

# Terraform on AWS EKS Kubernetes

Author: Nho Luong

Skill: DevOps Engineer Lead



The screenshot shows the AWS Certmetrics portal at [cp.certmetrics.com/amazon/en/credentials/status](https://cp.certmetrics.com/amazon/en/credentials/status). The interface includes the Alpine Testing Solutions and AWS logos. On the left, a sidebar lists navigation options: HOME, PROFILE, EXAM REGISTRATION, EXAM HISTORY, CERTIFICATIONS (with "Certification status" selected), BENEFITS, DIGITAL BADGES, and SUPPORT AND FAQS. The main content area displays "Certification status" for four active certificates:

- PROFESSIONAL**: **AWS Certified Solutions Architect - Professional**, SAP, ACTIVE DATE 2024-06-21, EXPIRATION DATE 2027-06-21. [VIEW MORE >](#)
- SPECIALTY**: **AWS Certified Security - Specialty**, SCS, ACTIVE DATE 2024-06-19, EXPIRATION DATE 2027-06-19. [VIEW MORE >](#)
- PROFESSIONAL**: **AWS Certified DevOps Engineer - Professional**, DOP, ACTIVE DATE 2024-06-17, EXPIRATION DATE 2027-06-17. [VIEW MORE >](#)
- ASSOCIATE**: **AWS Certified Solutions Architect - Associate**, SAA, ACTIVE DATE 2024-06-15, EXPIRATION DATE 2027-06-21. [VIEW MORE >](#)

On the right side of the main content area, there is a profile summary for "NHO LUONG" with the identifier "AWS04440050".





# Terraform on AWS EKS Kubernetes **Demos Code**



Terraform Fundamentals

Kubernetes Fundamentals

AWS VPC 3-Tier Network

EC2 Bastion Host in Public Subnets

EKS Cluster  
(Public and Private Node Group)

Kubernetes Deployment & Services using TF k8s Provider

Terraform Remote State Storage  
(Key Concept for Real-World Implementation)



EKS Cluster



Nho Luong  
DevOps Engineer Lead



EKS IRSA

(AWS IAM Roles for k8s Service Accounts)

Kubernetes Storage: EBS CSI

Kubernetes Storage: EFS CSI

EKS Load Balancer Controller Ingress

EKS IAM

EKS Fargate

Autoscaling  
(Cluster Autoscaler, HPA, VPA)

EKS Monitoring and Logging  
(CloudWatch Agent + FluentBit)

# GitHub Step-by-Step Documentation

## terraform-on-aws-eks

- > 01-Infrastructure-as-Code-IaC-Basics
- > 02-Terraform-Basics
- > 03-Terraform-Settings-Providers-Resources
- > 04-Terraform-Variables-and-Datasources
- > 05-Terraform-Loops-MetaArguments-SplatOperator
- > 06-AWS-VPC
- > 07-AWS-EC2-BastionHost
- > 08-AWS-EKS-Cluster-Basics
- > 09-Docker-and-Kubernetes-Fundamentals
- > 10-Kubernetes-Deployment-and-Service
- > 11-Kubernetes-Resources-via-Terraform
- > 12-Terraform-Remote-State-Storage
- > 13-EKS-IRSA
- > 14-EBS-CSl-Install-Kubernetes-Storage
- > 15-EBS-Kubernetes-SampleApp-YAML
- > 16-EBS-Kubernetes-SampleApp-Terraform
- > 17-EBS-Resizing-on-EKS
- > 18-EBS-CSl-Install-using-EKS-AddOn
- > 19-EKS-Admins-AWS-Admin-User
- > 20-EKS-Admins-AWS-Basic-User

- > 20-EKS-Admins-AWS-Basic-User
- > 21-EKS-Admins-as-AWS-IAM-Users
- > 22-EKS-Admins-with-AWS-IAM-Roles
- > 23-EKS-Admins-with-AWS-IAM-Roles-TF
- > 24-EKS-ReadOnly-IAM-Users
- > 25-EKS-DeveloperAccess-IAM-Users
- > 26-EKS-with-LoadBalancer-Controller
- > 27-EKS-Ingress-Basics
- > 28-EKS-Ingress-Context-Path-Routing
- > 29-EKS-Ingress-SSL-SSLRedirect
- > 30-EKS-ExternalDNS-Install
- > 31-EKS-ExternalDNS-with-Ingress-Service
- > 32-EKS-ExternalDNS-with-k8s-Service
- > 33-EKS-Ingress-NameBasedVirtualHost-Routing
- > 34-EKS-Ingress-SSLDiscivery-Host
- > 35-EKS-Ingress-SSLDiscivery-TLS
- > 36-EKS-Ingress-Groups
- > 37-EKS-Ingress-TargetType-IP
- > 38-EKS-Ingress-InternalLB
- > 39-EKS-Ingress-Cross-Namespace
- > 40-EKS-NLB-Basics

- > 41-EKS-NLB-TLS-externaldns
- > 42-EKS-NLB-InternalLB
- > 43-EKS-Fargate-Profiles
- > 44-EKS-Run-k8s-workloads-on-Fargate
- > 45-Fargate-Only-EKS-Cluster
- > 46-EKS-EFS-CSl-Install
- > 47-EKS-EFS-Static-Provisioning
- > 48-EKS-EFS-Dynamic-Provisioning
- > 49-EKS-EFS-Fargate
- > 50-EKS-Cluster-Autoscaler
- > 51-EKS-Cluster-Autoscaler-Testing
- > 52-EKS-Horizontal-Pod-Autoscaler
- > 53-EKS-Vertical-Pod-Autoscaler-Install
- > 54-EKS-Monitoring-Logging-kubectl
- > 55-EKS-Monitoring-Logging-Terraform
- > course-presentation

# How are Terraform Projects organized ?

4 Terraform Projects  
1 YAML Project

- ✓ 31-EKS-ExternalDNS-with-Ingress-Service
  - > 01-ekscluster-terraform-manifests
  - > 02-lbc-install-terraform-manifests
  - > 03-externaldns-install-terraform-manifests
  - > 04-kube-manifests-ingress-externaldns
  - > 05-ingress-externaldns-terraform-manifests

- ✓ 31-EKS-ExternalDNS-with-Ingress-Service
  - > 01-ekscluster-terraform-manifests
    - > local-exec-output-files
    - > private-key
  - > c1-versions.tf
  - > c2-01-generic-variables.tf
  - > c2-02-local-values.tf
  - > c3-01-vpc-variables.tf
  - > c3-02-vpc-module.tf
  - > c3-03-vpc-outputs.tf
  - > c4-01-ec2bastion-variables.tf
  - > c4-02-ec2bastion-outputs.tf
  - > c4-03-ec2bastion-securitygroups.tf
  - > c4-04-ami-datasource.tf
  - > c4-05-ec2bastion-instance.tf
  - > c4-06-ec2bastion-elasticip.tf
  - > c4-07-ec2bastion-provisioners.tf
  - > c5-01-eks-variables.tf
  - > c5-02-eks-outputs.tf
  - > c5-03-iamrole-for-eks-cluster.tf
  - > c5-04-iamrole-for-eks-nodegroup.tf
  - > c5-05-securitygroups-eks.tf
  - > c5-06-eks-cluster.tf
  - > c5-07-eks-node-group-public.tf

EKS Cluster Terraform Project

- > c5-08-eks-node-group-private.tf
- > c6-01-iam-oidc-connect-provider-variables.tf
- > c6-02-iam-oidc-connect-provider.tf
- > c7-01-kubernetes-provider.tf
- > c7-02-kubernetes-configmap.tf
- > c8-01-iam-admin-user.tf
- > c8-02-iam-basic-user.tf
- > c9-01-iam-role-eksadmins.tf
- > c9-02-iam-group-and-user-eksadmins.tf
- > c10-01-iam-role-eksreadonly.tf
- > c10-02-iam-group-and-user-eksreadonly.tf
- > c10-03-k8s-clusterrole-clusterrolebinding.tf
- > c11-01-iam-role-eksdeveloper.tf
- > c11-02-iam-group-and-user-eksdeveloper.tf
- > c11-03-k8s-clusterrole-clusterrolebinding.tf
- > c11-04-namespaces.tf
- > c11-05-k8s-role-rolebinding.tf
- > ec2bastion.auto.tfvars
- > eks.auto.tfvars
- > terraform.tfvars
- > vpc.auto.tfvars

# How are Terraform Projects organized ?

## AWS Load Balancer Controller Terraform Project

```
✓ 31-EKS-ExternalDNS-with-Ingress-Service
  > 01-ekscluster-terraform-manifests
  ✓ 02-lbc-install-terraform-manifests
    ↴ c1-versions.tf
    ↴ c2-remote-state-datasource.tf
    ↴ c3-01-generic-variables.tf
    ↴ c3-02-local-values.tf
    ↴ c4-01-lbc-datasources.tf
    ↴ c4-02-lbc-iam-policy-and-role.tf
    ↴ c4-03-lbc-helm-provider.tf
    ↴ c4-04-lbc-install.tf
    ↴ c4-05-lbc-outputs.tf
    ↴ c5-01-kubernetes-provider.tf
    ↴ c5-02-ingress-class.tf
    ↴ terraform.tfvars
```

## AWS External DNS Terraform Project

```
✓ 31-EKS-ExternalDNS-with-Ingress-Service
  > 01-ekscluster-terraform-manifests
  > 02-lbc-install-terraform-manifests
  ✓ 03-externaldns-install-terraform-manifests
    ↴ c1-versions.tf
    ↴ c2-remote-state-datasource.tf
    ↴ c3-01-generic-variables.tf
    ↴ c3-02-local-values.tf
    ↴ c4-01-externaldns-iam-policy-and-role.tf
    ↴ c4-02-externaldns-helm-provider.tf
    ↴ c4-03-externaldns-install.tf
    ↴ c4-04-externaldns-outputs.tf
    ↴ terraform.tfvars
  > 04-kube-manifests-ingress-externaldns
  > 05-ingress-externaldns-terraform-manifests
```

## Ingress + External DNS YAML Project

```
✓ 31-EKS-ExternalDNS-with-Ingress-Service
  > 01-ekscluster-terraform-manifests
  > 02-lbc-install-terraform-manifests
  > 03-externaldns-install-terraform-manifests
  ✓ 04-kube-manifests-ingress-externaldns
    ! 01-Nginx-App1-Deployment-and-NodePortService.yml
    ! 02-Nginx-App2-Deployment-and-NodePortService.yml
    ! 03-Nginx-App3-Deployment-and-NodePortService.yml
    ! 04-ALB-Ingress-SSL-Redirect-ExternalDNS.yml
  > 05-ingress-externaldns-terraform-manifests
```

# How are Terraform Projects organized ?

Ingress + External DNS  
Terraform Project

```
✓ 31-EKS-ExternalDNS-with-Ingress-Service
  > 01-ekscluster-terraform-manifests
  > 02-lbc-install-terraform-manifests
  > 03-externaldns-install-terraform-manifests
  > 04-kube-manifests-ingress-externaldns
  ✓ 05-ingress-externaldns-terraform-manifests
    > listen-ports
    ✓ c1-versions.tf
    ✓ c2-remote-state-datasource.tf
    ✓ c3-providers.tf
    ✓ c4-kubernetes-app1-deployment.tf
    ✓ c5-kubernetes-app2-deployment.tf
    ✓ c5-kubernetes-app3-deployment.tf
    ✓ c7-kubernetes-app1-nodeport-service.tf
    ✓ c8-kubernetes-app2-nodeport-service.tf
    ✓ c9-kubernetes-app3-nodeport-service.tf
    ✓ c10-kubernetes-ingress-service.tf
    ✓ c11-acm-certificate.tf
```

In short, every AWS EKS demo is  
**automated** with Terraform

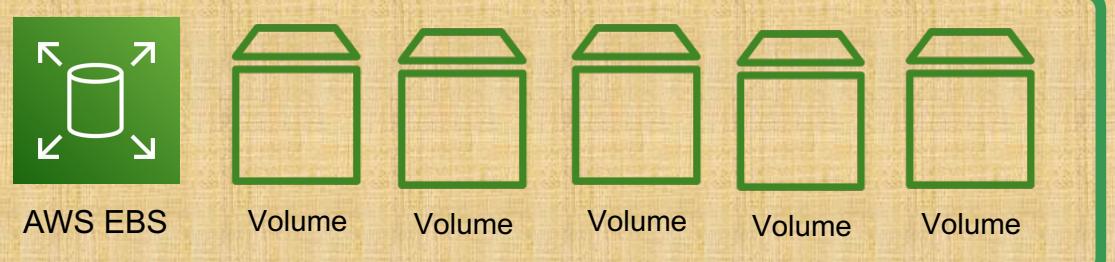
Every Terraform Project is **organized**  
in GitHub and contains **step by step**  
**documentation**

# AWS EKS Kubernetes Storage



Terraform

## EBS CSI Controller



D1

AWS EBS CSI Install on EKS

D2

Kubernetes Sample App with EBS Volumes (YAML Manifests)

D3

Kubernetes Sample App with EBS Volumes (Terraform Manifests)

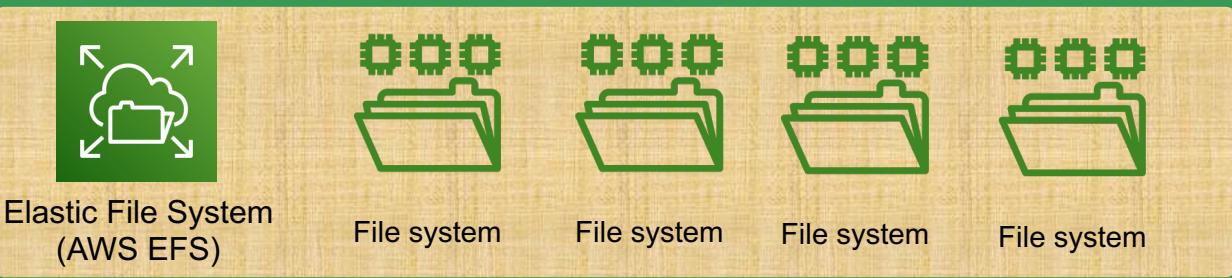
D4

EBS Volumes Resizing

D5

EBS CSI Install using EKS Add-On

## EFS CSI Controller



D1

AWS EFS CSI Install on EKS

D2

AWS EFS Static Provisioning

D3

AWS EFS Dynamic Provisioning

D4

AWS EFS Mounts for Fargate Workloads

# AWS EKS Autoscaling, Monitoring & Logging



Terraform

## Autoscaling

D1

Kubernetes Cluster Autoscaler Install

D2

Cluster Autoscaler Testing using Sample App

D3

Horizontal Pod Autoscaler with Sample App

D4

Vertical Pod Autoscaler Install

D5

Vertical Pod Autoscaler Testing with Sample App

## Monitoring & Logging

D1

EKS Monitoring & Logging  
(Implement using kubectl)

CloudWatch Agent

FluentBit

D2

EKS Monitoring & Logging  
(Automate with Terraform)

CloudWatch Agent

FluentBit

# AWS EKS IAM Demos



Section-19

Provision AWS IAM Admins as EKS Admins

Section-20

Provision AWS IAM Basic User as EKS Admin

Section-21

Automate Section-19,20 with Terraform

Section-22

Provision EKS Admins with AWS IAM Roles & IAM Groups

Section-23

Automate Section-22 with Terraform

Section-24

Provision EKS Read-Only Users with IAM Roles, IAM Groups, Kubernetes Cluster Role and Cluster Role Binding

Section-25

A Developer requesting full access to a namespace (dev) in EKS Cluster and in addition, also asking for read-only access to EKS Cluster

1. Kubernetes Role
2. Kubernetes Role Binding
3. Kubernetes Cluster Role
4. Kubernetes Cluster Role Binding



# AWS EKS Load Balancer Demos

AWS Load Balancer Controller Install

Ingress Basics

Ingress Context Path Routing

Ingress SSL & SSL Redirect

Ingress Cross Namespaces

External DNS Install

Ingress + External DNS

D1

D2

D3

D4

D5

D6

D7



AWS Load Balancer Controller

AWS Application Load Balancer

Kubernetes Ingress Service

Terraform Kubernetes Provider

D8

D9

D10

D11

D12

D13

D14

k8s Service + External DNS

Ingress Name based Virtual Host Routing

SSL Discovery - Host

SSL Discovery - TLS

Ingress Groups

Ingress Target Type - IP

Ingress Internal ALB

# AWS EKS Load Balancer Controller Demos

AWS Load  
Balancer  
Controller

AWS  
Network  
Load  
Balancer

Kubernetes  
Service

Terraform  
Kubernetes  
Provider



Elastic  
Load Balancing



Application Load  
Balancer



Network Load  
Balancer

D15

Network Load Balancer  
Basics

D16

Network Load Balancer  
TLS + External DNS

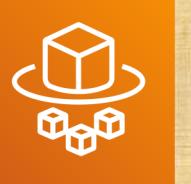
D17

Network Load Balancer  
Internal LB

# AWS Fargate



EKS Cluster



Fargate

D1

EKS Fargate Profiles

D2

Run k8s Workloads on  
Fargate

D3

Fargate Only EKS Cluster  
**(VERY IMPORTANT)**



# Terraform Providers

AWS

Kubernetes

Kubectl

Terraform  
Providers

HELM

NUL

HTTP

# GitHub Repository

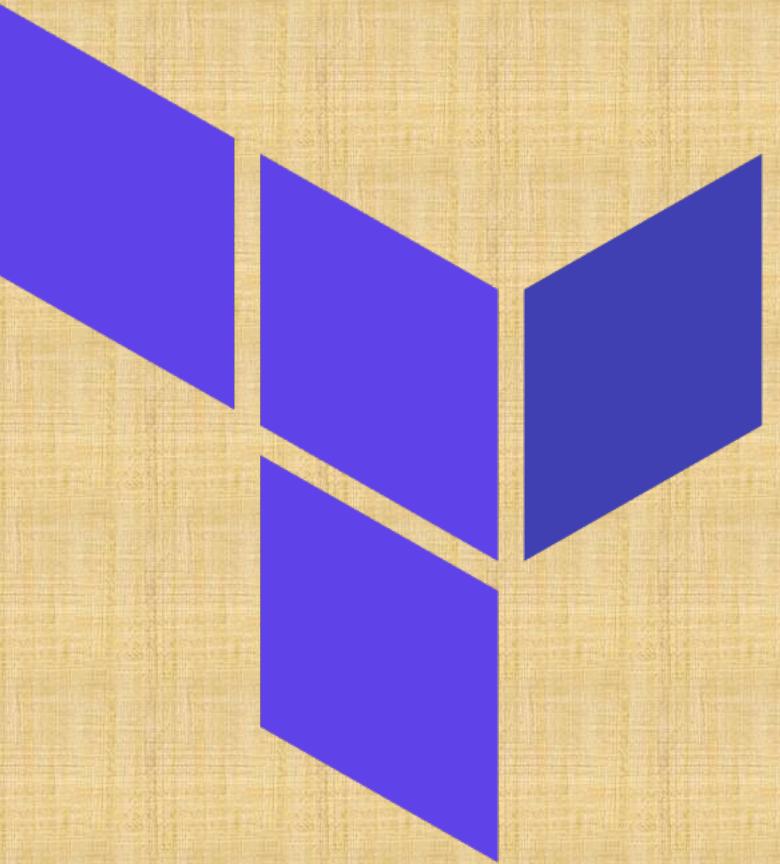
Repository Used For	Repository URL
Course Main Repository with step-by-step documentation	<a href="https://github.com/nholuongut/terraform-on-aws-eks">https://github.com/nholuongut/terraform-on-aws-eks</a>

