# Kinesis Data Analytics for SQL applications





# Kinesis Data Analytics (SQL application)

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

# Kinesis Data Analytics for Apache Flink

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

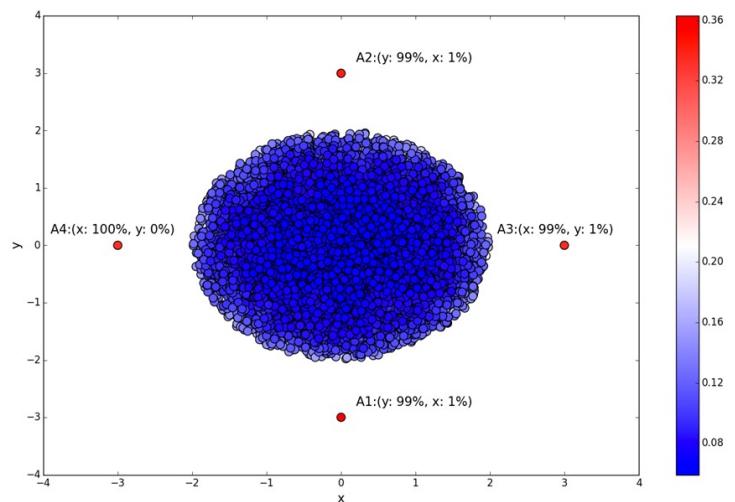


- Run any Apache Flink application on a managed cluster on AWS
  - provisioning compute resources, parallel computation, automatic scaling
  - application backups (implemented as checkpoints and snapshots)
  - Use any Apache Flink programming features
  - Flink does not read from Firehose (use Kinesis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

# Machine Learning on Kinesis Data Analytics

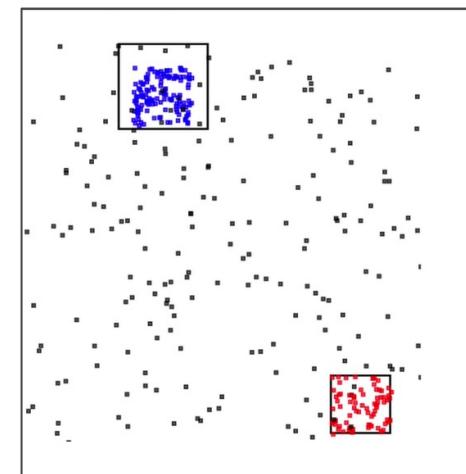
- **RANDOM\_CUT\_FOREST**

- SQL function used for **anomaly detection** on numeric columns in a stream
- Example: detect anomalous subway ridership during the NYC marathon
- Uses **recent history** to compute model



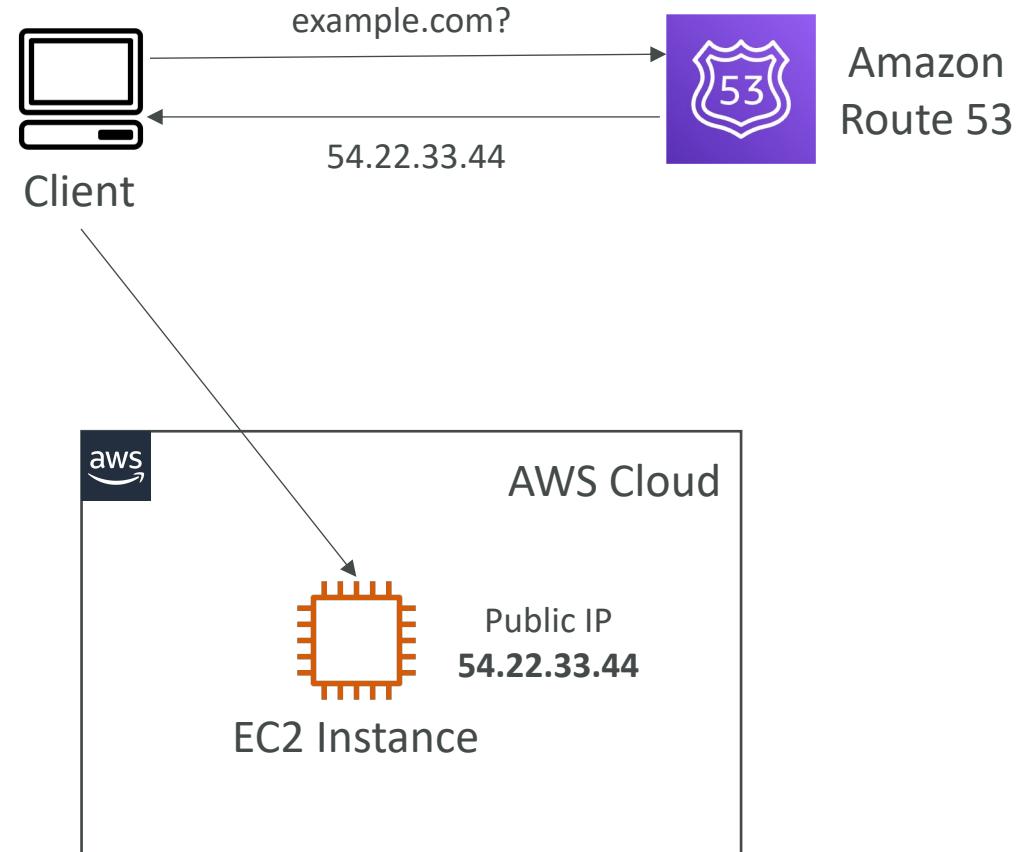
- **HOTSPOTS**

- locate and return information about relatively dense regions in your data
- Example: a collection of overheated servers in a data center



# Amazon Route 53

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



# Route 53 – Records

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

# Route 53 – Record Types

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

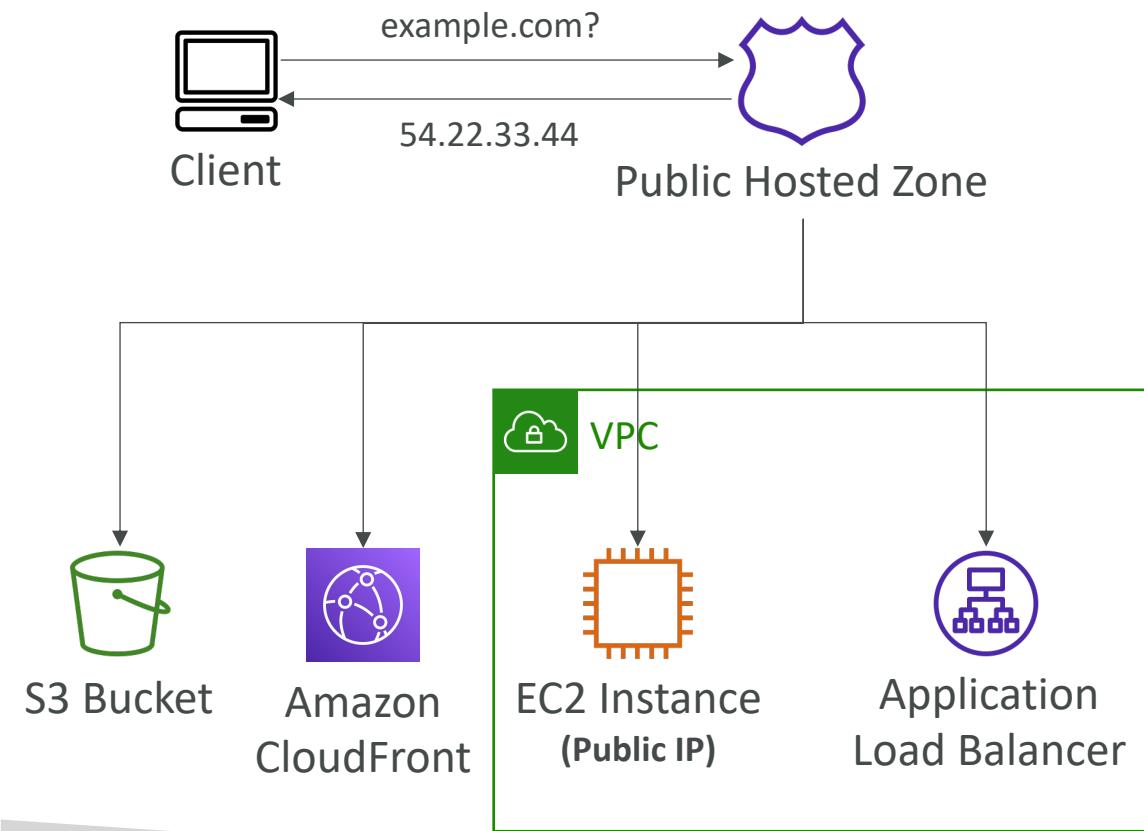


# Route 53 – Hosted Zones

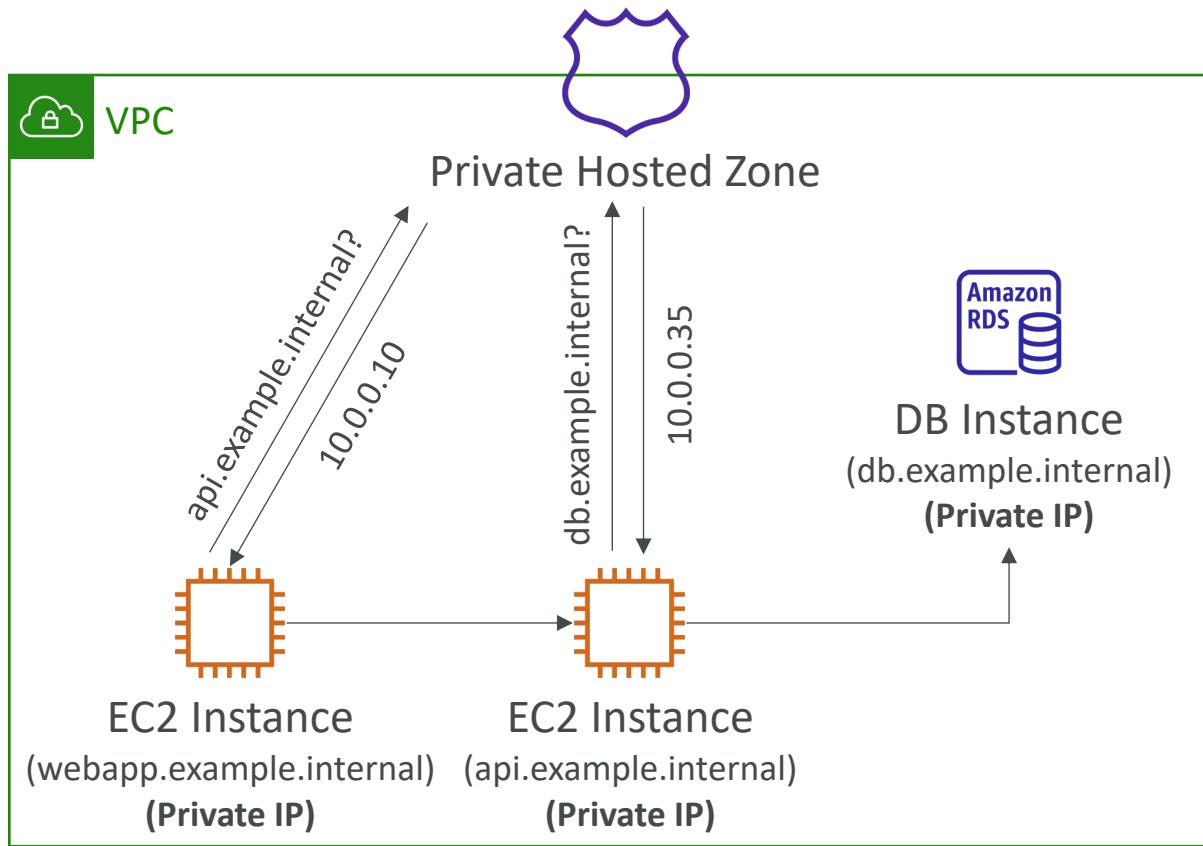
- A container for records that define how to route traffic to a domain and its subdomains
- **Public Hosted Zones** – contains records that specify how to route traffic on the Internet (public domain names)  
[application1.mypublicdomain.com](http://application1.mypublicdomain.com)
- **Private Hosted Zones** – contain records that specify how you route traffic within one or more VPCs (private domain names)  
[application1.company.internal](http://application1.company.internal)
- You pay \$0.50 per month per hosted zone

# Route 53 – Public vs. Private Hosted Zones

## Public Hosted Zone

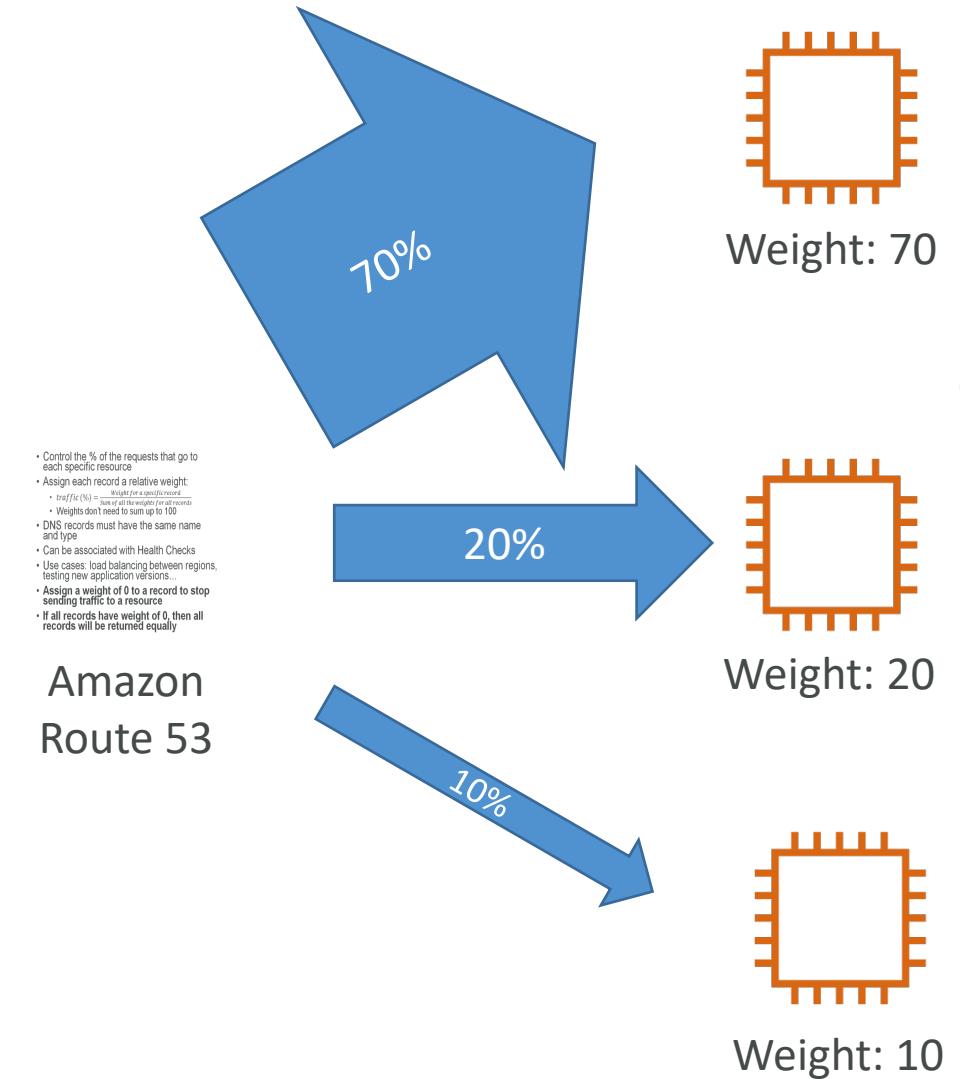


## Private Hosted Zone



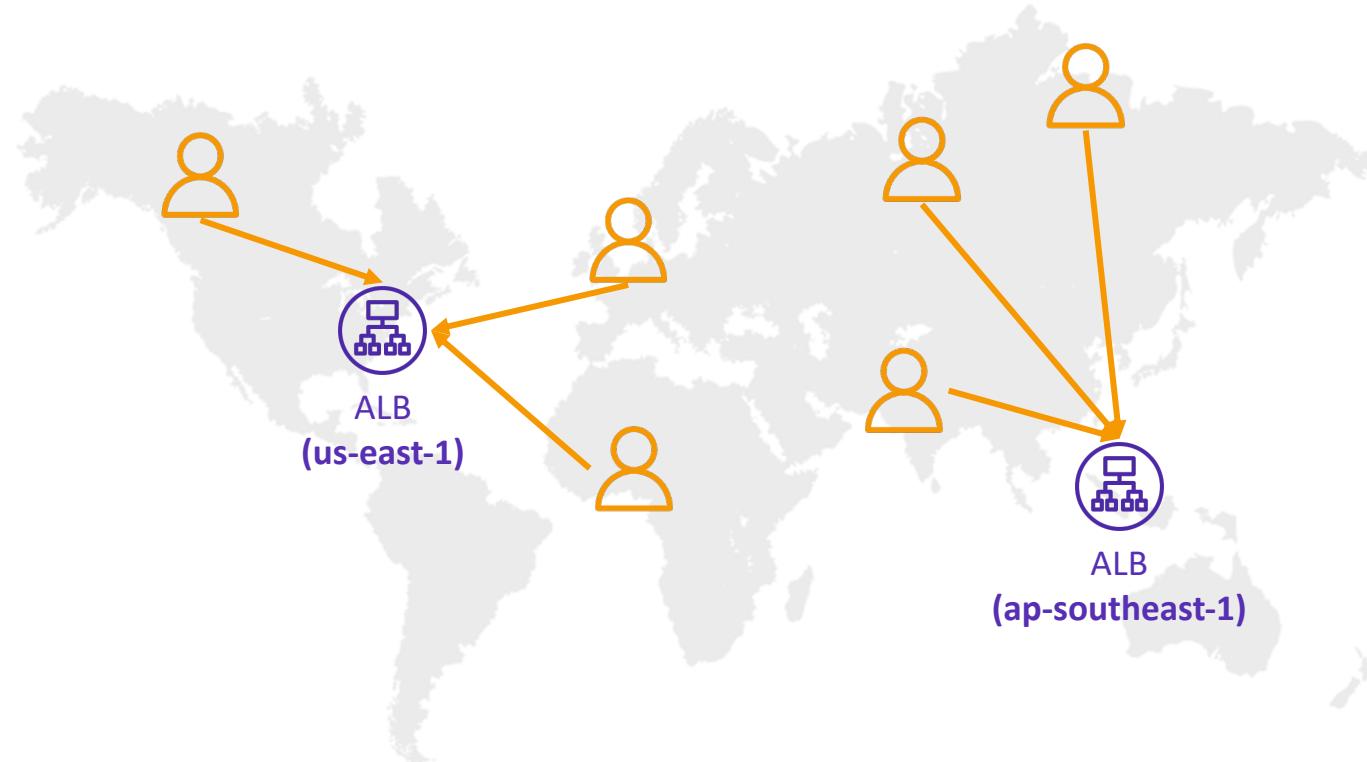
# Routing Policies – Weighted

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

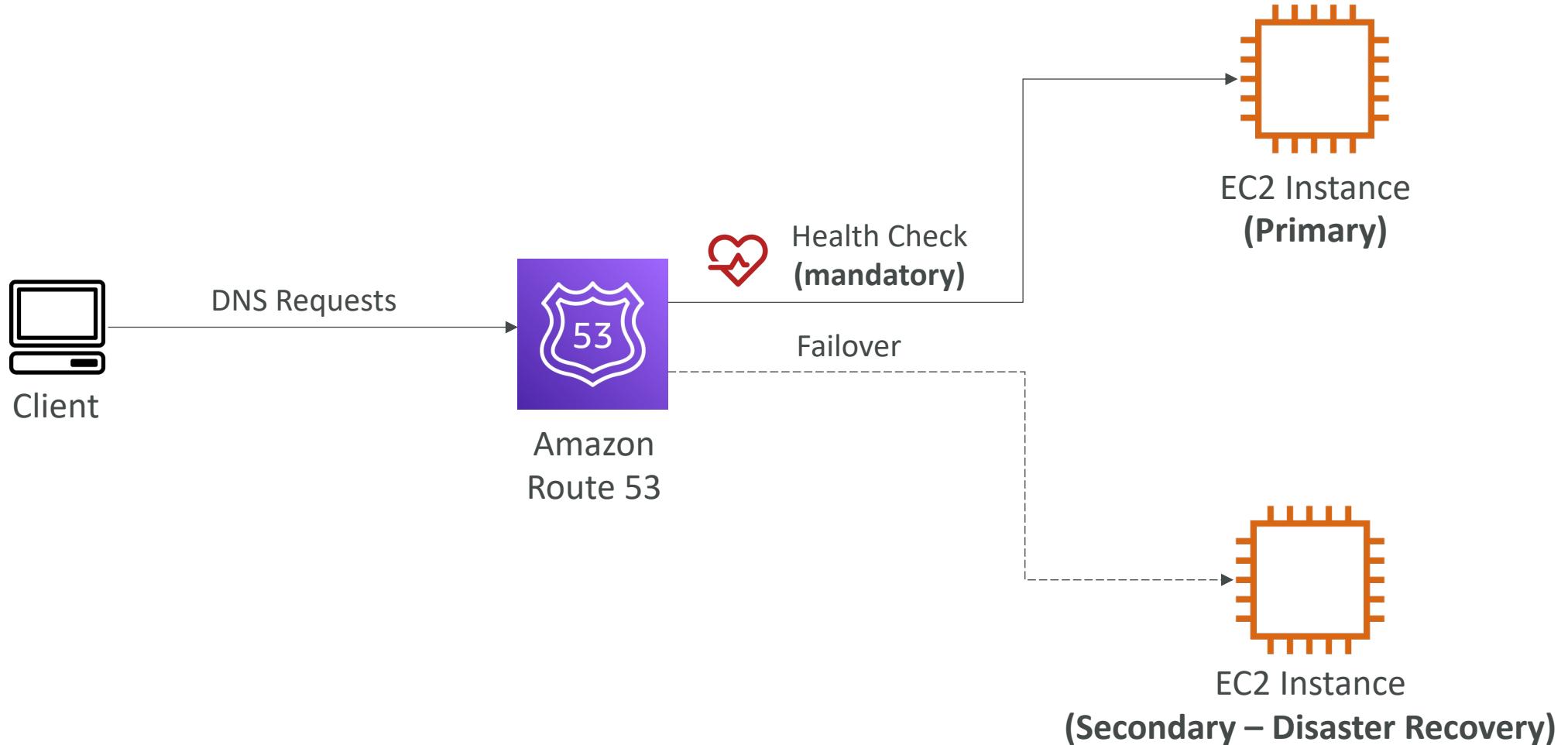


# Routing Policies – Latency-based

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

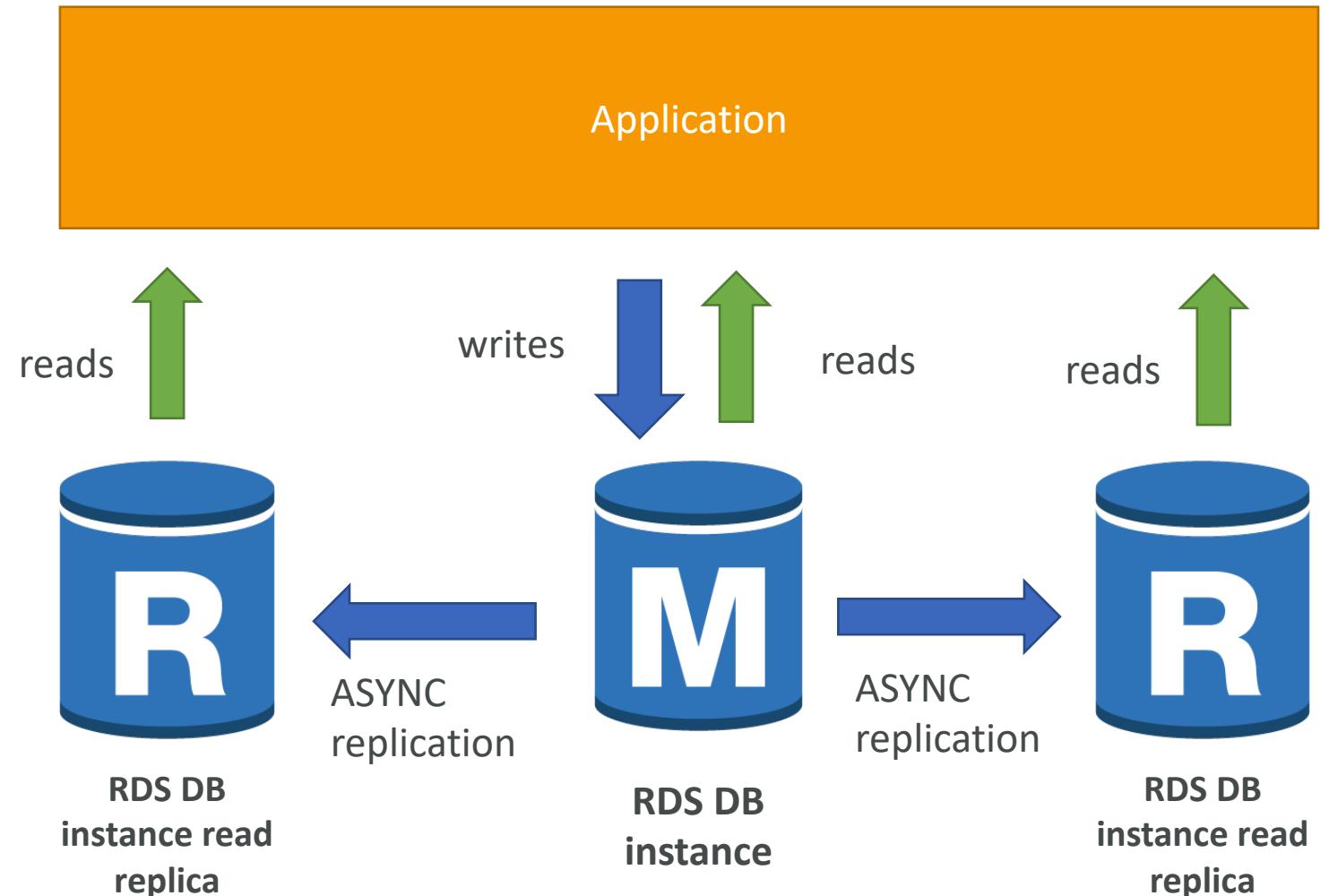


# Routing Policies – Failover (Active-Passive)



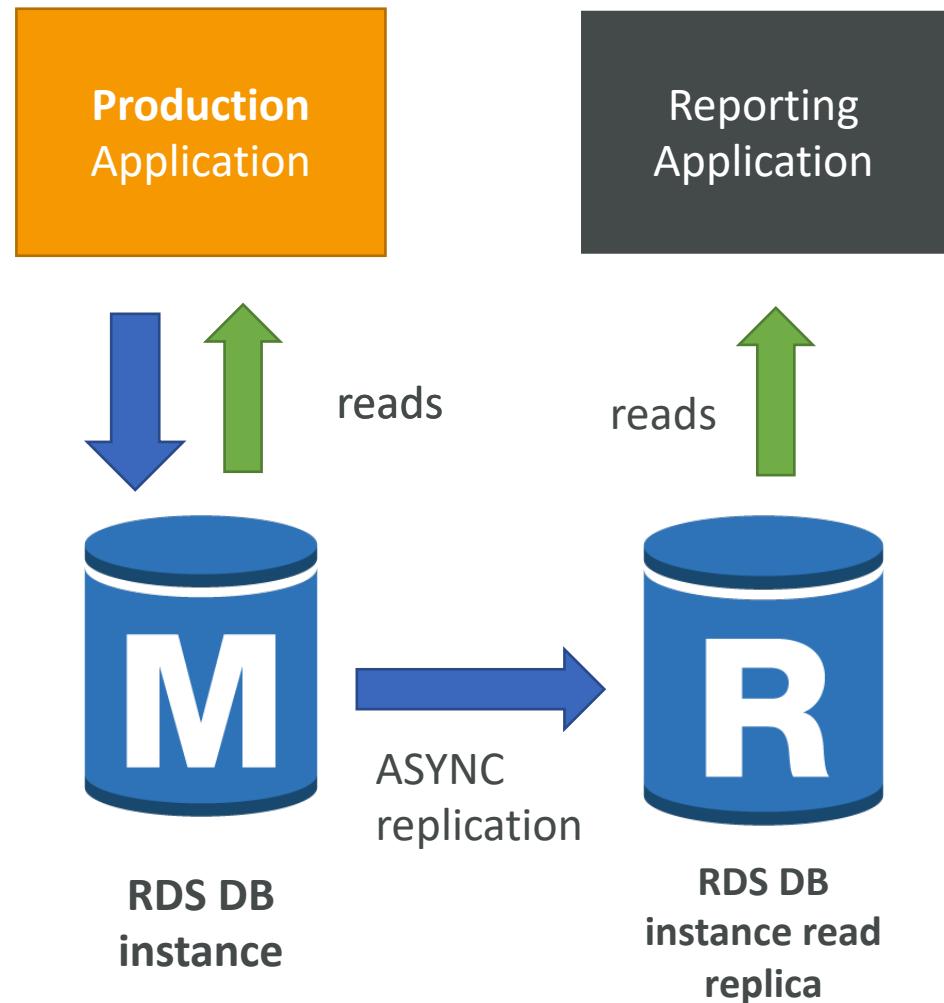
# RDS Read Replicas for read scalability

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



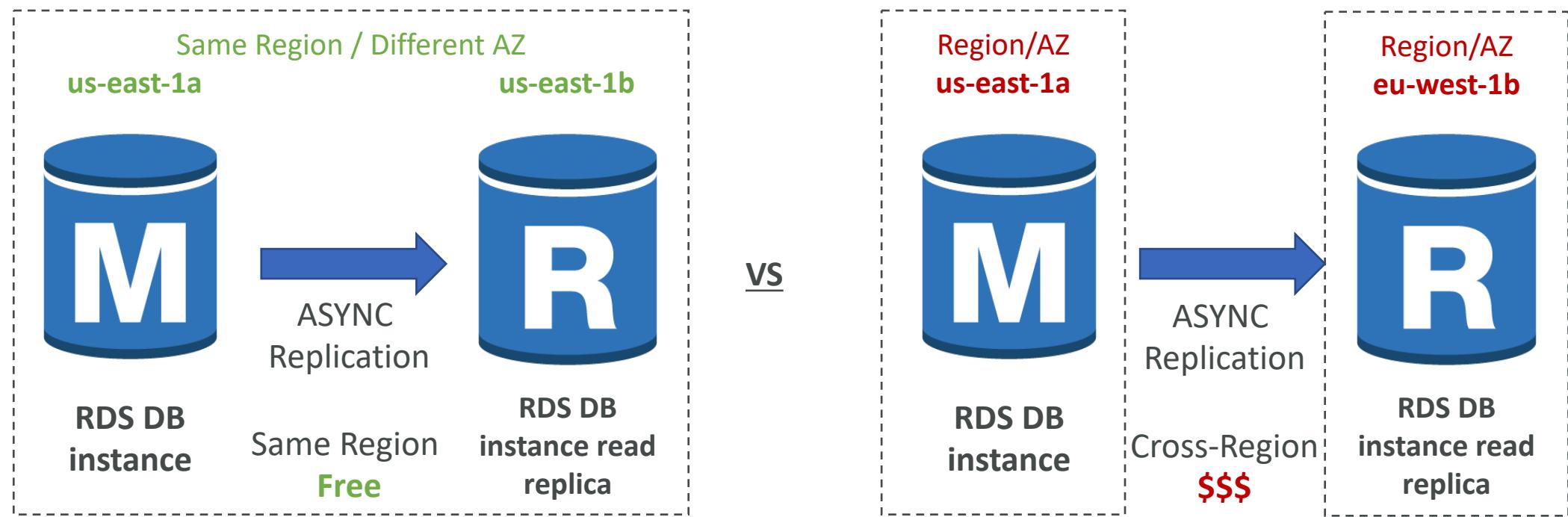
# RDS Read Replicas – Use Cases

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



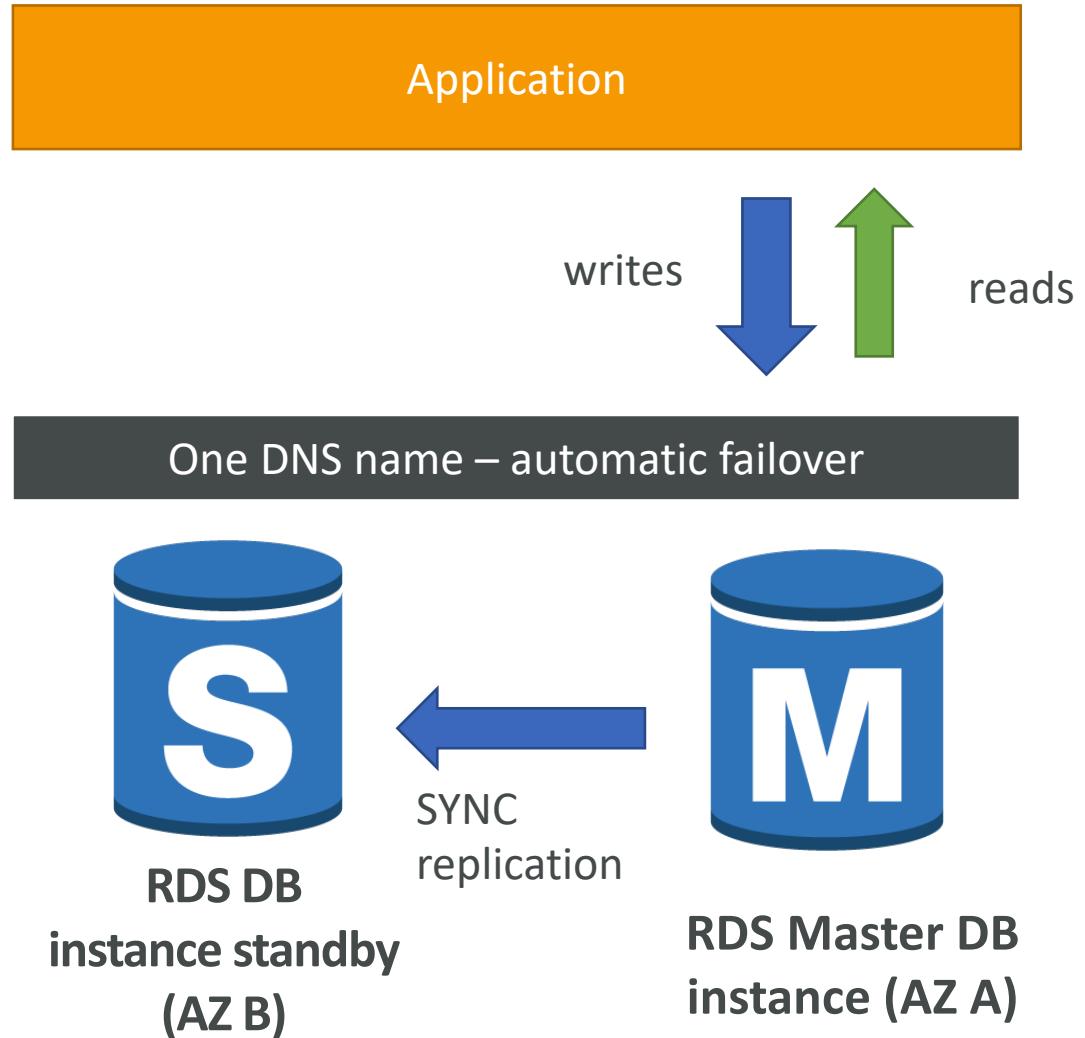
# RDS Read Replicas – Network Cost

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



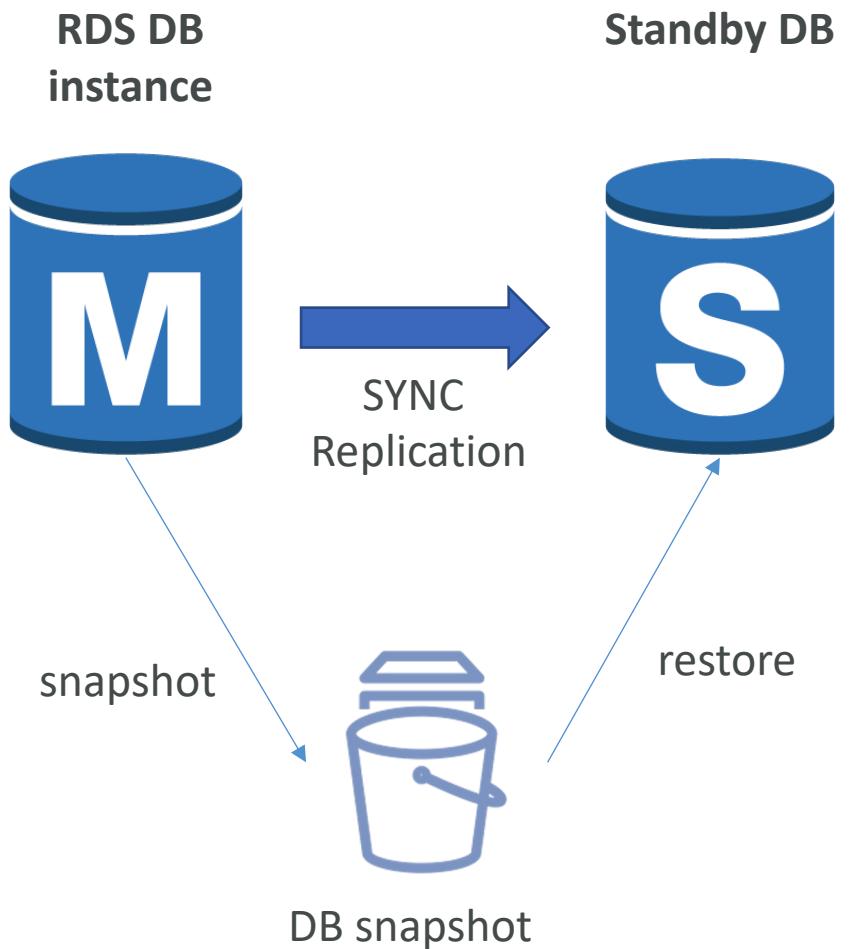
# RDS Multi AZ (Disaster Recovery)

- SYNC replication
- One DNS name – automatic app failover to standby
- Increase availability
- Failover in case of loss of AZ, loss of network, instance or storage failure
- No manual intervention in apps
- Not used for scaling
- Note: The Read Replicas be setup as Multi AZ for Disaster Recovery (DR)

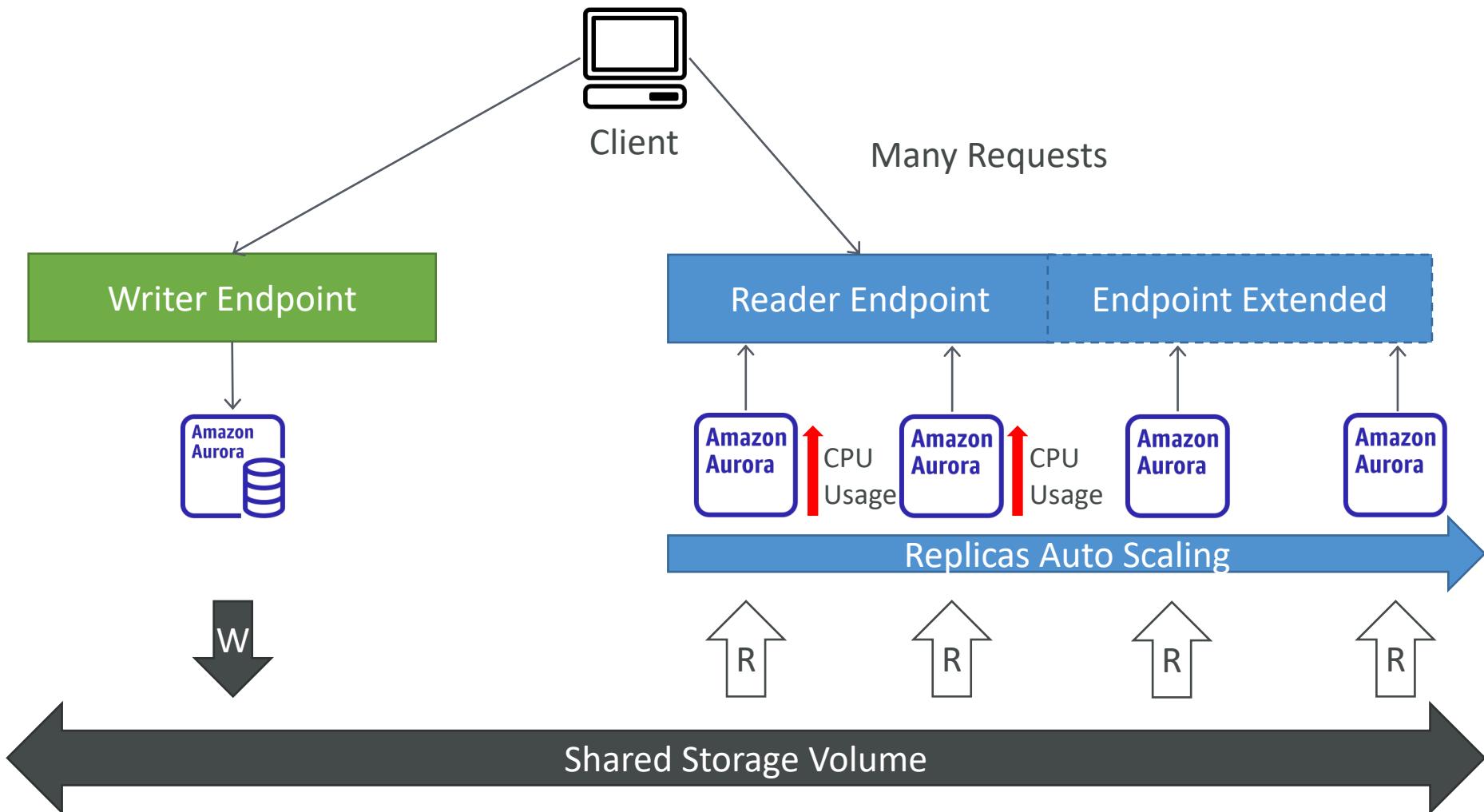


# RDS – From Single-AZ to Multi-AZ

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

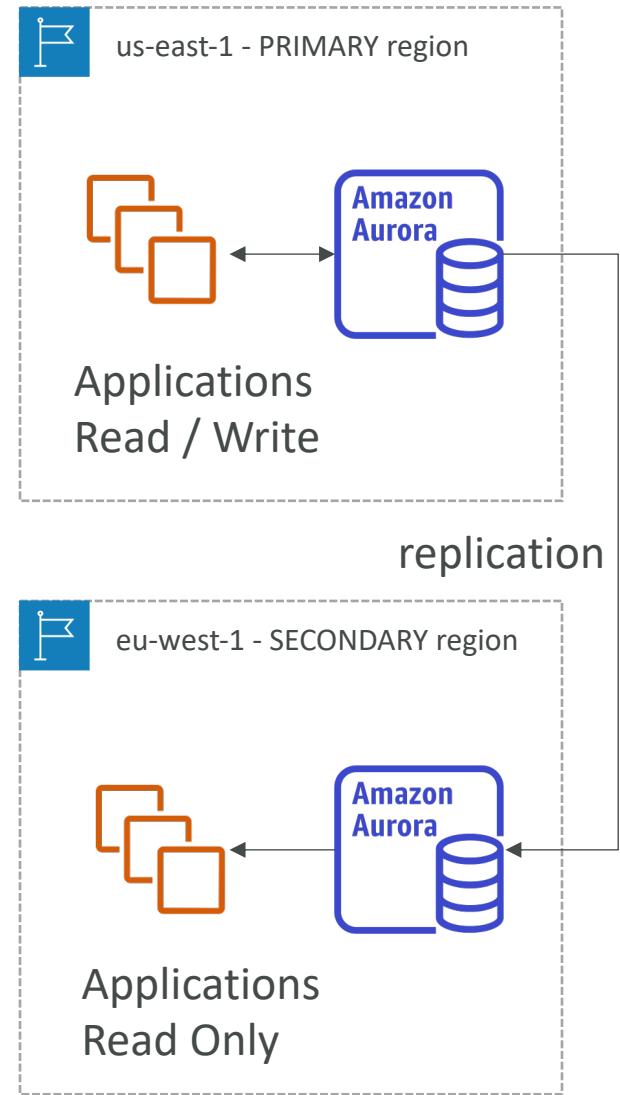


# Aurora Replicas - Auto Scaling

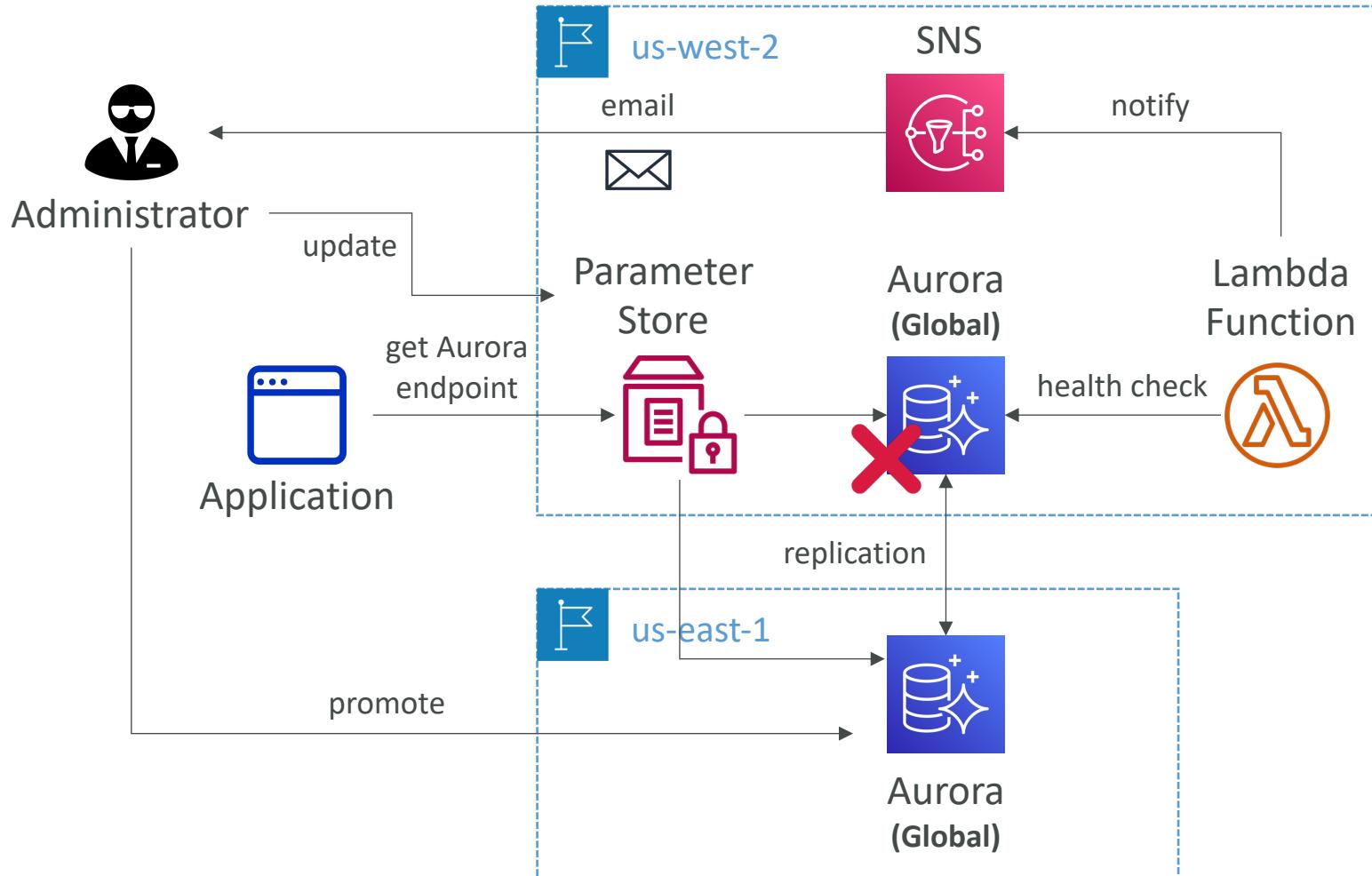


# Global Aurora

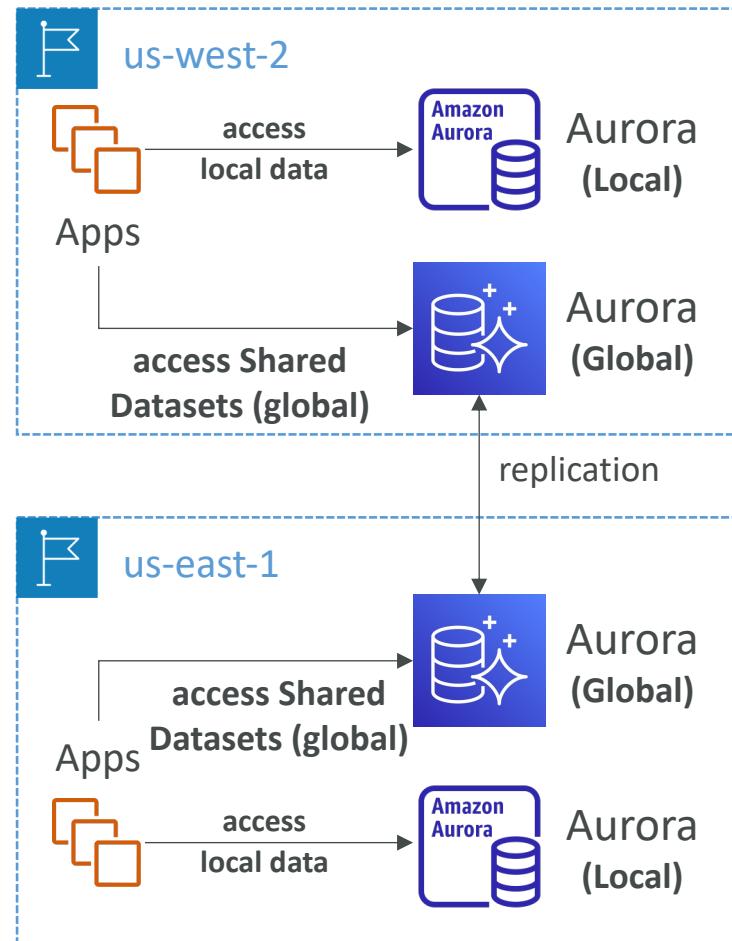
- Aurora Cross Region Read Replicas:
  - Useful for disaster recovery
  - Simple to put in place
- Aurora Global Database (recommended):
  - 1 Primary Region (read / write)
  - Up to 5 secondary (read-only) regions, replication lag is less than 1 second
  - Up to 16 Read Replicas per secondary region
  - Helps for decreasing latency
  - Promoting another region (for disaster recovery) has an RTO of < 1 minute
  - Typical cross-region replication takes less than 1 second



# Amazon Aurora – Unplanned Failover



# Amazon Aurora – Global Application



# Amazon ElastiCache Overview

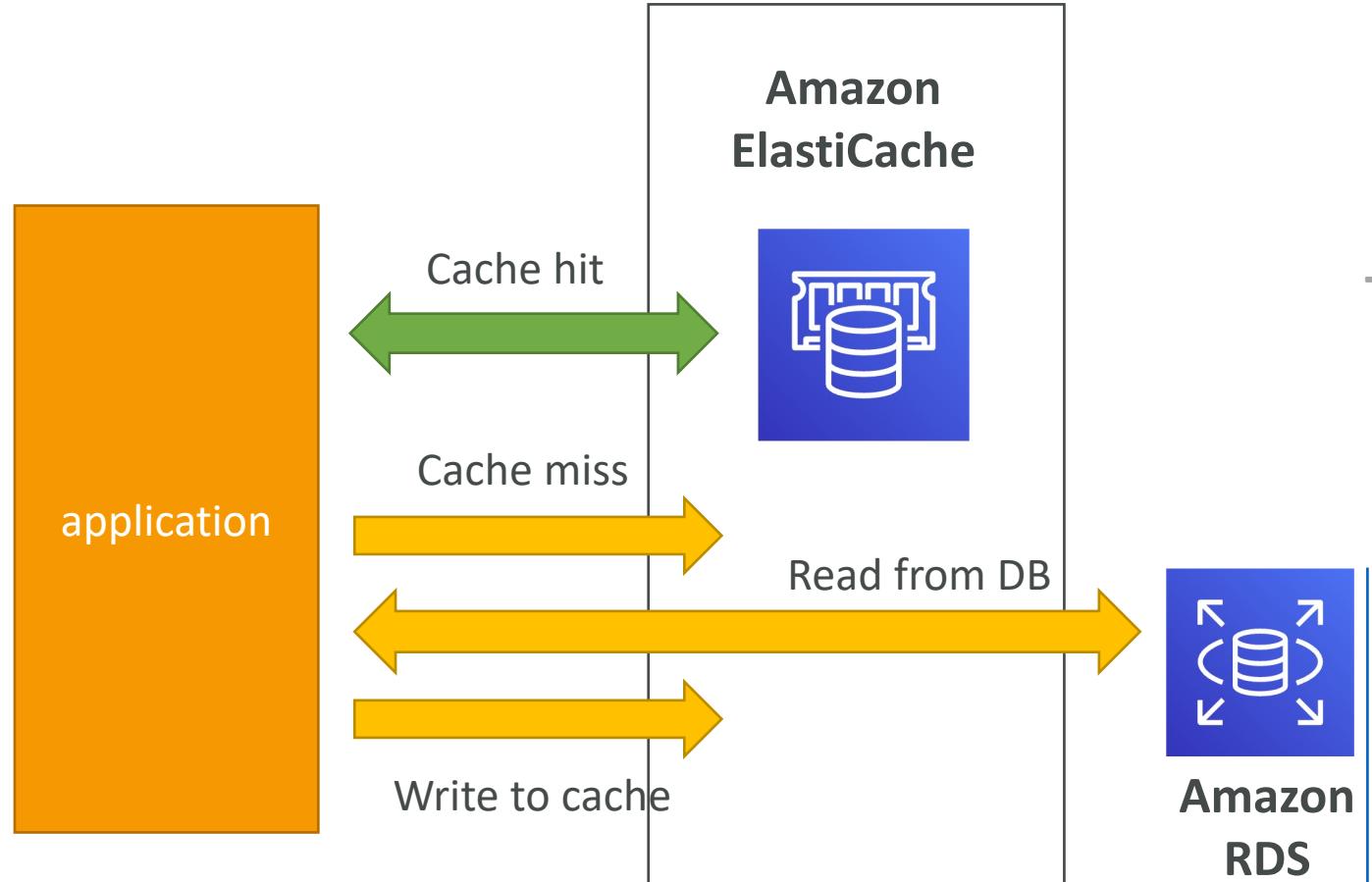


- The same way RDS is to get managed Relational Databases...
- ElastiCache is to get managed Redis or Memcached
- Caches are in-memory databases with really high performance, low latency
- Helps reduce load off of databases for read intensive workloads
- Helps make your application stateless
- AWS takes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

# ElastiCache

## Solution Architecture - DB Cache

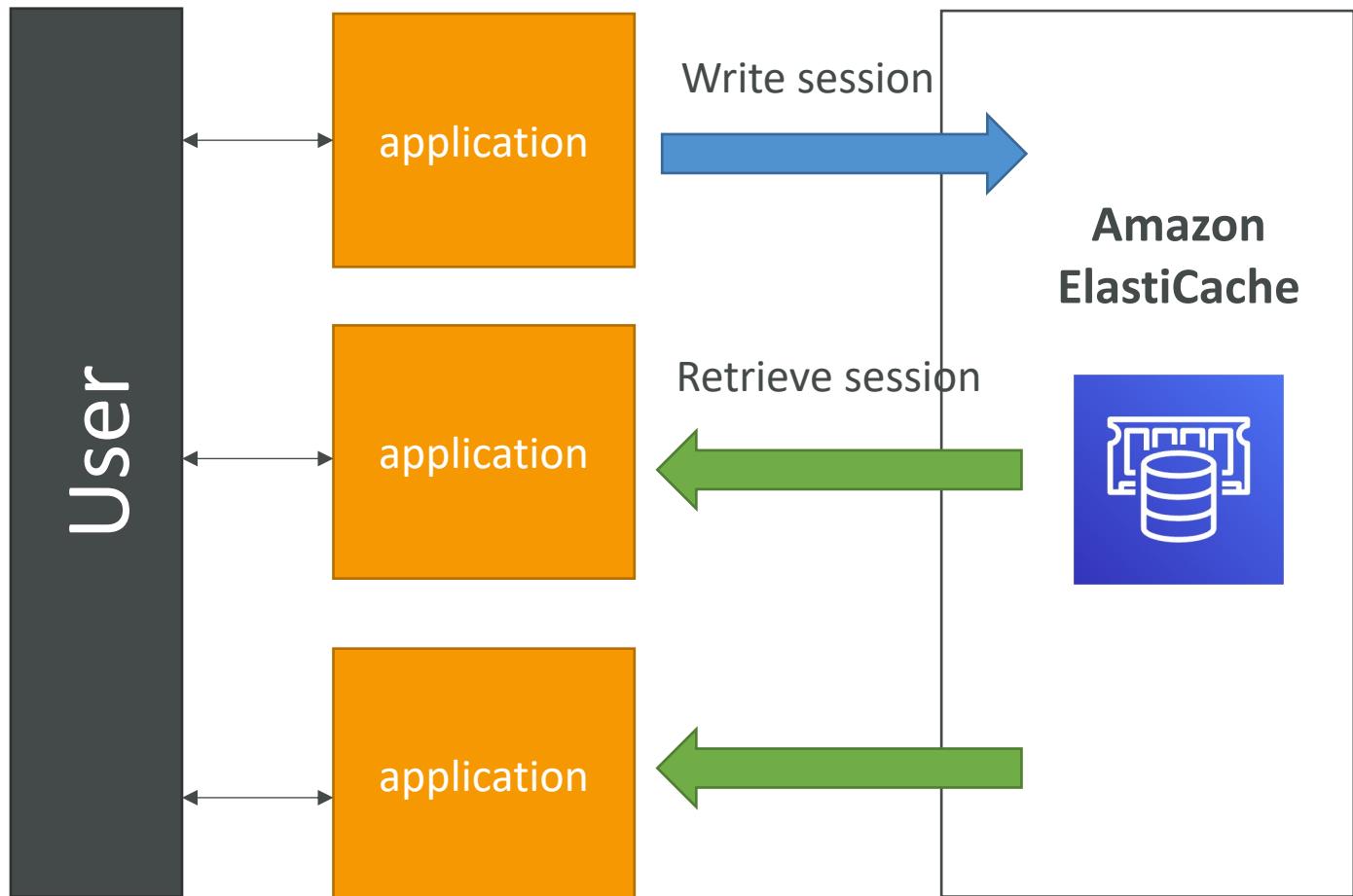
- Applications queries ElastiCache, if not available, get from RDS and store in ElastiCache.
- Helps relieve load in RDS
- Cache must have an invalidation strategy to make sure only the most current data is used in there.



