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. Thanks!
- **Best of luck for the exam and happy learning!**

AWS Certified DevOps Engineer Professional Course

DOP-C01

Please do not skip this lecture

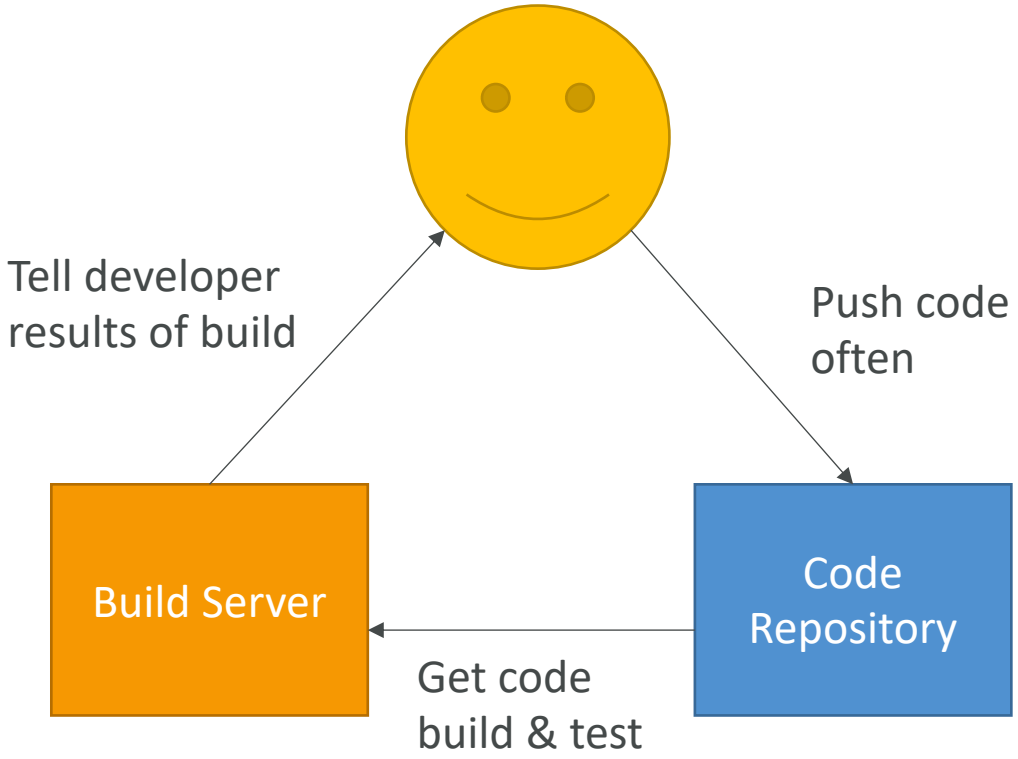
- ADVANCED, PROFESSIONAL-LEVEL COURSE
  - Do the AWS Certified Developer course & certification at a pre-requisite
  - It'll be easier if you do the AWS Certified SysOps course & certification as well
- ALL HANDS-ON
  - The AWS DevOps exam is hard and tests you on real-world experience (min 2 years)
  - This course provides you the opportunity to practice a lot
- 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 1 - SDLC Automation

Continuous Integration

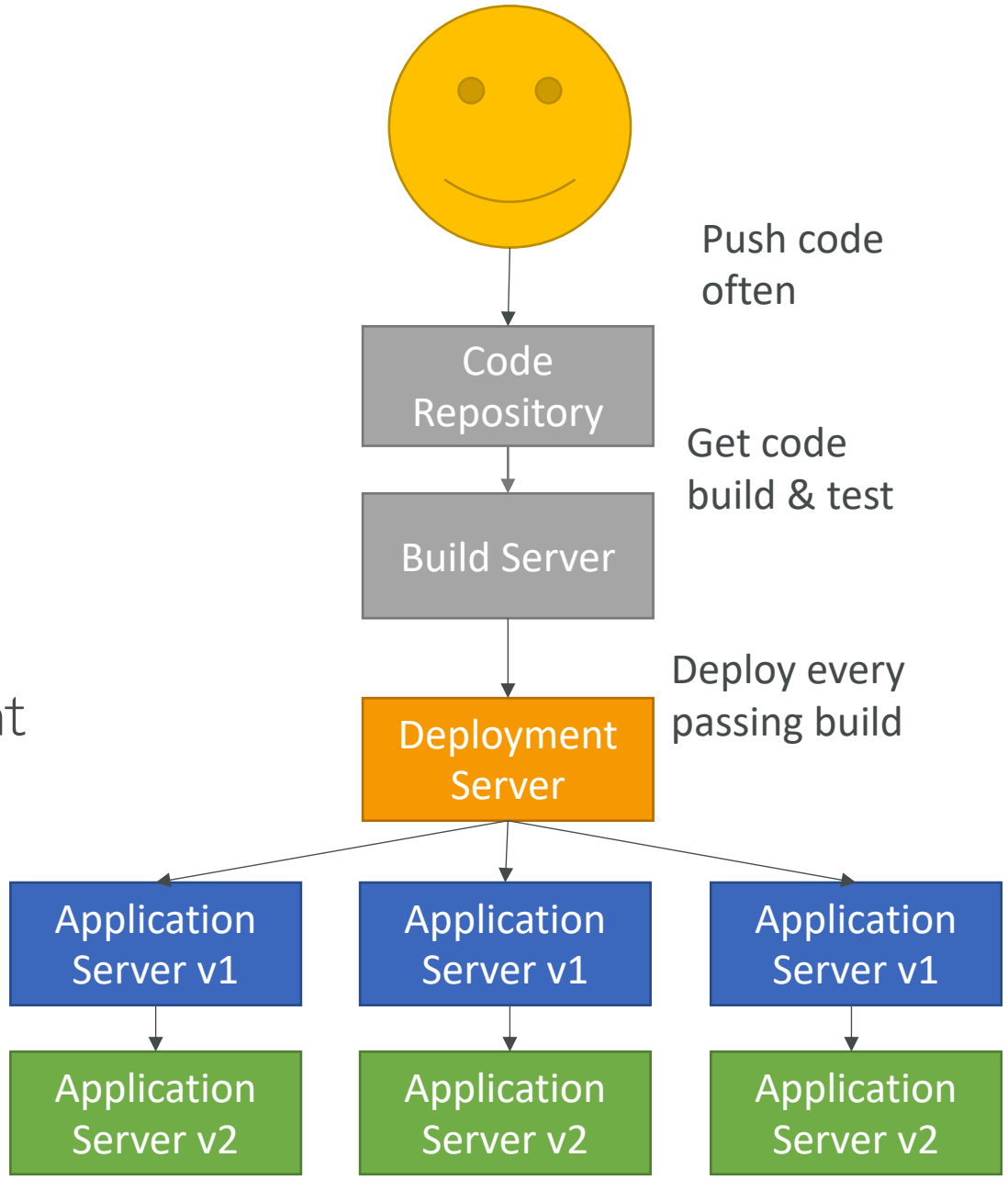
- Developers push the code to a code repository often (GitHub / CodeCommit / Bitbucket / etc…)
- A testing / build server checks the code as soon as it's pushed (CodeBuild / Jenkins CI / etc…)
- The developer gets feedback about the tests and checks that have passed / failed

- Find bugs early, fix bugs
- Deliver faster as the code is tested
- Deploy often
- Happier developers, as they're unblocked



Continuous Delivery

- Ensure that the software can be released reliably whenever needed.
- Ensures deployments happen often and are quick
- Shift away from "one release every 3 months" to "5 releases a day"
- That usually means automated deployment
  - CodeDeploy
  - Jenkins CD
  - Spinnaker
  - Etc…

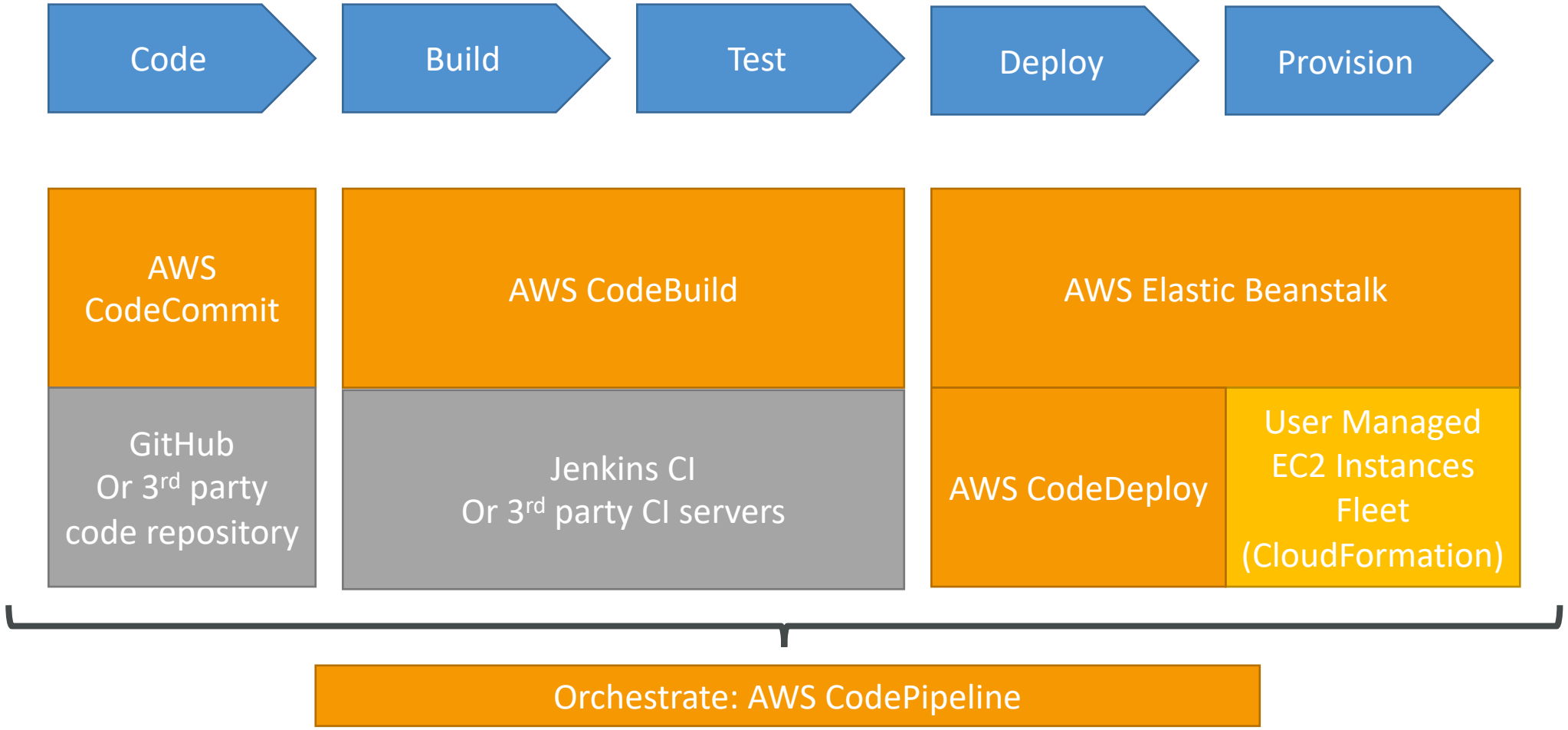


Continuous Delivery vs Continuous Deployment

- Continuous Delivery:
  - Ability to deploy often using automation
  - May involve a manual step to "approve" a deployment
  - The deployment itself is still automated and repeated!

- Continuous Deployment:
  - Full automation, every code change is deployed all the way to production
  - No manual intervention of approvals

Technology Stack for CICD

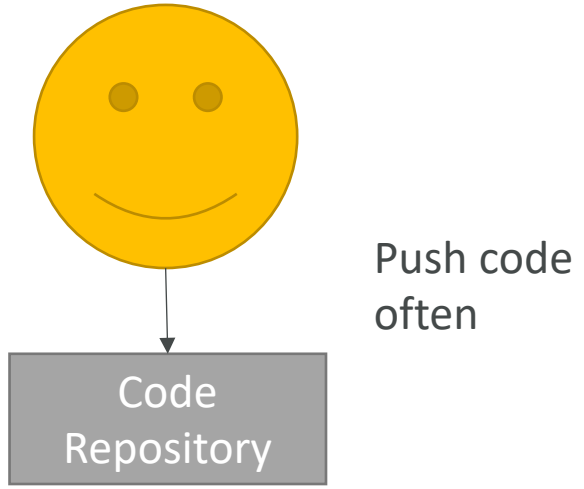


CodeCommit

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

CodeCommit

- Git repositories can be expensive.
- The industry includes:
  - GitHub: free public repositories, paid private ones
  - BitBucket
  - Etc...
- And AWS CodeCommit:
  - private Git repositories
  - No size limit on repositories (scale seamlessly)
  - Fully managed, highly available
  - Code only in AWS Cloud account => increased security and compliance
  - Secure (encrypted, access control, etc…)
  - Integrated with Jenkins / CodeBuild / other CI tools



CodeBuild Overview

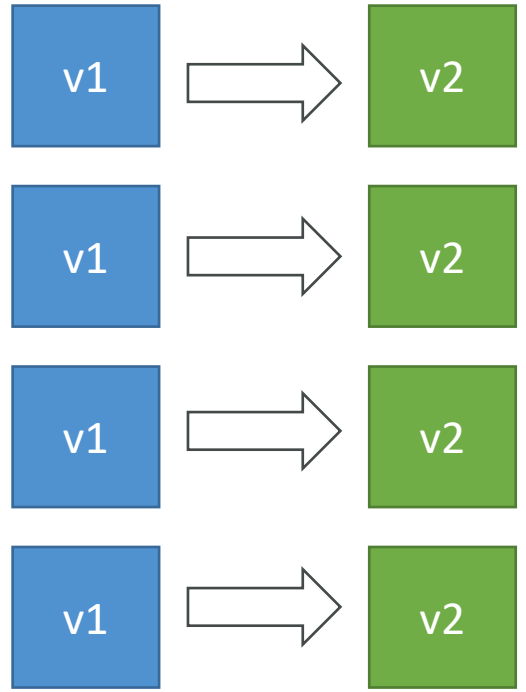
- Fully managed build service
- Alternative to other build tools such as Jenkins
- Continuous scaling (no servers to manage or provision – no build queue)
- Pay for usage: the time it takes to complete the builds
- Leverages Docker under the hood for reproducible builds
- Possibility to extend capabilities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

CodeBuild Overview

- Source Code from GitHub / CodeCommit / CodePipeline / S3…
- Build instructions can be defined in code (buildspec.yml file)
- Output logs to Amazon S3 & AWS CloudWatch Logs
- Metrics to monitor CodeBuild statistics
- Use CloudWatch Events to detect failed builds and trigger notifications
- Use CloudWatch Alarms to notify if you need "thresholds" for failures
- CloudWatch Events / AWS Lambda as a Glue
- SNS notifications

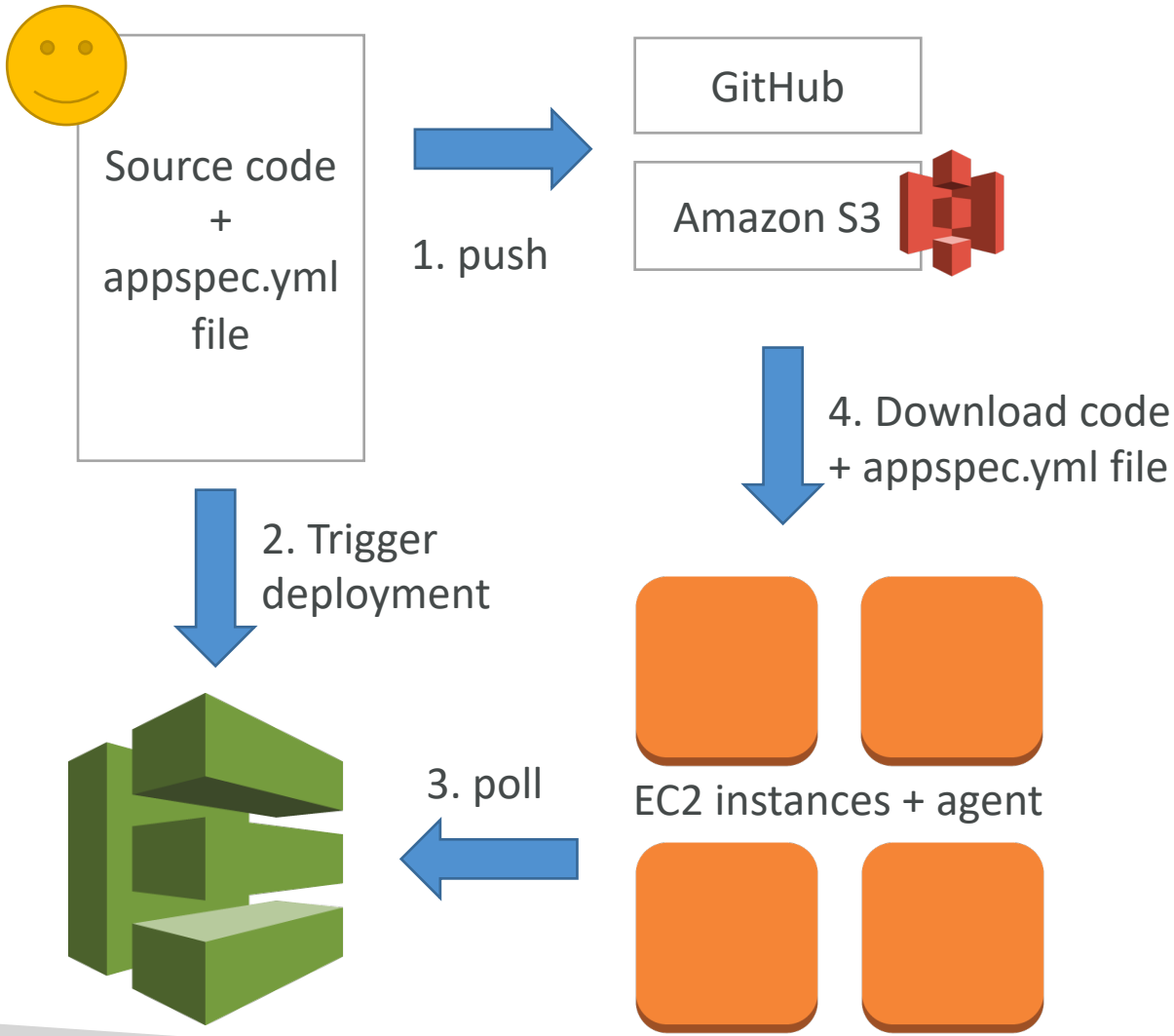
AWS CodeDeploy

- We want to deploy our application automatically to many EC2 instances
- There are several ways to handle deployments using open source tools (Ansible, Terraform, Chef, Puppet, etc…)
- We can use the managed Service AWS CodeDeploy



AWS CodeDeploy – Steps to make it work

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



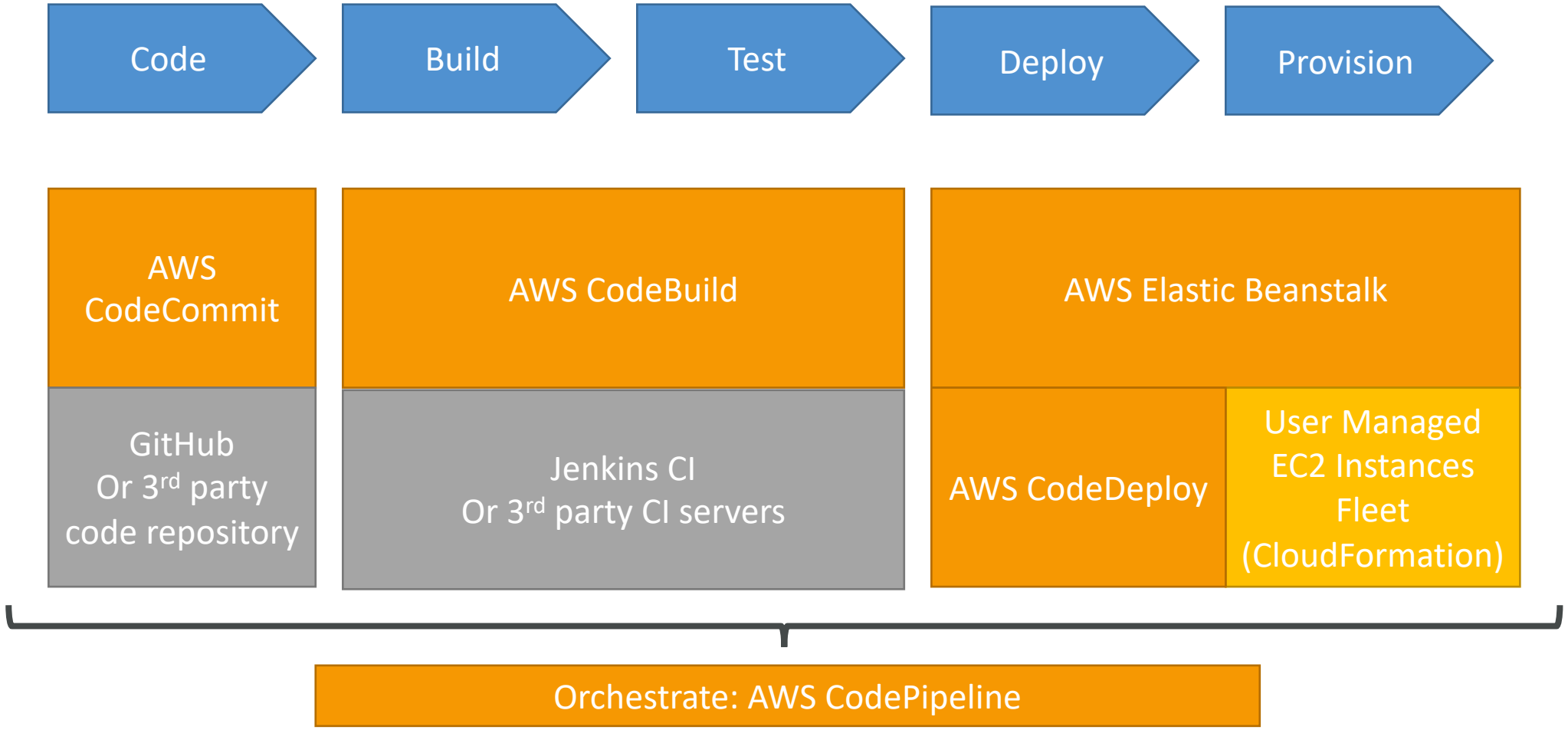
AWS CodeDeploy

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

CodePipeline

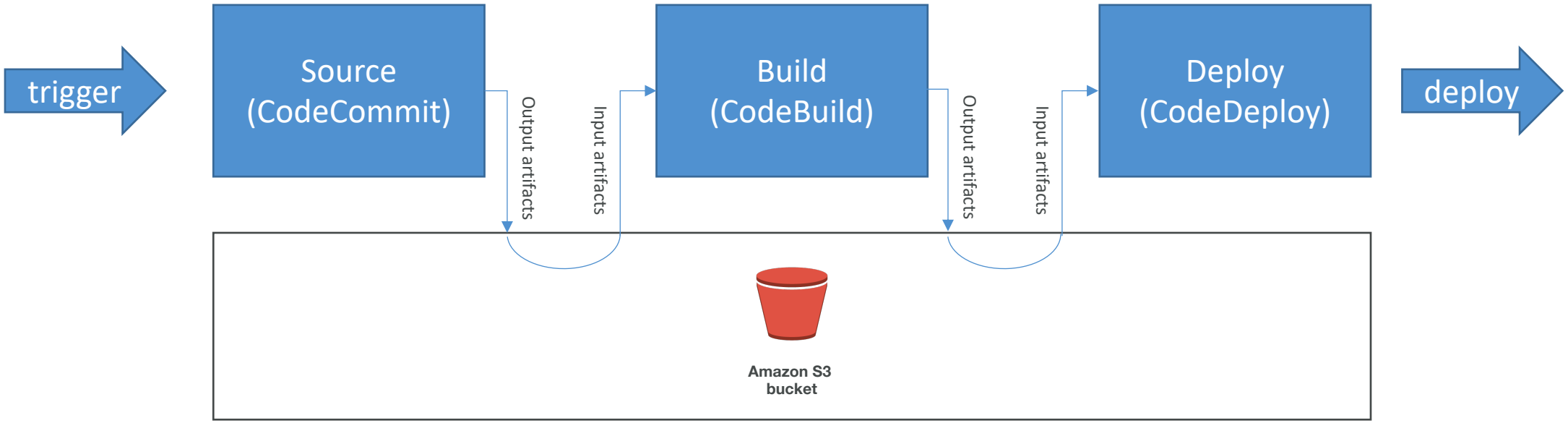
- Continuous delivery
- Visual workflow
- Source: GitHub / CodeCommit / Amazon S3
- Build: CodeBuild / Jenkins / etc…
- Load Testing: 3rd party tools
- Deploy: AWS CodeDeploy / Beanstalk / CloudFormation / ECS…
- Made of stages:
  - Each stage can have sequential actions and / or parallel actions
  -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

Technology Stack for CICD



AWS CodePipeline Artifacts

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



Jenkins on AWS

- Open Source CICD tool
- Can replace CodeBuild, CodePipeline & CodeDeploy
- Must be deployed in a Master / Slave configuration
- Must manage multi-AZ, deploy on EC2, etc...
- All projects must have a "Jenkinsfile" (similar to buildspec.yml) to tell Jenkins what to do

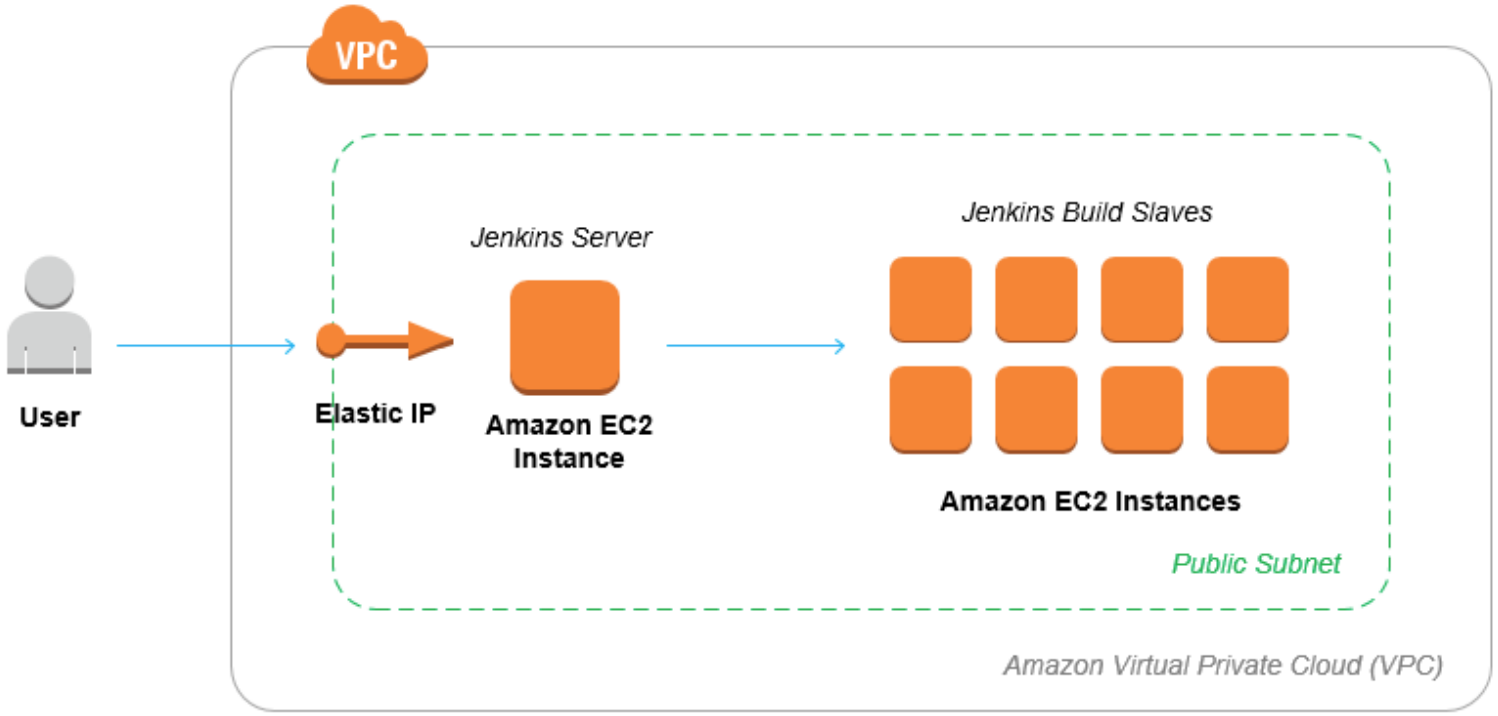
- Jenkins can be extended on AWS thanks to many plugins!