# END OF INTRODUCTION

# Terraform

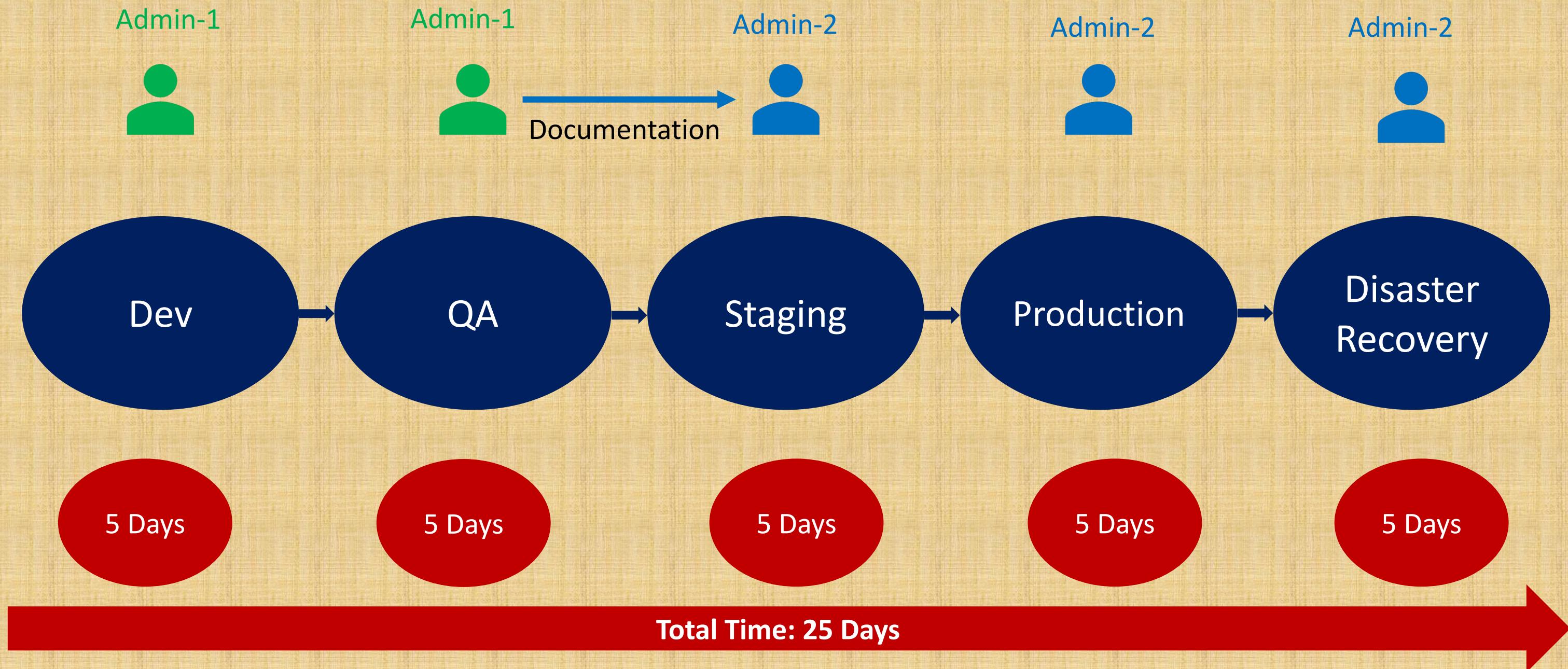
# Infrastructure as Code

# IaC



# What is Infrastructure as Code ?

# Traditional Way of Managing Infrastructure



# Traditional Way of Managing Infrastructure

Admin-1



Admin-1



Admin-2



Admin-2



Admin-2



No CI

Delays

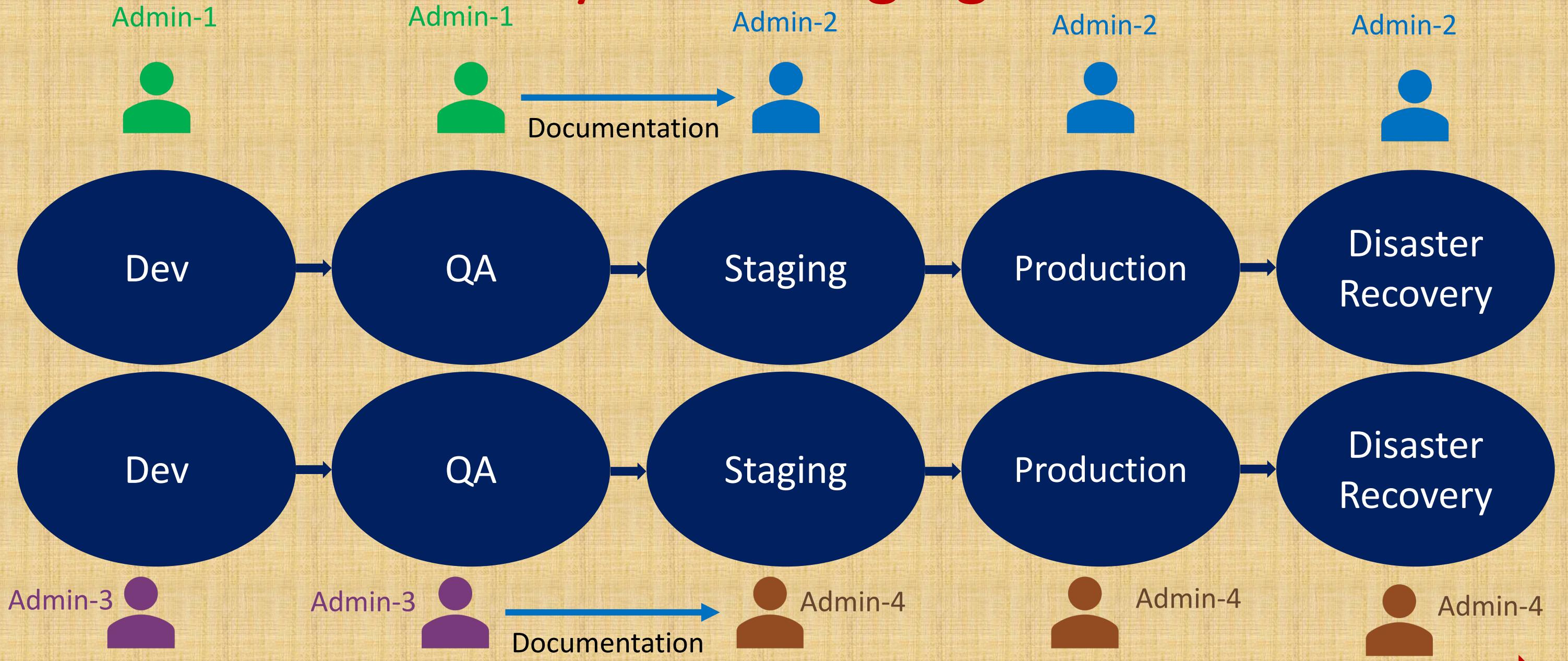
Issues

Outages

Not-in-Sync

Many Problems at many places in manual process

# Traditional Way of Managing Infrastructure



Infrastructure scalability – Workforce need to be increased to meet the timelines

# Traditional Way of Managing Infrastructure

Prod-1

Prod-2

Prod-3

Prod-4

Scale Up

Prod-1

Prod-2

Scale Down

On-Demand Scale-Up and Scale-Down is not an option

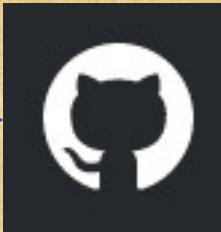
5 Days

Admin-1



Check-In TF Code

Github



Triggers  
TF Runs



Terraform  
Cloud

DevOps / CI CD for IaC

Scale-Up and Scale-Down On-Demand

Creates Infra

Dev

QA

Staging

Production

Disaster  
Recovery

One-Time  
Work

Re-Use  
Template  
s

Quick &  
Fast

Reliable

Tracked  
for Audit

Total Time: 25 Days reduced to 5 days, Provisioning environments will be in minutes or seconds

# Manage using IaC with Terraform

## Visibility

IaC serves as a very **clear reference** of what resources we created, and what their settings are. We don't have to **navigate** to the web console to check the parameters.

## Stability

If you **accidentally** change the **wrong** setting or delete the **wrong** resource in the web console you can **break things**. IaC helps **solve this**, especially when it is combined with **version control**, such as Git.

## Scalability

With IaC we can **write it once** and then **reuse it many times**. This means that one well written template can be used as the **basis for multiple services**, in multiple regions around the world, making it much easier to horizontally scale.

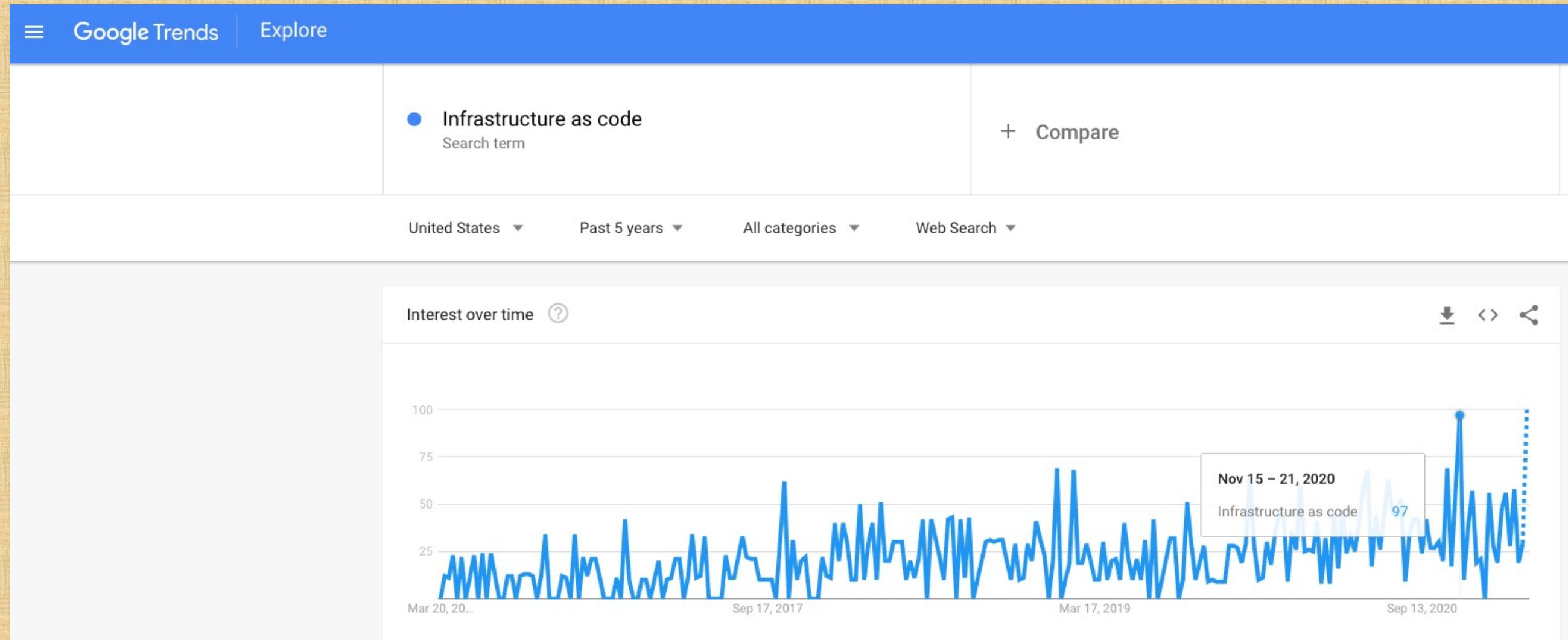
## Security

Once again IaC gives you a **unified template** for how to deploy our architecture. If we create one **well secured architecture** we can reuse it multiple times, and know that each deployed version is following the same settings.

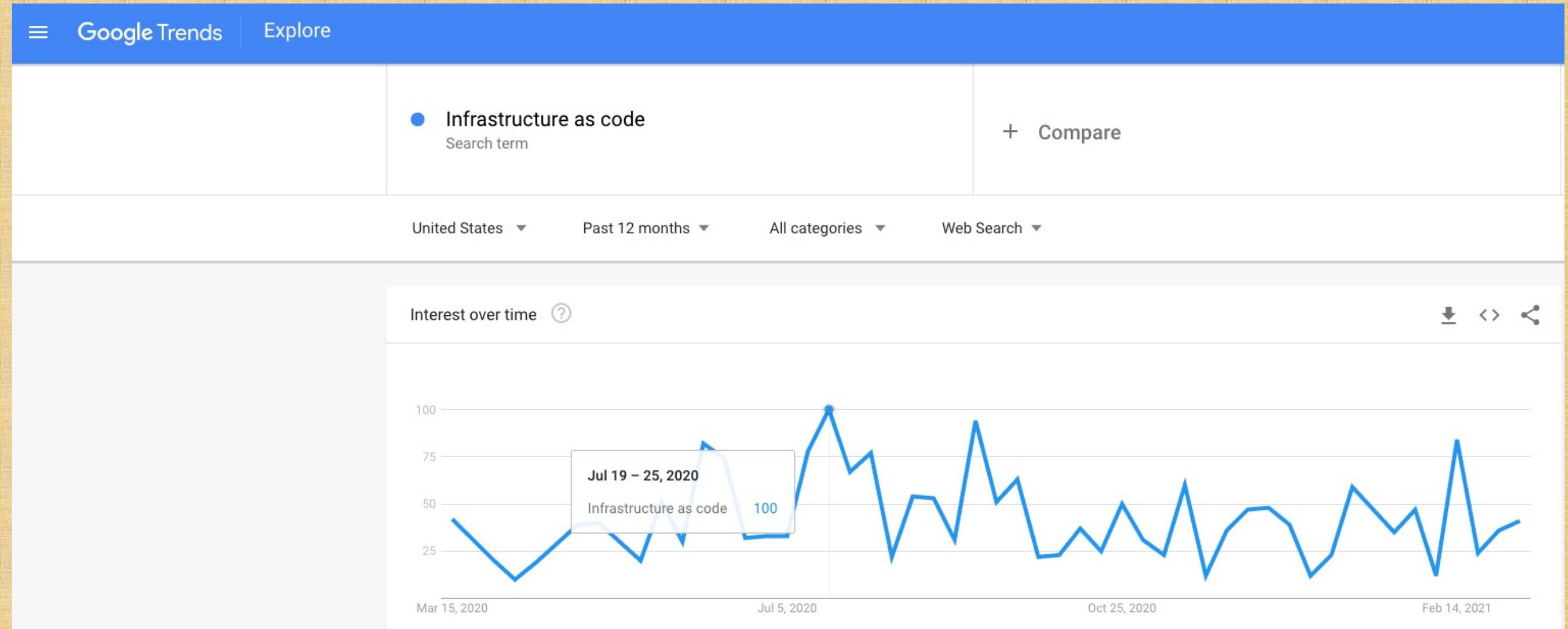
## Audit

Terraform not only creates resources it also **maintains the record** of what is created in real world cloud environments using its State files.

# Google Trends – Past 5 Years



# Google Trends – Past 1 Year

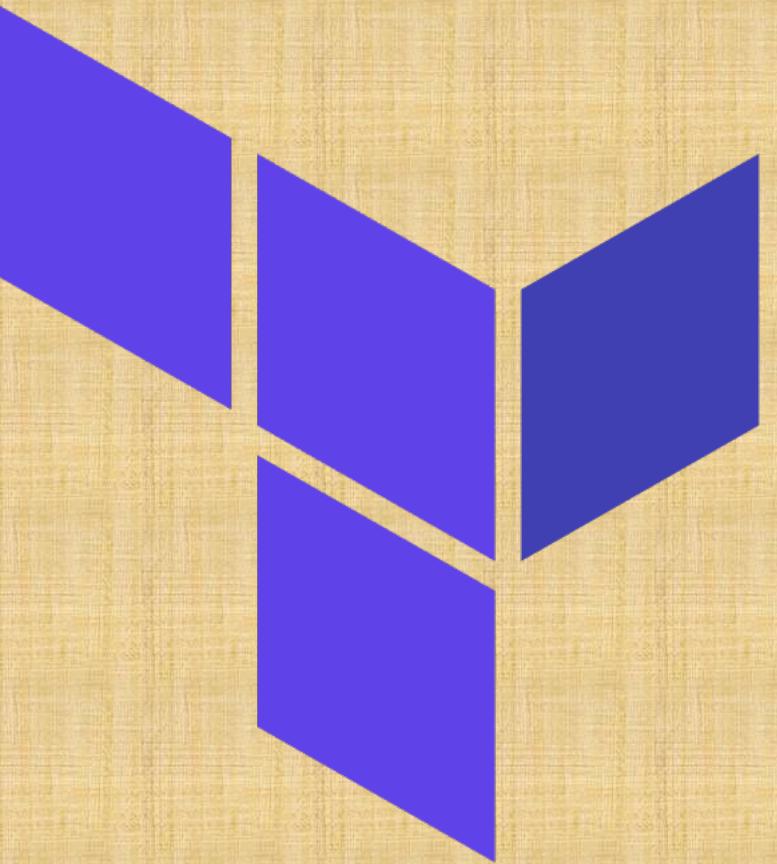


# Terraform

# Site Reliability

# Engineering

# SRE

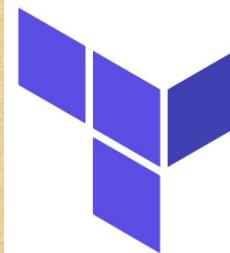


# What is SRE?

# Site Reliability Engineering - SRE

50%  
Infrastructure  
Automation

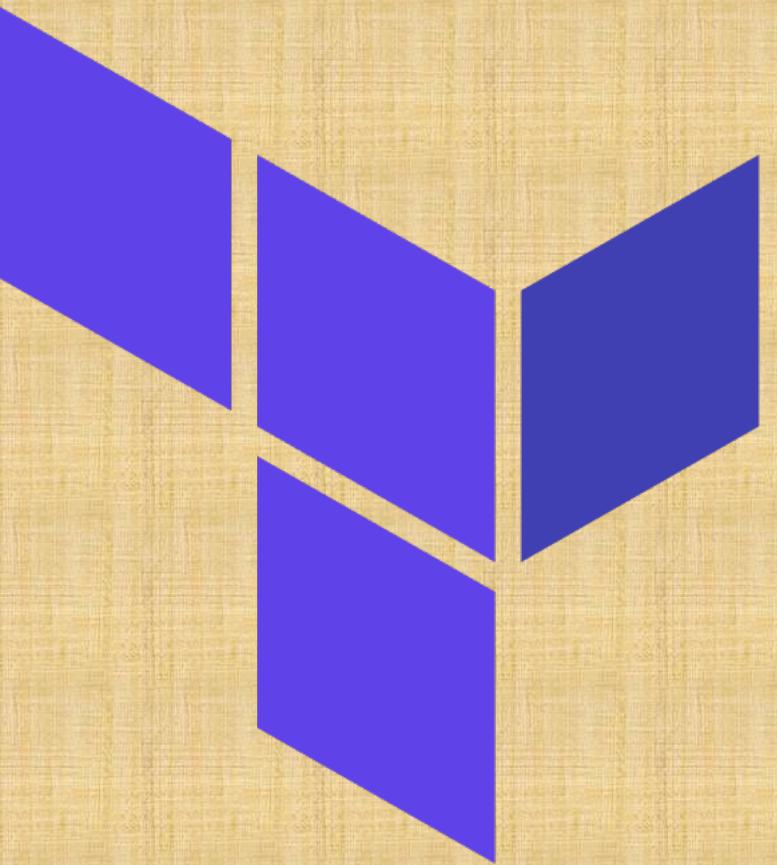
50%  
Operations



**Terraform**

Our course primarily focuses on Infrastructure Automation which will be a **key role** for an SRE even though weightage is 50%