# ElastiCache

## Solution Architecture – User Session Store

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



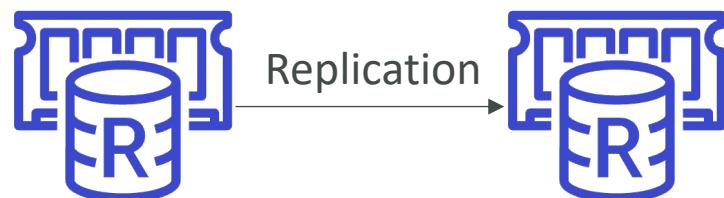
# ElastiCache – Redis vs Memcached

## REDIS

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

## MEMCACHED

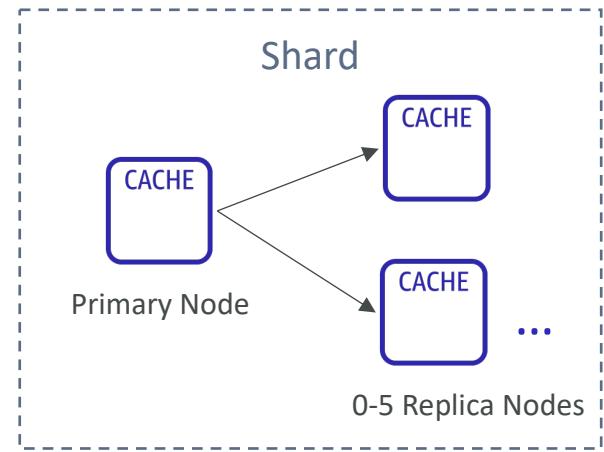
- Multi-node for partitioning of data (sharding)
- No high availability (replication)
- Non persistent
- No backup and restore
- Multi-threaded architecture



# ElastiCache Replication: Cluster Mode Disabled

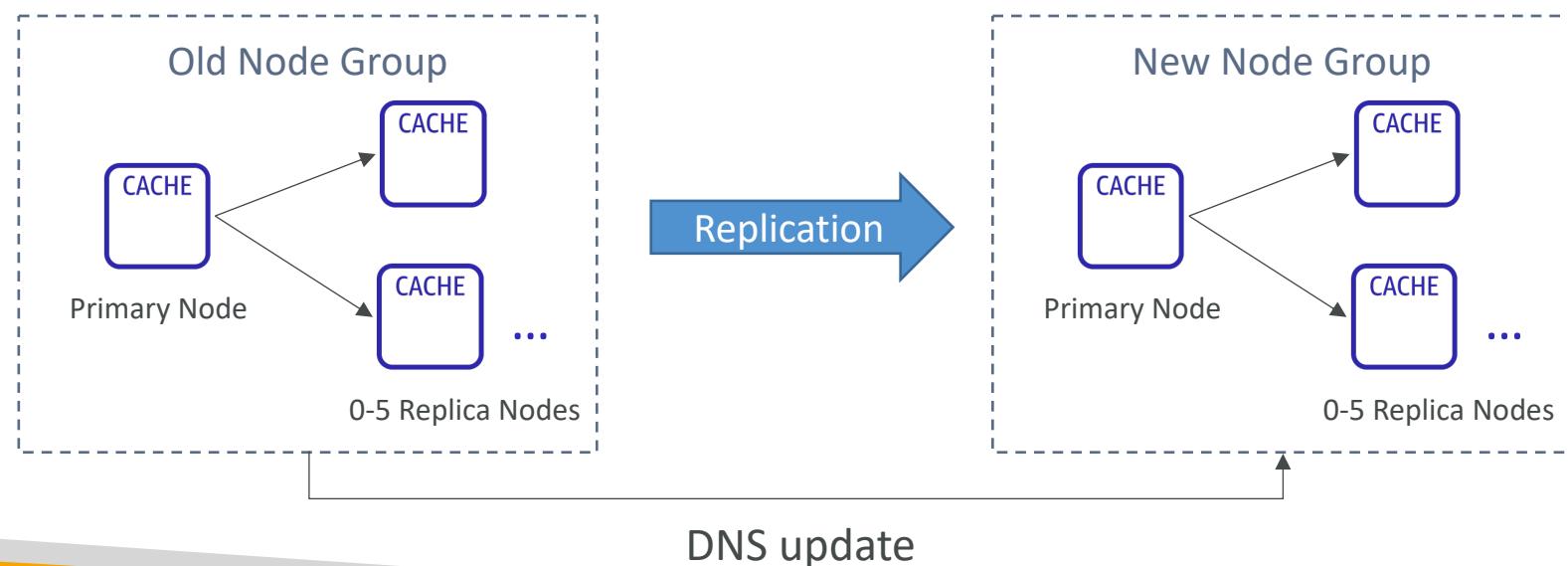
- One primary node, up to 5 replicas
- Asynchronous Replication
- The primary node is used for read/write
- The other nodes are read-only
- **One shard, all nodes have all the data**
- Guard against data loss if node failure
- Multi-AZ enabled by default for failover
- Helpful to scale read performance

Redis (cluster mode disabled) Cluster with Replication



# Redis Scaling – Cluster Mode Disabled

- Horizontal:
  - Scale out/in by adding/removing read replicas (max. 5 replicas)
- Vertical:
  - Scale up/down to larger/smaller node type
  - ElastiCache will internally create a new node group, then data replication and DNS update

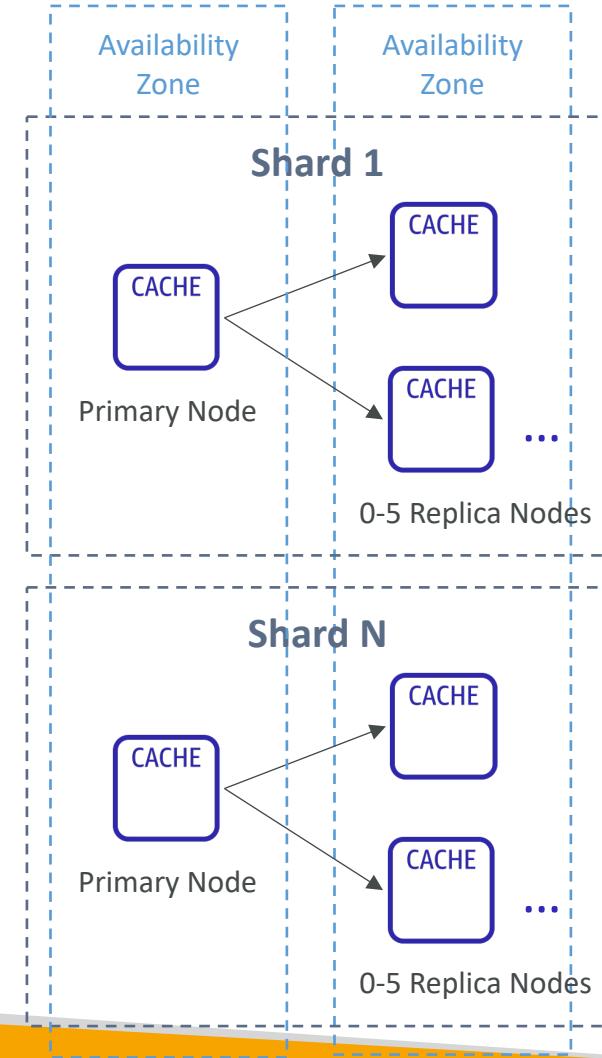


# ElastiCache Replication: Cluster Mode Enabled

- Data is partitioned across shards (helpful to scale writes)
- Each shard has a primary and up to 5 replica nodes (same concept as before)
- Multi-AZ capability
- Up to 500 nodes per cluster:
  - 500 shards with single master
  - 250 shards with 1 master and 1 replica
  - ...
  - 83 shards with one master and 5 replicas

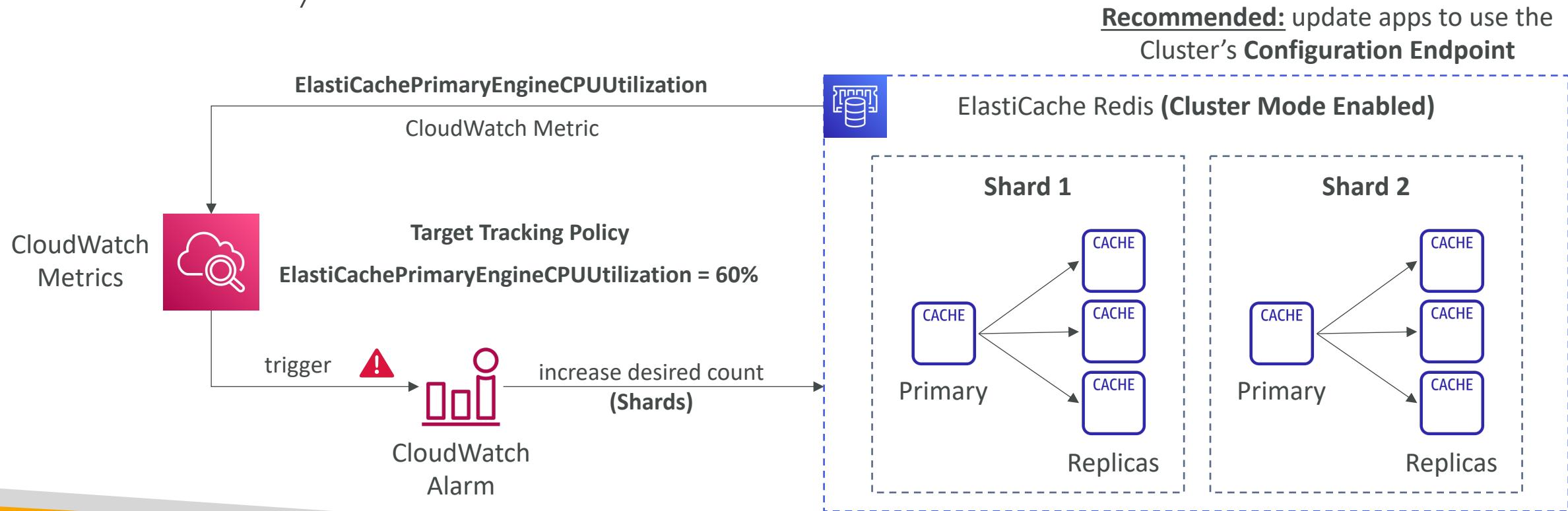
Redis (cluster mode enabled) Cluster

with Replication



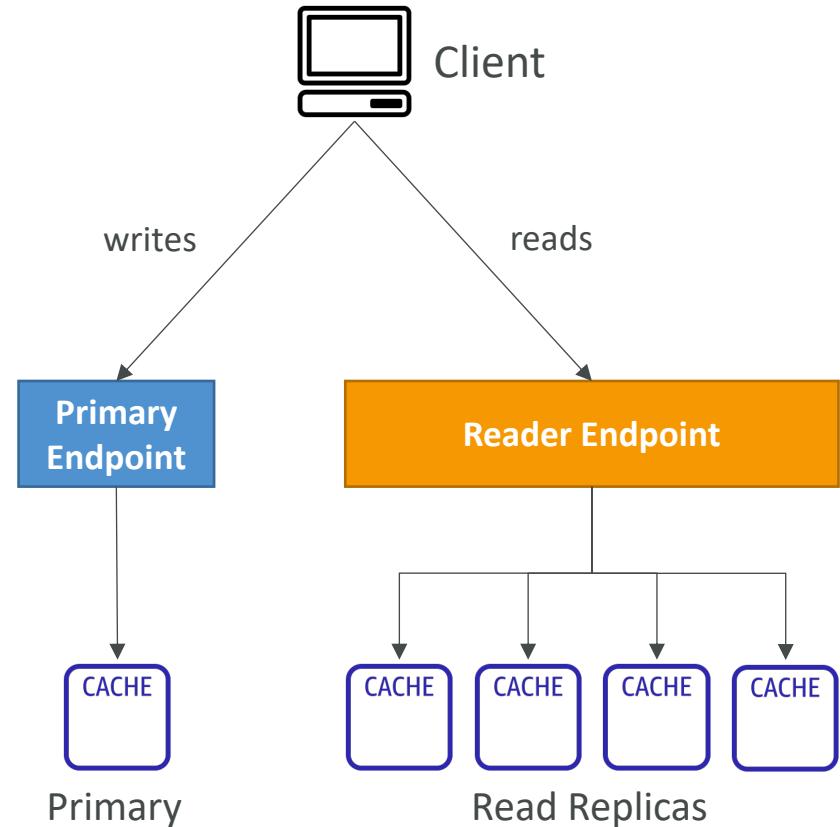
# ElastiCache for Redis – Auto Scaling

- Automatically increase/decrease the desired shards or replicas
- Supports both Target Tracking and Scheduled Scaling Policies
- Works only for Redis with Cluster Mode Enabled



# ElastiCache – Redis Connection Endpoints

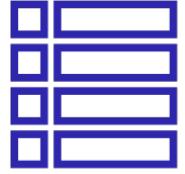
- Standalone Node
  - One endpoint for read and write operations
- Cluster Mode Disabled Cluster
  - Primary Endpoint – for all write operations
  - Reader Endpoint – evenly split read operations across all read replicas
  - Node Endpoint – for read operations
- Cluster Mode Enabled Cluster
  - Configuration Endpoint – for all read/write operations that support Cluster Mode Enabled commands
  - Node Endpoint – for read operations



# Amazon DynamoDB



- Fully managed, highly available with replication across multiple AZs
- NoSQL database - not a relational database - with transaction support
- Scales to massive workloads, distributed database
- Millions of requests per seconds, trillions of row, 100s of TB of storage
- Fast and consistent in performance (single-digit millisecond)
- Integrated with IAM for security, authorization and administration
- Low cost and auto-scaling capabilities
- No maintenance or patching, always available
- Standard & Infrequent Access (IA) Table Class



# DynamoDB - Basics

- DynamoDB is made of **Tables**
- Each table has a **Primary Key** (must be decided at creation time)
- Each table can have an infinite number of items (= rows)
- Each item has **attributes** (can be added over time – can be null)
- Maximum size of an item is **400KB**
- Data types supported are:
  - **Scalar Types** – String, Number, Binary, Boolean, Null
  - **Document Types** – List, Map
  - **Set Types** – String Set, Number Set, Binary Set
- Therefore, in DynamoDB you can rapidly evolve schemas

# DynamoDB – Table example

Primary Key		Attributes	
Partition Key	Sort Key	Score	Result
User_ID	Game_ID	Score	Result
7791a3d6...	4421	92	Win
873e0634...	1894	14	Lose
873e0634...	4521	77	Win

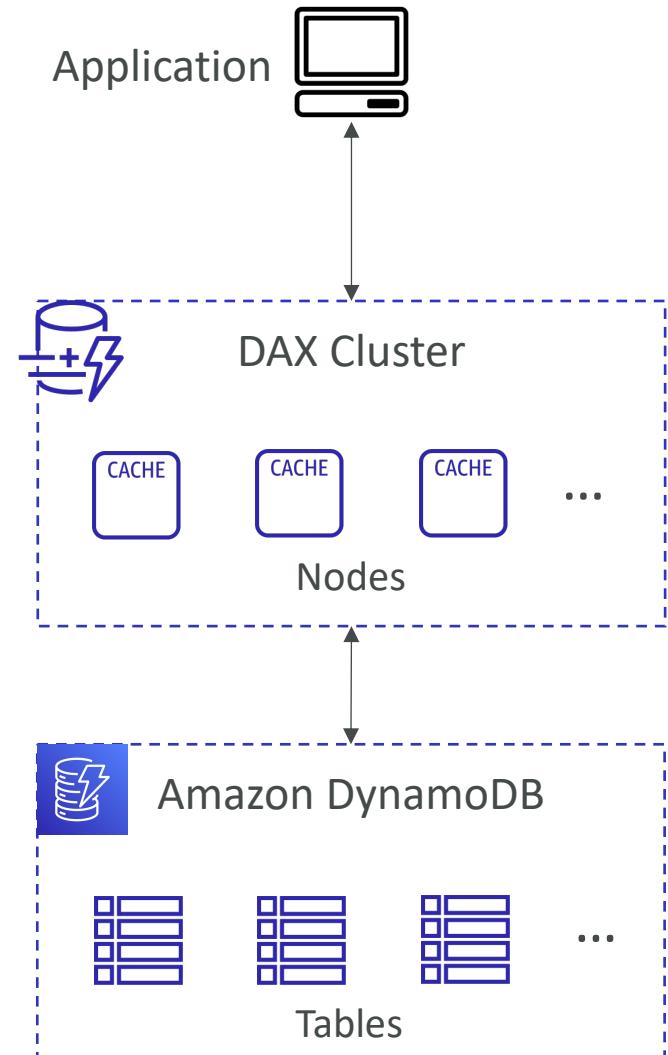
# DynamoDB – Read/Write Capacity Modes

- Control how you manage your table's capacity (read/write throughput)
- Provisioned Mode (default)
  - You specify the number of reads/writes per second
  - You need to plan capacity beforehand
  - Pay for provisioned Read Capacity Units (RCU) & Write Capacity Units (WCU)
  - Possibility to add auto-scaling mode for RCU & WCU
- On-Demand Mode
  - Read/writes automatically scale up/down with your workloads
  - No capacity planning needed
  - Pay for what you use, more expensive (\$\$\$)
  - Great for unpredictable workloads, steep sudden spikes

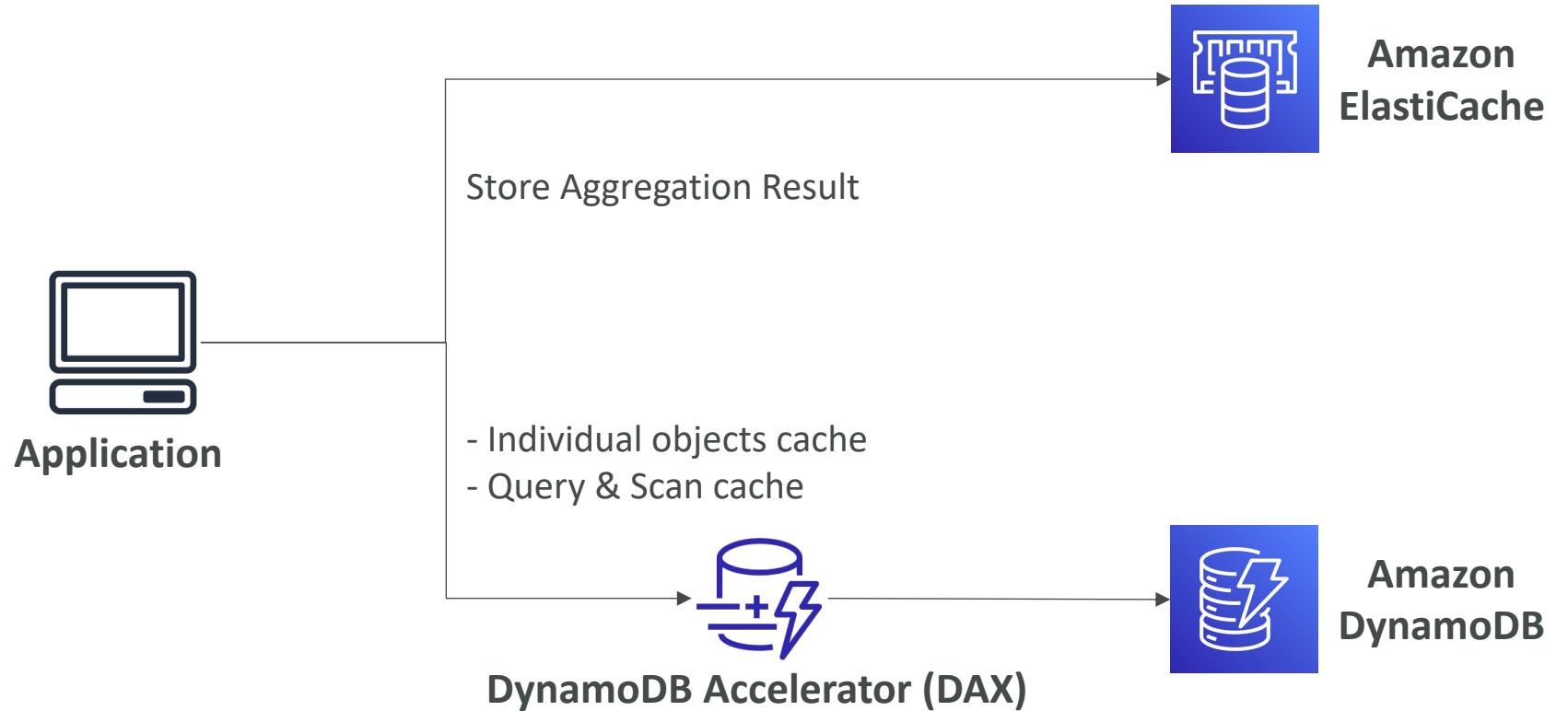
# DynamoDB Accelerator (DAX)

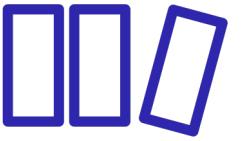


- Fully-managed, highly available, seamless in-memory cache for DynamoDB
- Help solve read congestion by caching
- Microseconds latency for cached data
- Doesn't require application logic modification (compatible with existing DynamoDB APIs)
- 5 minutes TTL for cache (default)



# DynamoDB Accelerator (DAX) vs. ElastiCache





# DynamoDB – Stream Processing

- Ordered stream of item-level modifications (create/update/delete) in a table
- Use cases:
  - React to changes in real-time (welcome email to users)
  - Real-time usage analytics
  - Insert into derivative tables
  - Implement cross-region replication
  - Invoke AWS Lambda on changes to your DynamoDB table

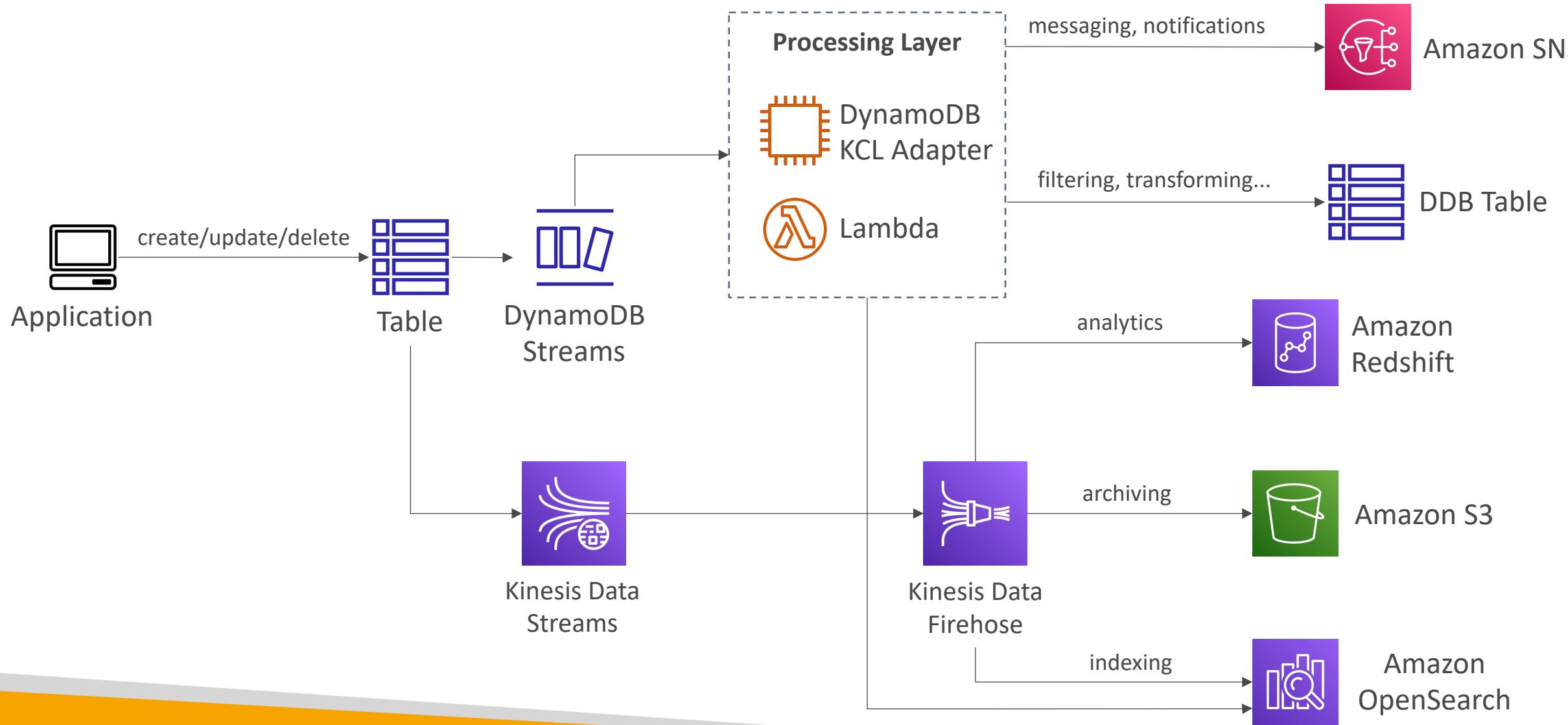
## DynamoDB Streams

- 24 hours retention
- Limited # of consumers
- Process using AWS Lambda Triggers, or DynamoDB Stream Kinesis adapter

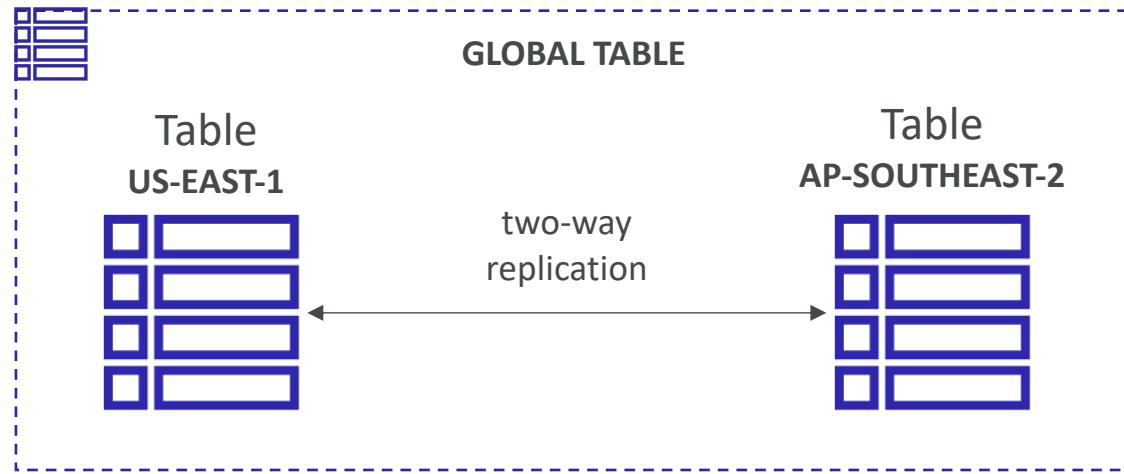
## Kinesis Data Streams (newer)

- 1 year retention
- High # of consumers
- Process using AWS Lambda, Kinesis Data Analytics, Kinesis Data Firehose, AWS Glue Streaming ETL...

# DynamoDB Streams



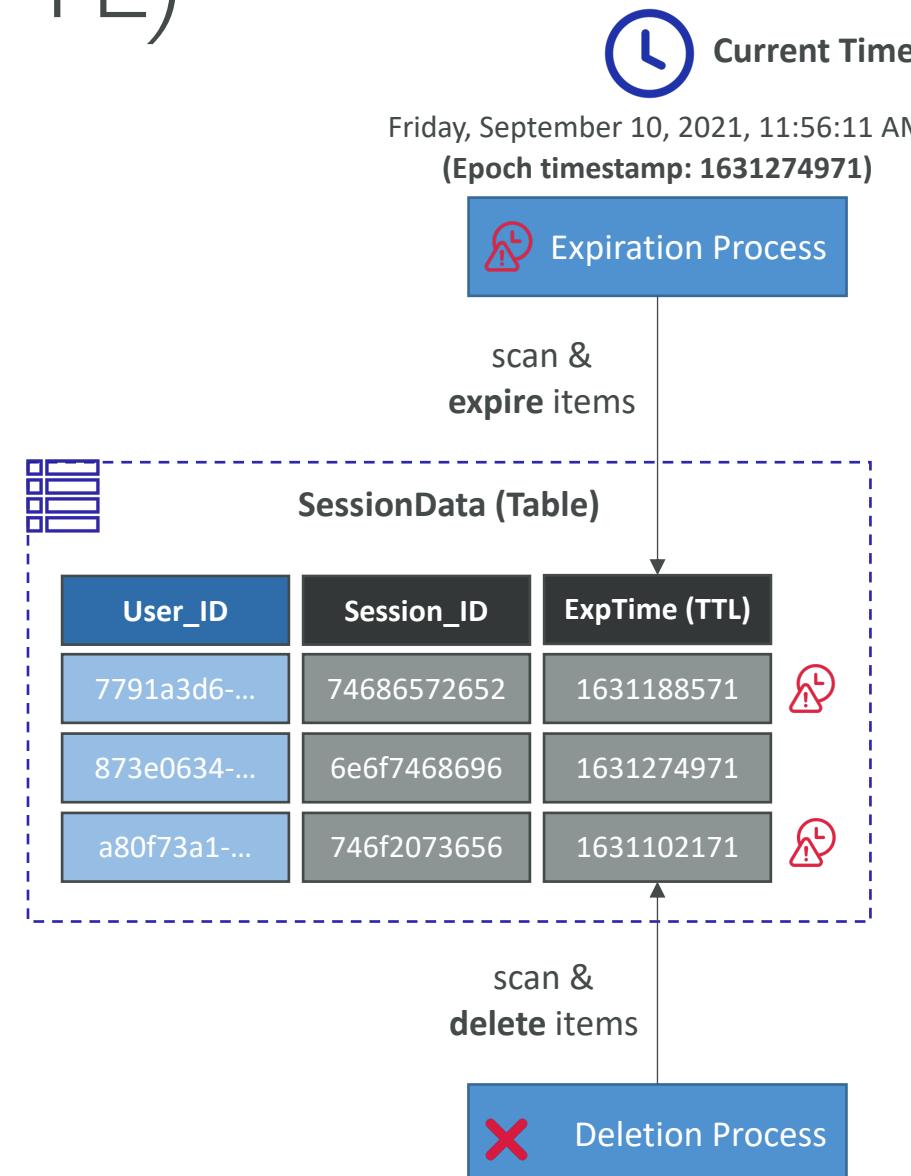
# DynamoDB Global Tables



- Make a DynamoDB table accessible with **low latency** in multiple-regions
- Active-Active replication
- Applications can READ and WRITE to the table in any region
- Must enable DynamoDB Streams as a pre-requisite

# DynamoDB – Time To Live (TTL)

- Automatically delete items after an expiry timestamp
- Use cases: reduce stored data by keeping only current items, adhere to regulatory obligations, web session handling...

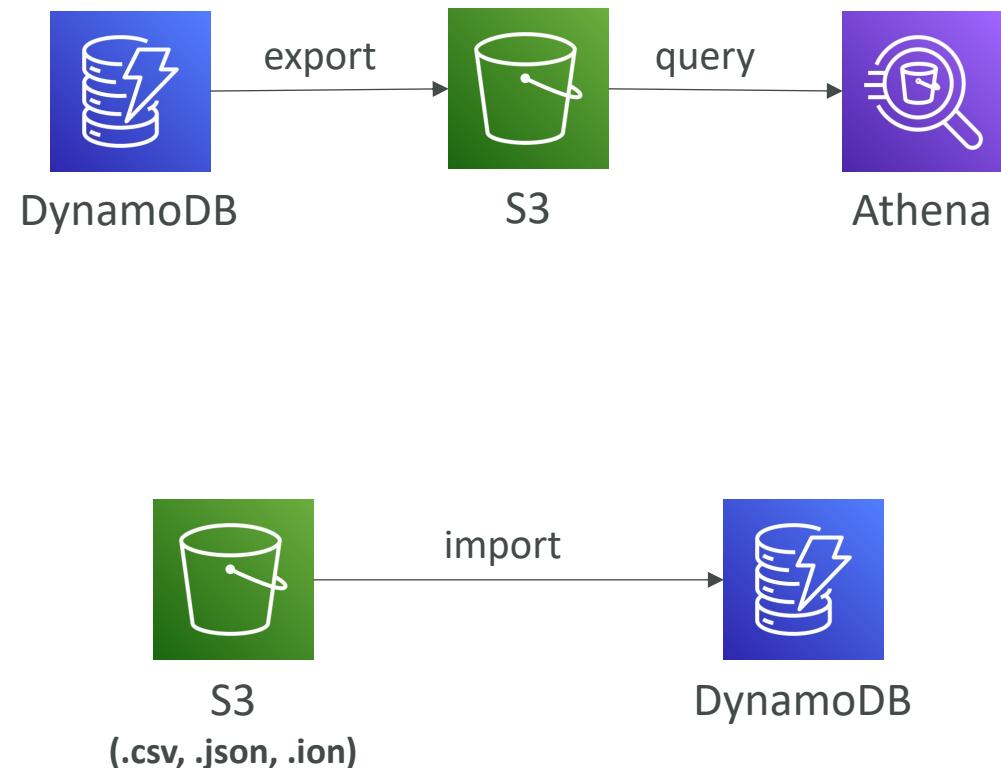


# DynamoDB – Backups for disaster recovery

- Continuous backups using point-in-time recovery (PITR)
  - Optionally enabled for the last 35 days
  - Point-in-time recovery to any time within the backup window
  - The recovery process creates a new table
- On-demand backups
  - Full backups for long-term retention, until explicitly deleted
  - Doesn't affect performance or latency
  - Can be configured and managed in AWS Backup (enables cross-region copy)
  - The recovery process creates a new table

# DynamoDB – Integration with Amazon S3

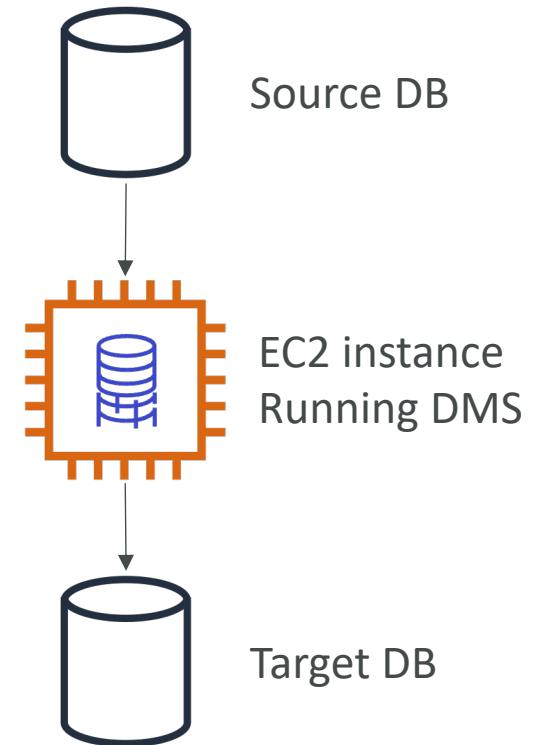
- Export to S3 (must enable PITR)
  - Works for any point of time in the last 35 days
  - Doesn't affect the read capacity of your table
  - Perform data analysis on top of DynamoDB
  - Retain snapshots for auditing
  - ETL on top of S3 data before importing back into DynamoDB
  - Export in DynamoDB JSON or ION format
- Import from S3
  - Import CSV, DynamoDB JSON or ION format
  - Doesn't consume any write capacity
  - Creates a new table
  - Import errors are logged in CloudWatch Logs





# DMS – Database Migration Service

- Quickly and securely migrate databases to AWS, resilient, self healing
- The source database remains available during the migration
- Supports:
  - Homogeneous migrations: ex Oracle to Oracle
  - Heterogeneous migrations: ex Microsoft SQL Server to Aurora
- Continuous Data Replication using CDC
- You must create an EC2 instance to perform the replication tasks



# DMS Sources and Targets

## SOURCES:

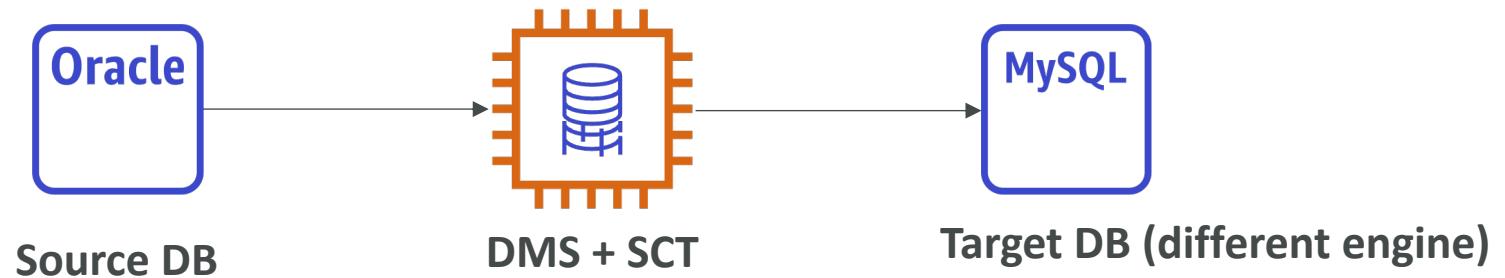
- On-Premises and EC2 instances databases: Oracle, MS SQL Server, MySQL, MariaDB, PostgreSQL, MongoDB, SAP, DB2
- Azure: *Azure SQL Database*
- Amazon RDS: all including Aurora
- Amazon S3
- DocumentDB

## TARGETS:

- On-Premises and EC2 instances databases: Oracle, MS SQL Server, MySQL, MariaDB, PostgreSQL, SAP
- Amazon RDS
- Redshift, DynamoDB, S3
- OpenSearch Service
- Kinesis Data Streams
- Apache Kafka
- DocumentDB & Amazon Neptune
- Redis & Babelfish

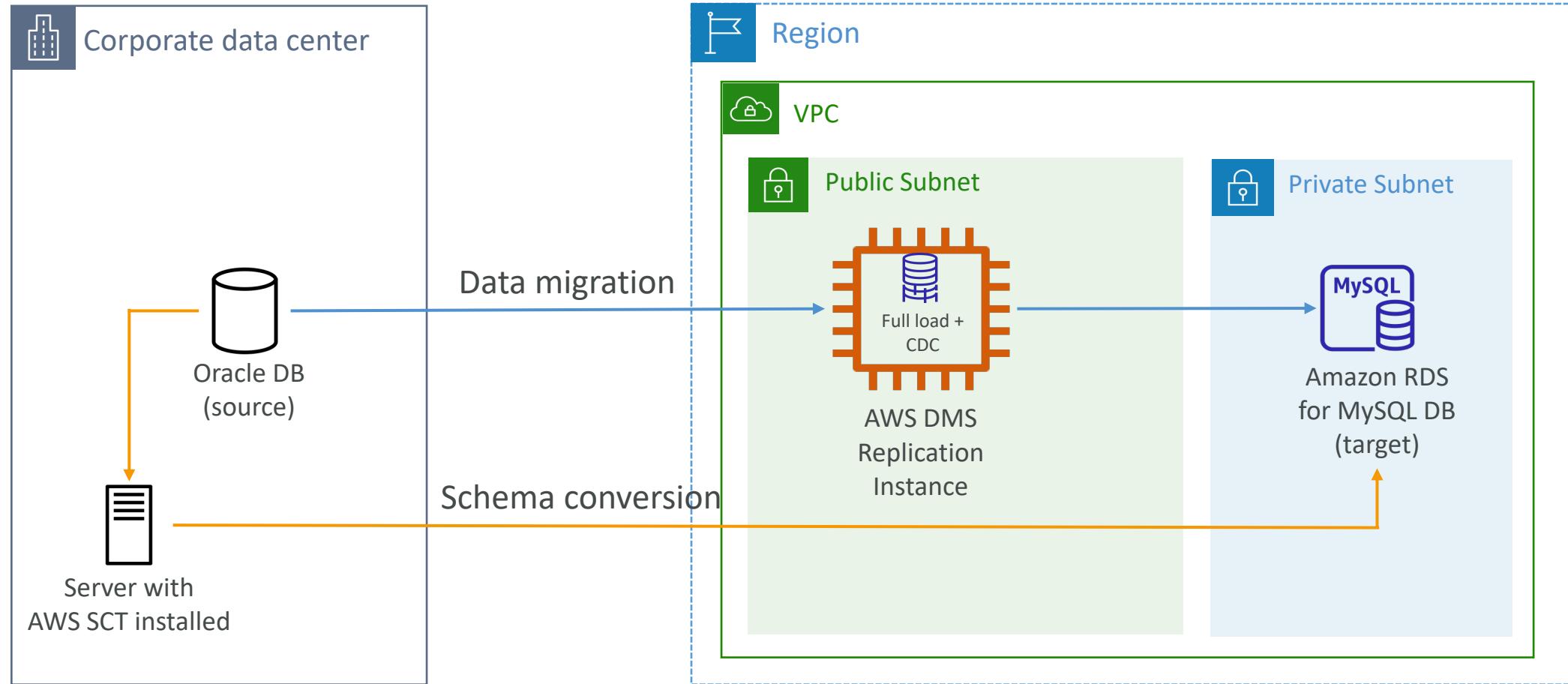
# AWS Schema Conversion Tool (SCT)

- Convert your Database's Schema from one engine to another
- Example OLTP: (SQL Server or Oracle) to MySQL, PostgreSQL, Aurora
- Example OLAP: (Teradata or Oracle) to Amazon Redshift
- Prefer compute-intensive instances to optimize data conversions



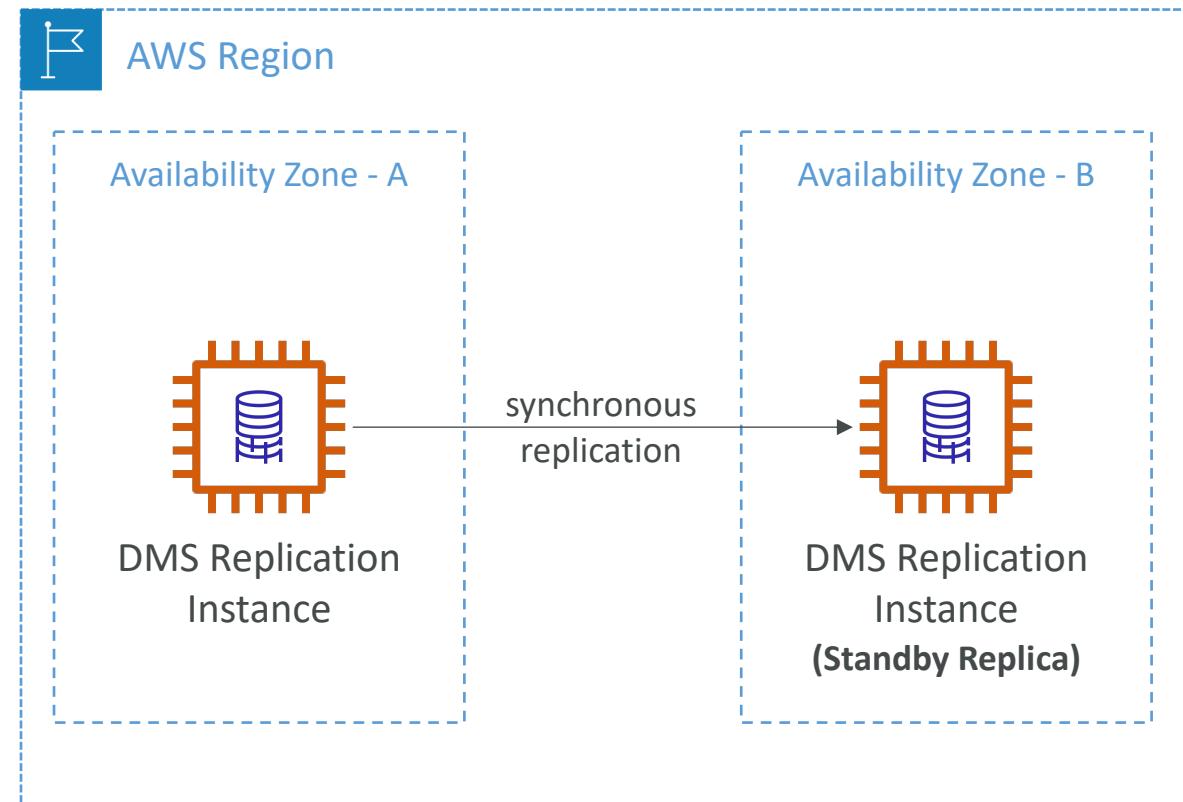
- You do not need to use SCT if you are migrating the same DB engine
  - Ex: On-Premise PostgreSQL => RDS PostgreSQL
  - The DB engine is still PostgreSQL (RDS is the platform)

# DMS - Continuous Replication



# AWS DMS – Multi-AZ Deployment

- When Multi-AZ Enabled, DMS provisions and maintains a synchronously stand replica in a different AZ
- Advantages:
  - Provides Data Redundancy
  - Eliminates I/O freezes
  - Minimizes latency spikes



# AWS DMS – Replication Task Monitoring

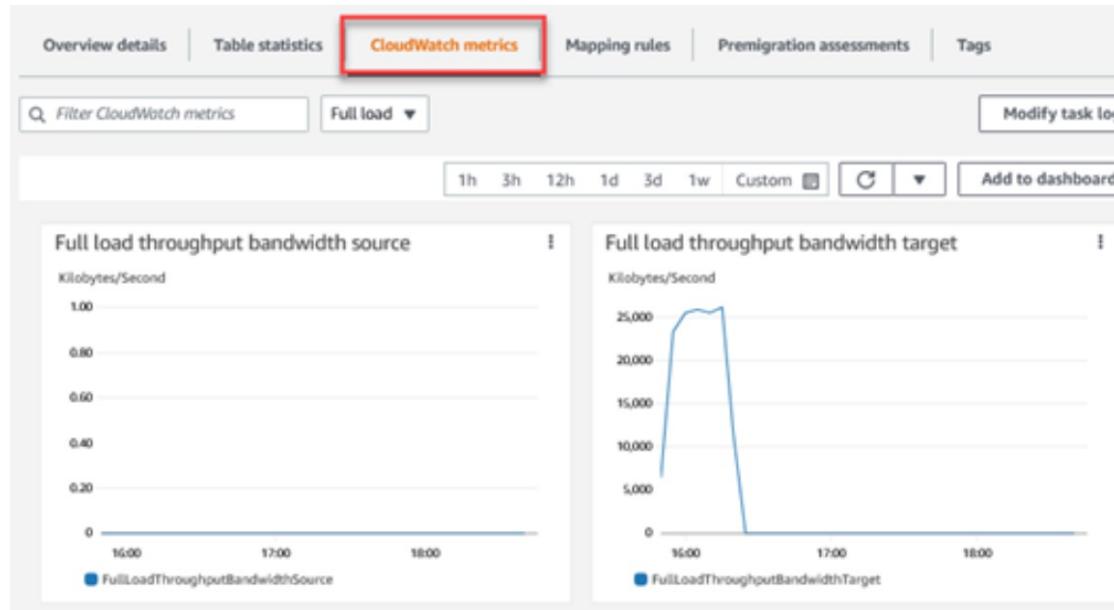
- Task Status
  - Indicates the condition of the Task (e.g., Creating, Running, Stopped...)
  - Task Status Bar – gives an estimation of the Task's progress
- Table State
  - Includes the current state of the tables (e.g., Before load, Table completed...)
  - Number of inserts, deletions, and updates for each table

Schema name	Table	Load state	Elapsed load time	Inserts	Deletes	Updates	DDLs	Applied inserts
mysql	user	Table error	1 s	0	0	0	0	0
mysql	server_cost	Table completed	1 s	0	0	0	0	0
dms_sample	seat_type	Table completed	< 1 s	0	0	0	0	0

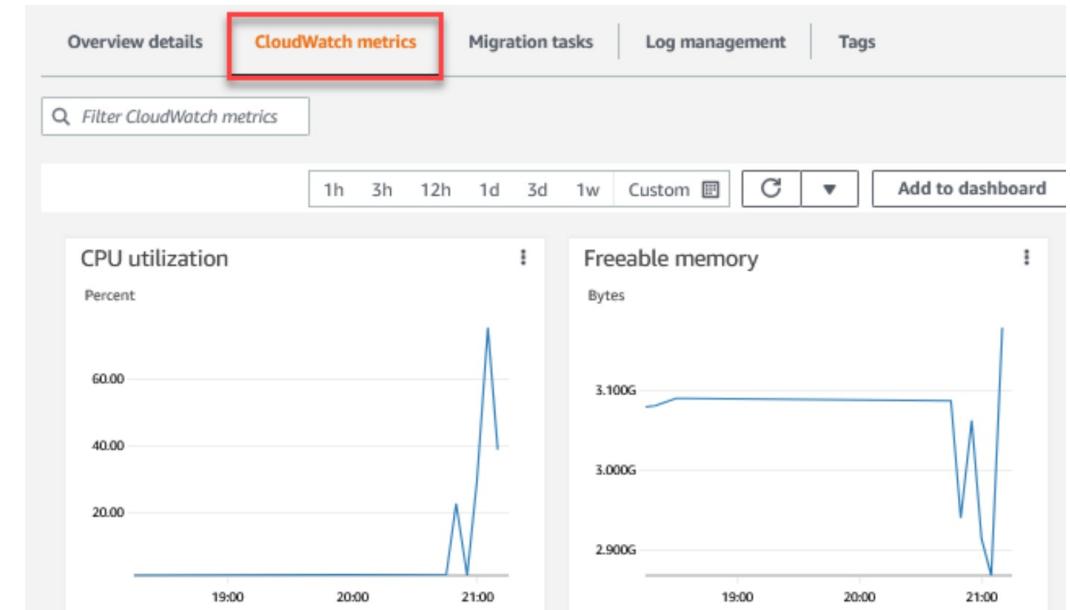
## Table State

[https://docs.aws.amazon.com/dms/latest/userguide/CHAP\\_Monitoring.html](https://docs.aws.amazon.com/dms/latest/userguide/CHAP_Monitoring.html)

# AWS DMS – CloudWatch Metrics



Replication Task Metrics



Host Metrics

[https://docs.aws.amazon.com/dms/latest/userguide/CHAP\\_Monitoring.html](https://docs.aws.amazon.com/dms/latest/userguide/CHAP_Monitoring.html)

# AWS DMS – CloudWatch Metrics

- **Host Metrics**

- Performance and utilization statistics for the replication host
  - CPUUtilization, FreeableMemory, FreeStorageSpace, WriteIOPS...

- **Replication Task Metrics**

- Statistics for Replication Task including incoming and committed changes, latency between the Replication Host and both source and target databases
  - FullLoadThroughputRowsSource, FullLoadThroughputRowsTarget, CDCThroughputRowsSource, CDCThroughputRowsTarget...

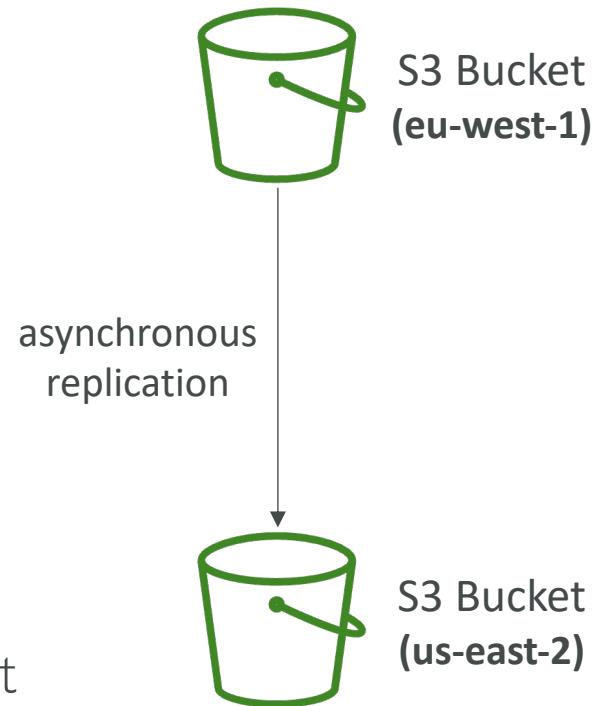
- **Table Metrics**

- Statistics for tables that are in the process of being migrated, including the number of insert, update, delete, and DDL statements completed

# Amazon S3 – Replication (CRR & SRR)



- Must enable Versioning in source and destination buckets
- Cross-Region Replication (CRR)
- Same-Region Replication (SRR)
- Buckets can be in different AWS accounts
- Copying is asynchronous
- Must give proper IAM permissions to S3
- Use cases:
  - CRR – compliance, lower latency access, replication across accounts
  - SRR – log aggregation, live replication between production and test accounts



# Amazon S3 – Replication (Notes)

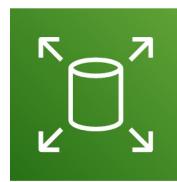
- After you enable Replication, only new objects are replicated
- Optionally, you can replicate existing objects using **S3 Batch Replication**
  - Replicates existing objects and objects that failed replication
- For DELETE operations
  - Can replicate delete markers from source to target (optional setting)
  - Deletions with a version ID are not replicated (to avoid malicious deletes)
- There is no “chaining” of replication
  - If bucket 1 has replication into bucket 2, which has replication into bucket 3
  - Then objects created in bucket 1 are not replicated to bucket 3

# Hybrid Cloud for Storage

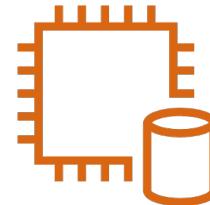
- AWS is pushing for "hybrid cloud"
  - Part of your infrastructure is on-premises
  - Part of your infrastructure is on the cloud
- This can be due to
  - Long cloud migrations
  - Security requirements
  - Compliance requirements
  - IT strategy
- S3 is a proprietary storage technology (unlike EFS / NFS), so how do you expose the S3 data on-premise?
- AWS Storage Gateway!

# AWS Storage Cloud Native Options

## BLOCK



Amazon EBS



EC2 Instance  
Store

## FILE



Amazon EFS

## OBJECT



Amazon S3



Glacier

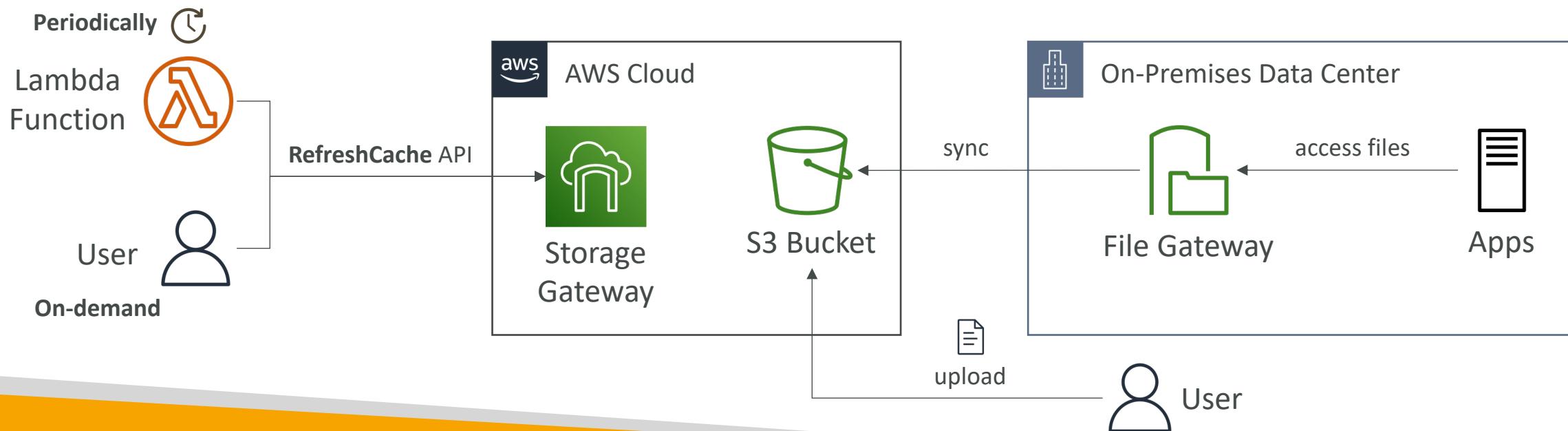
# AWS Storage Gateway

- Bridge between on-premise data and cloud data in S3
- Hybrid storage service to allow on-premises to seamlessly use the AWS Cloud
- Use cases: disaster recovery, backup & restore, tiered storage
- Types of Storage Gateway:
  - File Gateway
  - Volume Gateway
  - Tape Gateway
- No need to know the types at the exam



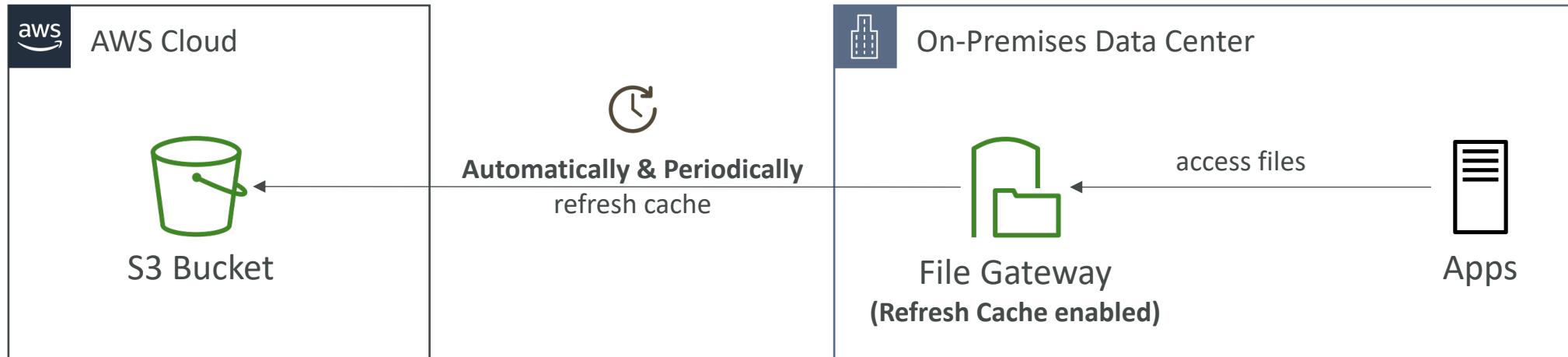
# File Gateway – RefreshCache API

- Storage Gateway updates the File Share Cache automatically when you write files to the File Gateway
- When you upload files directly to the S3 bucket, users connected to the File Gateway may not see the files on the File Share (accessing stale data)
- You need to invoke the RefreshCache API



# File Gateway – Automating Cache Refresh

- **Automated Cache Refresh** – a File Gateway feature that enables you to automatically refresh File Gateway cache to stay up to date with the changes in their S3 buckets
- Ensure that users are not accessing stale data on their file shares
- No need to manually and periodically invoke the RefreshCache API

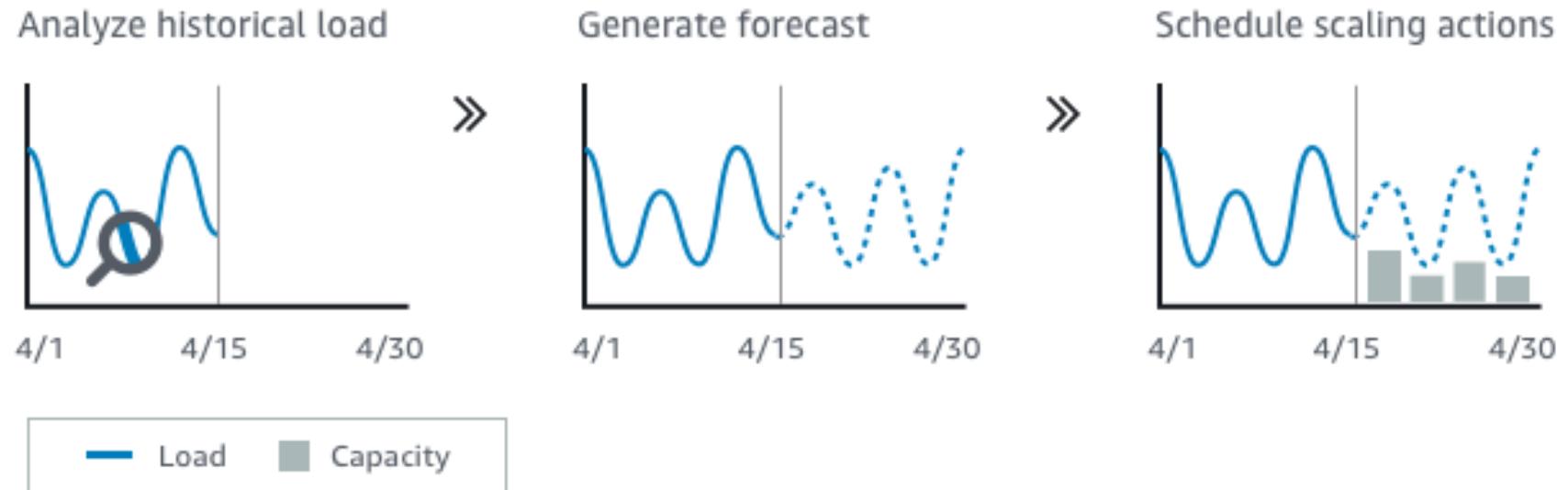


# Auto Scaling Groups – Scaling Policies

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

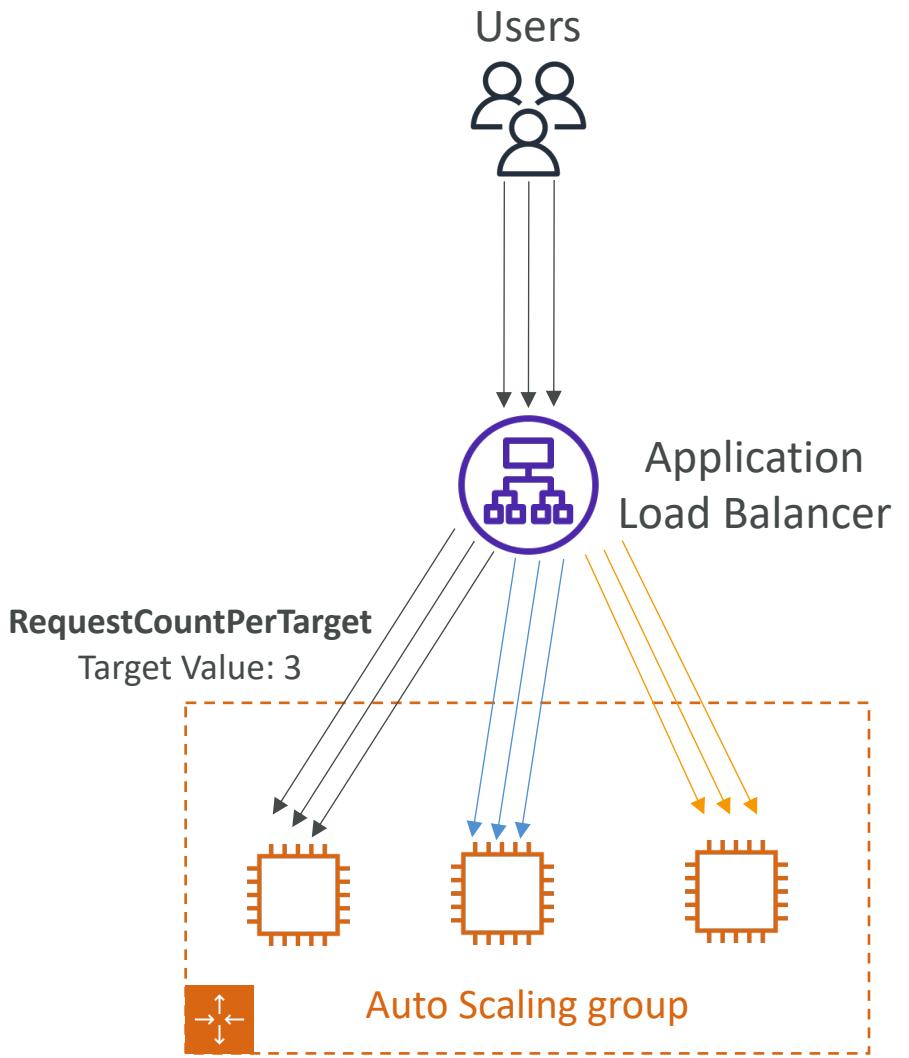
# Auto Scaling Groups – Scaling Policies

- Predictive scaling: continuously forecast load and schedule scaling ahead



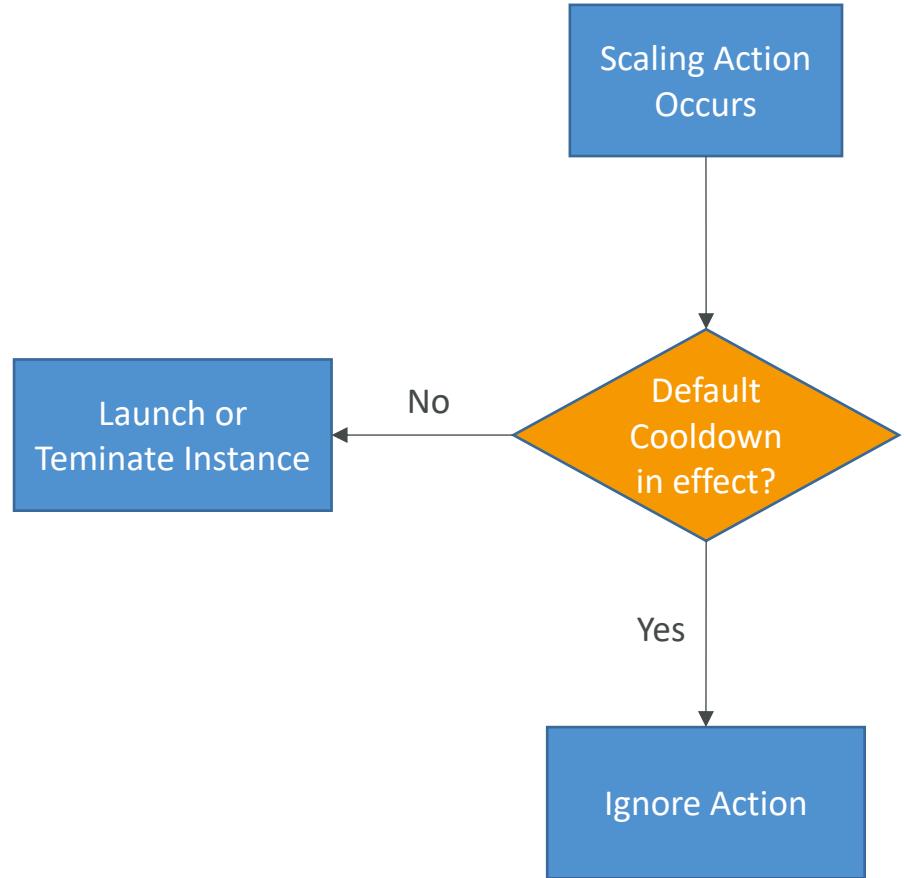
# Good metrics to scale on

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



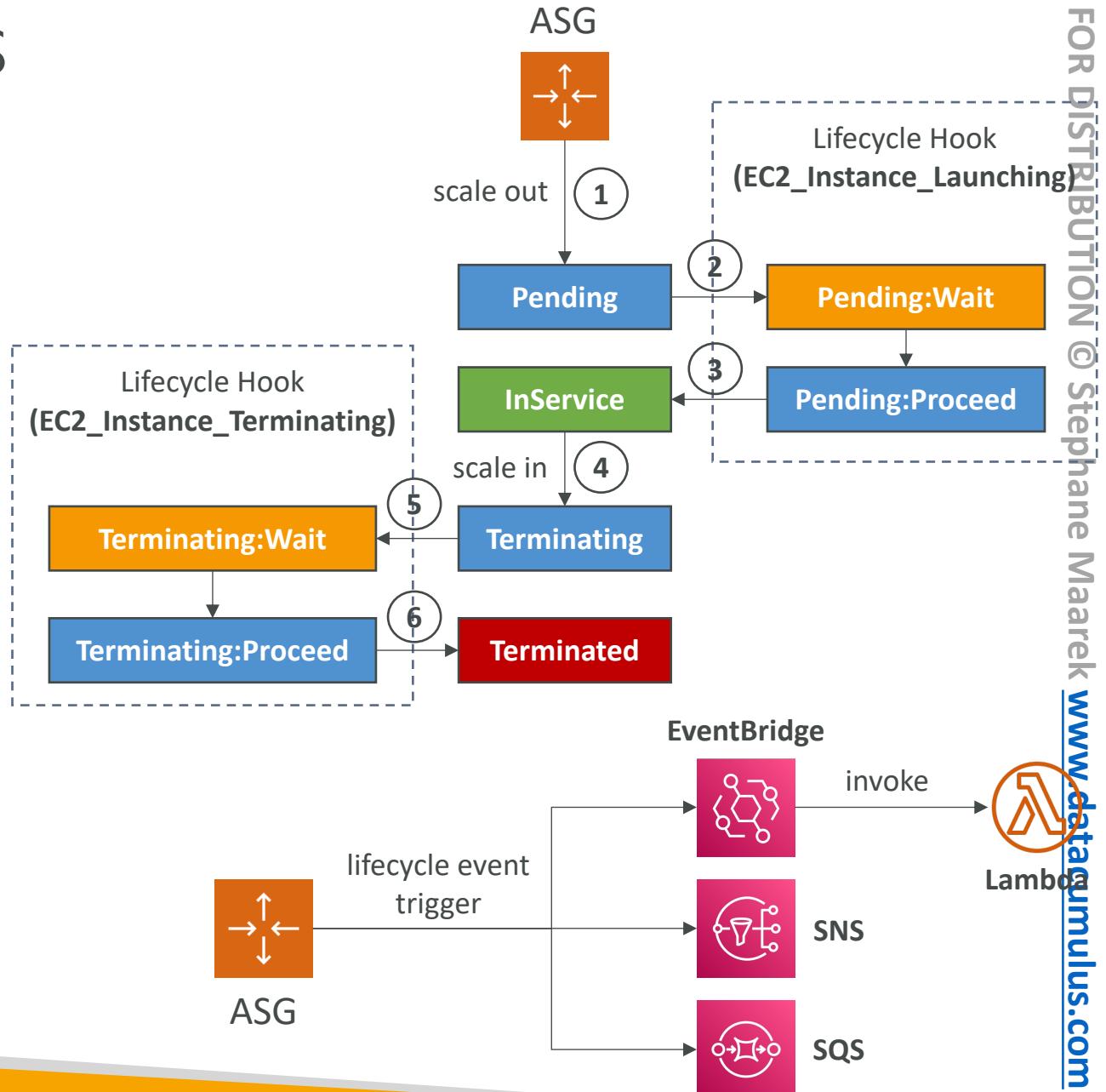
# Auto Scaling Groups - Scaling Cooldowns

- After a scaling activity happens, you are in the cooldown period (default 300 seconds)
- During the cooldown period, the ASG will not launch or terminate additional instances (to allow for metrics to stabilize)
- Advice: Use a ready-to-use AMI to reduce configuration time in order to be serving request faster and reduce the cooldown period