Jenkins Master / Slave (build farm)

Figure 2: Master and Worker deployment options

From whitepaper: Jenkins on AWS
https://d1.awsstatic.com/whitepapers/jenkins-on-aws.pdf

Jenkins on AWS



Source: https://aws.amazon.com/getting-started/projects/setup-jenkins-build-server/

Jenkins Master / Slave

Figure 3: Jenkins deployment strategies

From whitepaper: Jenkins on AWS
https://d1.awsstatic.com/whitepapers/jenkins-on-aws.pdf

Jenkins with CodePipeline

Figure 9: A simple workflow using AWS services and Jenkins

From whitepaper: Jenkins on AWS
https://d1.awsstatic.com/whitepapers/jenkins-on-aws.pdf

Jenkins with ECS

Figure 10: Using AWS services and Jenkins to deploy a container

From whitepaper: Jenkins on AWS
https://d1.awsstatic.com/whitepapers/jenkins-on-aws.pdf

Jenkins with Device Farm

Figure 11: Using AWS services and Jenkins to test a mobile application

From whitepaper: Jenkins on AWS
https://d1.awsstatic.com/whitepapers/jenkins-on-aws.pdf

Jenkins with AWS Lambda

Figure 12: Using AWS and Jenkins for serverless code management

From whitepaper: Jenkins on AWS
https://d1.awsstatic.com/whitepapers/jenkins-on-aws.pdf

Jenkins with CloudFormation

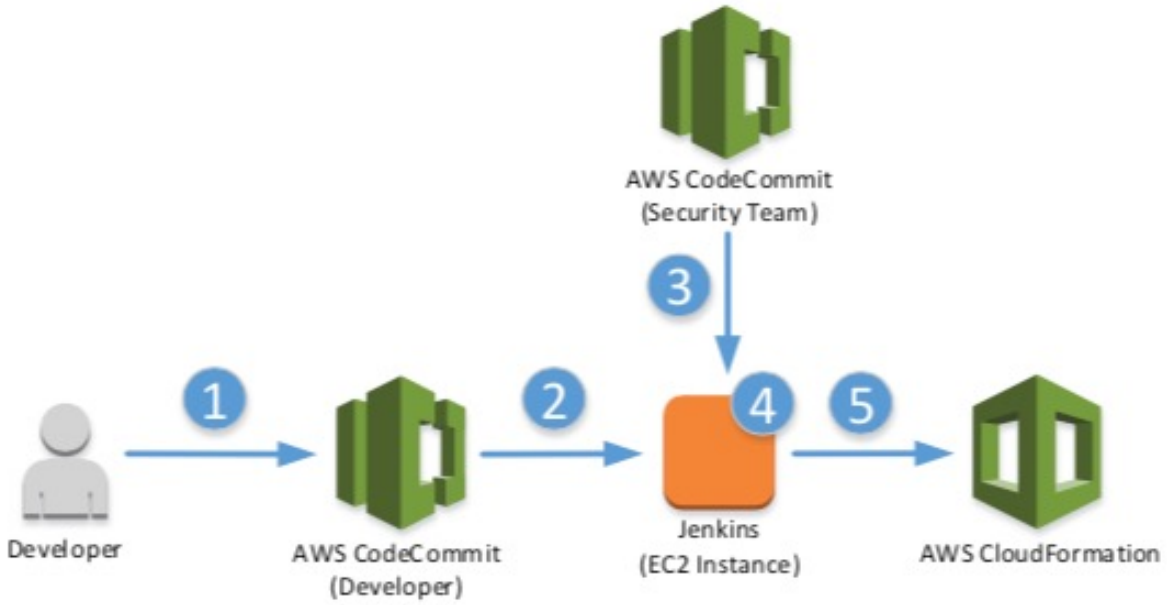


Figure 13: First example—Using AWS services and Jenkins to automate a security framework

From whitepaper: Jenkins on AWS
https://d1.awsstatic.com/whitepapers/jenkins-on-aws.pdf

Infrastructure as Code

- Currently, we have been doing a lot of manual work
- All this manual work will be very tough to reproduce:
  - In another region
  - in another AWS account
  - Within the same region if everything was deleted

- Wouldn't it be great, if all our infrastructure was… code?
- That code would be deployed and create / update / delete our infrastructure

What is CloudFormation

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

Benefits of AWS CloudFormation (1/2)

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

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

Benefits of AWS CloudFormation (2/2)

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

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

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

How CloudFormation Works

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

Deploying CloudFormation templates

- Manual way:
  - Editing templates in the CloudFormation Designer
  - Using the console to input parameters, etc

- Automated way:
  - Editing templates in a YAML file
  - Using the AWS CLI (Command Line Interface) to deploy the templates
  - Recommended way when you fully want to automate your flow

CloudFormation Building Blocks

Templates components (one course section for each):

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

Templates helpers:

1. References
2. Functions

Note:
This is an introduction to CloudFormation

- It can take over 3 hours to properly learn and master CloudFormation
- This section is meant so you get a good idea of how it works
- We'll be slightly less hands-on than in other sections

- We'll learn everything we need to answer questions for the exam
- The exam does not require you to actually write CloudFormation
- The exam expects you to understand how to read CloudFormation

Introductory Example

- We're going to create a simple EC2 instance.
- Then we're going to create to add an Elastic IP to it
- And we're going to add two security groups to it
- For now, forget about the code syntax.
- We'll look at the structure of the files later on

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

YAML Crash Course

```yaml
invoice:        34843
date   :        2001-01-23
bill-to:
    given  :    Chris
    family :    Dumars
    address:
        lines: |
            458 Walkman Dr.
            Suite #292
        city     : Royal Oak
        state    : MI
        postal   : 48046
product:
    - 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!

What are resources?

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

- AWS figures out creation, updates and deletes of resources for us
- There are over 224 types of resources (!)
- Resource types identifiers are of the form:

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

How do I find resources documentation?

- I can't teach you all of the 224 resources, but I can teach you how to learn how to use them.
- All the resources can be found here: http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-template-resource-type-ref.html
- Then, we just read the docs ☺
- Example here (for an EC2 instance): http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-ec2-instance.html

Analysis of CloudFormation Template

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

FAQ for resources

- Can I create a dynamic amount of resources?

➢ No, you can't. Everything in the CloudFormation template has to be declared. You can't perform code generation there

- Is every AWS Service supported?

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

➢ You can work around that using AWS Lambda Custom Resources

What are parameters?

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

When should you use a parameter?

- Ask yourself this:
  - Is this CloudFormation resource configuration likely to change in the future?
  - If so, make it a parameter.

- You won't have to re-upload a template to change its content ☺

```
Parameters:
  SecurityGroupDescription:
    Description: Security Group Description
    (Simple parameter)
    Type: String
```

Parameters Settings

Parameters can be controlled by all these settings:

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

How to Reference a Parameter

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

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

Concept: Pseudo Parameters

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

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

What are mappings?

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

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

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

When would you use mappings vs parameters ?

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

- Use parameters when the values are really user specific

Fn::FindInMap
Accessing Mapping Values

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

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

What are outputs?

- The Outputs section declares *optional* outputs values that we can import into other stacks (if you export them first)!
- You can also view the outputs in the AWS Console or in using the AWS CLI
- They're very useful for example if you define a network CloudFormation, and output the variables such as VPC ID and your Subnet IDs
- It's the best way to perform some collaboration cross stack, as you let expert handle their own part of the stack
- You can't delete a CloudFormation Stack if its outputs are being referenced by another CloudFormation stack

Outputs Example

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

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

Cross Stack Reference

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

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

What are conditions used for?

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

How to define a condition?

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

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

Using a Condition

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

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

CloudFormation
Must Know Intrisic Functions

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

Fn::Ref

- The Fn::Ref function can be leveraged to reference
  - Parameters => returns the value of the parameter
  - Resources => returns the physical ID of the underlying resource (ex: EC2 ID)
- The shorthand for this in YAML is !Ref

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

Fn::GetAtt

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

```yaml
Resources:
  EC2Instance:
    Type: "AWS::EC2::Instance"
    Properties:
      ImageId: ami-1234567
      InstanceType: t2.micro
```

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

Fn::FindInMap
Accessing Mapping Values

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

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

Fn::ImportValue

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

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

Fn::Join

- Join values with a delimiter

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

- This creates "a:b:c"

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

Function Fn::Sub

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

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