# Terraform Installation



# Terraform Installation

Terraform CLI

AWS CLI

VS Code Editor

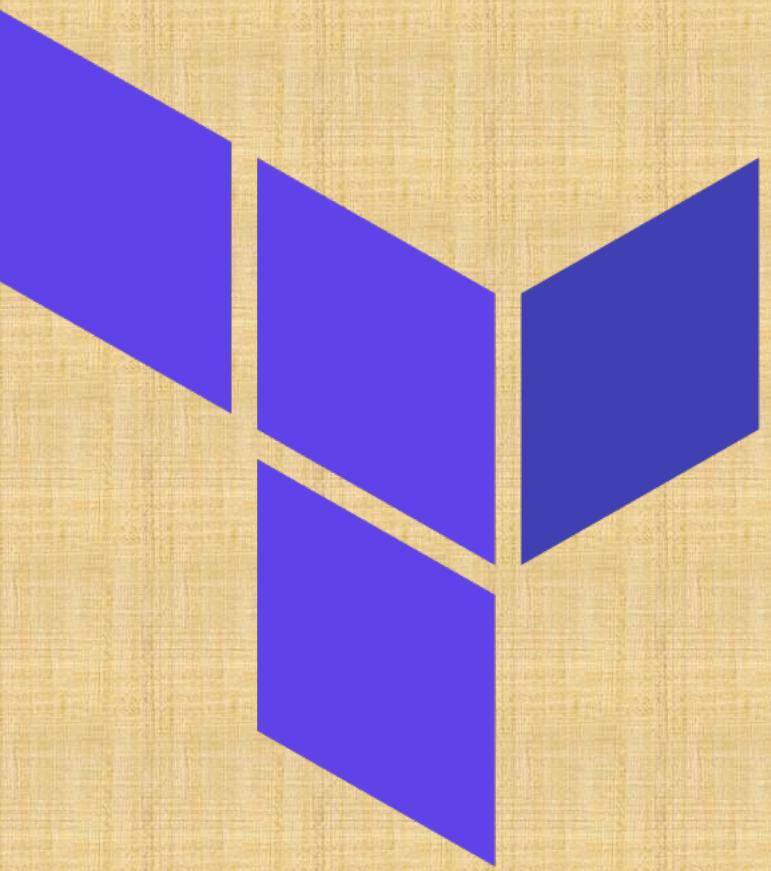
Terraform plugin  
for VS Code

Mac OS

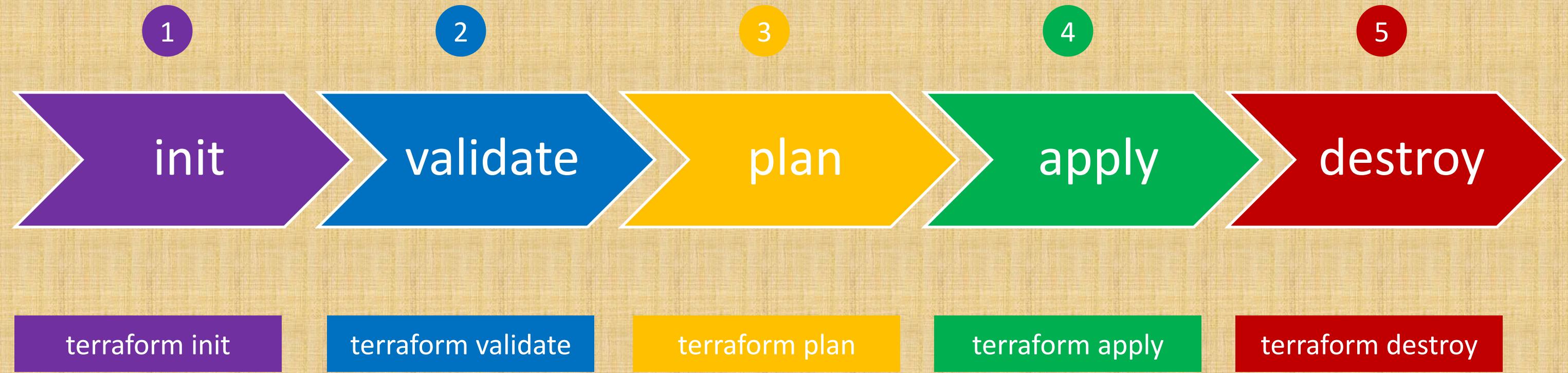
Windows OS

Linux OS

# Terraform Command Basics



# Terraform Workflow



# Terraform Workflow

1

init

2

validate

3

plan

4

apply

5

destroy

- Used to Initialize a working directory containing terraform config files
- This is the first command that should be run after writing a new Terraform configuration
- Downloads Providers

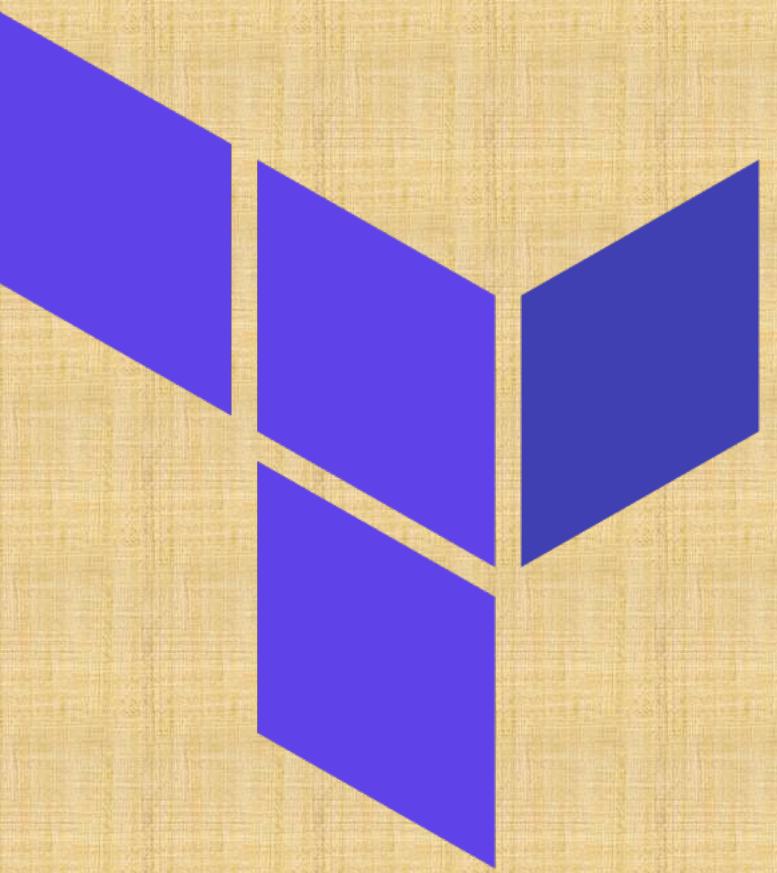
- Validates the terraform configurations files in that respective directory to ensure they are syntactically valid and internally consistent.

- Creates an execution plan
- Terraform performs a refresh and determines what actions are necessary to achieve the desired state specified in configuration files

- Used to apply the changes required to reach the desired state of the configuration.
- By default, apply scans the current directory for the configuration and applies the changes appropriately.

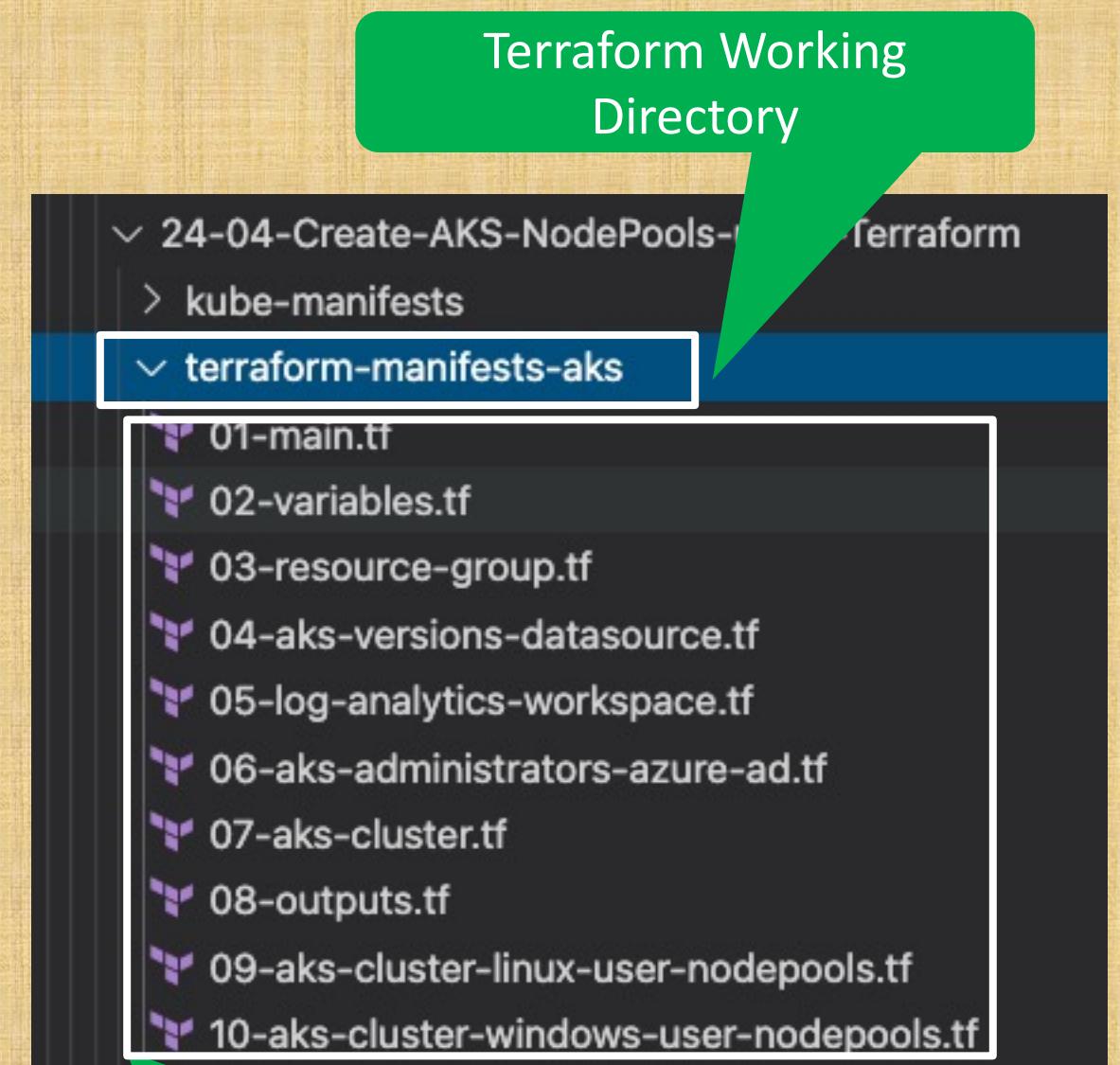
- Used to destroy the Terraform-managed infrastructure
- This will ask for confirmation before destroying.

# Terraform Language Basics



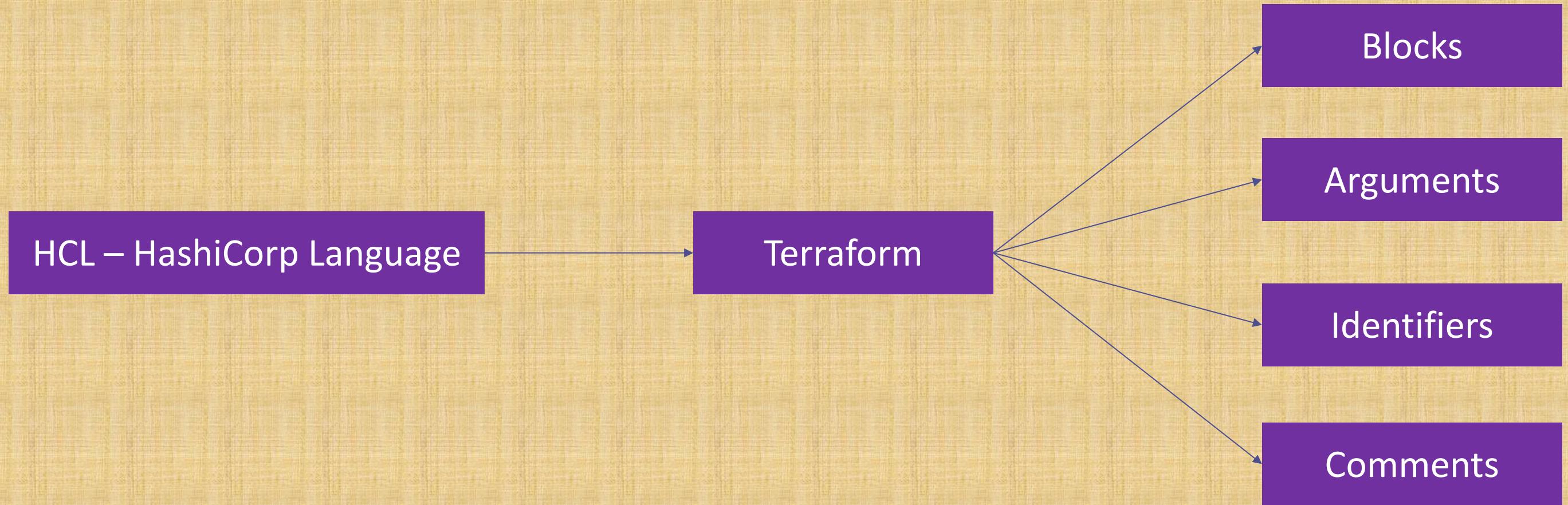
# Terraform Language Basics – Files

- Code in the Terraform language is stored in plain text files with the `.tf` file extension.
- There is also a JSON-based variant of the language that is named with the `.tf.json` file extension.
- We can call the files containing terraform code as Terraform Configuration Files or Terraform Manifests



Terraform Configuration Files  
ending with `.tf` as extension

# Terraform Language Basics – Configuration Syntax



# Terraform Language Basics – Configuration Syntax

```
# Template
<BLOCK TYPE> "<BLOCK LABEL>" "<BLOCK LABEL>"  {
    # Block body
    <IDENTIFIER> = <EXPRESSION> # Argument
}

# AWS Example
resource "aws_instance" "ec2demo" {
    ami           = "ami-04d29b6f966df1537"
    instance_type = "t2.micro"
}
```

Block Type

Top Level &  
Block inside  
Blocks

**Top Level Blocks:** resource, provider

**Block Inside Block:** provisioners,  
resource specific blocks like tags

Arguments

Block Labels

Based on Block  
Type block labels  
will be 1 or 2

**Example:**  
Resource – 2  
labels

Variables – 1 label

# Terraform Language Basics – Configuration Syntax

Argument  
Name  
[or]  
Identifier

```
# Template
<BLOCK TYPE> "<BLOCK LABEL>" "<BLOCK LABEL>"  {
    # Block body
    <IDENTIFIER> = <EXPRESSION> # Argument
}

# AWS Example
resource "aws_instance" "ec2demo" {
    ami           = "ami-04d29b6f966df1537"
    instance_type = "t2.micro"
}
```

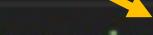
Argument  
Value  
[or]  
Expression

# Terraform Language Basics – Configuration Syntax

Single Line Comments with # or //

```
# EC2 Instance Resource
resource "aws_instance" "ec2demo" {
    ami                  = "ami-0885b1f6bd170450c" // Ubuntu 20.04 LTS
    instance_type        = "t2.micro"
    /*
    Multi-line comments
    Line-1
    Line-2
    */
}
```

Multi-line comment



**Terraform** language uses a **limited** number of **top-level block** types, which are **blocks** that can appear **outside** of any other **block** in a TF configuration file.

## Terraform Top-Level Blocks

Most of **Terraform's features** are implemented as **top-level** blocks.

Terraform Block

Providers Block

Resources Block

Fundamental Blocks

Input Variables Block

Output Values Block

Local Values Block

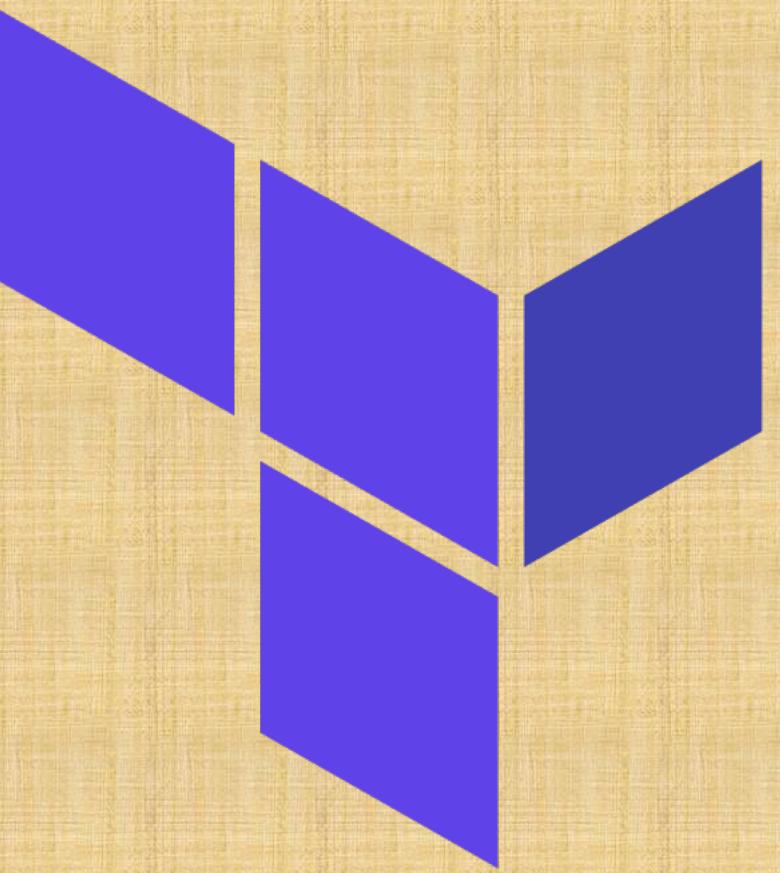
Variable Blocks

Data Sources Block

Modules Block

Calling / Referencing Blocks

# Terraform Fundamental Blocks



# Terraform Basic Blocks

## Terraform Block

Special block used to configure some **behaviors**

Required Terraform Version

List Required Providers

Terraform Backend

## Provider Block

**HEART** of Terraform

Terraform relies on providers to **interact** with Remote Systems

Declare providers for Terraform to **install** providers & use them

Provider configurations belong to **Root Module**

## Resource Block

Each Resource Block describes one or more Infrastructure Objects

**Resource Syntax:** How to declare Resources?

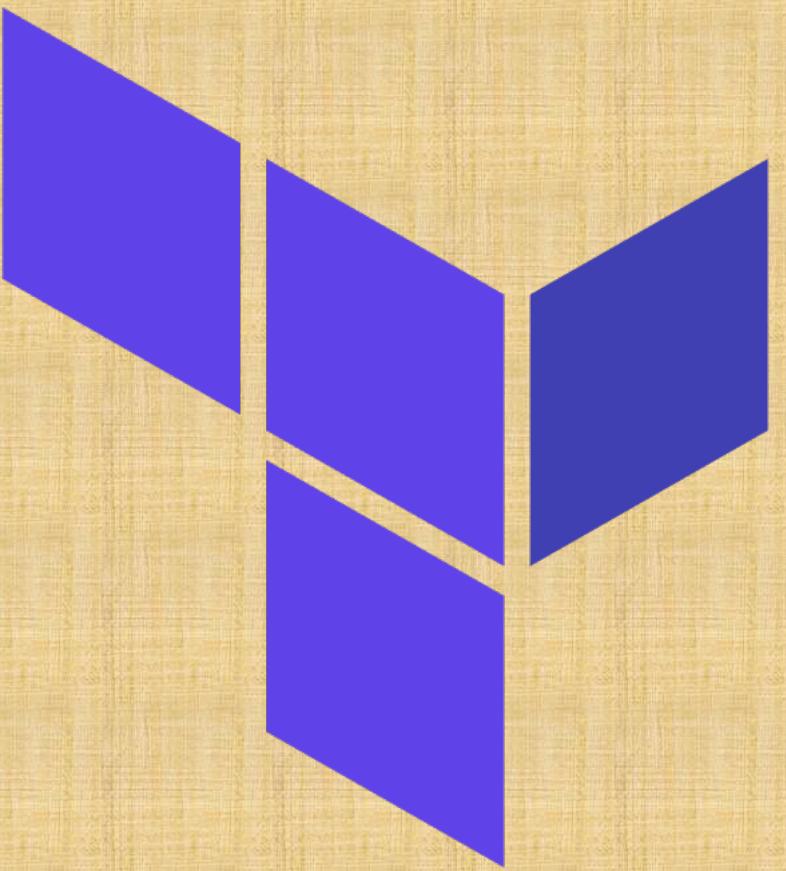
**Resource Behavior:** How Terraform handles resource declarations?

**Provisioners:** We can configure Resource post-creation actions

c1-versions.tf

c2-resource-name.tf

# Terraform Block



# Terraform Block

- This block can be called in 3 ways. All means the same.
  - Terraform Block
  - Terraform Settings Block
  - Terraform Configuration Block
- Each terraform block can contain a number of settings related to Terraform's behavior.
- **VERY VERY IMPORTANT TO MEMORIZE**
  - Within a terraform block, **only constant values can be used**; arguments **may not refer** to named objects such as resources, input variables, etc, and **may not use any** of the Terraform language built-in functions.