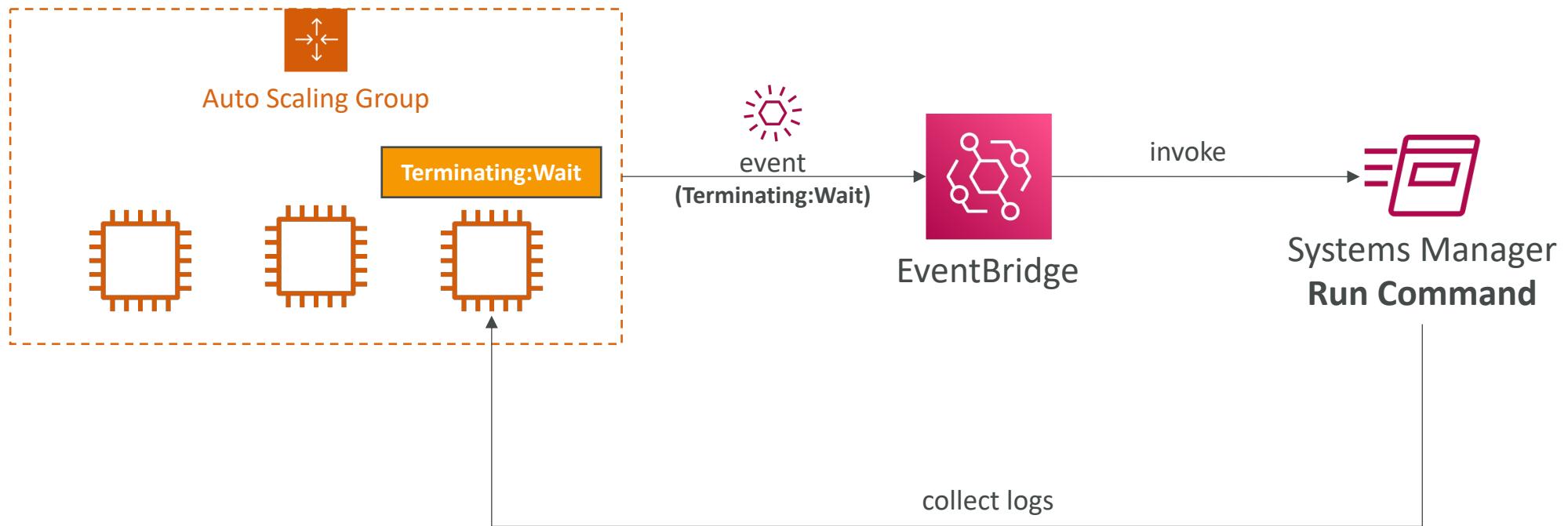


# ASG – Lifecycle Hooks

- By default, as soon as an instance is launched in an ASG it's in service
- You can perform extra steps before the instance goes in service (Pending state)
  - Define a script to run on the instances as they start
- You can perform some actions before the instance is terminated (Terminating state)
  - Pause the instances before they're terminated for troubleshooting
- Use cases: cleanup, log extraction, special health checks
- Integration with EventBridge, SNS, and SQS

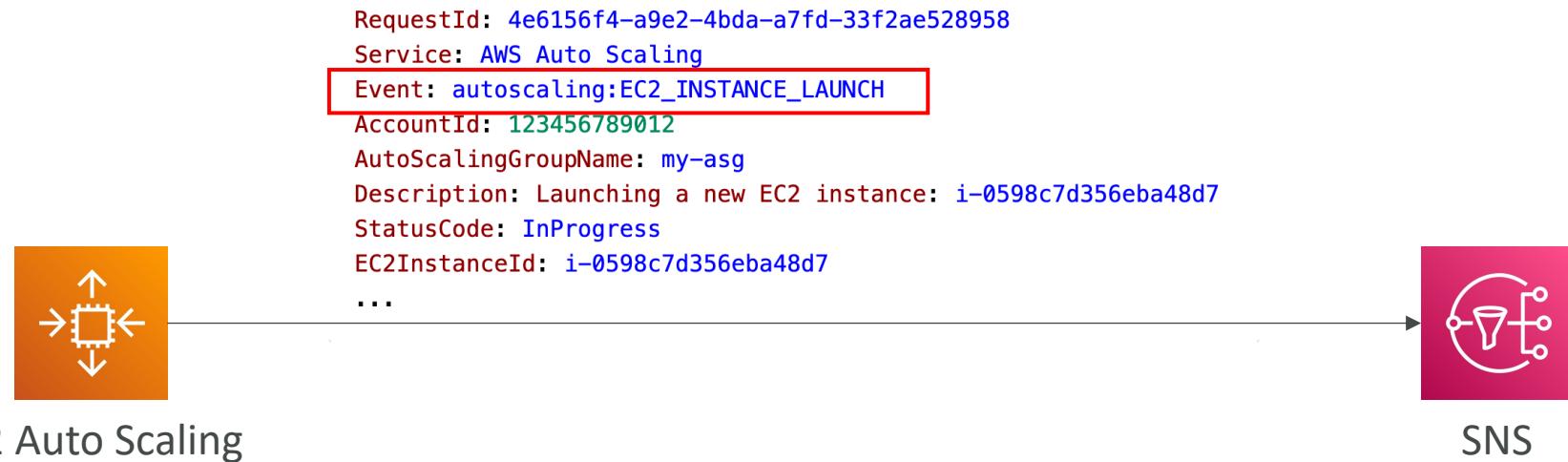


# ASG – Lifecycle Hooks



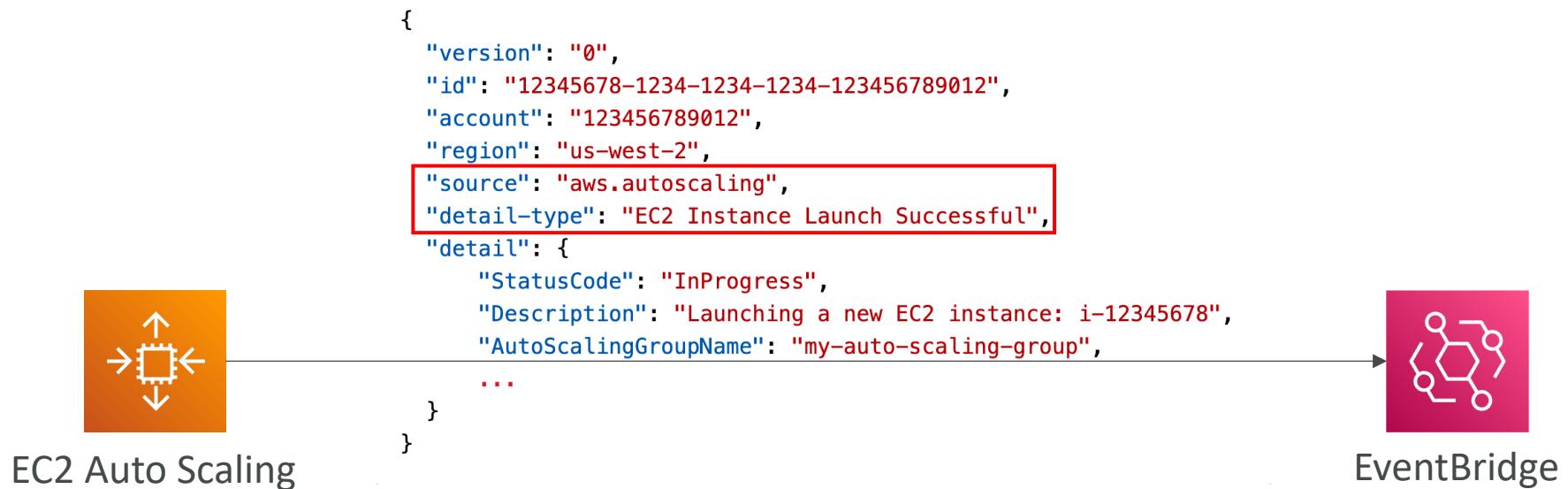
# ASG – SNS Notifications

- ASG supports sending SNS notifications for the following events:
  - autoscaling:EC2\_INSTANCE\_LAUNCH
  - autoscaling:EC2\_INSTANCE\_LAUNCH\_ERROR
  - autoscaling:EC2\_INSTANCE\_TERMINATE
  - autoscaling:EC2\_INSTANCE\_TERMINATE\_ERROR



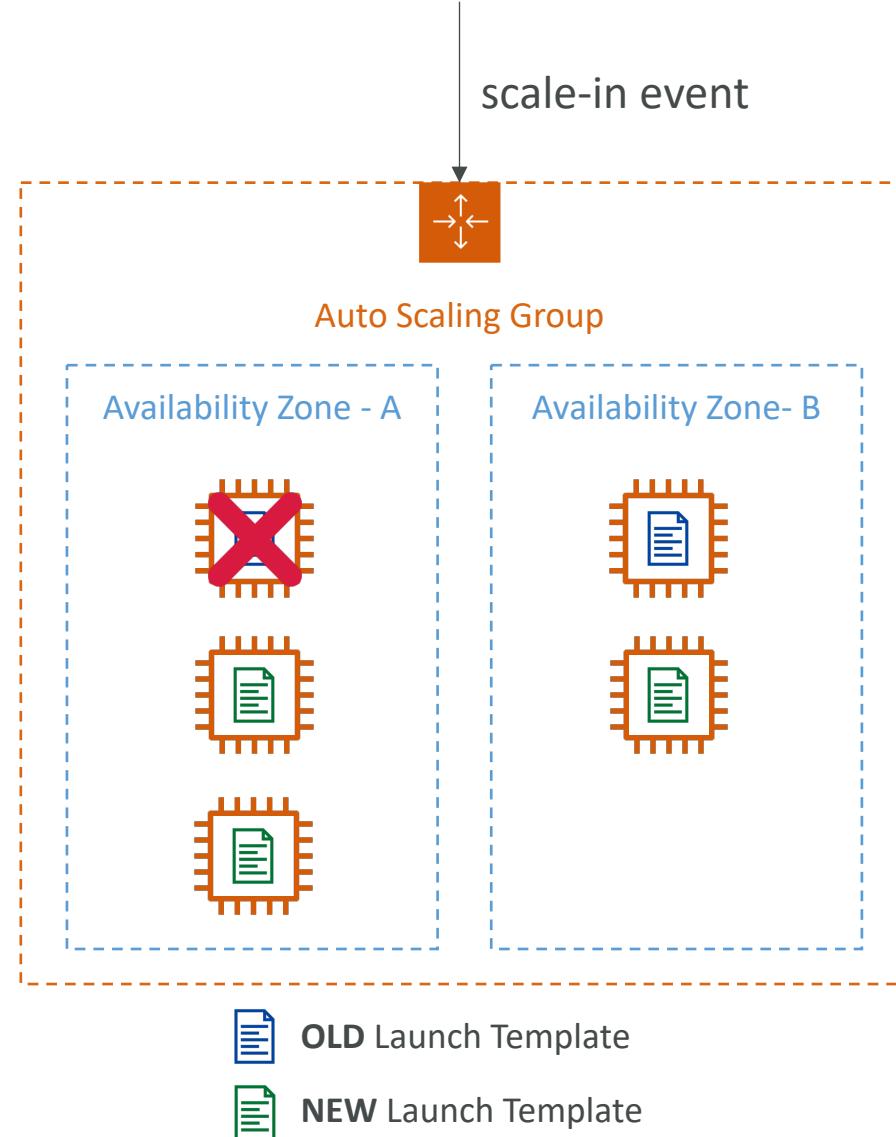
# ASG – EventBridge Events

- You can create Rules that match the following ASG events:
  - EC2 Instance Launching, EC2 Instance Launch Successful/Unsuccessful
  - EC2 Instance Terminating, EC2 Instance Terminate Successful/Unsuccessful
  - EC2 Auto Scaling Instance Refresh Checkpoint Reached
  - EC2 Auto Scaling Instance Refresh Started, Succeeded, Failed, Cancelled



# ASG – Termination Policies

- Determine which instances to terminates first during scale-in events, Instance Refresh, and AZ Rebalancing
- **Default Termination Policy**
  - Select AZ with more instances
  - Terminate instance with oldest Launch Template or Launch Configuration
  - If instances were launched using the same Launch Template, terminate the instance that is closest to the next billing hour



# ASG – Different Termination Policies

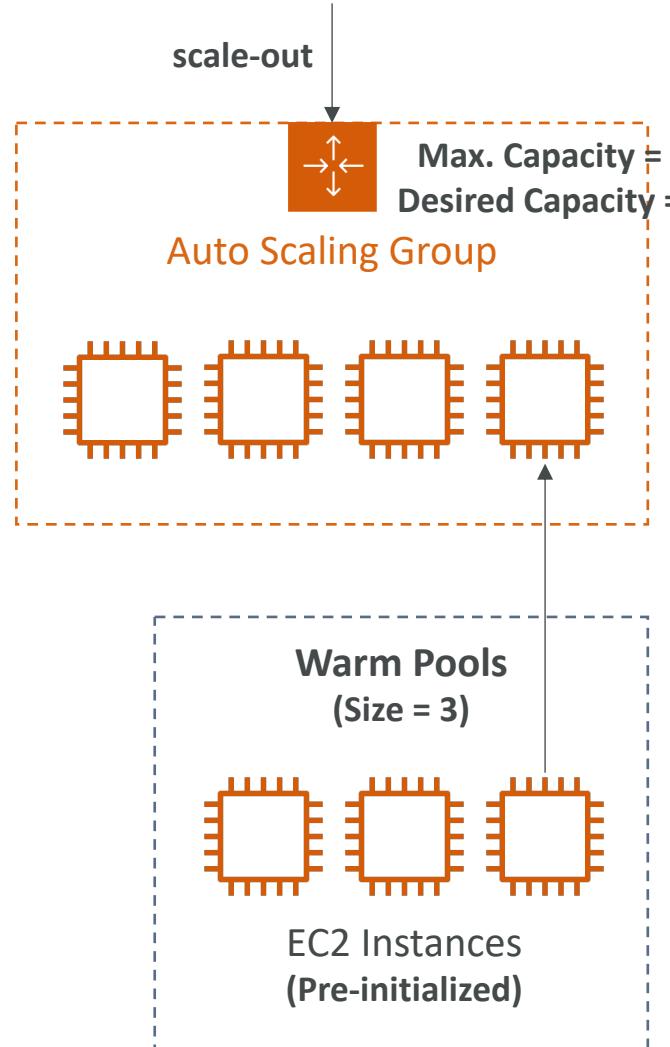
- **Default** – terminates instances according to Default Termination Policy
- **AllocationStrategy** – terminates instances to align the remaining instances to the Allocation Strategy (e.g., lowest-price for Spot Instances, or lower priority On-Demand Instances)
- **OldestLaunchTemplate** – terminates instances that have the oldest Launch Template
- **OldestLaunchConfiguration** – terminates instances that have the oldest Launch Configuration
- **ClosestToNextInstanceHour** – terminates instances that are closest to the next billing hour
- **NewestInstance** – terminates the newest instance (testing new launch template)
- **OldestInstance** – terminates the oldest instance (upgrading instance size, not launch template)
- **Note:** you can use one or more policies and specify the evaluation order
- **Note:** can define **Custom Termination Policy** backed by a Lambda function

# ASG – Scale-out Latency Problem

- When an ASG scales out, it tries to launch instances as fast as possible
- Some applications contain a lengthy unavoidable latency that exists at the application initialization/bootstrap layer (several minutes or more)
- Processes that can only happen at initial boot: applying updates, data or state hydration, running configuration scripts...
- Solution was to over-provision compute resources to absorb unexpected demand increases (increased cost) or use Golden Images to try to reduce boot time
- New solution: ASG Warm Pools !!

# ASG – Warm Pools

- Reduces scale-out latency by maintaining a pool of pre-initialized instances
- In a scale-out event, ASG uses the pre-initialized instances from the Warm Pool instead of launching new instances
- Warm Pool Size Settings
  - Minimum warm pool size (always in the warm pool)
  - Max prepared capacity = Max capacity of ASG (default)
  - OR Max prepared capacity = Set number of instances
- Warm Pool Instance State – what state to keep your Warm Pool instances in after initialization  
**(Running, Stopped, Hibernated)**
- Warm Pools instances don't contribute to ASG metrics that affect Scaling Policies



# ASG – Warm Pools Pricing: m5.large

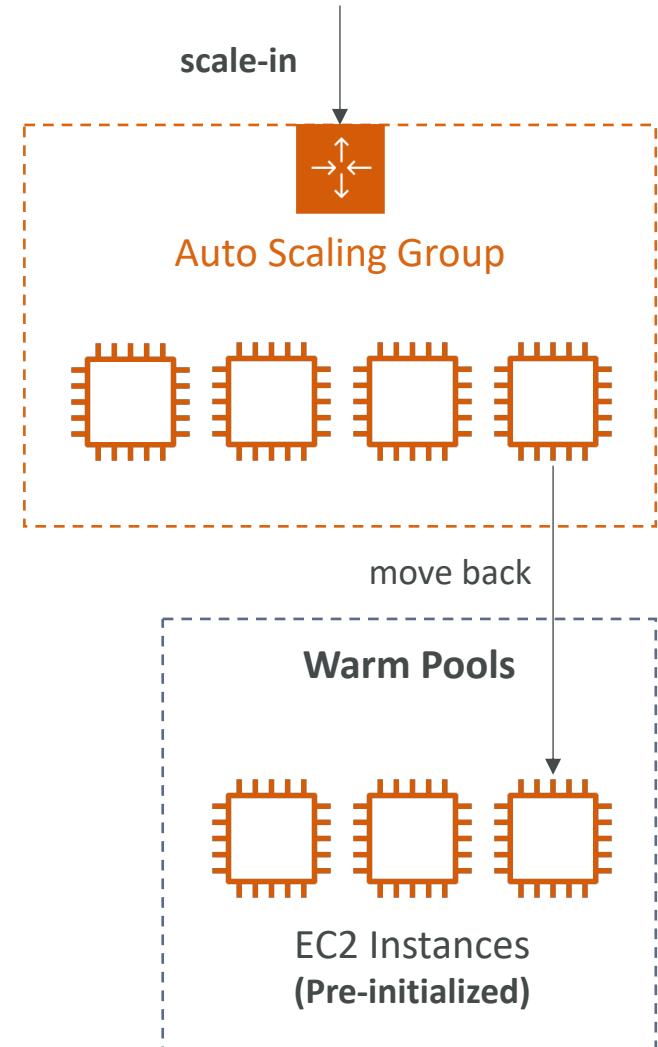
- If we "over provision an EC2 instance" in an ASG
- Running Cost =  $\$0.096/\text{hour} * 24 \text{ hours/day} * 30 \text{ days} = \$69.12$   
+ EBS charges
- If the EC2 instance is stopped, we only pay for the attached EBS volume
- EBS Volume Cost =  $\$0.10/\text{GB-month} * 10\text{GB} = \$1.00$

# ASG – Warm Pools – Instance States

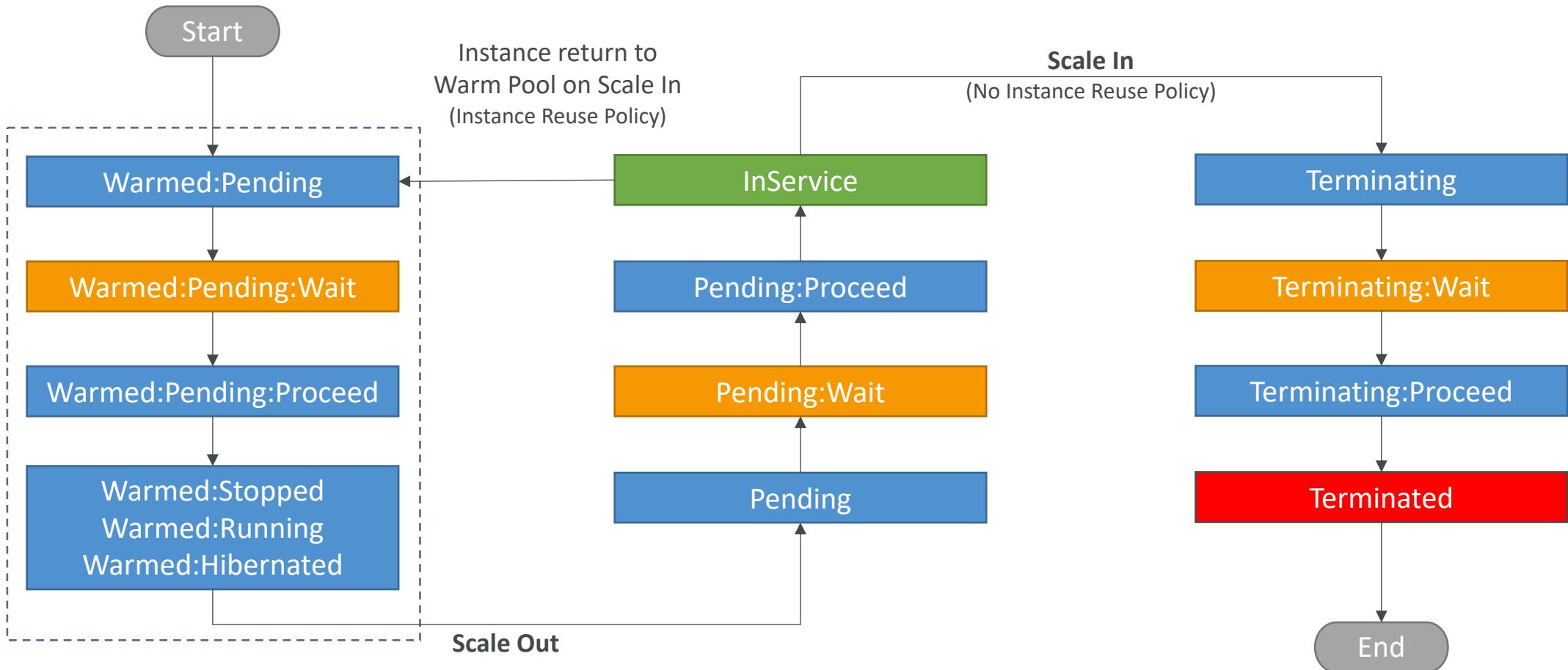
	<b>Running</b>	<b>Stopped</b>	<b>Hibernated</b>
<b>Scale Out Delay</b>	<b>Faster</b> – immediately available to accept traffic	<b>Slower</b> – need to be started before traffic can be served which adds some delay	<b>Medium</b> – pre-initialized EC2 instances (disk, memory...)
<b>Start Up Delay</b>	<b>Lower</b> – EC2 instances are already running	<b>Slower</b> – needs to go through the application startup process which may need some time especially if there are many application components to initialize	<b>Medium</b> – faster startup process than EC2 instances in the Stopped state because components and applications are already initialized in memory
<b>Costs</b>	<b>Higher</b> – incur costs for their usage even when they're not serving traffic	<b>Lower Cost</b> – do not incur running costs, only pay for the attached resources, more cost effective	<b>Lower Cost</b> – do not incur running costs, only pay for the attached resources, more cost effective  The hibernation state must be able to fit onto the EBS volume (might need a bigger volume)

# ASG – Warm Pools – Instance Reuse Policy

- By default, ASG terminates instances when ASG scales in, then it launches new instances into the Warm Pool
- Instance Reuse Policy allows you to return instances to the Warm Pool when a scale-in event happens

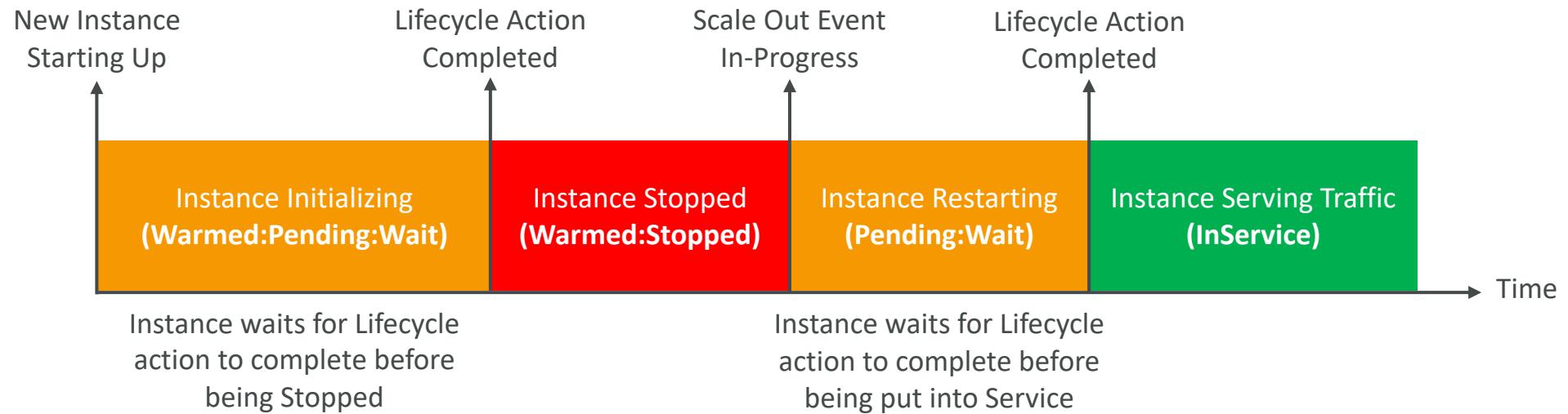


# ASG – Warm Pools – Lifecycle Hooks

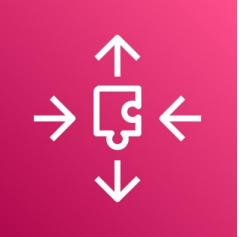


# ASG – Warm Pools – Lifecycle Hooks

## Scale Out Event Summary

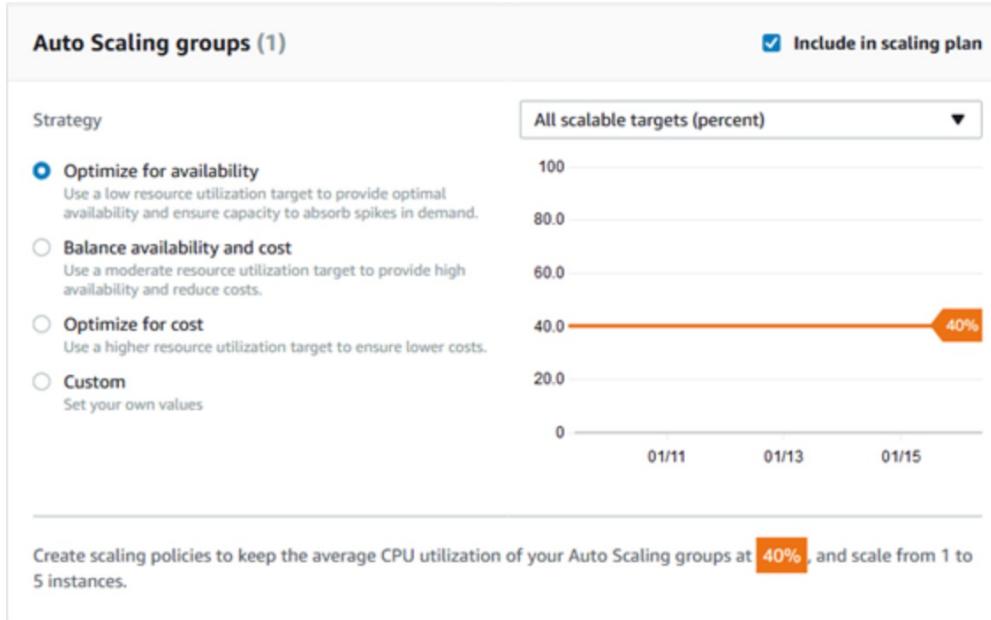
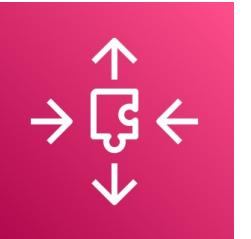


# AWS Application Auto Scaling

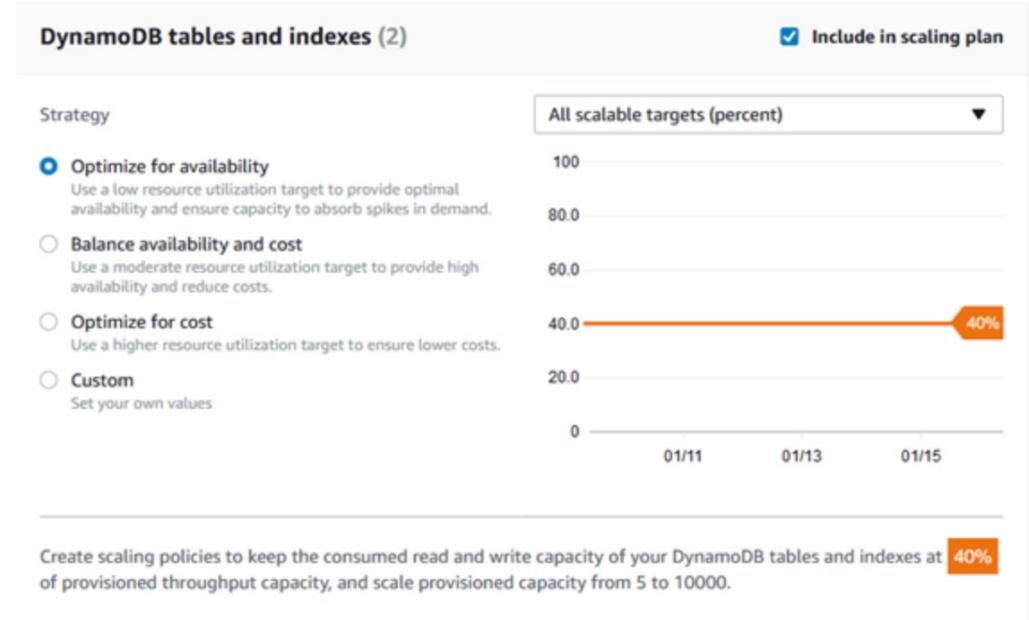


- Monitors your apps and automatically adjusts capacity to maintain steady, predictable performance at lowest cost
- Setup scaling for multiple resources across multiple services from a single place (no need to navigate across different services)
- Point to your app and select the services and resources you want to scale (no need to setup alarms and scaling actions for each service)
- Search for resources/services using CloudFormation Stack, Tags, or EC2 ASG
- Build **Scaling Plans** to automatically add/remove capacity from your resources in real-time as demand changes
- Supports Target Tracking, Step, and Scheduled Scaling Policies

# AWS Application Auto Scaling



## EC2 Auto Scaling Groups



## DynamoDB Tables

# AWS Application Auto Scaling – Integrated AWS Services



AppStream 2.0  
Fleets



Aurora  
Replicas



Comprehend  
Document classification and  
Entity recognizer endpoints



DynamoDB  
Tables & GSI



ECS  
Services



ElastiCache for Redis  
Replication Groups



EMR  
Clusters



KeySpaces  
Tables



Lambda  
Provisioned Concurrency



MSK  
Broker Storage



Neptune  
Clusters



SageMaker  
Endpoint Variants



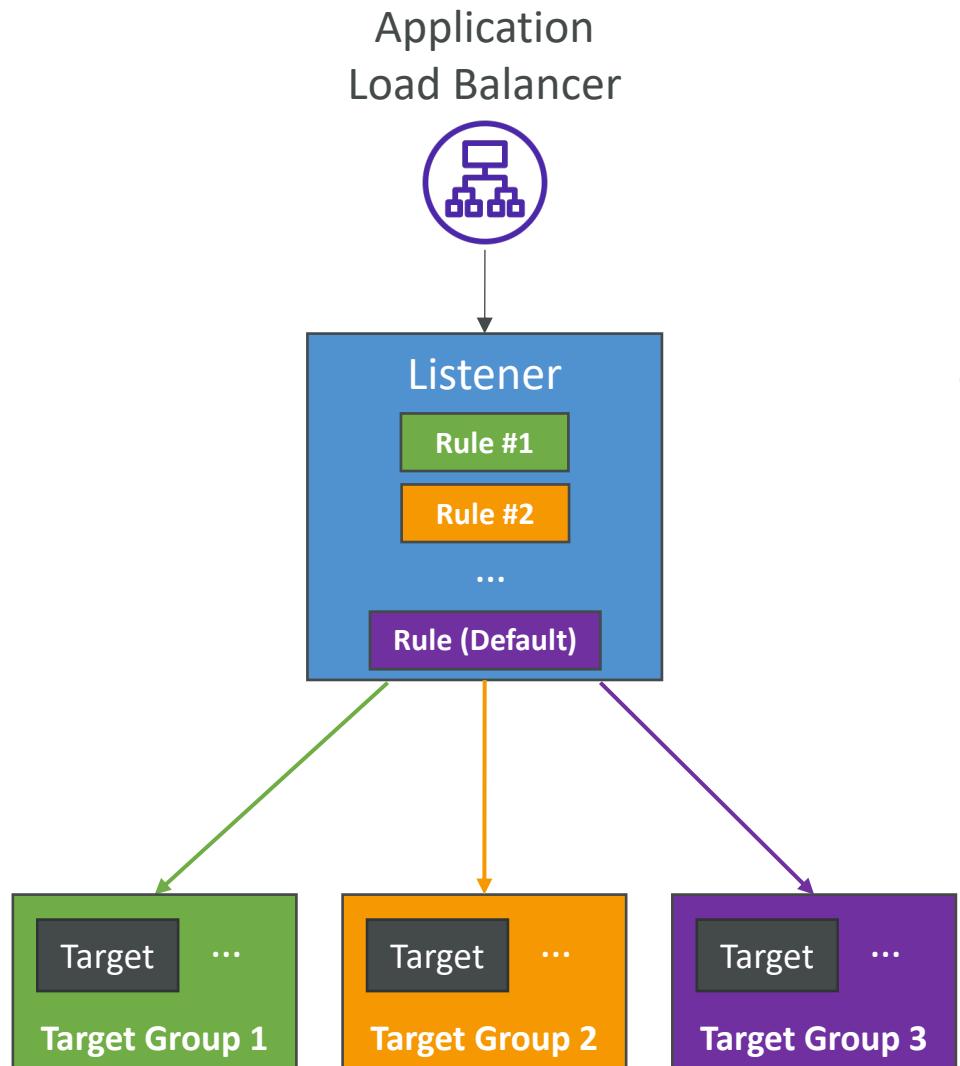
Spot Fleet  
Requests



Custom  
Resources

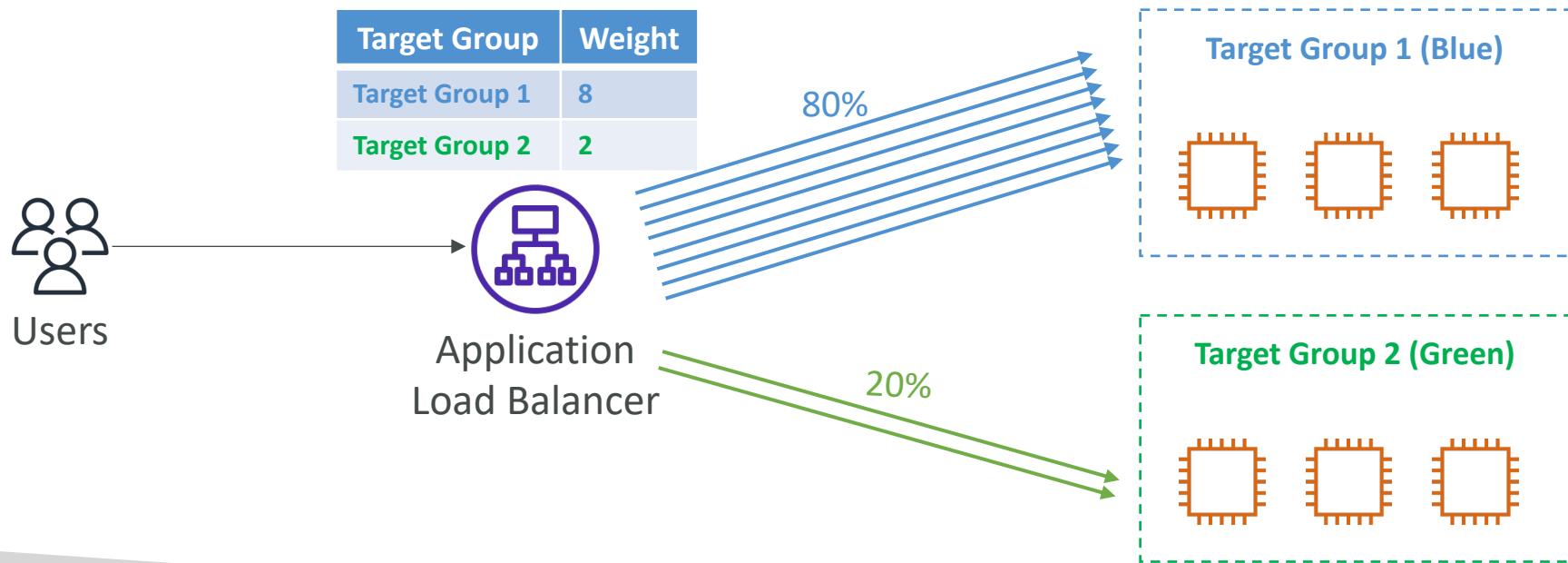
# ALB – Listener Rules

- Processed in order (with Default Rule)
- Supported Actions (forward, redirect, fixed-response)
- Rule Conditions:
  - host-header
  - http-request-method
  - path-pattern
  - source-ip
  - http-header
  - query-string



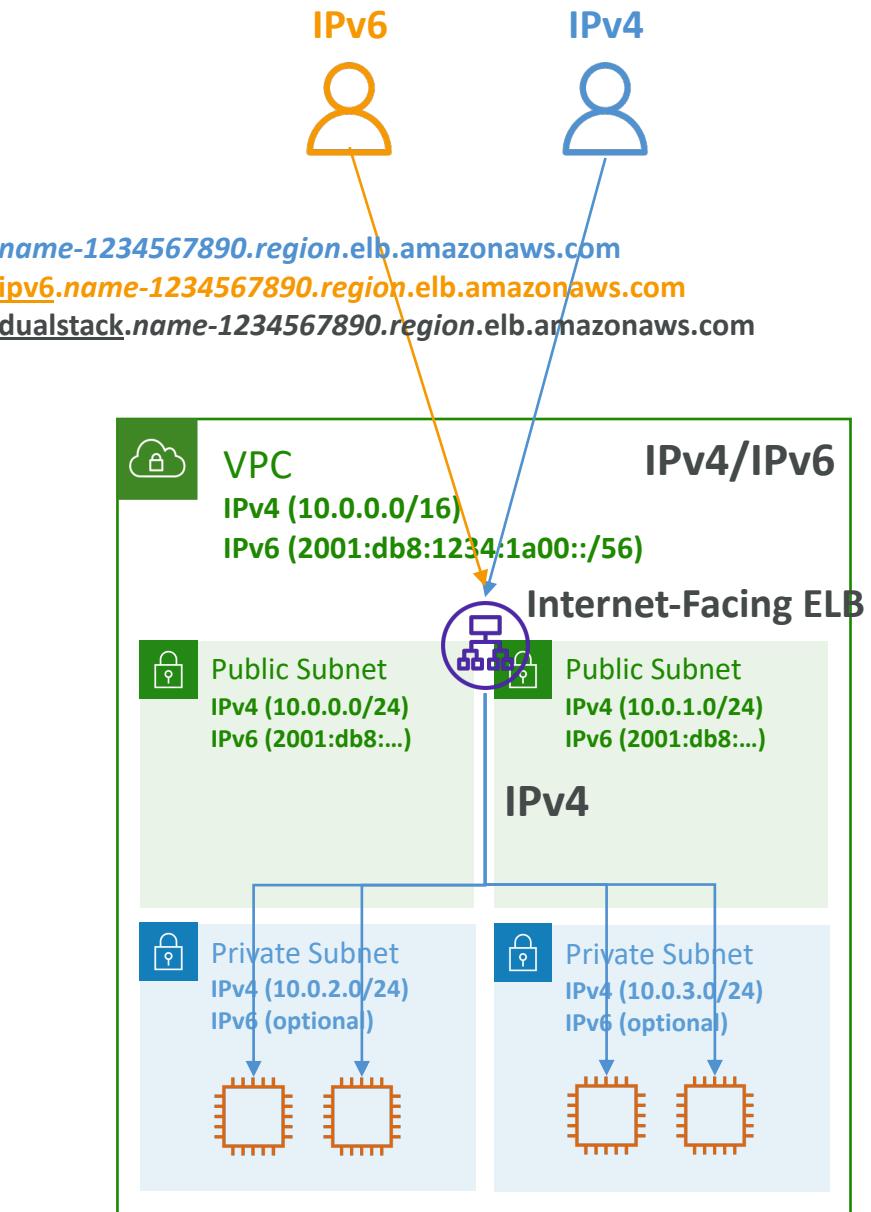
# Target Group Weighting

- Specify weight for each Target Group on a single Rule
- Example: multiple versions of your app, blue/green deployment
- Allows you to control the distribution of the traffic to your applications



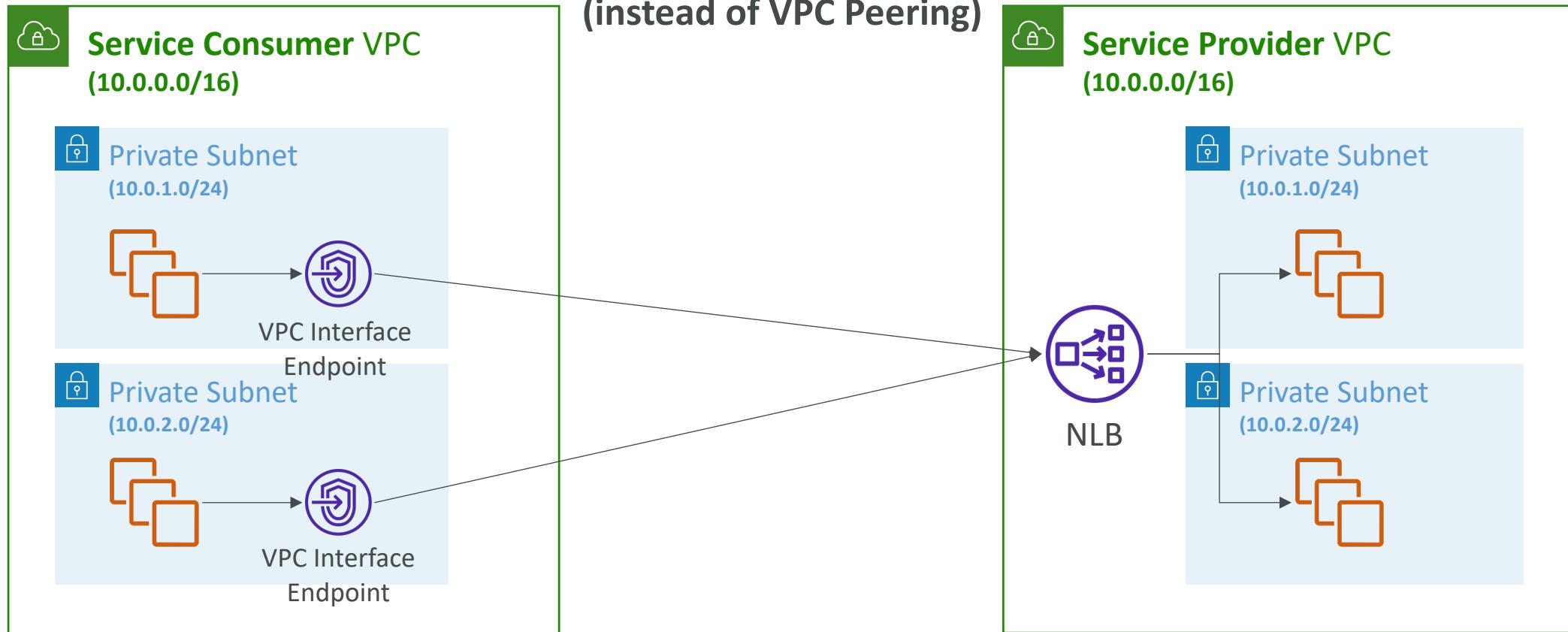
# ELB – DualStack Networking

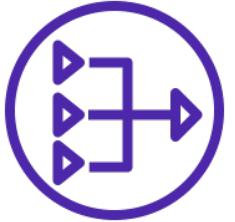
- Allows clients communicate with the ELB using both IPv4 and IPv6
- Supports both ALB and NLB
- ALB and NLB can have mixed IPv4 and IPv6 targets in separate target groups.
- ELB DualStack ensures compatibility between client and target IP versions
  - IPv4 clients communicate with IPv4 targets, and IPv6 clients communicate with IPv6 targets.
  - If you only have IPv4 targets, the ELB automatically converts requests from IPv6 to IPv4
- **Note:** AZ must be added/enabled for instances to receive traffic



# NLB – PrivateLink Integration

**Exposing Service in VPCs with  
Overlapping IP addresses  
(instead of VPC Peering)**

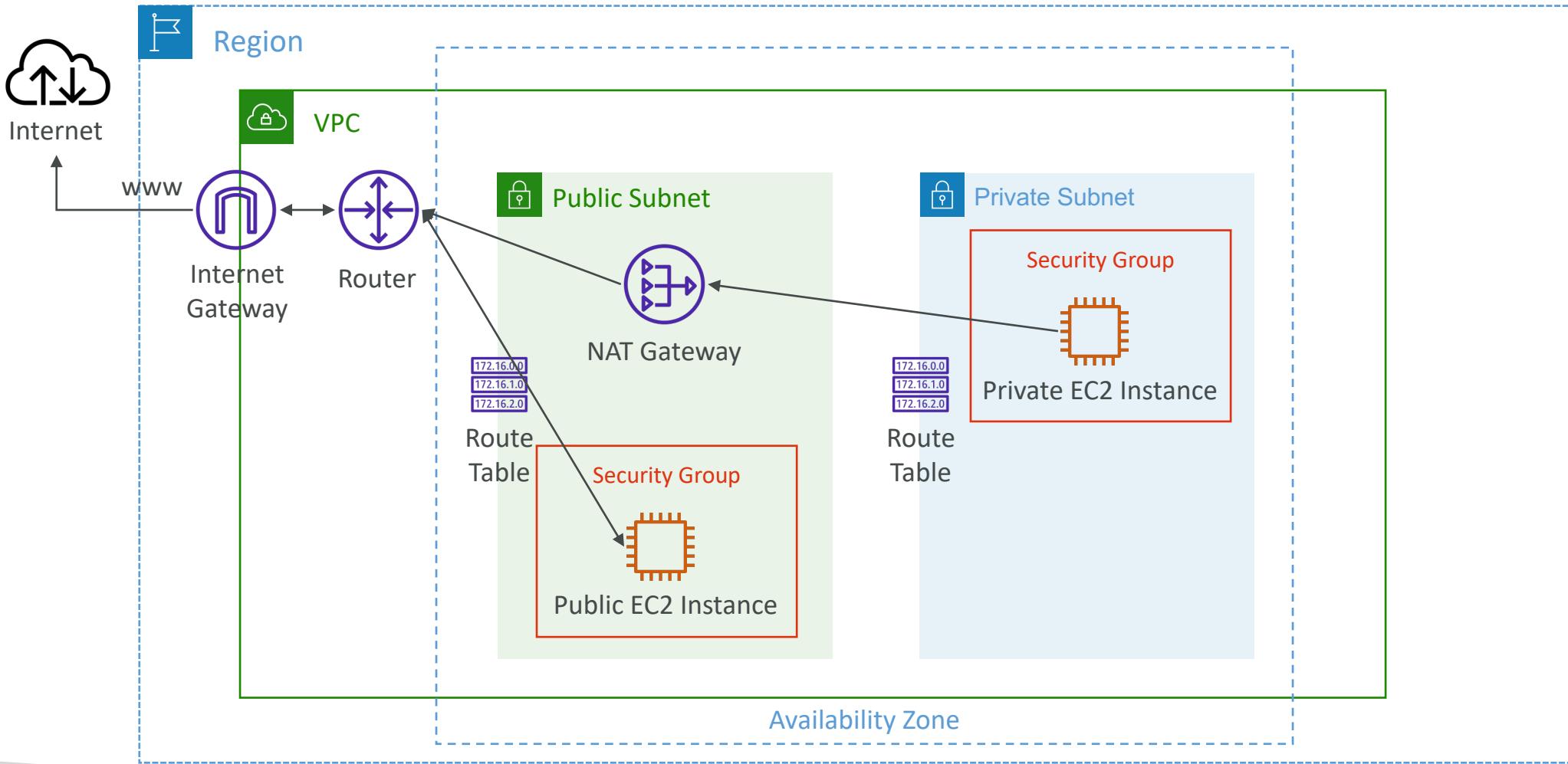




# NAT Gateway

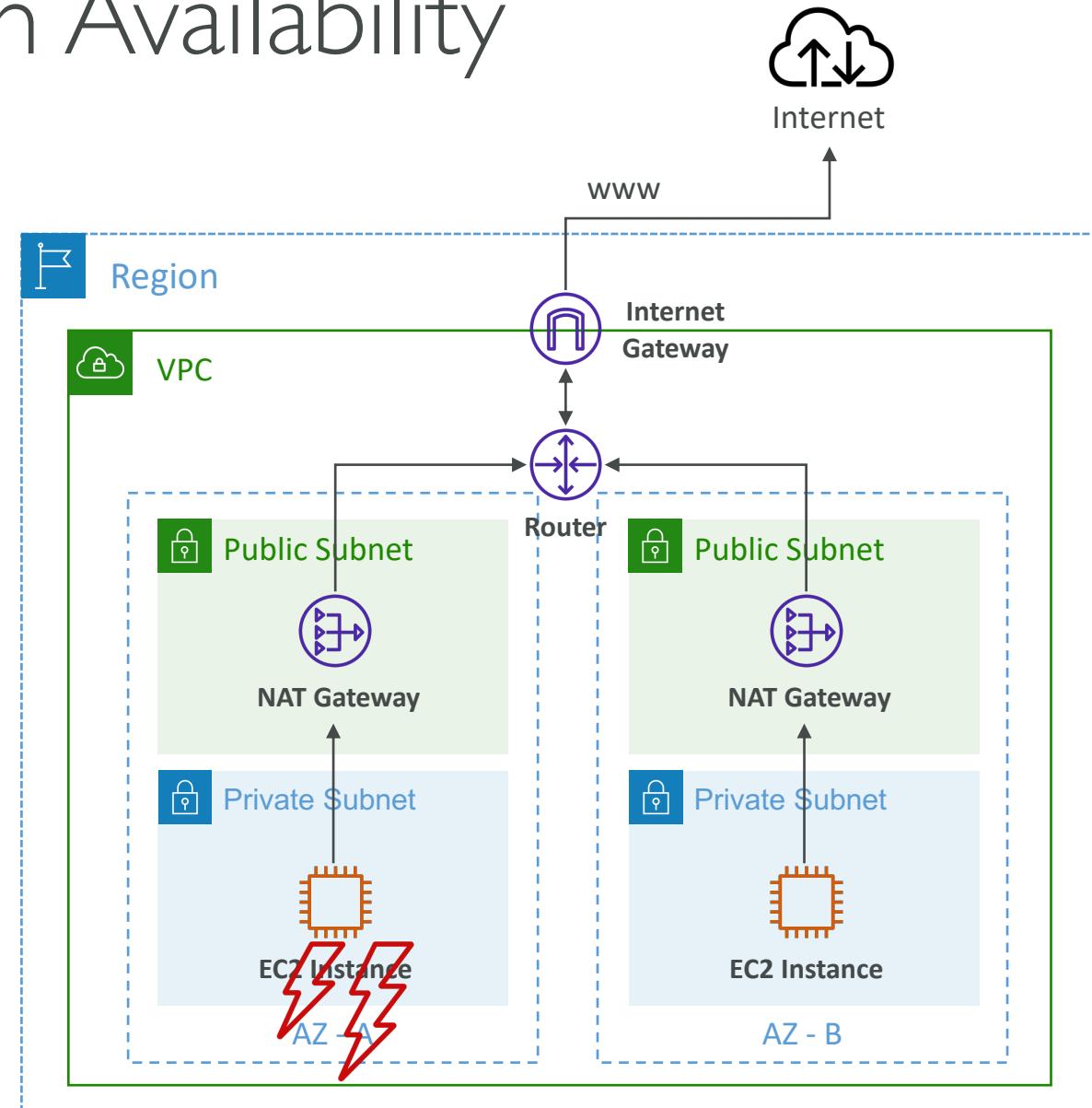
- AWS-managed NAT, higher bandwidth, high availability, no administration
- Pay per hour for usage and bandwidth
- NATGW is created in a specific Availability Zone, uses an Elastic IP
- Can't be used by EC2 instance in the same subnet (only from other subnets)
- Requires an IGW (Private Subnet => NATGW => IGW)
- 5 Gbps of bandwidth with automatic scaling up to 100 Gbps
- No Security Groups to manage / required

# NAT Gateway



# NAT Gateway with High Availability

- NAT Gateway is resilient within a single Availability Zone
- Must create multiple NAT Gateways in multiple AZs for fault-tolerance
- There is no cross-AZ failover needed because if an AZ goes down it doesn't need NAT



# NAT Gateway vs. NAT Instance

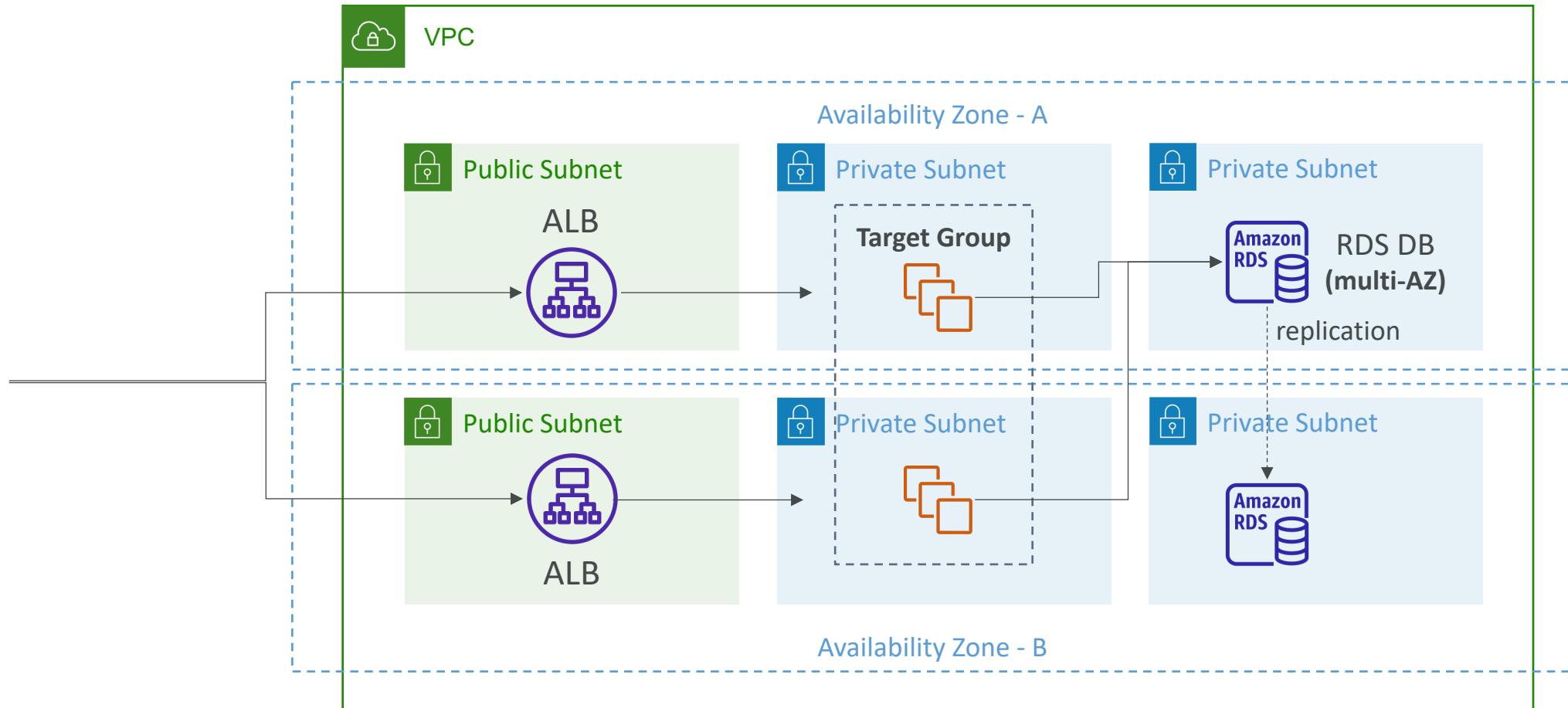
	NAT Gateway	NAT Instance
<b>Availability</b>	Highly available within AZ (create in another AZ)	Use a script to manage failover between instances
<b>Bandwidth</b>	Up to 100 Gbps	Depends on EC2 instance type
<b>Maintenance</b>	Managed by AWS	Managed by you (e.g., software, OS patches, ...)
<b>Cost</b>	Per hour & amount of data transferred	Per hour, EC2 instance type and size, + network \$
<b>Public IPv4</b>	✓	✓
<b>Private IPv4</b>	✓	✓
<b>Security Groups</b>	✗	✓
<b>Use as Bastion Host?</b>	✗	✓

More at: <https://docs.aws.amazon.com/vpc/latest/userguide/vpc-nat-comparison.html>

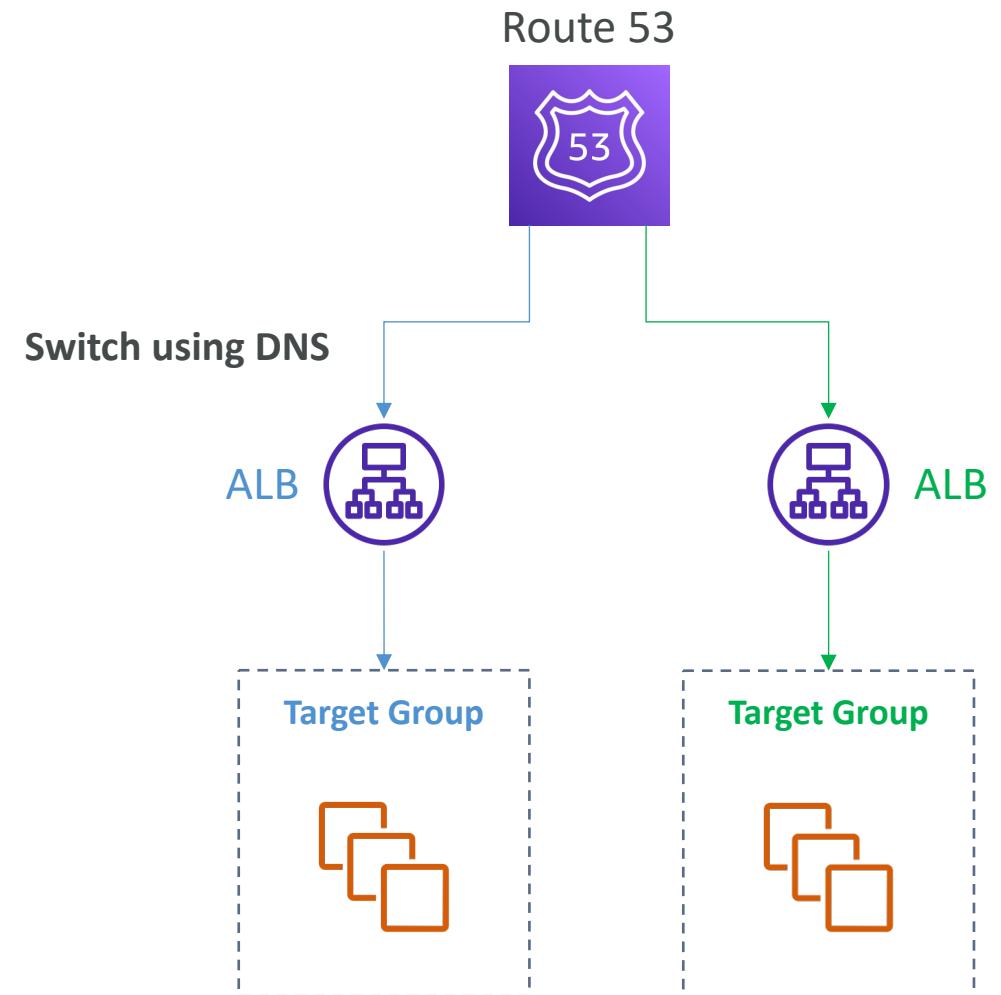
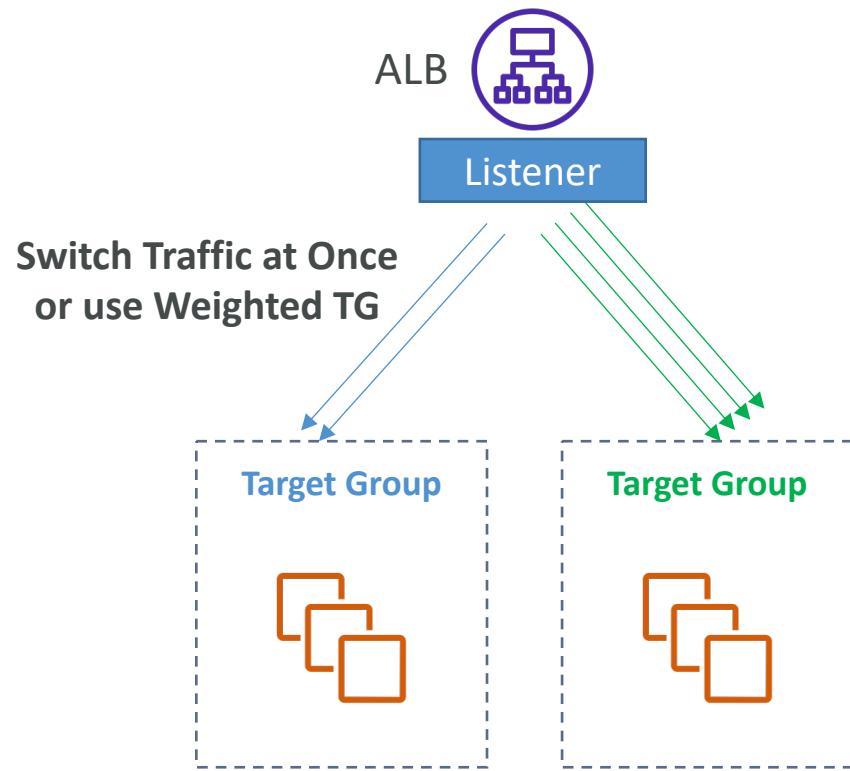
# Multi AZ in AWS

- Services where Multi-AZ must be enabled manually:
  - EFS, ELB, ASG, Beanstalk: assign AZ
  - RDS, ElastiCache: multi-AZ (synchronous standby DB for failovers)
  - Aurora:
    - data is stored automatically across multi-AZ
    - Can have multi-AZ for the DB itself (same as RDS)
  - OpenSearch (managed): multi master
  - Jenkins (self deployed): multi master
- Service where Multi-AZ is implicitly there:
  - S3 (except OneZone-Infrequent Access)
  - DynamoDB
  - All of AWS' proprietary, managed services

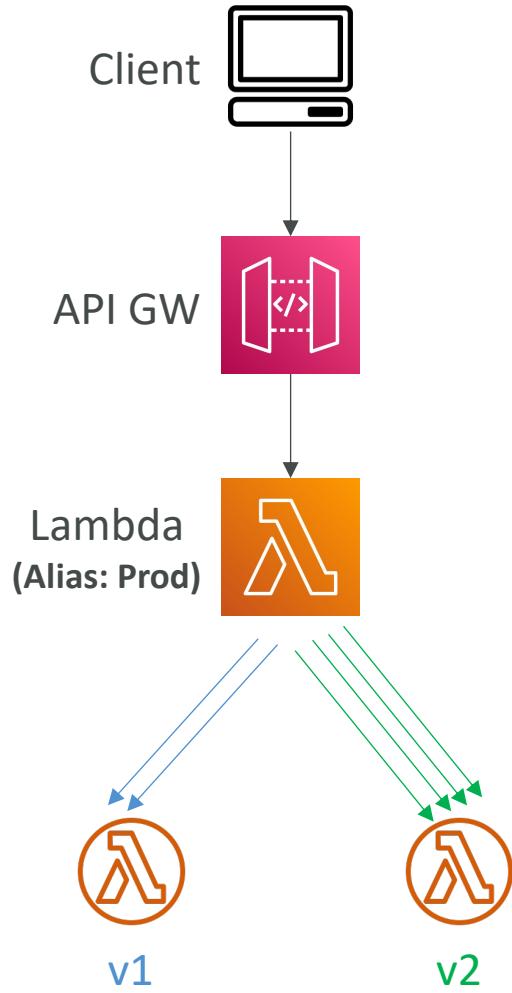
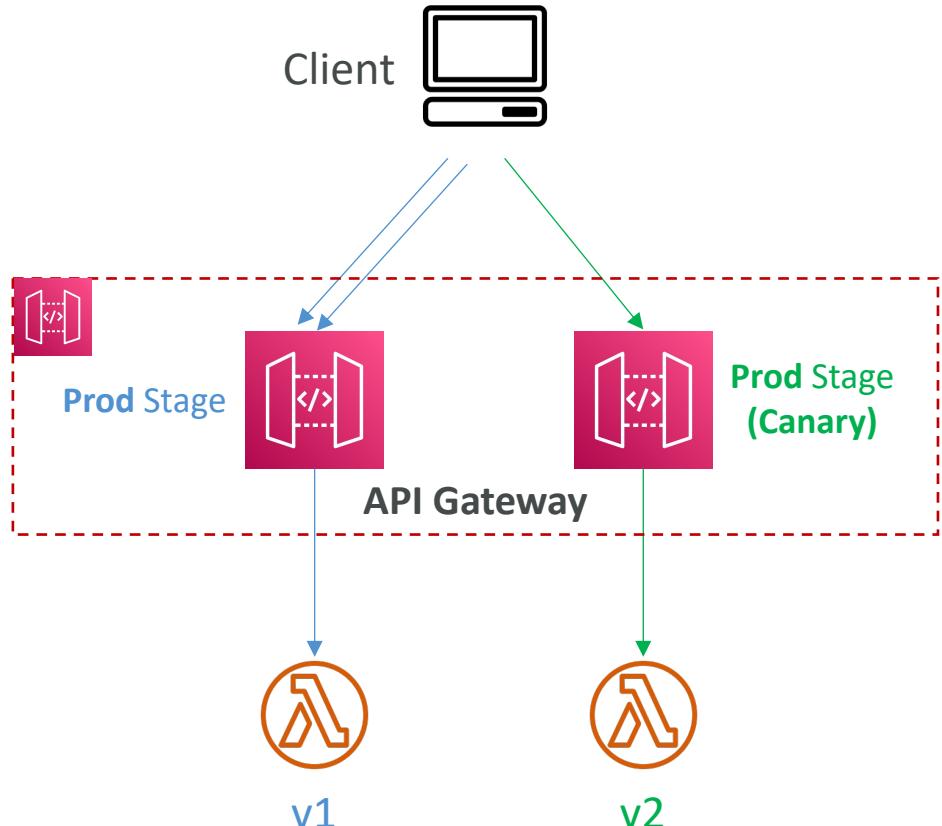
# Multi-AZ Architectures