```
!Sub String
```

Condition Functions

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

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

User Data in EC2 for CloudFormation

- We can have user data at EC2 instance launch through the console
- We can also include it in CloudFormation

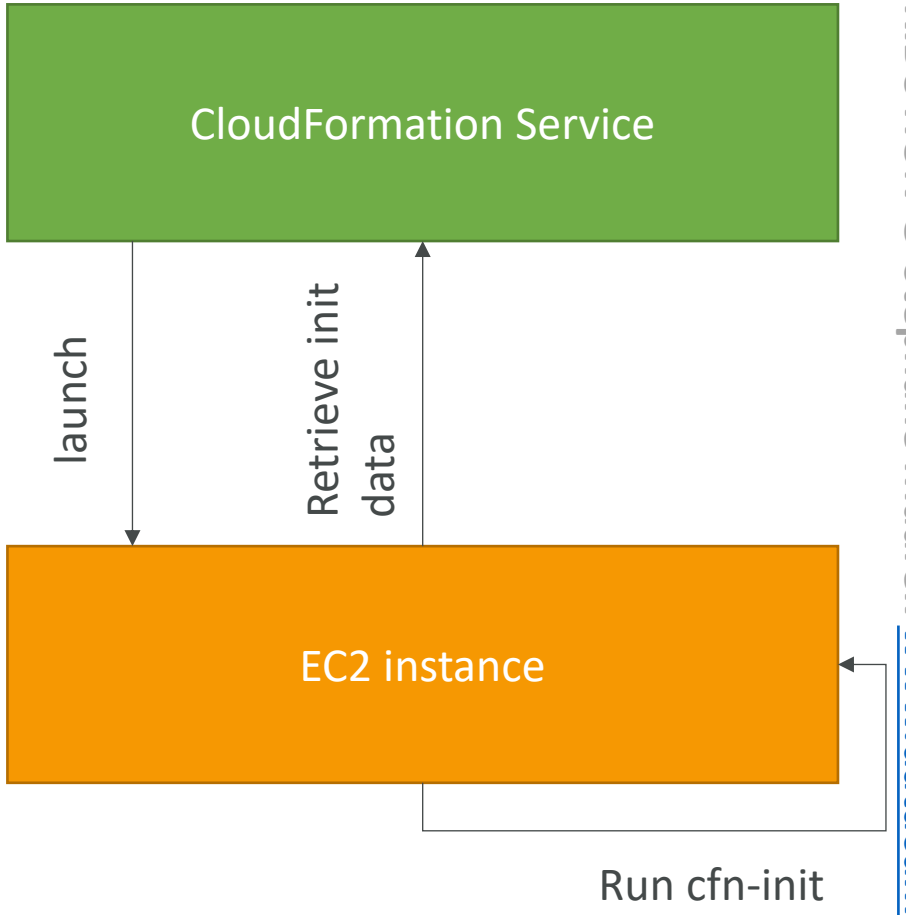
- The important thing to pass is the entire script through the function Fn::Base64
- Good to know: user data script log is in /var/log/cloud-init-output.log

- Let's see how to do this in CloudFormation

cfn-init

- AWS::CloudFormation::Init must be in the Metadata of a resource
- With the cfn-init script, it helps make complex EC2 configurations readable
- The EC2 instance will query the CloudFormation service to get init data
- Logs go to **/var/log/cfn-init.log**

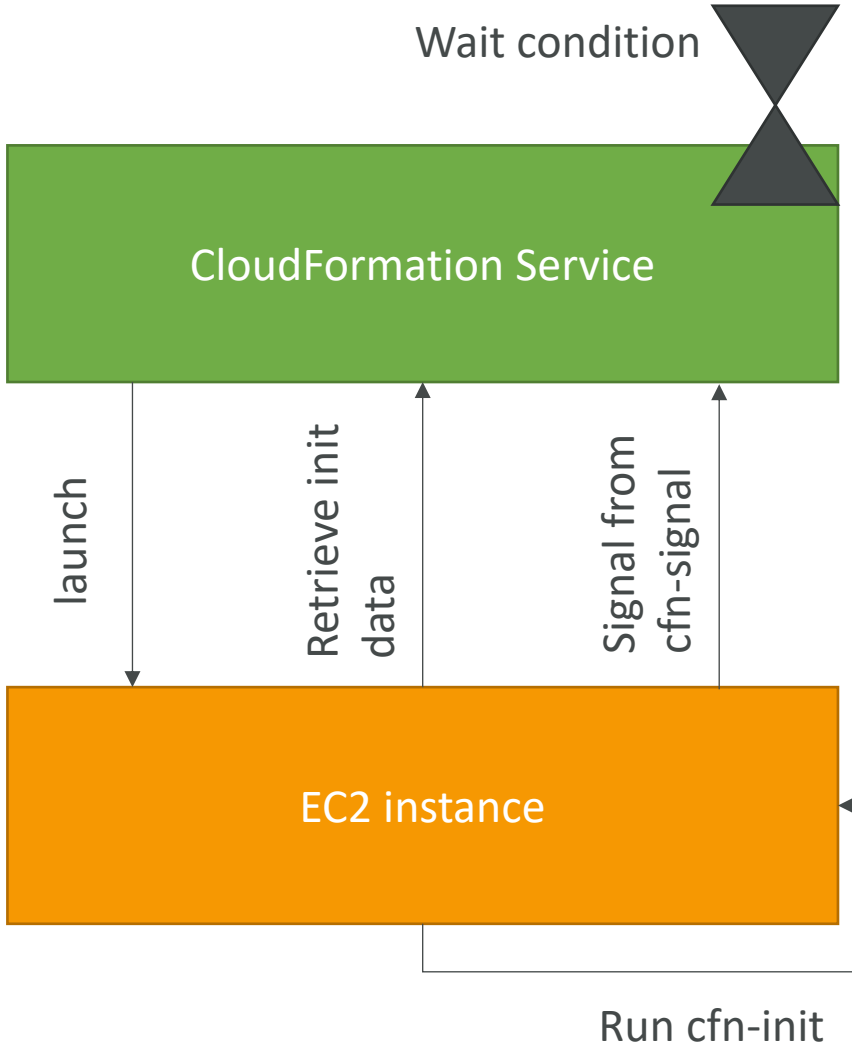
- Let's see how it works through a sample CloudFormation



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 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)



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 is 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

Rollbacks on failures

- Stack Creation Fails: (CreateStack API)
  - Default: everything rolls back (gets deleted). We can look at the log **OnFailure=ROLLBACK**
  - Troubleshoot: Option to disable rollback and manually troubleshoot **OnFailure=DO_NOTHING**
  - Delete: get rid of the stack entirely, do not keep anything **OnFailure=DELETE**

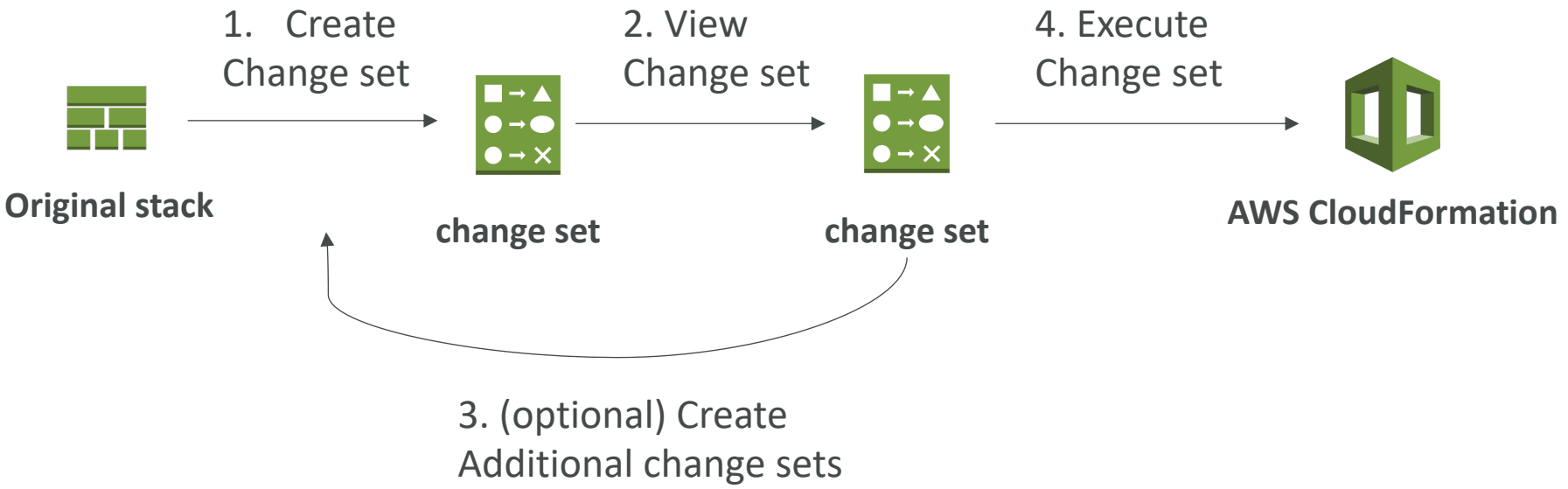
- Stack Update Fails: (UpdateStack API)
  - The stack automatically rolls back to the previous known working state
  - Ability to see in the log what happened and error messages

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)

ChangeSets

- When you update a stack, you need to know what changes before it happens for greater confidence
- ChangeSets won't say if the update will be successful



From: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/using-cfn-updating-stacks-changesets.html

Retaining Data on Deletes

- You can put a DeletionPolicy on any resource to control what happens when the CloudFormation template is deleted
- **DeletionPolicy=Retain**:
  - Specify on resources to preserve / backup in case of CloudFormation deletes
  - To keep a resource, specify **Retain** (works for any resource / nested stack)
- **DeletionPolicy=Snapshot:**
  - EBS Volume, ElastiCache Cluster, ElastiCache ReplicationGroup
  - RDS DBInstance, RDS DBCluster, Redshift Cluster
- **DeletePolicy=Delete (default behavior)**:
  - Note: for **AWS::RDS::DBCluster** resources, the default policy is Snapshot
  - Note: to delete an S3 bucket, you need to first empty the bucket of its content

Termination Protection on Stacks

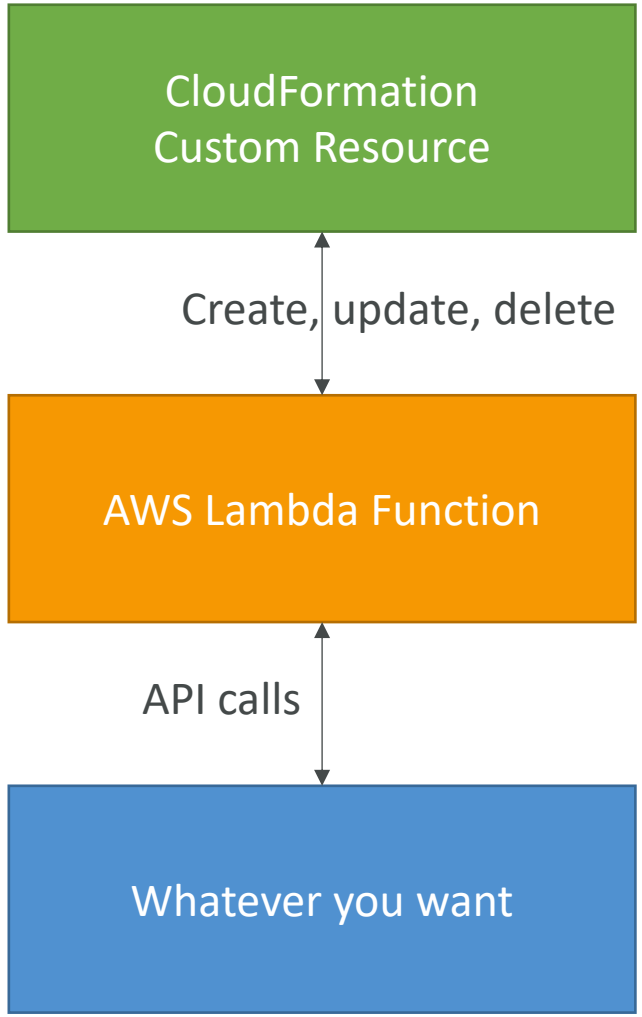
- To prevent accidental deletes of CloudFormation templates, use TerminationProtection

- Let's see this quickly!

CloudFormation Custom Resources (Lambda)

- You can define a Custom Resource in CloudFormation to address any of these use cases:

- An AWS resource is yet not covered (new service for example)
- An On-Premise resource
- Emptying an S3 bucket before being deleted
- Fetch an AMI id
- Anything you want…!



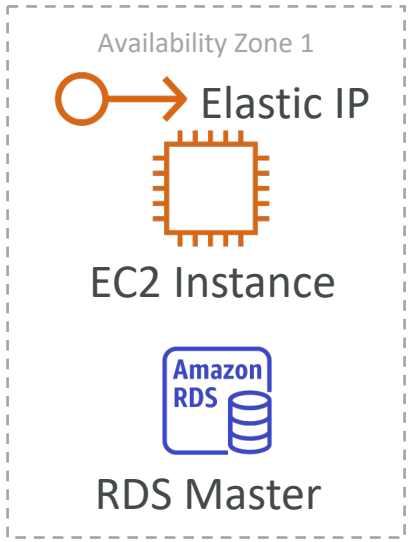
CloudFormation Custom Resources (Lambda)

- The Lambda Function will get invoked only if there is a Create, Update or Delete event, not every time you run the template

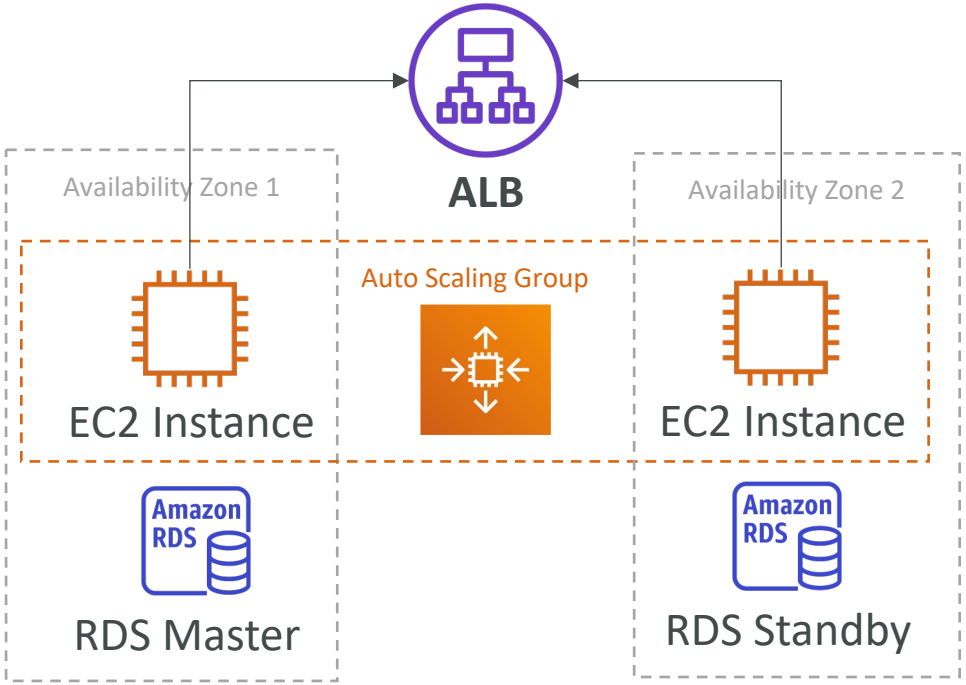
```
{
  "RequestType" : "Create",
  "ResponseURL" : "http://pre-signed-S3-url-for-response",
  "StackId" : "arn:aws:cloudformation:us-west-2:123456789012:stack/stack-name/guid",
  "RequestId" : "unique id for this create request",
  "ResourceType" : "Custom::TestResource",
  "LogicalResourceId" : "MyTestResource",
  "ResourceProperties" : {
    "Name" : "Value",
    "List" : [ "1", "2", "3" ]
  }
}
```

Elastic Beanstalk Deployment Modes

Single Instance
Great for dev



High Availability with Load Balancer
Great for prod

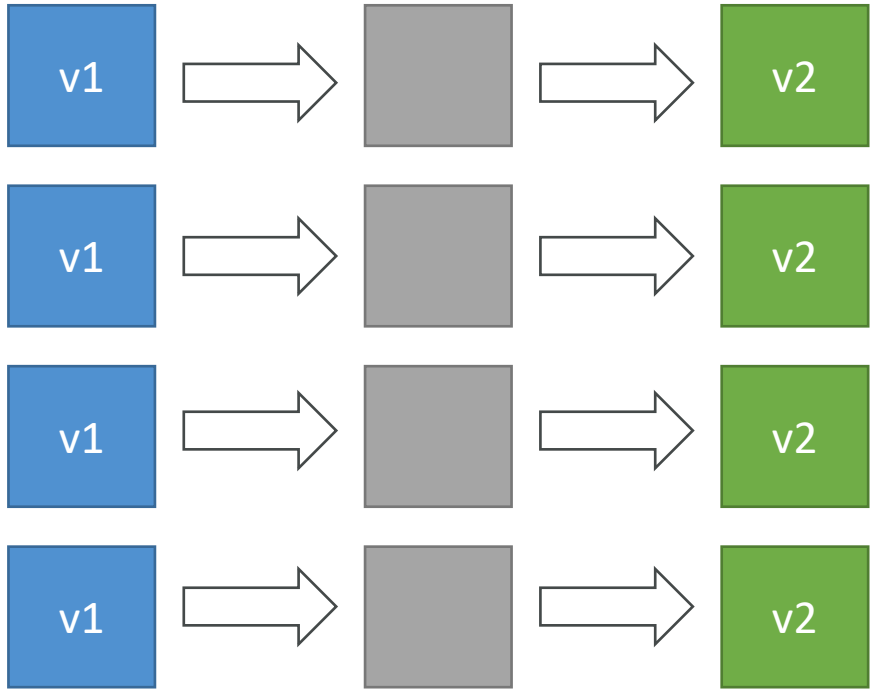


Beanstalk Deployment Options for Updates

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

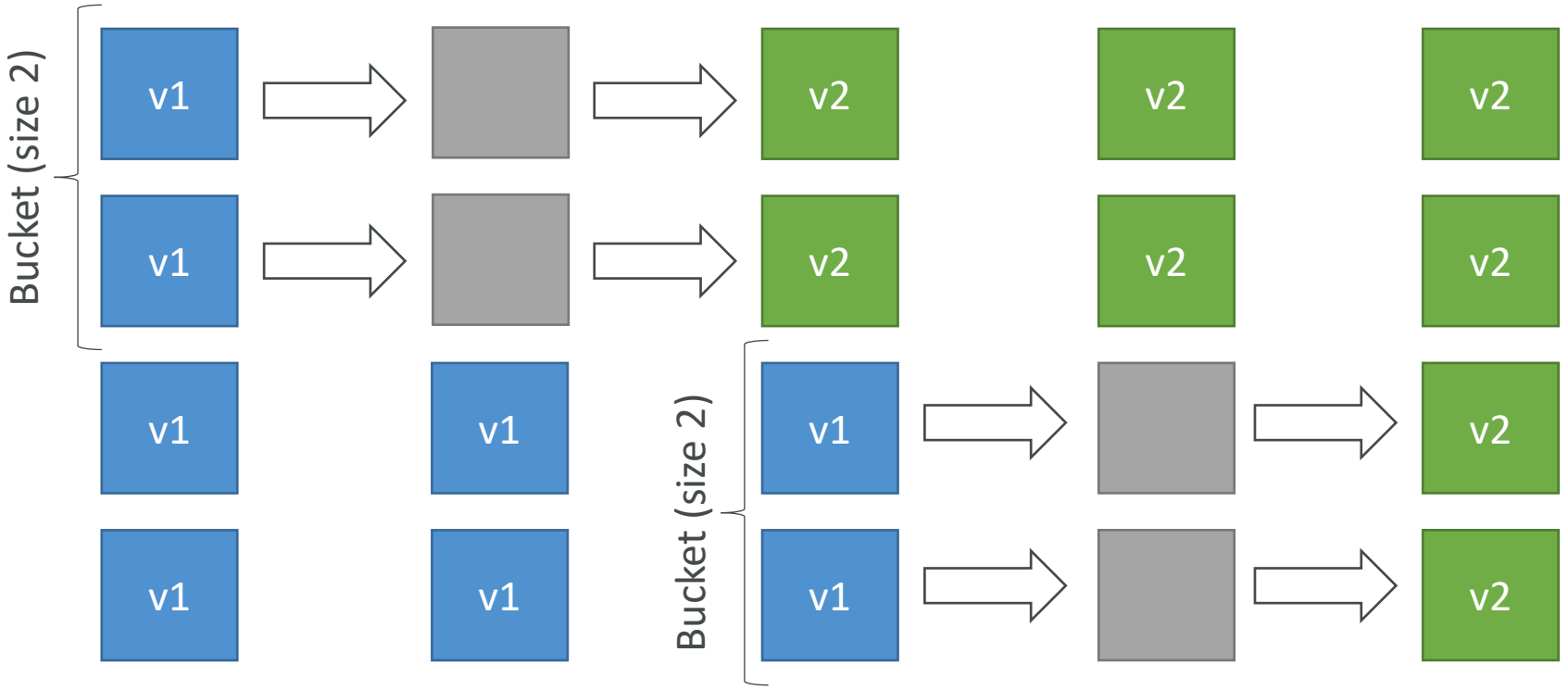
Elastic Beanstalk Deployment
All at once

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



Elastic Beanstalk Deployment Rolling

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

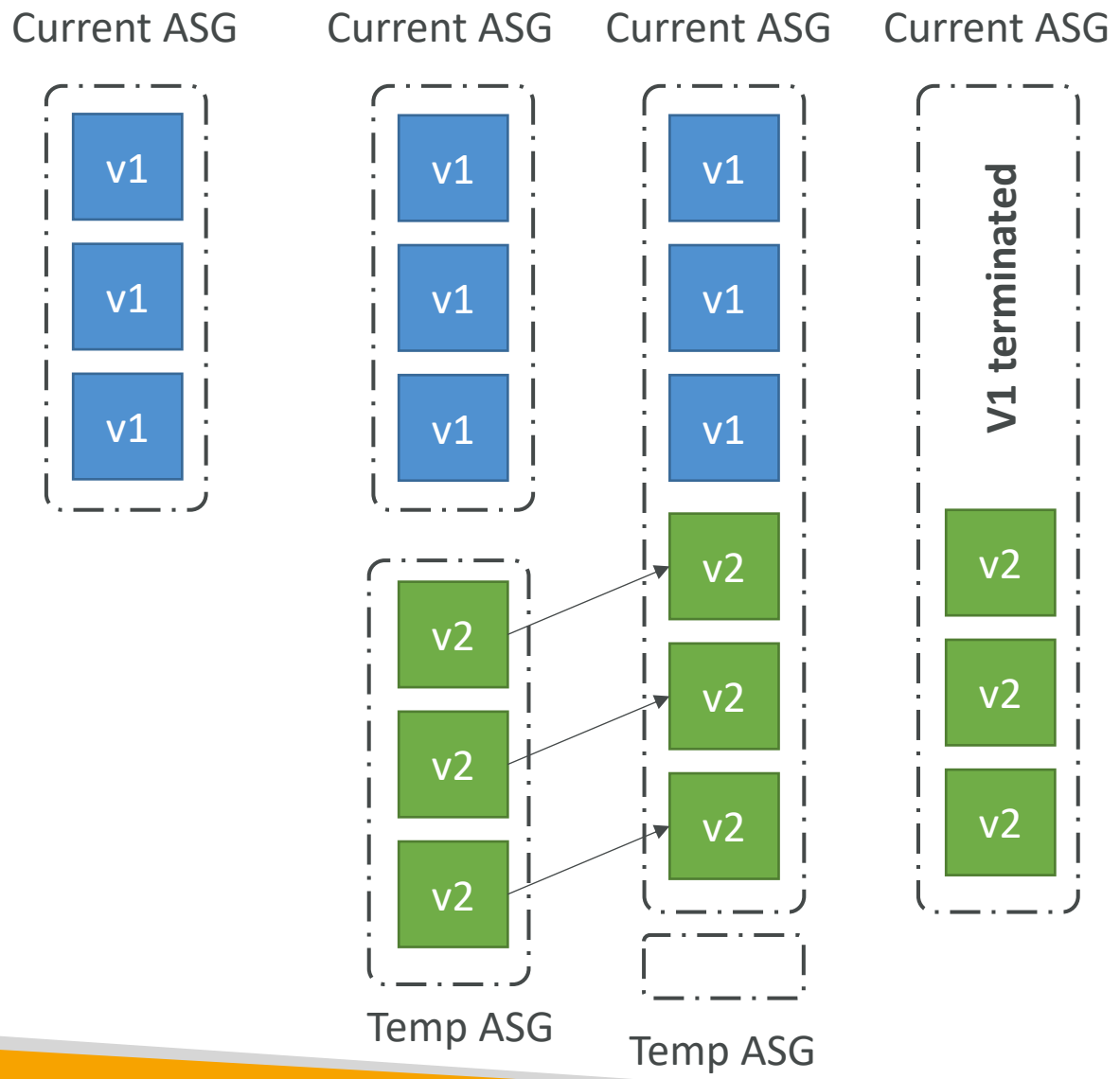


Elastic Beanstalk Deployment Rolling with additional batches

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