# Terraform Block from 0.13 onwards

Terraform 0.12 and earlier:

```
# Configure the AWS Provider
provider "aws" {
    version = "~> 3.0"
    region  = "us-east-1"
}

# Create a VPC
resource "aws_vpc" "example" {
    cidr_block = "10.0.0.0/16"
}
```

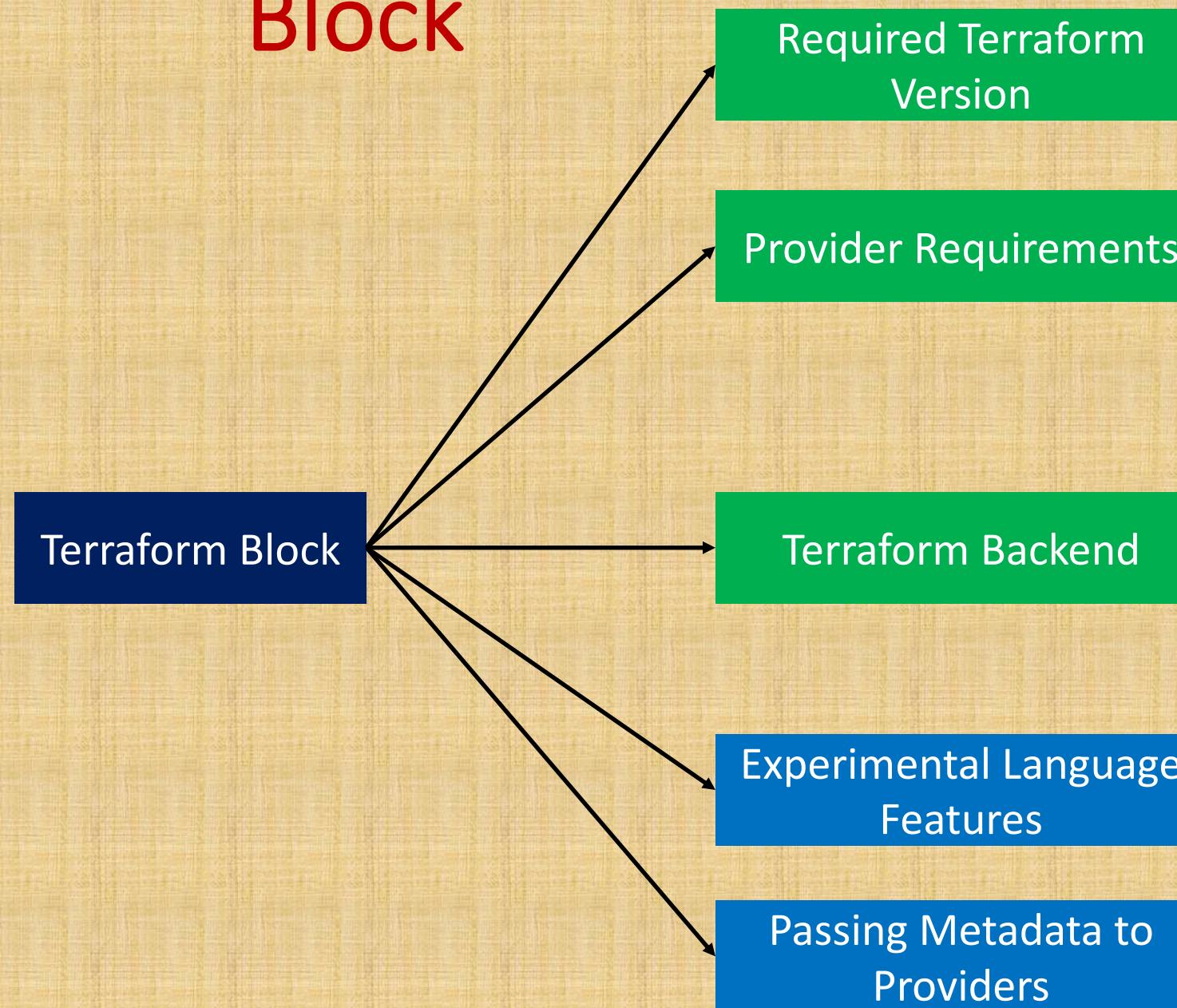
Terraform 0.13 and later:

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 3.0"
    }
  }
}

# Configure the AWS Provider
provider "aws" {
    region = "us-east-1"
}

# Create a VPC
resource "aws_vpc" "example" {
    cidr_block = "10.0.0.0/16"
}
```

# Terraform Block



```
terraform {
  # Required Terraform Version
  required_version = "~> 0.14.3"

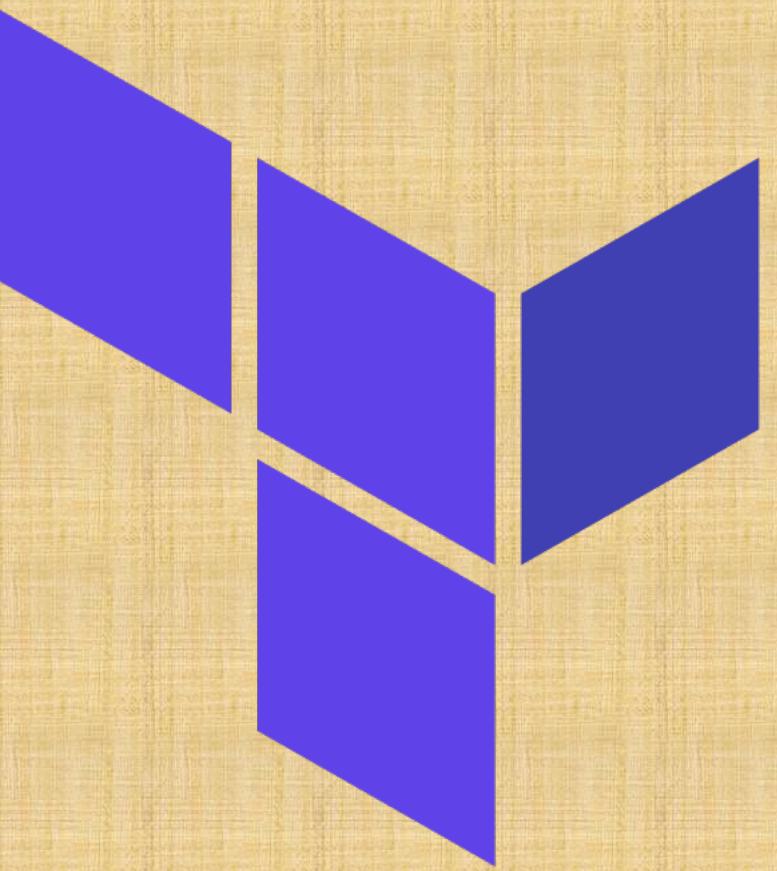
  # Required Providers and their Versions
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 3.21" # Optional but recommended
    }
  }

  # Remote Backend for storing Terraform State in S3 bucket
  backend "s3" {
    bucket = "mybucket"
    key    = "path/to/my/key"
    region = "us-east-1"
  }

  # Experimental Features (Not required)
  experiments = [ example ]

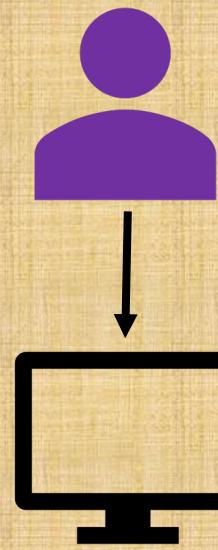
  # Passing Metadata to Providers (Super Advanced – Terraform 0.12+)
  provider_meta "my-provider" {
    hello = "world"
  }
}
```

# Terraform Providers



# Terraform Providers

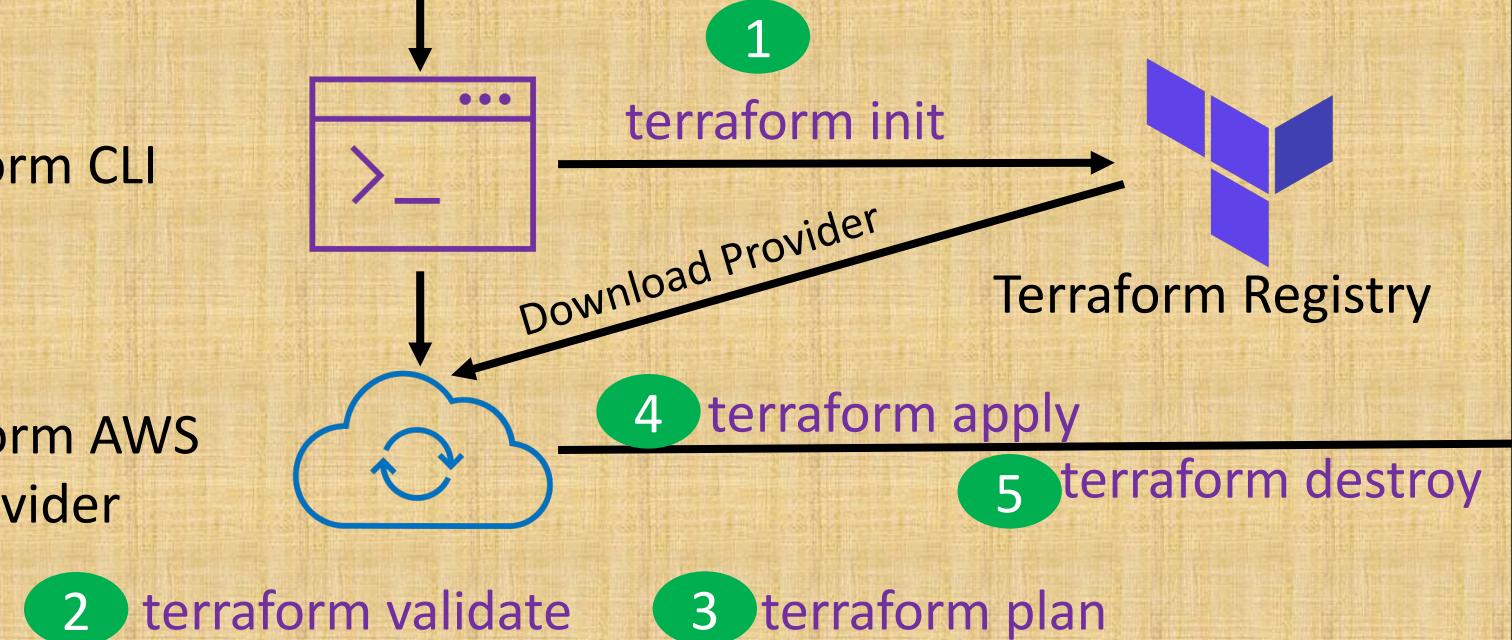
Terraform Admin



Local Desktop

Terraform CLI

Terraform AWS Provider



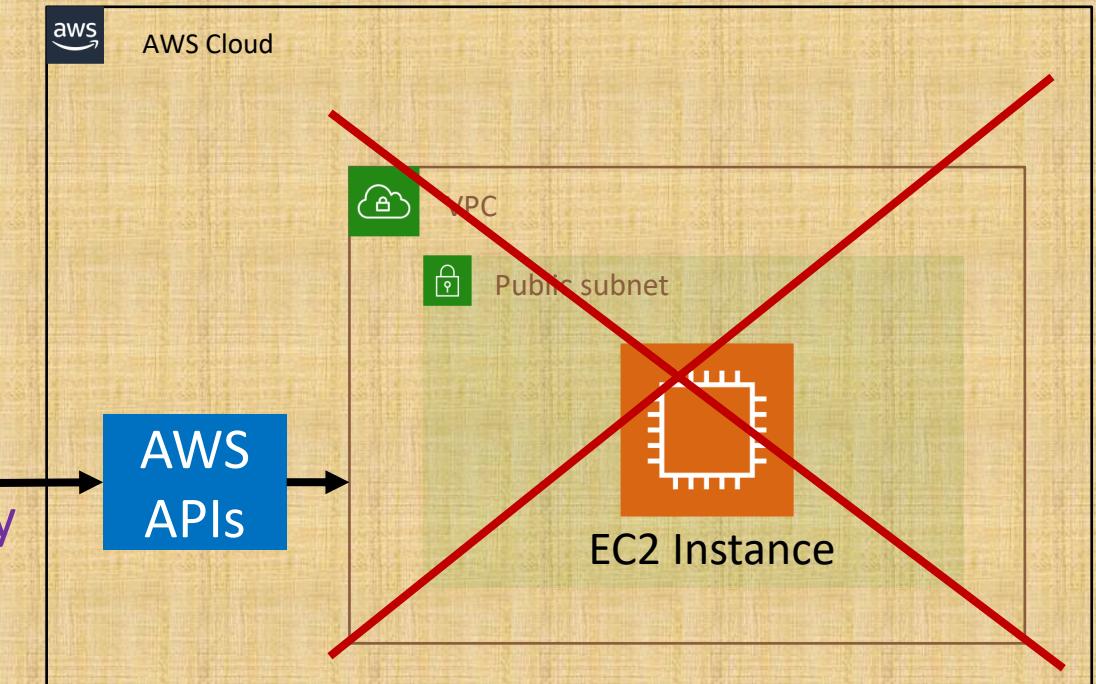
Providers are **HEART** of Terraform

Every **Resource Type** (example: EC2 Instance), is implemented by a Provider

Without Providers Terraform **cannot** manage any infrastructure.

Providers are distributed separately from Terraform and each provider has its own **release cycles** and **Version Numbers**

Terraform **Registry** is publicly available which contains many Terraform Providers for most **major** Infra Platforms



## Terraform Providers

### Provider Requirements

```
# Terraform Block
terraform {
  required_version = "~> 0.14.3"
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "~> 3.0"
    }
  }
}
```

### Provider Configuration

```
# Provider Block
provider "aws" {
  profile = "default"
  region  = "us-east-1"
}
```

### Dependency Lock File

```
└─ terraform-manifests
    └─ .terraform
        └─ .terraform.lock.hcl
    └─ ec2-instance.tf
    └─ terraform.tfstate
    └─ terraform.tfstate.backup

# This file is maintained automatically by "terraform init".
# Manual edits may be lost in future updates.

provider "registry.terraform.io/hashicorp/aws" {
  version = "3.22.0"
  hashes = [
    "h1:f/Tz8zv1zb78ZaiyJk0OMGIViZwbYrLuQk3kojPM91c=",
    "zh:4a9a66caf1964cd3b61fb3eb0da417195a529ccb8e496f266b0778335d11c8",
    "zh:514f2f006ae68db715d86781673faf9483292deab235c7402ff306e0e92ea11a",
    "zh:5277b61109fddb9011728f6650ef01a639a0590aeffe34ed7de7ba10d0c31803",
    "zh:67784dc8375ab37103eea1258c3334ee92be6de033c2b37e3a2a65d0005142",
    "zh:76d4c8be2ca4a3294fb51fb58de1fe03361d3bc403820270cc8e71a04c5fa006",
    "zh:8f900b1cfdf6e8fb1a9d0382ecaa5056a3a84c94e313fb9e92c89de271cdede",
    "zh:d0ac346519d0df124df89be2d803eb53f373434890f6ee3fb27112802f9eac59",
    "zh:d6256feedada82cbfb3b1dd6dd9ad02048f23120ab50e6146a541cb11a108cc1",
    "zh:db2fe0d2e77c02e9a74e1ed694aa352295a50283f9a1cf896e5be252af14e9f4",
    "zh:eda61e889b579bd90046939a5b40cf5dc9031fb5a819fc3e4667a78bd432bdb2",
  ]
}
```

# Dependency Lock File

```
# This file is maintained automatically by "terraform init".  
# Manual edits may be lost in future updates.  
  
provider "registry.terraform.io/hashicorp/aws" {  
    version = "3.22.0"
```

# Required Providers

```
# Terraform Block
terraform {
  required_version = "~> 0.14.3"
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "~> 3.0"
    }
  }
}
```

```
# Provider Block
provider "aws" {
  profile = "default"
  region  = "us-east-1"
}
```

## Local Names

Local Names are **Module specific** and should be **unique per-module**

Terraform configurations always refer to **local name** of provider **outside** required\_provider block

Users of a provider can choose **any local name** for it (myaws, aws1, aws2).

Recommended way of choosing local name is to use preferred local name of that provider (For AWS Provider: hashicorp/aws, **preferred local name** is aws)

## Source

It is the **primary location** where we can download the Terraform Provider

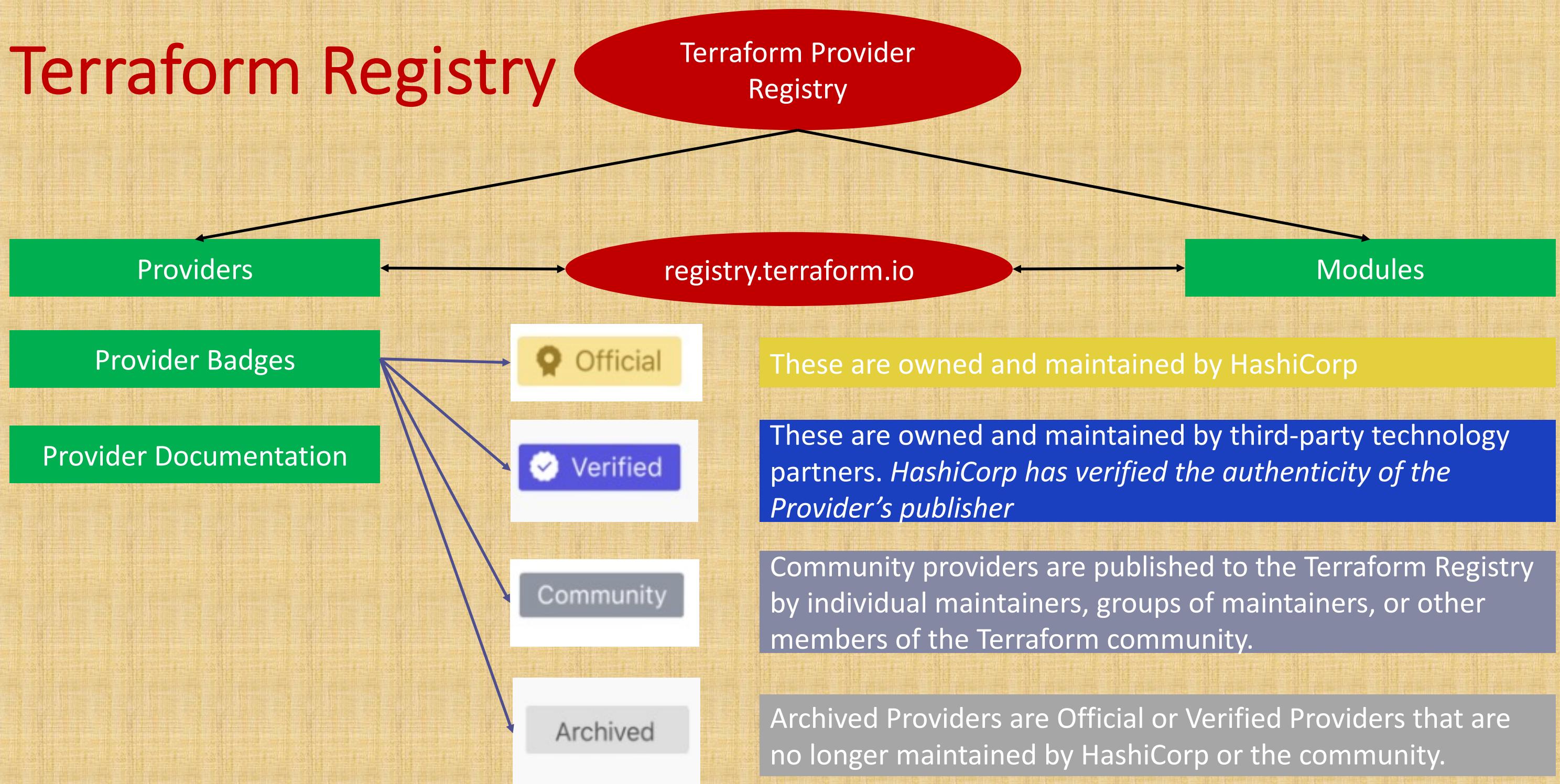
Source addresses consist of **three parts** delimited by **slashes (/)**

[<HOSTNAME>/]<NAMESPACE>/<TYPE>

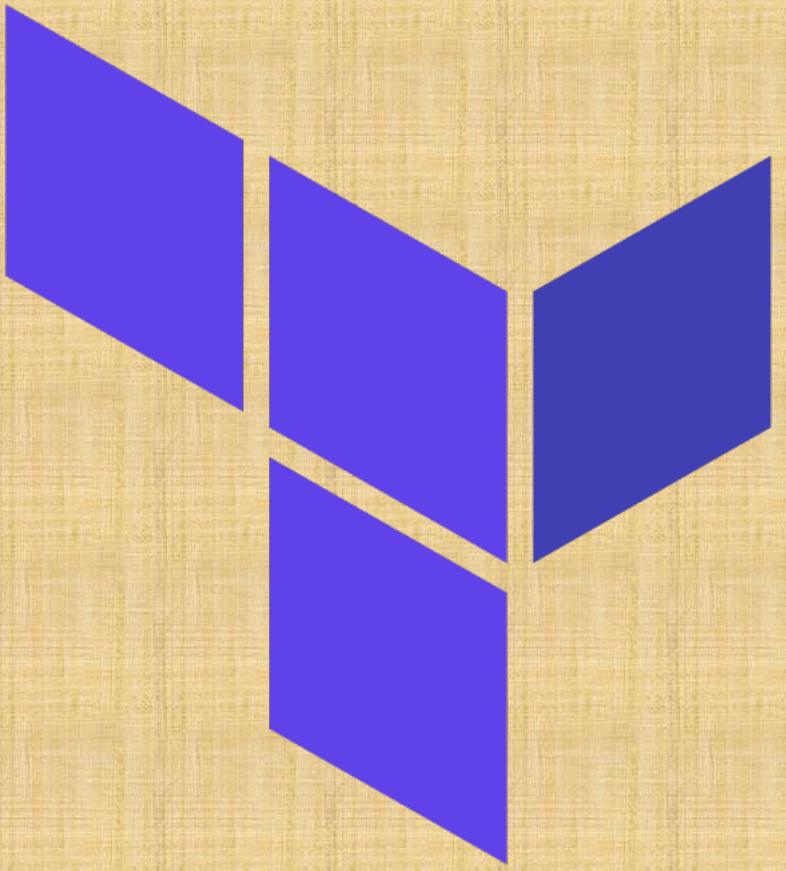
registry.terraform.io/hashicorp/aws

Registry Name is **optional** as default is going to be Terraform Public Registry

# Terraform Registry



# Terraform Resources Introduction



# Terraform Language Basics – Configuration Syntax