# Blue-Green Architectures



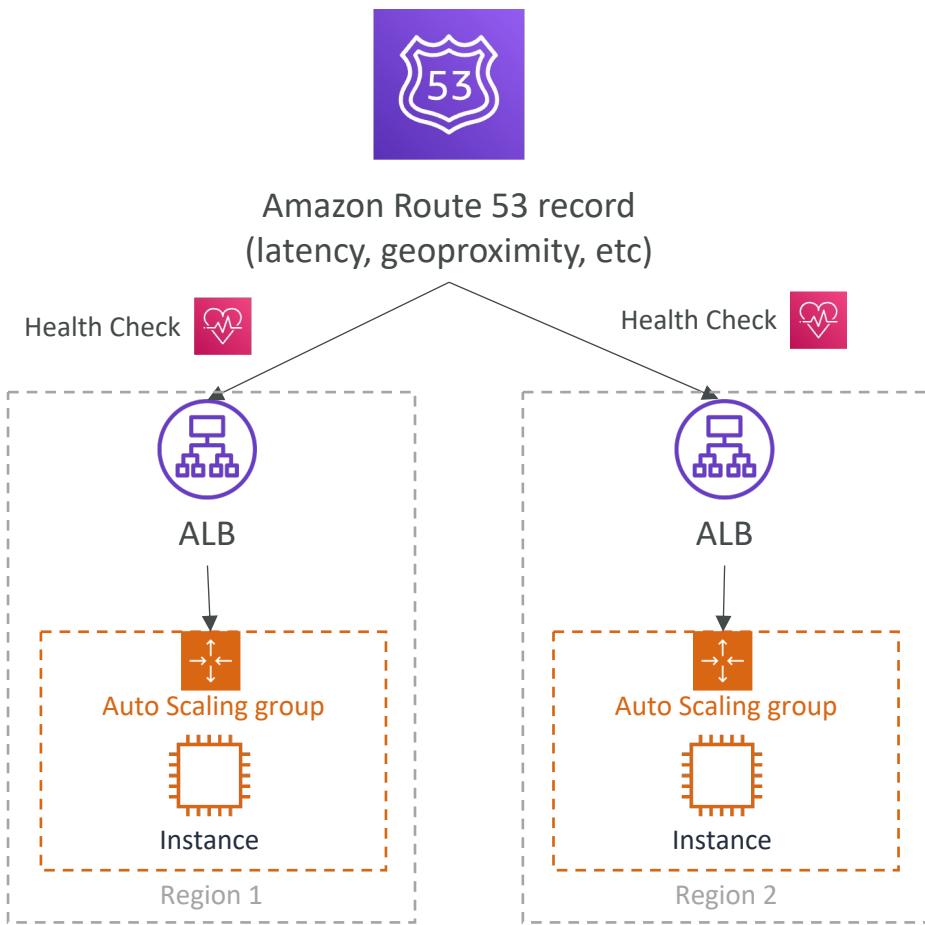
# Blue-Green Architectures



# Multi Region Services

- DynamoDB Global Tables (multi-way replication, enabled by Streams)
- AWS Config Aggregators (multi region & multi account)
- RDS Cross Region Read Replicas (used for Read & DR)
- Aurora Global Database (one region is master, other is for Read & DR)
- EBS volumes snapshots, AMI, RDS snapshots can be copied to other regions
- VPC peering to allow private traffic between regions
- Route53 uses a global network of DNS servers
- S3 Cross Region Replication
- CloudFront for Global CDN at the Edge Locations
- Lambda@Edge for Global Lambda function at Edge Locations (A/B testing)

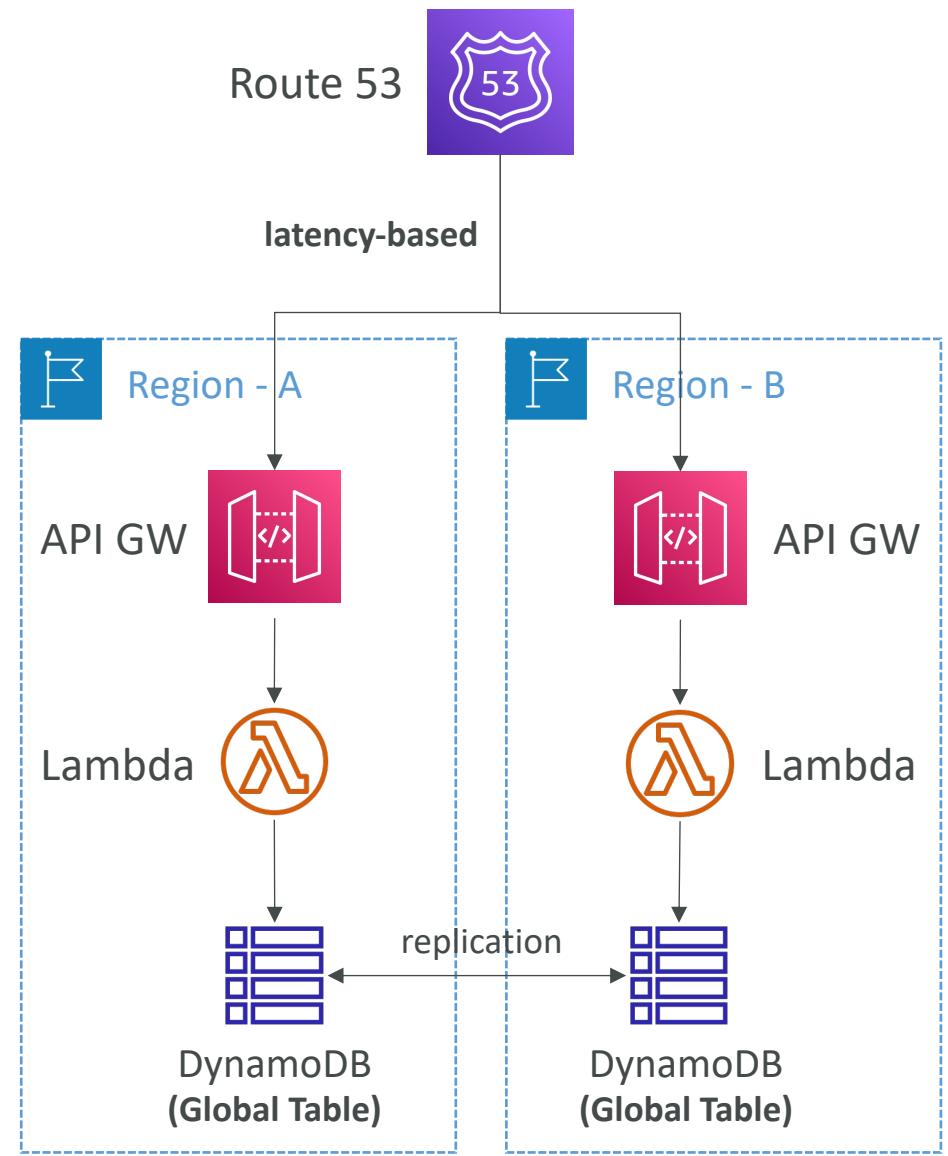
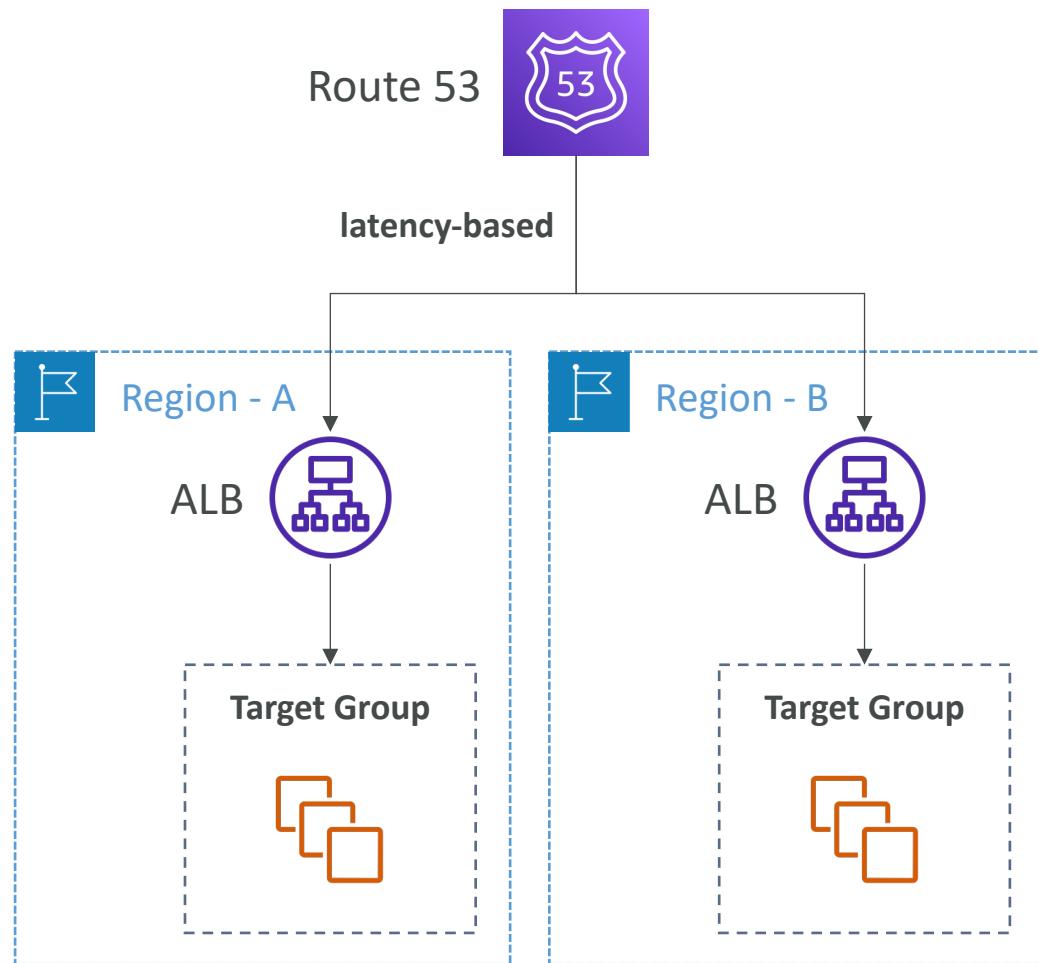
# Multi Region with Route 53



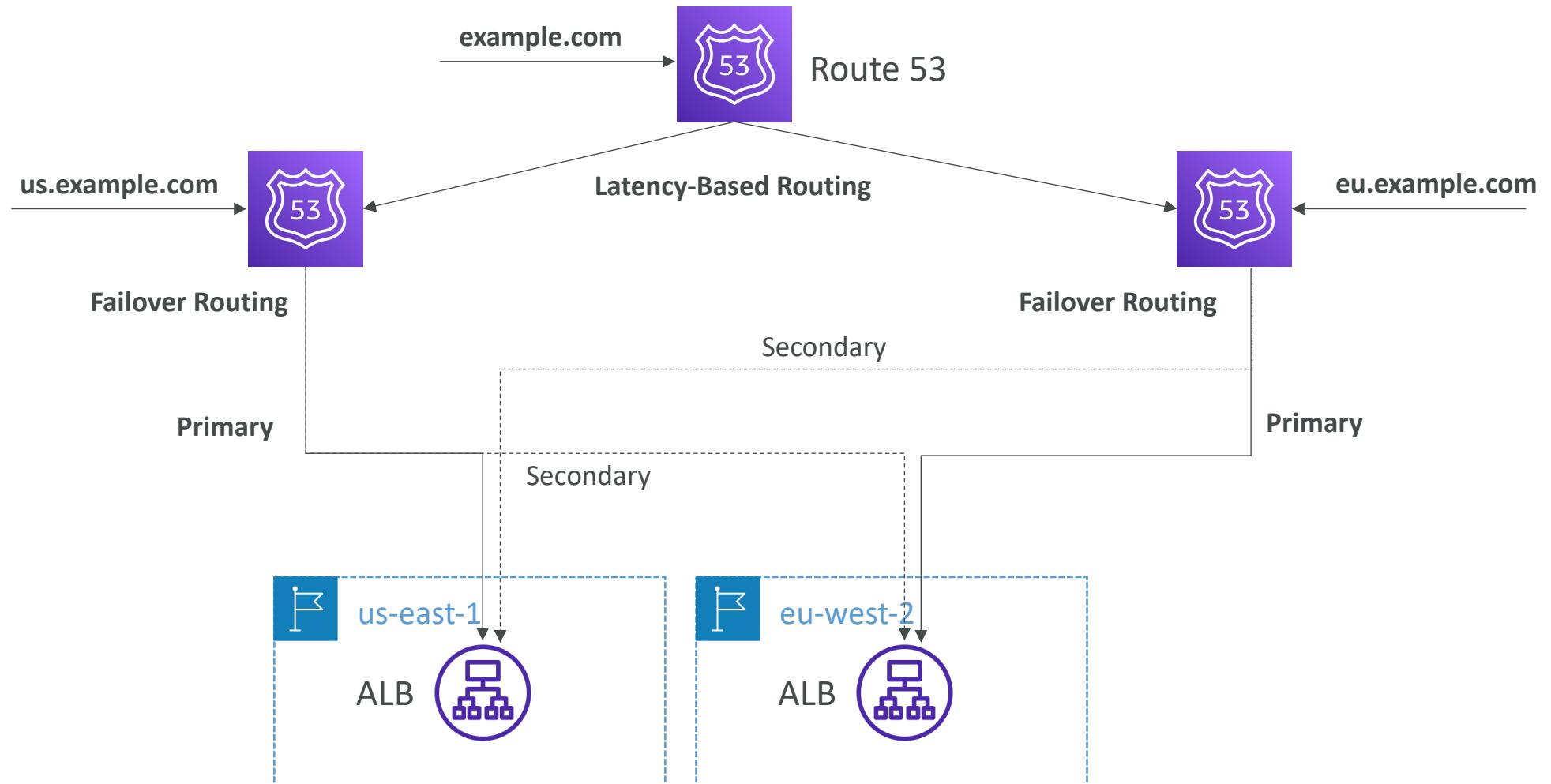
- Health Check => automated DNS failovers:
  1. Health checks that monitor an endpoint (application server, other AWS resource)  
**Helpful to create a /health route**
  2. Health checks that monitor other health checks (calculated health checks)
  3. Health checks that monitor CloudWatch alarms (full control !!) – e.g. throttles of DynamoDB, custom metrics, etc

Health Checks are integrated with CW metrics

# Multi-Region Architectures



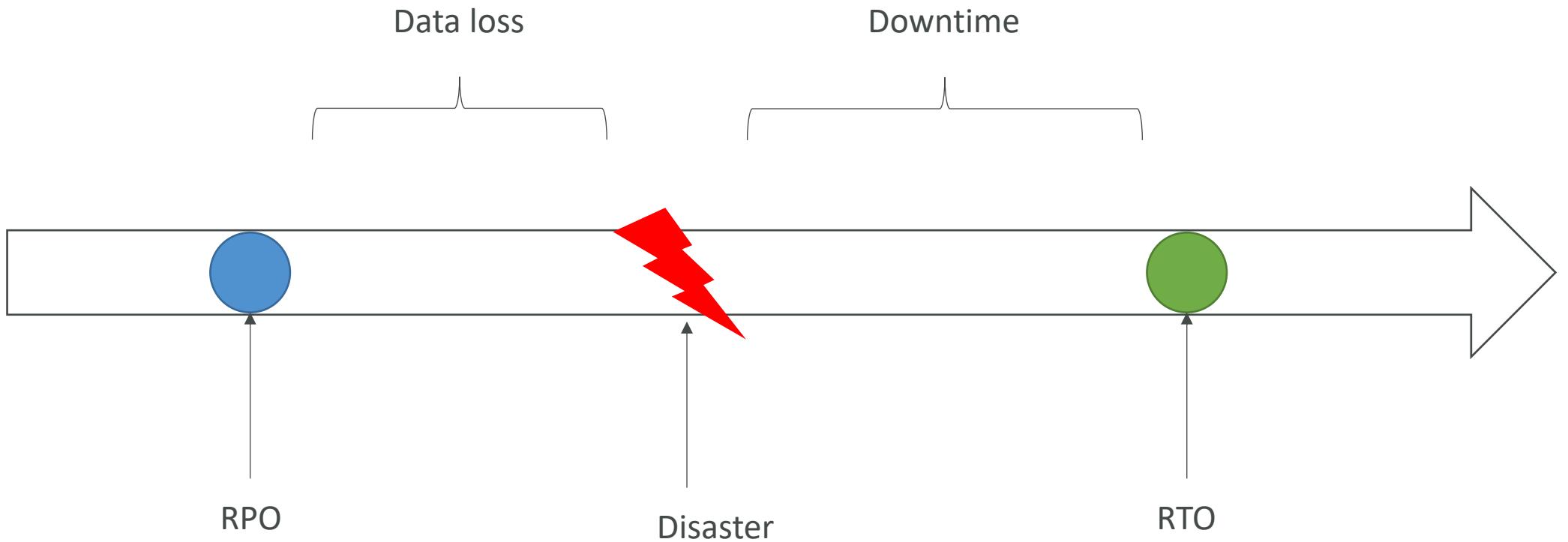
# Multi-Region Architectures – Complex Routing



# Disaster Recovery Overview

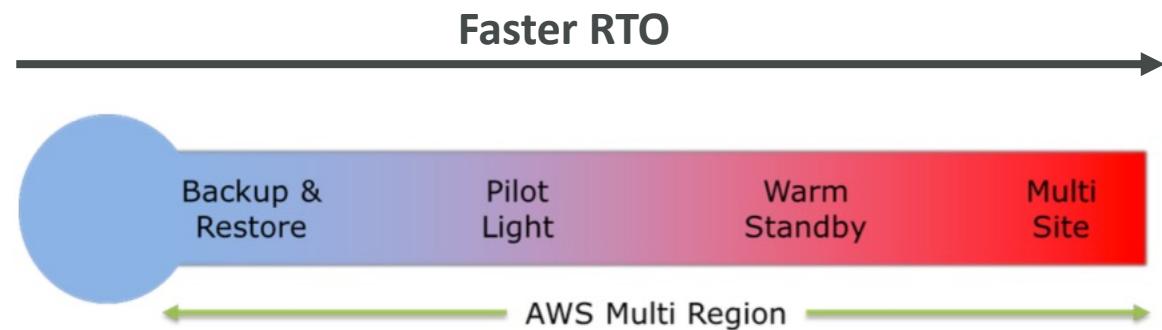
- Any event that has a negative impact on a company's business continuity or finances is a disaster
- Disaster recovery (DR) is about preparing for and recovering from a disaster
- What kind of disaster recovery?
  - On-premise => On-premise: traditional DR, and very expensive
  - On-premise => AWS Cloud: hybrid recovery
  - AWS Cloud Region A => AWS Cloud Region B
- Need to define two terms:
  - RPO: Recovery Point Objective
  - RTO: Recovery Time Objective

# RPO and RTO

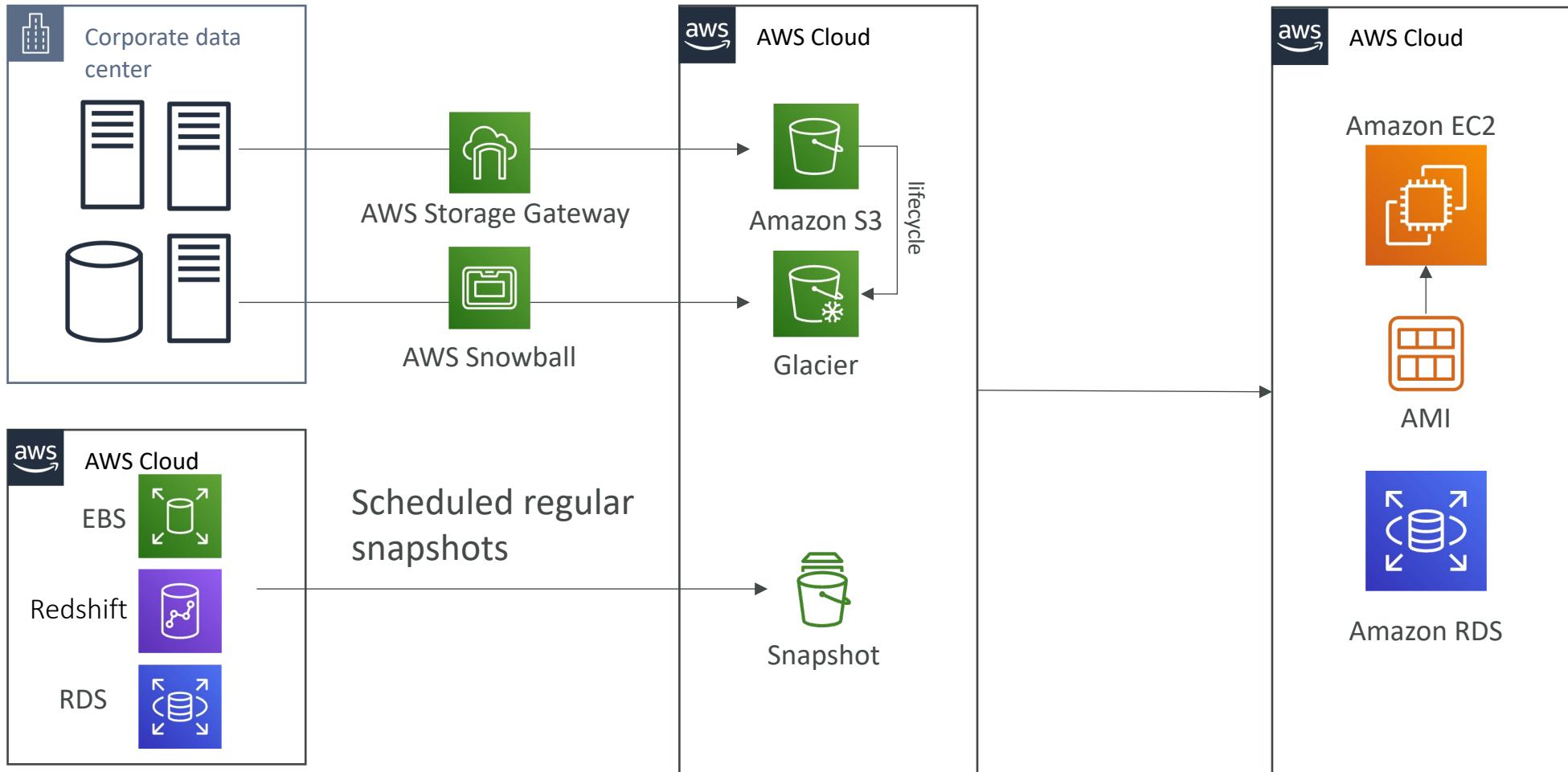


# Disaster Recovery Strategies

- Backup and Restore
- Pilot Light
- Warm Standby
- Hot Site / Multi Site Approach

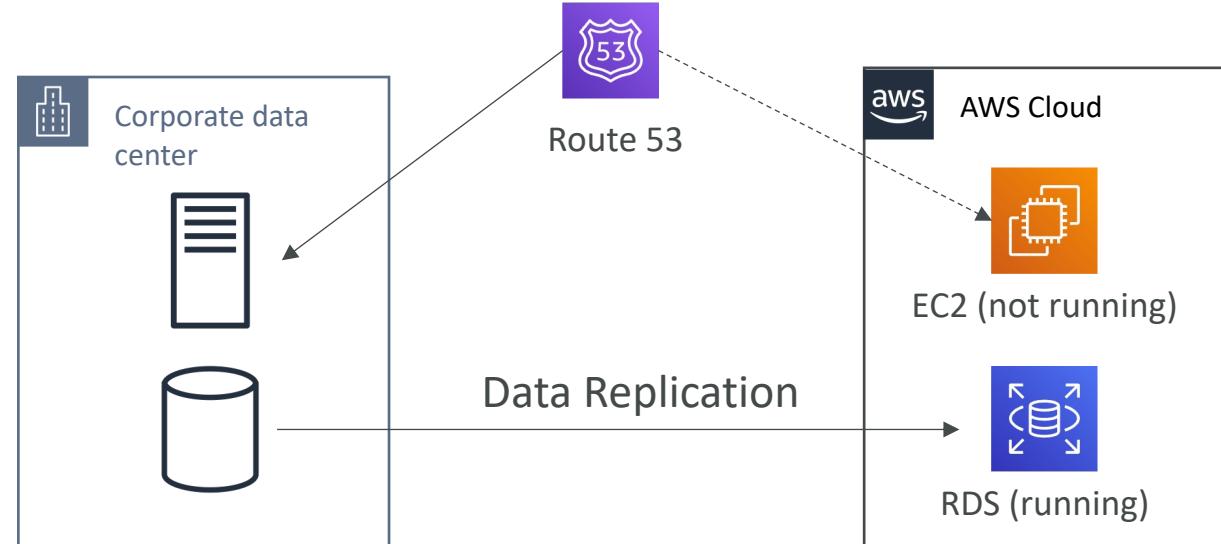


# Backup and Restore (High RPO)



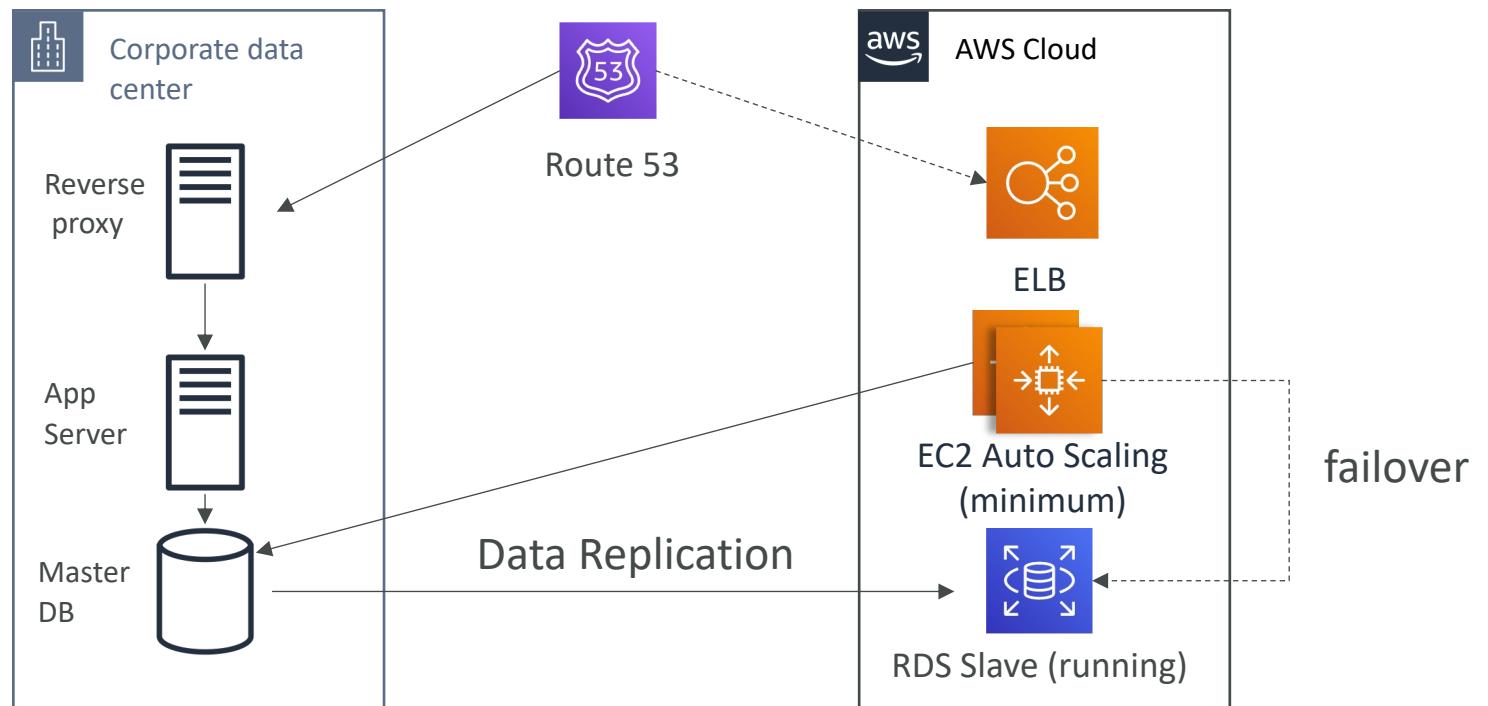
# Disaster Recovery – Pilot Light

- A small version of the app is always running in the cloud
- Useful for the critical core (pilot light)
- Very similar to Backup and Restore
- Faster than Backup and Restore as critical systems are already up



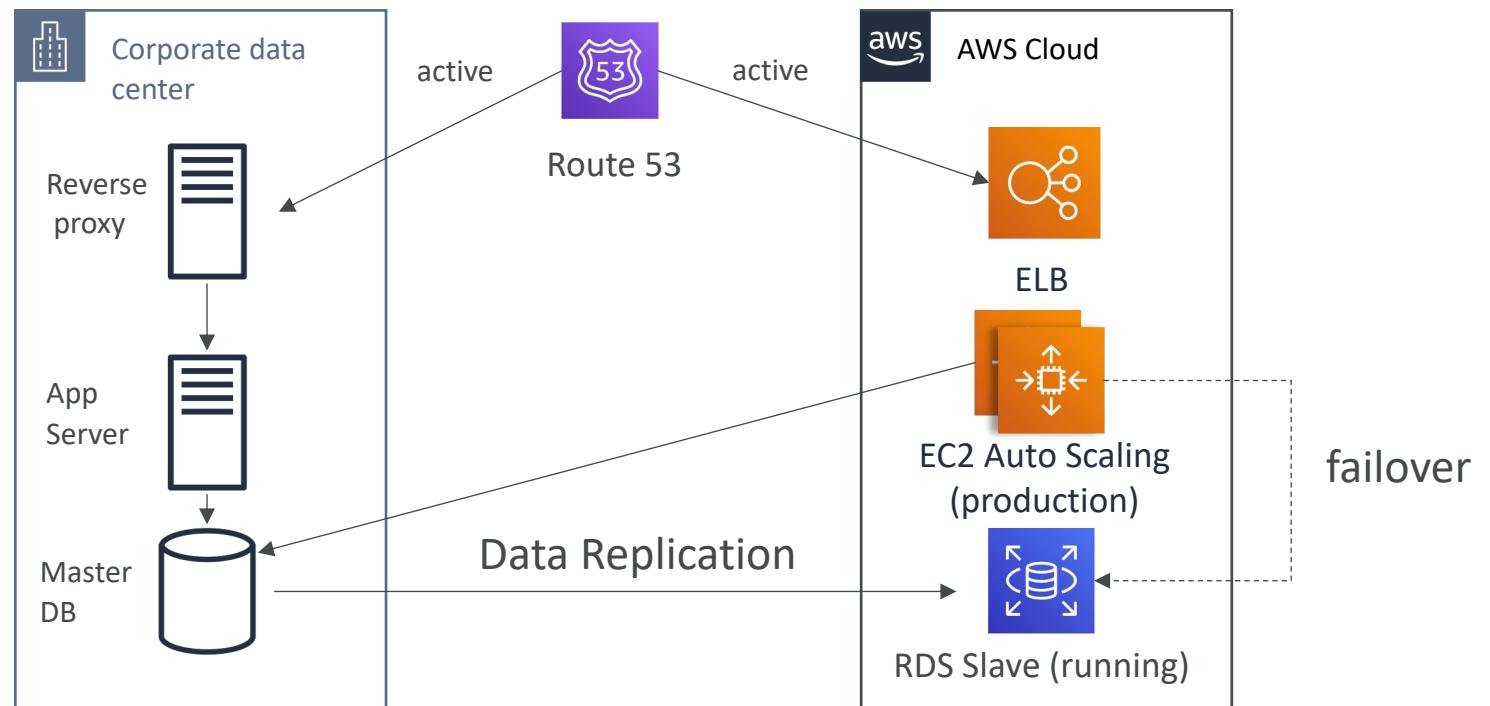
# Warm Standby

- Full system is up and running, but at minimum size
- Upon disaster, we can scale to production load

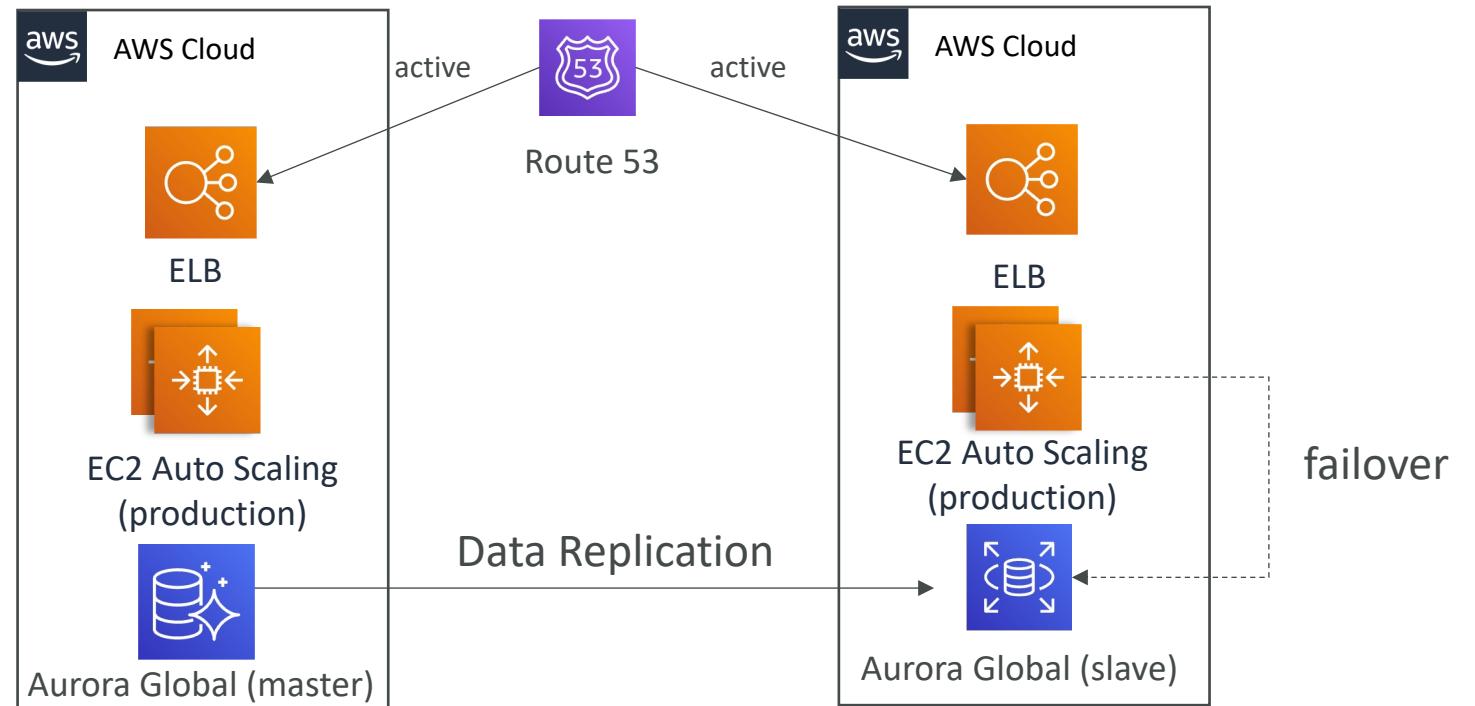


# Multi Site / Hot Site Approach

- Very low RTO (minutes or seconds) – very expensive
- Full Production Scale is running AWS and On Premise



# All AWS Multi Region



# Disaster Recovery Tips

- **Backup**
  - EBS Snapshots, RDS automated backups / Snapshots, etc...
  - Regular pushes to S3 / S3 IA / Glacier, Lifecycle Policy, Cross Region Replication
  - From On-Premise: Snowball or Storage Gateway
- **High Availability**
  - Use Route53 to migrate DNS over from Region to Region
  - RDS Multi-AZ, ElastiCache Multi-AZ, EFS, S3
  - Site to Site VPN as a recovery from Direct Connect
- **Replication**
  - RDS Replication (Cross Region), AWS Aurora + Global Databases
  - Database replication from on-premise to RDS
  - Storage Gateway
- **Automation**
  - CloudFormation / Elastic Beanstalk to re-create a whole new environment
  - Recover / Reboot EC2 instances with CloudWatch if alarms fail
  - AWS Lambda functions for customized automations
- **Chaos**
  - Netflix has a “simian-army” randomly terminating EC2

# Domain 4 – Monitoring and Logging

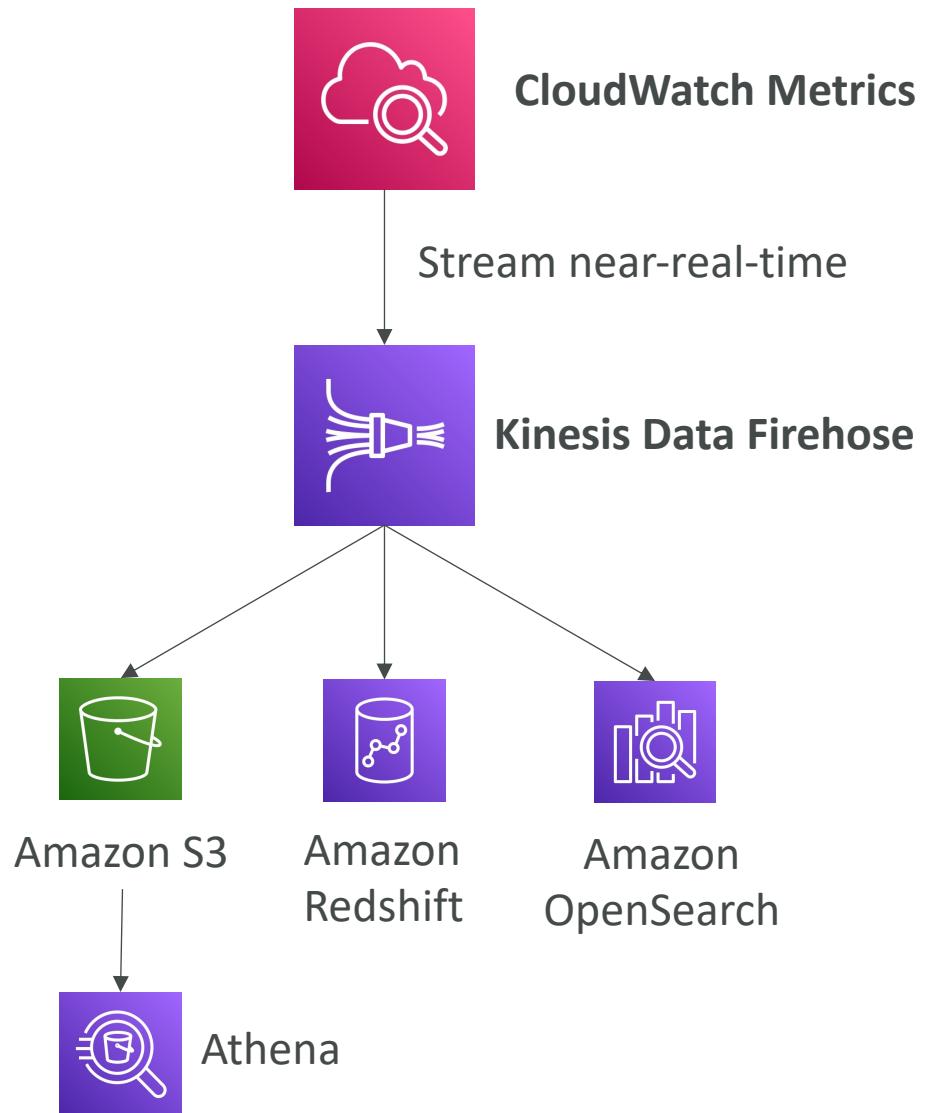
# Amazon CloudWatch Metrics



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

# CloudWatch Metric Streams

- Continually stream CloudWatch metrics to a destination of your choice, with **near-real-time delivery** and low latency.
  - Amazon Kinesis Data Firehose (and then its destinations)
  - 3<sup>rd</sup> party service provider: Datadog, Dynatrace, New Relic, Splunk, Sumo Logic...
- Option to **filter metrics** to only stream a subset of them



# CloudWatch Custom Metrics

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

# CloudWatch Anomaly Detection

- Continuously analyze metrics to determine normal baselines and surface anomalies using ML algorithms
- It creates a model of the metric's expected values (based on metric's past data)
- Shows you which values in the graph are out of the normal range
- Allows you to create Alarms based on metric's expected value (instead of Static Threshold)
- Ability to exclude specified time periods or events from being trained

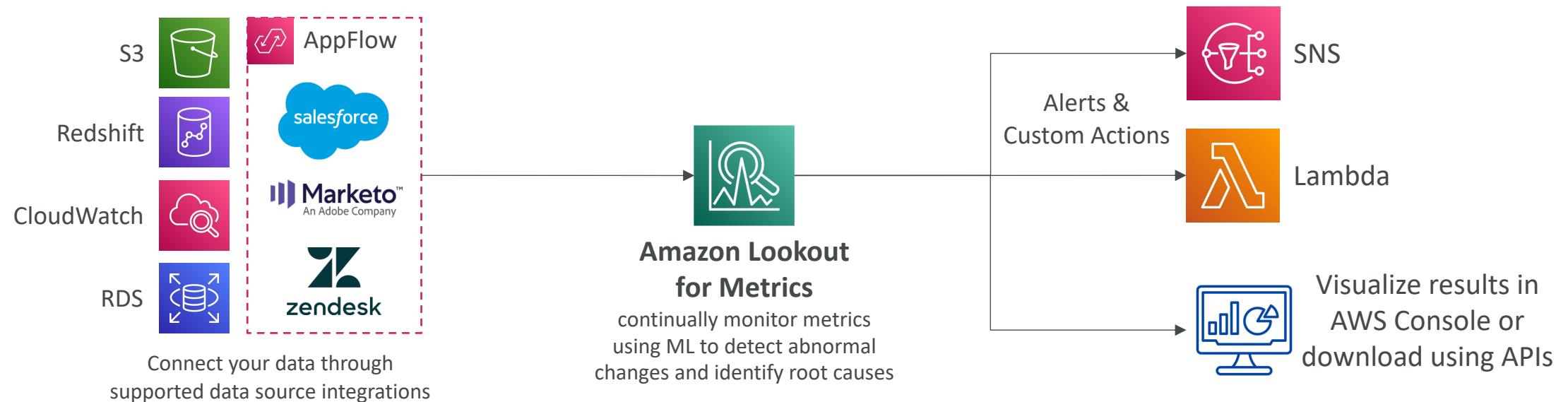


[https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/CloudWatch\\_Anomaly\\_Detection.html](https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/CloudWatch_Anomaly_Detection.html)

# Amazon Lookout for Metrics



- Automatically detect anomalies within metrics and identify their root causes using Machine Learning
- It detects and diagnoses errors within your data with no manual intervention
- Integrates with different AWS Services and 3rd party SaaS apps through AppFlow
- Send alerts to SNS, Lambda, Slack, Webhooks...





# CloudWatch Logs

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

# CloudWatch Logs - Sources

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

# CloudWatch Logs Insights

The screenshot shows the CloudWatch Logs Insights interface. At the top, there's a navigation bar with 'CloudWatch > Logs Insights'. Below it is a search bar labeled 'Select log group(s)' containing 'application.log' with a clear button. To the right is a time range selector from '2021-11-09 (06:40:02)' to '2021-11-09 (06:55:17)'. A large orange box highlights the time range with the text 'Change the time range here.' An arrow points from this box to the time range selector. Another orange box highlights the log group selection with the text 'Discovered Fields in your log groups.' An arrow points from this box to the 'application.log' field.

In the center, there's a code editor area with a query:

```
1 fields @timestamp, @message
2 | sort @timestamp desc
3 | limit 20
```

Below the code editor are three buttons: 'Run query', 'Save', and 'History'. A note below says 'Queries are allowed to run for up to 15 minutes.' An orange box highlights the 'Run query' button with the text 'Write your query here.' An arrow points from this box to the button.

To the right of the code editor, there are two more orange boxes: one for 'Export results' and one for 'Add to dashboard'. Arrows point from these boxes to their respective buttons in the interface.

On the far right, there's a sidebar with three items: 'Fields', 'Queries', and 'Help'. 'Fields' is currently selected, indicated by a grey background.

The main content area shows a histogram of log records over time, with the x-axis from 06:40 to 06:55 and the y-axis from 0 to 400. The histogram shows several peaks, with the highest peak around 06:45. Below the histogram, a table lists two log entries:

#	@timestamp	@message
► 1	2021-11-09T06:54:17.62...	{"Severity": "INFO", "message": "This is where the message detail would go", "IP Address": "10.30.86.98", "Timestamp": "2021-11-09T11:54:17.620Z"}
► 2	2021-11-09T06:54:13.38...	{"Severity": "INFO", "message": "This is where the message detail would go", "IP Address": "192.168.0.43", "Timestamp": "2021-11-09T11:54:13.380Z"}

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

# CloudWatch Logs Insights

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

Sample queries [Learn more ↗](#)

- ▶ Lambda
- ▶ VPC Flow Logs
- ▶ CloudTrail
- ▼ Common queries

▼ 25 most recently added log events

```
fields @timestamp, @message
| sort @timestamp desc
| limit 25
```

[Apply](#)

▼ Number of exceptions logged every 5 minutes

```
filter @message like /Exception/
| stats count(*) as exceptionCount by
bin(5m)
| sort exceptionCount desc
```

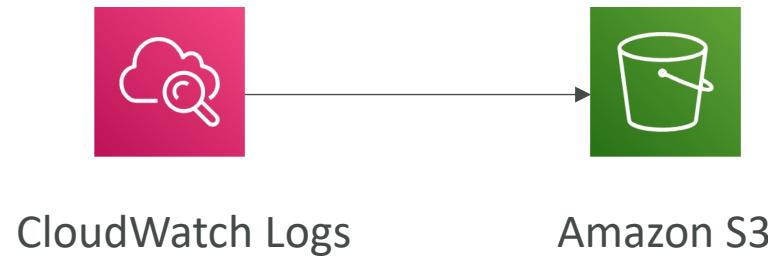
[Apply](#)

▼ List of log events that are not exceptions

```
fields @message
| filter @message not like /Exception/
```

[Apply](#)

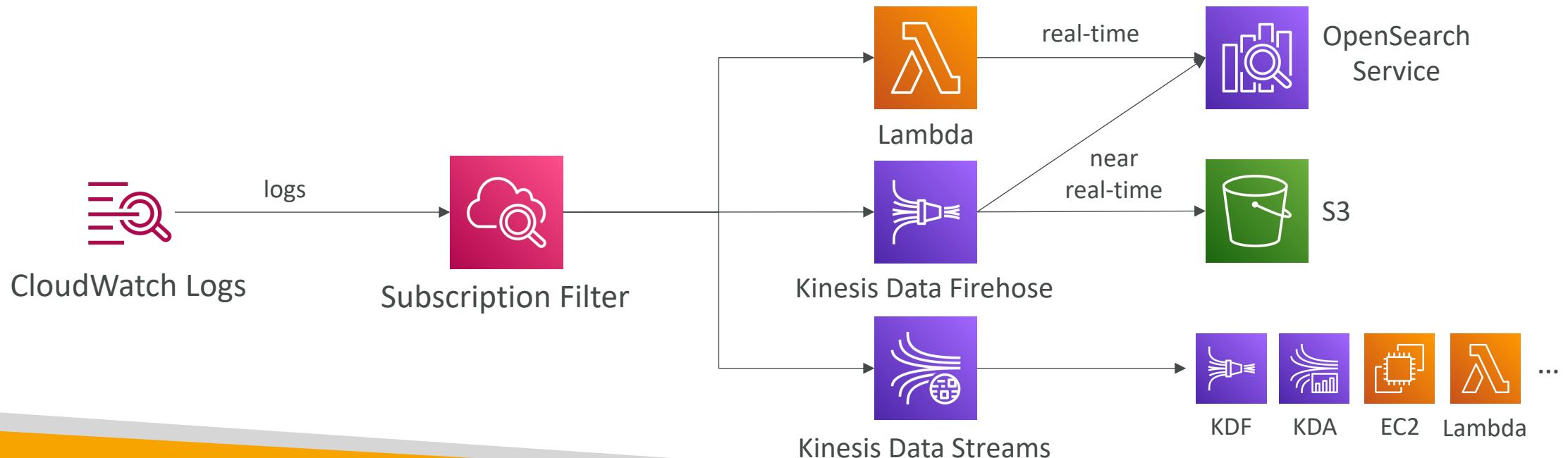
# CloudWatch Logs – S3 Export



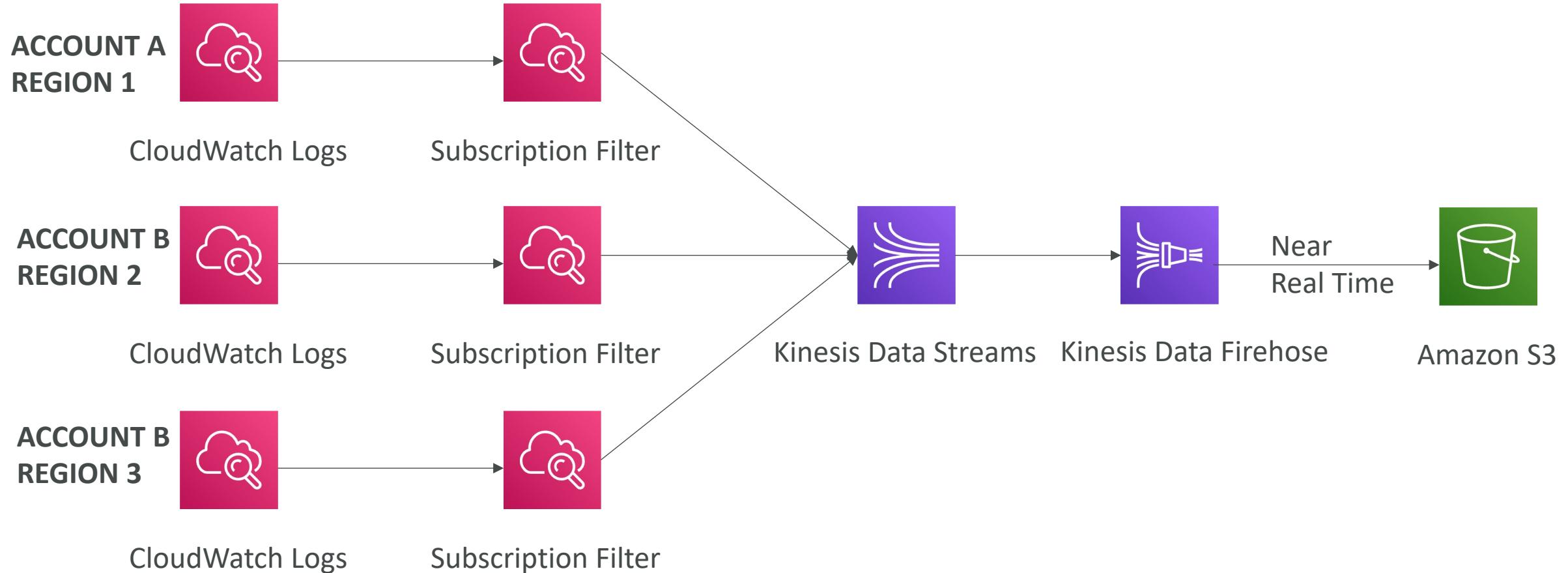
- Log data can take up to 12 hours to become available for export
- The API call is `CreateExportTask`
- Not near-real time or real-time... use Logs Subscriptions instead

# CloudWatch Logs Subscriptions

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

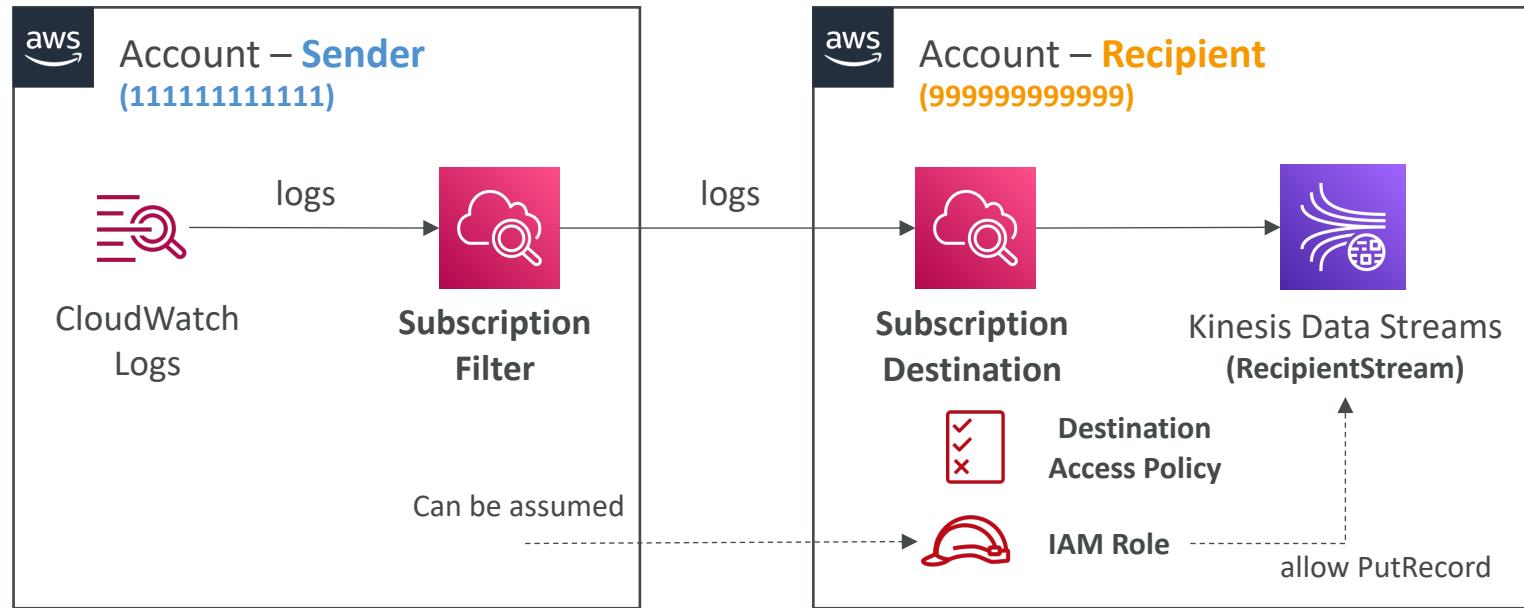


# CloudWatch Logs Aggregation Multi-Account & Multi Region



# CloudWatch Logs Subscriptions

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



**IAM Role (Cross-Account)**

```

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

```

**Destination Access Policy**

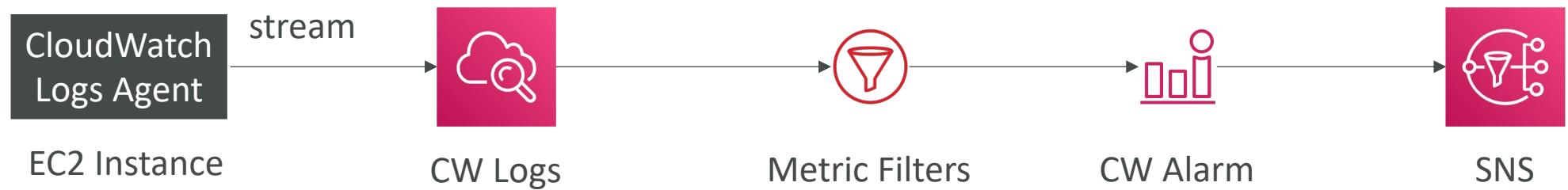
```

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

```

# CloudWatch Logs Metric Filter

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



# All kind of Logs

- Application Logs
  - Logs that are produced by your application code
  - Contains custom log messages, stack traces, and so on
  - Written to a local file on the filesystem
  - Usually streamed to CloudWatch Logs using a CloudWatch Agent on EC2
  - If using Lambda, direct integration with CloudWatch Logs
  - If using ECS or Fargate, direct integration with CloudWatch Logs
  - If using Elastic Beanstalk, direct integration with CloudWatch Logs
- Operating System Logs (Event Logs, System Logs)
  - Logs that are generated by your operating system (EC2 or on-premise instance)
  - Informing you of system behavior (ex: /var/log/messages or /var/log/auth.log)
  - Usually streamed to CloudWatch Logs using a CloudWatch Agent

# All kind of Logs

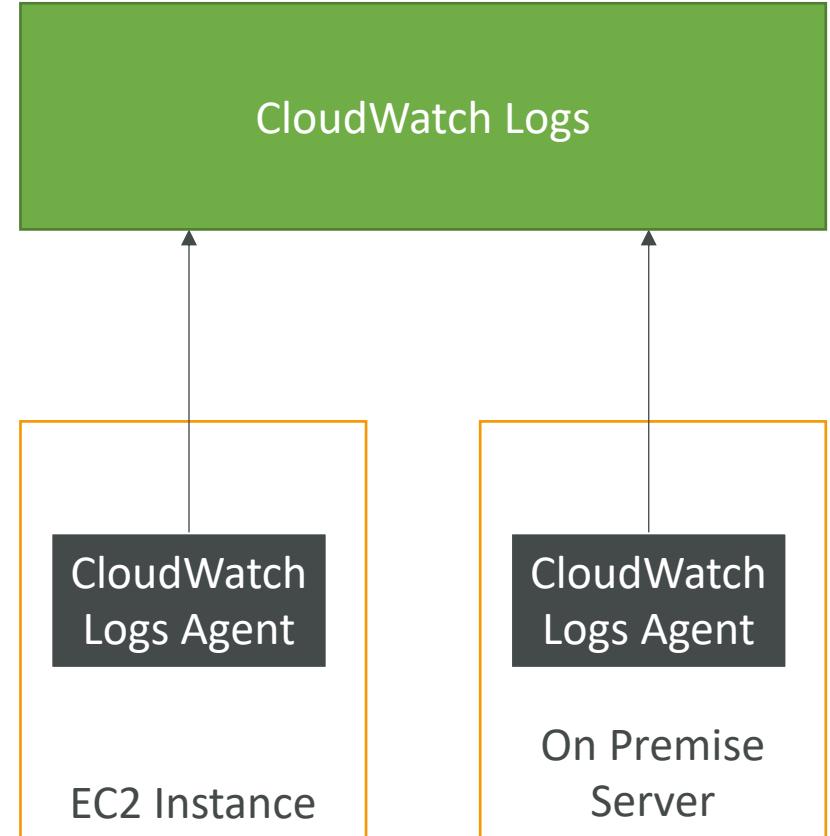
- Access Logs
  - list of all the requests for individual files that people have requested from a website
  - Example for httpd: /var/log/apache/access.log
  - Usually for load balancers, proxies, web servers, etc...
  - AWS provides some access logs

# AWS Managed Logs

- **Load Balancer Access Logs (ALB, NLB, CLB)** => to S3
  - Access logs for your Load Balancers
- **CloudTrail Logs** => to S3 and CloudWatch Logs
  - Logs for API calls made within your account
- **VPC Flow Logs** => to S3 and CloudWatch Logs
  - Information about IP traffic going to and from network interfaces in your VPC
- **Route 53 Access Logs** => to CloudWatch Logs
  - Log information about the queries that Route 53 receives
- **S3 Access Logs** => to S3
  - Server access logging provides detailed records for the requests that are made to a bucket
- **CloudFront Access Logs** => to S3
  - Detailed information about every user request that CloudFront receives

# CloudWatch Logs for EC2

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



# CloudWatch Logs Agent & Unified Agent

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

# CloudWatch Unified Agent – Metrics

- Collected directly on your Linux server / EC2 instance
- CPU (active, guest, idle, system, user, steal)
- Disk metrics (free, used, total), Disk IO (writes, reads, bytes, iops)
- RAM (free, inactive, used, total, cached)
- Netstat (number of TCP and UDP connections, net packets, bytes)
- Processes (total, dead, bloqued, idle, running, sleep)
- Swap Space (free, used, used %)
- Reminder: out-of-the box metrics for EC2 – disk, CPU, network (high level)

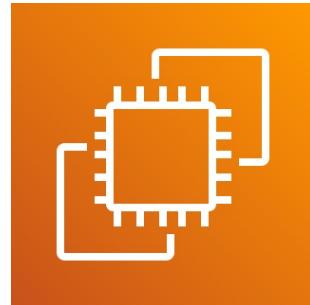
# CloudWatch Alarms



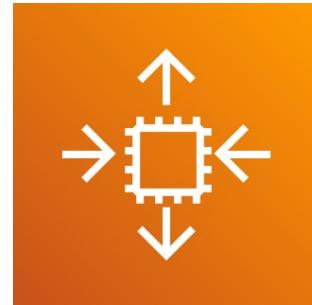
- Alarms are used to trigger notifications for any metric
- Various options (sampling, %, max, min, etc...)
- Alarm States:
  - OK
  - INSUFFICIENT\_DATA
  - ALARM
- Period:
  - Length of time in seconds to evaluate the metric
  - High resolution custom metrics: 10 sec, 30 sec or multiples of 60 sec

# CloudWatch Alarm Targets

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



Amazon EC2



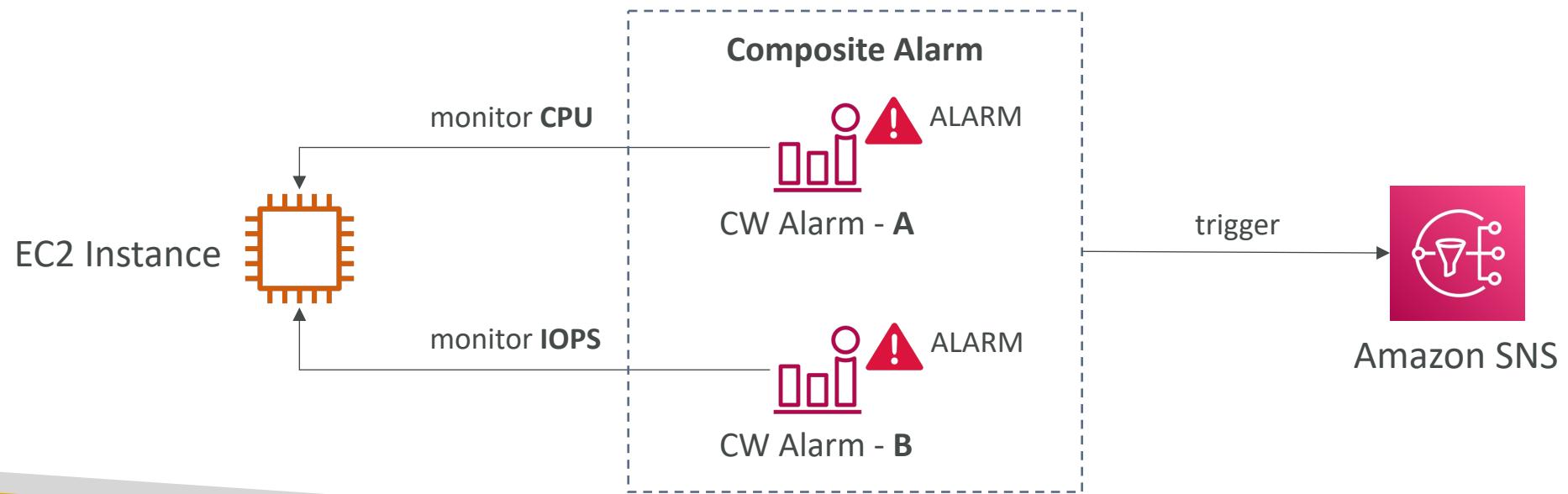
EC2 Auto Scaling



Amazon SNS

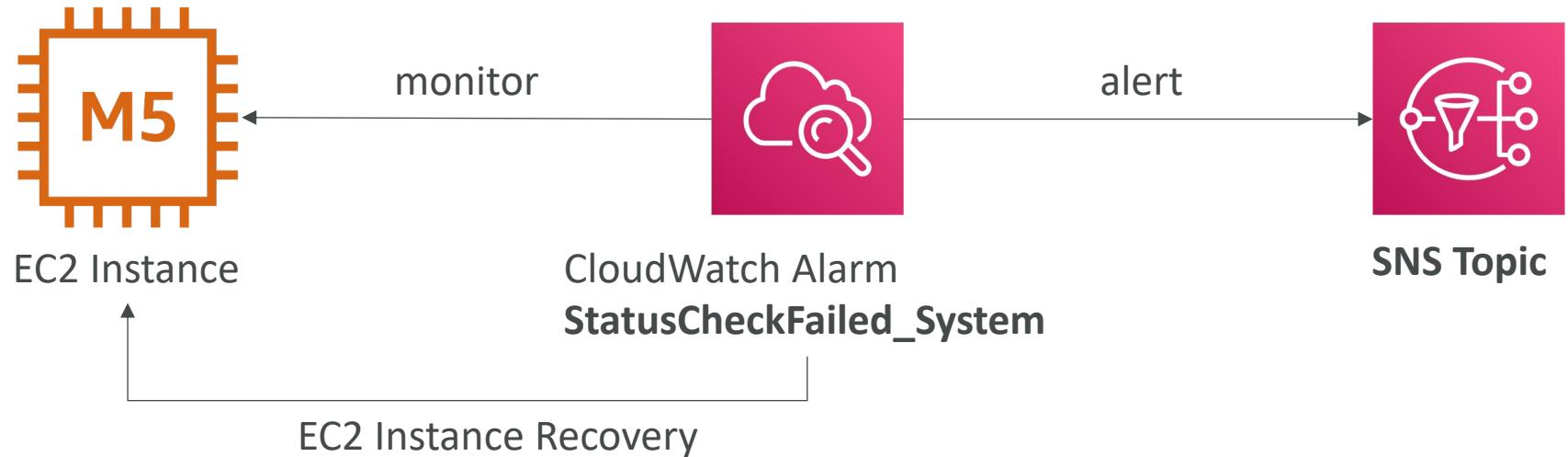
# CloudWatch Alarms – Composite Alarms

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



# EC2 Instance Recovery

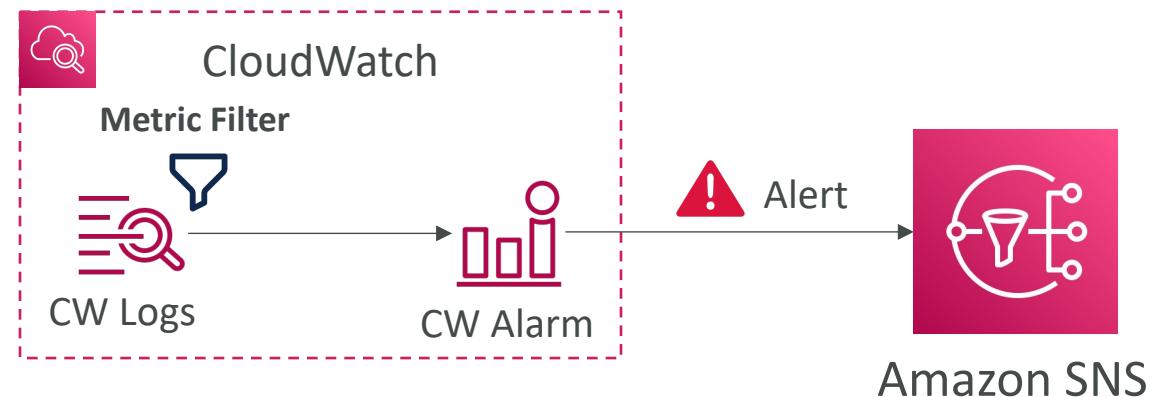
- Status Check:
  - Instance status = check the EC2 VM
  - System status = check the underlying hardware