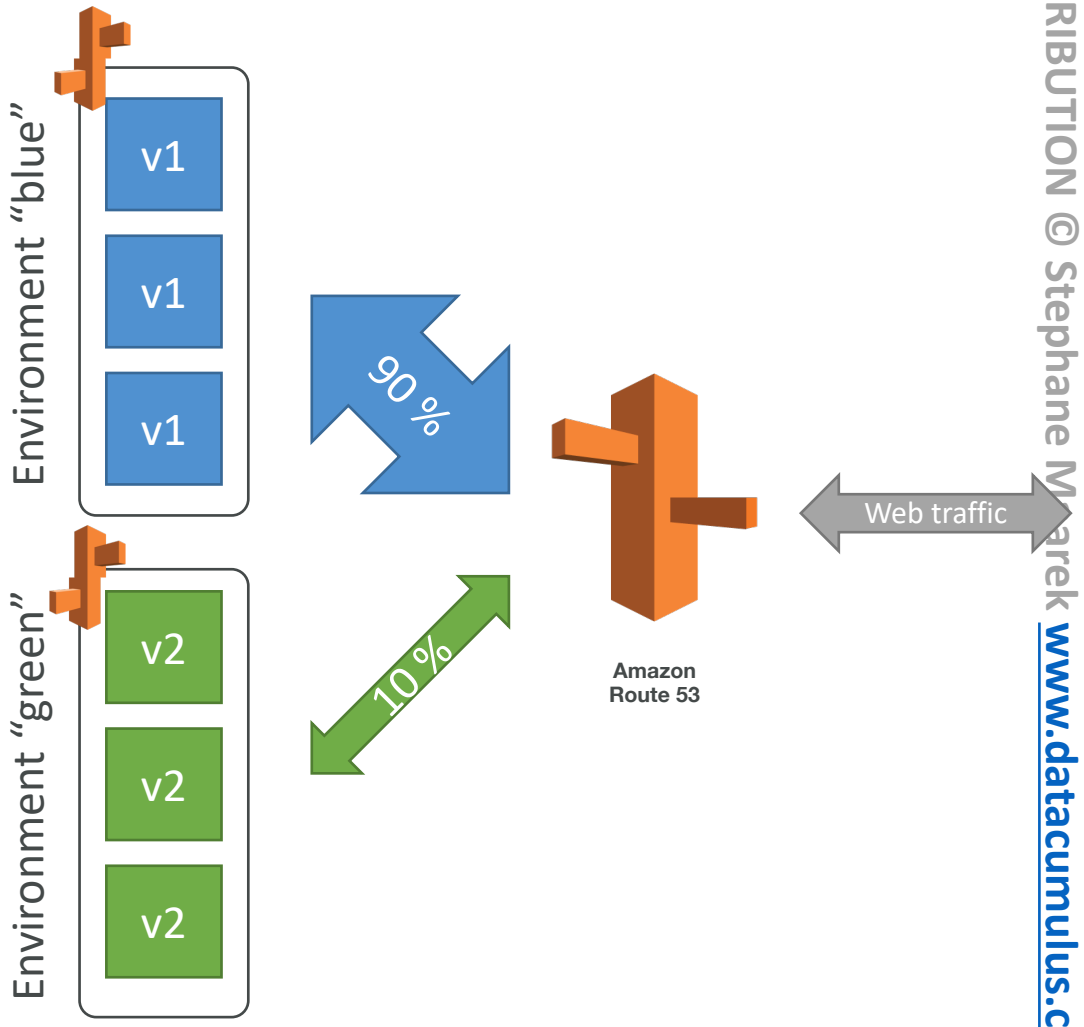
Elastic Beanstalk Deployment Immutable

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



Elastic Beanstalk Deployment Blue / Green

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



Elastic Beanstalk Deployment Summary from AWS Doc

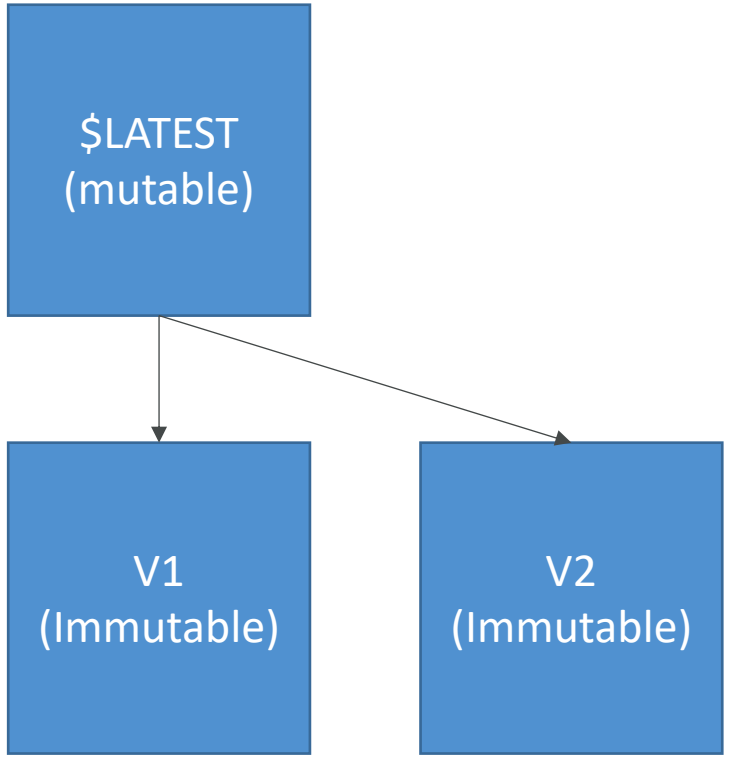
- https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features.deploy-existing-version.html

Deployment Methods

Method	Impact of Failed Deployment	Deploy Time	Zero Downtime	No DNS Change	Rollback Process	Code Deployed To
All at once	Downtime	◷	X	✓	Manual Redeploy	Existing instances
Rolling	Single batch out of service; any successful batches prior to failure running new application version	◷ ◷ †	✓	✓	Manual Redeploy	Existing instances
Rolling with additional batch	Minimal if first batch fails, otherwise, similar to **Rolling**	◷ ◷ ◷ †	✓	✓	Manual Redeploy	New and existing instances
Immutable	Minimal	◷ ◷ ◷ ◷	✓	✓	Terminate New Instances	New instances
Blue/green	Minimal	◷ ◷ ◷ ◷	✓	X	Swap URL	New instances

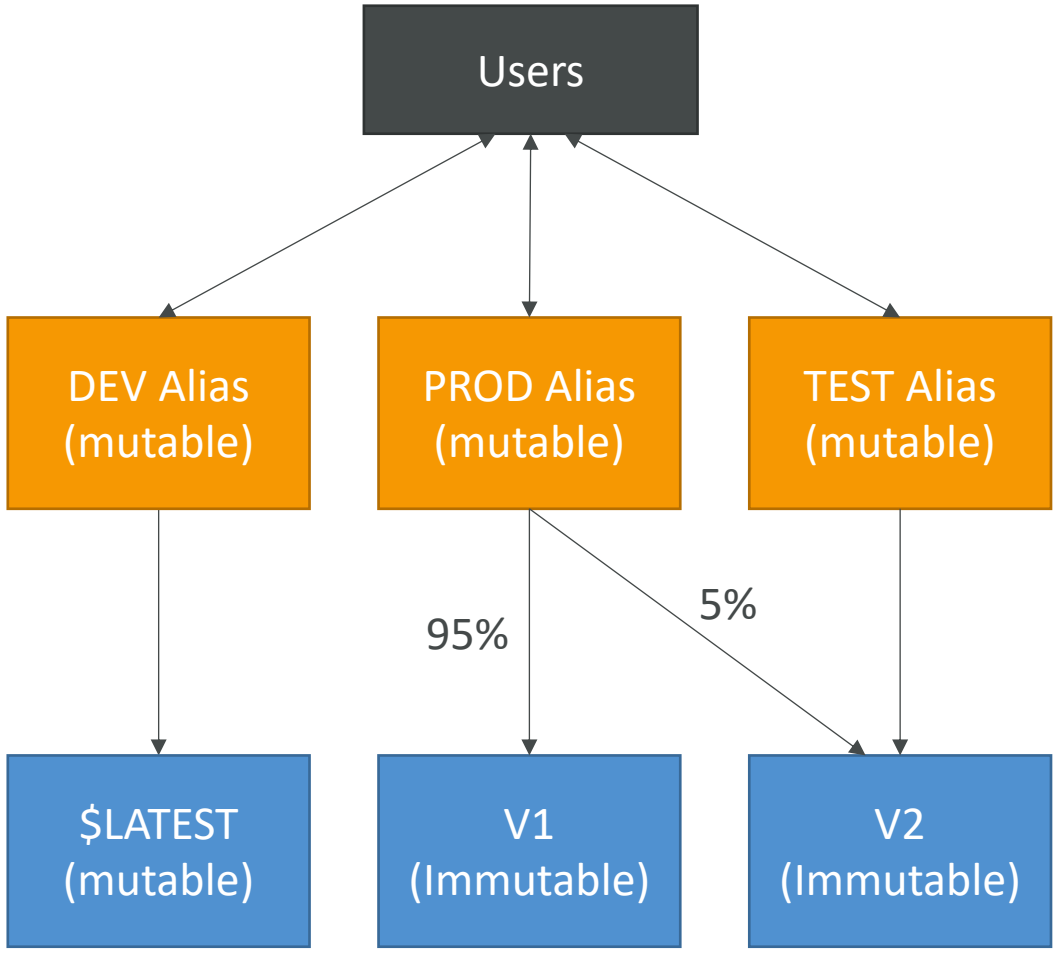
AWS Lambda Versions

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



AWS Lambda Aliases

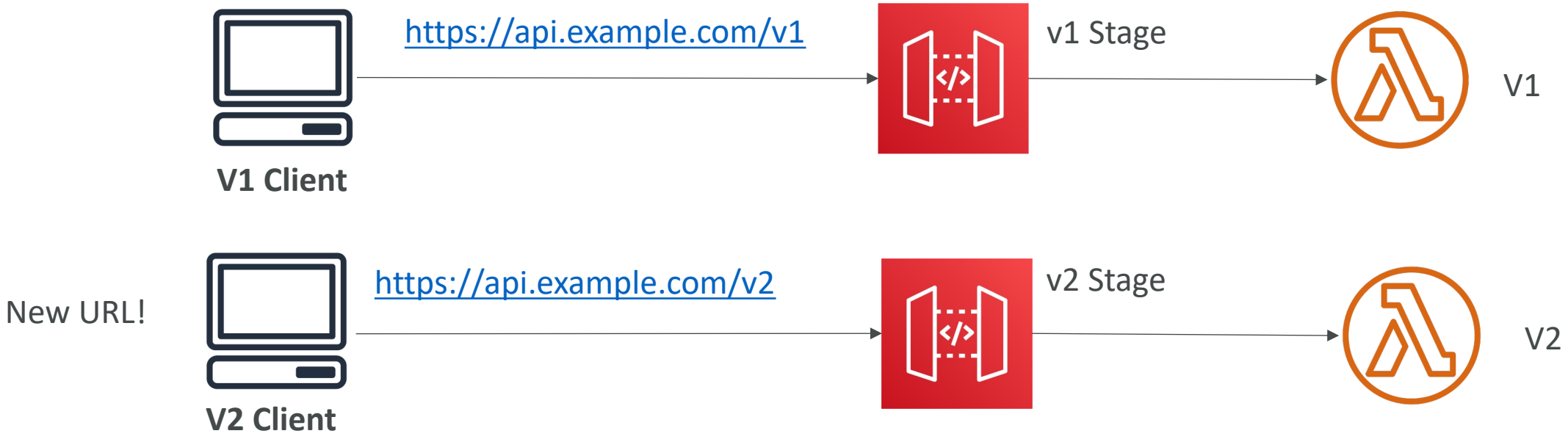
- Aliases are "pointers" to Lambda function versions
- We can define a "dev", "test", "prod" aliases and have them point at different lambda versions
- Aliases are mutable
- Aliases enable Blue / Green deployment by assigning weights to lambda functions
- Aliases enable stable configuration of our event triggers / destinations
- Aliases have their own ARNs



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

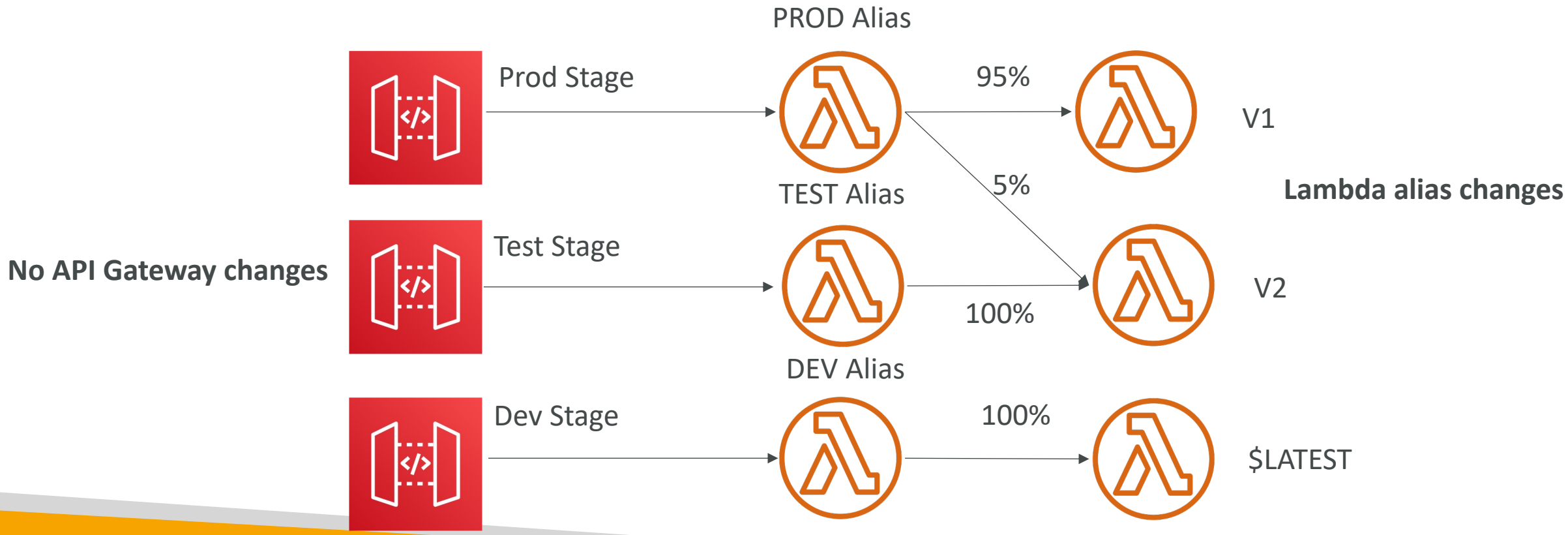


API Gateway – Stage Variables

- 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

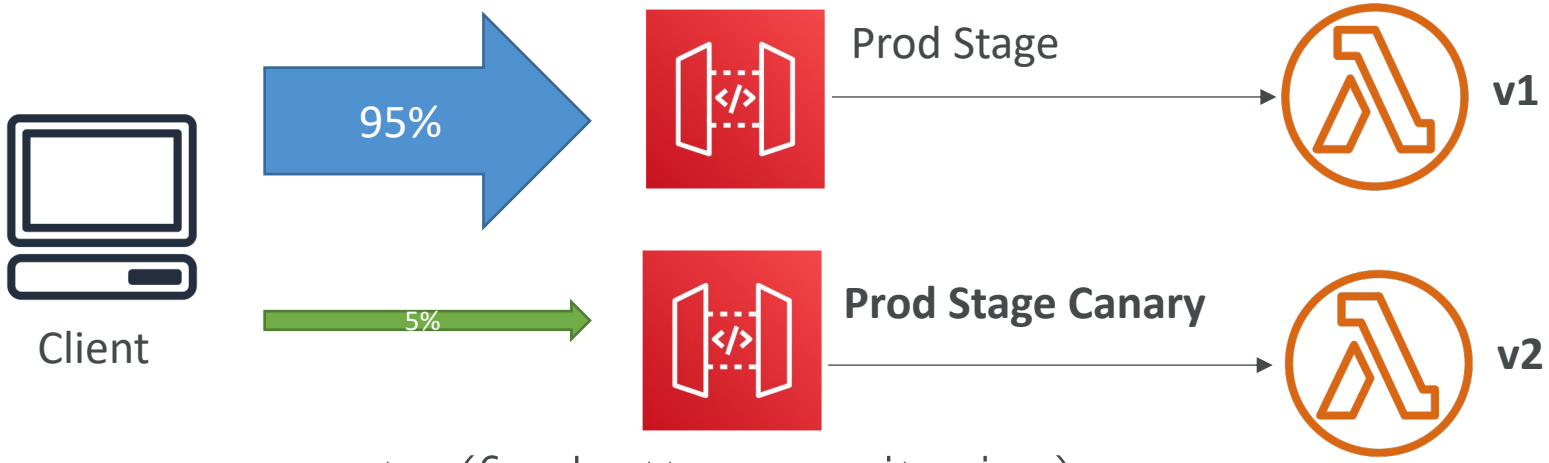
API Gateway Stage Variables & Lambda Aliases

- 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

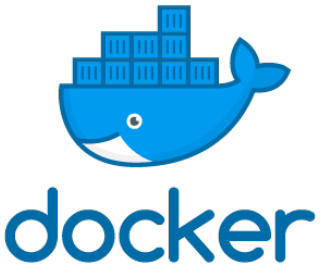


- Metrics & Logs are separate (for better monitoring)
- Possibility to override stage variables for canary
- This is blue / green deployment with AWS Lambda & API Gateway

AWS Step Functions – When to Use?

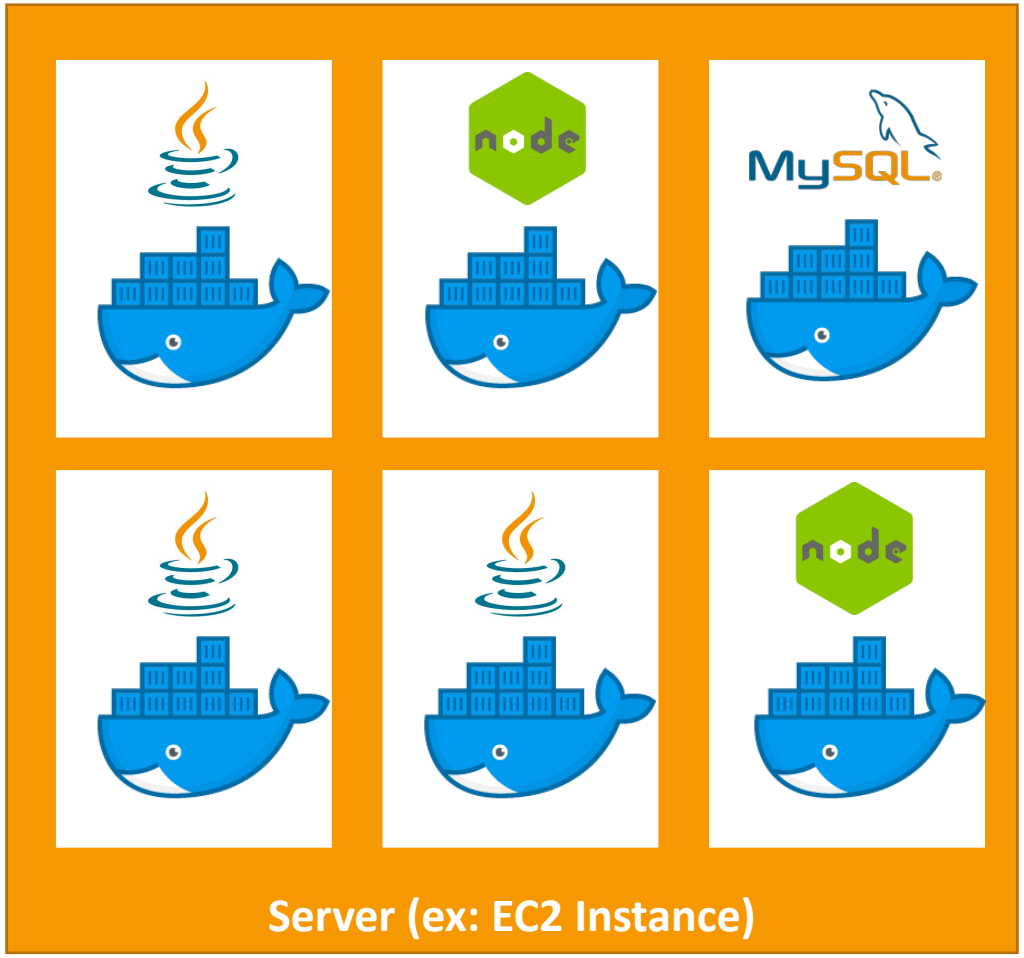
- Use to design workflows
- Easy visualizations
- Advanced Error Handling and Retry mechanism outside the code
- Audit of the history of workflows
- Ability to "Wait" for an arbitrary amount of time
- Max execution time of a State Machine is 1 year
- Example:
  - Payment Workflow
  - Complex flows
  - Long running workflows (days) to go over the Lambda limit of 15 minutes

What is Docker?



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

Docker on an OS



Where Docker images are stored?

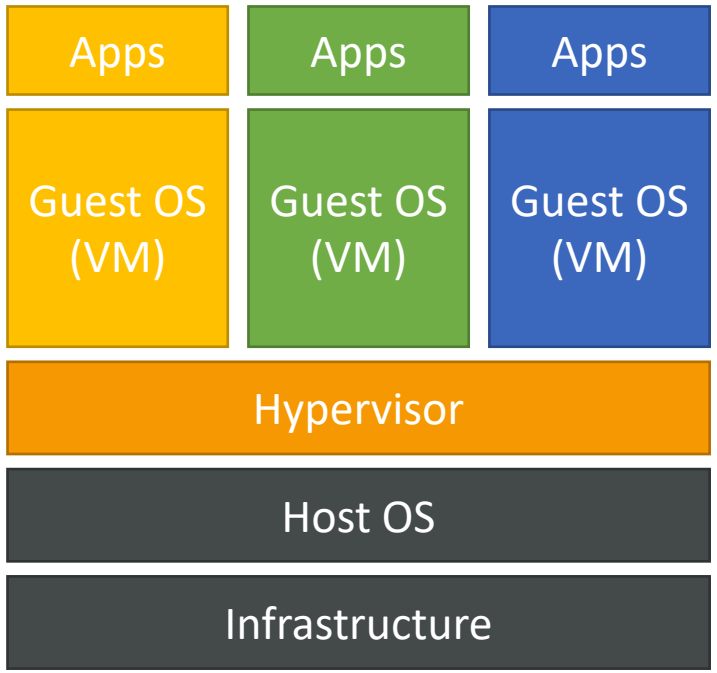
- Docker images are stored in Docker Repositories

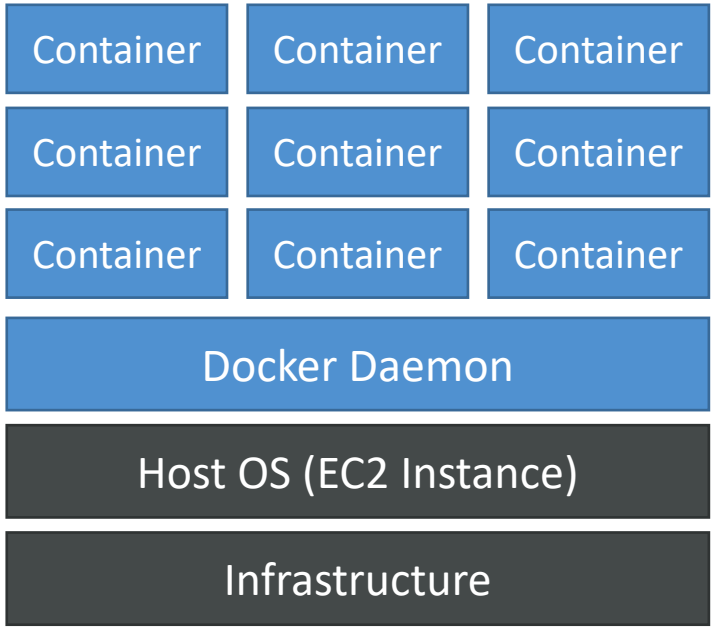
- Public: Docker Hub https://hub.docker.com/
  - Find base images for many technologies or OS:
  - Ubuntu
  - MySQL
  - NodeJS, Java…

- Private: Amazon ECR (Elastic Container Registry)

Docker versus Virtual Machines

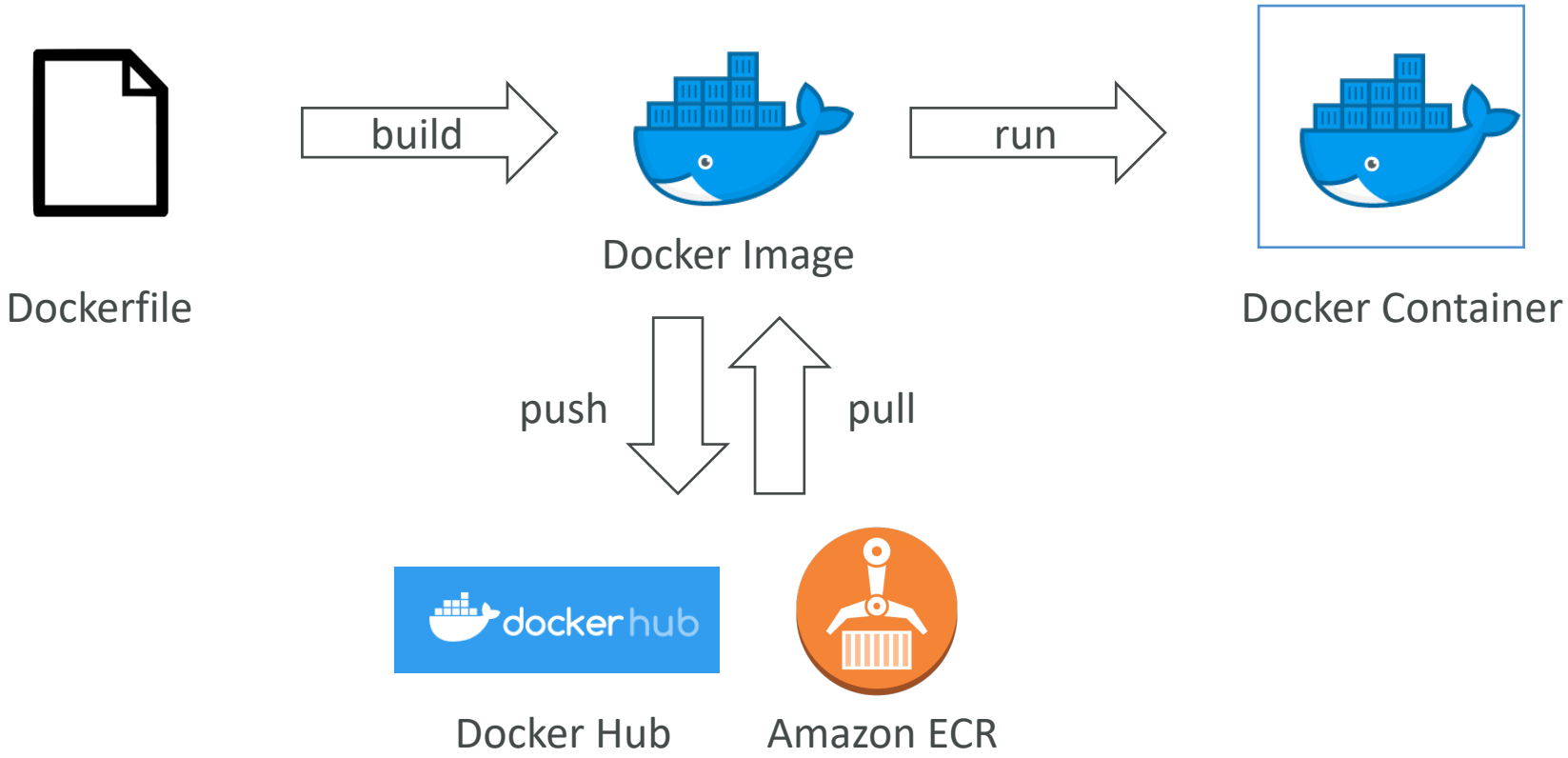
- Docker is "sort of" a virtualization technology, but not exactly
- Resources are shared with the host => many containers on one server





Getting Started with Docker

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



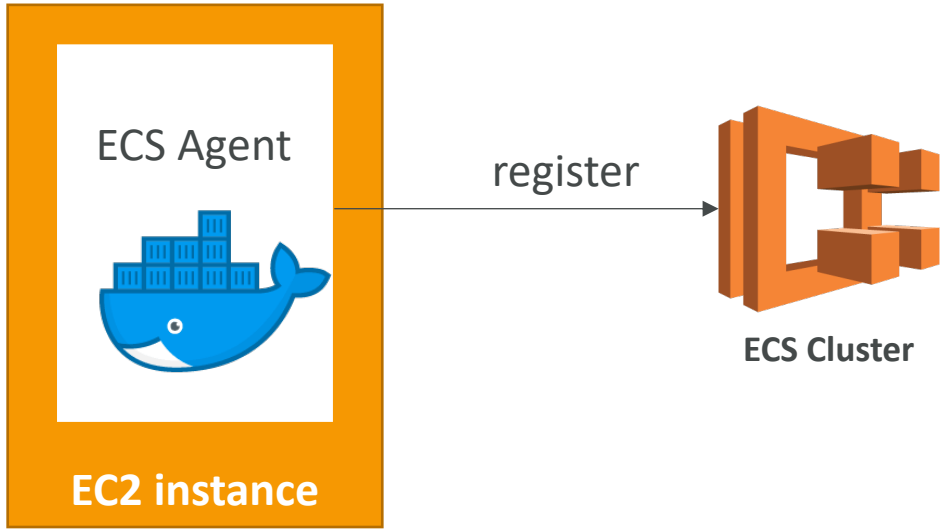
Docker Containers Management

- To manage containers, we need a container management platform

- Three choices:
- ECS: Amazon's own platform
- Fargate: Amazon's own Serverless platform
- EKS: Amazon's managed Kubernetes (open source)

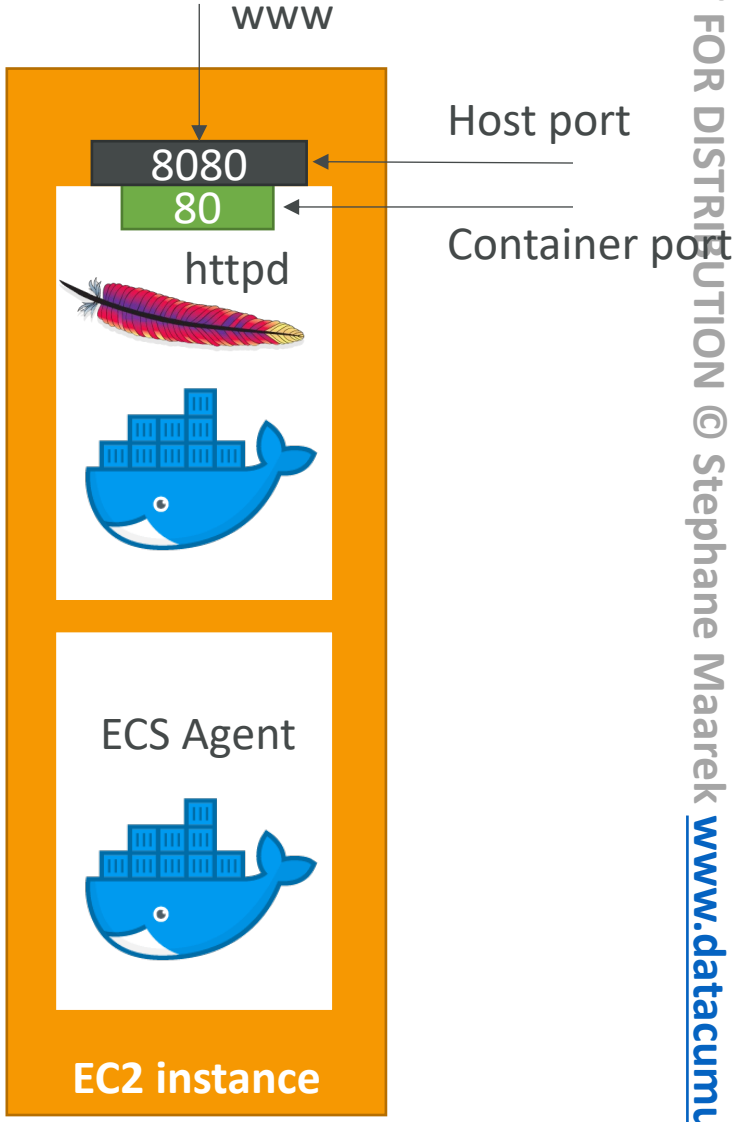
ECS Clusters Overview

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



ECS Task Definitions

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

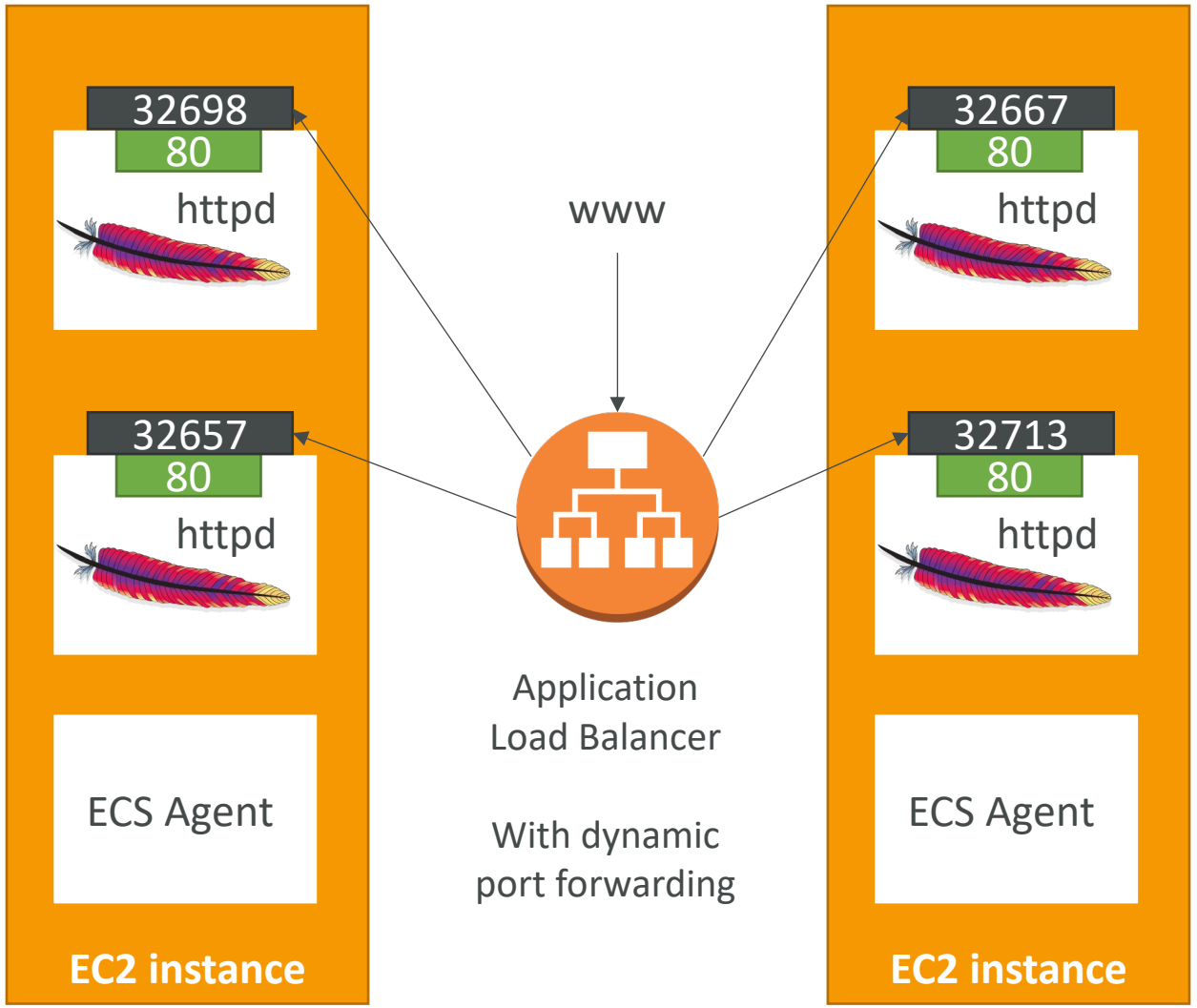


ECS Service

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

- Let's make our first service!

ECS Service with Load Balancer



ECR

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

Fargate

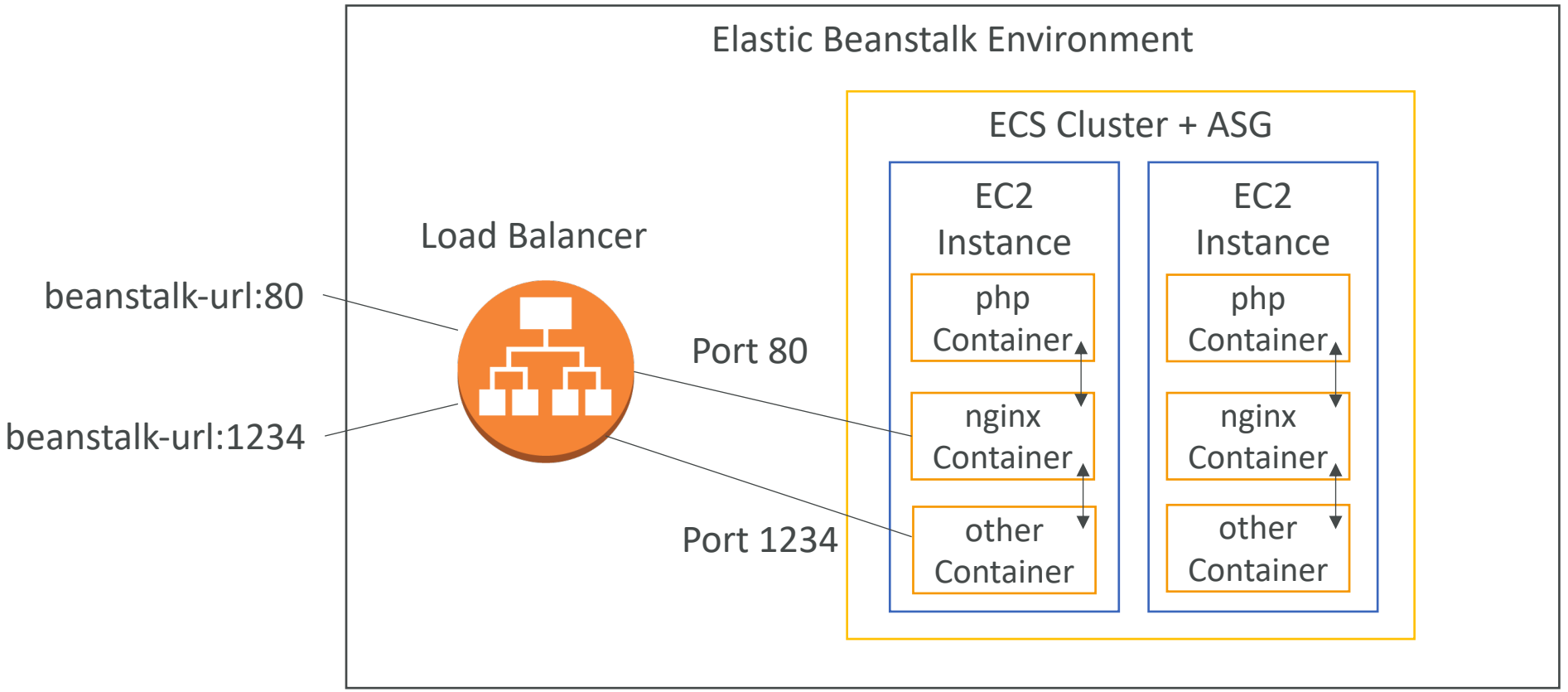
- When launching an ECS Cluster, we have to create our EC2 instances
- If we need to scale, we need to add EC2 instances
- So we manage infrastructure…

- With Fargate, it's all Serverless!
- We don't provision EC2 instances
- We just create task definitions, and AWS will run our containers for us
- To scale, just increase the task number. Simple! No more EC2 ☺

Elastic Beanstalk + ECS

- You can run Elastic Beanstalk in Single & Multi Docker Container mode