```
# Template
<BLOCK TYPE> "<BLOCK LABEL>" "<BLOCK LABEL>"  {
    # Block body
    <IDENTIFIER> = <EXPRESSION> # Argument
}

# AWS Example
resource "aws_instance" "ec2demo" {
    ami           = "ami-04d29b6f966df1537"
    instance_type = "t2.micro"
}
```

Block Type

Top Level &  
Block inside  
Blocks

**Top Level Blocks:** resource, provider

**Block Inside Block:** provisioners,  
resource specific blocks like tags

Arguments

Block Labels

Based on Block  
Type block labels  
will be 1 or 2

**Example:**  
Resource – 2  
labels

Variables – 1 label

# Resource Syntax

**Resource Type:** It determines the kind of **infrastructure object** it manages and what arguments and other attributes the resource supports.

**Resource Local Name:** It is used to refer to this resource from elsewhere in the same Terraform module, but has **no significance outside** that module's scope.  
The resource type and name together serve as an identifier for a given resource and so must be **unique** within a module

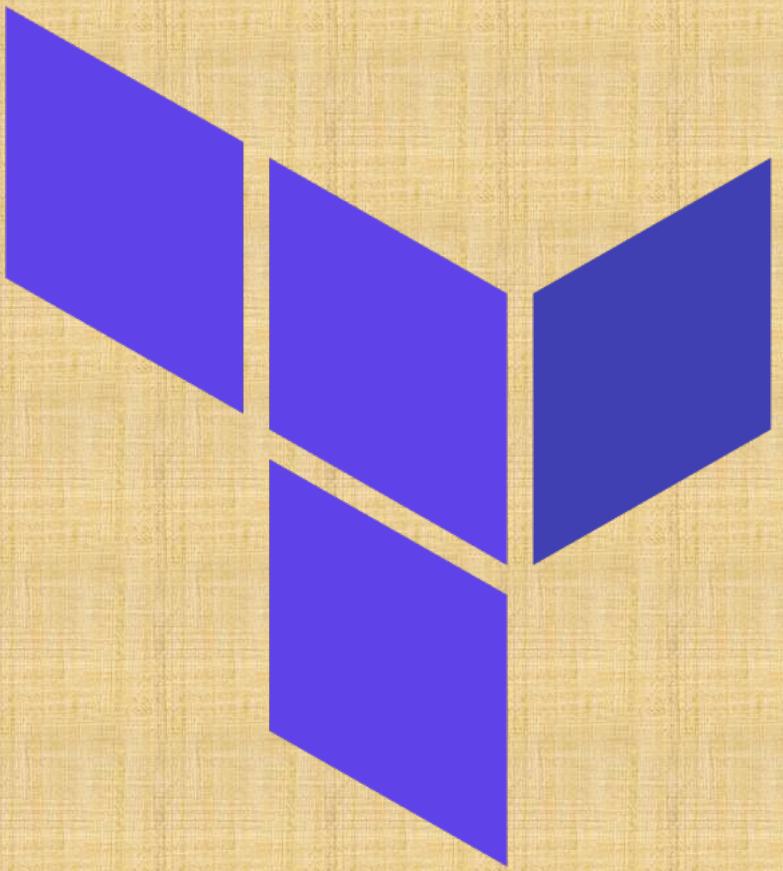
**Meta-Arguments:** Can be used with any resource to change the behavior of resources

**Resource Arguments:** Will be specific to resource type. Argument Values can make use of **Expressions** or other Terraform **Dynamic Language Features**

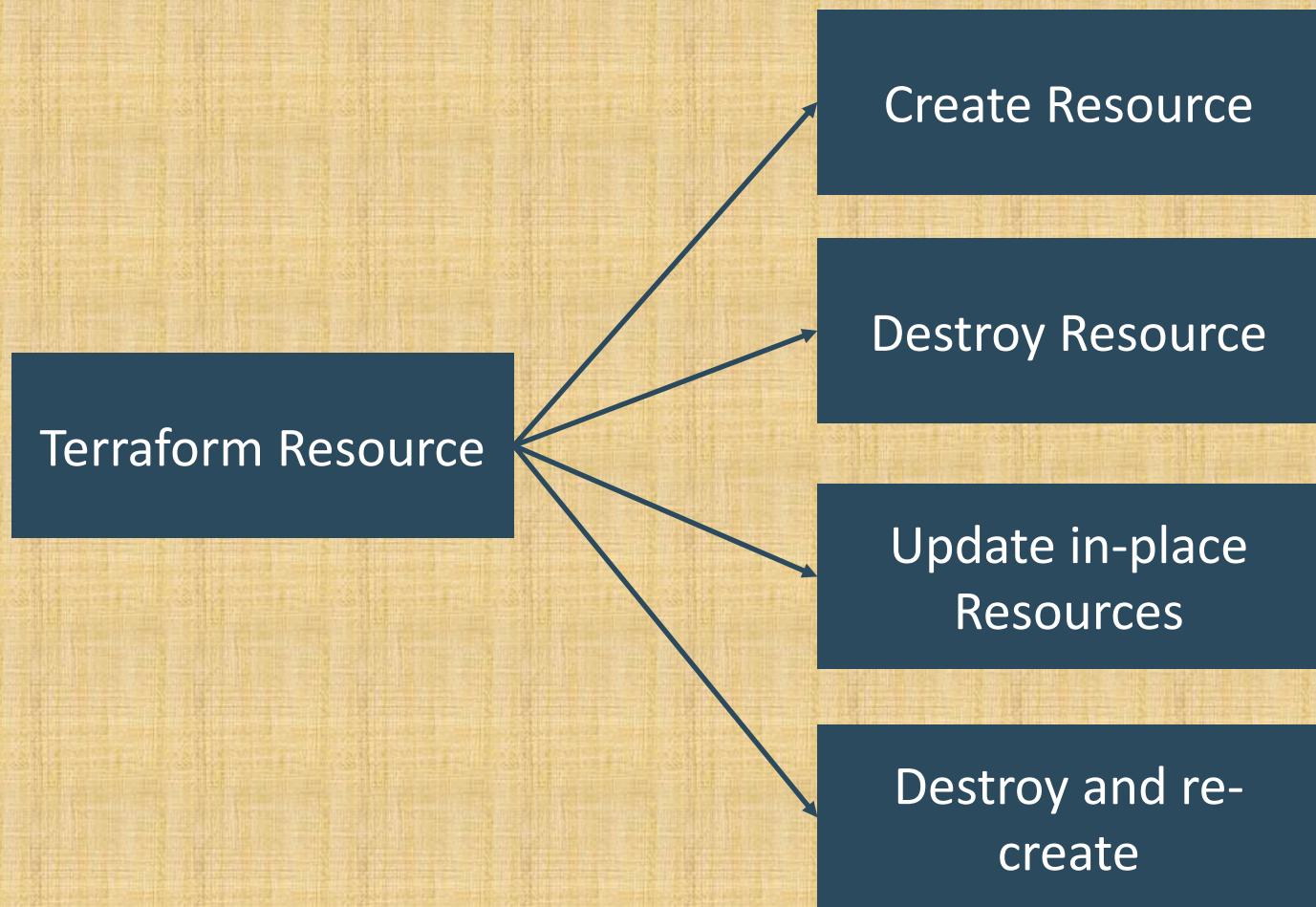
```
# Provider-2 for us-west-1
provider "aws" {
  region = "us-west-1"
  profile = "default"
  alias   = "aws-west-1"
}

# Resource Block to Create VPC
resource "aws_vpc" "vpc_us-west-1" {
  provider = aws.aws-west-1
  cidr_block = "10.2.0.0/16"
  tags = {
    "Name" = "vpc-1"
  }
}
```

# Terraform State



# Resource Behavior



## Terraform State

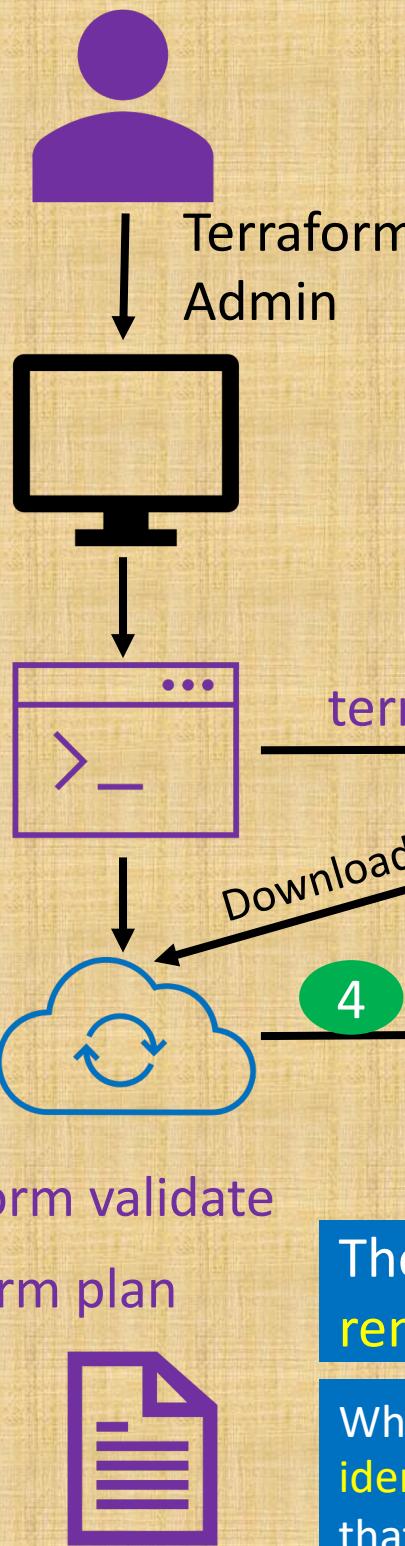
# Terraform State

Local Desktop

Terraform CLI

Terraform AWS Provider

Terraform State File  
`terraform.tfstate`



Terraform must **store state** about your managed infrastructure and configuration

This state is used by Terraform to map **real world resources** to your **configuration (.tf files)**, keep track of metadata, and to improve performance for large infrastructures.

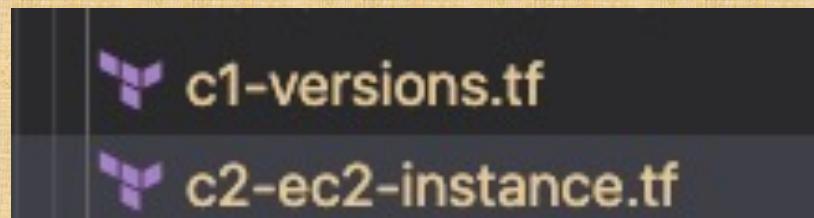
This state is stored by default in a local file named "**terraform.tfstate**", but it can also be stored **remotely**, which works better in a **team** environment.

The **primary purpose** of Terraform state is to store **bindings** between objects in a **remote system** and resource instances **declared** in your configuration.

When Terraform creates a remote object in response to a change of configuration, it will record the **identity** of that remote object against a particular resource instance, and then **potentially update or delete** that object in response to future configuration changes.

# Desired & Current Terraform States

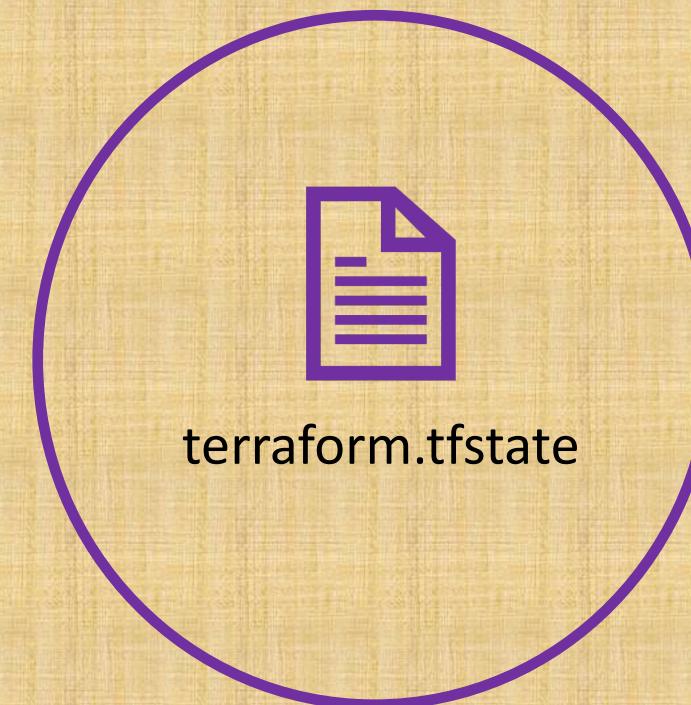
Terraform Configuration Files



Real World Resource – EC2 Instance

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
web	i-0663449fef49e9cf5	Running	t2.micro	2/2 checks ...	No alarms +	us-east-1b
Instance: i-0663449fef49e9cf5 (web)						
Details	Security	Networking	Storage	Status checks	Monitoring	Tags
Instance ID i-0663449fef49e9cf5 (web)	Public IPv4 address 54.144.73.100   <a href="#">open address</a>	Private IPv4 addresses 172.31.94.137	Instance state Running	Public IPv4 DNS ec2-54-144-73-100.compute-1.amazonaws.com   <a href="#">open address</a>	Private IPv4 DNS ip-172-31-94-137.ec2.internal	Instance type t2.micro
Elastic IP addresses –	IAM Role –	VPC ID vpc-54972d2e (default-vpc)	AWS Compute Optimizer finding Opt-in to AWS Compute Optimizer for recommendations.	Subnet ID subnet-d2e590fc		

Desired State



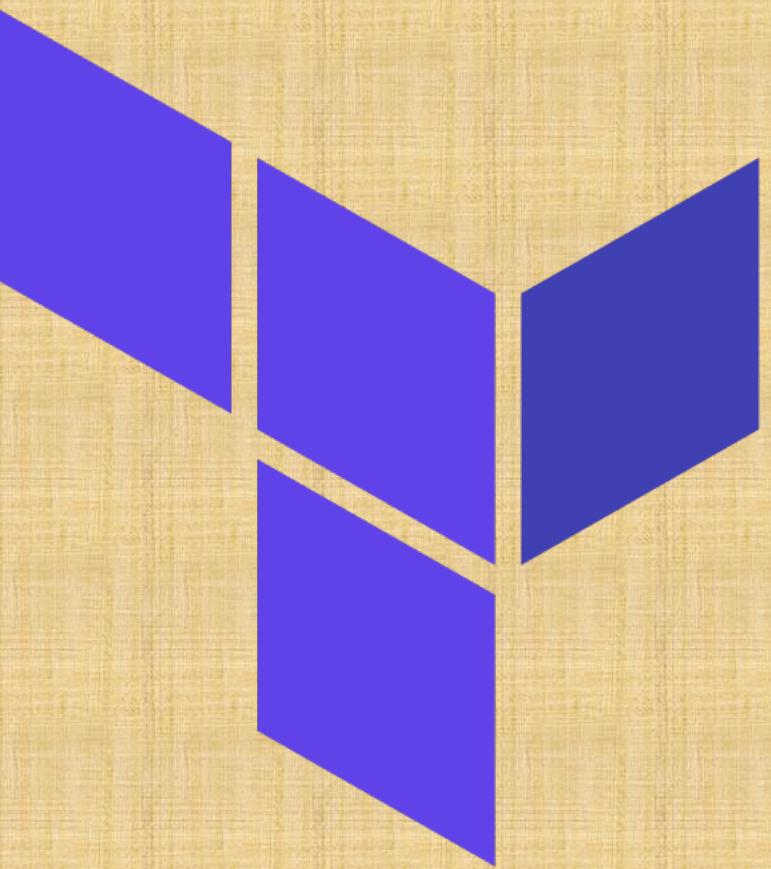
Current State

# Terraform

## Input Variables

## Datasources

## Outputs



# What are we going to learn ?

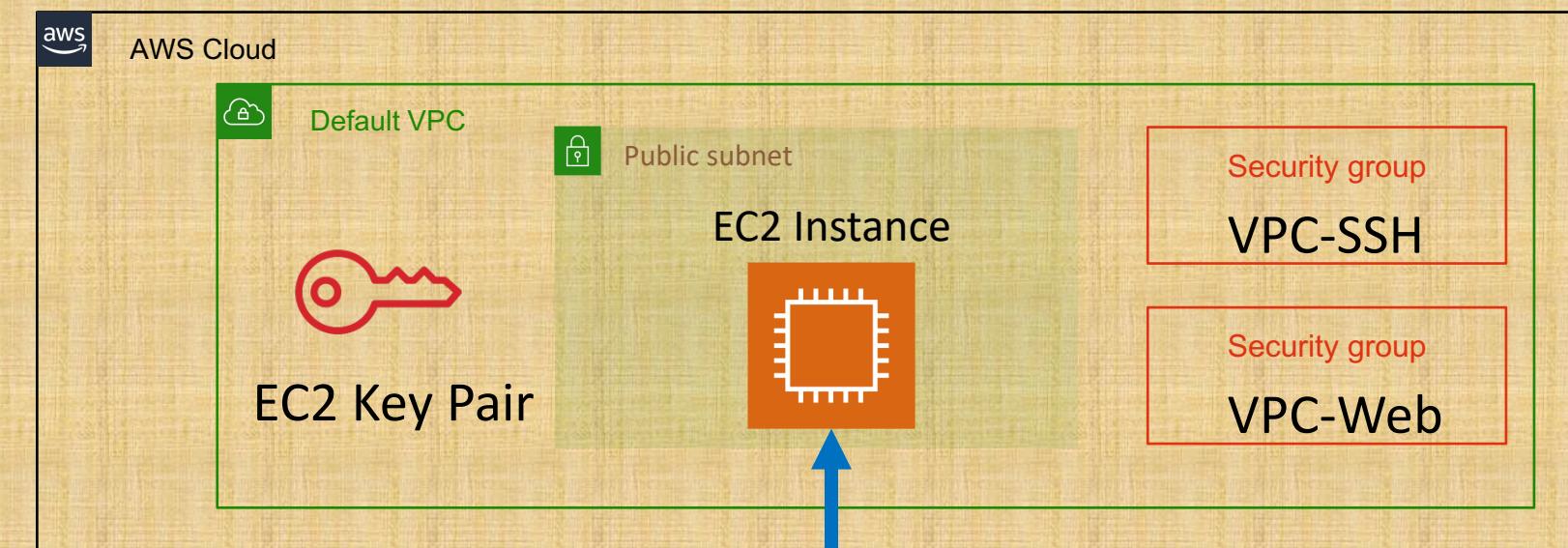
Terraform  
Concepts

Terraform Input Variables

Terraform Output Values

Terraform Datasources

AWS Services



Dynamically get latest AMI ID

# Terraform Variables

Terraform  
Input  
Variables

Terraform  
Output  
Values

Terraform  
Local  
Values

# Terraform Input Variables

Input variables serve as **parameters** for a Terraform module, allowing aspects of the module to be **customized** without altering the module's own source code, and allowing modules to be **shared** between different configurations.

Input Variables - Basics

1

Provide Input Variables when prompted during **terraform plan** or **apply**

2

Override default variable values using CLI argument **-var**

3

Override default variable values using Environment Variables (**TF\_var\_aa**)

4

Provide Input Variables using **terraform.tfvars** files

5

Provide Input Variables using **<any-name>.tfvars** file with CLI argument **-var-file**

7

Provide Input Variables using **auto.tfvars** files

8

Implement complex type **constructors** like **List & Map** in Input Variables

9

Implement **Custom Validation Rules** in Variables

10

Protect **Sensitive** Input Variables

Terraform  
Input  
Variables

# Terraform Datasources

*Data sources* allow data to be fetched or computed for use elsewhere in Terraform configuration.

Use of data sources allows a Terraform configuration to make use of information defined outside of Terraform, or defined by another separate Terraform configuration.

A data source is accessed via a special kind of resource known as a *data resource*, declared using a **data** block

Each data resource is associated with a single data source, which determines the kind of object (or objects) it reads and what **query constraint arguments** are available

Data resources have the **same dependency resolution behavior** as defined for managed resources. Setting the **depends\_on** meta-argument within data blocks **defers** reading of the data source until after all changes to the dependencies have been applied.