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

# CloudWatch Alarm: good to know

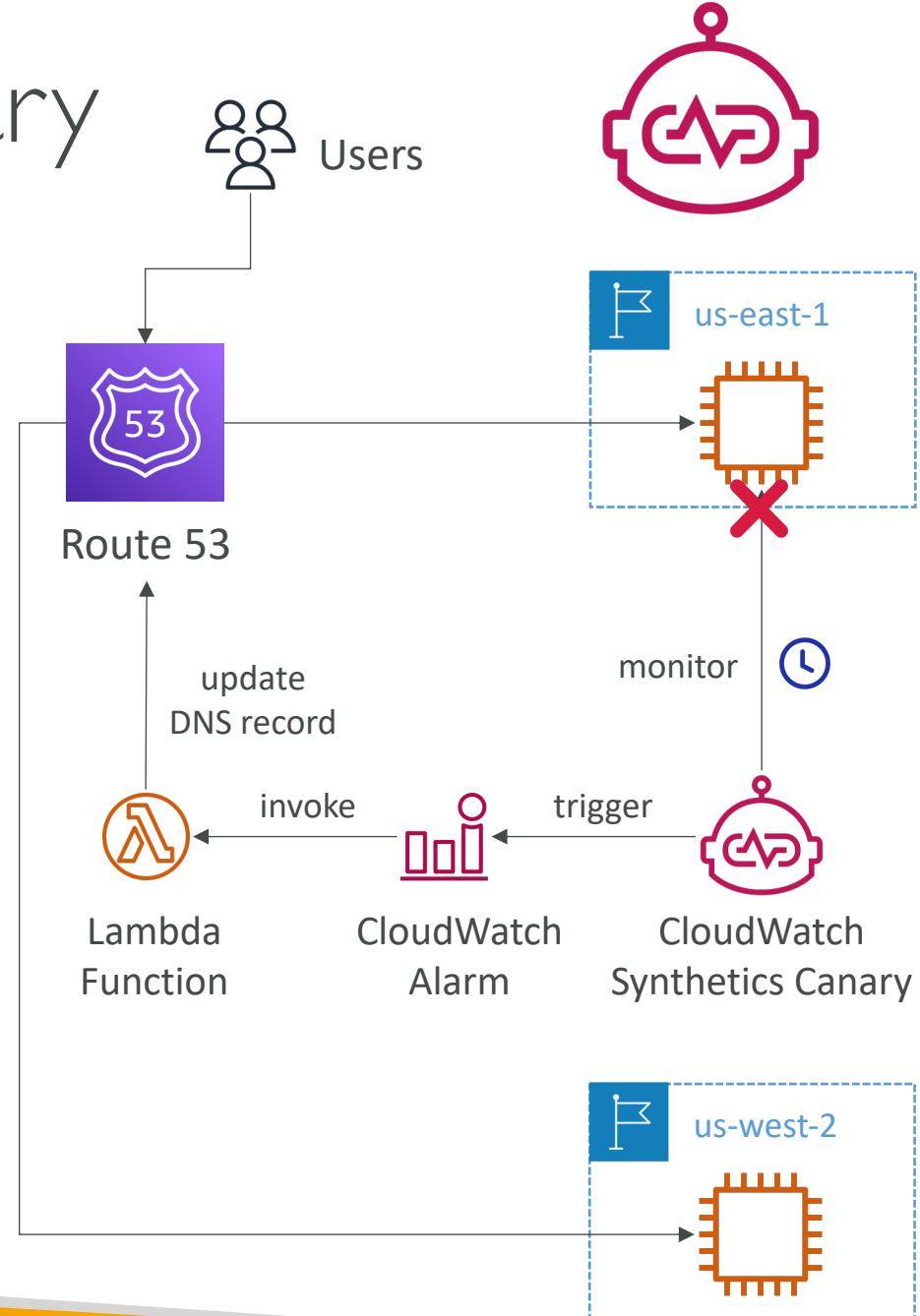
- Alarms can be created based on CloudWatch Logs Metrics Filters



- To test alarms and notifications, set the alarm state to Alarm using CLI  
`aws cloudwatch set-alarm-state --alarm-name "myalarm" --state-value ALARM --state-reason "testing purposes"`

# CloudWatch Synthetics Canary

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



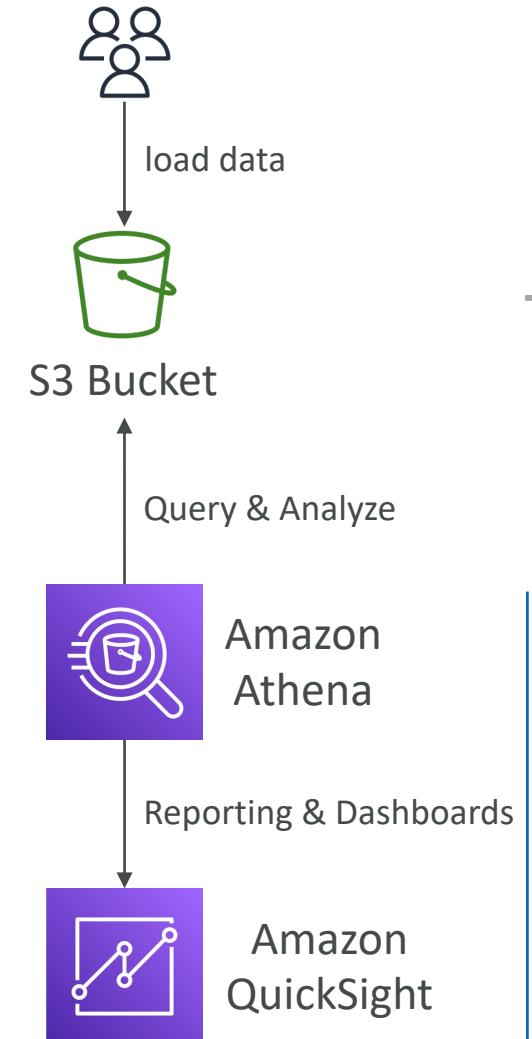
# CloudWatch Synthetics Canary Blueprints

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

# Amazon Athena



- Serverless query service to analyze data stored in Amazon S3
- Uses standard SQL language to query the files (built on Presto)
- Supports CSV, JSON, ORC, Avro, and Parquet
- Pricing: \$5.00 per TB of data scanned
- Commonly used with Amazon Quicksight for reporting/dashboards
- **Use cases:** Business intelligence / analytics / reporting, analyze & query VPC Flow Logs, ELB Logs, CloudTrail trails, etc...
- **Exam Tip:** analyze data in S3 using serverless SQL, use Athena

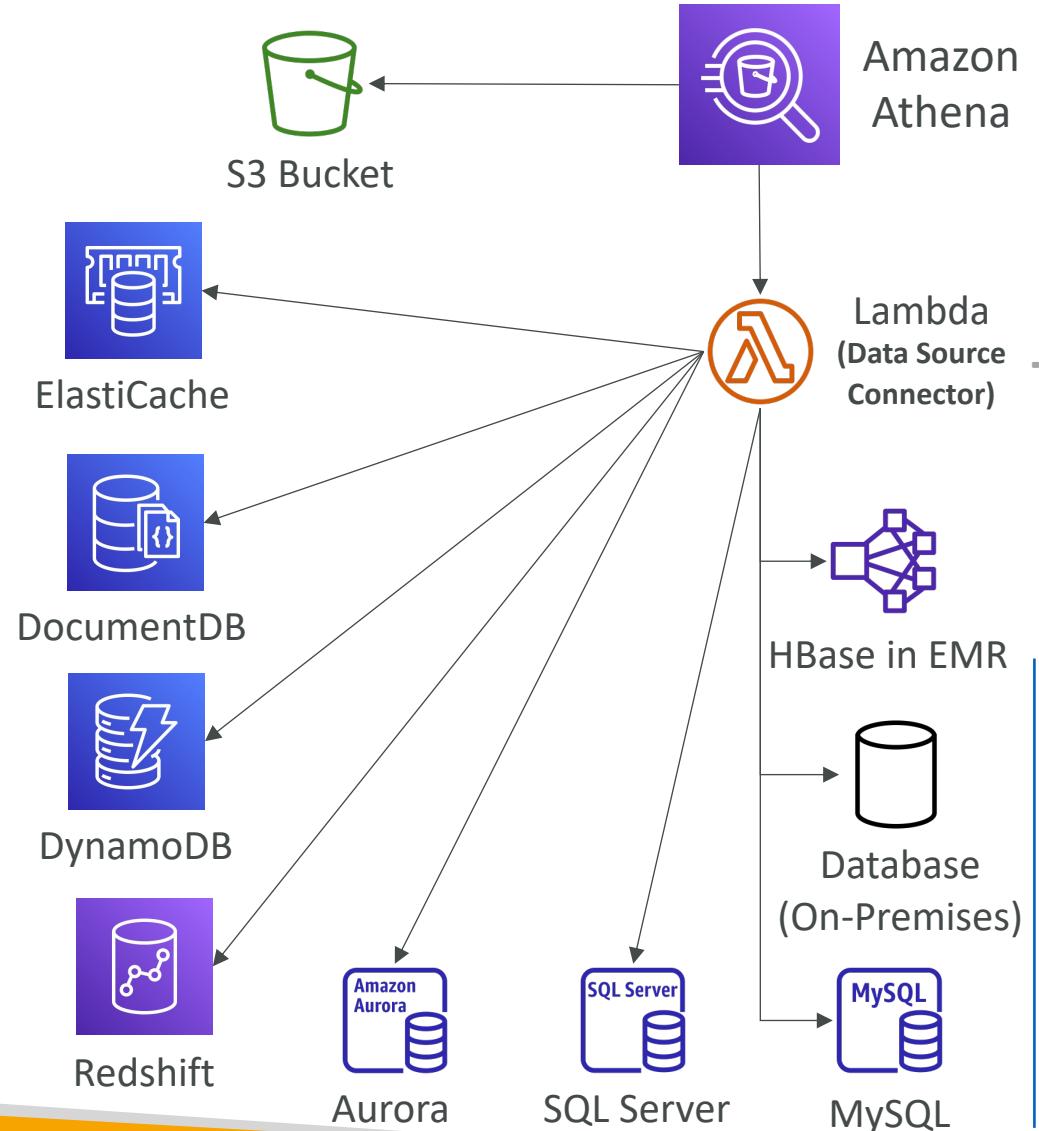


# Amazon Athena – Performance Improvement

- Use **columnar data** for cost-savings (less scan)
  - Apache Parquet or ORC is recommended
  - Huge performance improvement
  - Use Glue to convert your data to Parquet or ORC
- **Compress data** for smaller retrievals (bzip2, gzip, lz4, snappy, zlip, zstd...)
- **Partition** datasets in S3 for easy querying on virtual columns
  - s3://yourBucket/pathToTable  
  / <PARTITION\_COLUMN\_NAME>=<VALUE>  
  / <PARTITION\_COLUMN\_NAME>=<VALUE>  
  / <PARTITION\_COLUMN\_NAME>=<VALUE>  
  /etc...
  - Example: s3://athena-examples/flight/parquet/year=1991/month=1/day=1/
- Use **larger files** (> 128 MB) to minimize overhead

# Amazon Athena – Federated Query

- Allows you to run SQL queries across data stored in relational, non-relational, object, and custom data sources (AWS or on-premises)
- Uses Data Source Connectors that run on AWS Lambda to run Federated Queries (e.g., CloudWatch Logs, DynamoDB, RDS...)
- Store the results back in Amazon S3

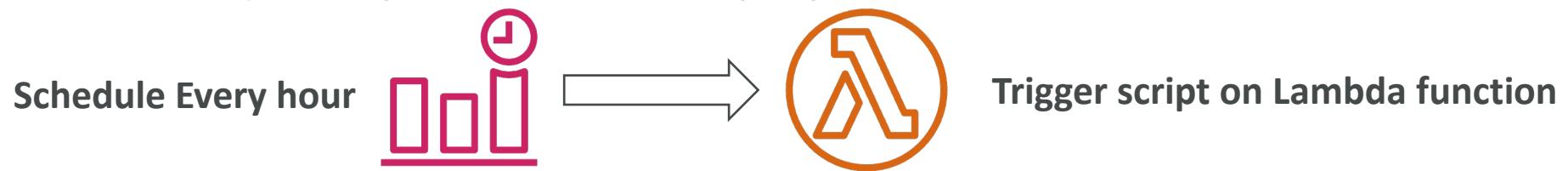


# Domain 5 – Incident and Event Response

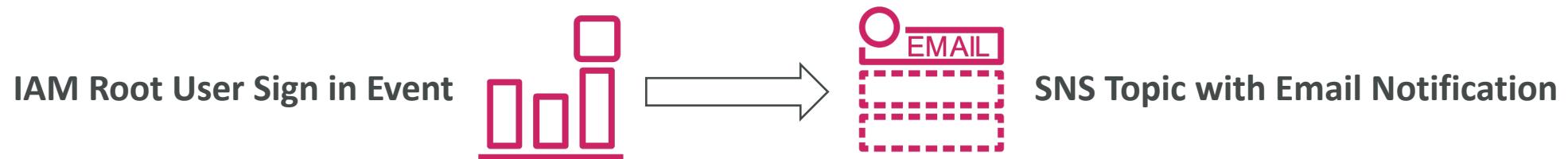
# Amazon EventBridge (formerly CloudWatch Events)



- Schedule: Cron jobs (scheduled scripts)

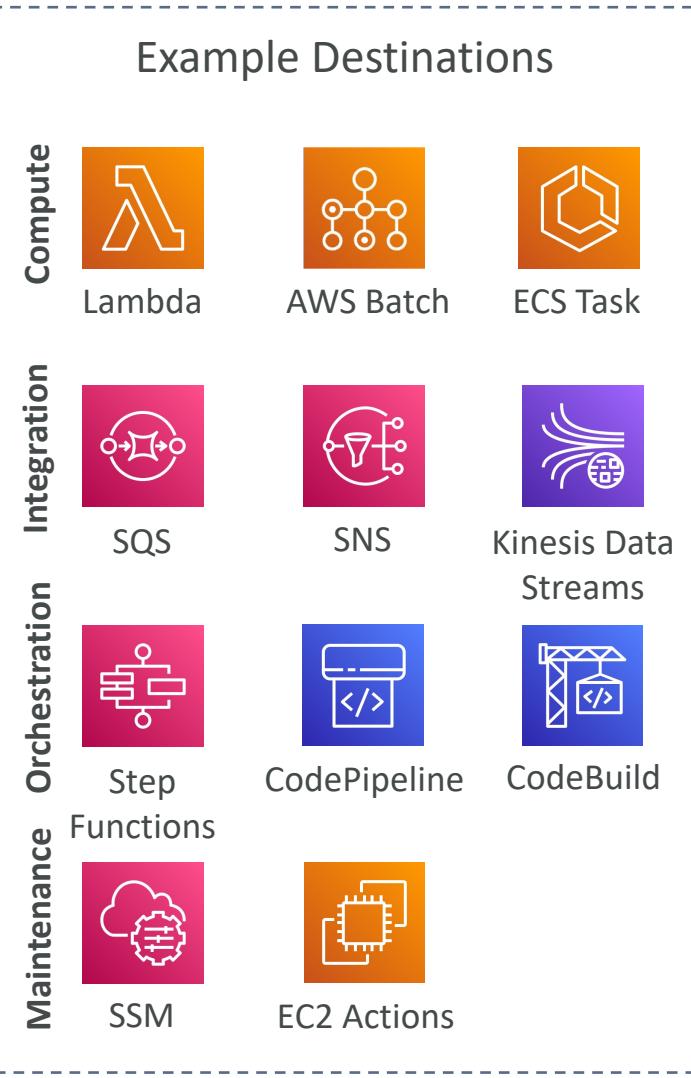
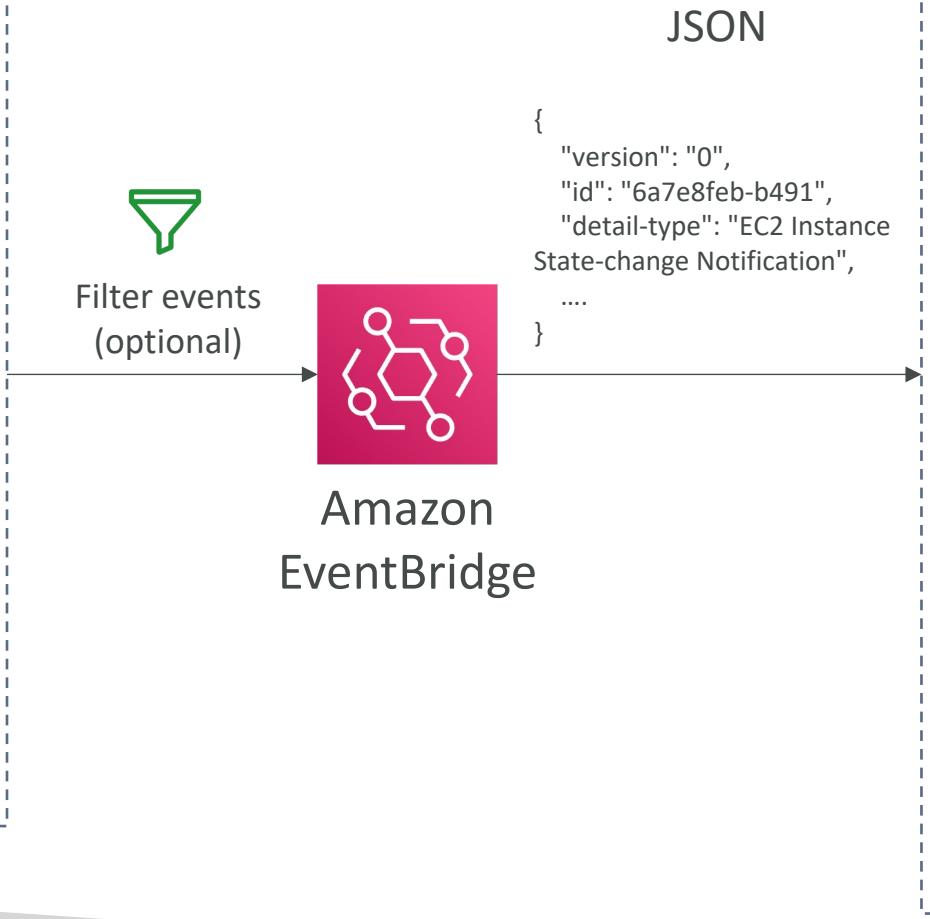
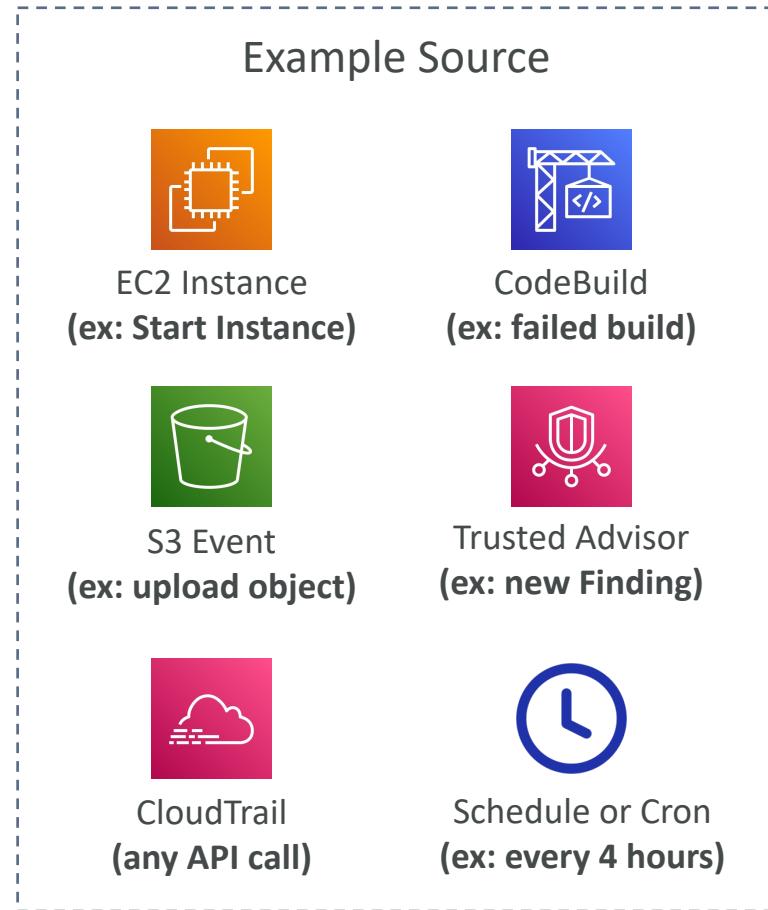


- Event Pattern: Event rules to react to a service doing something

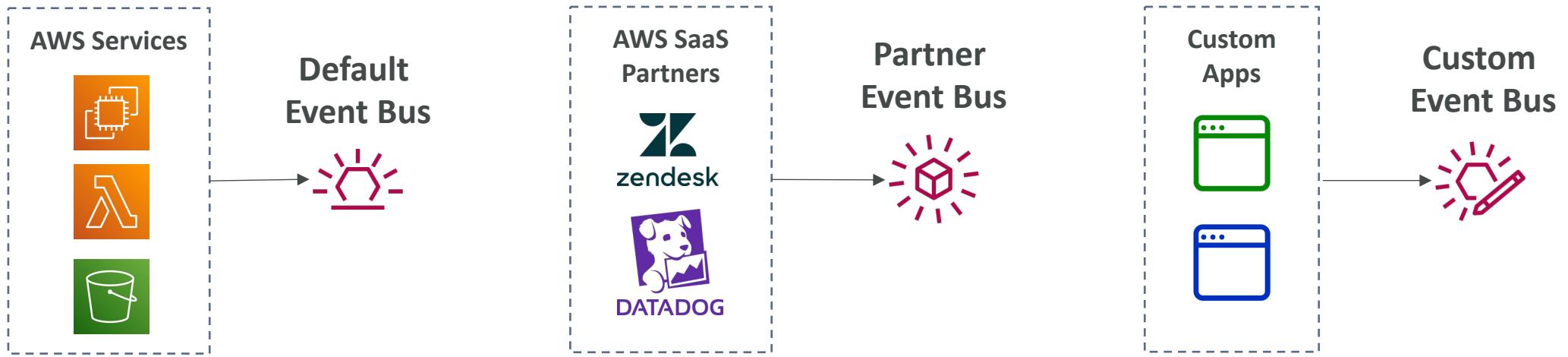


- Trigger Lambda functions, send SQS/SNS messages...

# Amazon EventBridge Rules



# Amazon EventBridge



- Event buses can be accessed by other AWS accounts using Resource-based Policies
- You can **archive events** (all/filter) sent to an event bus (indefinitely or set period)
- Ability to **replay archived events**

# Amazon EventBridge – Schema Registry

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

The screenshot shows the AWS Schema Registry interface. At the top, there's a header with the URL "aws.codepipeline@CodePipelineActionExecutionStateChange". Below it is a section titled "Schema details" containing the following information:

Schema name	Last modified	Schema ARN
aws.codepipeline@CodePipelineActionExecutionStateChange	Dec 1, 2019, 12:11 AM GMT	-
Description	Schema for event type CodePipelineActionExecutionStateChange, published by AWS service aws.codepipeline	Schema registry aws.events Number of versions 1 Schema type OpenAPI 3.0

Below this is a section titled "Version 1 Created on Dec 1, 2019, 12:11 AM GMT" with a "Download code bindings" button. A code editor window shows the following JSON schema:

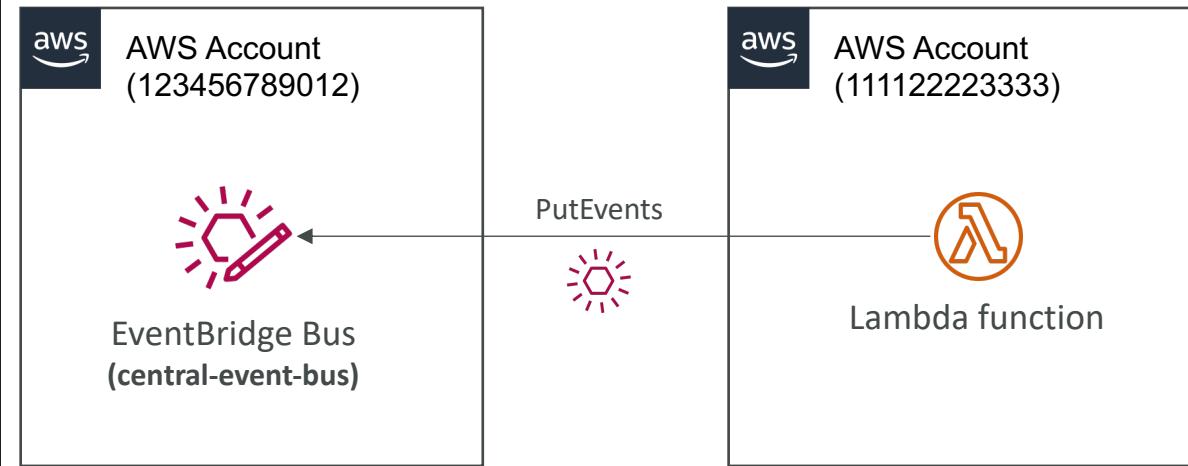
```
1 {
2   "openapi": "3.0.0",
3   "info": {
4     "version": "1.0.0",
5     "title": "CodePipelineActionExecutionStateChange"
6   },
7   "paths": {},
8   "components": {
9     "schemas": {
10       "AWSEvent": {
```

# Amazon EventBridge – Resource-based Policy

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

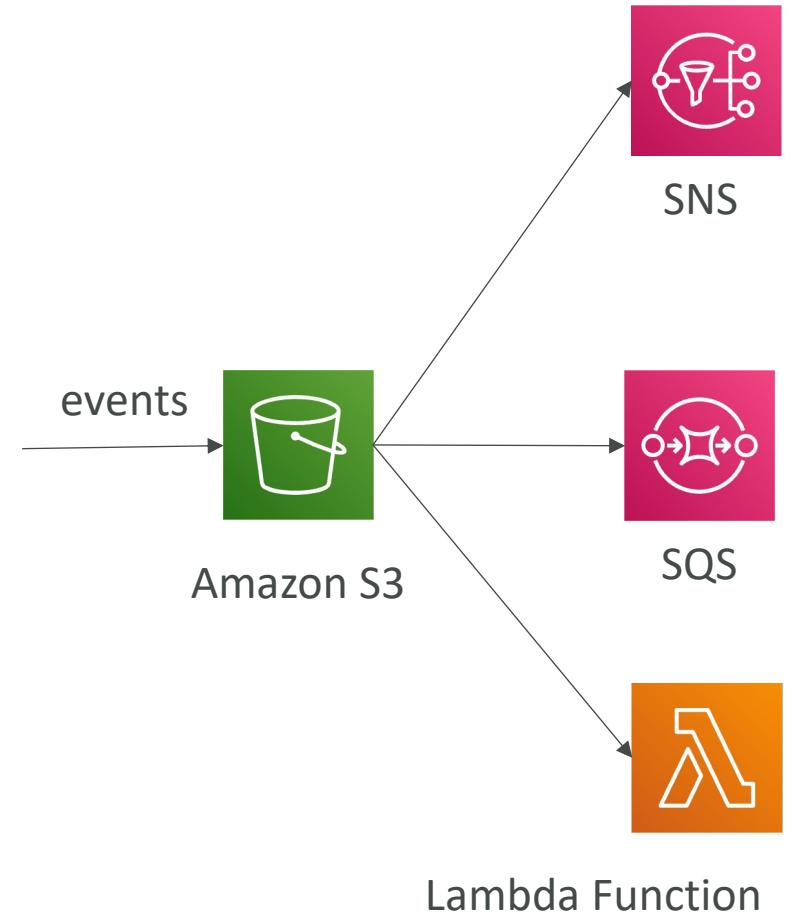
```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "events:PutEvents",  
            "Principal": { "AWS": "111122223333" },  
            "Resource": "arn:aws:events:us-east-1:123456789012:  
event-bus/central-event-bus"  
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

Allow **events** from another AWS account

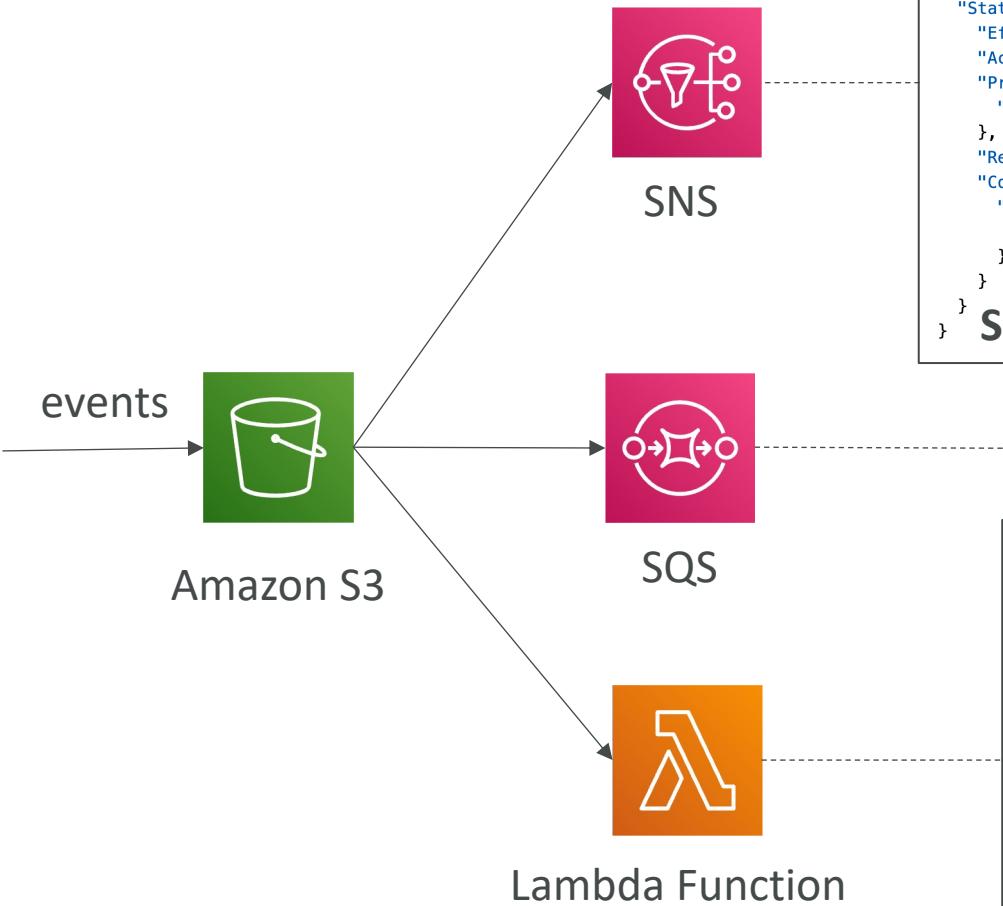


# S3 Event Notifications

- S3:ObjectCreated, S3:ObjectRemoved, S3:ObjectRestore, S3:Replication...
- Object name filtering possible (\*.jpg)
- Use case: generate thumbnails of images uploaded to S3
- Can create as many “S3 events” as desired
- S3 event notifications typically deliver events in seconds but can sometimes take a minute or longer



# S3 Event Notifications – IAM Permissions



```
{  
    "Version": "2012-10-17",  
    "Statement": {  
        "Effect": "Allow",  
        "Action": "SNS:Publish",  
        "Principal": {  
            "Service": "s3.amazonaws.com"  
        },  
        "Resource": "arn:aws:sns:us-east-1:123456789012:MyTopic",  
        "Condition": {  
            "ArnLike": {  
                "aws:SourceArn": "arn:aws:s3:::MyBucket"  
            }  
        }  
    }  
}
```

**SNS Resource (Access) Policy**

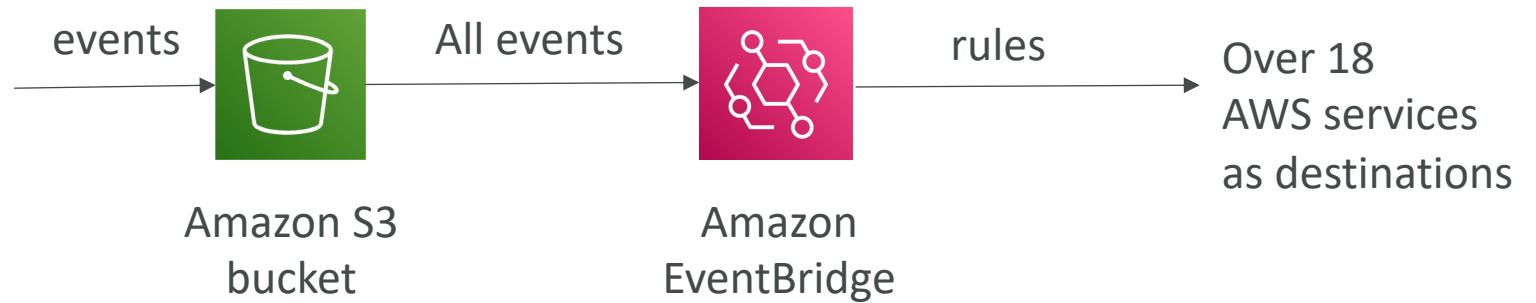
```
{  
    "Version": "2012-10-17",  
    "Statement": {  
        "Effect": "Allow",  
        "Action": "SQS:SendMessage",  
        "Principal": {  
            "Service": "s3.amazonaws.com"  
        },  
        "Resource": "arn:aws:sqs:us-east-1:123456789012:MyQueue",  
        "Condition": {  
            "ArnLike": {  
                "aws:SourceArn": "arn:aws:s3:::MyBucket"  
            }  
        }  
    }  
}
```

**SQS Resource (Access) Policy**

```
{  
    "Version": "2012-10-17",  
    "Statement": {  
        "Effect": "Allow",  
        "Action": "lambda:InvokeFunction",  
        "Principal": {  
            "Service": "s3.amazonaws.com"  
        },  
        "Resource": "arn:aws:lambda:us-east-1:123456789012:function:MyFunction",  
        "Condition": {  
            "ArnLike": {  
                "AWS:SourceArn": "arn:aws:s3:::MyBucket"  
            }  
        }  
    }  
}
```

**Lambda Resource Policy**

# S3 Event Notifications with Amazon EventBridge



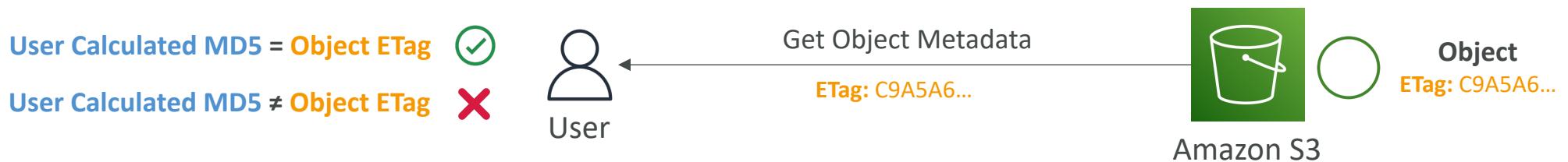
- Advanced filtering options with JSON rules (metadata, object size, name...)
- Multiple Destinations – ex Step Functions, Kinesis Streams / Firehose...
- EventBridge Capabilities – Archive, Replay Events, Reliable delivery

# Amazon S3 – Object Integrity

- S3 uses checksum to validate the integrity of uploaded objects
- Using MD5



- Using MD5 & Etag
  - ETag – represents a specific version of the object, ETag = MD5 (if SSE-S3)



- Other supported checksums: SHA-1, SHA-256, CRC32, CRC32C

# AWS Health Dashboard - Service History



- Shows all regions, all services health
- Shows historical information for each day
- Has an RSS feed you can subscribe to
- Previously called AWS Service Health Dashboard

Service history									
The following table is a running log of AWS service interruptions for the past 12 months. Choose a status icon to see status updates for that service. All dates and times are reported in UTC. To update your time zone, see <a href="#">Time zone settings</a> .									
Find an AWS service or Region							2023/01/10	CSV	
North America	South America	Europe	Africa	Asia Pacific	Middle East				
Alexa for Business (N. Virginia)						🟢	🟢	🟢	🟢
Amazon EventBridge Scheduler (N. Virginia)						🟢	🟢	🟡	🟡
Amazon EventBridge Scheduler (Ohio)						🟢	🟡	🟡	🟡
Amazon EventBridge Scheduler (Oregon)						🟡	🟡	🟡	🟡
Amazon API Gateway (Montreal)						🟡	🟡	🟡	🟡
Amazon API Gateway (N. California)						🟡	🟡	🟡	🟡
Amazon API Gateway (N. Virginia)						🟡	🟡	🟡	🟡
Amazon API Gateway (Ohio)						🟡	🟡	🟡	🟡

# AWS Health Dashboard – Your Account



- Previously called AWS Personal Health Dashboard (PHD)
- AWS Account Health Dashboard provides **alerts and remediation guidance** when AWS is experiencing **events that may impact you**.
- While the Service Health Dashboard displays the general status of AWS services, Account Health Dashboard gives you a **personalized view into the performance and availability of the AWS services underlying your AWS resources**.
- The dashboard displays **relevant and timely** information to help you manage events in progress and provides proactive notification to help you plan for scheduled activities.
- Can aggregate data from an entire AWS Organization

# AWS Health Dashboard – Your Account



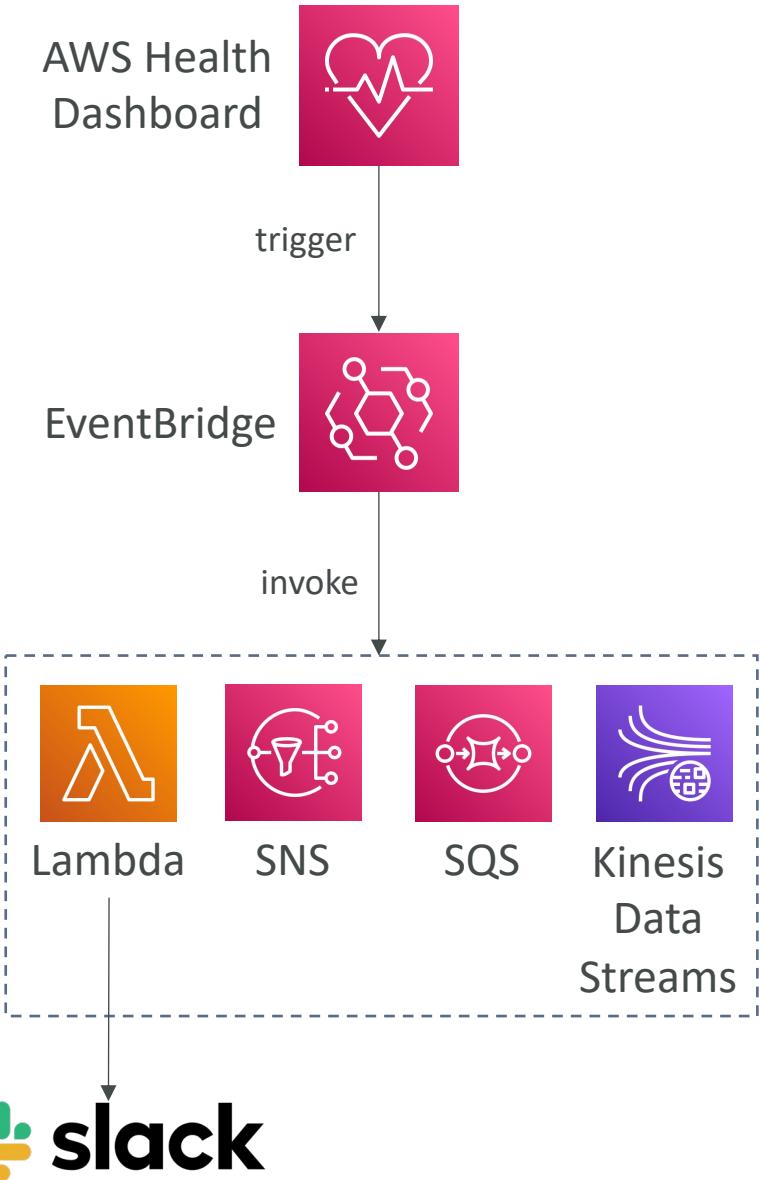
- Global service
- Shows how AWS outages directly impact you & your AWS resources
- Alert, remediation, proactive, scheduled activities

Open issues	0
Scheduled changes	0
Other notifications	0
Event log	

Event log						
<input type="text"/> Add filter						
Event	Status	Event category	Region / Zone	Start time	Last update time	Affected resources
Operational issue - EC2 (Ohio)	Closed	Issue	us-east-2	December 24, 2022 at 2:25:00 AM UTC	December 24, 2022 at 2:38:53 AM UTC	-
Operational issue - Codestar (Oregon)	Closed	Issue	us-west-2	December 21, 2022 at 3:03:57 PM UTC	December 21, 2022 at 4:50:47 PM UTC	-
Operational issue - Amplify (N. Virginia)	Closed	Issue	us-east-1	December 17, 2022 at 2:24:17 PM UTC	December 17, 2022 at 2:43:21 PM UTC	-
Operational issue - Multiple services (Singapore)	Closed	Issue	ap-southeast-1	December 13, 2022 at 10:00:55 PM UTC	December 14, 2022 at 1:01:16 AM UTC	-

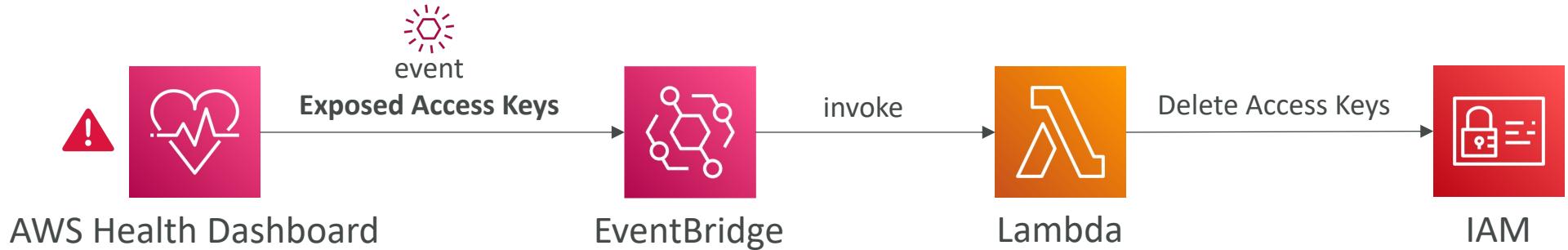
# Health Event Notifications

- Use EventBridge to react to changes for AWS Health events in your AWS account
- Example: receive email notifications when EC2 instances in your AWS account are scheduled for updates
- This is possible for Account events (resources that are affected in your account) and Public Events (Regional availability of a service)
- Use cases: send notifications, capture event information, take corrective action...

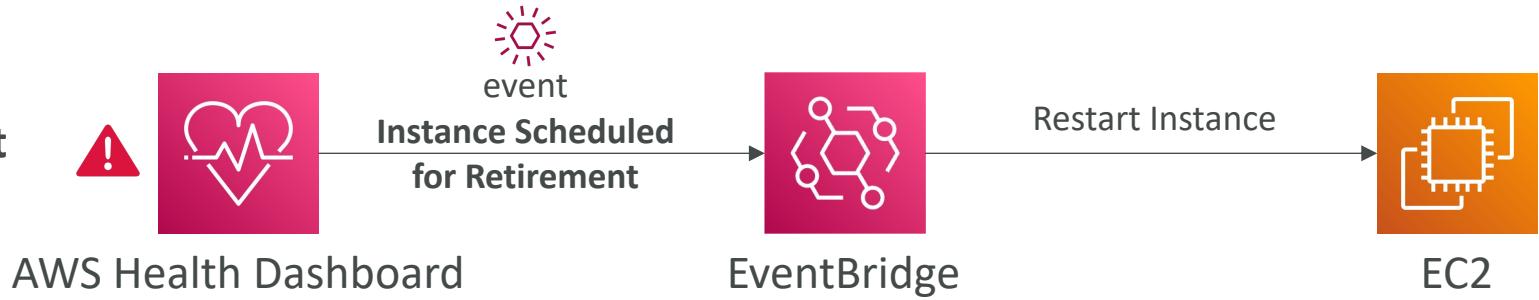


# AWS Health Dashboard – Examples

Remediating Exposed IAM Access Keys

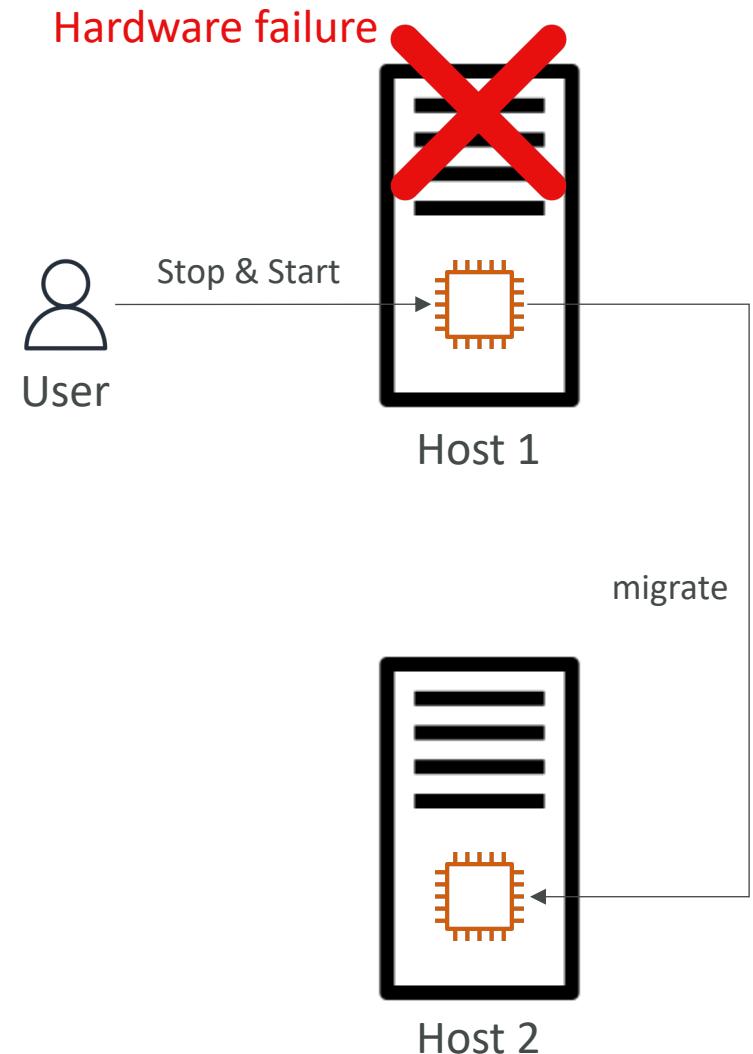


Restarting Instances That are Scheduled for Retirement



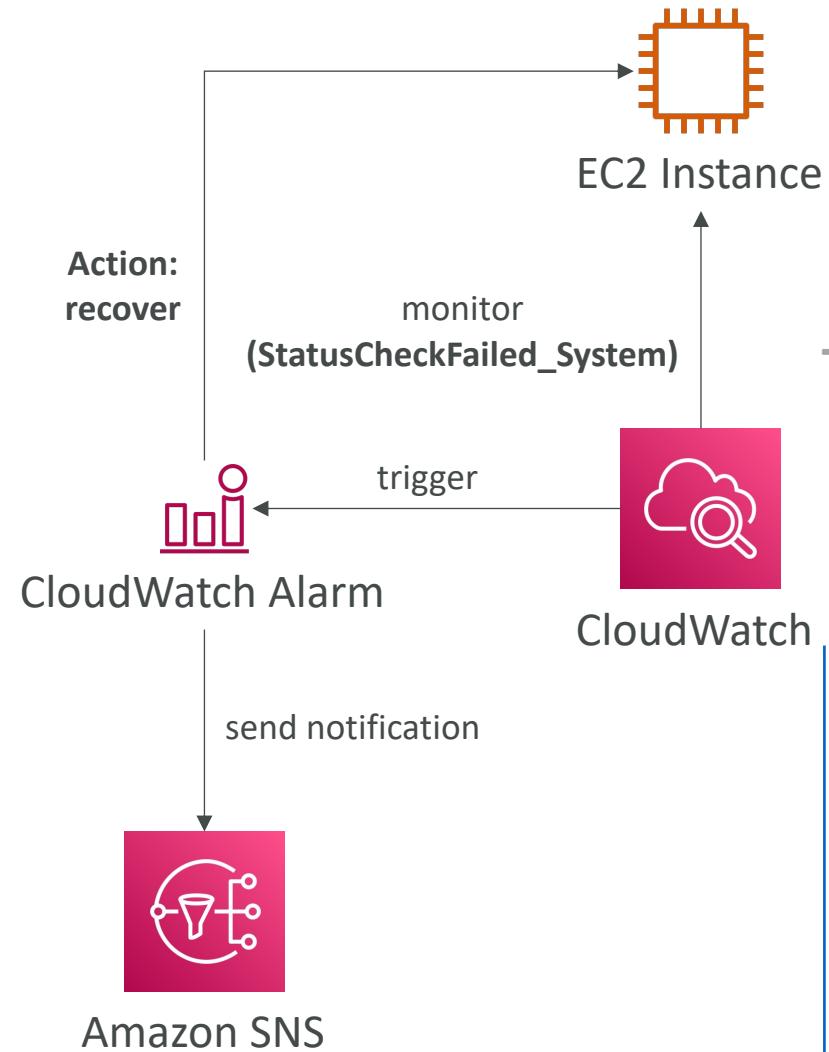
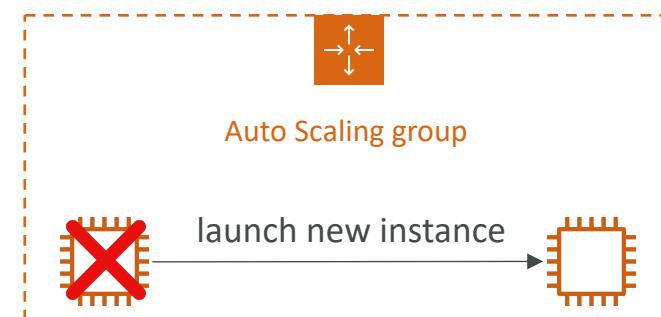
# Status Checks

- Automated checks to identify hardware and software issues
- **System Status Checks**
  - Monitors problems with AWS systems (software/hardware issues on the physical host, loss of system power...)
  - Check **Personal Health Dashboard** for any scheduled critical maintenance by AWS to your instance's host
  - Resolution: stop and start the instance (instance migrated to a new host)
- **Instance Status Checks**
  - Monitors software/network configuration of your instance (invalid network configuration, exhausted memory...)
  - Resolution: reboot the instance or change instance configuration



# Status Checks - CW Metrics & Recovery

- CloudWatch Metrics (1 minute interval)
  - StatusCheckFailed\_System
  - StatusCheckFailed\_Instance
  - StatusCheckFailed (for both)
- Option 1: CloudWatch Alarm
  - Recover EC2 instance with the same private/public IP, EIP, metadata, and Placement Group
  - Send notifications using SNS
- Option 2: Auto Scaling Group
  - Set min/max/desired 1 to recover an instance but won't keep the same private and elastic IP

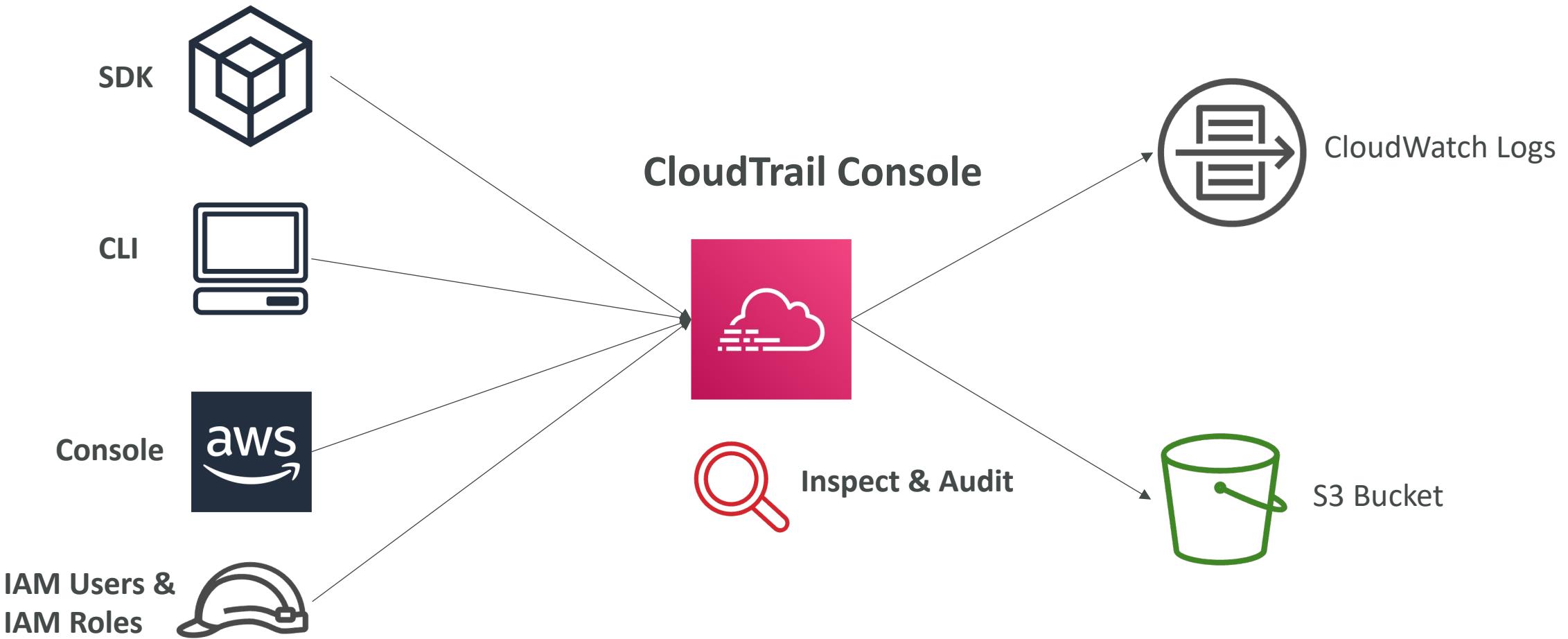


# AWS CloudTrail



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

# CloudTrail Diagram





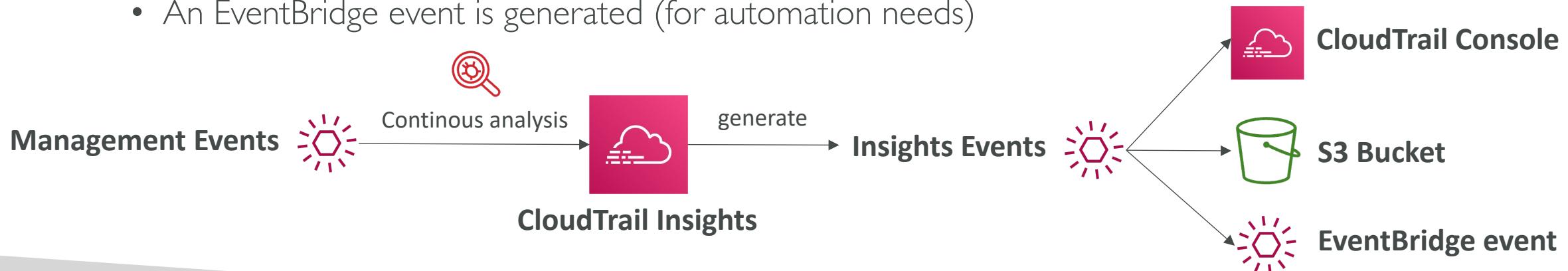
# CloudTrail Events

- Management Events:
  - Operations that are performed on resources in your AWS account
  - Examples:
    - Configuring security (IAM `AttachRolePolicy`)
    - Configuring rules for routing data (Amazon EC2 `CreateSubnet`)
    - Setting up logging (AWS CloudTrail `CreateTrail`)
  - By default, trails are configured to log management events.
  - Can separate Read Events (that don't modify resources) from Write Events (that may modify resources)
- Data Events:
  - By default, data events are not logged (because high volume operations)
  - Amazon S3 object-level activity (ex: `GetObject`, `DeleteObject`, `PutObject`): can separate Read and Write Events
  - AWS Lambda function execution activity (the `Invoke` API)
- CloudTrail Insights Events:
  - See next slide ☺



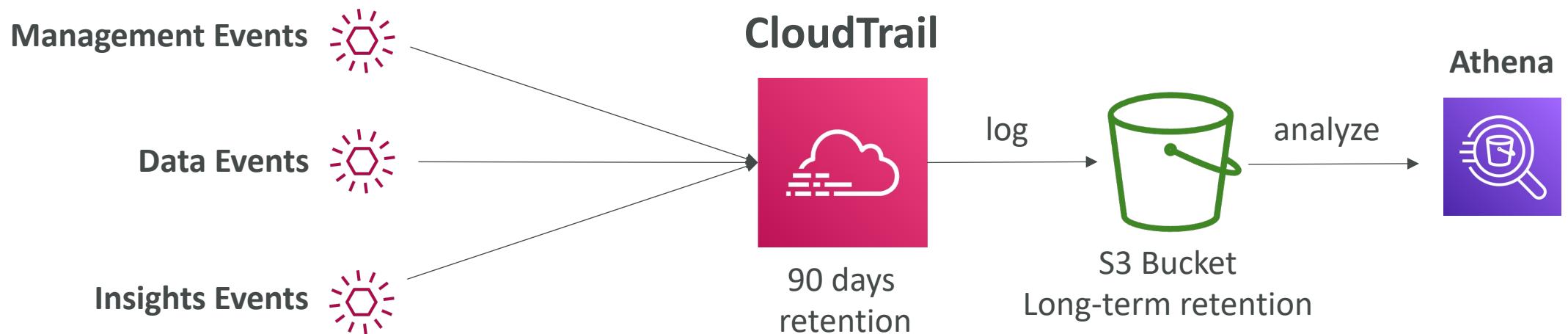
# CloudTrail Insights

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

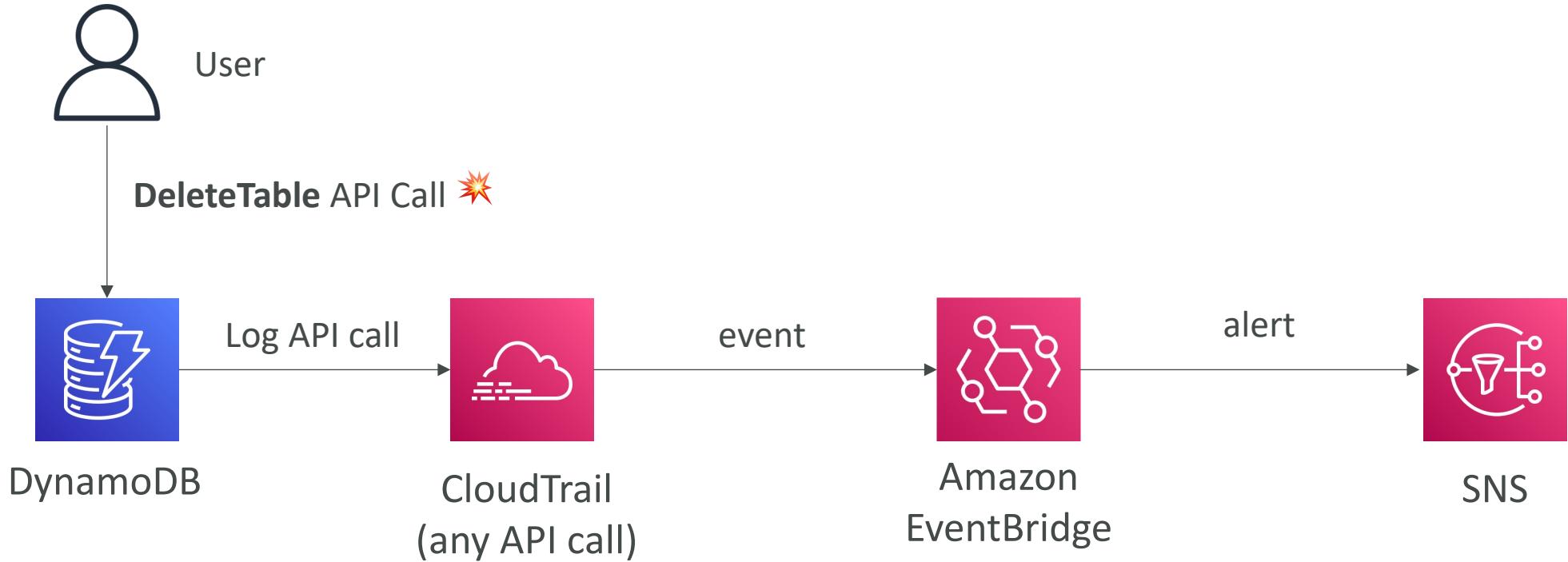


# CloudTrail Events Retention

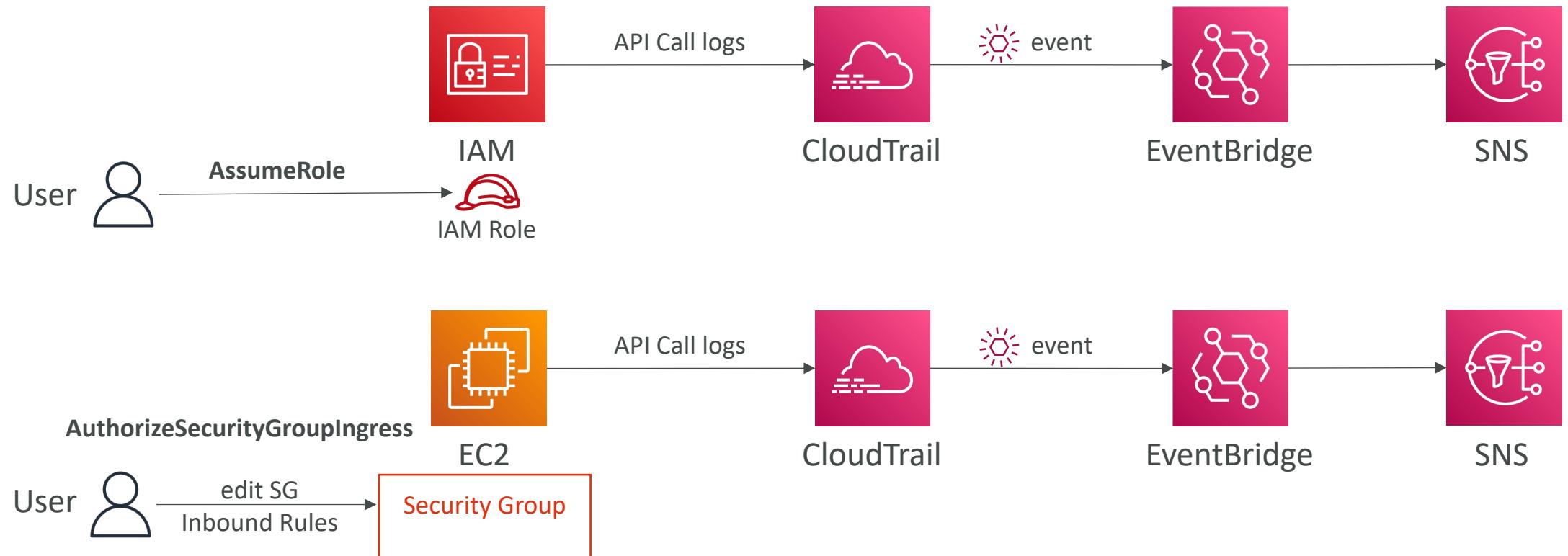
- Events are stored for 90 days in CloudTrail
- To keep events beyond this period, log them to S3 and use Athena



# Amazon EventBridge – Intercept API Calls

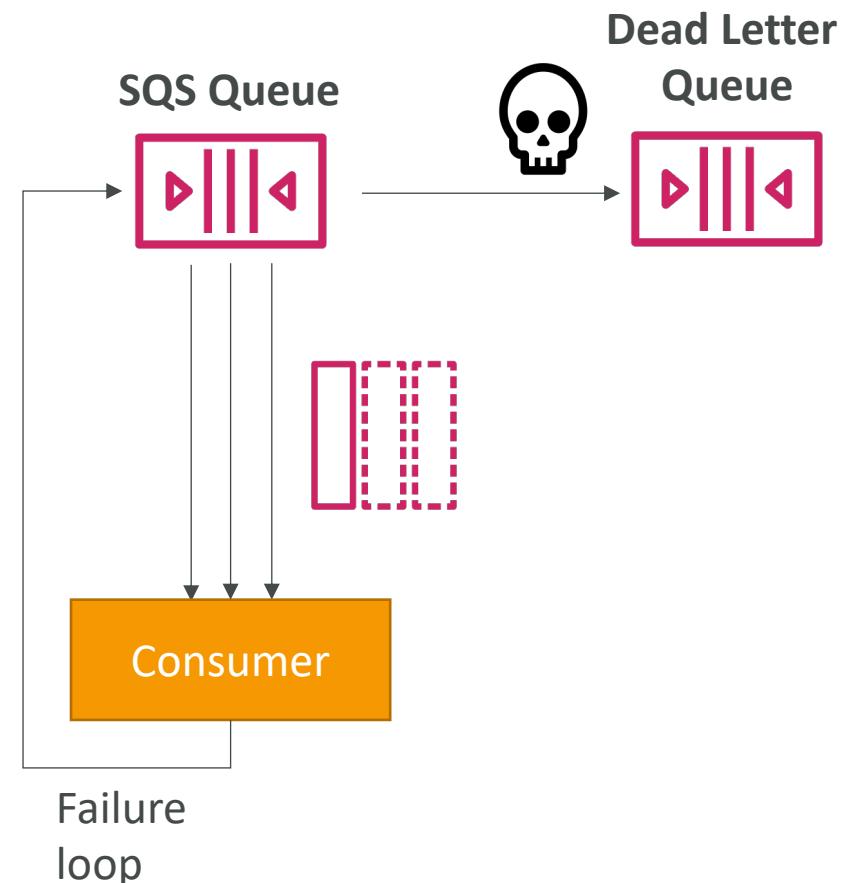


# Amazon EventBridge + CloudTrail



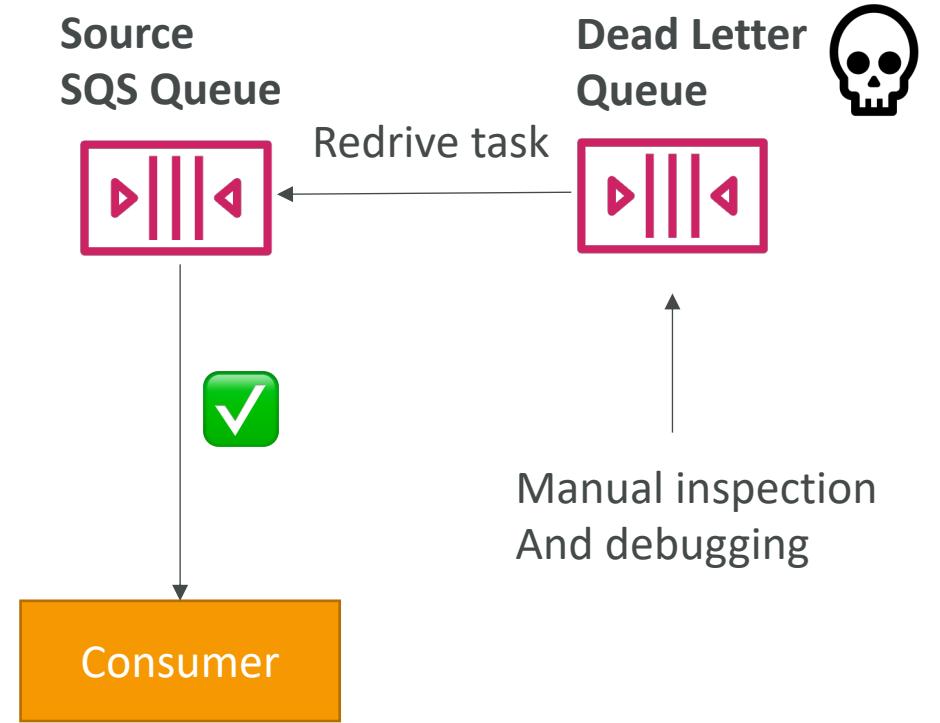
# Amazon SQS – Dead Letter Queue (DLQ)

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



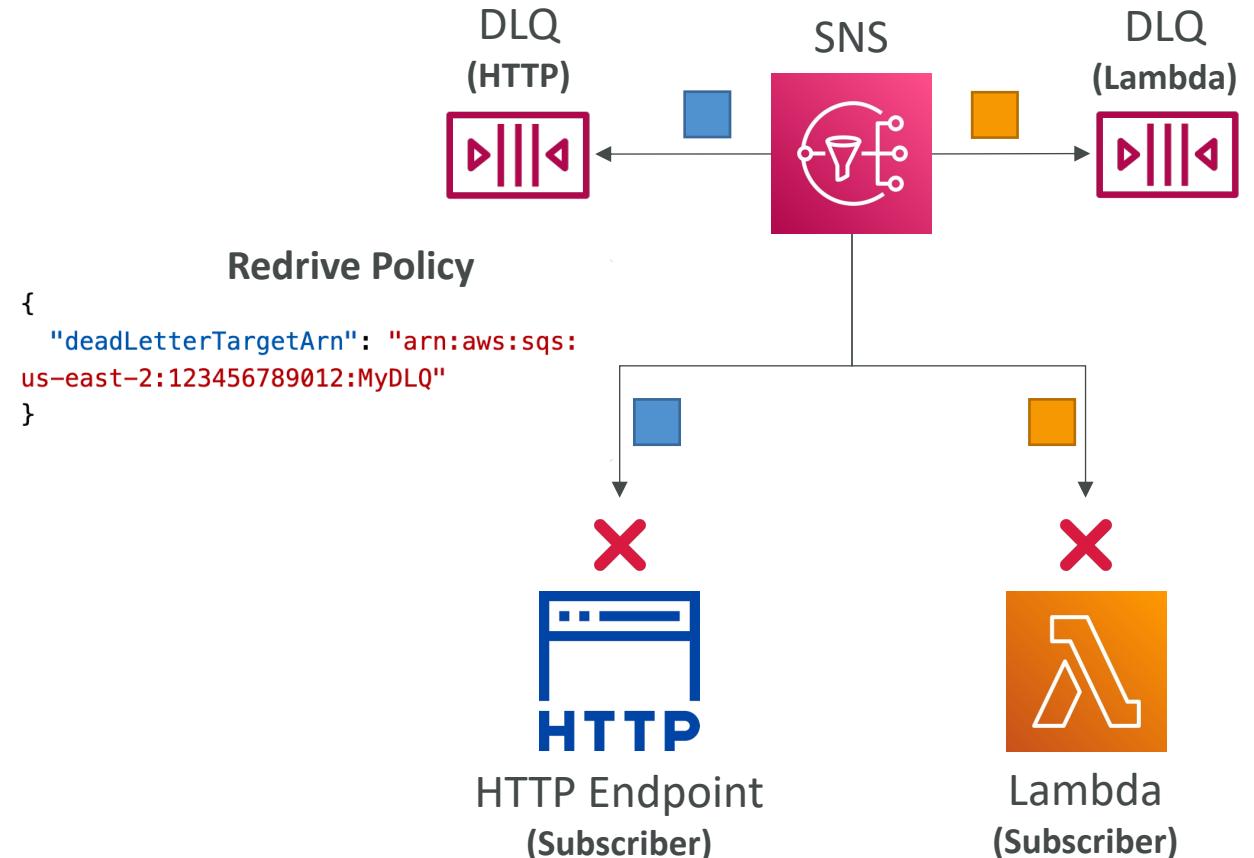
# SQS DLQ – Redrive to Source

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



# Amazon SNS – Dead Letter Queue (DLQ)

- After exhausting the delivery policy (delivery retries), messages that haven't been delivered are discarded unless you set a DLQ (Dead Letter Queue)
- **Redrive Policy** – JSON object that refers to the ARN of the DLQ (SQS or SQS FIFO)
- DLQ is attached to SNS Subscription-level (rather than the SNS Topic)



# AWS X-Ray

## Visual analysis of our applications

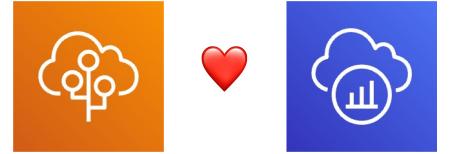


# X-Ray



- Tracing requests across your microservices (distributed systems)
- Integrations with:
  - EC2 – install the X-Ray agent
  - ECS – install the X-Ray agent or Docker container
  - Lambda
  - Beanstalk - agent is automatically installed
  - API Gateway – helpful to debug errors (such as 504)
- The X-Ray agent or services need IAM permissions to X-Ray

# X-Ray with Elastic Beanstalk

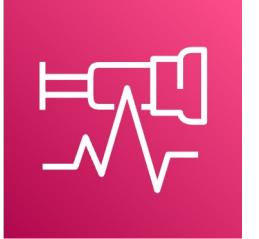


- AWS Elastic Beanstalk platforms include the X-Ray daemon
- You can run the daemon by setting an option in the Elastic Beanstalk console or with a configuration file (in .ebextensions/xray-daemon.config)

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

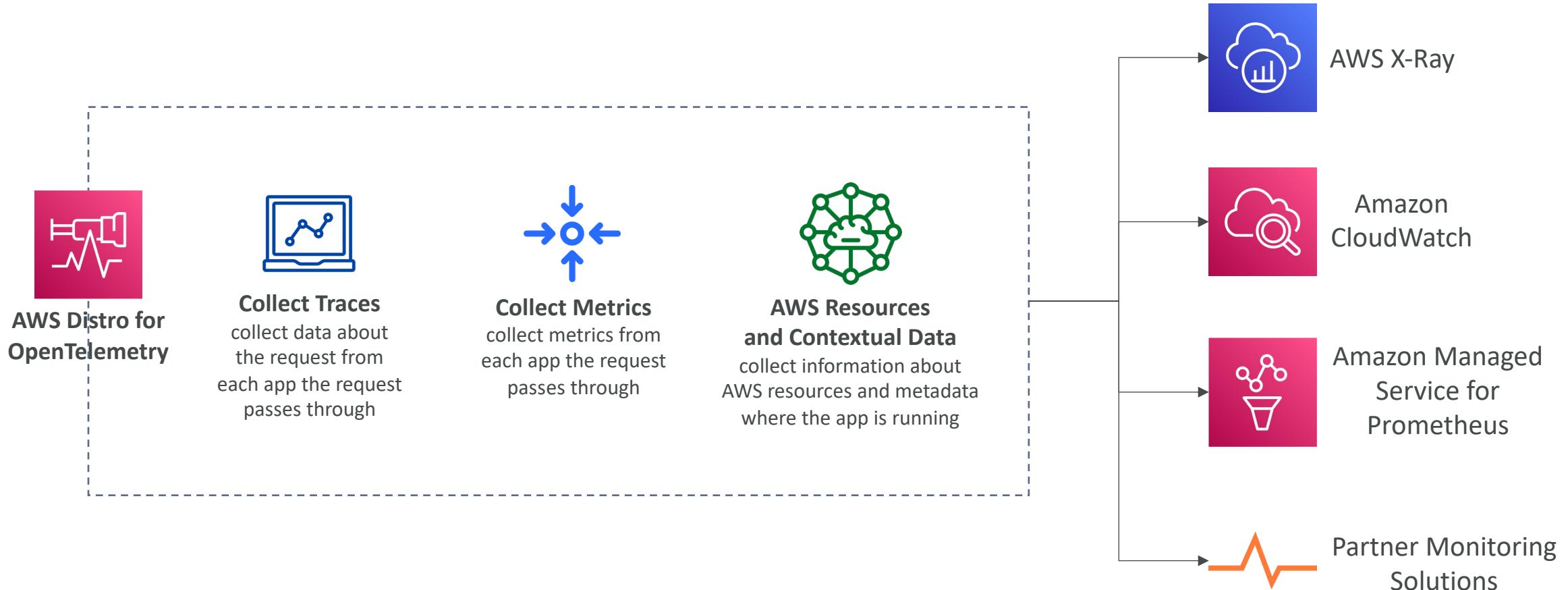
- Make sure to give your instance profile the correct IAM permissions so that the X-Ray daemon can function correctly
- Then make sure your application code is instrumented with the X-Ray SDK
- Note: The X-Ray daemon is not provided for Multicontainer Docker

# AWS Distro for OpenTelemetry



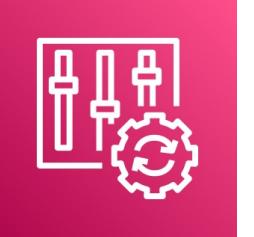
- Secure, production-ready AWS-supported distribution of the open-source project OpenTelemetry project
- Provides a single set of APIs, libraries, agents, and collector services
- Collects distributed traces and metrics from your apps
- Collects metadata from your AWS resources and services
- Auto-instrumentation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

# AWS Distro for OpenTelemetry



# Domain 6 – Security and Compliance

# AWS Config



- Helps with auditing and recording **compliance** of your AWS resources
- Helps record configurations and changes over time
- Questions that can be solved by AWS Config:
  - Is there unrestricted SSH access to my security groups?
  - Do my buckets have any public access?
  - How has my ALB configuration changed over time?
- You can receive alerts (SNS notifications) for any changes
- AWS Config is a per-region service
- Can be aggregated across regions and accounts
- Possibility of storing the configuration data into S3 (analyzed by Athena)

# Config Rules

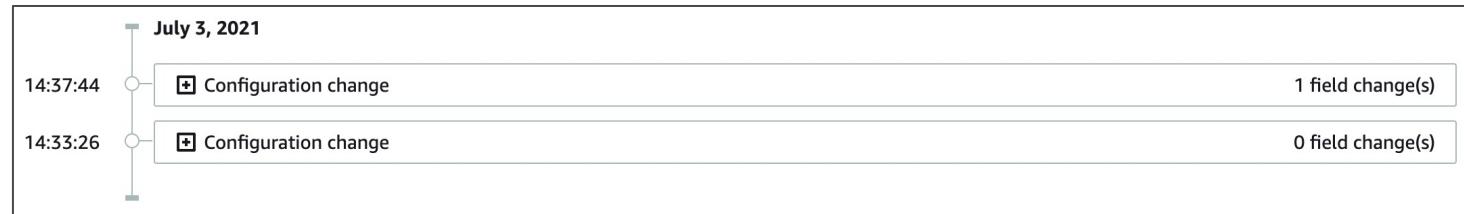
- Can use AWS managed config rules (over 75)
- Can make custom config rules (must be defined in AWS Lambda)
  - Ex: evaluate if each EBS disk is of type gp2
  - Ex: evaluate if each EC2 instance is t2.micro
- Rules can be evaluated / triggered:
  - For each config change
  - And / or: at regular time intervals
- AWS Config Rules does not prevent actions from happening (no deny)
- Pricing: no free tier, \$0.003 per configuration item recorded per region, \$0.001 per config rule evaluation per region

# AWS Config Resource

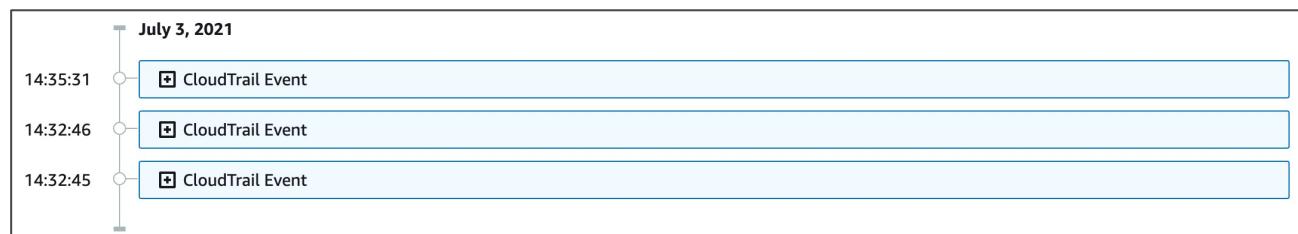
- View compliance of a resource over time

<input type="radio"/> sg-077b425b1649da83e	EC2 SecurityGroup	 Compliant
<input type="radio"/> sg-0831434f1876c0c74	EC2 SecurityGroup	 Noncompliant
<input type="radio"/> sg-09f10ed254d464f30	EC2 SecurityGroup	 Compliant

- View configuration of a resource over time

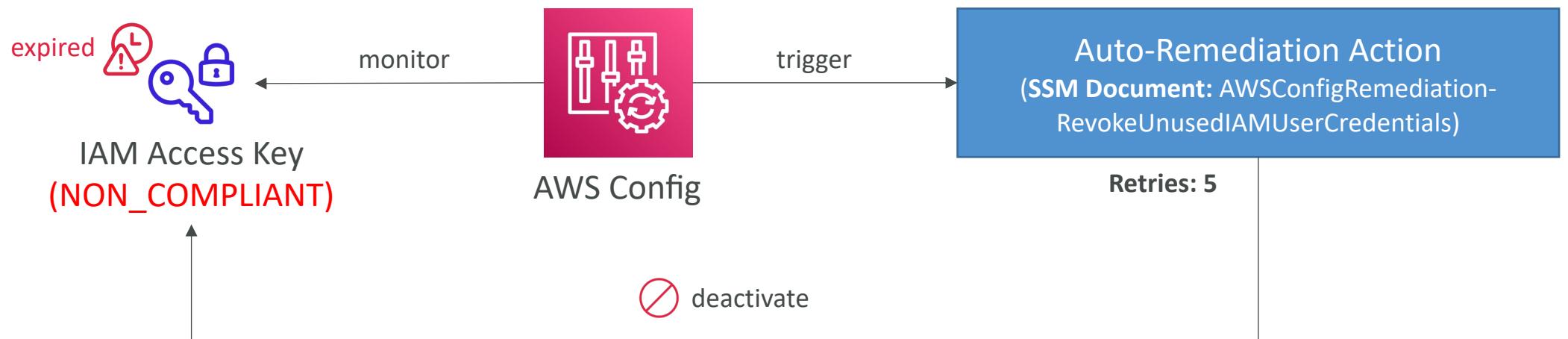


- View CloudTrail API calls of a resource over time



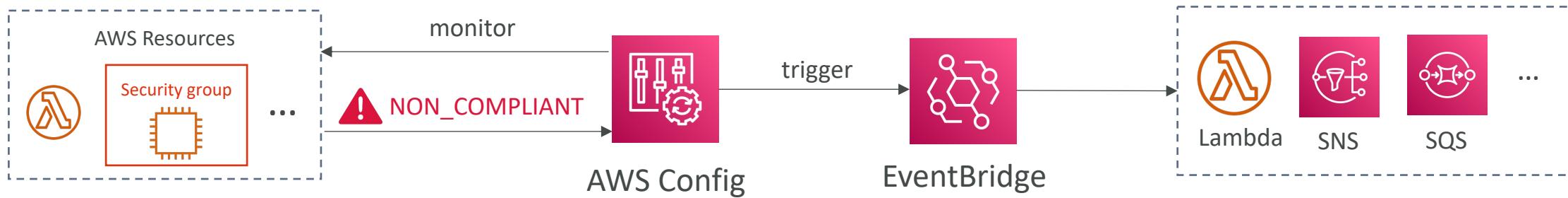
# Config Rules – Remediations

- Automate remediation of non-compliant resources using SSM Automation Documents
- Use AWS-Managed Automation Documents or create custom Automation Documents
  - Tip: you can create custom Automation Documents that invokes Lambda function
- You can set **Remediation Retries** if the resource is still non-compliant after auto-remediation

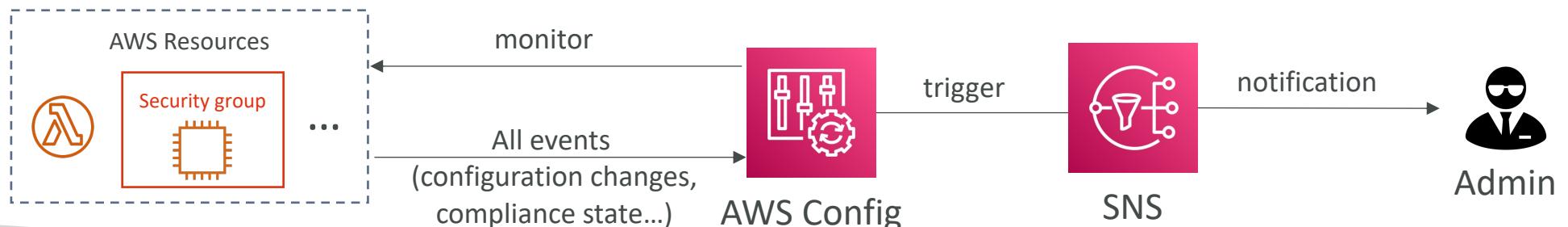


# Config Rules – Notifications

- Use EventBridge to trigger notifications when AWS resources are non-compliant

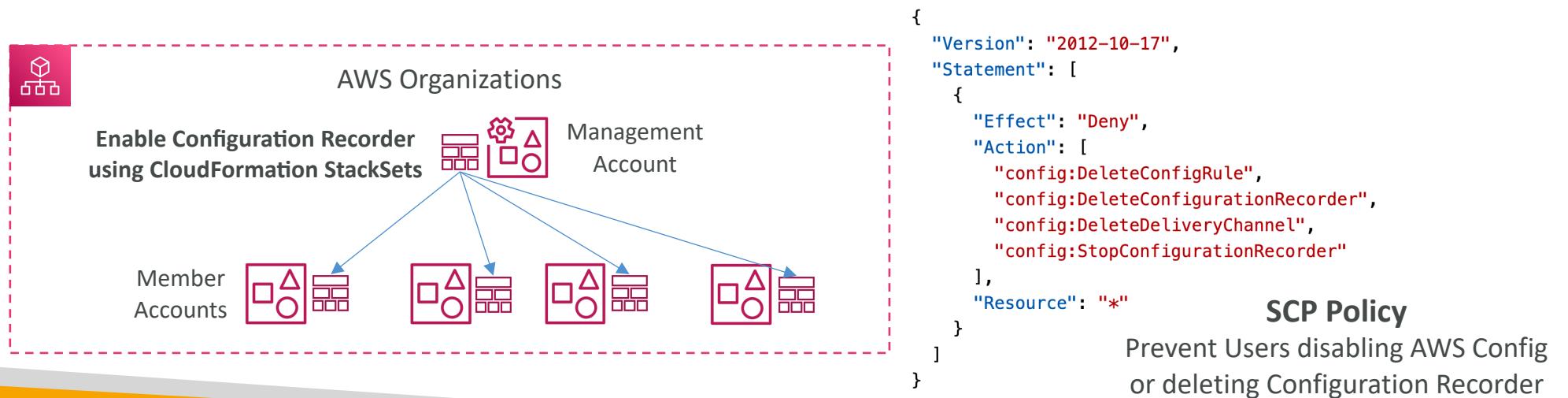


- Ability to send configuration changes and compliance state notifications to SNS (all events – use SNS Filtering or filter at client-side)

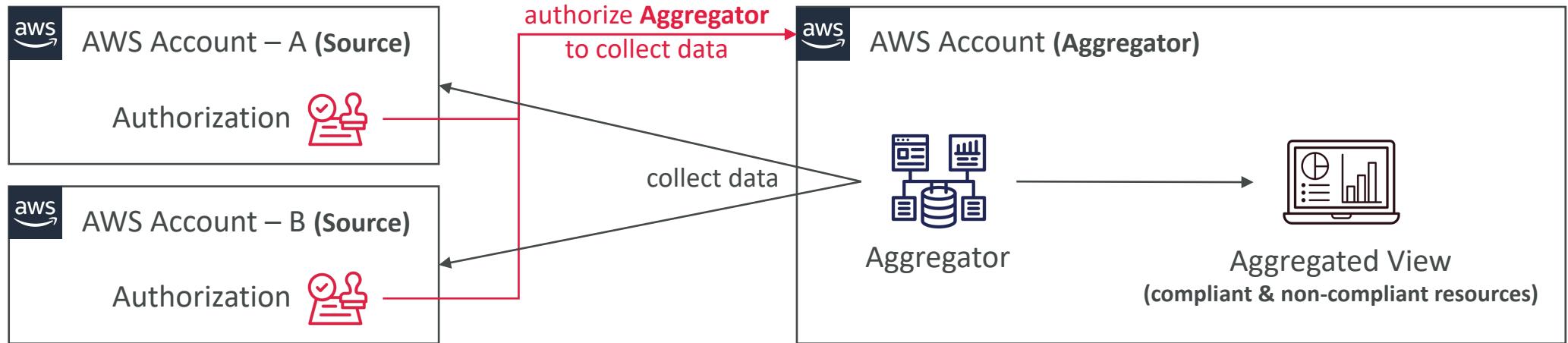


# AWS Config – Configuration Recorder

- Stores the configurations of your AWS resources as Configuration Items
- **Configuration Item** – a point-in-time view of the various attributes of an AWS resource. Created whenever AWS Config detects a change to the resource (e.g., attributes, relationships, config., events...)
- Custom Configuration Recorder to record only the resource types that you specify
- Must be created before AWS Config can track your resources (created automatically when you enable AWS Config using AWS CLI or AWS Console)



# AWS Config – Aggregators



- The aggregator is created in one central aggregator account
- Aggregates rules, resources, etc... across multiple accounts & regions
- If using AWS Organizations, no need for individual Authorization
- Rules are created in each individual source AWS account
- Can deploy rules to multiple target accounts using CloudFormation StackSets

# AWS Config – Conformance Pack

- Collection of AWS Config Rules and Remediation actions
- Packs are created in YAML-formatted files (similar to CloudFormation)
- Deploy to an AWS account and regions or across an AWS Organization
- Pre-built sample Packs or create your own **Custom Conformance Packs**
- Can include **custom Config Rules** which are backed by Lambda functions to evaluate whether your resources are compliant with the Config Rules
- Can pass inputs via Parameters section to make it more flexible
- Can designate a Delegated Administrator to deploy Conformance Packs to your AWS Organization (can be Member account)

# AWS Config – Conformance Pack

## Parameters