- Multi Docker helps run multiple containers per EC2 instance in EB
- This will create for you:
  - ECS Cluster
  - EC2 instances, configured to use the ECS Cluster
  - Load Balancer (in high availability mode)
  - Task definitions and execution
- Requires a config file **Dockerrun.aws.json** at the root of source code
- Your Docker images must be pre-built and stored in ECR for example

Elastic Beanstalk + ECS

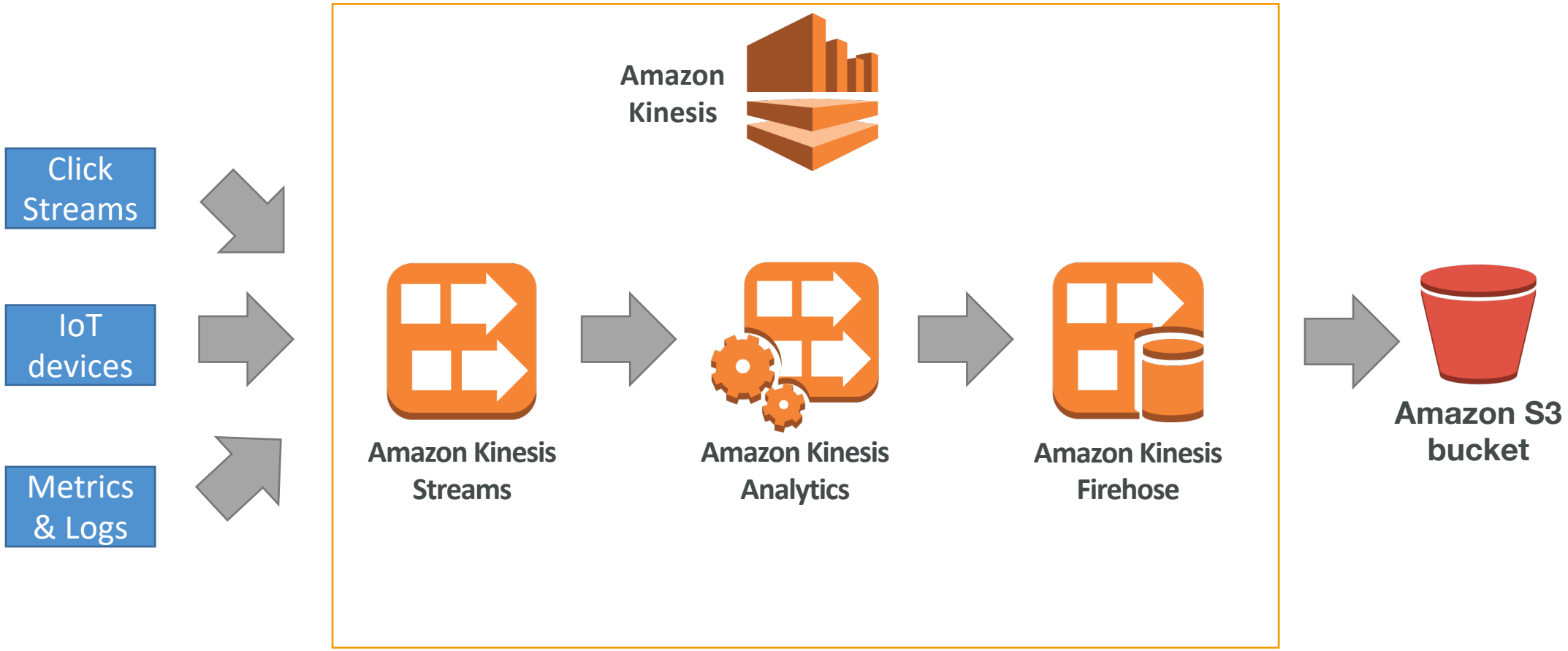


AWS Kinesis Overview

- **Kinesis** is a managed alternative to Apache Kafka
- Great for application logs, metrics, IoT, clickstreams
- Great for "real-time" big data
- Great for streaming processing frameworks (Spark, NiFi, etc…)
- Data is automatically replicated to 3 AZ

- **Kinesis Streams:** low latency streaming ingest at scale
- **Kinesis Analytics:** perform real-time analytics on streams using SQL
- **Kinesis Firehose:** load streams into S3, Redshift, ElasticSearch…

Kinesis Example



Kinesis Streams Overview

- Streams are divided in ordered Shards / Partitions

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

Kinesis Streams Shards

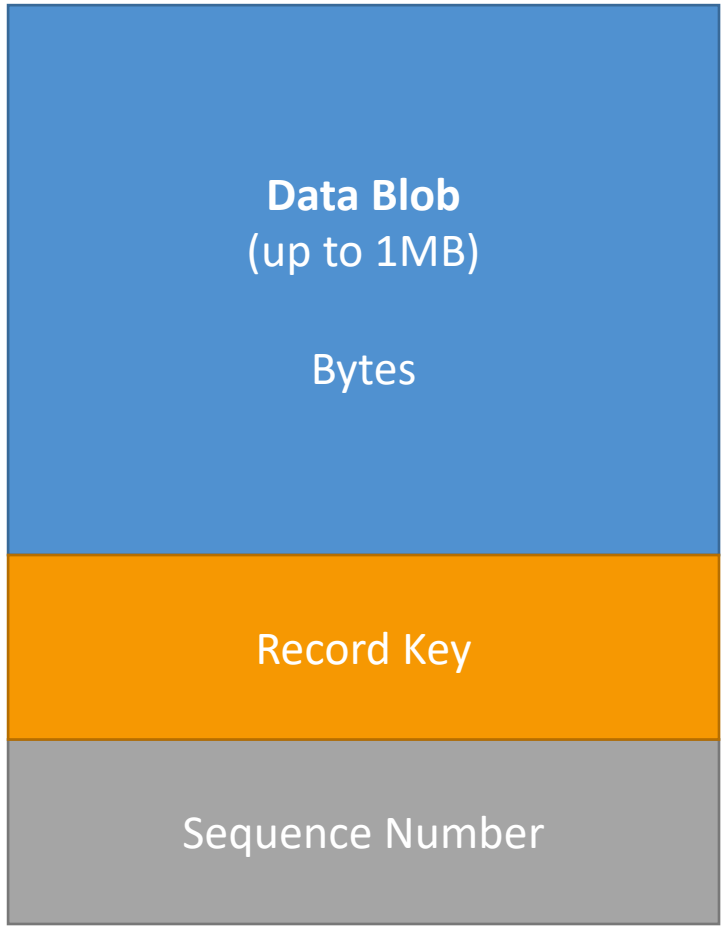
- One stream is made of many different shards
- Billing is per shard provisioned, can have as many shards as you want
- Batching available or per message calls.
- The number of shards can evolve over time (reshard / merge)
- **Records are ordered per shard**

Kinesis Streams Records

- Data Blob: data being sent, serialized as **bytes**. Up to 1 MB. Can represent anything

- Record Key:
  - sent alongside a record, helps to group records in Shards. Same key = Same shard.
  - Use a highly distributed key to avoid the "hot partition" problem

- Sequence number: Unique identifier for each records put in shards. Added by Kinesis after ingestion



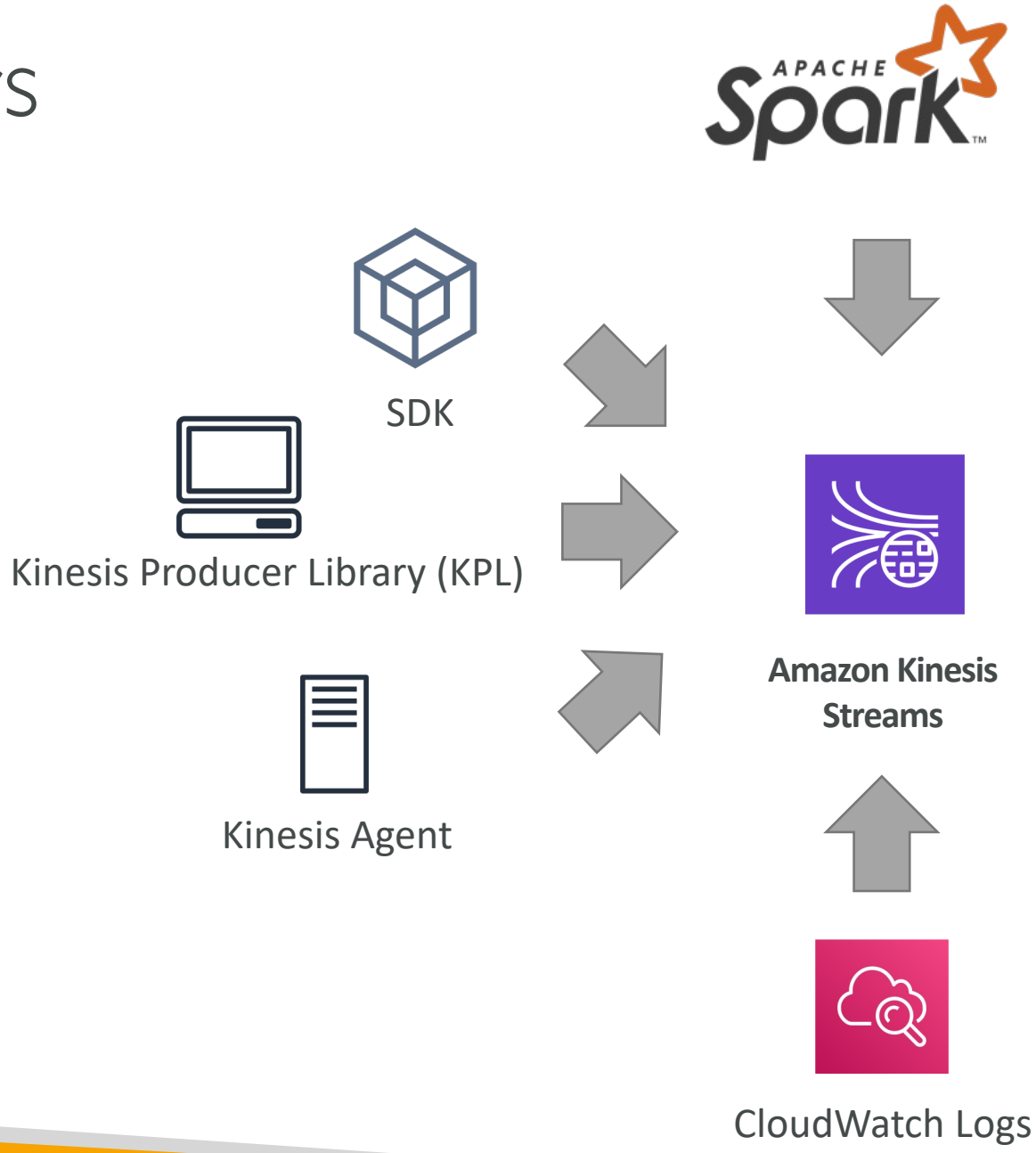
Kinesis Data Streams Limits to know

- Producer:
  - 1MB/s or 1000 messages/s at write PER SHARD
  - "ProvisionedThroughputException" otherwise
- Consumer Classic:
  - **2MB/s at read PER SHARD across all consumers**
  - 5 API calls per second PER SHARD across all consumers
  - **= if 3 different applications are consuming, possibility of throttling**
- Data Retention:
  - 24 hours data retention by default
  - Can be extended to 7 days

Kinesis Producers

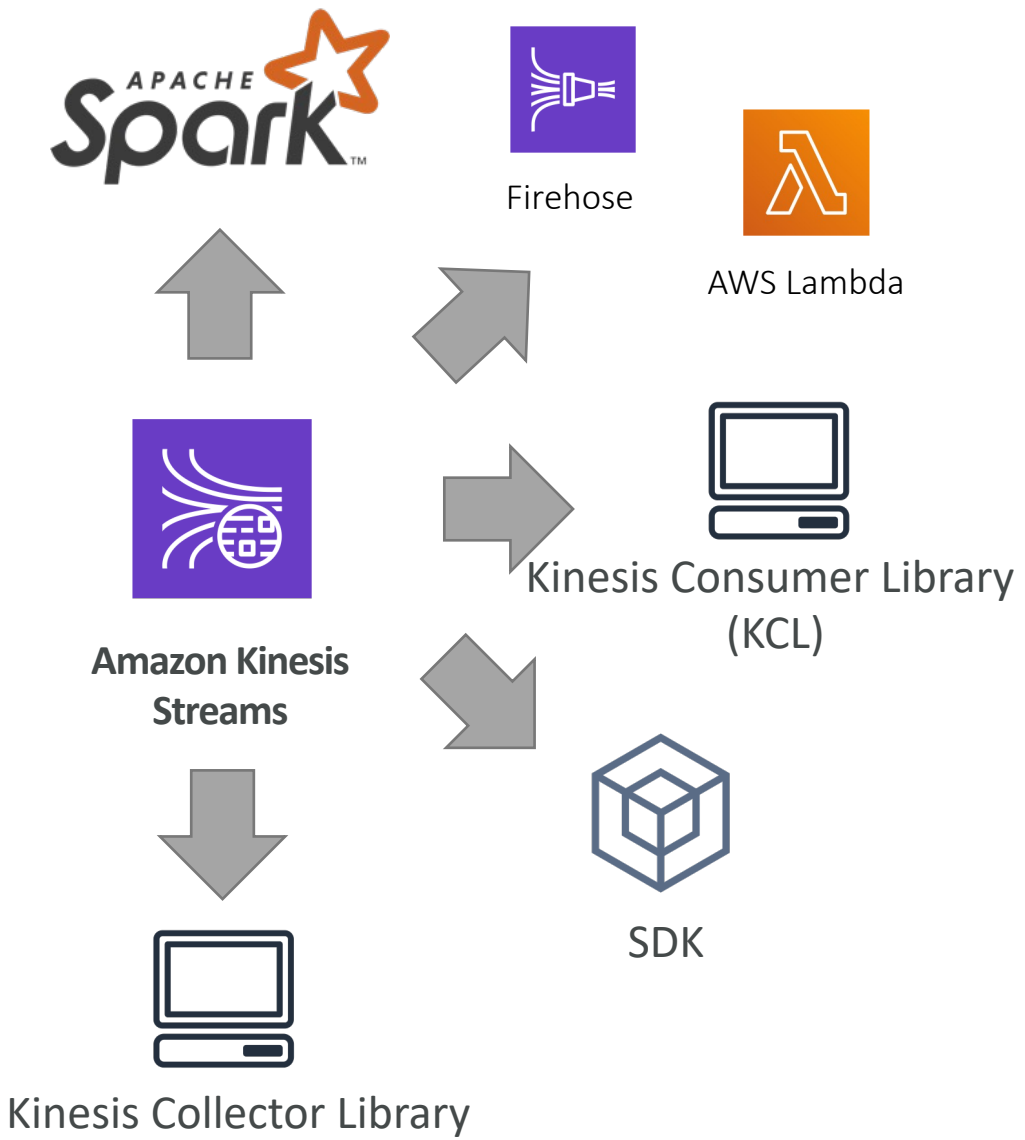
- Kinesis SDK
- Kinesis Producer Library (KPL)
- Kinesis Agent
- CloudWatch Logs

- 3rd party libraries: Spark, Log4J Appenders, Flume, Kafka Connect, NiFi…



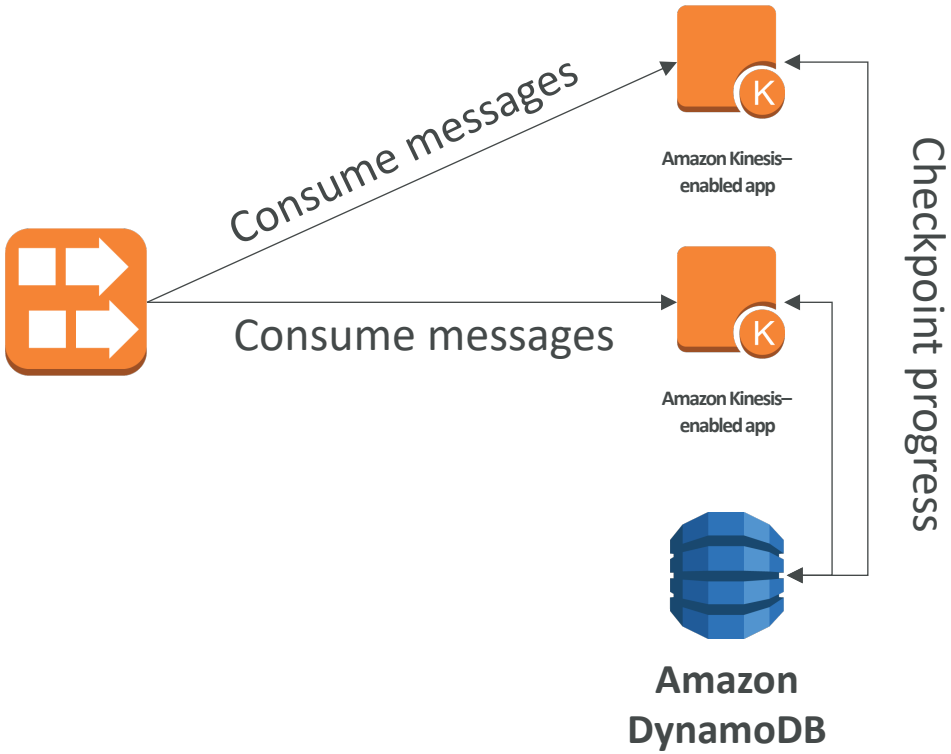
Kinesis Consumers

- Kinesis SDK
- Kinesis Client Library (KCL)
- Kinesis Connector Library
- Kinesis Firehose
- AWS Lambda
- 3rd party libraries: Spark, Log4J Appenders, Flume, Kafka Connect…

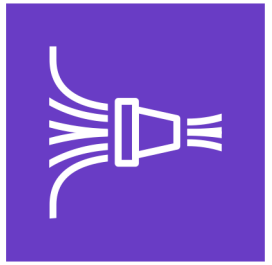


AWS Kinesis KCL

- KCL uses DynamoDB to checkpoint offsets
- KCL uses DynamoDB to track other workers and share the work amongst shards
- Great for reading in a distributed manner

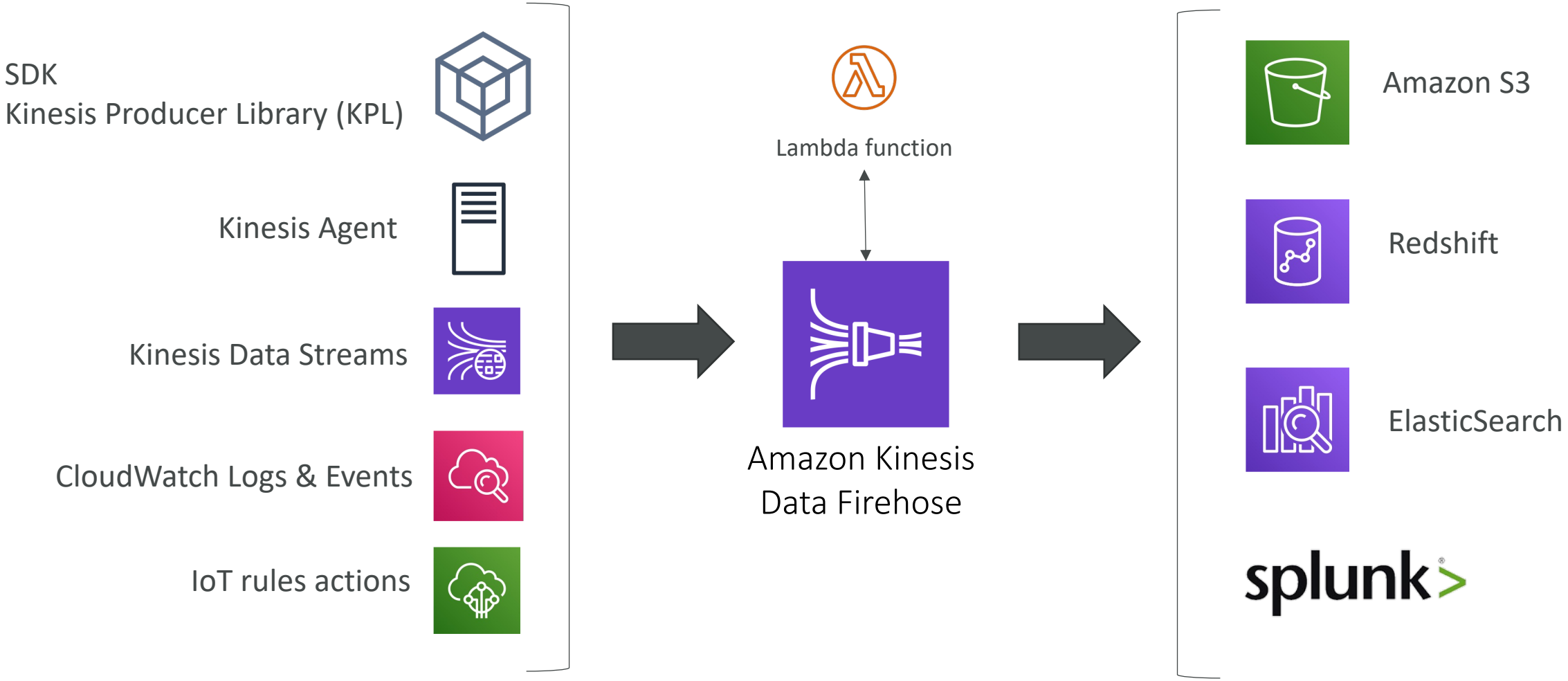


AWS Kinesis Data Firehose



- Fully Managed Service, no administration
- **Near Real Time** (60 seconds latency minimum for non full batches)
- Load data into **Redshift / Amazon S3 / ElasticSearch / Splunk**
- Automatic scaling
- Data Transformation through AWS Lambda (ex: CSV => JSON)
- Supports compression when target is Amazon S3 (GZIP, ZIP, and SNAPPY)
- Pay for the amount of data going through Firehose

Kinesis Data Firehose Diagram



Kinesis Data Streams vs Firehose

- Streams
  - Going to write custom code (producer / consumer)
  - Real time (~200 ms latency for classic)
  - Must manage scaling (shard splitting / merging)
  - Data Storage for 1 to 7 days, replay capability, multi consumers
  - Use with Lambda to insert data in real-time to ElasticSearch (for example)

- Firehose
  - Fully managed, send to S3, Splunk, Redshift, ElasticSearch
  - Serverless data transformations with Lambda
  - **Near** real time (lowest buffer time is 1 minute)
  - Automated Scaling
  - No data storage

AWS Kinesis Data Analytics

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

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

Amazon ElasticSearch

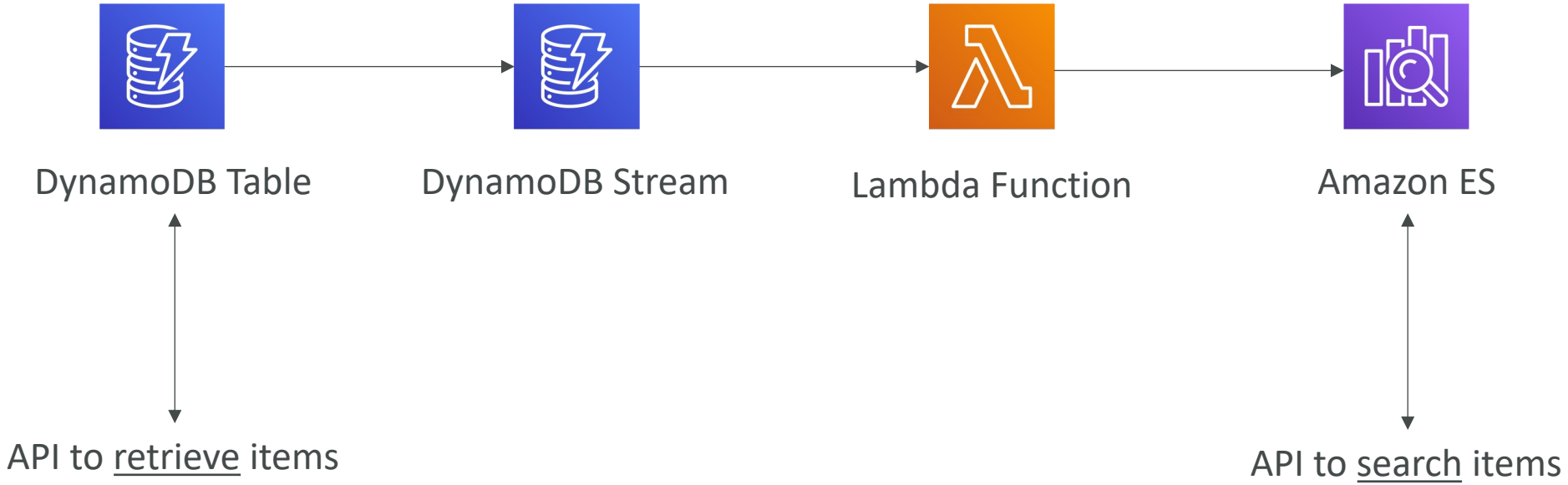
- May be called Amazon ES at the exam

- Managed version of ElasticSearch (open source project)
- Needs to run on servers (not a serverless offering)
- Use cases:
  - Log Analytics
  - Real Time application monitoring
  - Security Analytics
  - Full Text Search
  - Clickstream Analytics
  - Indexing