```
# Get latest AMI ID for Amazon Linux2 OS
data "aws_ami" "amzlinux" {
    most_recent      = true
    owners           = ["amazon"]
    filter {
        name   = "name"
        values = ["amzn2-ami-hvm-*"]
    }
    filter {
        name   = "root-device-type"
        values = ["ebs"]
    }
    filter {
        name   = "virtualization-type"
        values = ["hvm"]
    }
    filter {
        name   = "architecture"
        values = ["x86_64"]
    }
}
```

# Terraform Datasources

We can refer the data resource in a resource as depicted

## Meta-Arguments for Datasources

```
# Create EC2 Instance - Amazon Linux
resource "aws_instance" "my-ec2-vm" {
    ami           = data.aws_ami.amzlinux.id
    instance_type = var.ec2_instance_type
    key_name      = "terraform-key"
    user_data     = file("apache-install.sh")
    vpc_security_group_ids = [aws_security_group...
    tags = {
        "Name" = "amz-linux-vm"
    }
}
```

Data resources support the **provider** meta-argument as defined for managed resources, with the **same syntax** and behavior.

Data resources **do not currently have** any customization settings available for their **lifecycle**, but the **lifecycle** nested block is **reserved** in case any are added in future versions.

Data resources support **count** and **for\_each** meta-arguments as defined for managed resources, with the **same syntax** and behavior.

Each instance will **separately read** from its data source with its own variant of the constraint arguments, producing an **indexed result**.

# Terraform Variables – Output Values

Output values are like the return values of a Terraform module and have several uses

1

A root module can use outputs to **print** certain values in the **CLI** output after running **terraform apply**.

## Terraform Variables Outputs

2

A child module can use outputs to **expose a subset** of its resource attributes to a **parent module**.

When using **remote state**, root module outputs can be accessed by other configurations via a **terraform\_remote\_state** data source.

3

Advanced

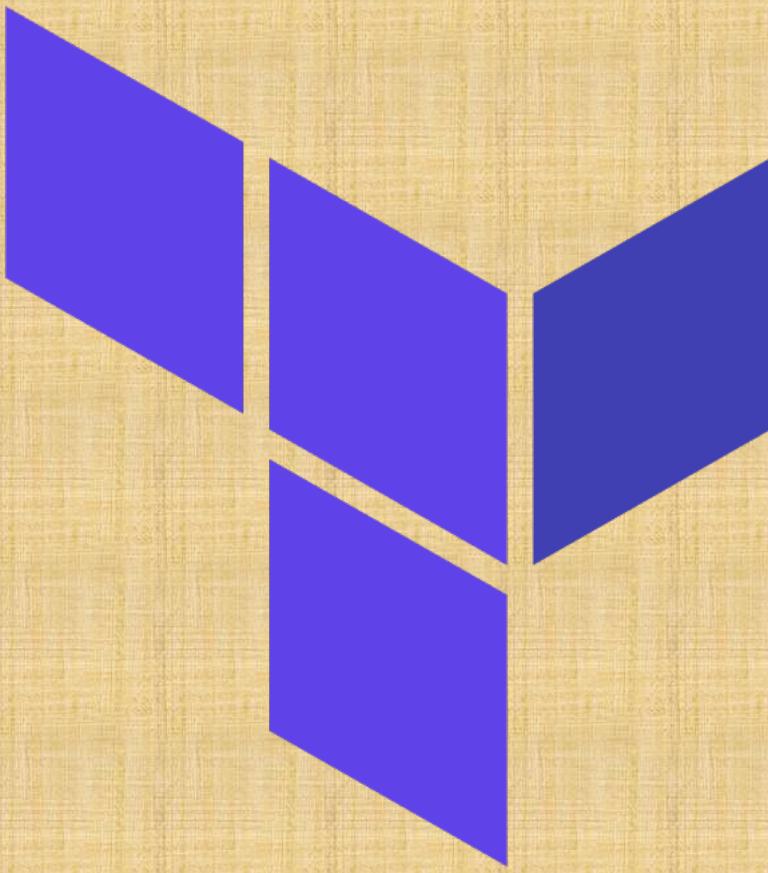
# Terraform

## For Loops

## Lists & Maps

## Meta-Argument

## Count & for\_each



## Section-1

Meta-Argument: count

Variables: Lists & Maps

For Loop with Lists

For Loop with Maps

For Loops with Advanced Maps

Legacy Splat Operator (.\*.)

Latest Splat Operator [\*]

## Section-2

Meta-Argument: for\_each

Function: toset

Function: tomap

Datasource:  
aws\_availability\_zones

## Section-4

Fix issues in Section-2 with  
section-3

Final Output

## Section-3

Datasource:  
aws\_ec2\_instance\_type\_offerings

Datasource:  
aws\_availability\_zones

For Loop with Maps

For Loop with if

Function: keys

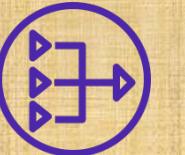
Utility Project  
with  
Datasources



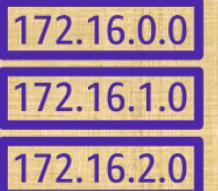
Amazon Virtual Private Cloud  
(Amazon VPC)



Internet gateway



NAT gateway



Route table



Public Subnet

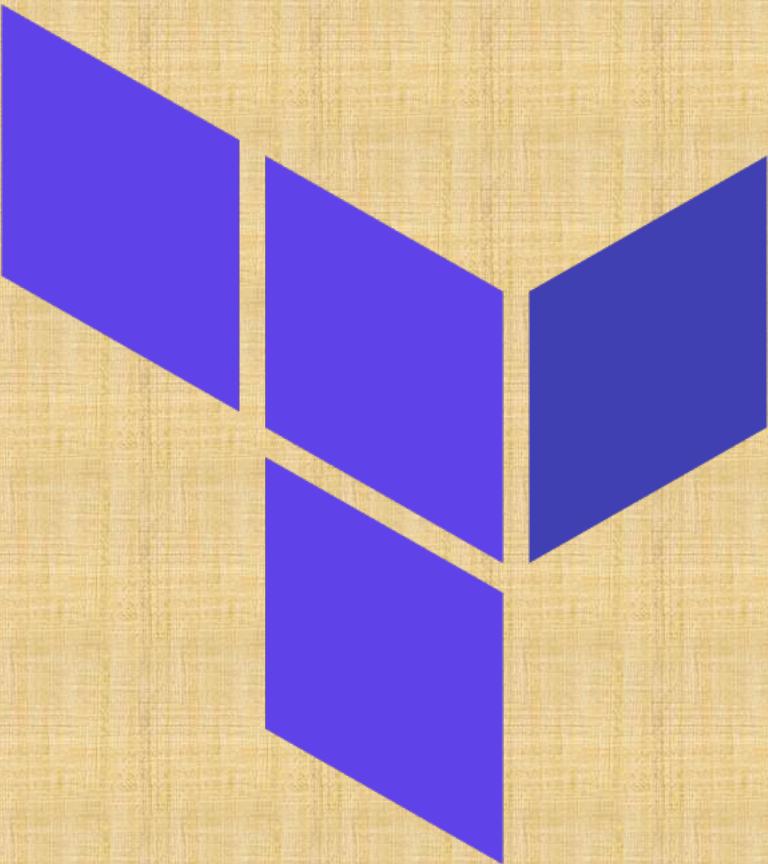


Private Subnet

# AWS VPC

## 3-Tier

# Web, App and DB



# AWS VPC 3-Tier Architecture

Build manually using AWS Mgmt Console

Build same using Terraform

Terraform Modules

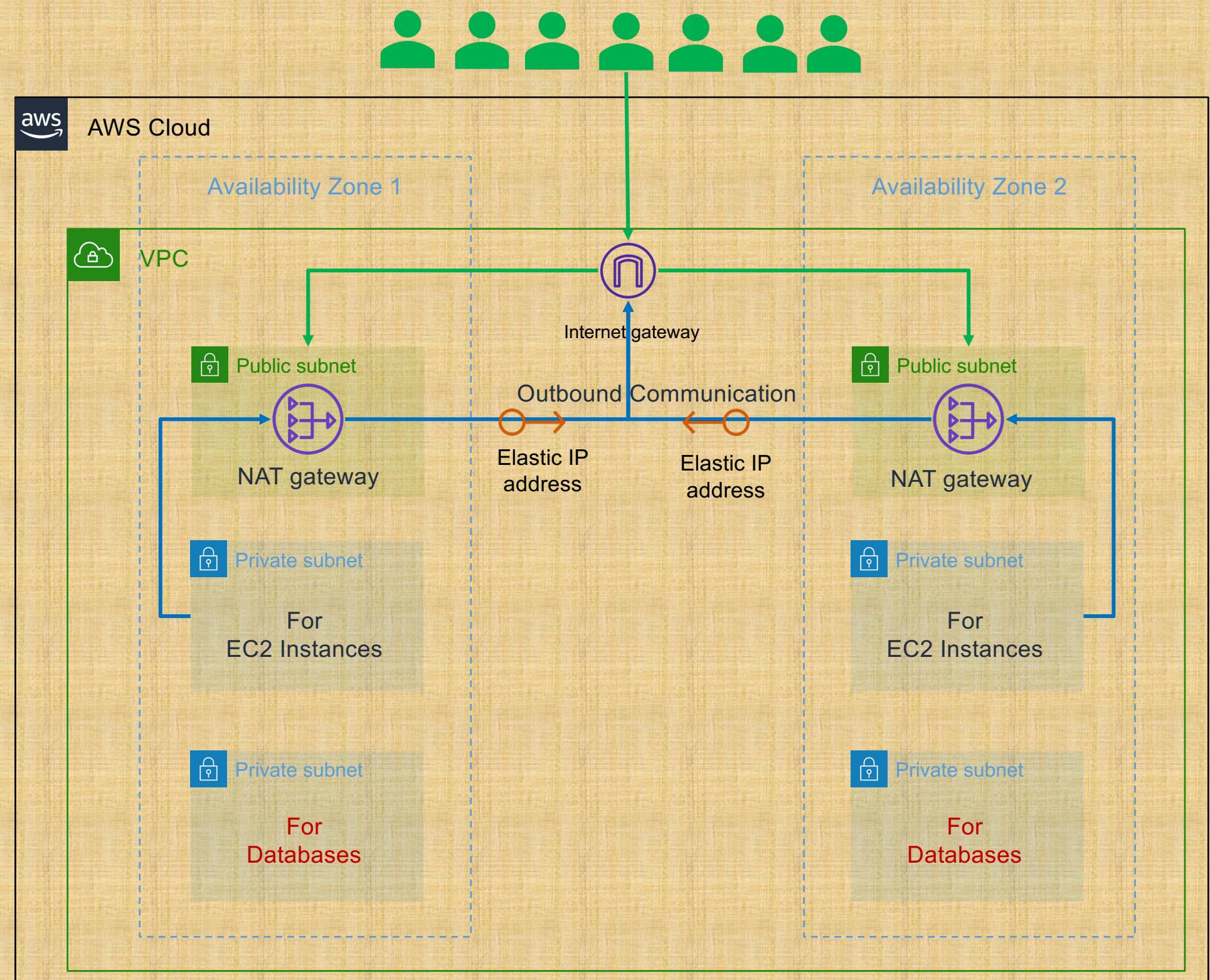
Terraform Local Values

Terraform Variables – `terraform.tfvars`

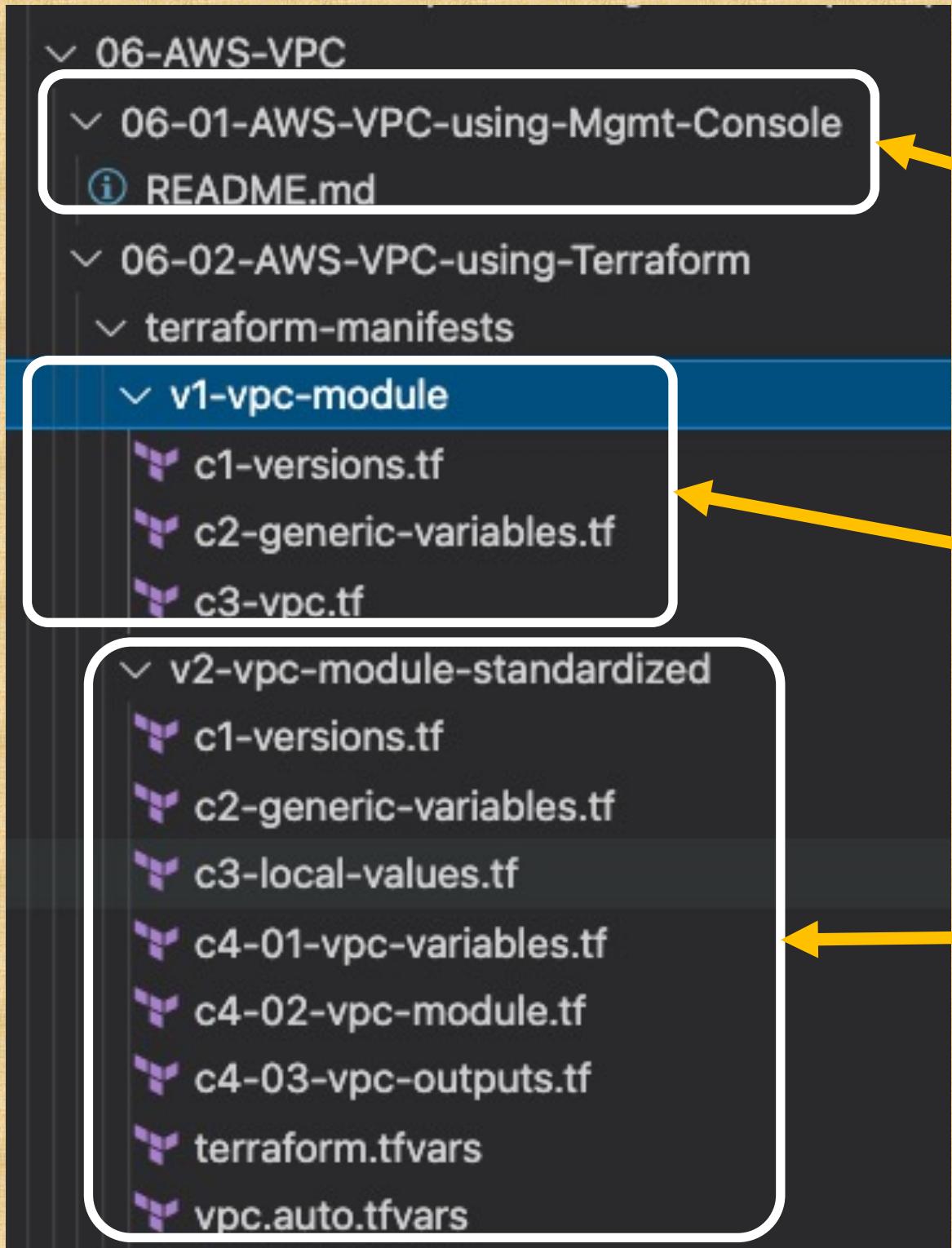
Terraform Variables – `vpc.auto.tfvars`

Terraform Version Constraints

Terraform Code Organizing – Production Grade style



# What are we going to Learn ?



Build VPC manually using AWS Management Console

Build VPC using Terraform Modules

Standardize the Terraform Code at Production Grade

# Terraform Modules

Modules are **containers for multiple resources** that are used together. A module consists of a collection of .tf files kept together in a directory.

Modules are the main way to **package** and reuse resource configurations with Terraform.

Every Terraform configuration has at least one module, known as its **root module**, which consists of the resources defined in the .tf files in the **main working directory**.