```
Parameters:
  IamPasswordPolicyParamMinimumPasswordLength:
    Default: "14"
    Type: String
  AccessKeysRotatedParamMaxAccessKeyAge:
    Default: "90"
    Type: String
  IamUserUnusedCredentialsCheckParamMaxCredentialUsageAge:
    Default: "45"
    Type: String
```

```
Resources:
  IamPasswordPolicy:
    Properties:
      ConfigRuleName: iam-password-policy
    InputParameters:
      MinimumPasswordLength: IamPasswordPolicyParamMinimumPasswordLength
    Source:
      Owner: AWS
      SourceIdentifier: IAM_PASSWORD_POLICY
    Type: AWS::Config::ConfigRule
```

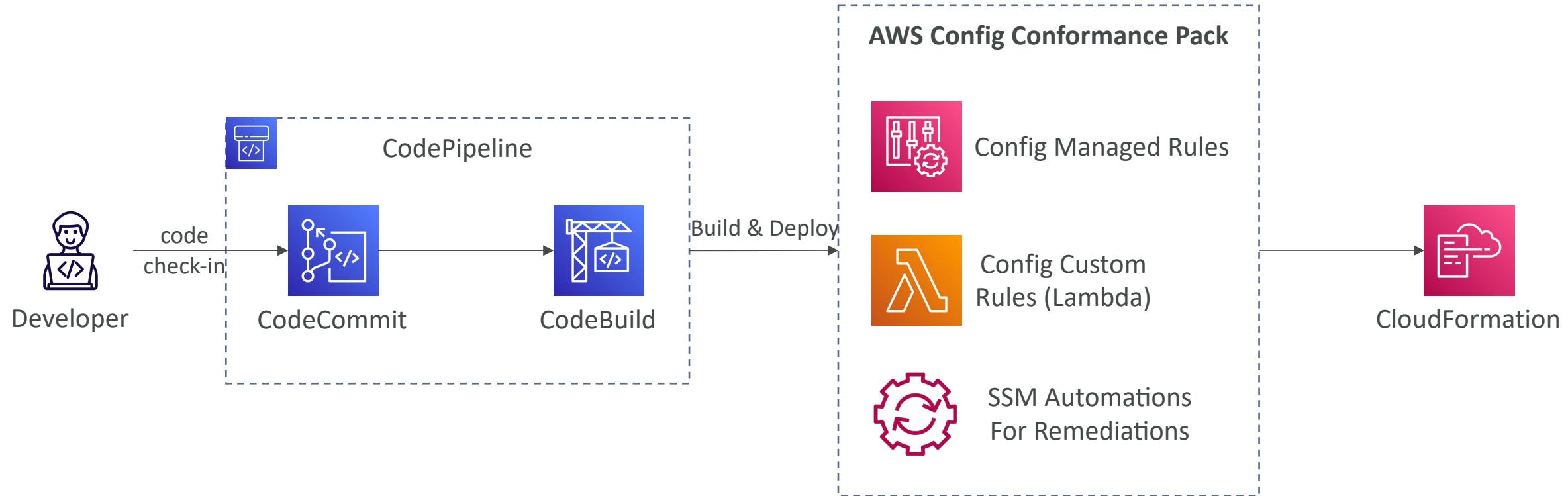
## Config Rule

```
AccessKeysRotated:
  Properties:
    ConfigRuleName: access-keys-rotated
  InputParameters:
    maxAccessKeyAge: AccessKeysRotatedParamMaxAccessKeyAge
  Source:
    Owner: AWS
    SourceIdentifier: ACCESS_KEYS_ROTATED
  Type: AWS::Config::ConfigRule
```

```
IamUserUnusedCredentialsCheck:
  Properties:
    ConfigRuleName: iam-user-unused-credentials-check
  InputParameters:
    maxCredentialUsageAge: IamUserUnusedCredentialsCheckParamMaxCredentialUsageAge
  Source:
    Owner: AWS
    SourceIdentifier: IAM_USER_UNUSED_CREDENTIALS_CHECK
  Type: AWS::Config::ConfigRule
```

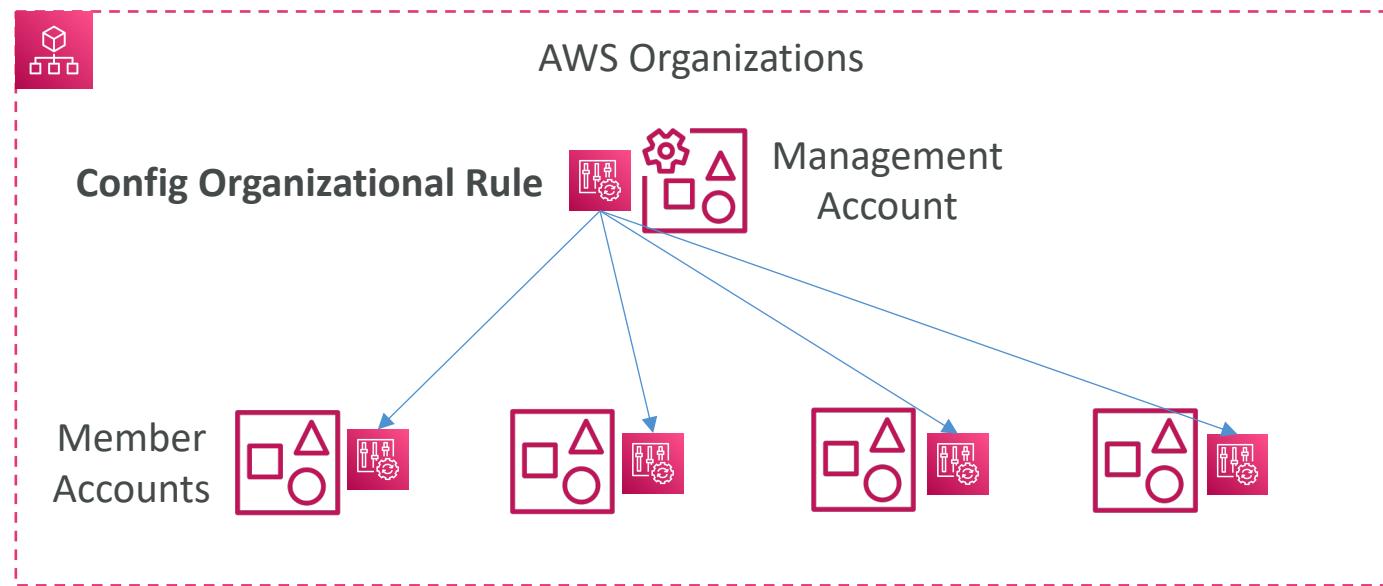
**AWS Config Managed Rule**  
use Owner: CUSTOM\_LAMBDA  
for AWS Config Custom Rules

# AWS Config – Conformance Pack – CICD



# AWS Config – Organizational Rules

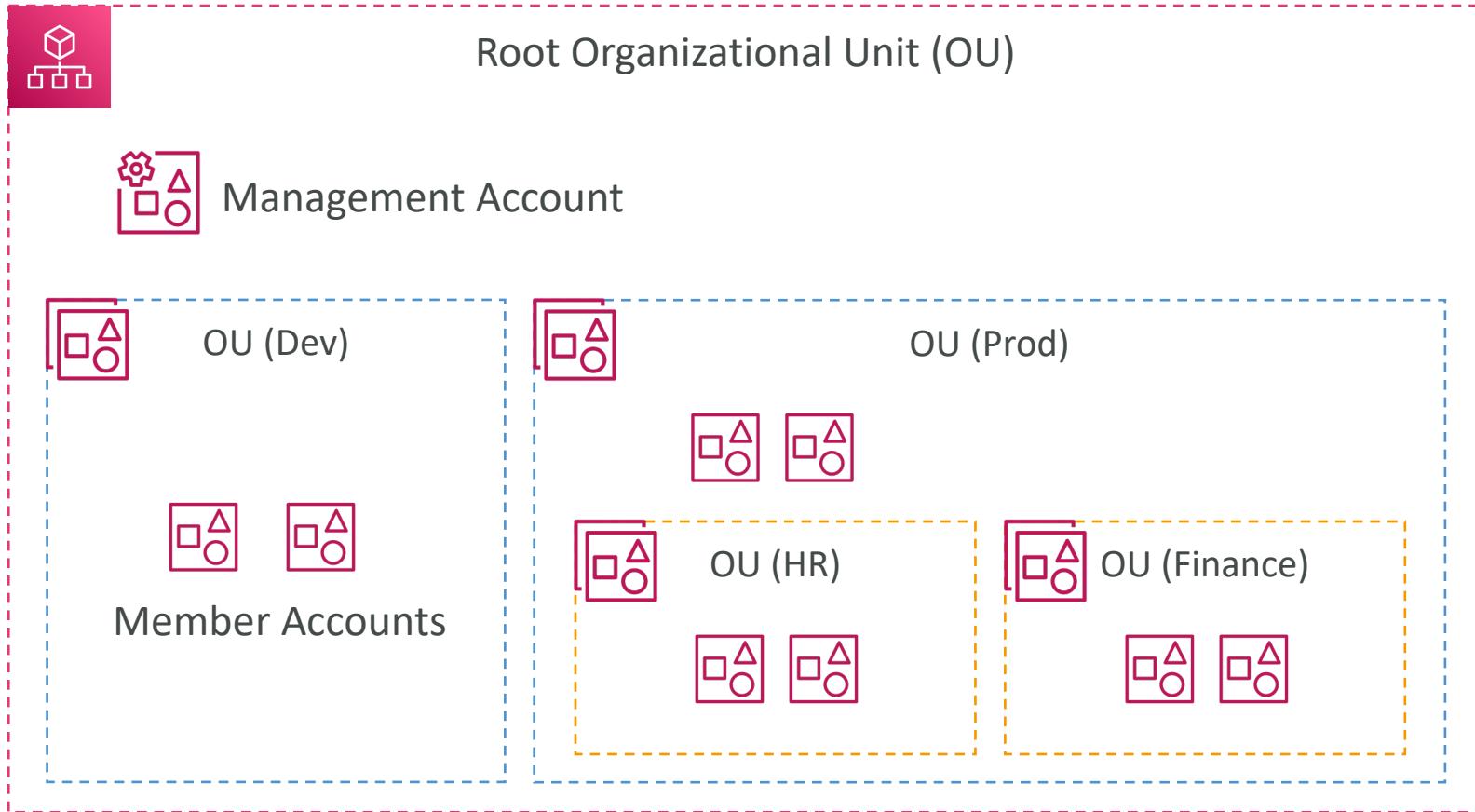
- AWS Config Rule that you can manage across all accounts within an AWS Organization



# AWS Config – Organizational Rules vs. Conformance Packs

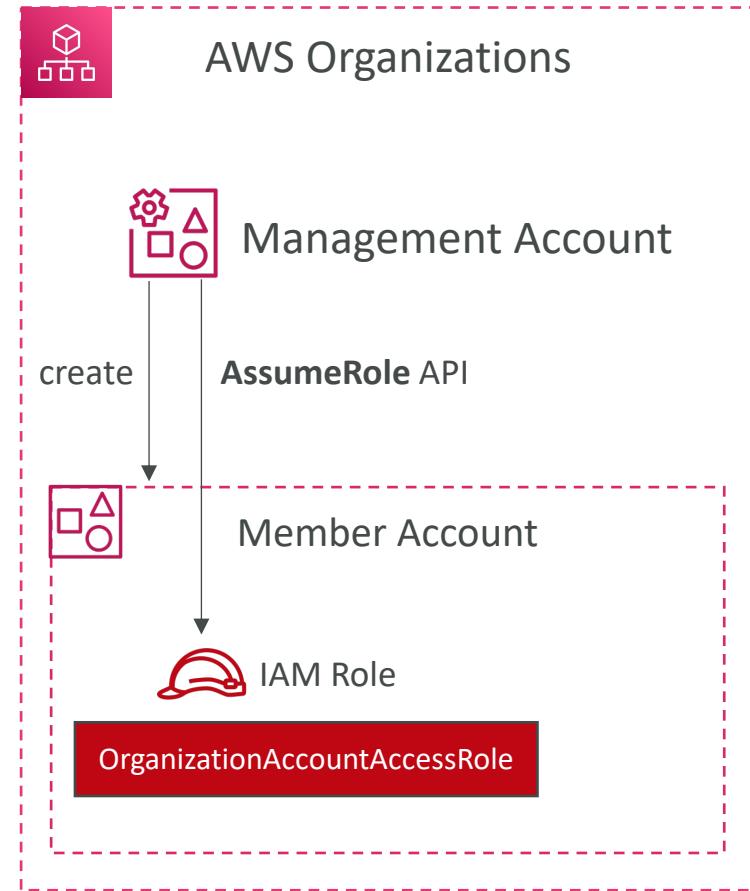
	Organizational Rules	Conformance Packs
Scope	AWS Organization	AWS Accounts and AWS Organization
Evaluation Type	Evaluate resources against predefined rules that are defined and enforced at Organization level	Evaluate resources against predefined rules that are defined and enforced at the Account level
Rules Count	One rule	Many rules at a time
Compliance Level	Managed at the Organizational level	Managed at the Account level

# AWS Organizations



# AWS Organizations - OrganizationAccountAccessRole

- IAM role which grants full administrator permissions in the Member account to the Management account
- Used to perform admin tasks in the Member accounts (e.g., creating IAM users)
- Could be assumed by IAM users in the Management account
- Automatically added to all new Member accounts created with AWS Organizations
- Must be created manually if you invite an existing Member account

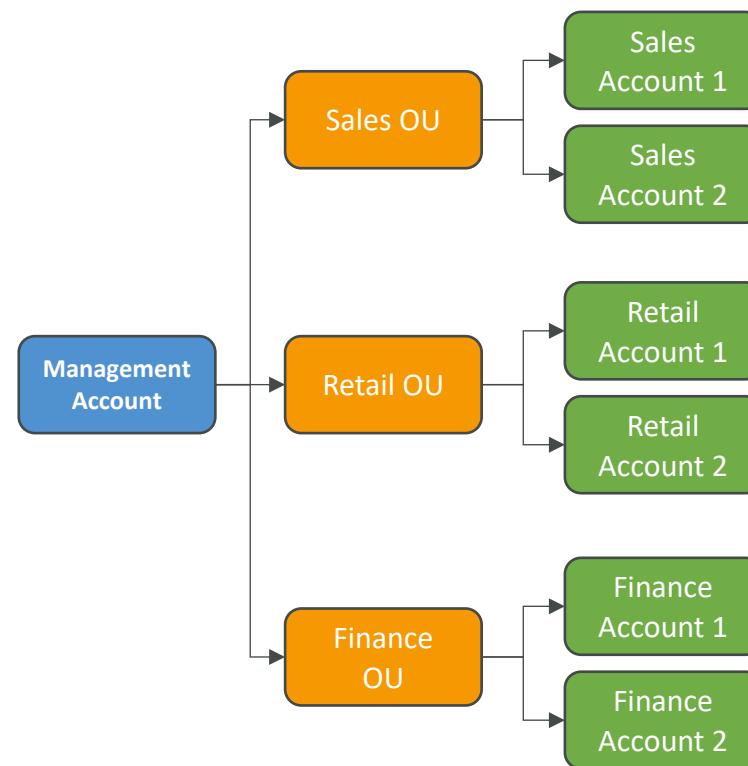


# Multi Account Strategies

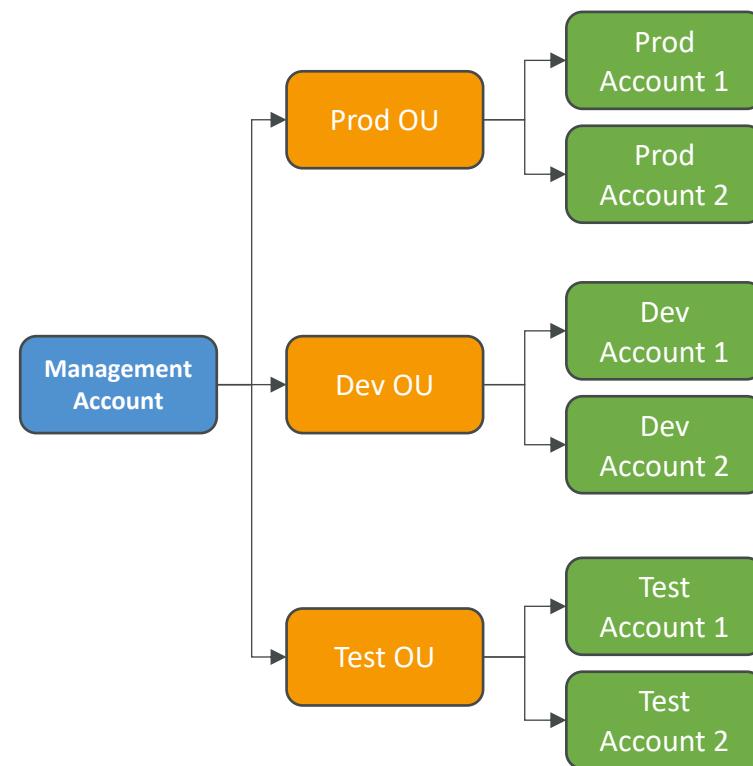
- Create accounts per **department**, per **cost center**, per **dev / test / prod**, based on **regulatory restrictions** (using SCP), for **better resource isolation** (ex:VPC), to have **separate per-account service limits**, isolated account for **logging**,
- Multi Account vs. One Account Multi VPC
- Use tagging standards for billing purposes
- Enable CloudTrail on all accounts, send logs to central S3 account
- Send CloudWatch Logs to central logging account
- Strategy to create an account for security

# Organizational Units (OU) - Examples

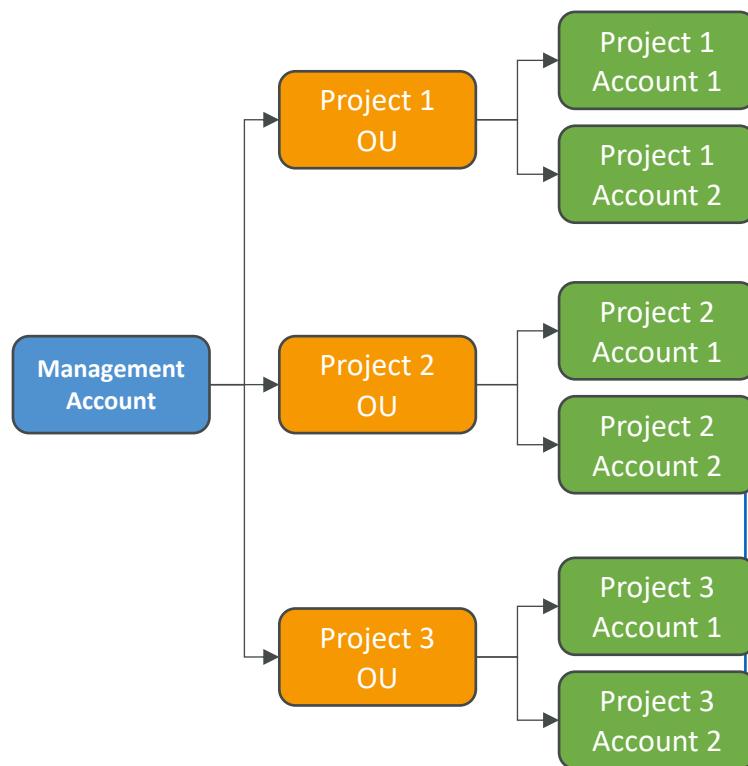
## Business Unit



## Environmental Lifecycle



## Project-Based



# AWS Organization - Feature Modes

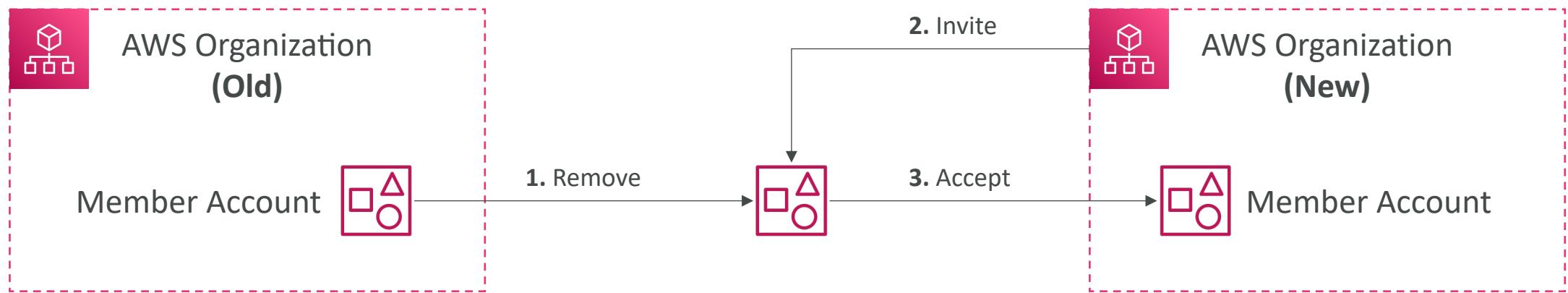
- Consolidated billing features:
  - Consolidated Billing across all accounts - single payment method
  - Pricing benefits from aggregated usage (volume discount for EC2, S3...)
- All Features (Default):
  - Includes consolidated billing features, SCP
  - Invited accounts must approve enabling all features
  - Ability to apply an SCP to prevent member accounts from leaving the org
  - Can't switch back to Consolidated Billing Features only

# AWS Organizations – Reserved Instances

- For billing purposes, the consolidated billing feature of AWS Organizations treats all the accounts in the organization as one account.
- This means that **all accounts** in the organization can receive the hourly cost benefit of Reserved Instances that are purchased by **any other account**.
- **The payer account (Management account) of an organization** can turn off Reserved Instance (RI) discount and Savings Plans discount sharing for any accounts in that organization, including the payer account
- This means that RIs and Savings Plans discounts aren't shared between any accounts that have sharing turned off.
- To share an RI or Savings Plans discount with an account, both accounts must have sharing turned on

# AWS Organizations – Moving Accounts

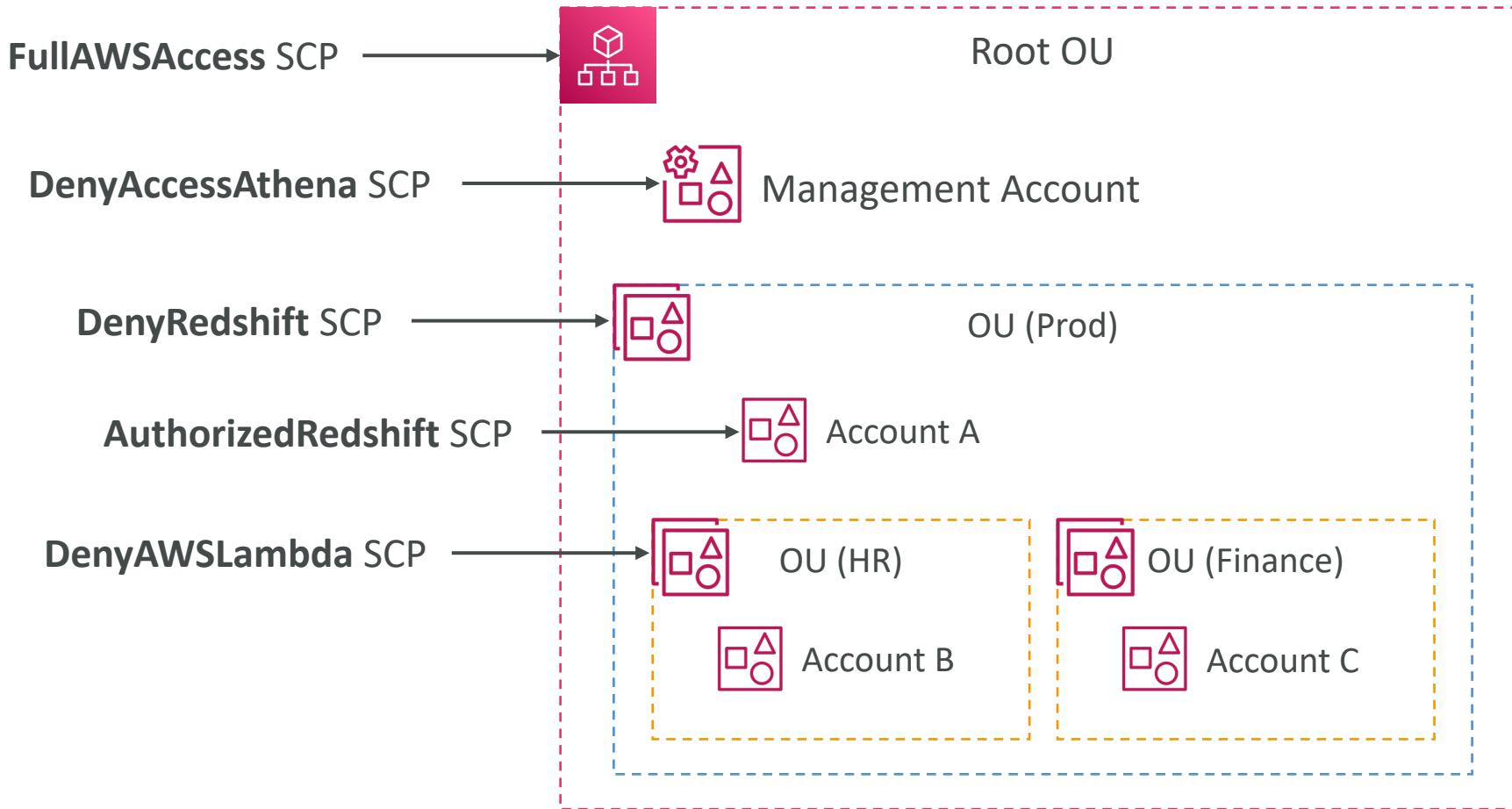
1. Remove the member account from the AWS Organization
2. Send an invite to the member account from the AWS Organization
3. Accept the invite to the new Organization from the member account



# Service Control Policies (SCP)

- Define allowlist or blocklist IAM actions
- Applied at the **OU** or **Account** level
- Does not apply to the Management Account
- SCP is applied to all the **Users and Roles** in the account, including Root user
- The SCP does not affect Service-linked roles
  - Service-linked roles enable other AWS services to integrate with AWS Organizations and can't be restricted by SCPs.
- SCP must have an explicit Allow (does not allow anything by default)
- Use cases:
  - Restrict access to certain services (for example: can't use EMR)
  - Enforce PCI compliance by explicitly disabling services

# SCP Hierarchy



- Management Account
  - Can do anything
  - (no SCP apply)
- Account A
  - Can do anything
  - EXCEPT access Redshift (explicit Deny from OU)
- Account B
  - Can do anything
  - EXCEPT access Redshift (explicit Deny from Prod OU)
  - EXCEPT access Lambda (explicit Deny from HR OU)
- Account C
  - Can do anything
  - EXCEPT access Redshift (explicit Deny from Prod OU)

# SCP Examples

## Blocklist and Allowlist strategies

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

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

More examples: [https://docs.aws.amazon.com/organizations/latest/userguide/orgs\\_manage\\_policies\\_example-scps.html](https://docs.aws.amazon.com/organizations/latest/userguide/orgs_manage_policies_example-scps.html)

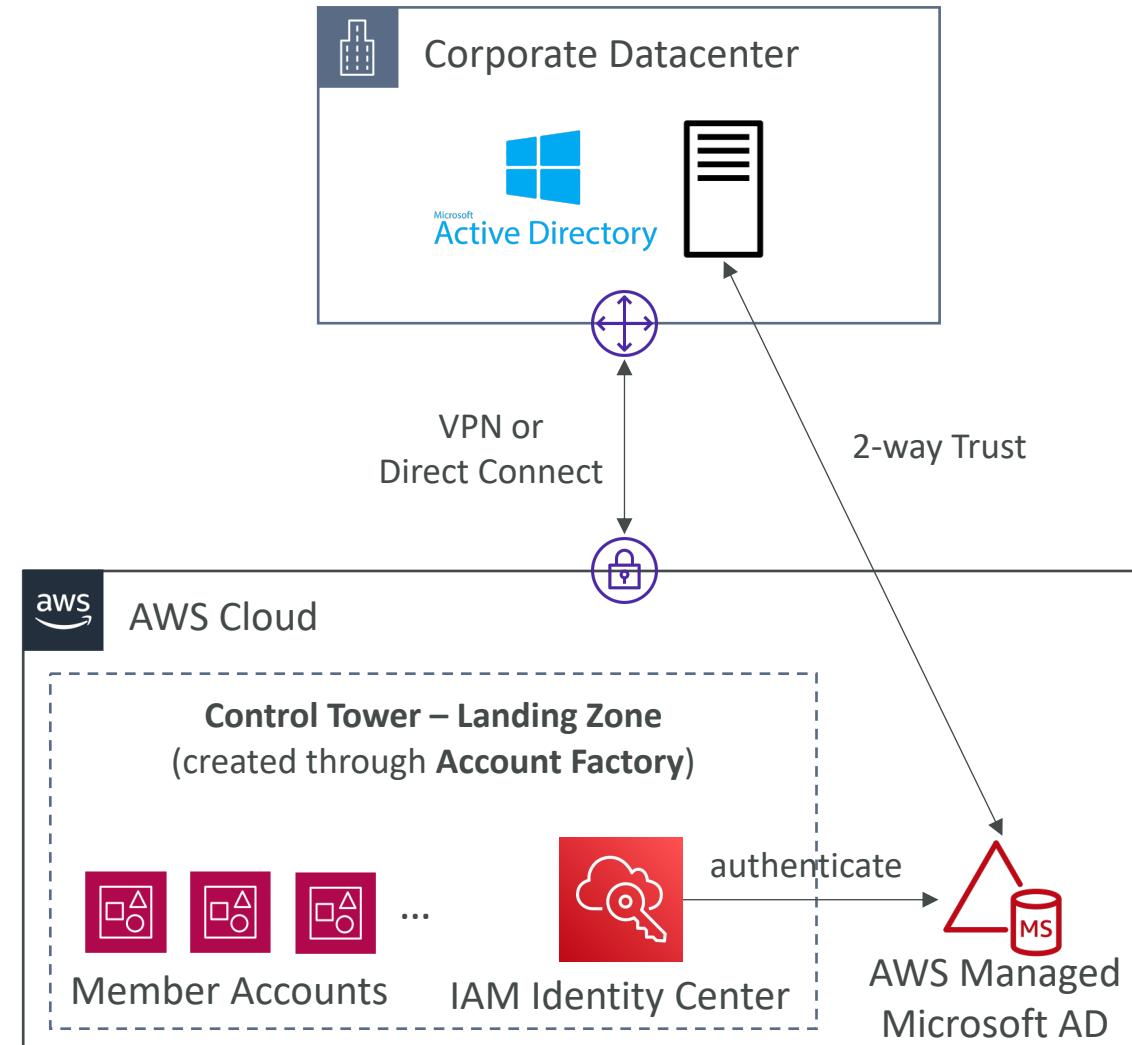
# AWS Control Tower



- Easy way to set up and govern a secure and compliant multi-account AWS environment based on best practices
- Benefits:
  - Automate the set up of your environment in a few clicks
  - Automate ongoing policy management using guardrails
  - Detect policy violations and remediate them
  - Monitor compliance through an interactive dashboard
- AWS Control Tower runs on top of AWS Organizations:
  - It automatically sets up AWS Organizations to organize accounts and implement SCPs (Service Control Policies)

# AWS Control Tower – Account Factory

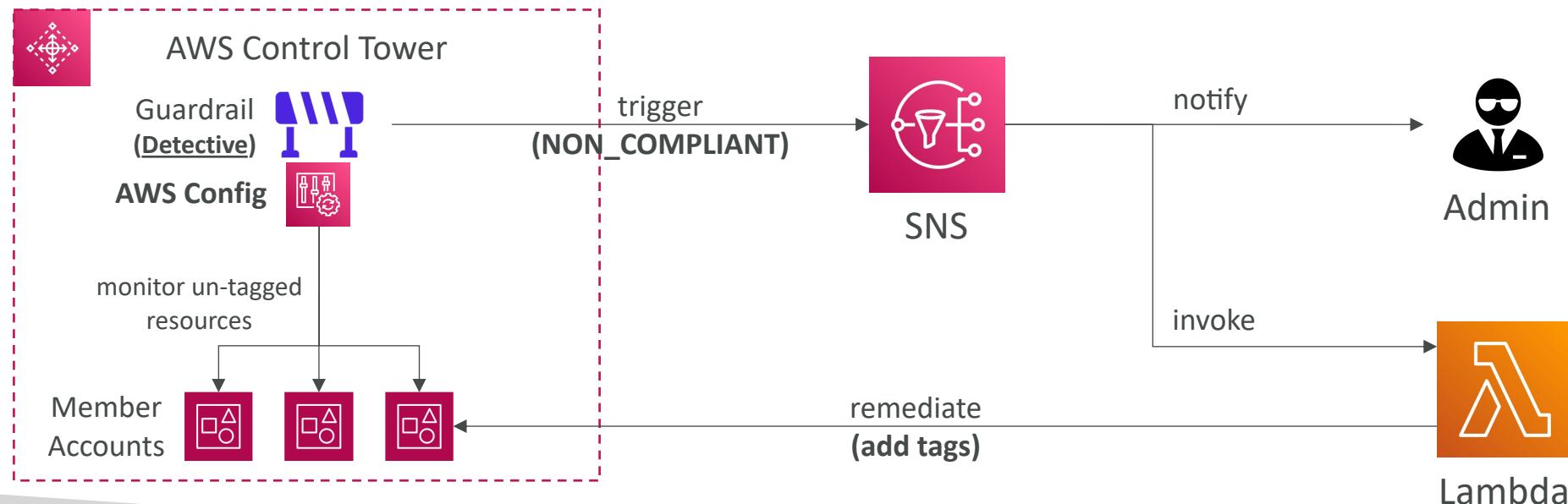
- Automates account provisioning and deployments
- Enables you to create pre-approved baselines and configuration options for AWS accounts in your organization (e.g., VPC default configuration, subnets, region, ...)
- Uses AWS Service Catalog to provision new AWS accounts



# AWS Control Tower – Detect and Remediate Policy Violations

- Guardrail

- Provides ongoing governance for your Control Tower environment (AWS Accounts)
- Preventive – using SCPs (e.g., Disallow Creation of Access Keys for the Root User)
- Detective – using AWS Config (e.g., Detect Whether MFA for the Root User is Enabled)
- Example: identify non-compliant resources (e.g., untagged resources)



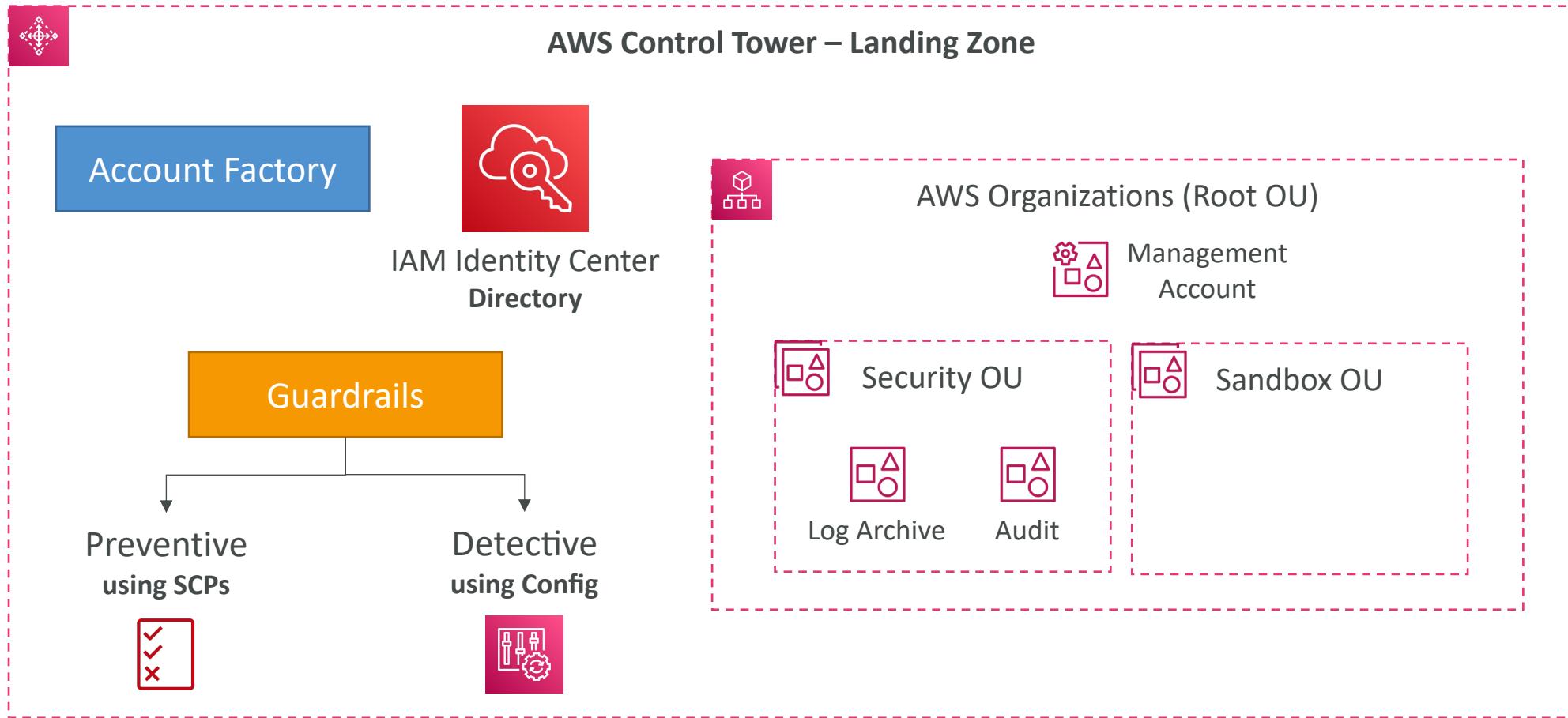
# AWS Control Tower – Guardrails Levels

- **Mandatory**
  - Automatically enabled and enforced by AWS Control Tower
  - Example: Disallow public Read access to the Log Archive account
- **Strongly Recommended**
  - Based on AWS best practices (optional)
  - Example: Enable encryption for EBS volumes attached to EC2 instances
- **Elective**
  - Commonly used by enterprises (optional)
  - Example: Disallow delete actions without MFA in S3 buckets

# AWS Control Tower – Landing Zone

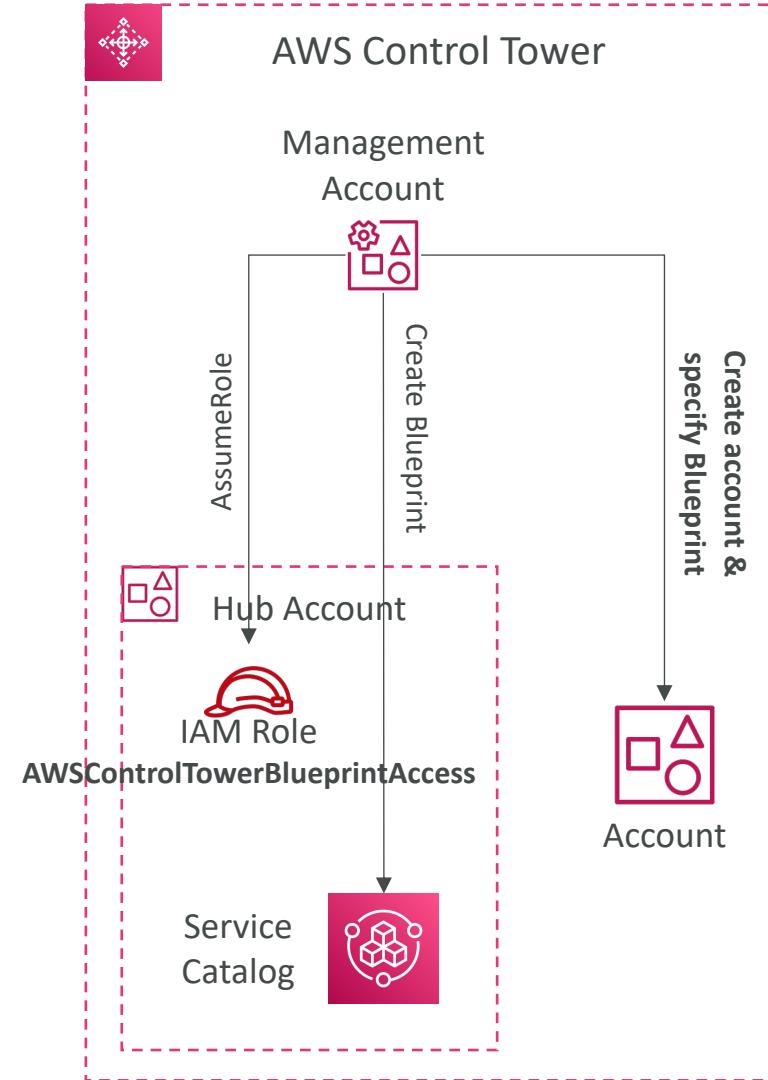
- Automatically provisioned, secure, and compliant multi-account environment based on AWS best practices
- Landing Zone consists of:
  - AWS Organization – create and manage multi-account structure
  - Account Factory – easily configure new accounts to adhere to config. and policies
  - Organizational Units (OUs) – group and categorize accounts based on purpose
  - Service Control Policies (SCPs) – enforce fine-grained permissions and restrictions
  - IAM Identity Center – centrally manage user access to accounts and services
  - Guardrails – rules and policies to enforce security, compliance and best practices
  - AWS Config – to monitor and assess your resources' compliance with Guardrails

# AWS Control Tower – Landing Zone



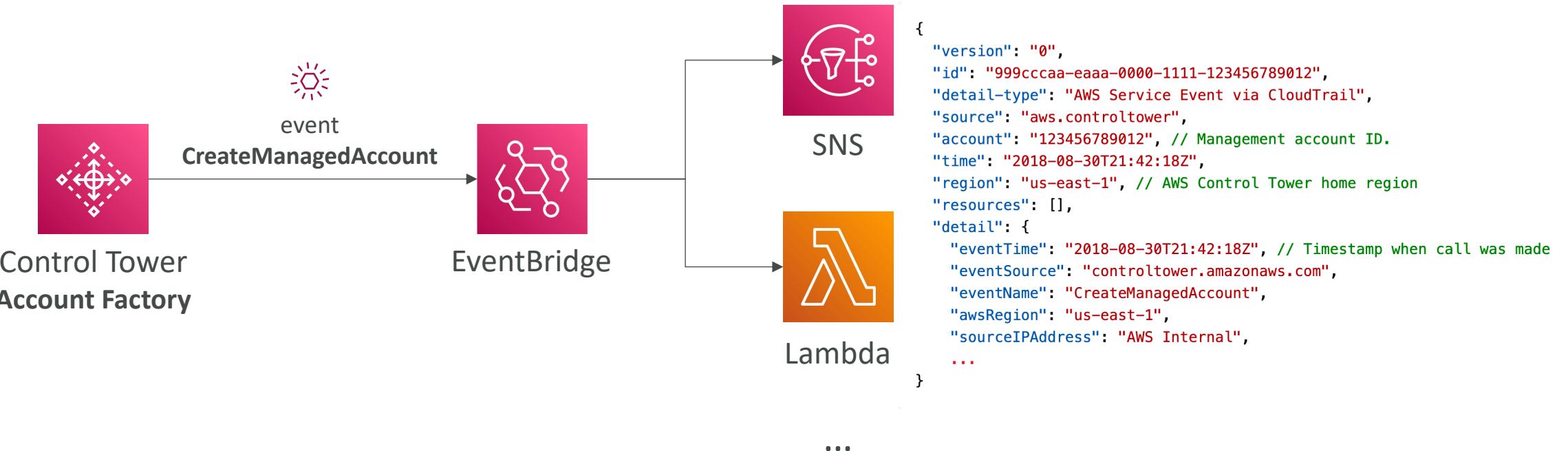
# AWS Control Tower – Account Factory Customization (AFC)

- Automatically customize resources in **new and existing accounts** created through Account Factory
- **Custom Blueprint**
  - CloudFormation template that defines the resources and configurations you want to customize in the account
  - Defined in the form on a Service Catalog Product
  - Stored in a **Hub Account**, which stores all the Custom Blueprints (**recommended**: don't use the Management account)
  - Also available pre-defined blueprints created by AWS Partners
- Only one blueprint can be deployed to the account

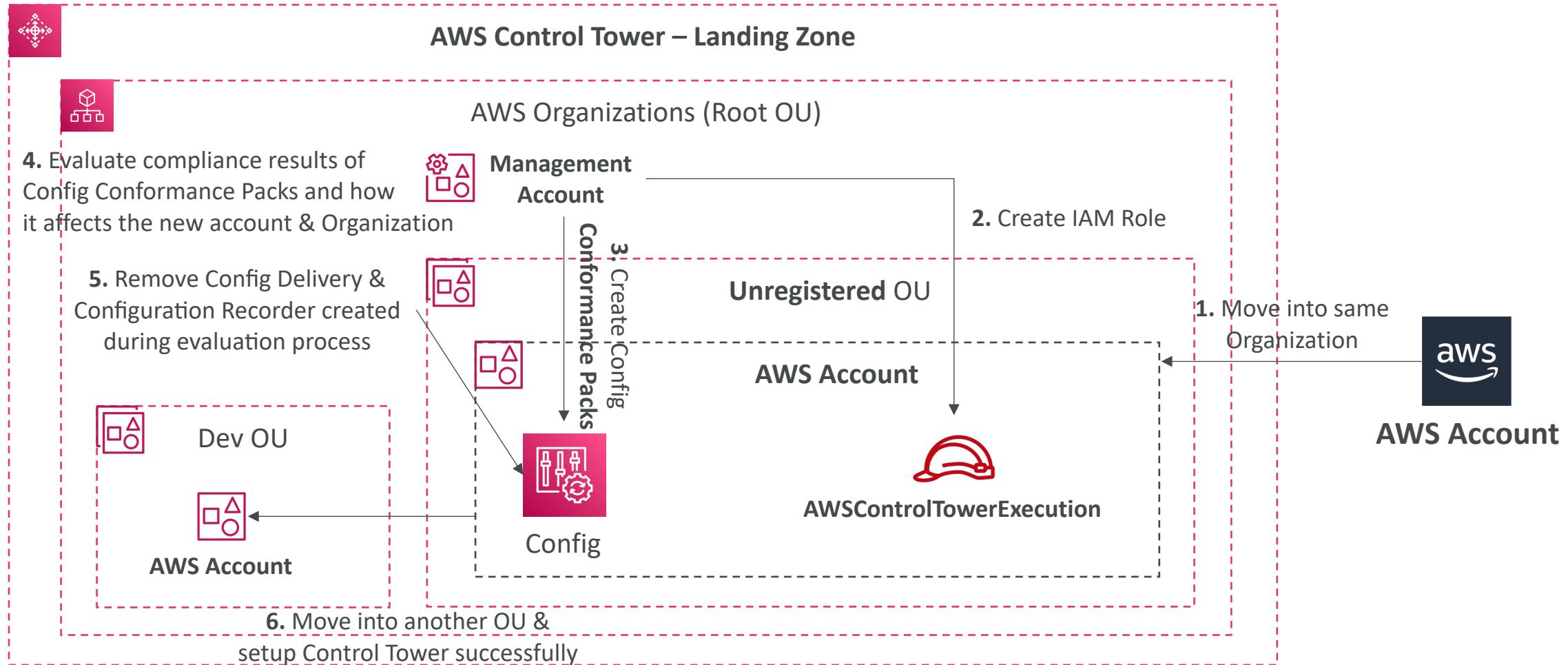


# AWS Control Tower – Account Factory

- You can react to new accounts created using EventBridge

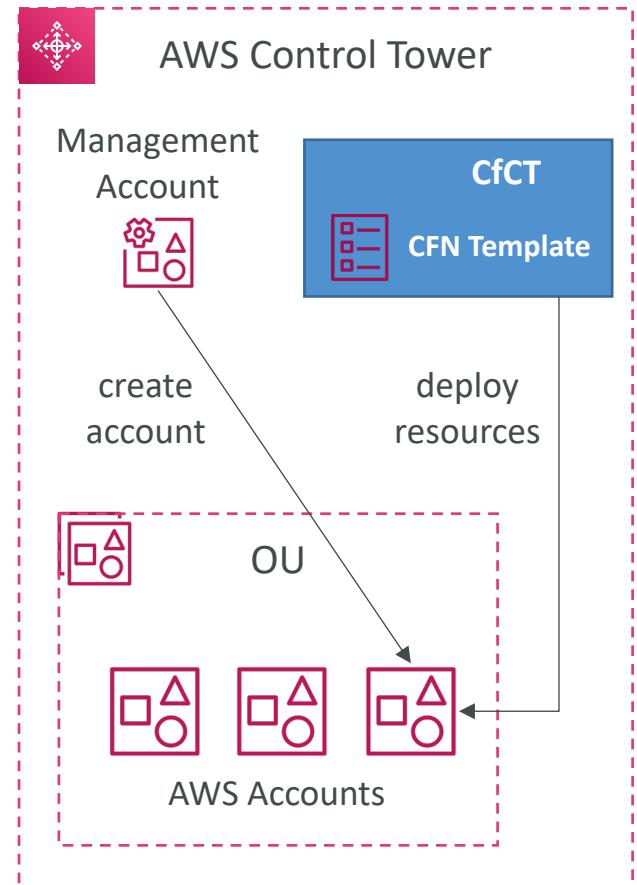


# Migrate an AWS Account to Control Tower

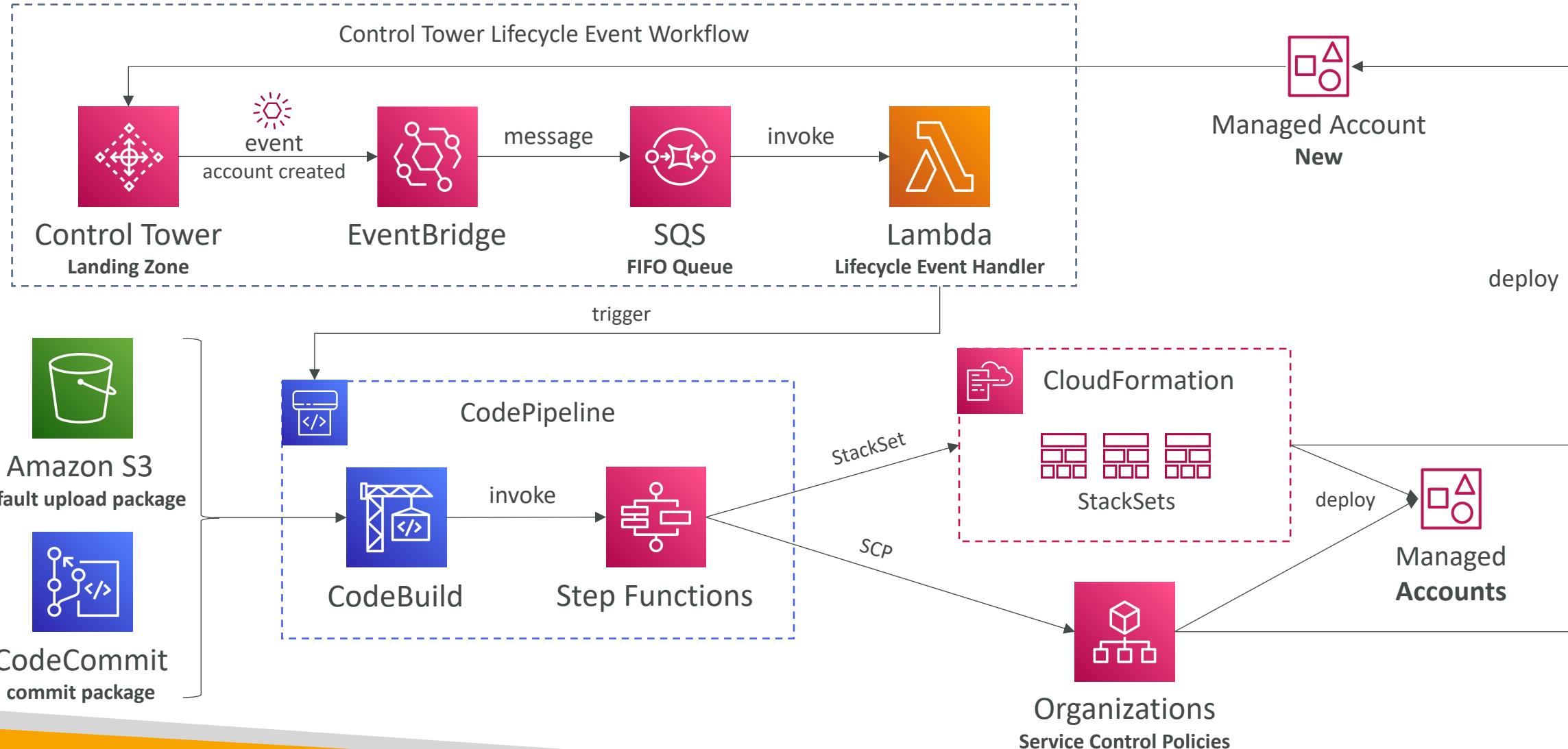


# Customizations for AWS Control Tower (CfCT)

- GitOps-style customization framework created by AWS
- Helps you add customizations to your Landing Zone using your custom CloudFormation templates and SCPs
- Automatically deploy resources to new AWS accounts created using Account Factory
- Note: CfCT is different from AFC (Account Factory Customization ; blueprint)

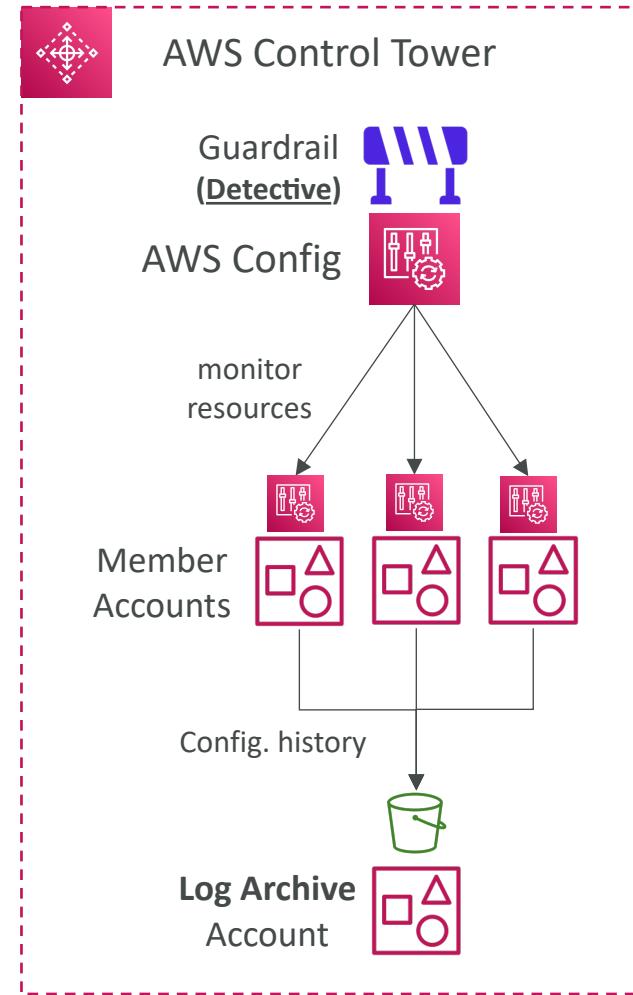


# Customizations for AWS Control Tower (CfCT)



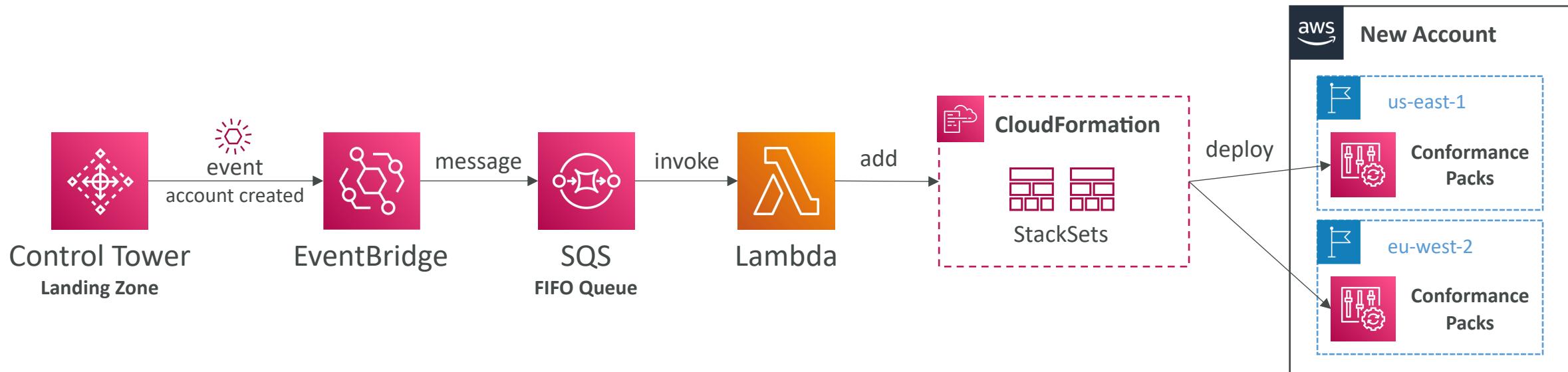
# AWS Control Tower – AWS Config Integration

- Control Tower uses AWS Config to implement **Detective Guardrails**
- Control Tower automatically enables AWS Config in enabled Regions
- AWS Config Configuration History and snapshots delivered to S3 bucket in a centralized Log Archive account
- Control Tower uses CloudFormation StackSets to create resources like Config Aggregator, CloudTrail, and centralized logging



# AWS Control Tower – AWS Config Conformance Packs

- Config Conformance Packs – enables you to create packages of Config Compliance Rules & Remediations for easy deployment at scale
- Deploy to individual AWS accounts or entire and AWS Organization



# AWS Control Tower – Account Factory for Terraform (AFT)

- Helps you provision and customize AWS accounts in Control Tower through Terraform using a deployment pipeline
- You create an *account request Terraform file* which triggers the AFT workflow for account provisioning
- Offers built-in feature options (disabled by default)
  - AWS CloudTrail Data Events – creates a Trail & enables CloudTrail Data Events
  - AWS Enterprise Support Plan – turns on the Enterprise Support Plan
  - Delete The AWS Default VPC – deletes the default VPCs in all AWS Regions
- Terraform module maintained by AWS
- Works with Terraform Open-source, Terraform Enterprise, and Terraform Cloud

# AWS Control Tower – Account Factory for Terraform (AFT)