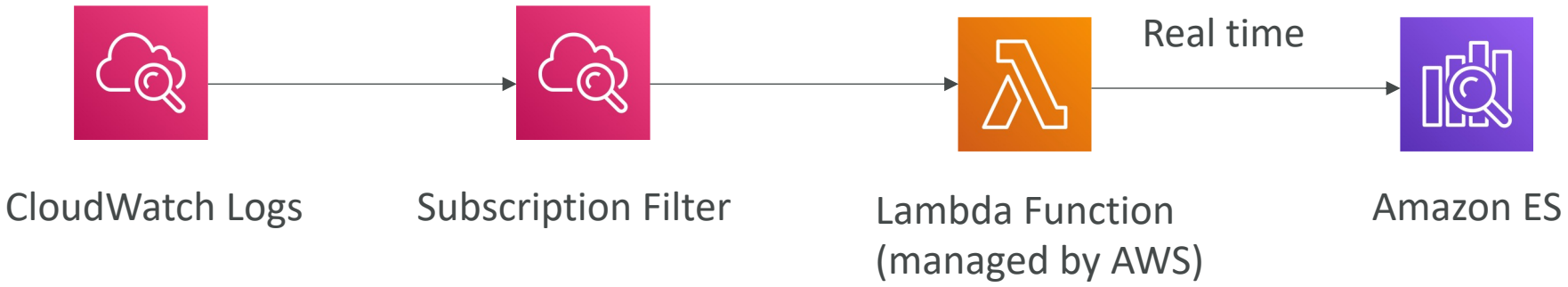
ElasticSearch + Kibana + Logstash

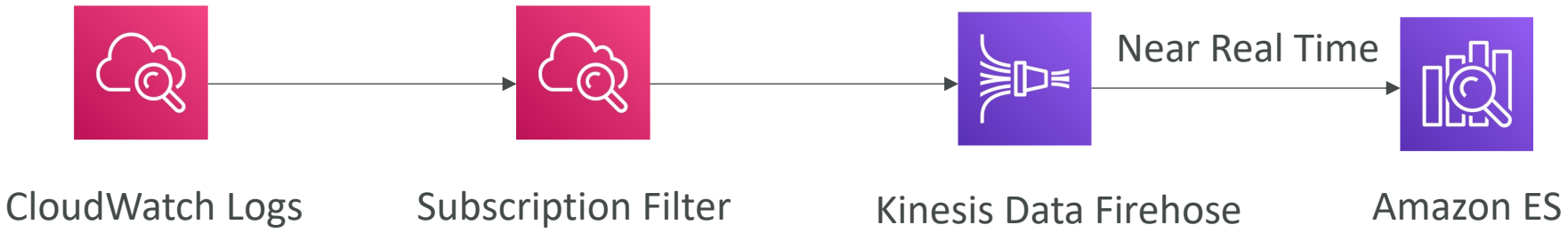
- ElasticSearch: provide search and indexing capability
  - You must specify instance types, multi-AZ, etc
- Kibana:
  - Provide real-time dashboards on top of the data that sits in ES
  - Alternative to CloudWatch dashboards (more advanced capabilities)
- Logstash:
  - Log ingestion mechanism, use the "Logstash Agent"
  - Alternative to CloudWatch Logs (you decide on retention and granularity)

Elastic Search patterns DynamoDB

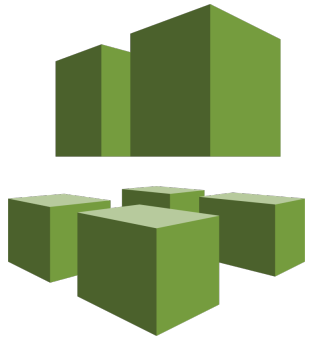


Elastic Search patterns
CloudWatch Logs





AWS Systems Manager Overview



- Helps you manage your **EC2 and On-Premise** 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

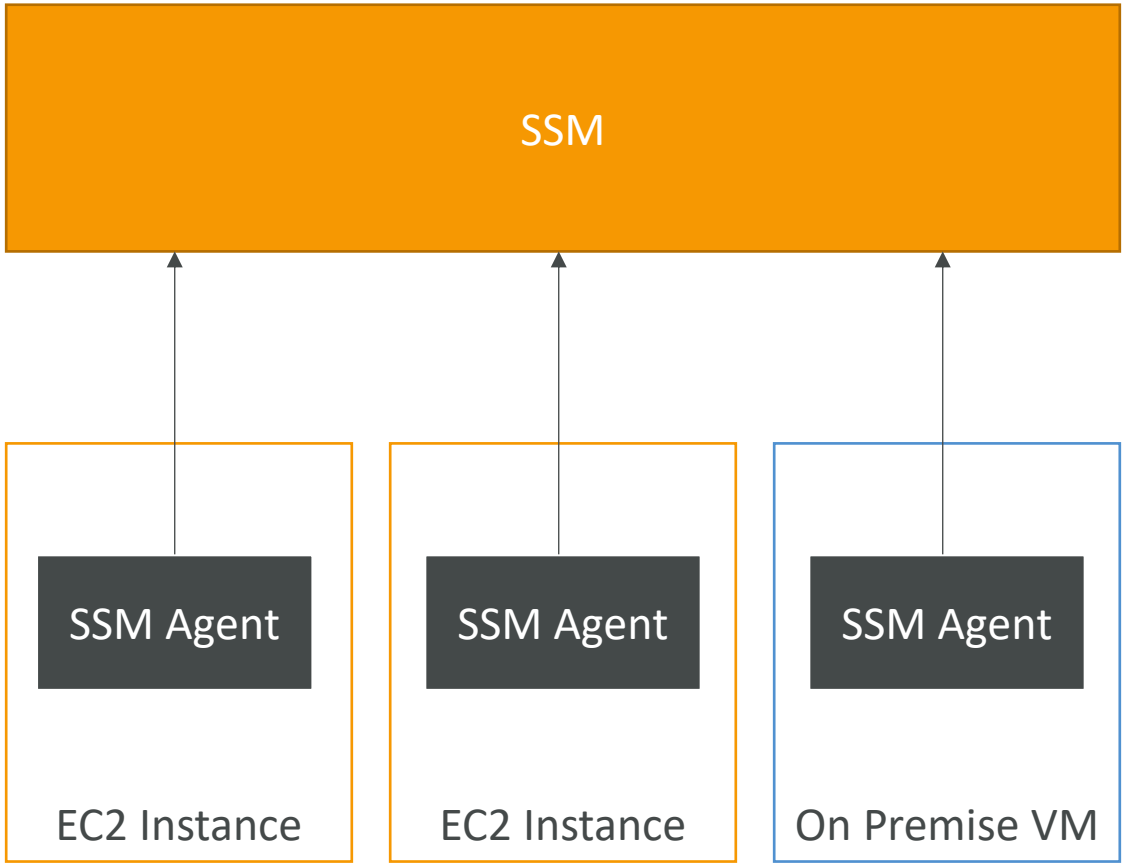
- Resource Groups
- Insights:
  - Insights Dashboard
  - Inventory: discover and audit the software installed
  - Compliance
- Parameter Store

Action:

- Automation (shut down EC2, create AMIs)
- Run Command
- Session Manager
- Patch Manager
- Maintenance Windows
- State Manager: define and maintaining configuration of OS and applications

How Systems Manager works

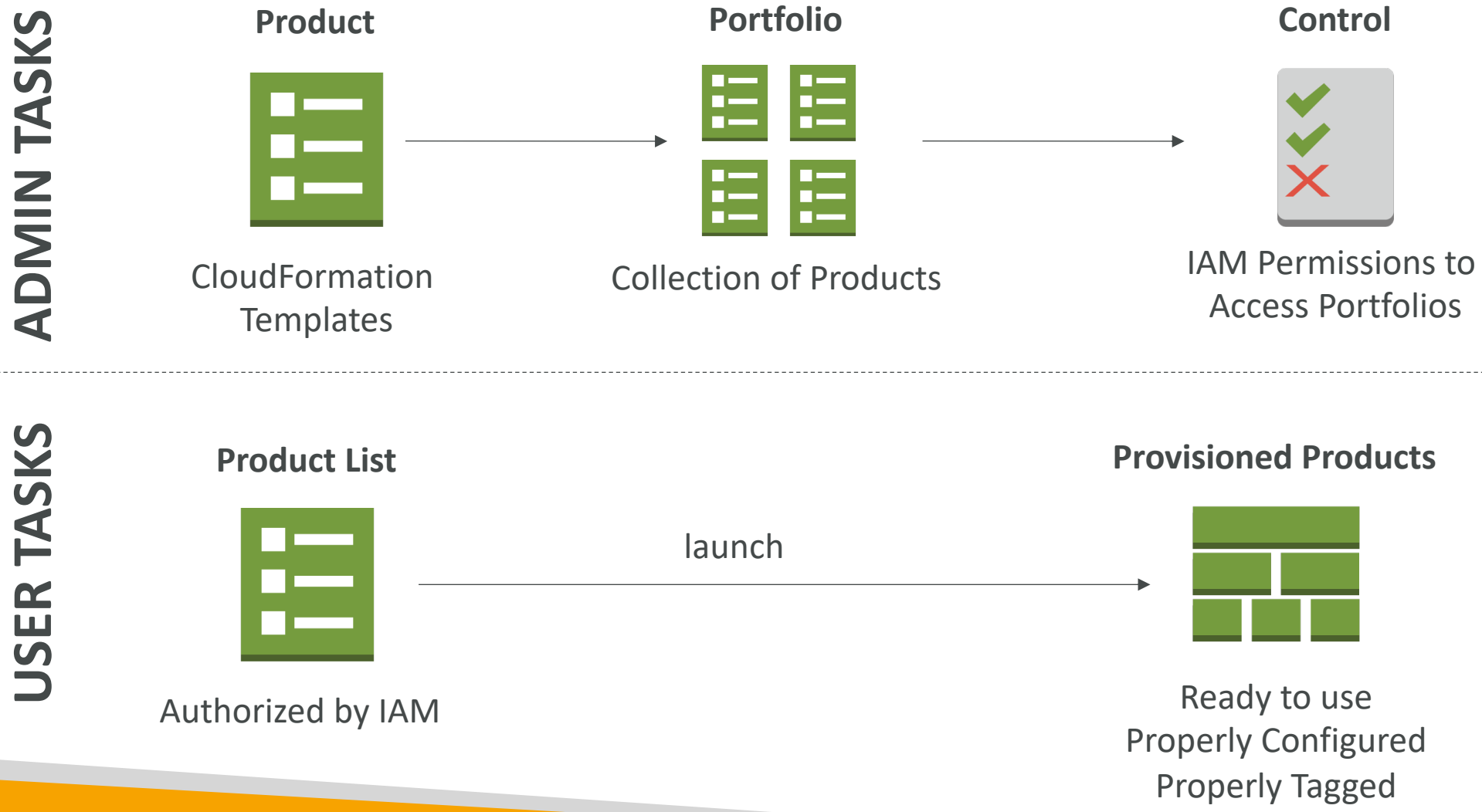
- We need to install the SSM agent onto the systems we control
- Installed by default on Amazon Linux AMI & some Ubuntu AMI
- If an instance can't be controlled with SSM, it's probably an issue with the SSM agent!
- Make sure the EC2 instances have a proper IAM role to allow SSM actions



AWS 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



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

EC2 Instance Compliance

- AWS Config
  - ensure instance has proper AWS configuration (not open SSH port, etc)
  - Audit and compliance over time
- Inspector
  - Security Vulnerabilities scan from within the OS using the agent
  - Or outside network scanning (no need for the agent)
- Systems Manager
  - Run automations, patches, commands, inventory at scale
- Service Catalog
  - Restrict how the EC2 instances can be launched to minimize configurations
  - Helpful to onboard beginner AWS users
- Configuration Management
  - SSM, Opsworks, Ansible, Chef, Puppet, User Data
  - Ensure the EC2 instances have proper configuration files

GuardDuty

- Intelligent Threat discovery to Protect AWS Account
- Uses Machine Learning algorithms, anomaly detection, 3rd party data
- One click to enable (30 days trial), no need to install software

- Input data includes:
  - CloudTrail Logs: unusual API calls, unauthorized deployments
  - VPC Flow Logs: unusual internal traffic, unusual IP address
  - DNS Logs: compromised EC2 instances sending encoded data within DNS queries

- Notifies you in case of findings
- Integration with AWS Lambda

AWS Cost Allocation Tags

- With Tags we can track resources that relate to each other
- With Cost Allocation Tags we can enable detailed costing reports
- **Just like Tags, but they show up as columns in Reports**
- AWS Generated Cost Allocation Tags
  - Automatically applied to the resource you create
  - Starts with Prefix **aws:** **(e.g. aws: createdBy)**
  - They're not applied to resources created before the activation
- User tags
  - Defined by the user
  - Starts with Prefix **user:**
- **Cost Allocation Tags just appear in the Billing Console**
- Takes up to 24 hours for the tags to show up in the report

AWS Data Protection

- TLS for in transit encryption
- ACM to manage SSL / TLS certificates
- Load Balancers
  - ELB, ALB & NLB provide SSL termination
  - Possible to have multiple SSL certificates per ALB
  - Optional SSL/TLS encryption between ALB and EC2 instances (else, HTTP)
- CloudFront with SSL
- All AWS services expose HTTPS endpoints
- You *could* (but *shouldn't*) use HTTP with S3

AWS Data Protection At Rest Encryption

- S3 encryption
  - SSE-S3: Server Side encryption using AWS' key
  - SSE-KMS: Server Side encryption using your own KMS key
  - SSE-C: Server Side encryption by providing your own key (AWS won't keep it)
  - Client side encryption: send encrypted content to AWS, no knowledge of key
  - Possibility to enable default encryption on S3 through setting
  - Possibility to enforce encryption through S3 bucket policy (x-amz-server-side-encryption)
  - Glacier is encrypted by default
- One quick setting for: EBS, EFS, RDS, ElastiCache, DynamoDB, etc
  - Usually uses either service encryption key or your own KMS key
- Category of data:
  - PHI = protected health information
  - PII = personally-identifying information

AWS Network Protection