A Terraform module (usually the **root module** of a configuration) can call **other modules** to include their resources into the configuration.

A module that has been called by another module is often referred to as a **child module**.

Child modules **can be called multiple times** within the same configuration, and **multiple configurations** can use the same child module.

In addition to modules from **the local filesystem**, Terraform can load modules from a **public or private registry**.

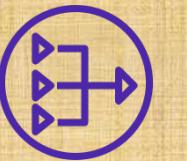
This makes it possible to **publish modules** for others to **use**, and to use modules that others have published.



Amazon Virtual Private Cloud  
(Amazon VPC)



Internet gateway



NAT gateway



Elastic IP  
address



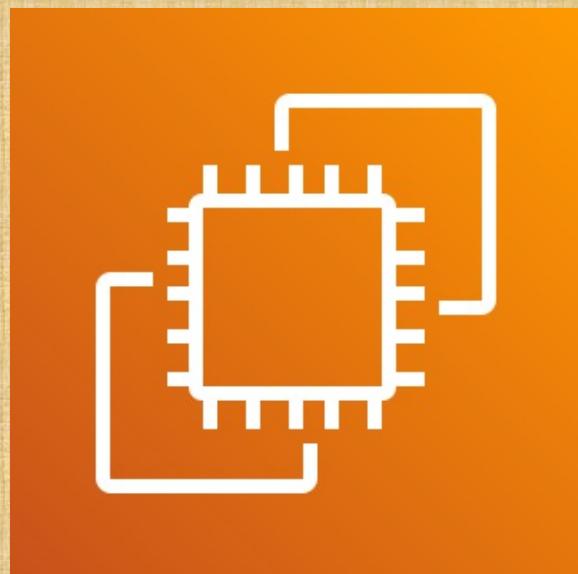
Route table



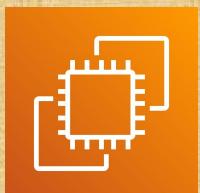
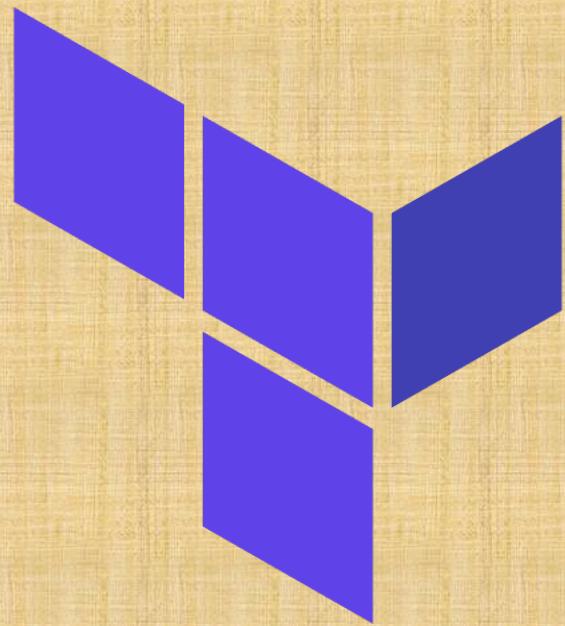
Public Subnet



Private Subnet



# AWS EC2 Bastion Host Terraform Provisioners



EC2 VM

# AWS VPC + EC2 Instance + Security Groups



## Terraform & AWS Concepts

Terraform Module: **VPC**

Terraform Datasource:  
`aws_availability_zones` (Dynamic)

Terraform Module: **Security Group**

Terraform Module: **AWS EC2 Instance**

Meta-Argument: `depends_on`

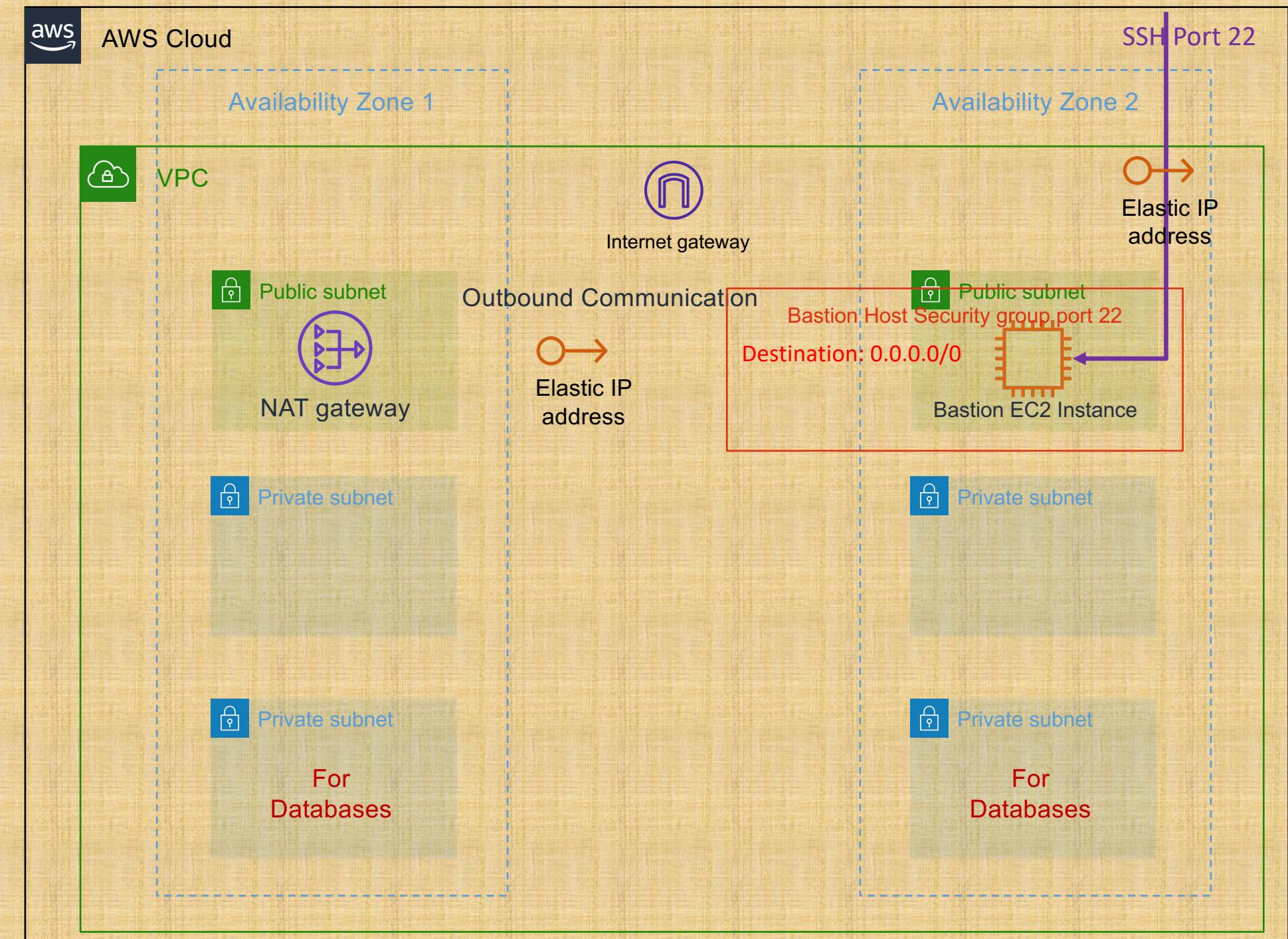
Terraform `null_resource`

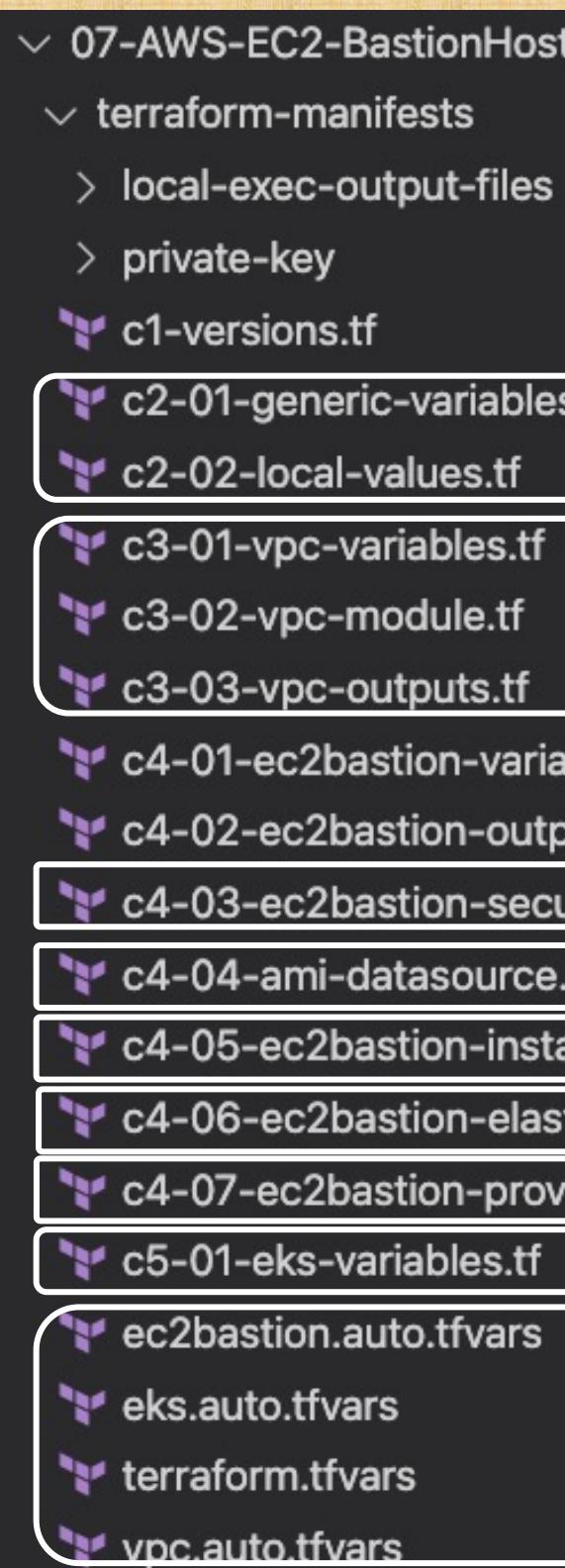
Terraform `File Provisioner`

Terraform `Remote-exec Provisioner`

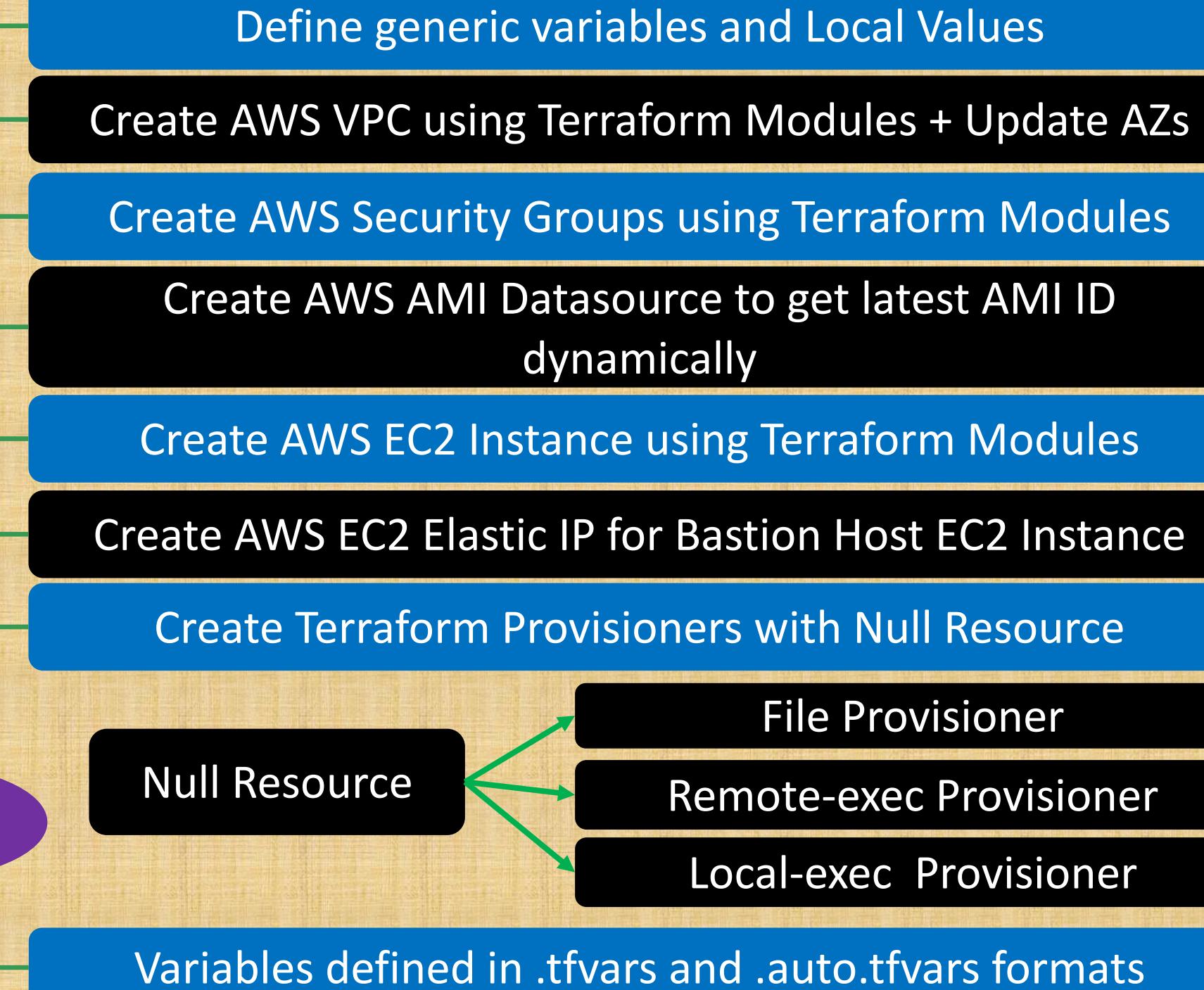
Terraform `Local-exec Provisioner`

EKS Cluster `Tags` for VPC Subnets

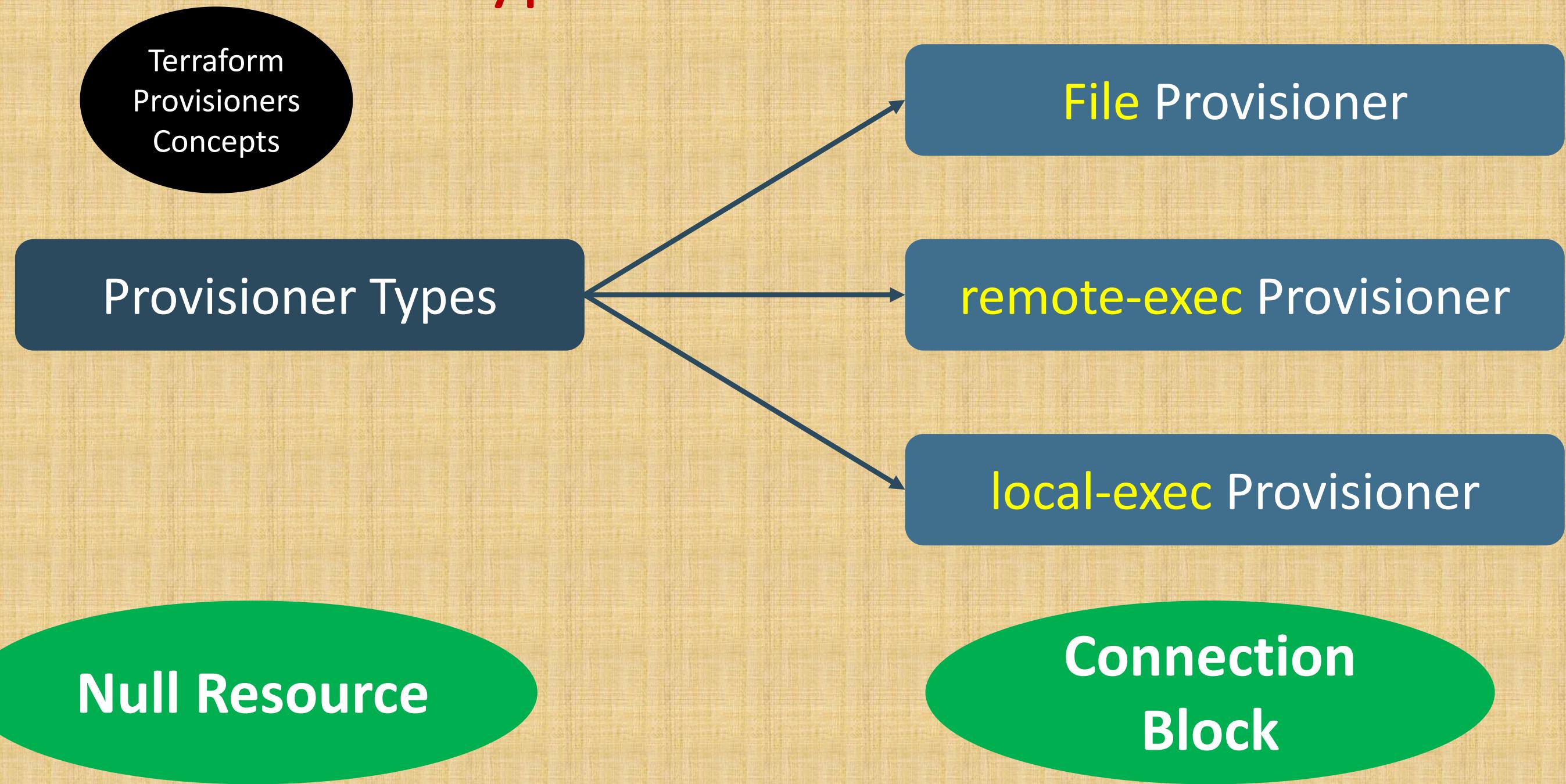




# What are we going to learn ?



# Types of Provisioners



# Terraform Provisioners

Provisioners can be used to model specific actions on the local machine or on a remote machine in order to prepare servers

Passing data into virtual machines and other compute resources

Running configuration management software (packer, chef, ansible)

Creation-Time Provisioners

Failure Behaviour: Continue: Ignore the error and continue with creation or destruction.

Connection Block

Most provisioners require access to the remote resource via SSH or WinRM, and expect a nested connection block with details about how to connect.

Provisioners are a Last Resort

First-class Terraform provider functionality may be available

Destroy-Time Provisioners

Failure Behaviour: Fail: Raise an error and stop applying (the default behavior). If creation provisioner fails, taint resource

# Provisioners Introduction

## File Provisioner

- File Provisioner is used to **copy files or directories** from the **machine executing Terraform** to the **newly created resource**.
- The file provisioner supports both **ssh** and **winrm** type of connections

## remote-exec Provisioner

- The **remote-exec** provisioner **invokes a script on a remote resource after it is created**.
- This can be used to **run a configuration management tool, bootstrap** into a cluster, etc.

## local-exec Provisioner

- The **local-exec** provisioner **invokes a local executable** after a resource is **created**.
- This **invokes a process** on the machine running **Terraform**, not on the resource.

## null\_resource

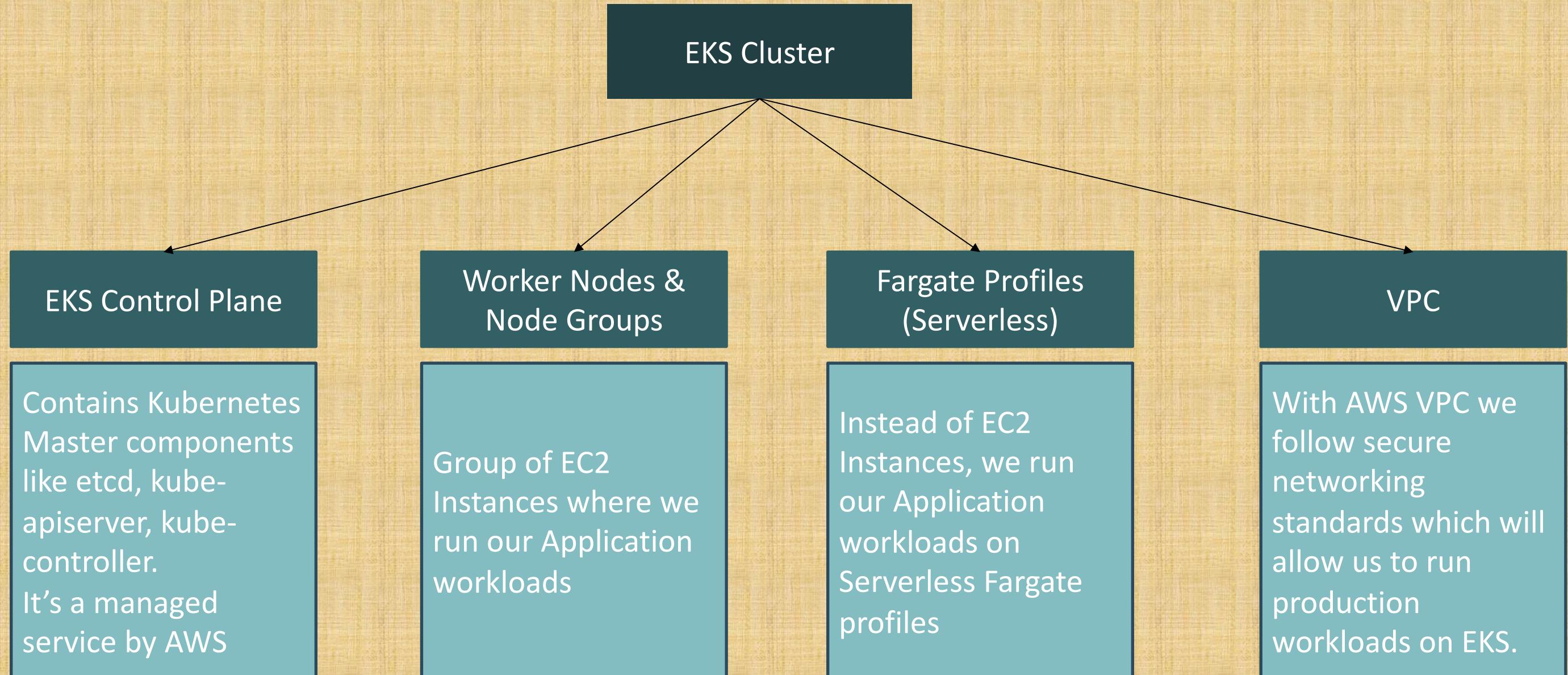
- If you need to run provisioners **that aren't directly associated** with a specific resource, you can associate them with a **null\_resource**.
- Instances of **null\_resource** are treated like normal resources, but they **don't do anything**.
- Same as other resource, you can **configure provisioners and connection details** on a null\_resource.



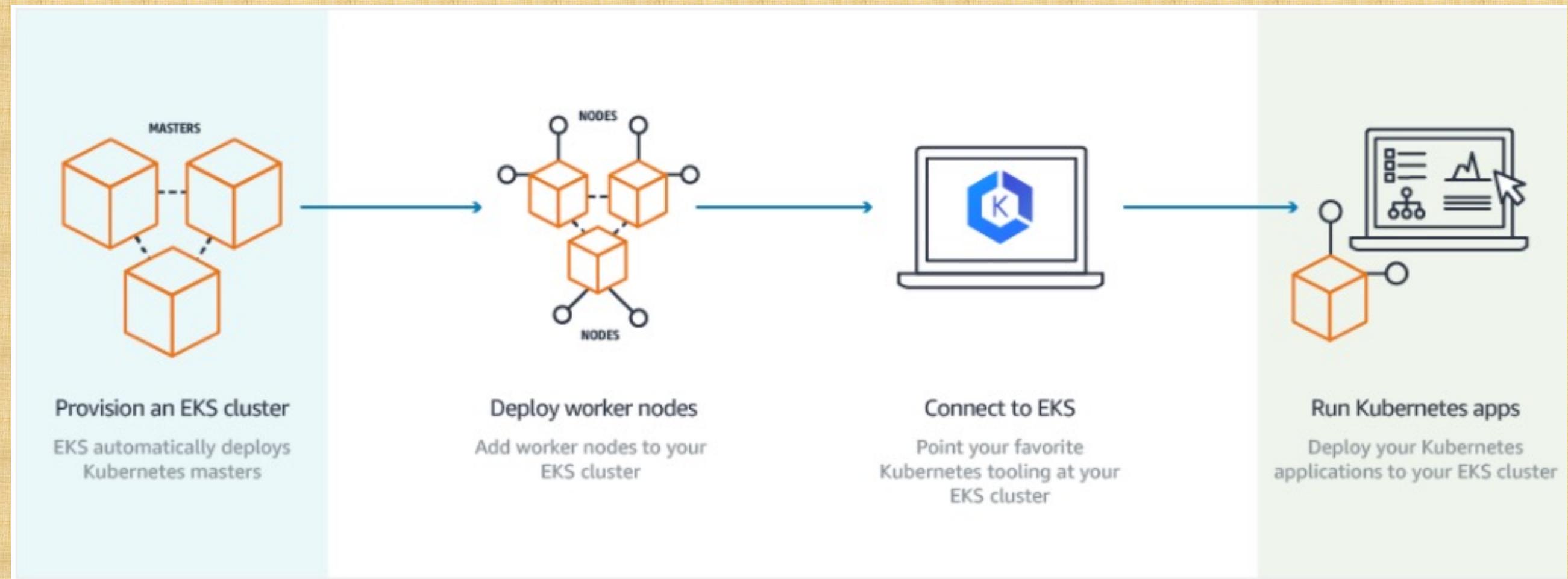
# AWS EKS Cluster



# AWS EKS – Core Objects



# How does EKS work?



# EKS Cluster – Core Objects Detailed

## EKS Control Plane

1. EKS runs a single tenant Kubernetes control plane for each cluster, and control plane infrastructure is **not shared** across clusters or AWS accounts.
2. This control plane consists of at least two API server nodes and three etcd nodes that run across **three Availability Zones within a Region**