```

module "aft" {
  source = "git@github.com:aws-ia/terraform-aws-control_
  tower_account_factory.git"

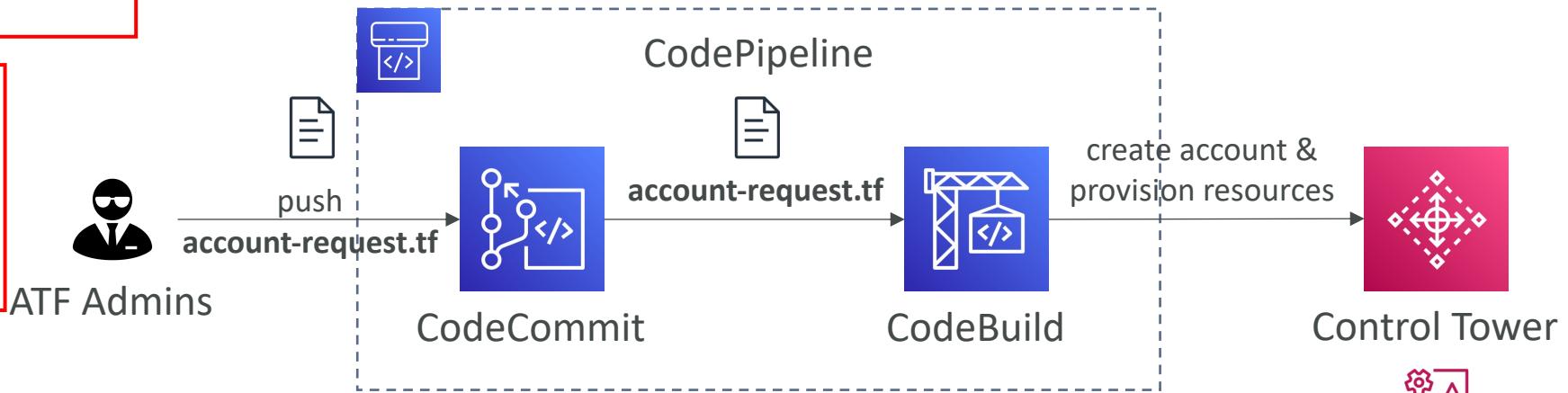
  # Required Parameters
  ct_management_account_id      = "123412341234"
  log_archive_account_id        = "234523452345"
  audit_account_id              = "345634563456"
  aft_management_account_id     = "456745674567"
  ct_home_region                = "us-east-1"
  tf_backend_secondary_region   = "us-west-2"

  # Optional Parameters
  terraform_distribution = "oss"
  vcs_provider           = "codecommit"

  # Optional Feature Flags
  aft_feature_delete_default_vpcs_enabled = false
  aft_feature_cloudtrail_data_events      = false
  aft_feature_enterprise_support          = true
}

account-request.tf

```

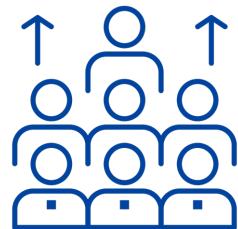


# AWS IAM Identity Center

(successor to AWS Single Sign-On)

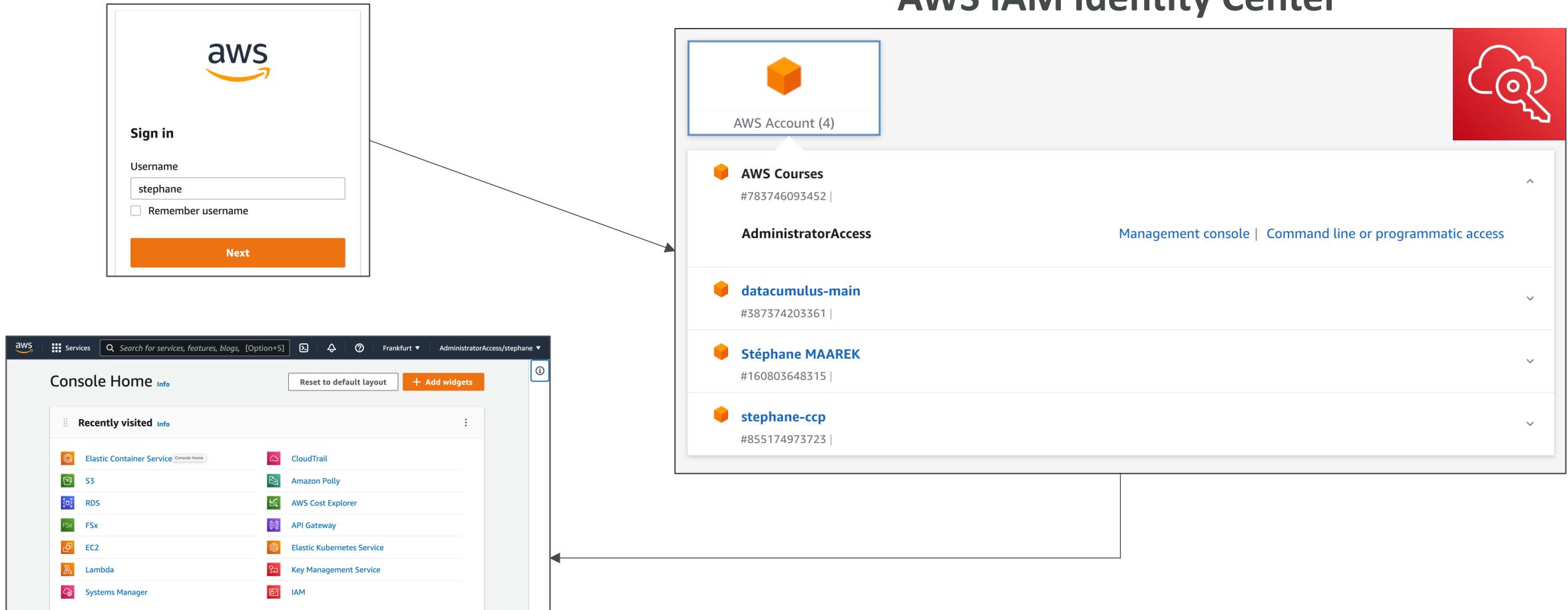


- One login (single sign-on) for all your
  - AWS accounts in AWS Organizations
  - Business cloud applications (e.g., Salesforce, Box, Microsoft 365...)
  - SAML2.0-enabled applications
  - EC2 Windows Instances
- Identity providers
  - Built-in identity store in IAM Identity Center
  - 3<sup>rd</sup> party: Active Directory (AD), OneLogin, Okta...

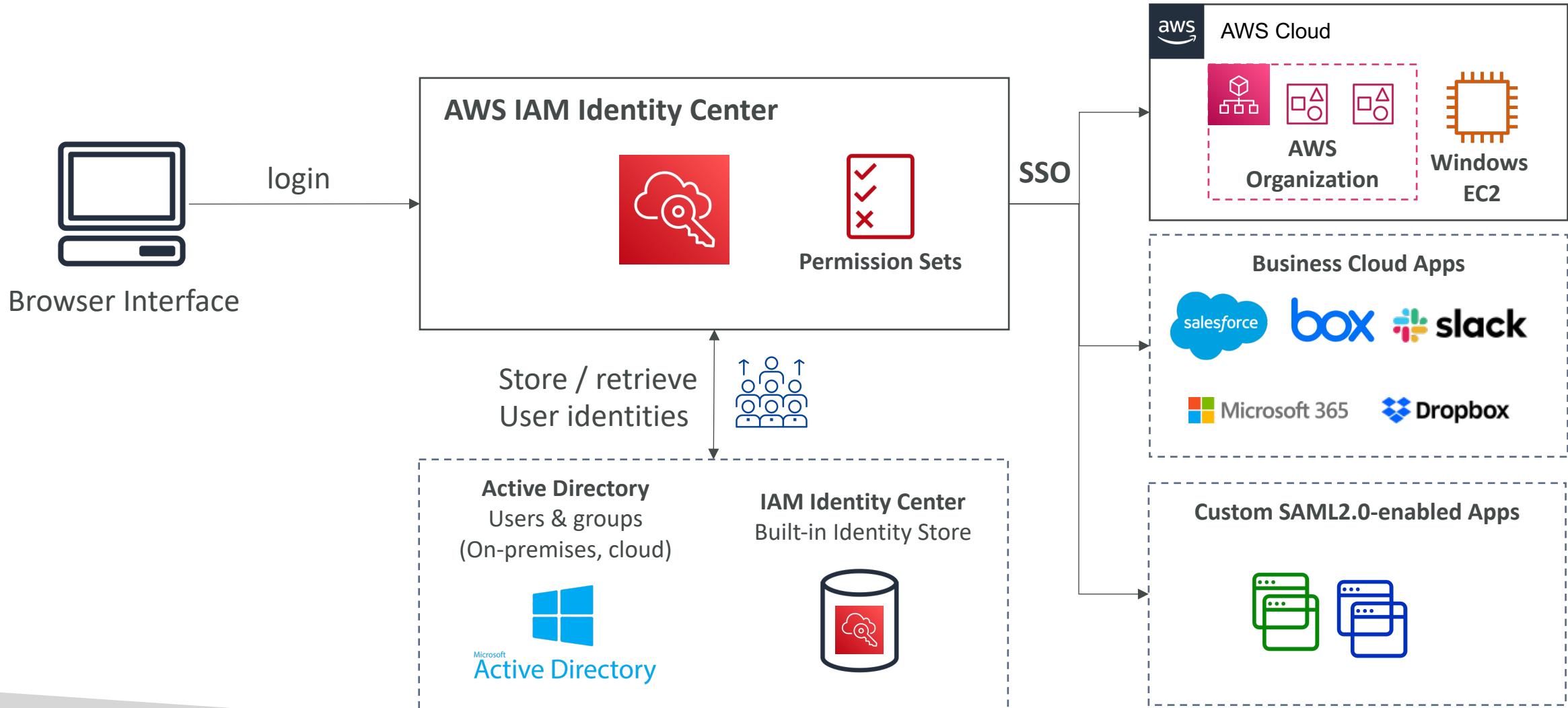


# AWS IAM Identity Center – Login Flow

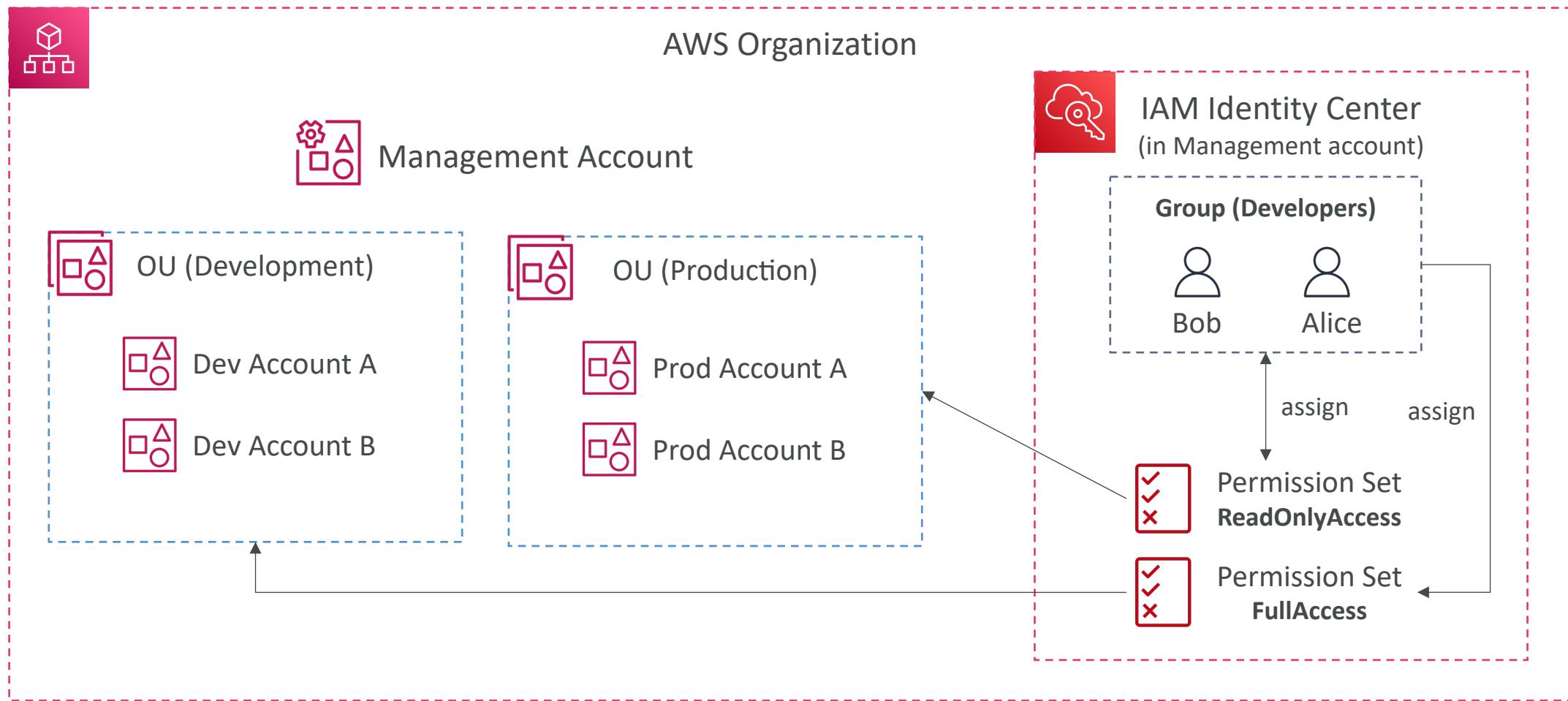
## AWS IAM Identity Center



# AWS IAM Identity Center



# IAM Identity Center

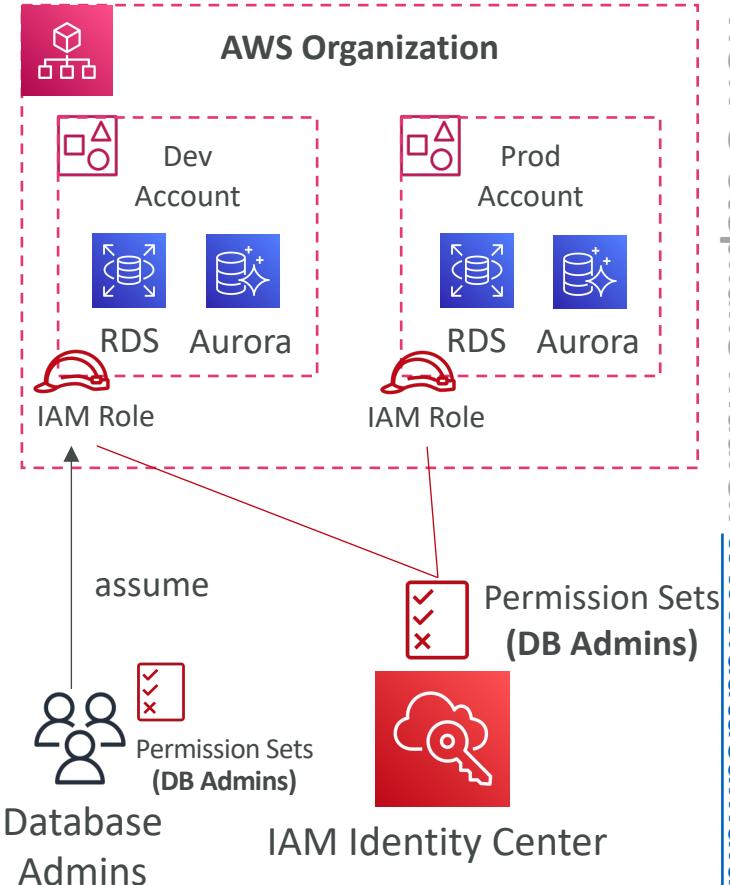


# AWS IAM Identity Center

## Fine-grained Permissions and Assignments

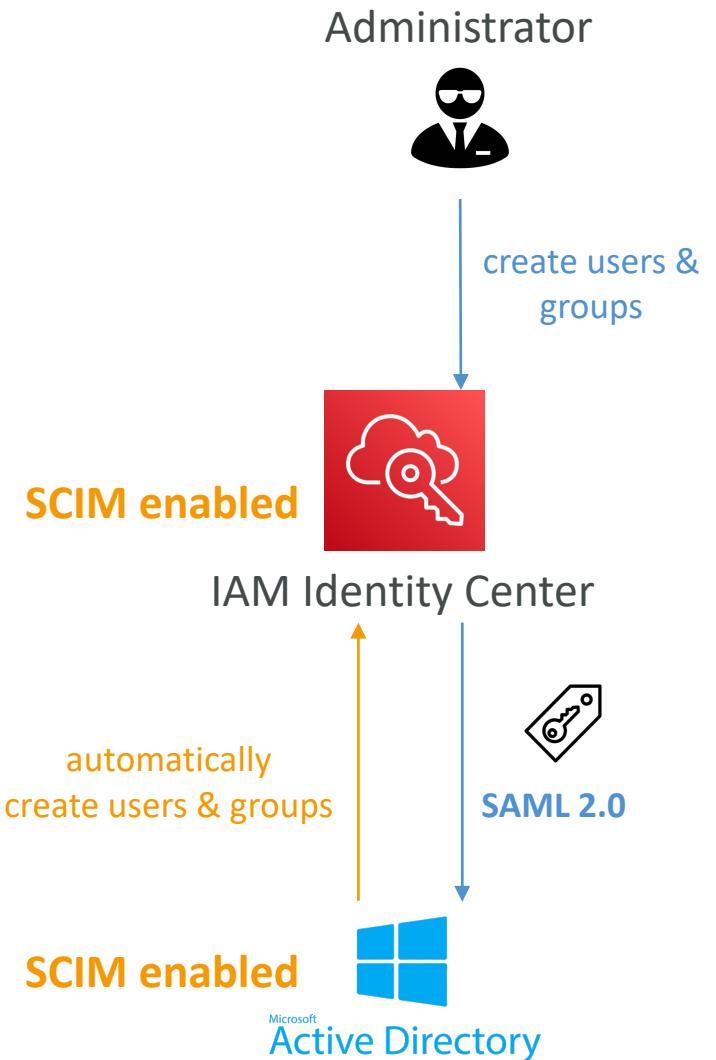


- Multi-Account Permissions
  - Manage access across AWS accounts in your AWS Organization
  - Permission Sets – a collection of one or more IAM Policies assigned to users and groups to define AWS access
- Application Assignments
  - SSO access to many SAML 2.0 business applications (Salesforce, Box, Microsoft 365...)
  - Provide required URLs, certificates, and metadata
- Attribute-Based Access Control (ABAC)
  - Fine-grained permissions based on users' attributes stored in IAM Identity Center Identity Store
  - Example: cost center, title, locale...
  - Use case: Define permissions once, then modify AWS access by changing the attributes



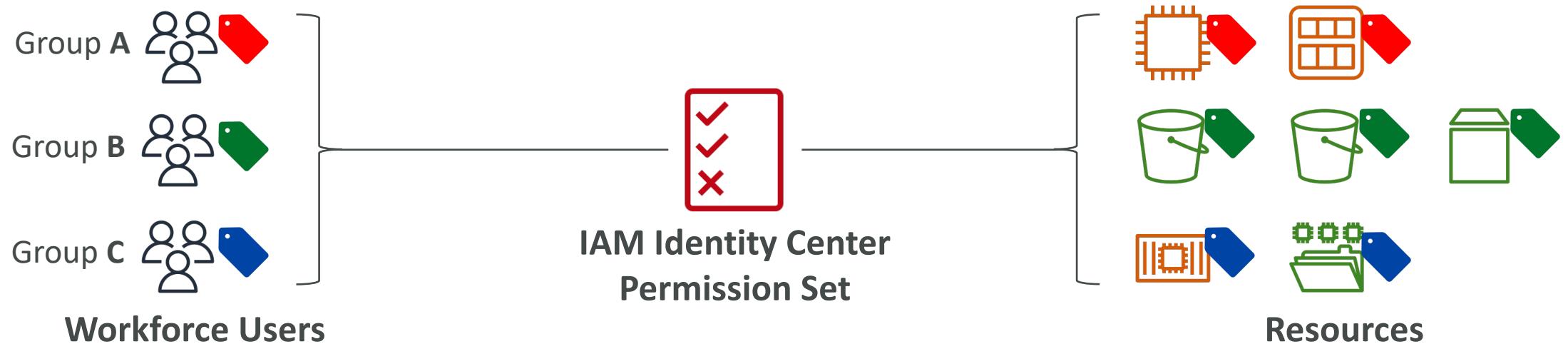
# IAM Identity Center – External IdPs

- **SAML 2.0**: authenticate identities from external IdP
  - Users sign into AWS access portal using their corporate identities
  - Supports Okta, Azure AD, OneLogin...
  - SAML 2.0 does NOT provide a way to query the IdP to learn about users and groups
  - You must create users and groups in the IAM Identity Center that are identical to the users and groups in the External IdP
- **SCIM**: automatic provisioning (synchronization) of user identities from an external IdP into IAM Identity Center
  - SCIM = System for Cross-domain Identity Management
  - Must be supported by the external IdP
  - Perfect complement to using SAML 2.0

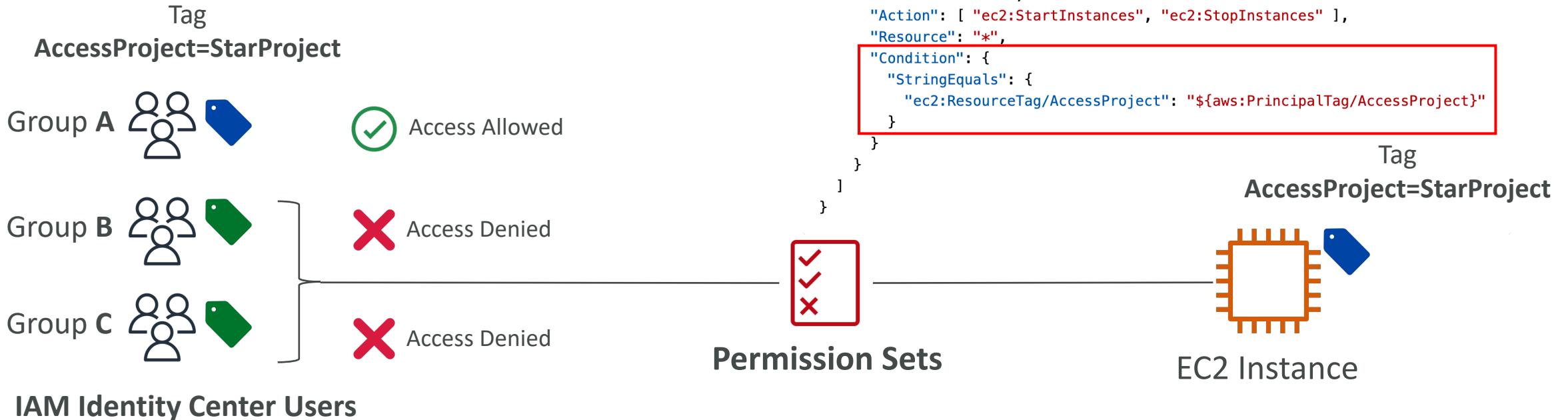


# IAM Identity Center – Attribute-Based Access Control (ABAC)

- Fine-grained permissions based on users' attributes stored in IAM Identity Center Identity Store
- User attributes can be used in IAM Identity Center Permission Sets and Resource-based Policies
- Example: cost center, title, locale...
- Use case: Define permissions once, then modify AWS access by changing the attributes
- User attributes are mapped from the IdP as key-value pairs

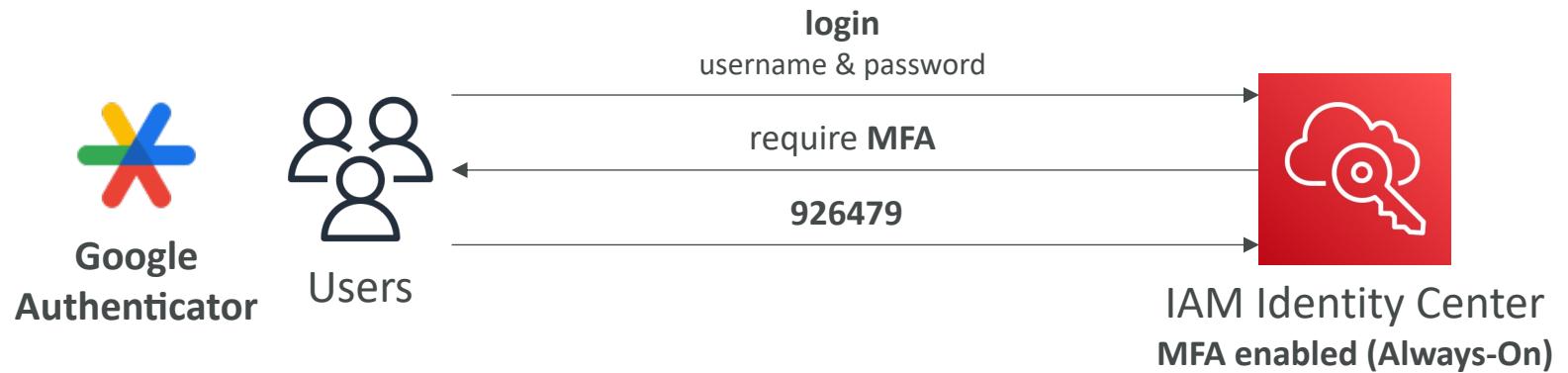


# IAM Identity Center – Attribute-Based Access Control (ABAC)



# IAM Identity Center – Multi-Factor Authentication (MFA)

- Supports Multi-Factor Authentication (MFA) with authentication modes:
  - Every Time They Sign-in (Always-on)
  - Only When Their Sign-in Context Changes (Context-aware) – analyzes user behavior (e.g., device, browser, location...)





# AWS WAF – Web Application Firewall

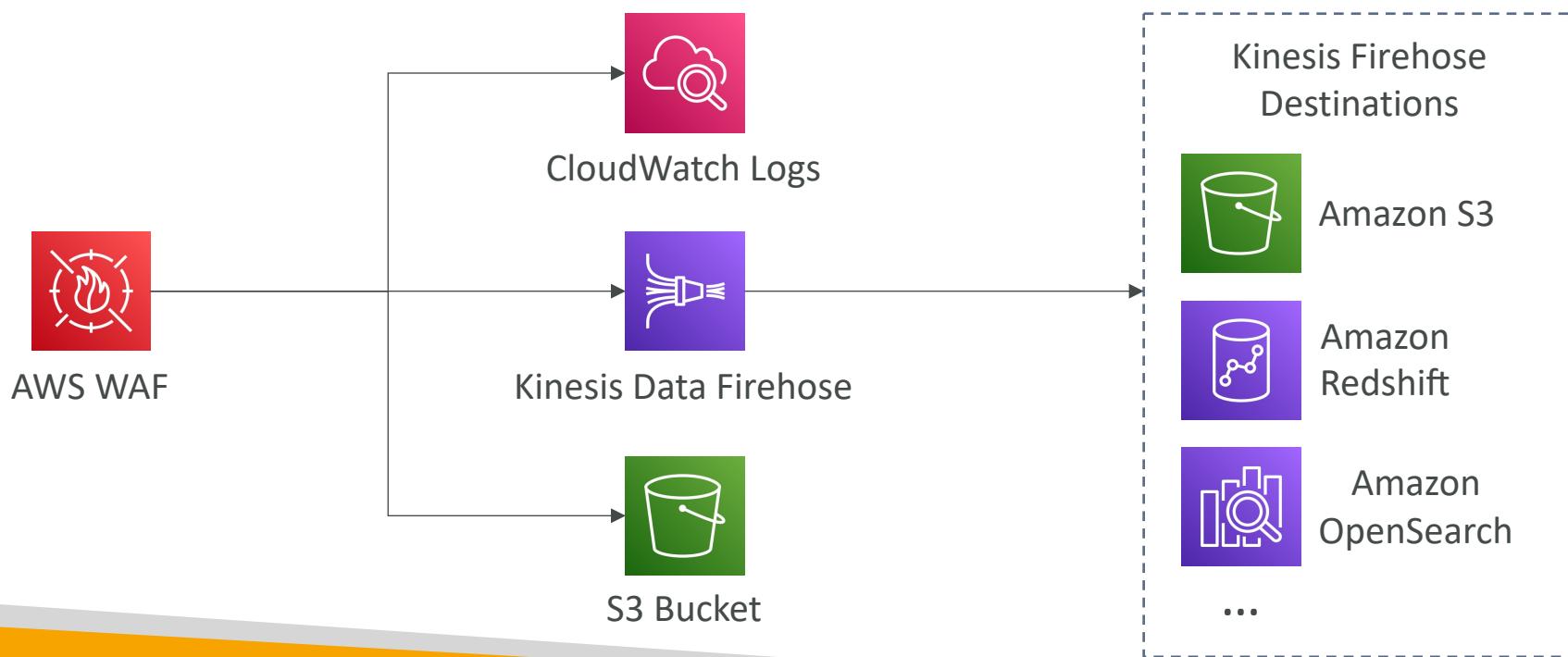
- Protects your web applications from common web exploits (Layer 7)
- Deploy on **Application Load Balancer** (localized rules)
- Deploy on **API Gateway** (rules running at the regional or edge level)
- Deploy on **CloudFront** (rules globally on edge locations)
  - Used to front other solutions: CLB, EC2 instances, custom origins, S3 websites
- Deploy on AppSync (protect your GraphQL APIs)
- WAF is not for DDoS protection
- Define Web ACL (Web Access Control List):
  - Rules can include **IP addresses**, HTTP headers, HTTP body, or URI strings
  - Protects from common attack - **SQL injection** and Cross-Site Scripting (**XSS**)
  - Size constraints, Geo match
  - Rate-based rules (to count occurrences of events)
- Rule Actions: Count | Allow | Block | CAPTCHA

# AWS WAF – Managed Rules

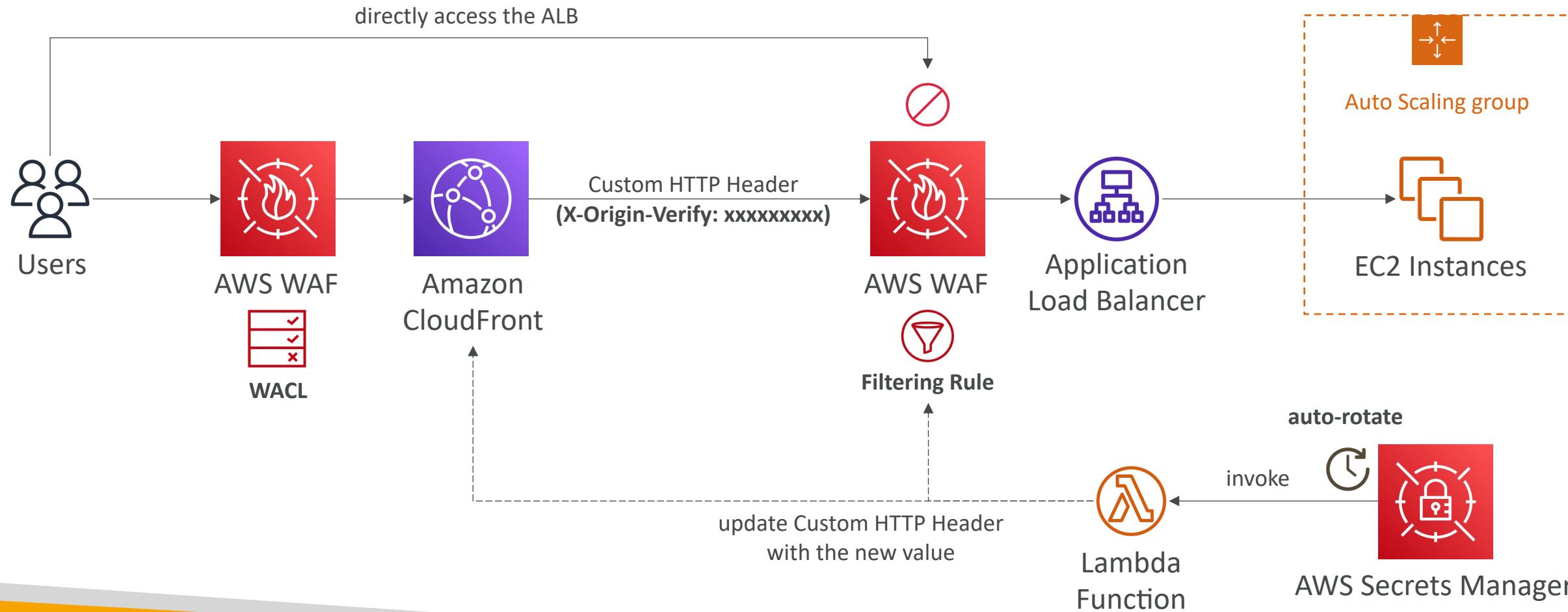
- Library of over 190 managed rules
- Ready-to-use rules that are managed by AWS and AWS Marketplace Sellers
- **Baseline Rule Groups** – general protection from common threats
  - AWSManagedRulesCommonRuleSet, AWSManagedRulesAdminProtectionRuleSet...
- **Use-case Specific Rule Groups** – protection for many AWS WAF use cases
  - AWSManagedRulesSQLiRuleSet, AWSManagedRulesWindowsRuleSet, AWSManagedRulesPHPRuleSet, AWSManagedRulesWordPressRuleSet...
- **IP Reputation Rule Groups** – block requests based on source (e.g., malicious IPs)
  - AWSManagedRulesAmazonIpReputationList, AWSManagedRulesAnonymousIpList
- **Bot Control Managed Rule Group** – block and manage requests from bots
  - AWSManagedRulesBotControlRuleSet

# WAF - Web ACL – Logging

- You can send your logs to an:
  - Amazon CloudWatch Logs log group – 5 MB per second
  - Amazon Simple Storage Service (Amazon S3) bucket – 5 minutes interval
  - Amazon Kinesis Data Firehose – limited by Firehose quotas



# Solution Architecture – Enhance CloudFront Origin Security with AWS WAF & AWS Secrets Manager



# AWS Firewall Manager



- Manage rules in all accounts of an AWS Organization
- Security policy: common set of security rules
  - WAF rules (Application Load Balancer, API Gateways, CloudFront)
  - AWS Shield Advanced (ALB, CLB, NLB, Elastic IP, CloudFront)
  - Security Groups for EC2, Application Load Balancer and ENI resources in VPC
  - AWS Network Firewall (VPC Level)
  - Amazon Route 53 Resolver DNS Firewall
  - Policies are created at the region level
- Rules are applied to new resources as they are created (good for compliance) across all and future accounts in your Organization

# WAF vs. Firewall Manager vs. Shield



AWS WAF



AWS Firewall Manager



AWS Shield

- WAF, Shield and Firewall Manager are used together for comprehensive protection
- Define your Web ACL rules in WAF
- For granular protection of your resources, WAF alone is the correct choice
- If you want to use AWS WAF across accounts, accelerate WAF configuration, automate the protection of new resources, use Firewall Manager with AWS WAF
- Shield Advanced adds additional features on top of AWS WAF, such as dedicated support from the Shield Response Team (SRT) and advanced reporting.
- If you're prone to frequent DDoS attacks, consider purchasing Shield Advanced

# AWS Firewall Manager – Security Policies

- Policy Type: AWS WAF
  - Enforce applying WebACLs to all ALBs in all accounts in AWS Organization
  - Identify resources that don't comply, but don't auto remediate: create the WebACL in each account without applying the WebACL to any resources
  - Auto remediate any noncompliant resources – automatically apply the WebACLs to existing resources
- Policy Type: Shield Advanced
  - Enforce Shield Advanced protections in all accounts in AWS Organization
  - Option to view only compliance (to assess impact) or auto remediate

# AWS Firewall Manager – Security Policies

- **Security Group Policy Type: Common Security Groups**
  - Enforce applying SGs to all EC2 instances in all accounts in AWS Organization
- **Security Group Policy Type: Auditing of Security Group Policy**
  - Check and manage SGs Rules in all accounts in AWS Organization
- **Security Group Policy Type: Usage Audit Security Group Policy**
  - Monitor unused and redundant SGs and optionally perform cleanup

# AWS Firewall Manager – Security Policies

- **Policy Type: Network Firewall**

- Centrally manage Network Firewall firewalls in all accounts in AWS Organization
- Distributed – creates and maintains firewall endpoint in each VPC
- Centralized – creates and maintains firewall endpoint in a centralized VPC
- Import Existing Firewalls – import existing firewalls using Resource Sets

- **Policy Type: Route 53 Resolver DNS Firewall**

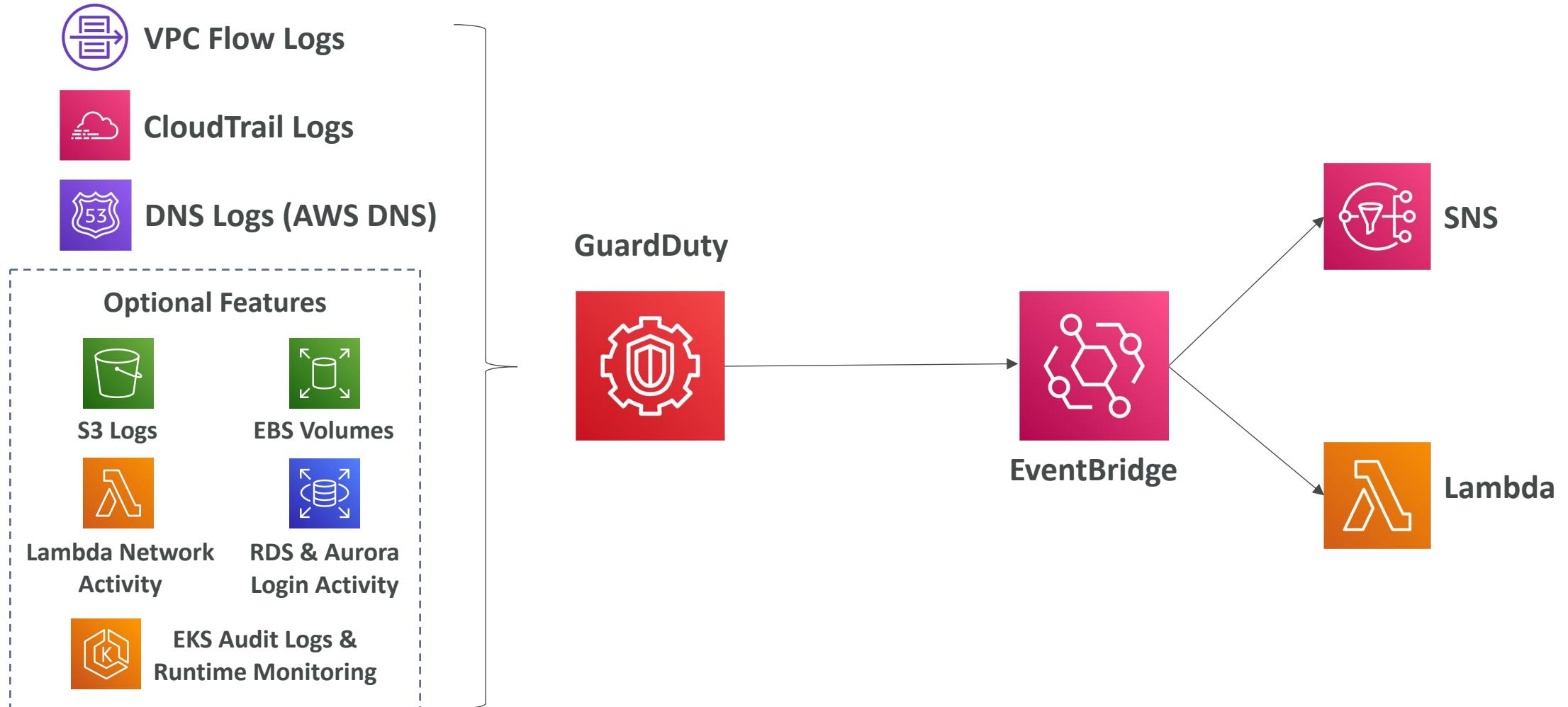
- Manage associations between Resolver DNS Firewall Rule Groups and VPCs in all accounts in AWS Organization



# Amazon GuardDuty

- Intelligent Threat discovery to protect your AWS Account
- Uses Machine Learning algorithms, anomaly detection, 3<sup>rd</sup> party data
- One click to enable (30 days trial), no need to install software
- Input data includes:
  - CloudTrail Events Logs – unusual API calls, unauthorized deployments
    - CloudTrail Management Events – create VPC subnet, create trail, ...
    - CloudTrail S3 Data Events – get object, list objects, delete object, ...
  - VPC Flow Logs – unusual internal traffic, unusual IP address
  - DNS Logs – compromised EC2 instances sending encoded data within DNS queries
  - Optional Feature – EKS Audit Logs, RDS & Aurora, EBS, Lambda, S3 Data Events...
- Can setup **EventBridge rules** to be notified in case of findings
- EventBridge rules can target AWS Lambda or SNS
- **Can protect against CryptoCurrency attacks** (has a dedicated “finding” for it)

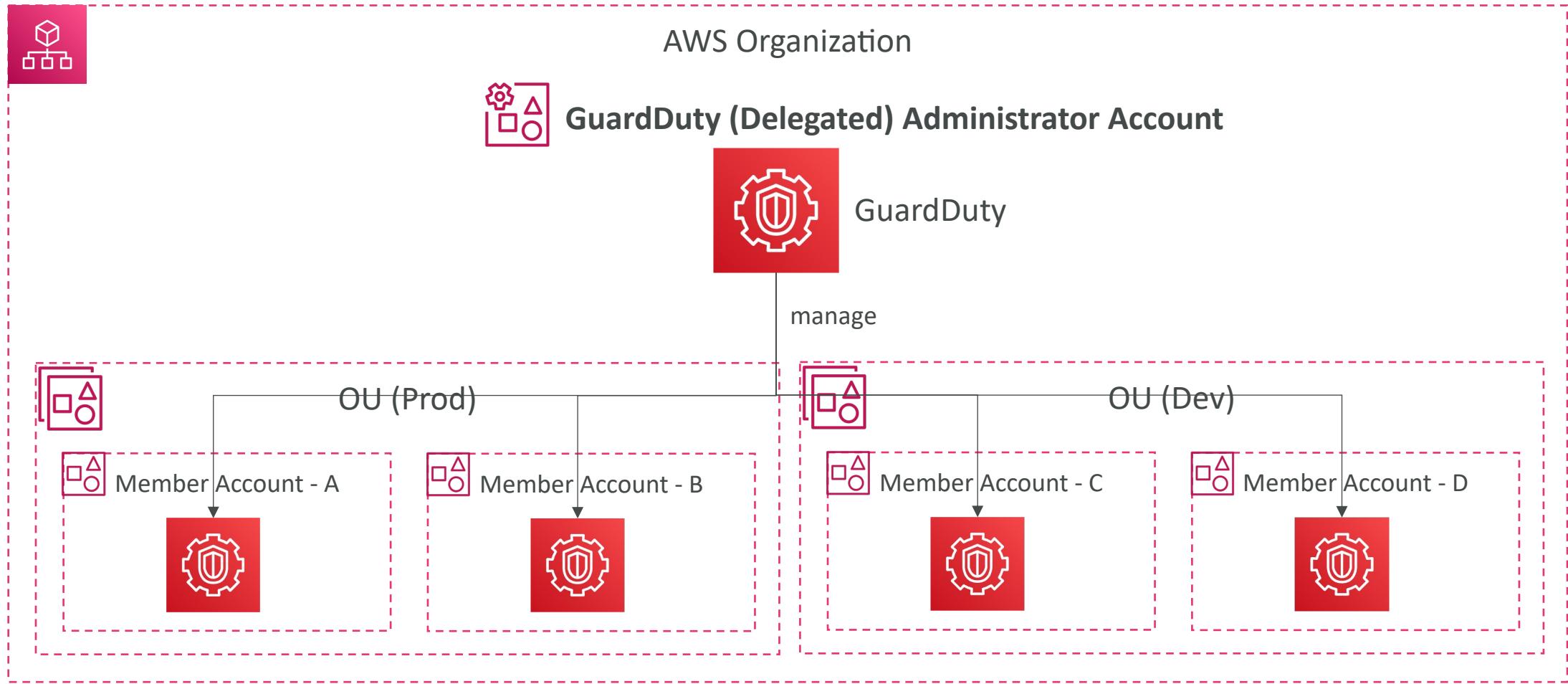
# Amazon GuardDuty



# Amazon GuardDuty – Multi-Account Strategy

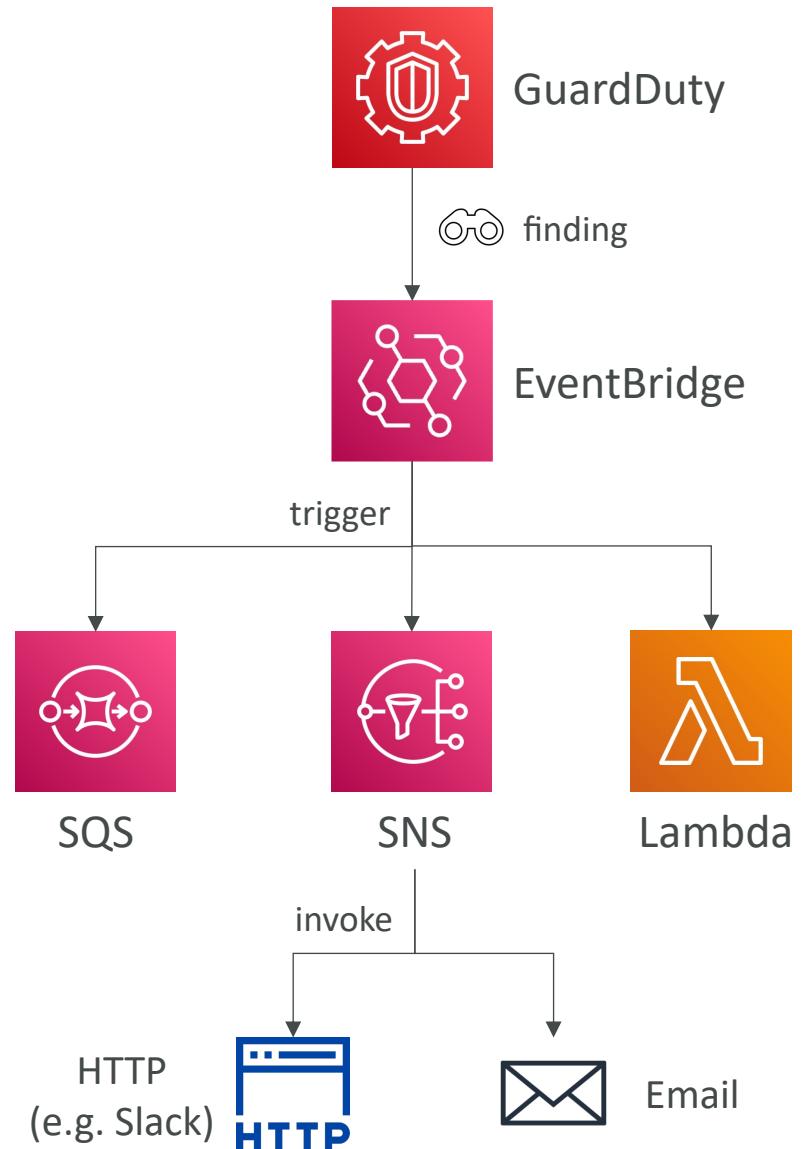
- You can manage multiple accounts in GuardDuty
- Associate the Member accounts with the Administrator account
  - Through an AWS Organization
  - Sending invitation through GuardDuty
- Administrator account can:
  - Add and remove member accounts
  - Manage GuardDuty within the associated member accounts
  - Manage findings, suppression rules, trusted IP lists, threat lists
- In an AWS Organization, you can specify a member account as the Organization's **delegated administrator for GuardDuty**

# Amazon GuardDuty – Multi-Account Strategy



# Amazon GuardDuty – Findings Automated Response

- Findings are potential security issue for malicious events happening in your AWS account
- Automate response to security issues revealed by GuardDuty Findings using EventBridge
- Send alerts to SNS (email, Lambda, Slack, Chime...)
- Events are published to both the administrator account and the member account that it is originated from



# Amazon GuardDuty – Findings

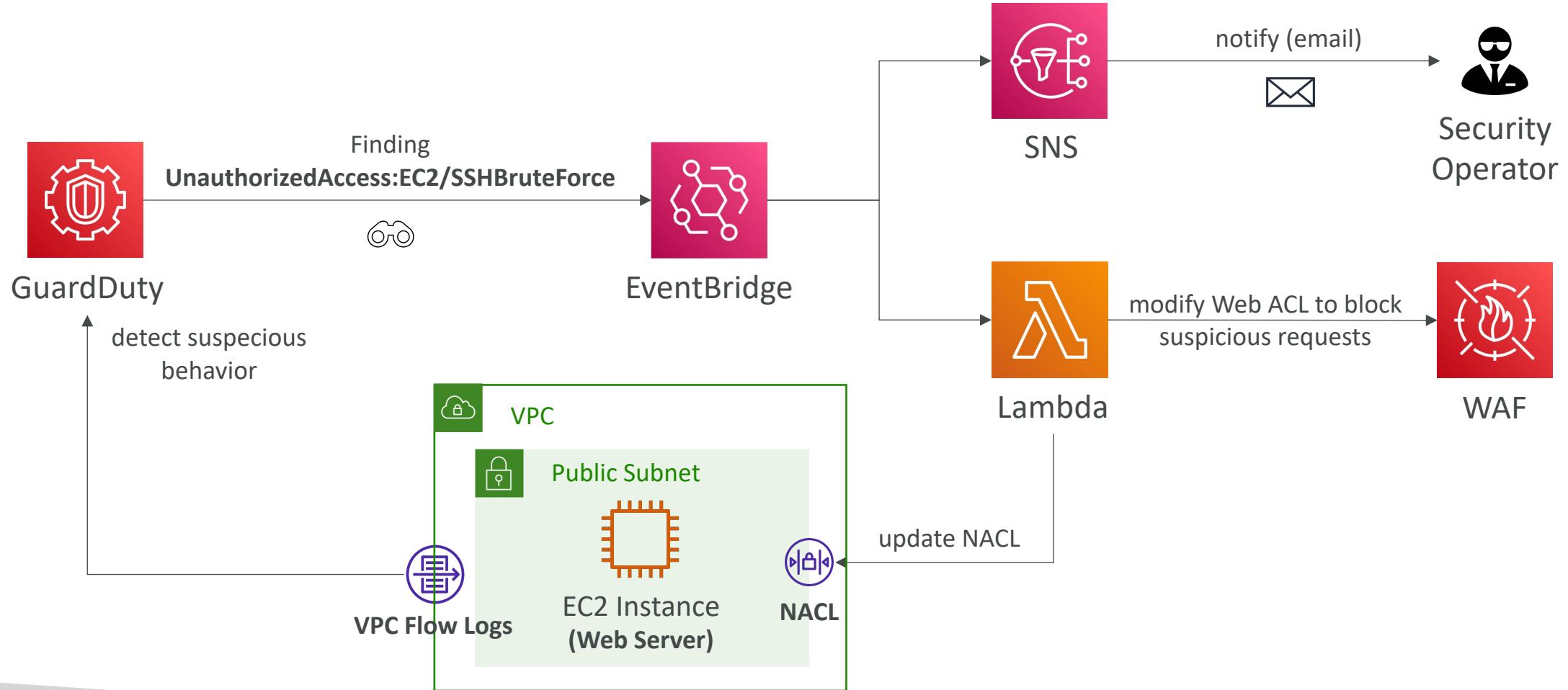


- GuardDuty pulls independent streams of data directly from CloudTrail logs (management events, data events), VPC Flow Logs or EKS logs
- Each finding has a severity value between 0.1 to 8+ (High, Medium, Low)
- Finding naming convention  
*ThreatPurpose:ResourceTypeAffected/ThreatFamilyName.DetectionMechanism!Artifact*
  - ThreatPurpose – primary purpose of the threat (e.g., Backdoor, CryptoCurrency)
  - ResourceTypeAffected – which AWS resource is the target (e.g., EC2, S3)
  - ThreatFamilyName – describes the potential malicious activity (e.g., NetworkPortUnusual)
  - DetectionMechanism – method GuardDuty detecting the finding (e.g., Tcp, Udp)
  - Artifact – describes a resource that is used in the malicious activity (e.g., DNS)
- You can generate sample findings in GuardDuty to test your automations

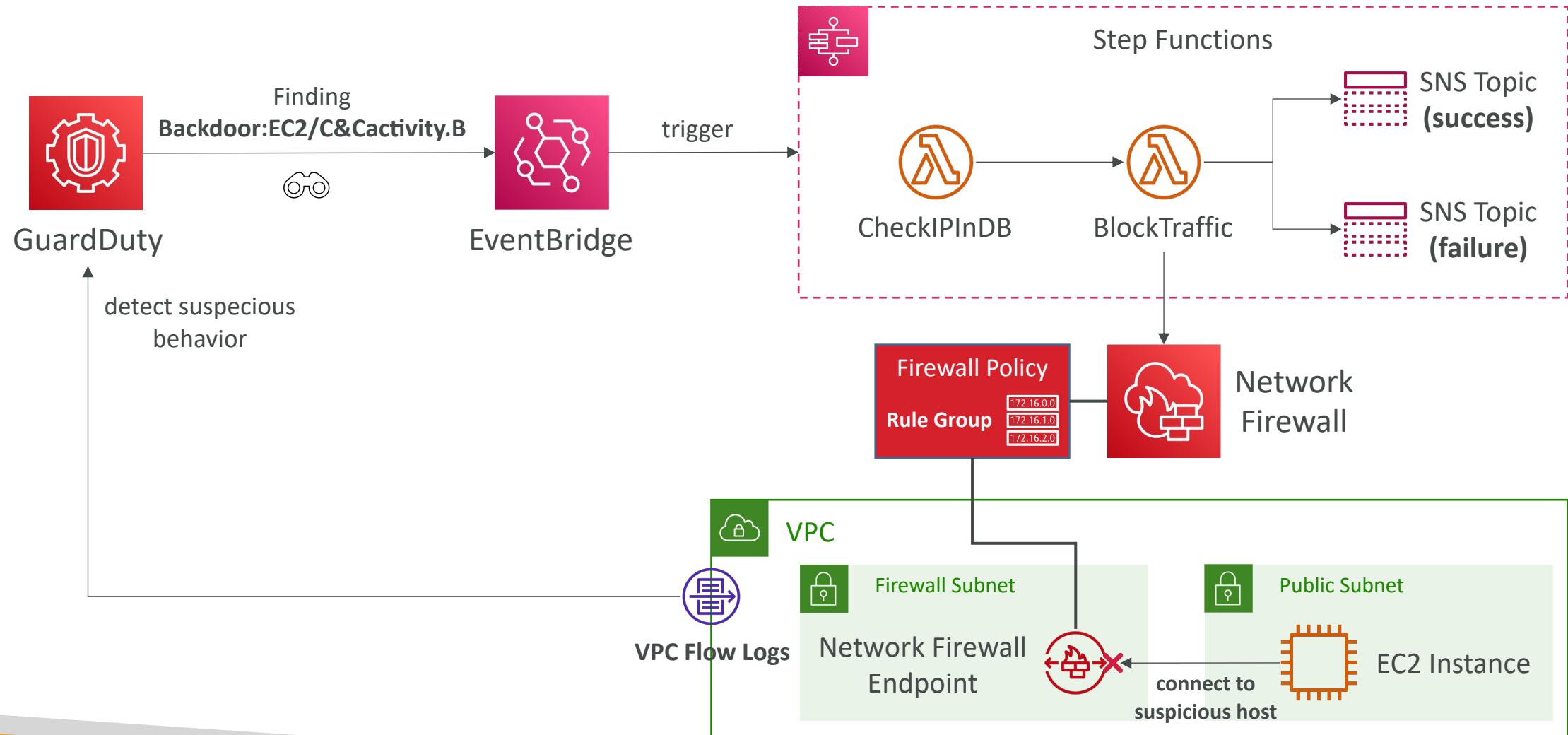
# Amazon GuardDuty – Findings Types

- EC2 Finding Types
  - UnauthorizedAccess:EC2/SSHBruteForce, CryptoCurrency:EC2/BitcoinToo.B!DNS
- IAM Finding Types
  - Stealth:IAMUser/CloudTrailLoggingDisabled, Policy:IAMUser/RootCredentialUsage
- Kubernetes Audit Logs Finding Types
  - CredentialAccess:Kubernetes/MaliciousIPCaller
- Malware Protection Finding Types
  - Execution:EC2/SuspiciousFile, Execution:ECS/ SuspiciousFile
- RDS Protection Finding Types
  - CredentialAccess:RDS/AnomalousBehavior:SuccessfulLogin
- S3 Finding Types
  - Policy:S3/AccountBlockPublicAccessDisabled, PenTest:S3/KaliLinux

# Amazon GuardDuty – Architectures

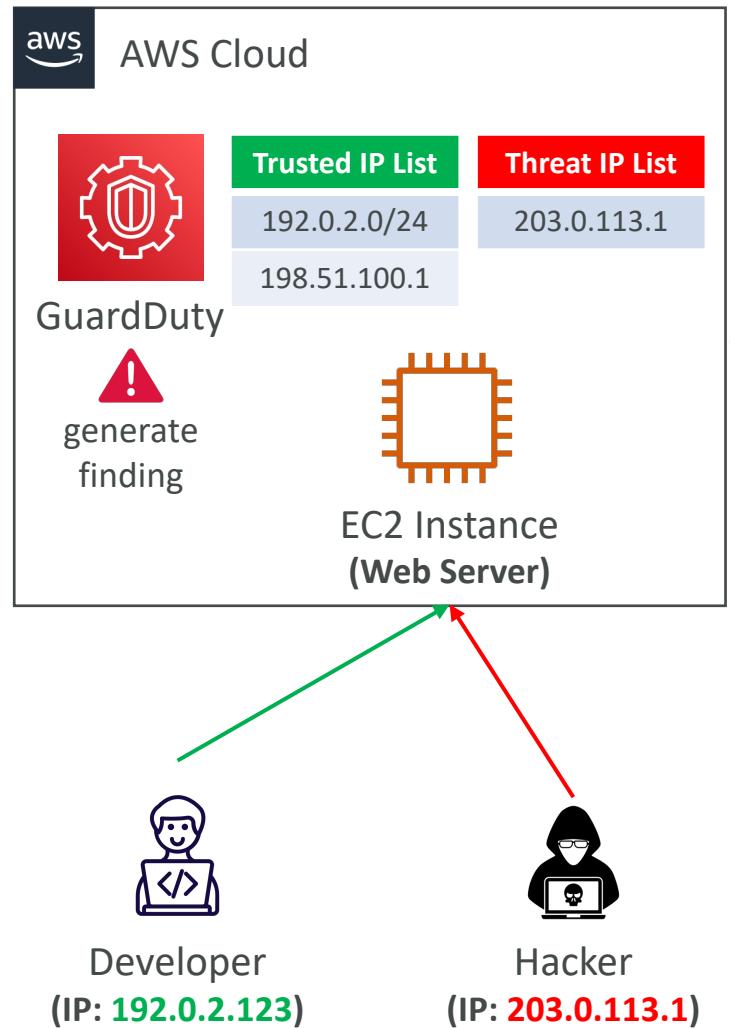


# Amazon GuardDuty – Architectures



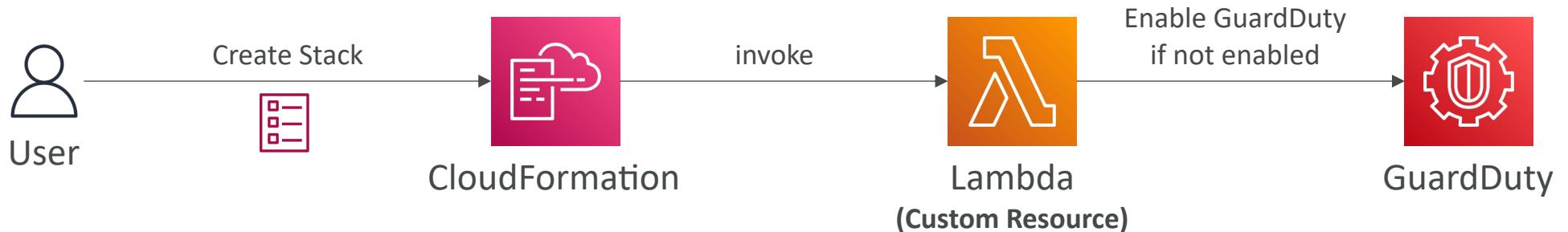
# Amazon GuardDuty – Trusted and Threat IP Lists

- Works only for public IP addresses
- **Trusted IP List**
  - List of IP addresses and CIDR ranges that you trust
  - GuardDuty doesn't generate findings for these trusted lists
- **Threat IP List**
  - List of known malicious IP addresses and CIDR ranges
  - GuardDuty generates findings based on these threat lists
  - Can be supplied by 3rd party threat intelligence or created custom for you
- In a multi-account GuardDuty setup, only the GuardDuty administrator account can manage those lists



# Amazon GuardDuty – CloudFormation

- You can enable GuardDuty using a CloudFormation template
- If GuardDuty is already enabled, the CloudFormation Stack deployment fails
- Use CloudFormation Custom Resource (Lambda) to conditionally enable GuardDuty if it is not already enabled
- Deploy across all the Organization using a Stack Set



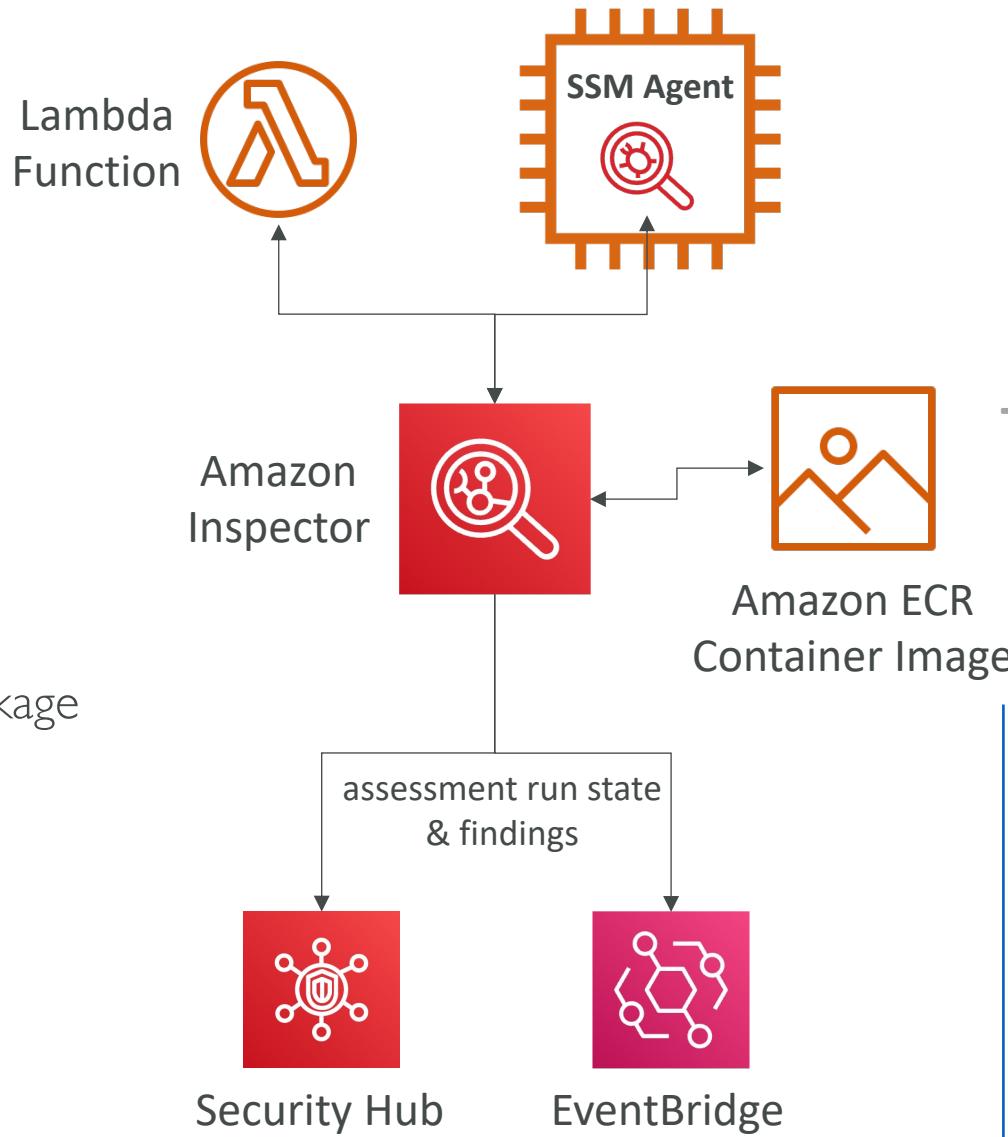
# Amazon Detective



- GuardDuty, Macie, and Security Hub are used to identify potential security issues, or findings
- Sometimes security findings require deeper analysis to isolate the root cause and take action – it's a complex process
- Amazon Detective analyzes, investigates, and quickly identifies the root cause of security issues or suspicious activities using ML and graphs
- Automatically collects and processes events from VPC Flow Logs, CloudTrail, and GuardDuty to create a unified view
- Produces visualizations with details and context to get to the root cause

# Amazon Inspector

- Automated Security Assessments
- For EC2 instances
  - Leveraging the AWS System Manager (SSM) agent
  - Analyze against unintended network accessibility
  - Analyze the running OS against known vulnerabilities
- For Container Images push to Amazon ECR
  - Assessment of Container Images as they are pushed
- For Lambda Functions
  - Identifies software vulnerabilities in function code and package dependencies
  - Assessment of functions as they are deployed
- Reporting & integration with AWS Security Hub
- Send findings to Amazon Event Bridge



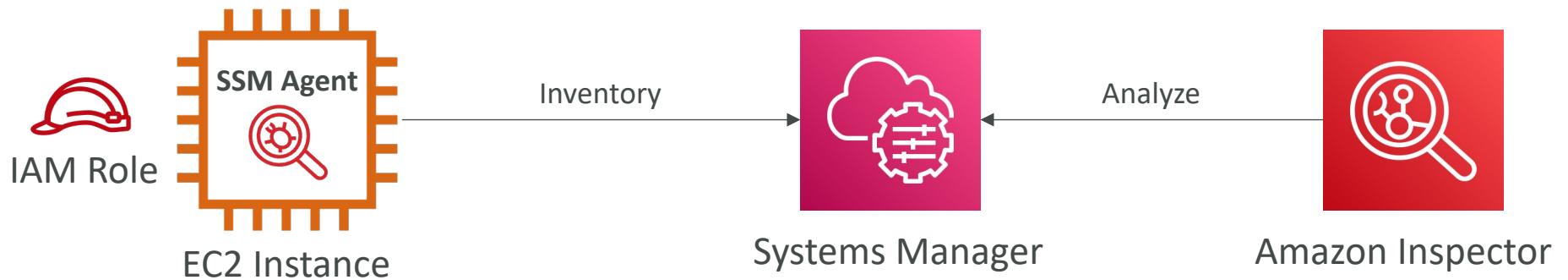
# What does Amazon Inspector evaluate?



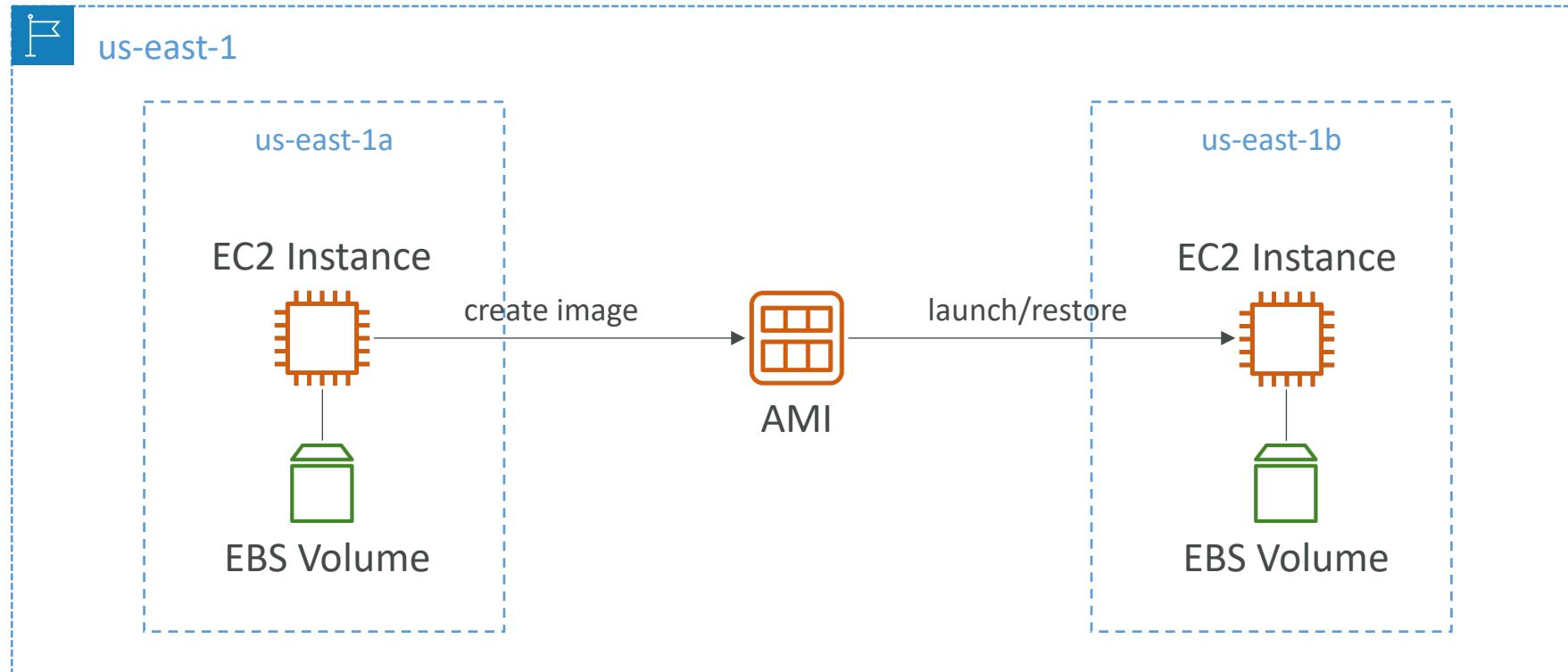
- Remember: only for EC2 instances, Container Images & Lambda functions
- Continuous scanning of the infrastructure, only when needed
- Package vulnerabilities (EC2, ECR & Lambda) – database of CVE
- Network reachability (EC2)
- A risk score is associated with all vulnerabilities for prioritization

# Amazon Inspector – Systems Manager

- EC2 instance setup in Inspector requires:
  - SSM Agent is running
  - Be an SSM Managed Instance (IAM Role or Default Host Management Config.)
  - Outbound 443 to Systems Manager endpoint

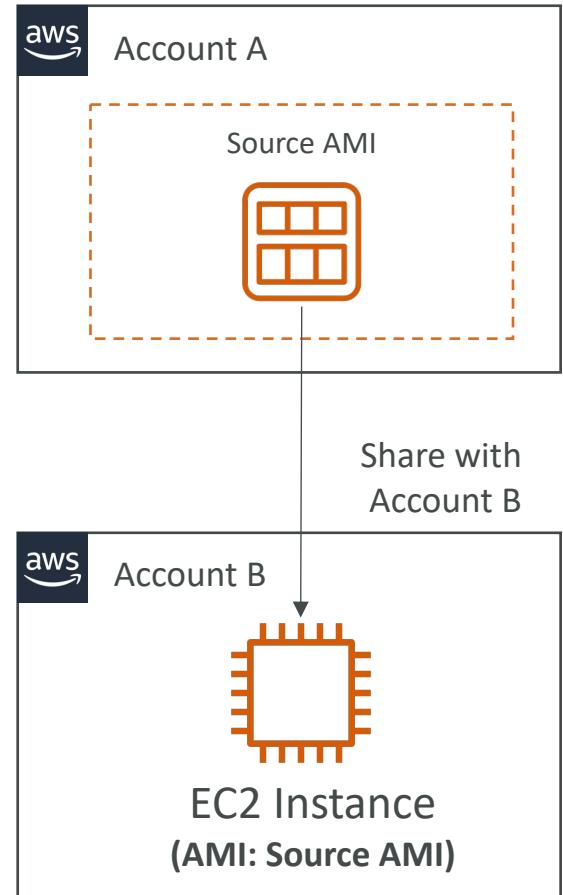


# EC2 Instance Migration between AZ

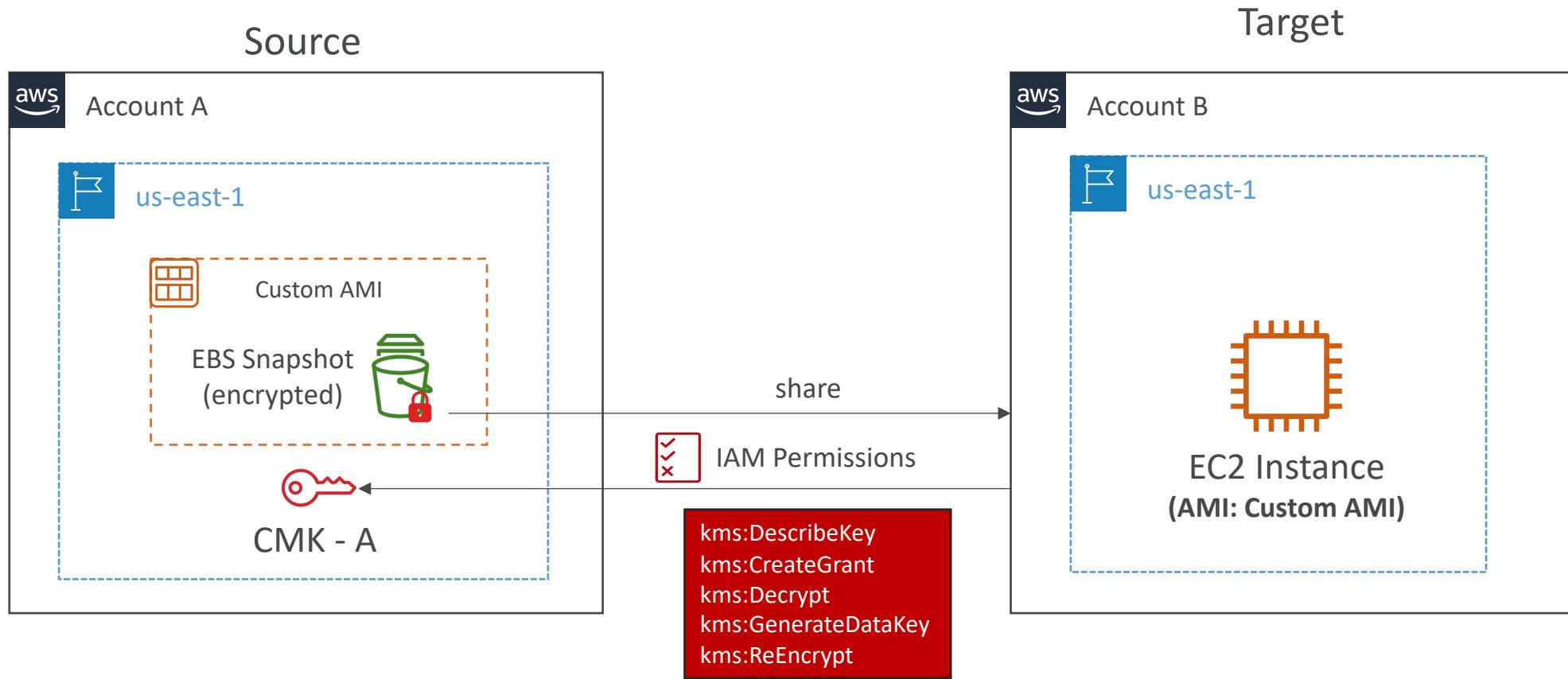


# Cross-Account AMI Sharing

- You can share an AMI with another AWS account
- Sharing an AMI does not affect the ownership of the AMI
- You can only share AMIs that have unencrypted volumes and volumes that are encrypted with a customer managed key
- If you share an AMI with encrypted volumes, you must also share any customer managed keys used to encrypt them.

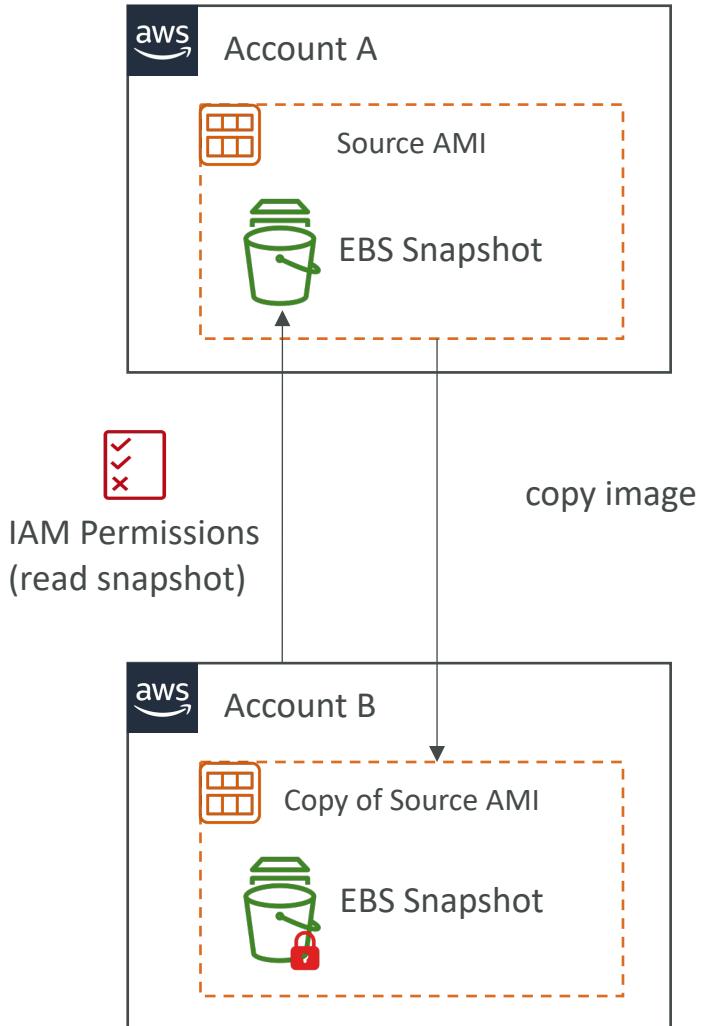


# AMI Sharing with KMS Encryption



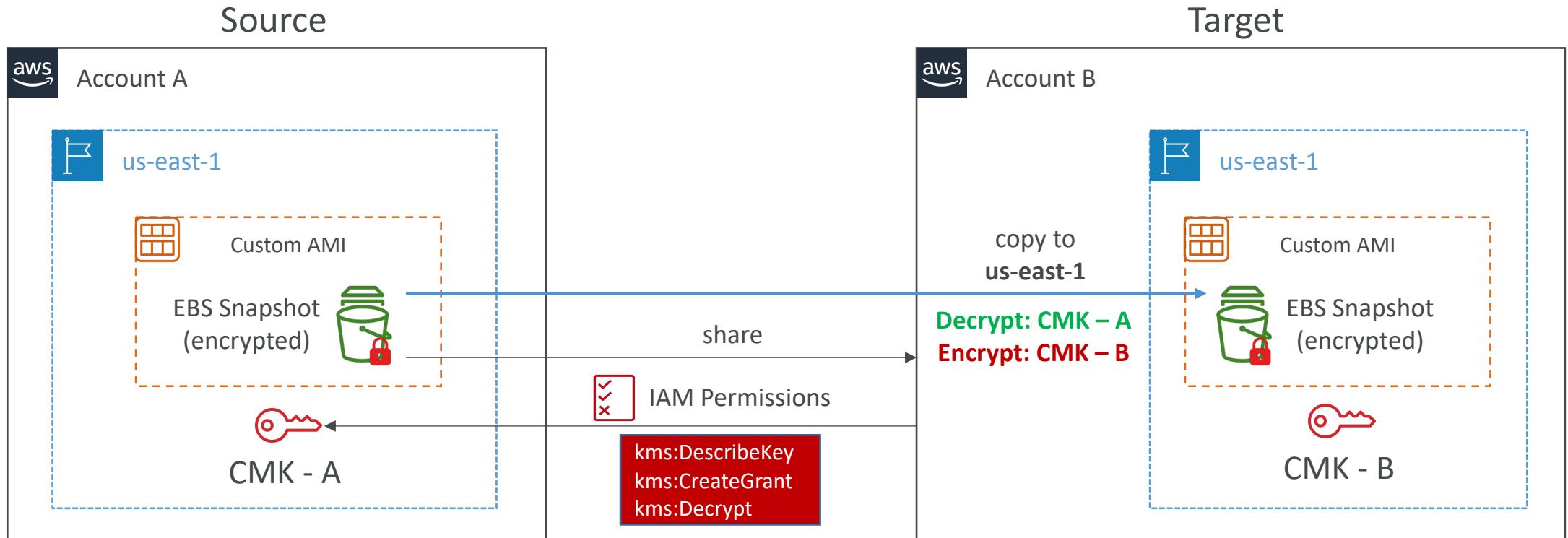
# Cross-Account AMI Copy

- If you copy an AMI that has been shared with your account, you are the owner of the target AMI in your account
- The owner of the source AMI must grant you read permissions for the storage that backs the AMI (EBS Snapshot)
- If the shared AMI has encrypted snapshots, the owner must share the key or keys with you as well
- Can encrypt the AMI with your own CMK while copying



# AMI Copy with KMS Encryption

## Cross-Region / Cross-Account Encrypted AMI Copy





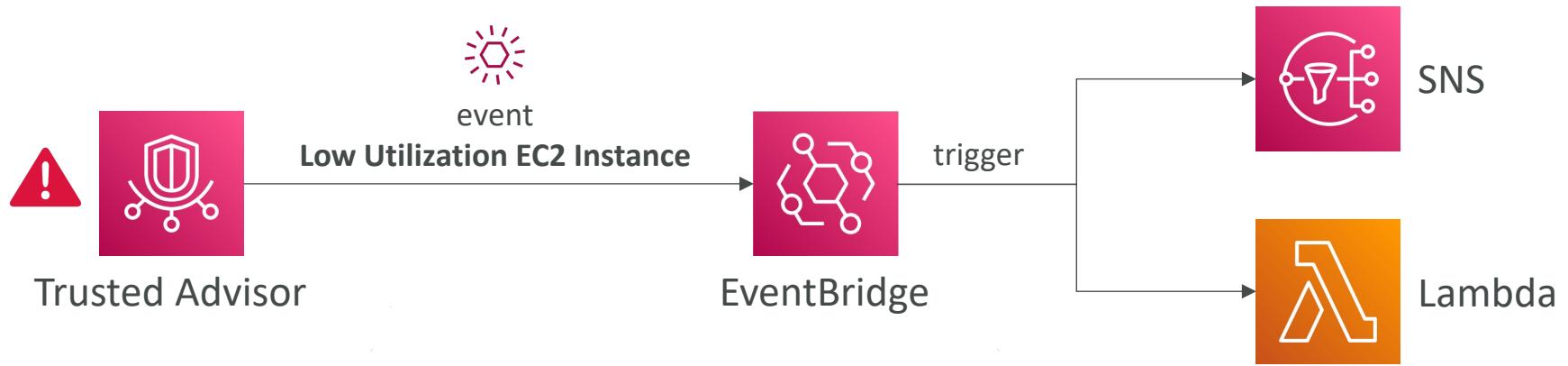
# Trusted Advisor

- No need to install anything – high level AWS account assessment
- Analyze your AWS accounts and provides recommendation on 6 categories:
  - Cost optimization
  - Performance
  - Security
  - Fault tolerance
  - Service limits
  - Operational Excellence
- **Business & Enterprise Support plan**
  - Full Set of Checks
  - Programmatic Access using AWS Support API

## Checks

- ▶ **Amazon EBS Public Snapshots**  
Checks the permission settings for your Amazon Elastic  
0 EBS snapshots are marked as public.
- ▶ **Amazon RDS Public Snapshots**  
Checks the permission settings for your Amazon Relation  
public.  
0 RDS snapshots are marked as public.
- ▶ **IAM Use**  
This check is intended to discourage the use of root acce  
At least one IAM user has been created for this account.

# Trusted Advisor – Monitoring

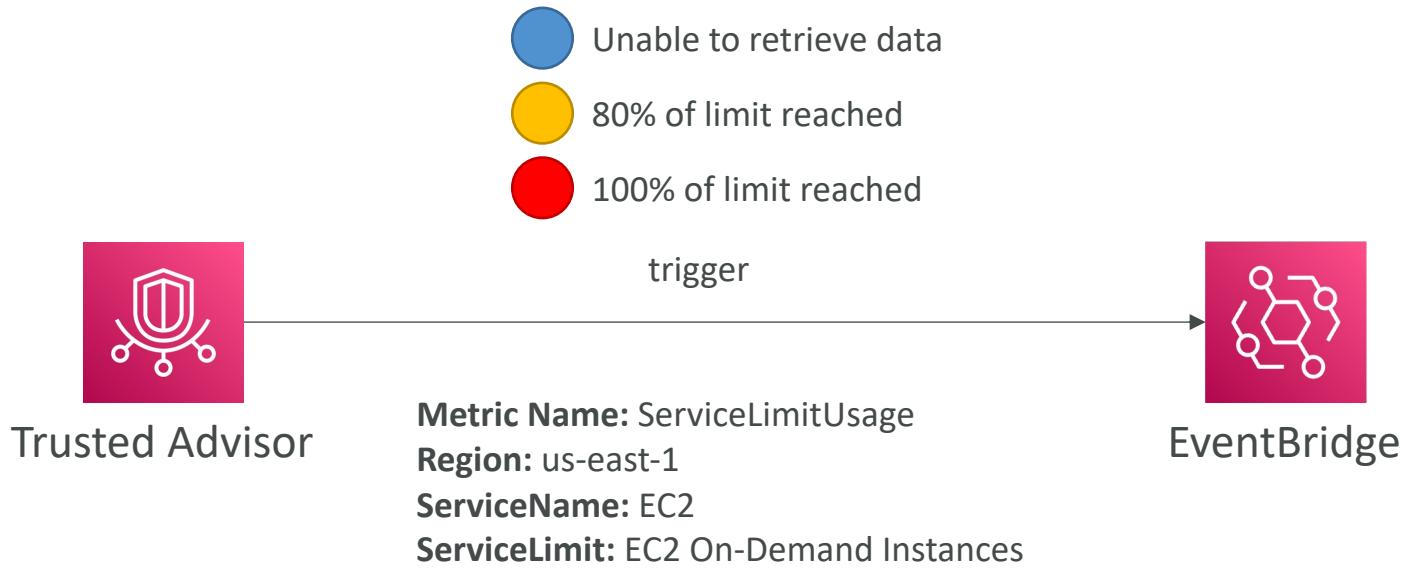