- Direct Connect: private, direct connection between site and AWS
- Public internet: use a VPN
  - Site-to-Site VPN supports Internet Protocol security (IPsec) VPN connections (for linking on-premise to the cloud)

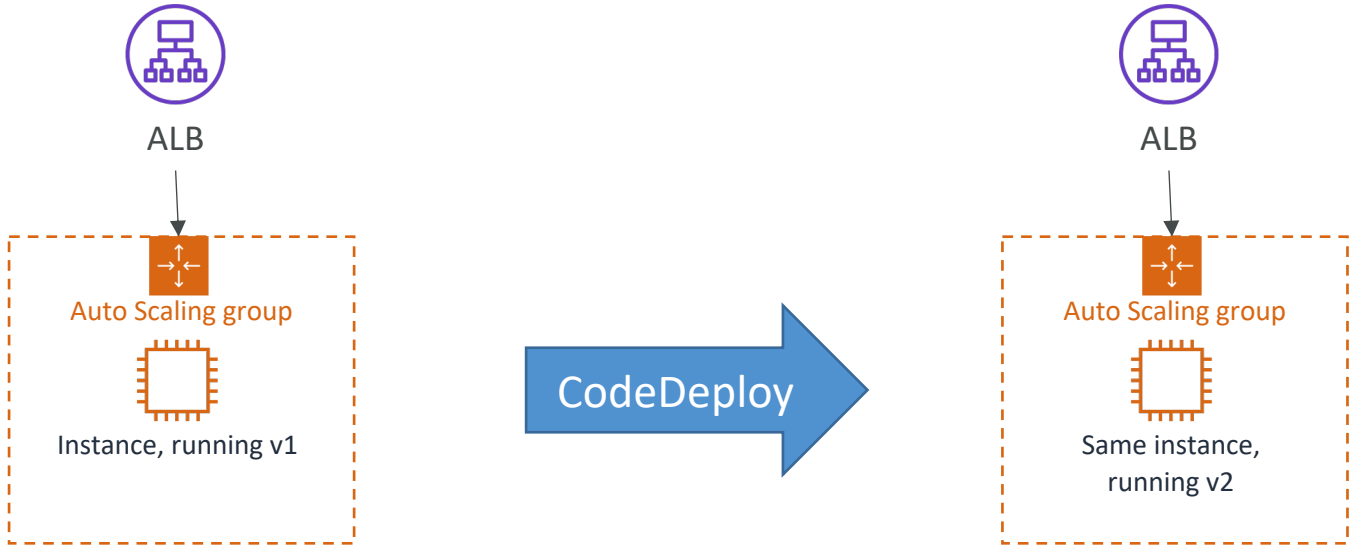
- Network ACL: stateless firewall at the VPC level
- WAF (Web Application Firewall): web security rules against exploits
- Security Groups: stateful firewall on the instance's underlying hypervisor
- System Firewalls: install your own firewall on EC2 instances

Coverage for Domain 5

- Troubleshoot issues and determine how to restore operations
  - CloudWatch, CloudFormation, Rollbacks, etc.
- Determine how to automate event management and alerting + Apply concepts required to set up event-driven automated actions
  - CloudWatch Events+++, CloudWatch Alarms, SNS
- Automated Healing:
  - CloudFormation (triggered by an alarm)
  - Beanstalk (easier)
  - OpsWorks (automatic host replacement, manages the infrastructure)
  - Autoscaling (we'll see in this section)

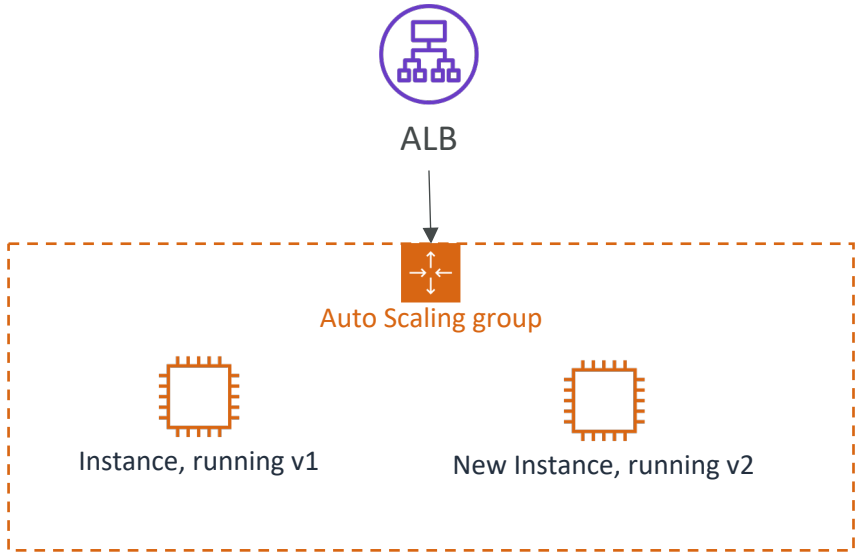
Deployment Strategies
Auto Scaling and ALB

- In place (one LB, one TG, one ASG)



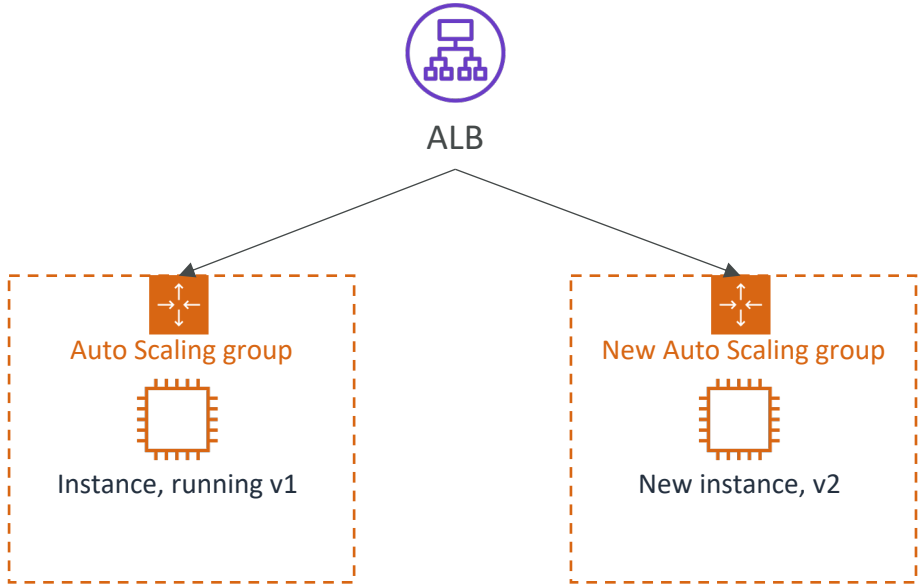
Deployment Strategies
Auto Scaling and ALB

- Rolling (one LB, one TG, one ASG, new instances)



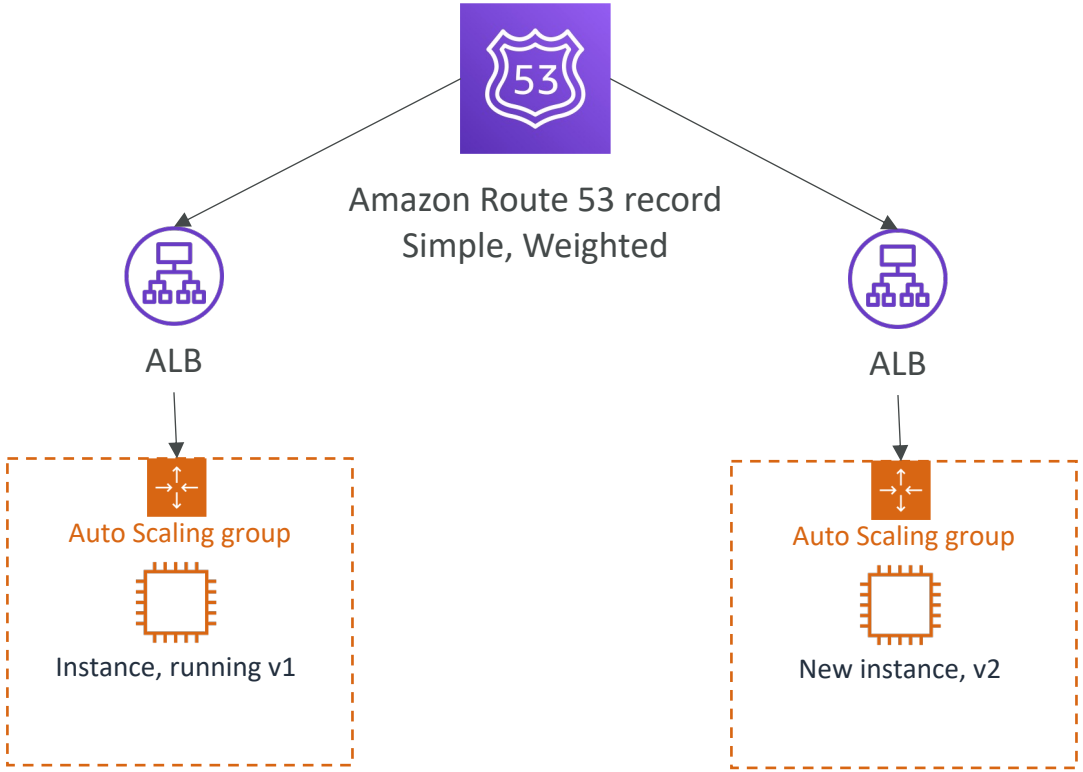
Deployment Strategies
Auto Scaling and ALB

- Replace (one LB, one TG, two ASG, new instances)



Deployment Strategies
Auto Scaling and ALB

- Blue / Green (two LB, two TG, two ASG, new instances, R53)

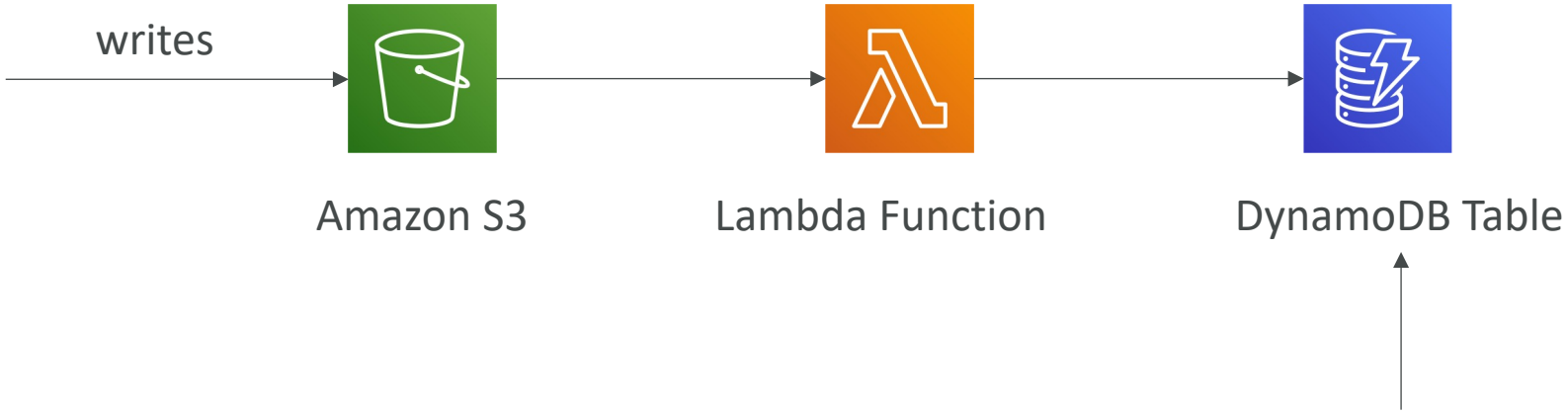


Deployment strategies

- Read more here:

- Blue/Green Deployments on AWS whitepaper, August 2016
  - https://d1.awsstatic.com/whitepapers/AWS_Blue_Green_Deployments.pdf

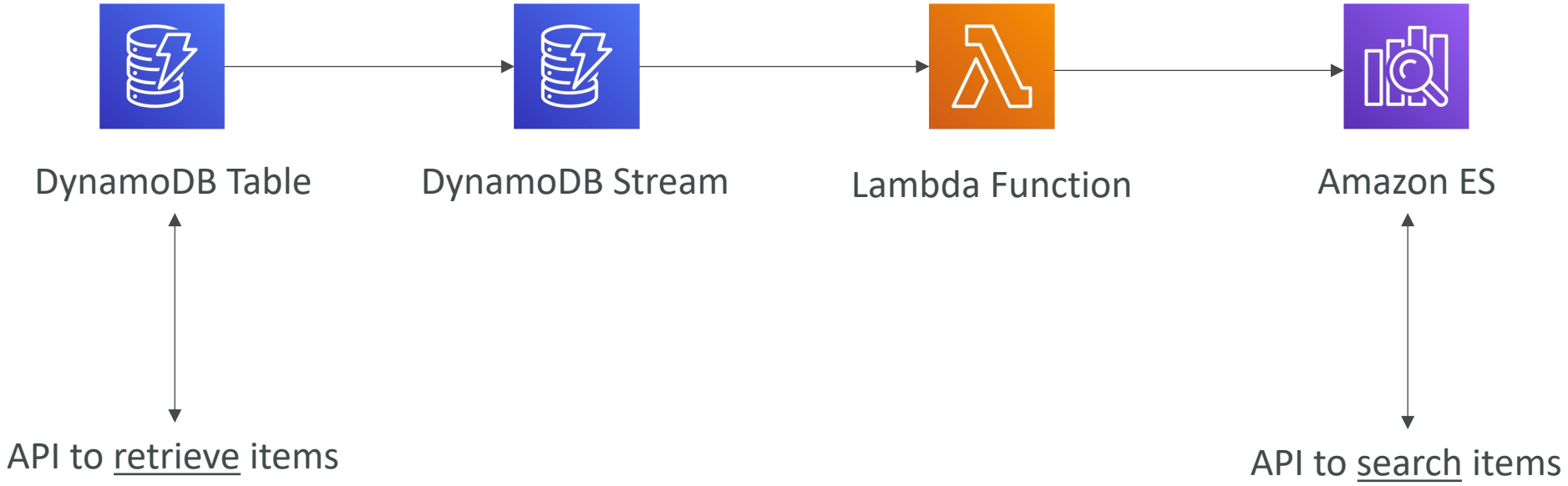
DynamoDB Patterns
S3 Metadata Index



API for object metadata

- Search by date
- Total storage used by a customer
- List of all objects with certain attributes
- Find all objects uploaded within a date range

DynamoDB patterns
Elastic Search



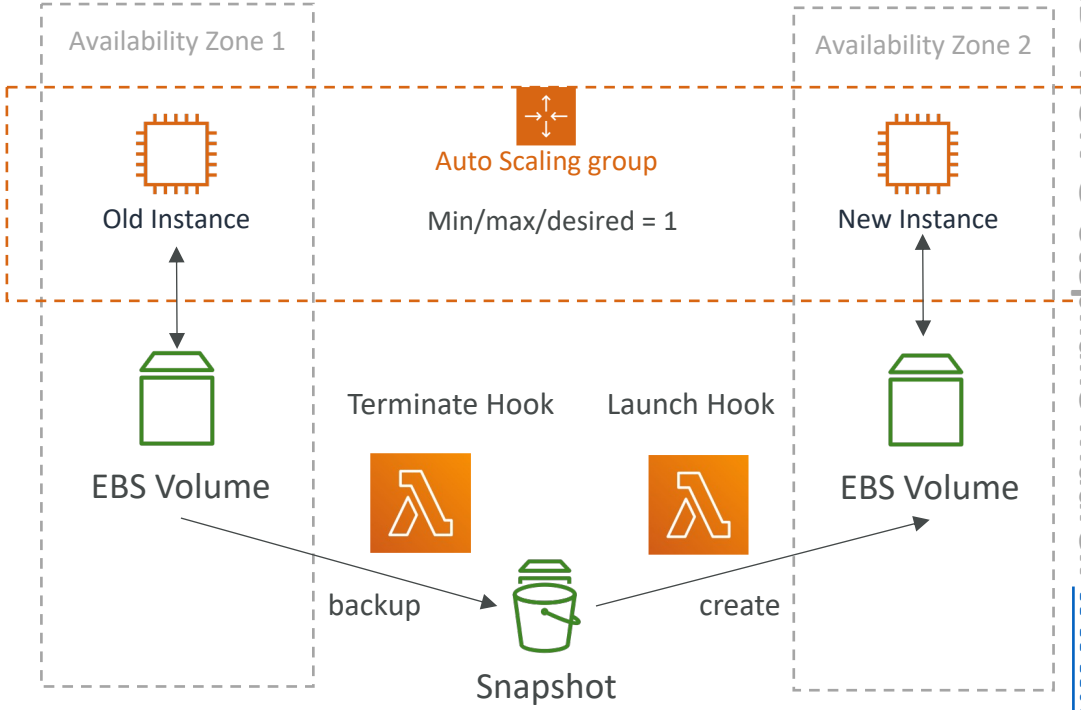
Multi AZ in AWS

- 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)
  - ElasticSearch (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

What about EBS?

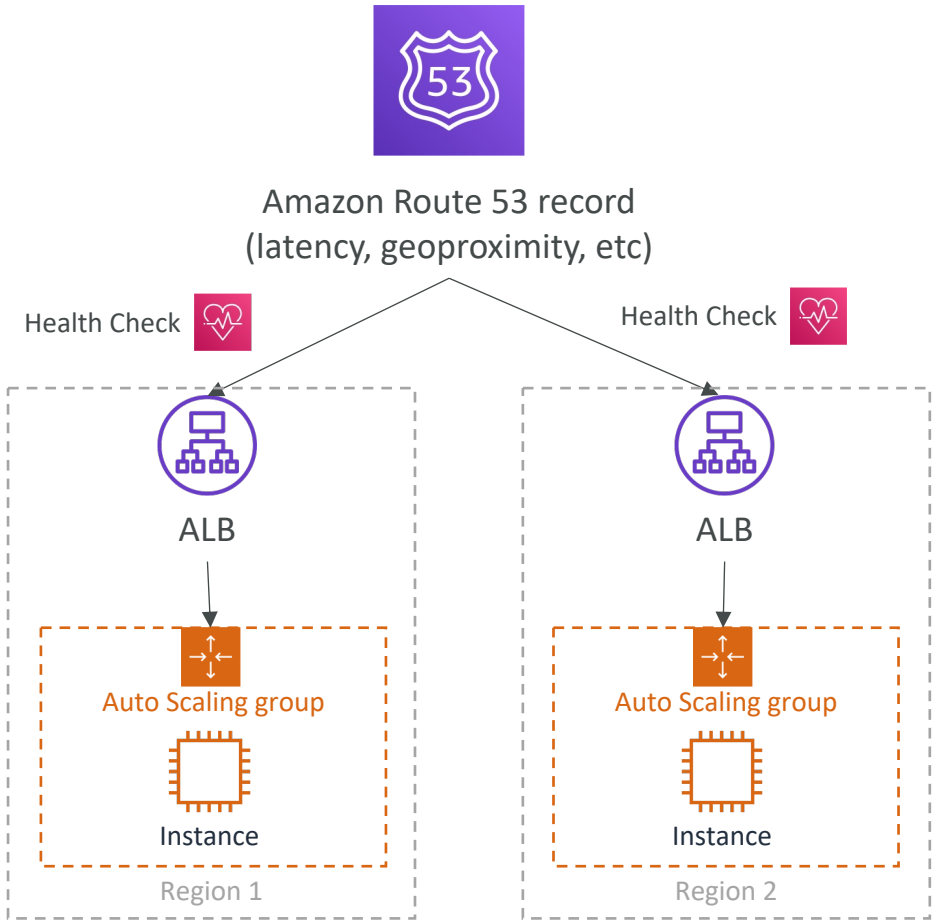
- EBS is tied to a single AZ
- How can you make EBS "multi-AZ" ?
  - ASG with 1 min/max/desired
  - Lifecycle hooks for Terminate: make a snapshot of the EBS volume
  - Lifecycle hook for start: copy the snapshot, create an EBS, attach to instance
- Note: for PIOPS volumes (io1), to get max performance after snapshot, read the entire volume once (pre-warming of IO blocks)



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)
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

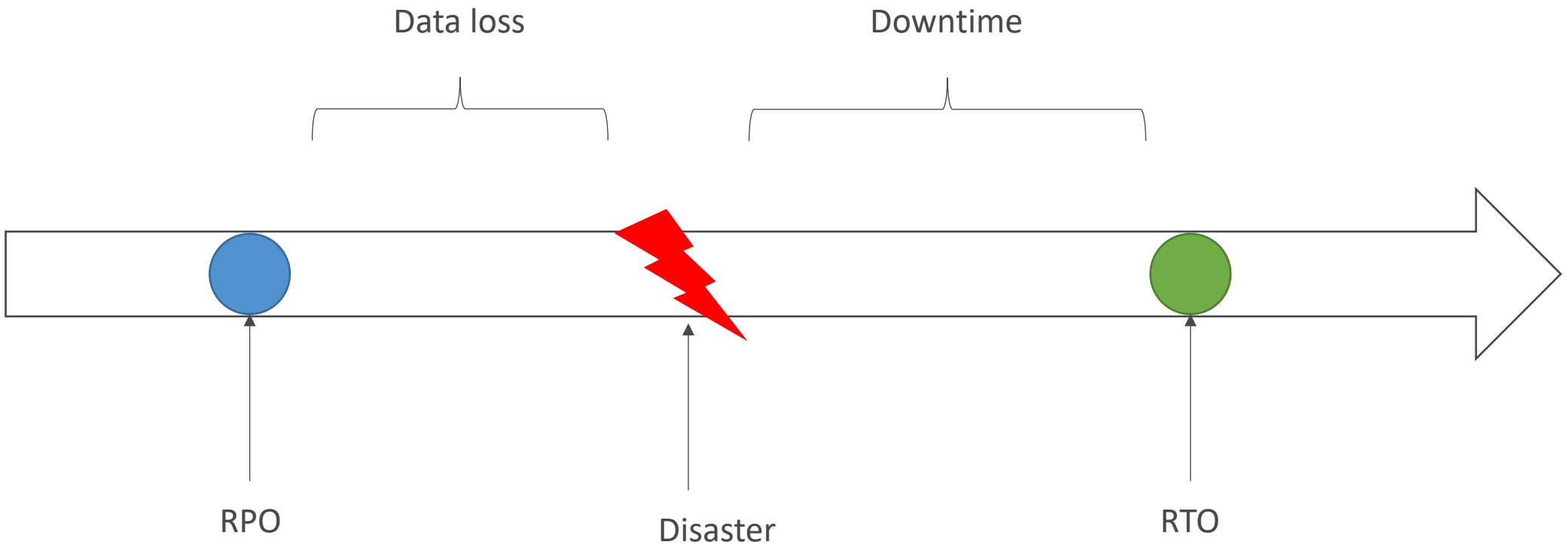
CloudFormation StackSets

- Create, update, or delete stacks across **multiple accounts and regions** with a single operation
- Administrator account to create StackSets
- Trusted 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.
- Ability to set a maximum concurrent actions on targets (# or %)
- Ability to set failure tolerance (# or %)

Disaster Recovery Overview

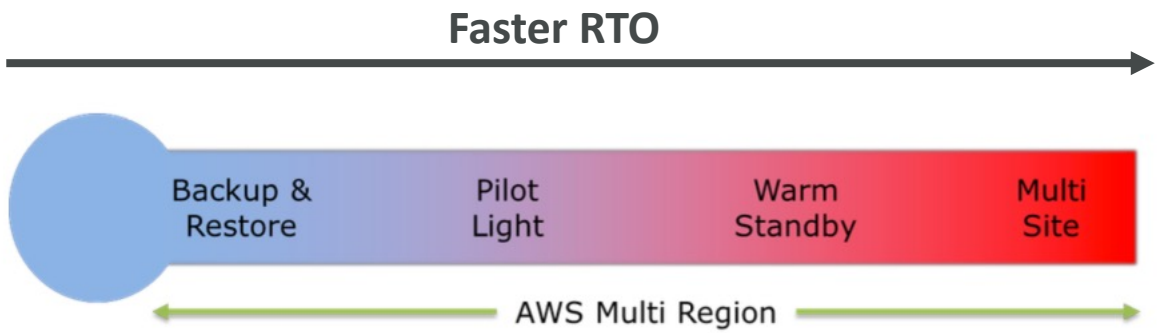
- Any event that has a negative impact on a company's business continuity or finances is a disaster
- Disaster recovery (DR) is about preparing for and recovering from a disaster
- What kind of disaster recovery?
  - On-premise => On-premise: traditional DR, and very expensive
  - On-premise => AWS Cloud: 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

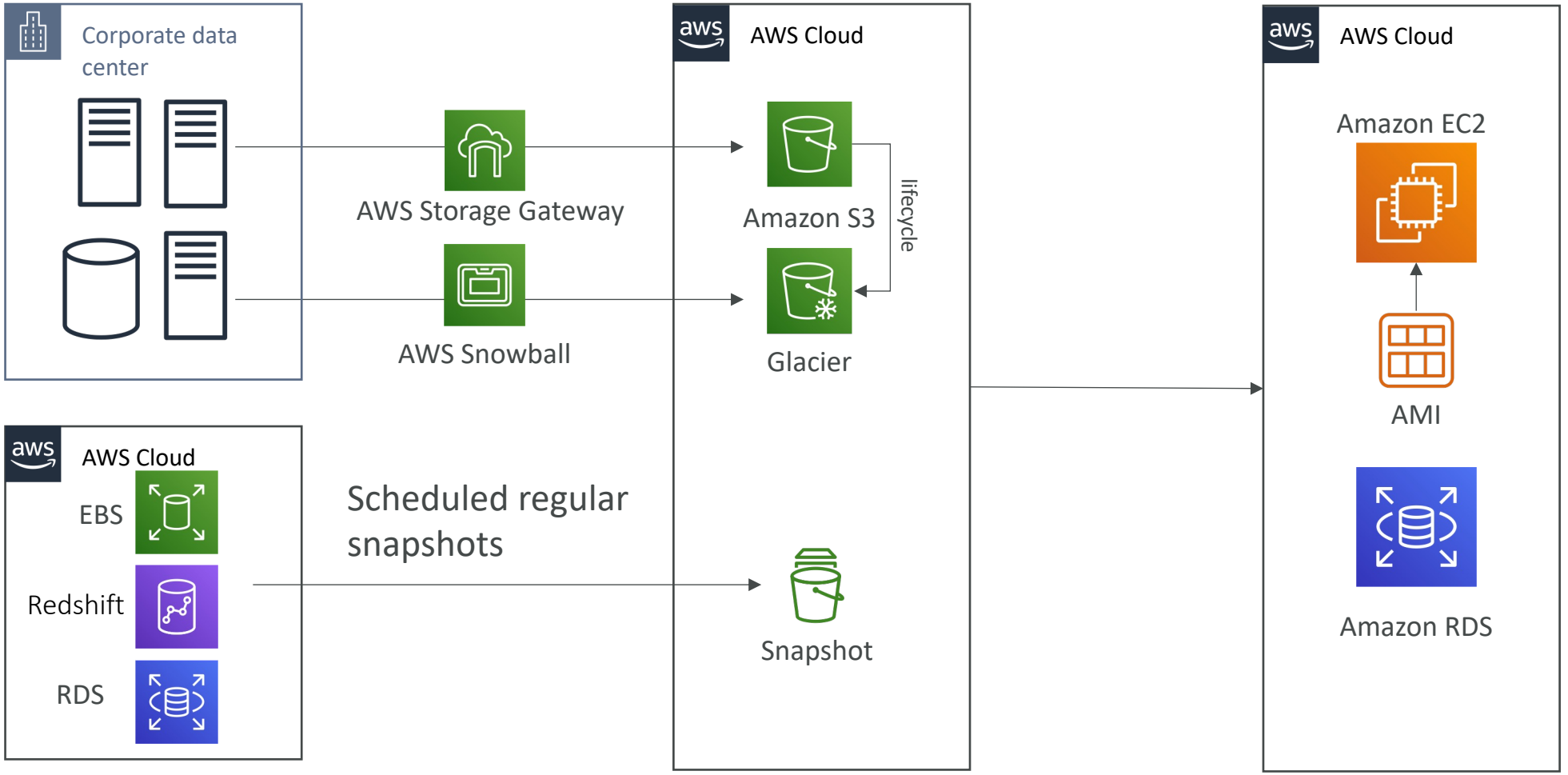


Disaster Recovery Strategies

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

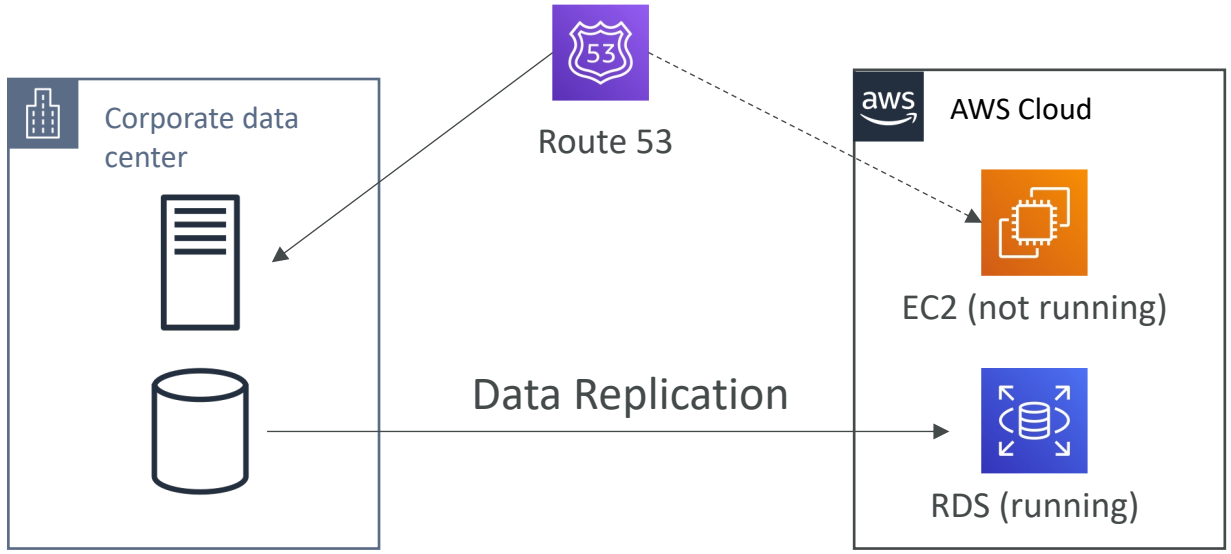


Backup and Restore (High RPO)



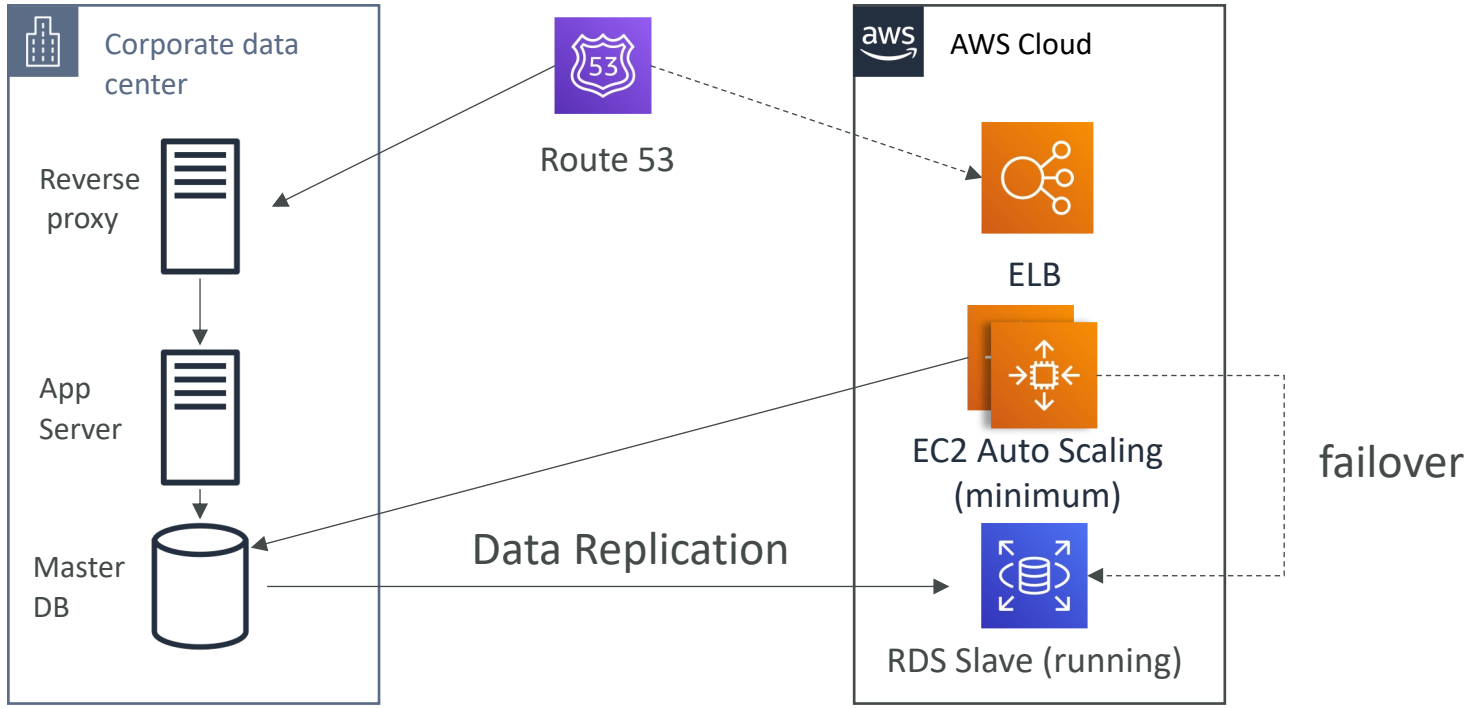
Disaster Recovery – Pilot Light

- 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



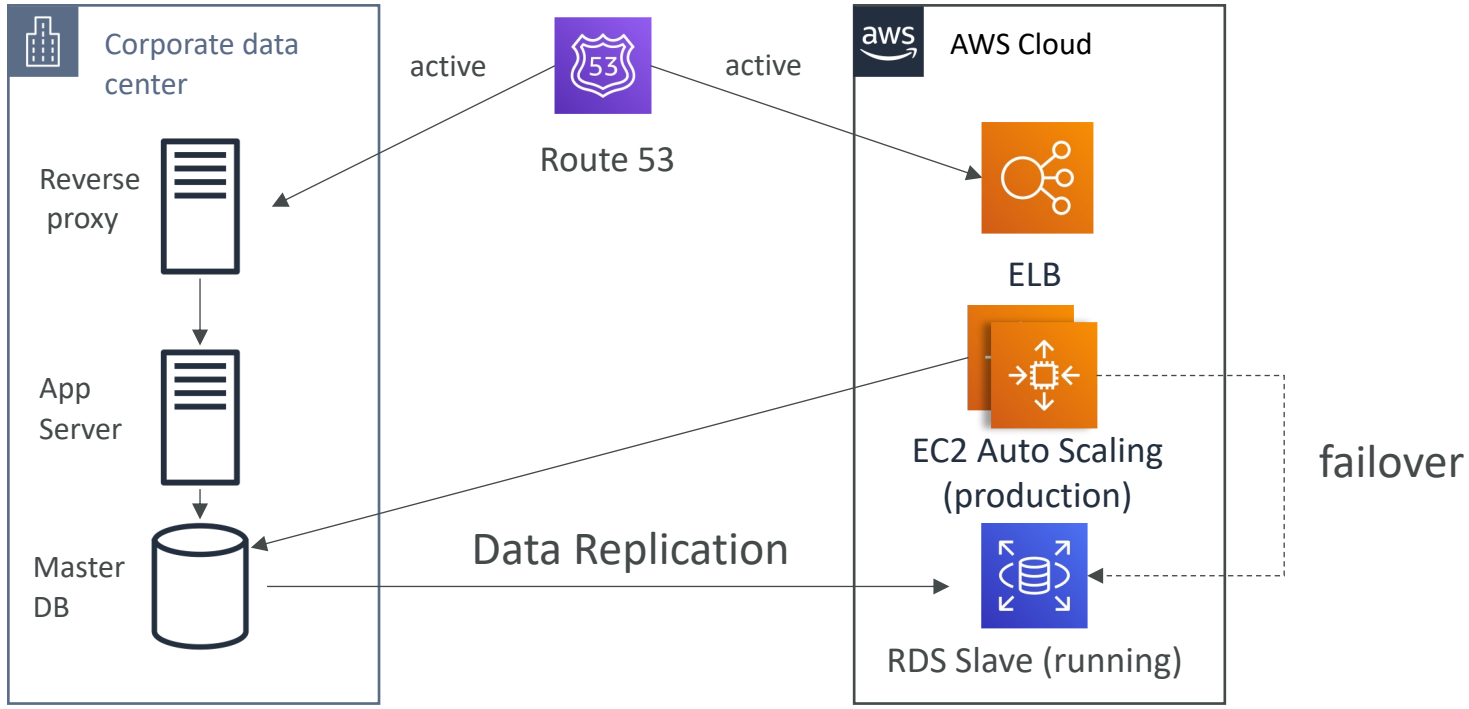
Warm Standby

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

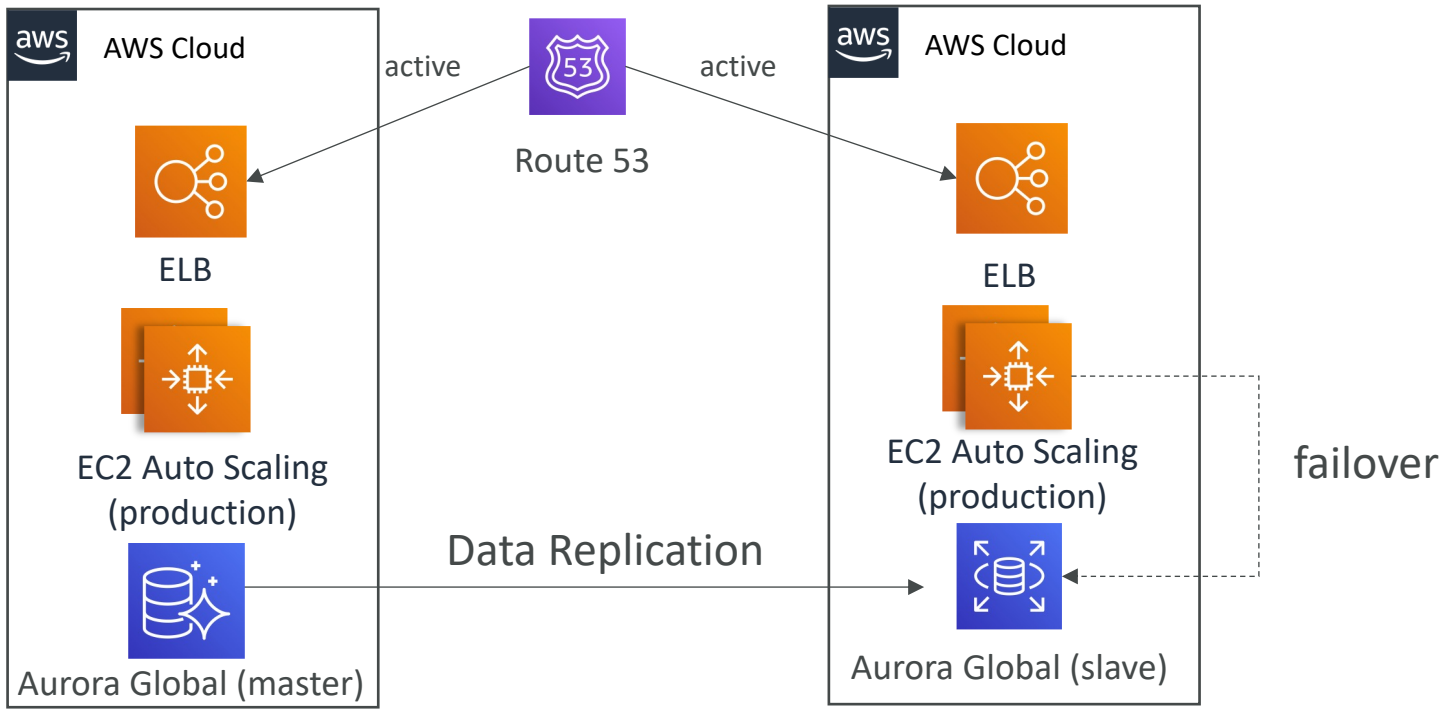


Multi Site / Hot Site Approach

- 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

Multi-Region Disaster Recovery Checklist

- Is my AMI copied? Is it stored in the parameter store?
- Is my CloudFormation StackSet working and tested to work in another region ?
- What's my RPO and RTO?
- Are Route53 Health Checks working correctly? Tied to a CW Alarm?
- How can I automate with CloudWatch Events to Trigger some Lambda functions and perform a RDS Read Replication promotion ?
- Is my data backed up? RPO & RTO? EBS, AMI, RDS, S3 CRR, Global DynamoDB Tables, RDS & Aurora Global Read Replicas

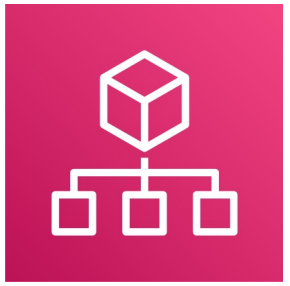
Backups & Multi-Region DR

- EFS Backup:
  - AWS Backup with EFS (frequency, when, retain time, lifecycle policy) - managed
  - EFS to EFS backup https://aws.amazon.com/solutions/efs-to-efs-backup-solution/
  - Multi-region idea: EFS => S3 => S3 CRR => EFS
- Route 53 Backup:
  - Use **ListResourceRecordSets** API for exports
  - Write your own script for imports into R53 or other DNS provider
- Elastic Beanstalk Backup:
  - Saved configurations using the eb cli or AWS console

On-Premise strategy with AWS

- Ability to download Amazon Linux 2 AMI as a VM (.iso format)
  - VMWare, KVM, VirtualBox (Oracle VM), Microsoft Hyper-V
- VM Import / Export
  - Migrate existing applications into EC2
  - Create a DR repository strategy for your on-premise VMs
  - Can export back the VMs from EC2 to on-premise
- AWS Application Discovery Service
  - Gather information about your on-premise servers to plan a migration
  - Server utilization and dependency mappings
  - Track with AWS Migration Hub
- AWS Database Migration Service (DMS)
  - replicate On-premise => AWS , AWS => AWS, AWS => On-premise
  - Works with various database technologies (Oracle, MySQL, DynamoDB, etc..)
- AWS Server Migration Service (SMS)
  - Incremental replication of on-premise live servers to AWS

AWS Organizations



- Global service
- Allows to manage multiple AWS accounts
- The main account is the master account – you can't change it
- Other accounts are member accounts
- Member accounts can only be part of one organization
- Consolidated Billing across all accounts - single payment method
- Pricing benefits from aggregated usage (volume discount for EC2, S3…)
- API is available to automate AWS account creation

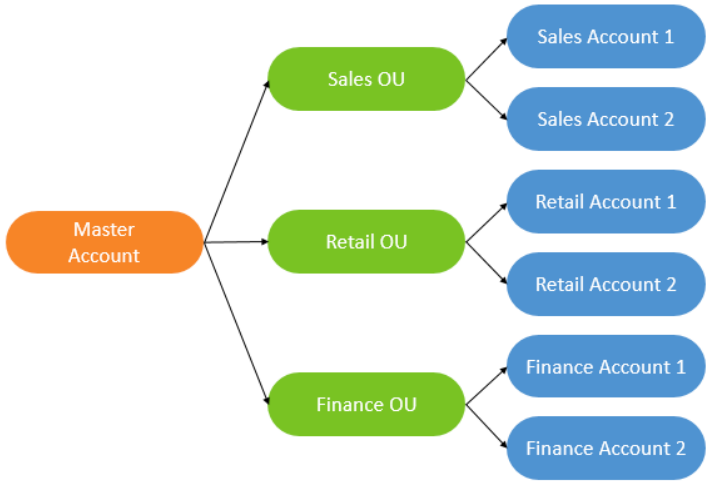
Multi Account Strategies

- 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**

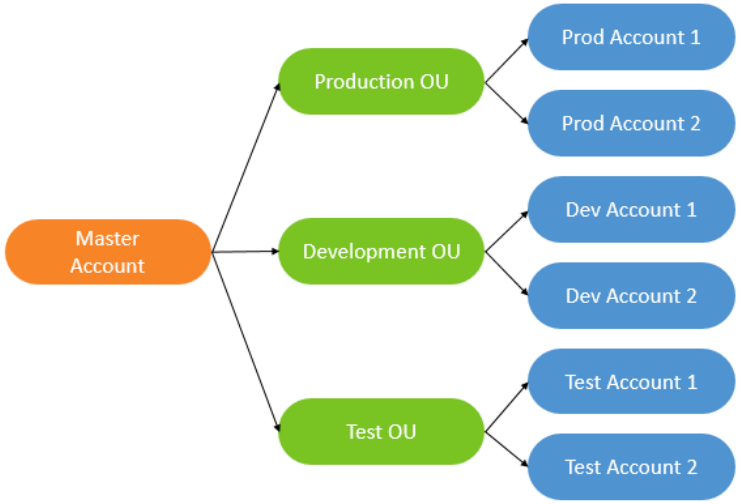
- Multi Account vs One Account Multi VPC
- Use tagging standards for billing purposes
- Enable CloudTrail on all accounts, send logs to central S3 account
- Send CloudWatch Logs to central logging account
- Establish Cross Account Roles for Admin purposes

Organizational Units (OU) - Examples

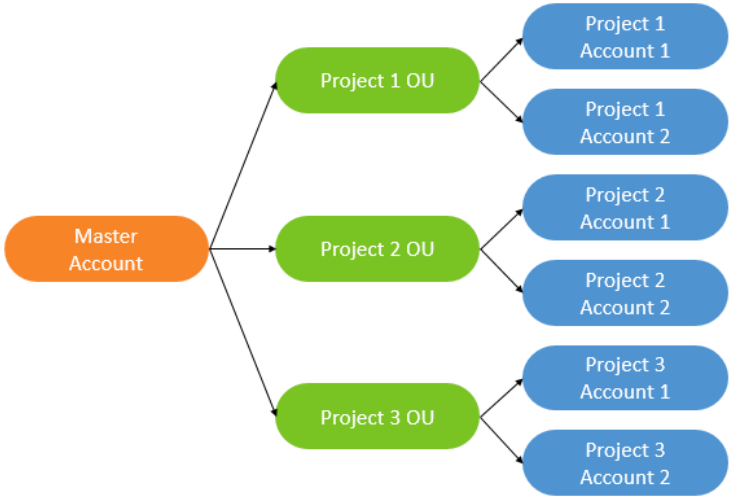
Business Unit



Environmental Lifecycle

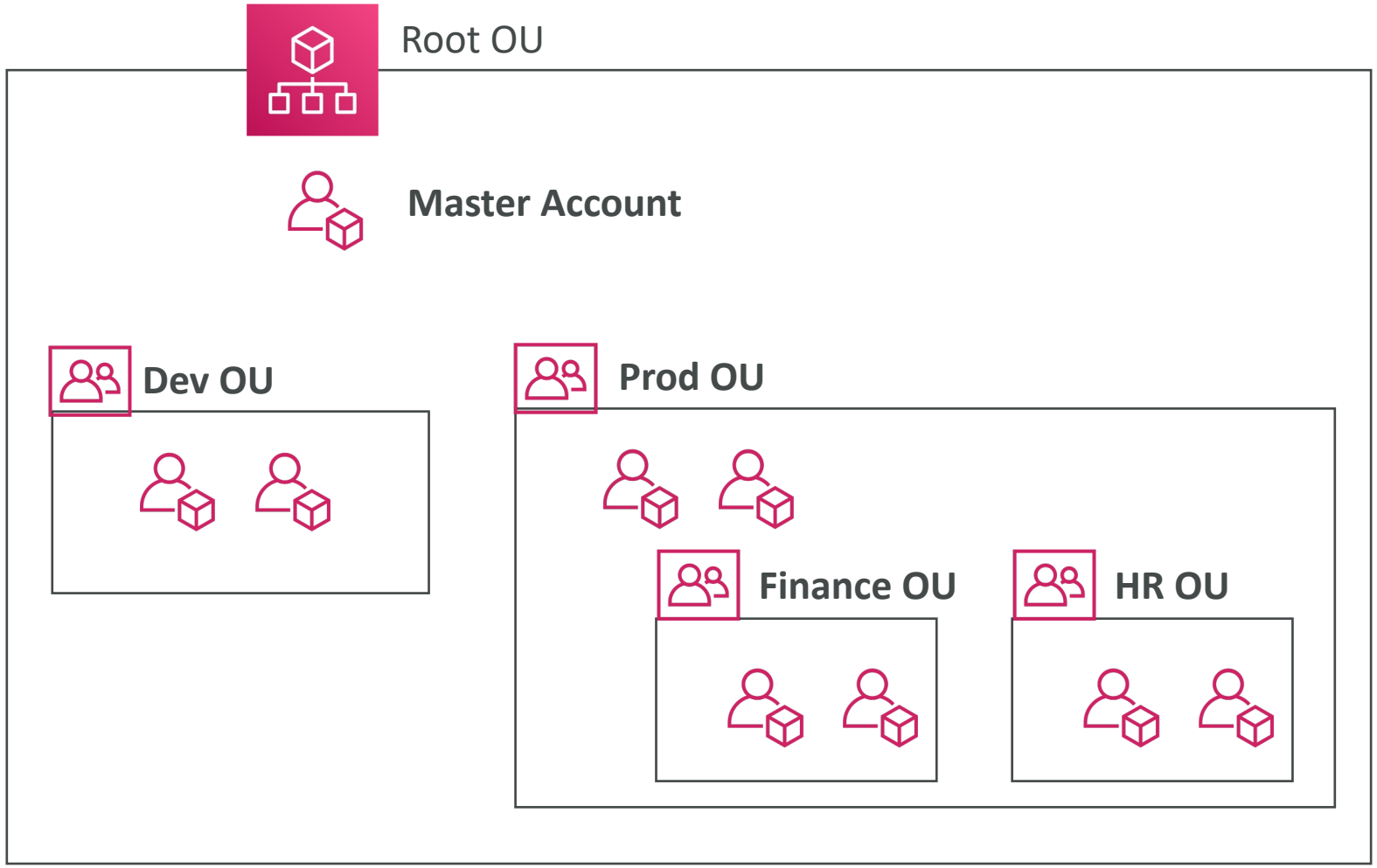


Project-based



https://aws.amazon.com/answers/account-management/aws-multi-account-billing-strategy/

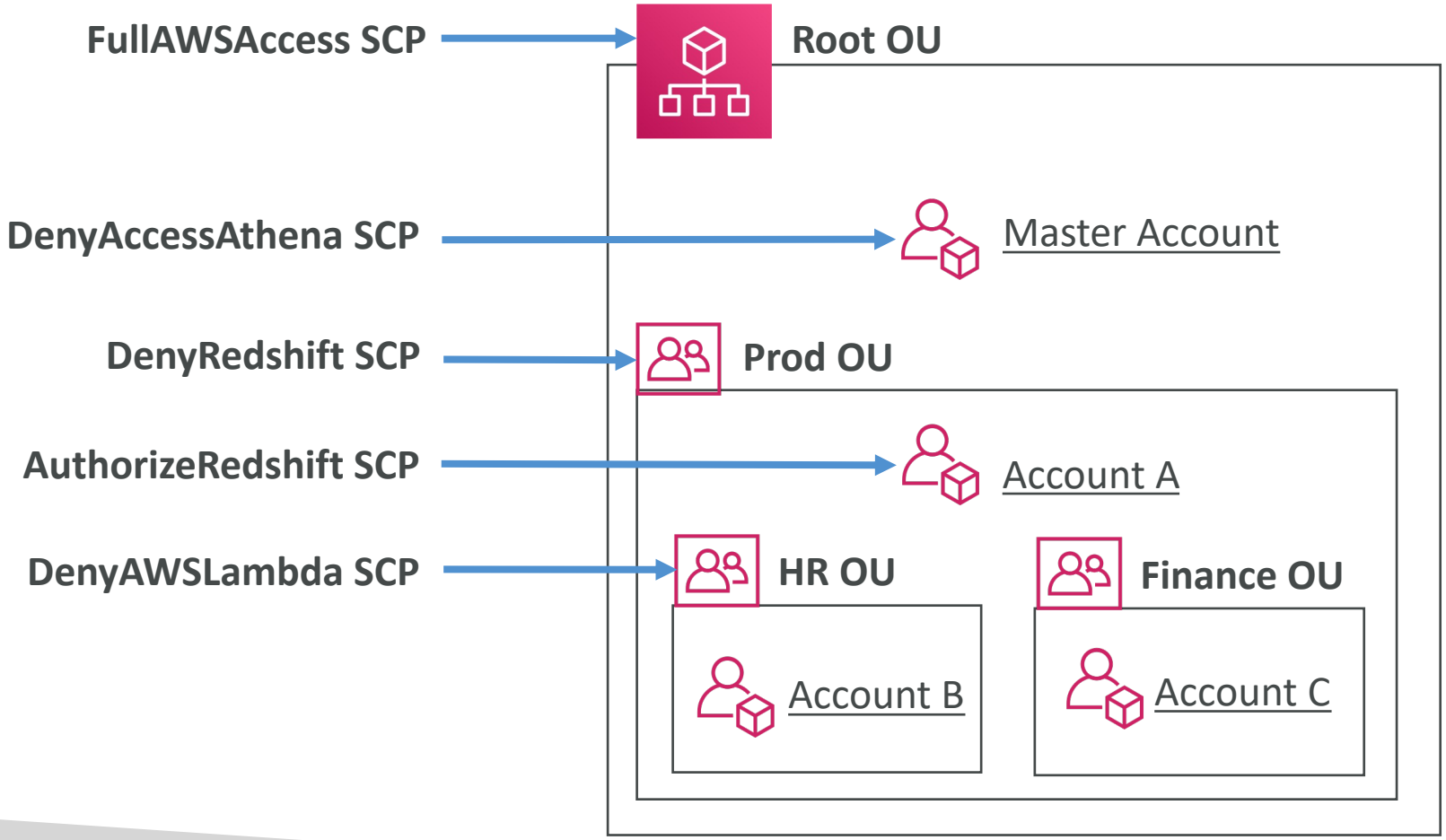
AWS Organization



Service Control Policies (SCP)

- Whitelist or blacklist IAM actions
- Applied at the **OU** or **Account** level
- Does not apply to the Master Account
- SCP is applied to all the **Users and Roles** of 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



- Master 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
Blacklist and Whitelist 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

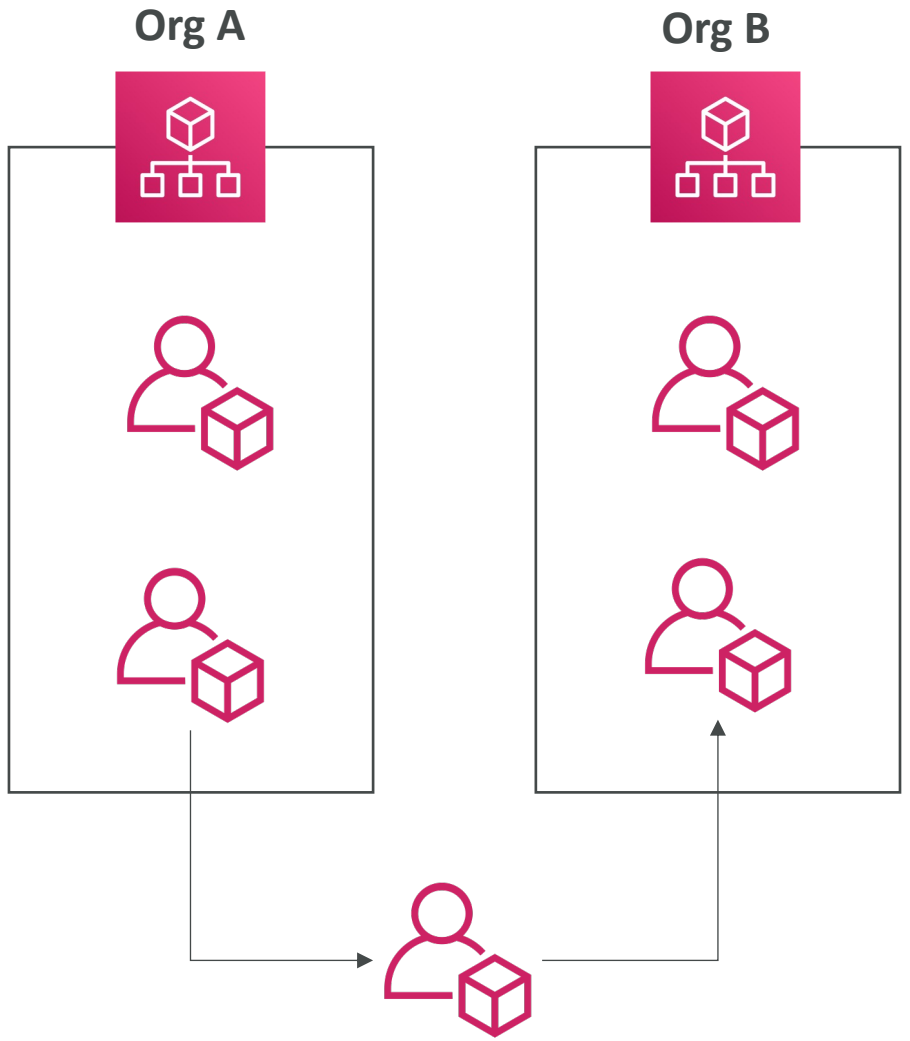
AWS Organization – Moving Accounts

To migrate accounts from one organization to another

1. Remove the member account from the old organization
2. Send an invite to the new organization
3. Accept the invite to the new organization from the member account

If you want the master account of the old organization to also join the new organization, do the following:

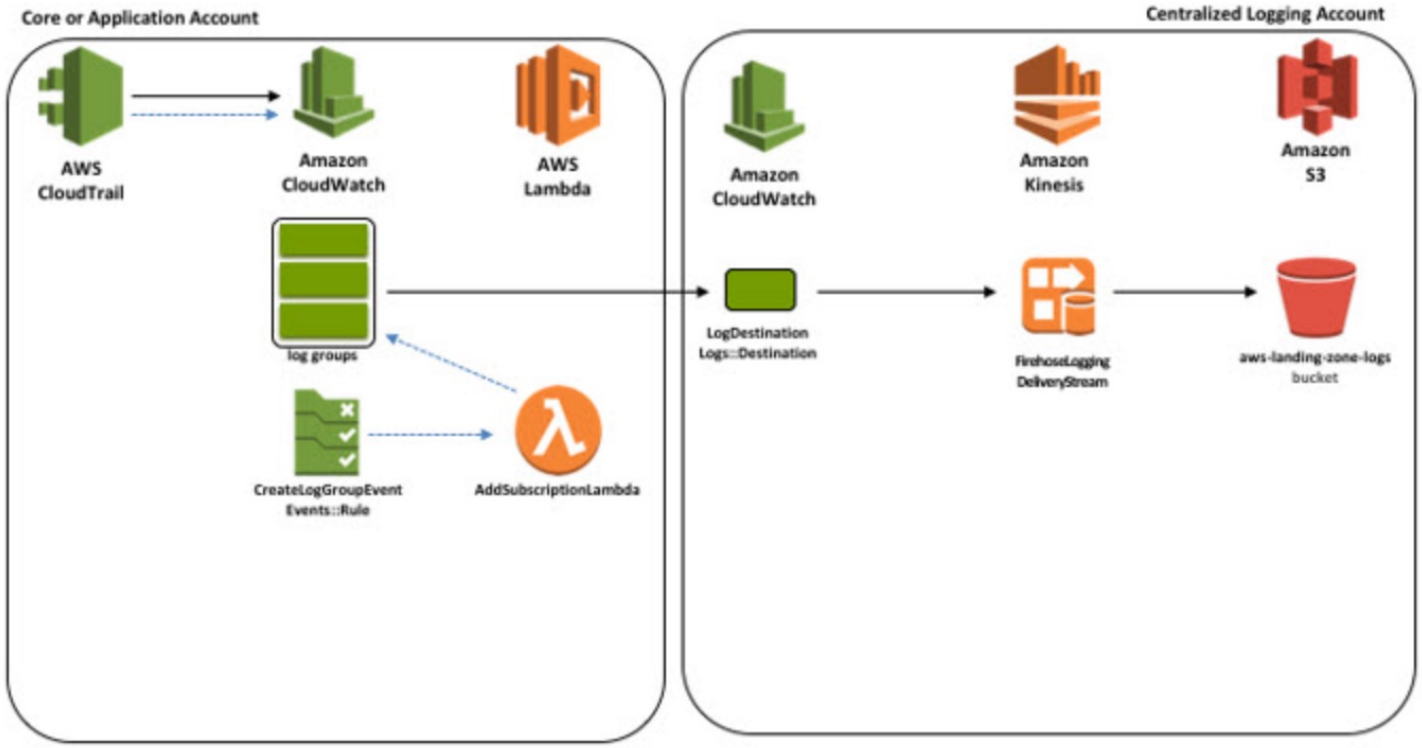
1. Remove the member accounts from the organizations using procedure above
2. Delete the old organization
3. Repeat the process above to invite the old master account to the new org



Multi Account with AWS

- Any cross account action requires to define IAM "trust"
- IAM roles can be assumed cross account
  - no need to share IAM creds
  - Uses AWS Security Token Service (STS)
- CodePipeline – cross account invocation of CodeDeploy for example
- AWS Config – aggregators
- CloudWatch Events – Event Bus = multi accounts events
- CloudFormation – StackSets

CloudWatch Logs: Centralize Logs Multi Account & Multi Region



https://aws.amazon.com/blogs/architecture/stream-amazon-cloudwatch-logs-t
a-centralized-account-for-audit-and-analysis/

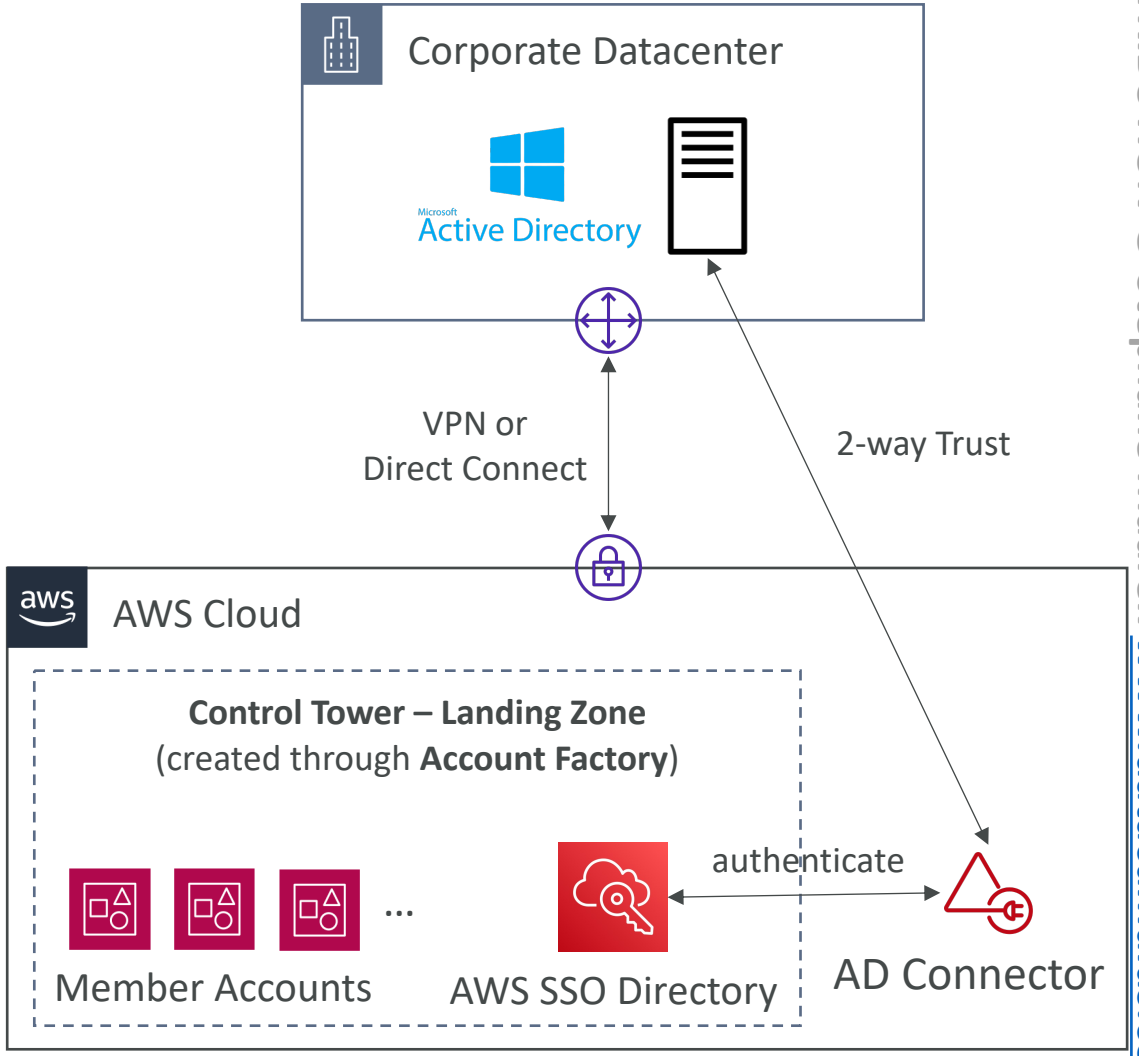
Other Services

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)

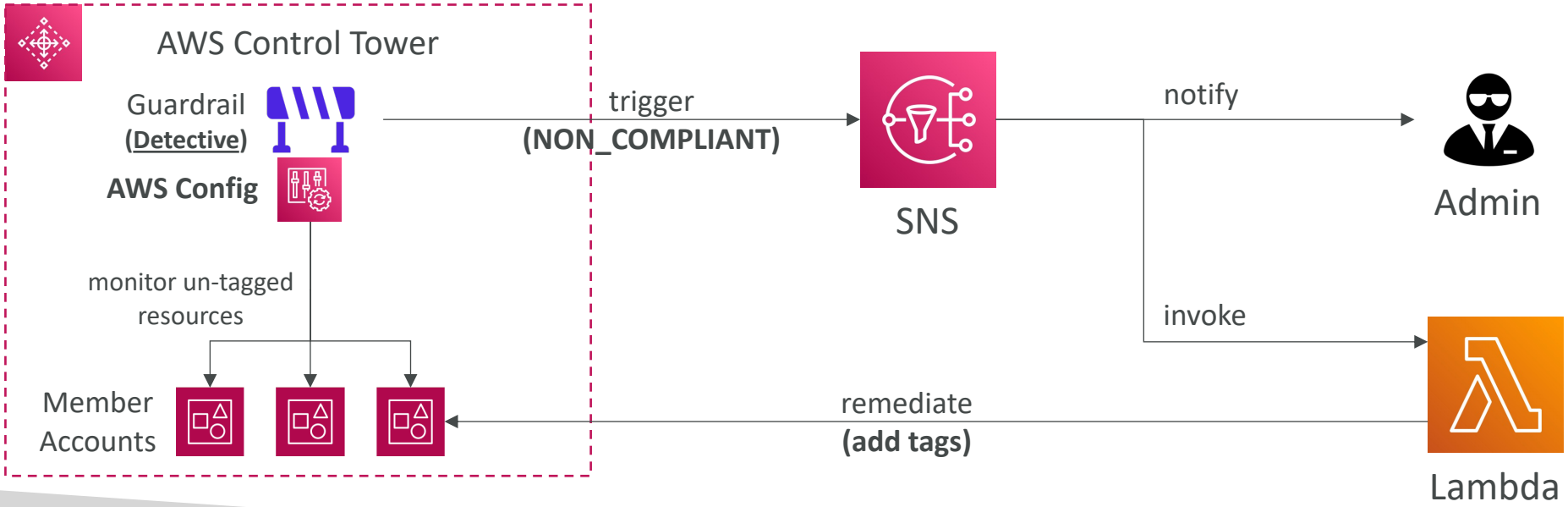
AWS Control Tower – Account Factory

- 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)



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!