```
{
  "version": "0",
  "id": "1234abcd-ab12-123a-123a-1234567890ab",
  "detail-type": "Trusted Advisor Check Item Refresh Notification",
  "source": "aws.trustedadvisor",
  "account": "123456789012",
  "time": "2018-01-12T20:07:49Z",
  "region": "us-east-2",
  "resources": [],
  "detail": {
    "check-name": "Low Utilization Amazon EC2 Instances",
    "status": "WARN",
    "resource_id": "arn:aws:ec2:ca-central-1:123456789012:instance/i-01234567890abcdef",
    "check-item-detail": {
      "Region/AZ": "ca-central-1a",
      "Instance Name": null,
      "Instance ID": "i-01234567890abcdef",
      "Instance Type": "t2.micro",
      "Estimated Monthly Savings": "$9.22",
      "14-Day Average CPU Utilization": "0.1%",
      "14-Day Average Network I/O": "0.00MB",
      "Number of Days Low Utilization": "14 days",
    }
  }
}

{
  "Day 1": "0.1% 0.00MB",
  "Day 2": "0.1% 0.00MB",
  "Day 3": "0.1% 0.00MB",
  "Day 4": "0.1% 0.00MB",
  "Day 5": "0.1% 0.00MB",
  "Day 6": "0.1% 0.00MB",
  "Day 7": "0.1% 0.00MB",
  "Day 8": "0.1% 0.00MB",
  "Day 9": "0.1% 0.00MB",
  "Day 10": "0.1% 0.00MB",
  "Day 11": "0.1% 0.00MB",
  "Day 12": "0.1% 0.00MB",
  "Day 13": "0.1% 0.00MB",
  "Day 14": "0.1% 0.00MB"
},
"uuid": "aa12345f-55c7-498e-b7ac-123456789012"
```

# Trusted Advisor – Service Quotas

## 50+ Service Quotas are supported



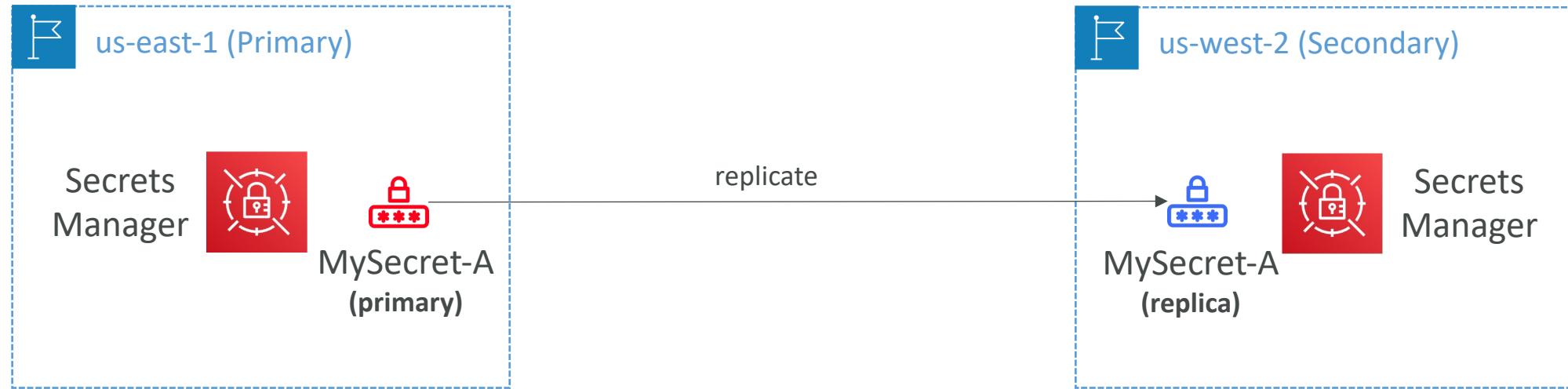
# AWS Secrets Manager



- Newer service, meant for storing secrets
- Capability to force **rotation of secrets** every X days
- Automate generation of secrets on rotation (uses Lambda)
- Integration with **Amazon RDS** (MySQL, PostgreSQL, Aurora)
- Secrets are encrypted using KMS
- Mostly meant for RDS integration

# AWS Secrets Manager – Multi-Region Secrets

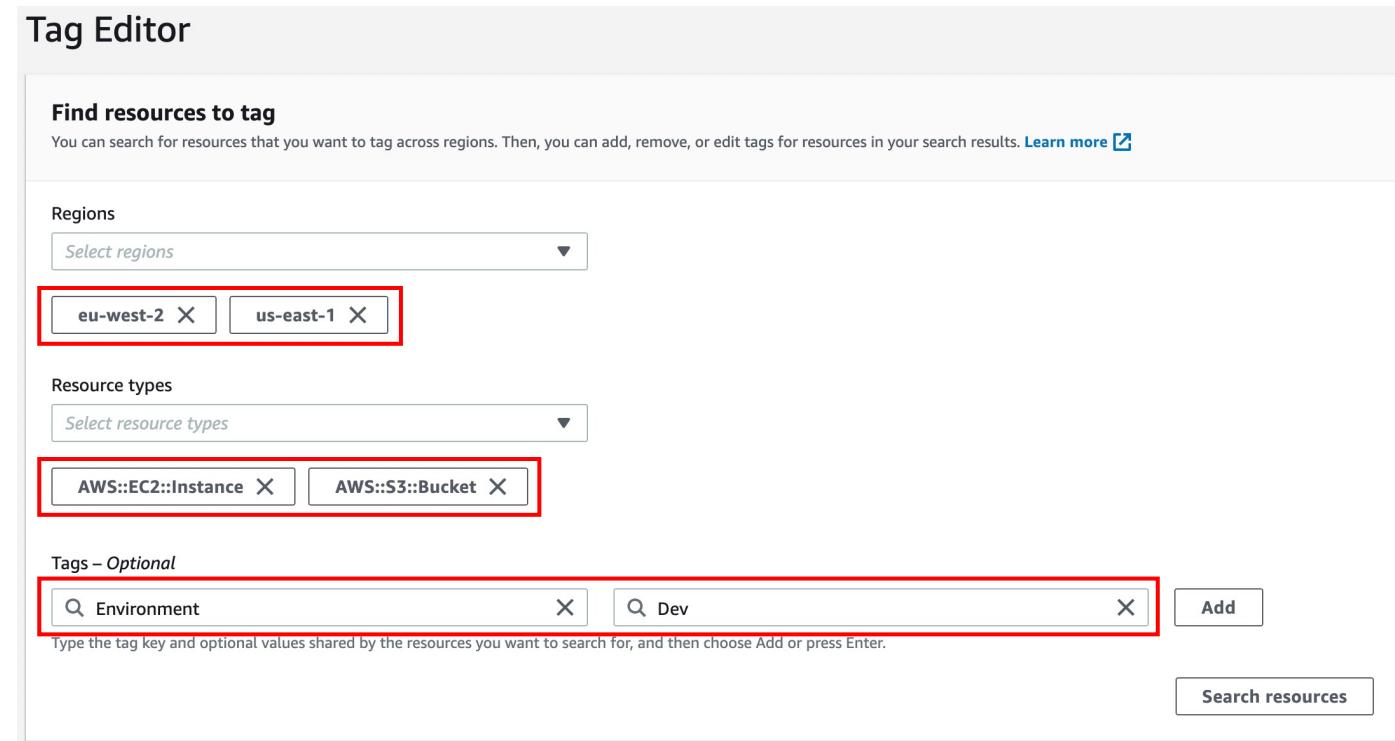
- Replicate Secrets across multiple AWS Regions
- Secrets Manager keeps read replicas in sync with the primary Secret
- Ability to promote a read replica Secret to a standalone Secret
- Use cases: multi-region apps, disaster recovery strategies, multi-region DB...



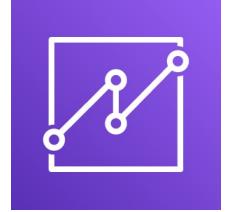
# Other Services

# AWS Tag Editor

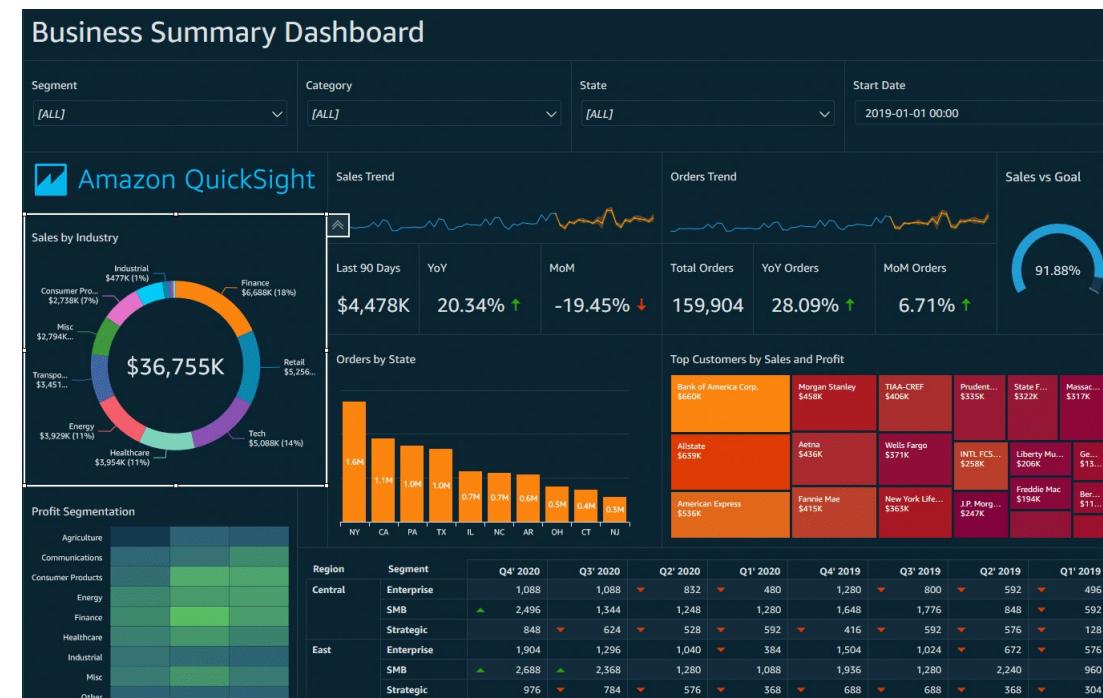
- Allows you to manage tags of multiple resources at once
- You can add/update/delete tags
- Search tagged/untagged resources in all AWS Regions



# Amazon QuickSight



- Serverless machine learning-powered business intelligence service to create interactive dashboards
- Fast, automatically scalable, embeddable, with per-session pricing
- Use cases:
  - Business analytics
  - Building visualizations
  - Perform ad-hoc analysis
  - Get business insights using data
- Integrated with RDS, Aurora, Athena, Redshift, S3...

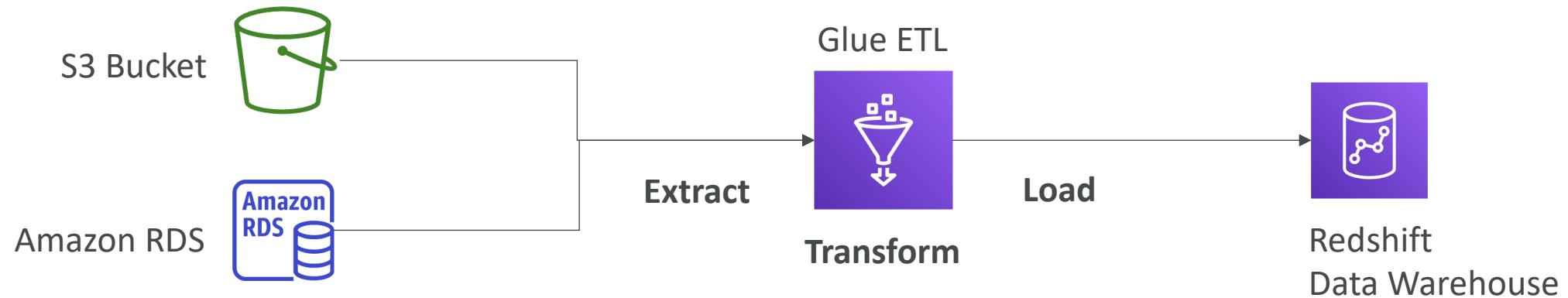


<https://aws.amazon.com/quicksight/>

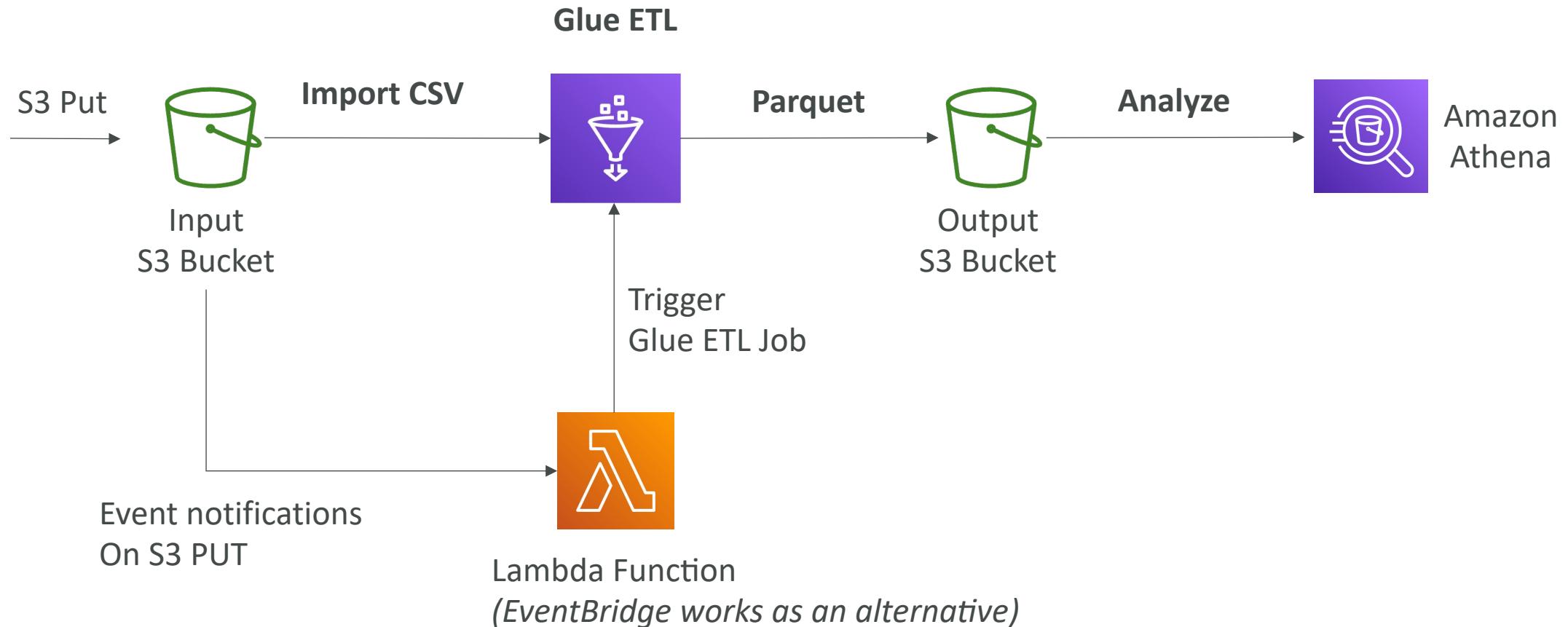
# AWS Glue



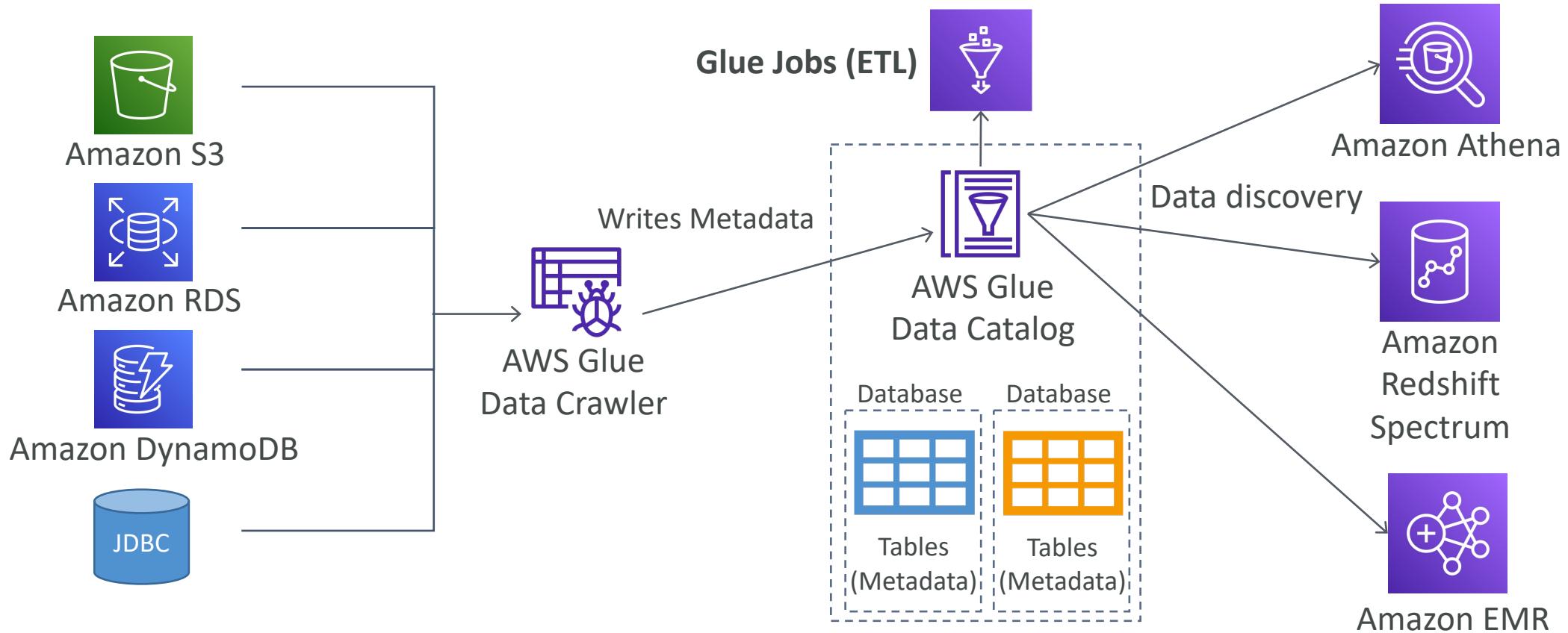
- Managed **extract, transform, and load (ETL)** service
- Useful to prepare and transform data for analytics
- Fully **serverless** service



# AWS Glue – Convert data into Parquet format



# Glue Data Catalog: catalog of datasets



# Glue – things to know at a high-level

- **Glue Job Bookmarks:** prevent re-processing old data
- **Glue Elastic Views:**
  - Combine and replicate data across multiple data stores using SQL
  - No custom code, Glue monitors for changes in the source data, serverless
  - Leverages a “virtual table” (materialized view)
- **Glue DataBrew:** clean and normalize data using pre-built transformation
- **Glue Studio:** new GUI to create, run and monitor ETL jobs in Glue
- **Glue Streaming ETL** (built on Apache Spark Structured Streaming): compatible with Kinesis Data Streaming, Kafka, MSK (managed Kafka)

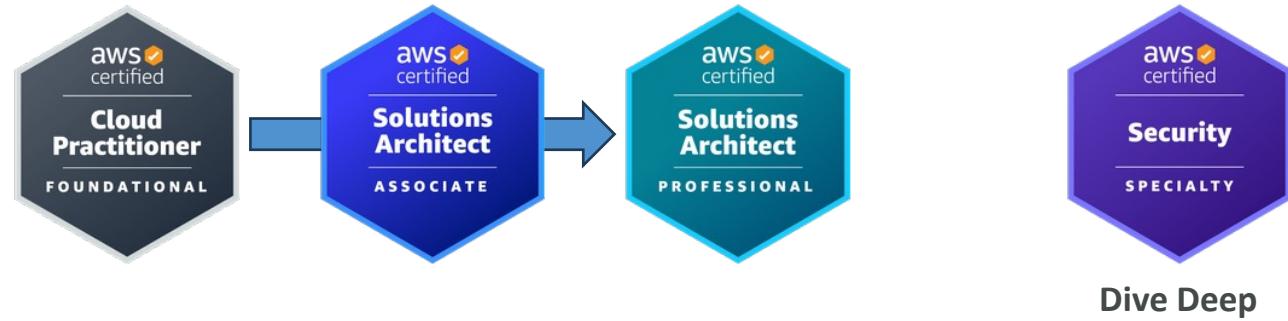
# Exam Preparation & Congratulations

# AWS Certification Paths – Architecture

## Architecture

### Solutions Architect

Design, develop, and manage cloud infrastructure and assets, work with DevOps to migrate applications to the cloud



Dive Deep

## Architecture

### Application Architect

Design significant aspects of application architecture including user interface, middleware, and infrastructure, and ensure enterprise-wide scalable, reliable, and manageable systems



Dive Deep

[https://d1.awsstatic.com/training-and-certification/docs/AWS\\_certification\\_paths.pdf](https://d1.awsstatic.com/training-and-certification/docs/AWS_certification_paths.pdf)

# AWS Certification Paths – Operations

## Operations

### Systems Administrator

Install, upgrade, and maintain computer components and software, and integrate automation processes



## Operations

### Cloud Engineer

Implement and operate an organization's networked computing infrastructure and Implement security systems to maintain data safety



# AWS Certification Paths – DevOps

## DevOps

### Test Engineer

Embed testing and quality best practices for software development from design to release, throughout the product life cycle



## DevOps

### Cloud DevOps Engineer

Design, deployment, and operations of large-scale global hybrid cloud computing environment, advocating for end-to-end automated CI/CD DevOps pipelines



## DevOps

### DevSecOps Engineer

Accelerate enterprise cloud adoption while enabling rapid and stable delivery of capabilities using CI/CD principles, methodologies, and technologies



# AWS Certification Paths – Security

## Security

### Cloud Security Engineer

Design computer security architecture and develop detailed cyber security designs.  
Develop, execute, and track performance of security measures to protect information



## Security

### Cloud Security Architect

Design and implement enterprise cloud solutions applying governance to identify, communicate, and minimize business and technical risks

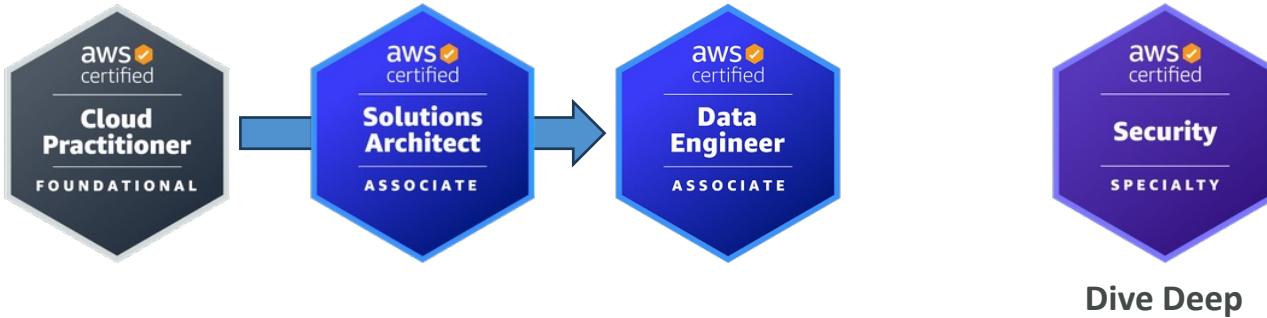


# AWS Certification Paths – Data Analytics & Development

## Data Analytics

### Cloud Data Engineer

Automate collection and processing of structured/semi-structured data and monitor data pipeline performance



Dive Deep

## Development

### Software Development Engineer

Develop, construct, and maintain software across platforms and devices

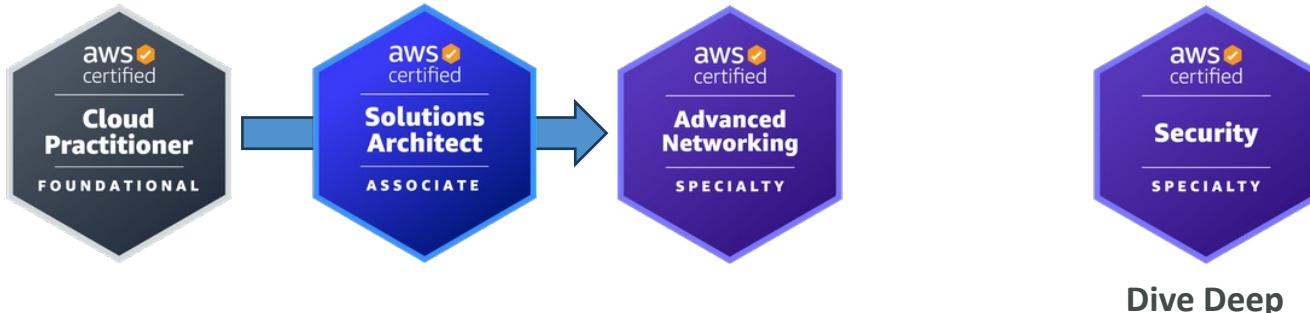


# AWS Certification Paths – Networking & AI/ML

## Networking

### Network Engineer

Design and implement computer and information networks, such as local area networks (LAN), wide area networks (WAN), intranets, extranets, etc.



## AI/ML

### Machine Learning Engineer

Research, build, and design artificial intelligence (AI) systems to automate predictive models, and design machine learning systems, models, and schemes



# Next steps

- Congratulations, you have covered all the domains!
- Make sure you revisit the lectures and practice as much as possible
- A good extra resource to do is the AWS Exam Readiness course at:
  - <https://www.aws.training/Details/eLearning?id=34146>
- Another good resource is to read the AWS DevOps blog:
  - <https://aws.amazon.com/blogs/devops/>
- The DevOps exam is hard, and tests experience...
- Practice, practice, practice!