3. EKS automatically detects and replaces **unhealthy** control plane instances, restarting them across the Availability Zones within the Region as needed.

## Worker Nodes & Node Groups

1. Worker machines in Kubernetes are called nodes. These are EC2 Instances
2. EKS worker nodes run in our AWS account and connect to our cluster's control plane via the **cluster API server endpoint**.
3. A node group is **one or more** EC2 instances that are deployed in an EC2 Autoscaling group.
4. All instances in a node group must
  1. Be the **same instance type**
  2. Be **running the same AMI**
  3. Use the **same EKS worker node IAM role**

# EKS Cluster – Core Objects Detailed

## Fargate Profiles

1. AWS Fargate is a technology that provides **on-demand**, right-sized compute capacity for containers
2. With Fargate, we **no longer** have to provision, configure, or scale groups of virtual machines to run containers.
3. Each pod running on Fargate has its **own isolation boundary** and does not share the underlying kernel, CPU resources, memory resources, or elastic network interface with another pod.
4. AWS specially built **Fargate controllers** that recognizes the pods belonging to fargate and schedules them on Fargate profiles.
5. We will see more in our Fargate learning section.

## VPC

1. EKS uses AWS VPC network policies **to restrict** traffic between control plane components to within a single cluster.
2. Control plane components for a EKS cluster **cannot view or receive** communication from other clusters or other AWS accounts, except as authorized with Kubernetes RBAC policies.
3. This **secure and highly-available configuration** makes EKS reliable and recommended for **production workloads**.

# Kubernetes Architecture

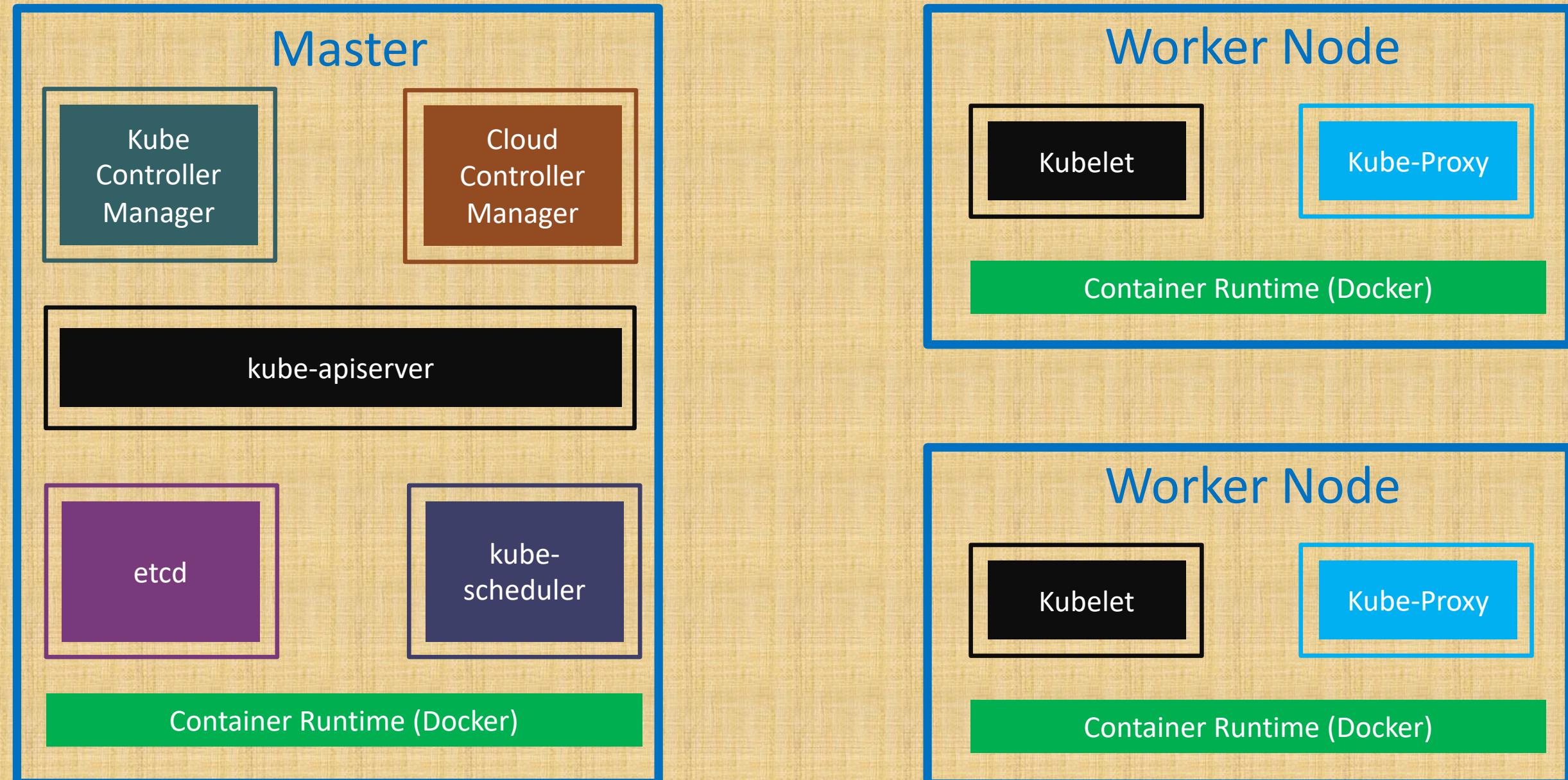




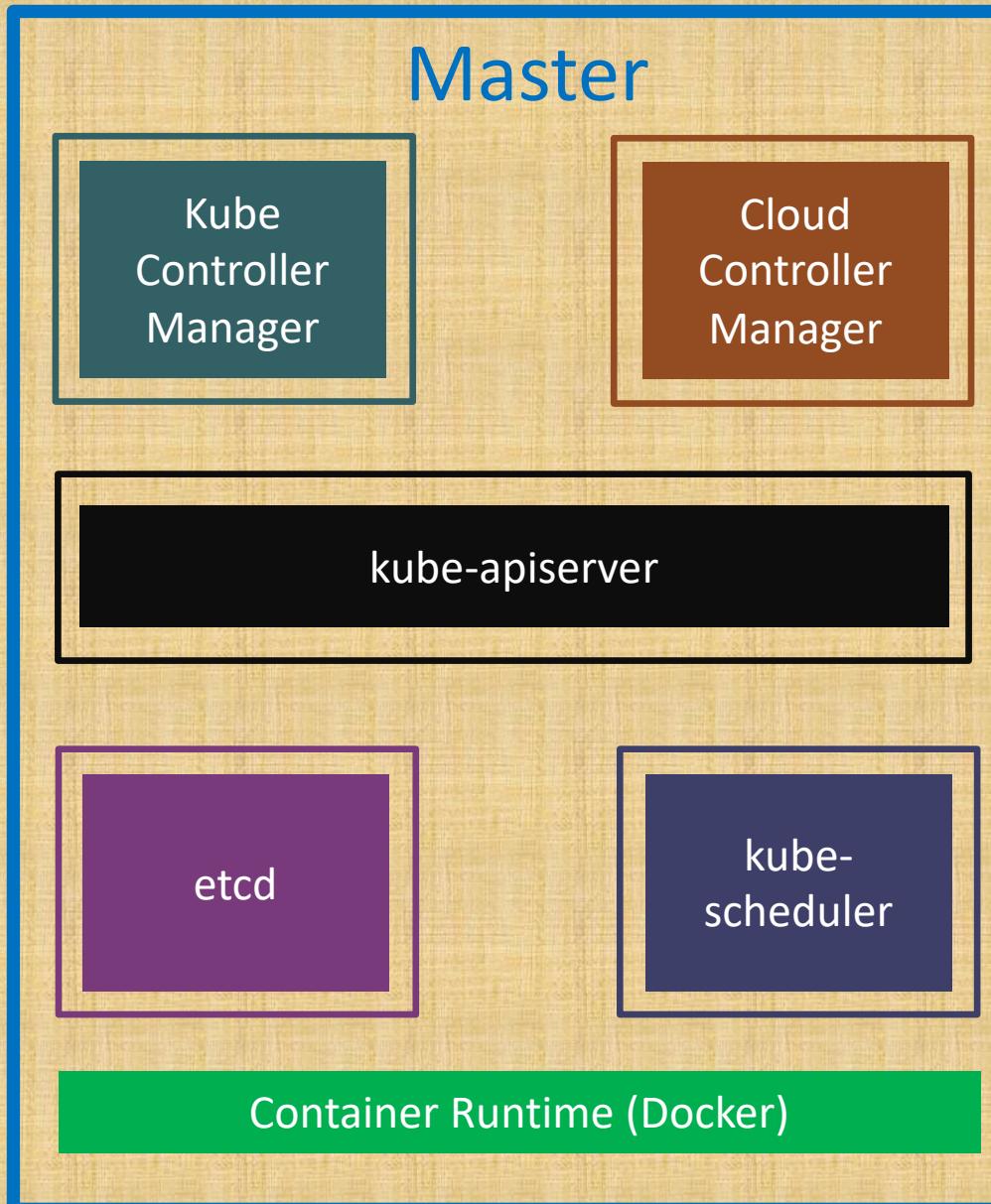
# Kubernetes Architecture



# Kubernetes - Architecture



# Kubernetes Architecture - Master



- **kube-apiserver**

- It acts as **front end** for the Kubernetes control plane. It **exposes** the Kubernetes API
- Command line tools (like kubectl), Users and even Master components (scheduler, controller manager, etcd) and Worker node components like (Kubelet) **everything talk** with API Server.

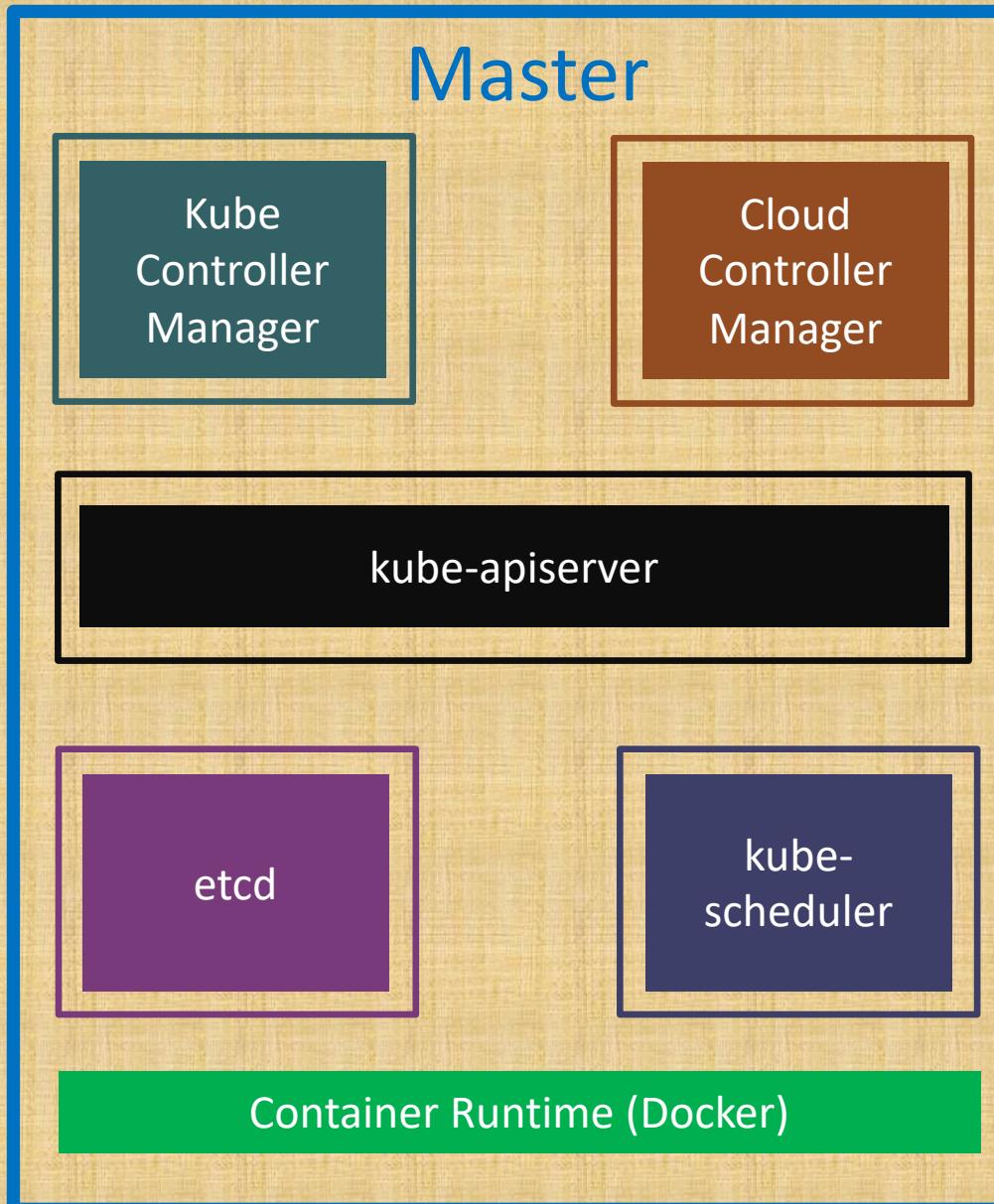
- **etcd**

- Consistent and highly-available **key value store** used as Kubernetes' **backing store** for all cluster data.
- It **stores** all the masters and worker node information.

- **kube-scheduler**

- Scheduler is responsible for distributing containers across multiple nodes.
- It watches for newly created Pods with no assigned node, and selects a node for them to run on.

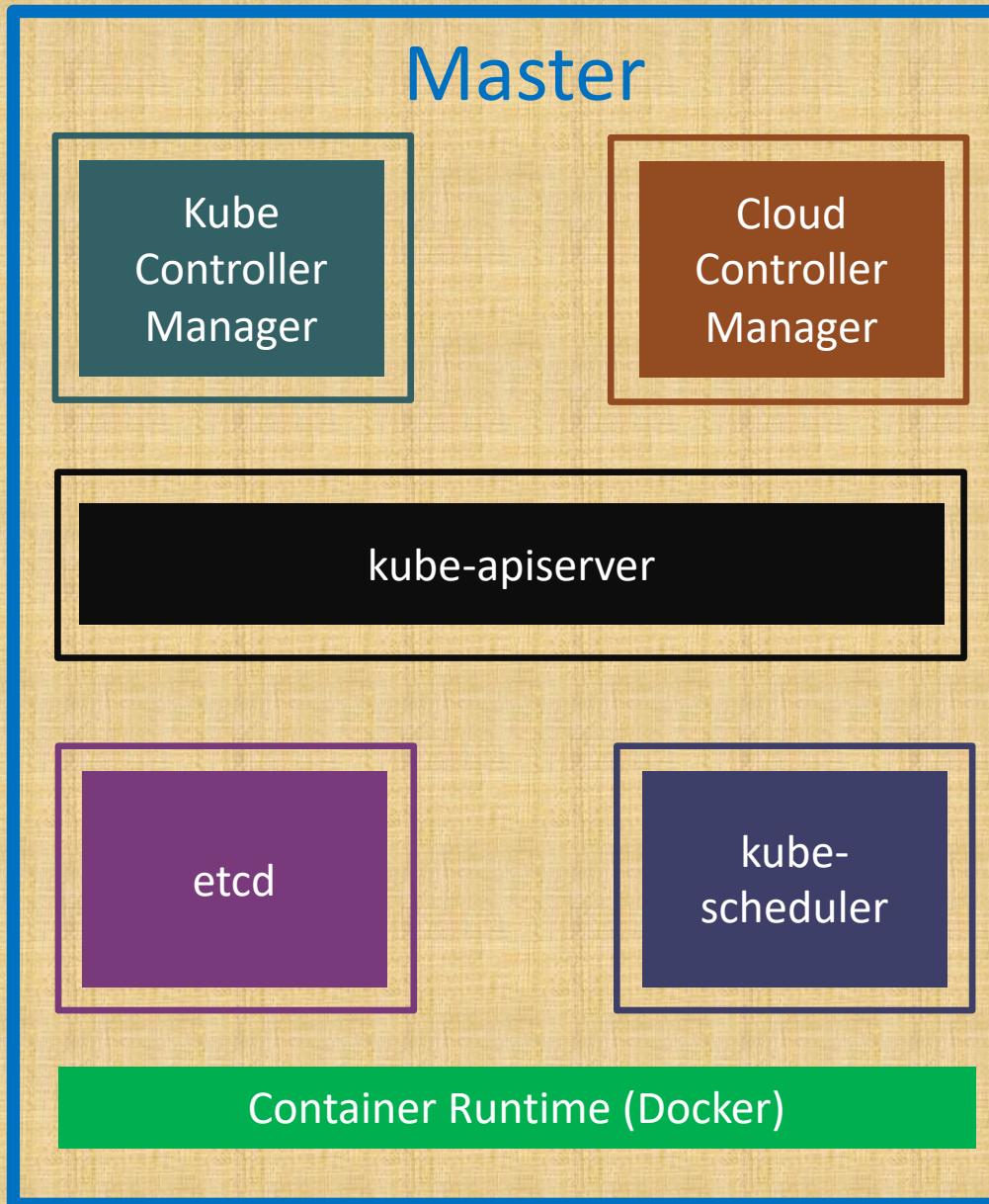
# Kubernetes Architecture - Master



- **kube-controller-manager**

- Controllers are responsible for noticing and responding when nodes, containers or endpoints go down. They make decisions to bring up new containers in such cases.
- **Node Controller:** Responsible for noticing and responding when **nodes go down**.
- **Replication Controller:** Responsible for maintaining the **correct number of pods** for every replication controller object in the system.
- **Endpoints Controller:** **Populates** the Endpoints object (that is, joins Services & Pods)
- **Service Account & Token Controller:** Creates default accounts and API Access for **new namespaces**.

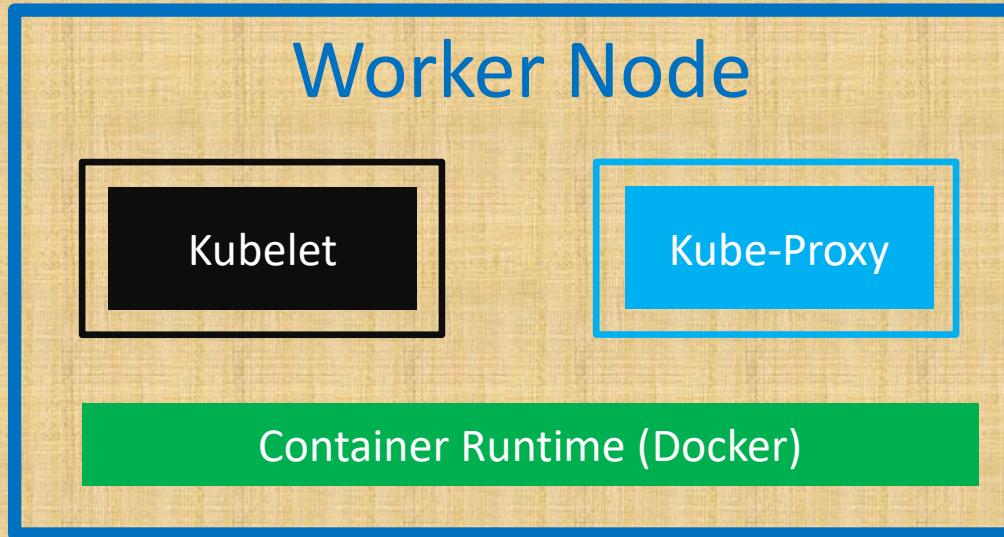
# Kubernetes Architecture - Master



- **cloud-controller-manager**

- A Kubernetes control plane component that embeds **cloud-specific control logic**.
- It only runs controllers that are **specific to your cloud provider**.
- **On-Premise** Kubernetes clusters will not have this component.
- **Node controller**: For **checking** the cloud provider to determine if a node has been deleted in the cloud after it stops responding
- **Route controller**: For setting up **routes** in the underlying cloud infrastructure
- **Service controller**: For creating, updating and deleting cloud provider **load balancer**

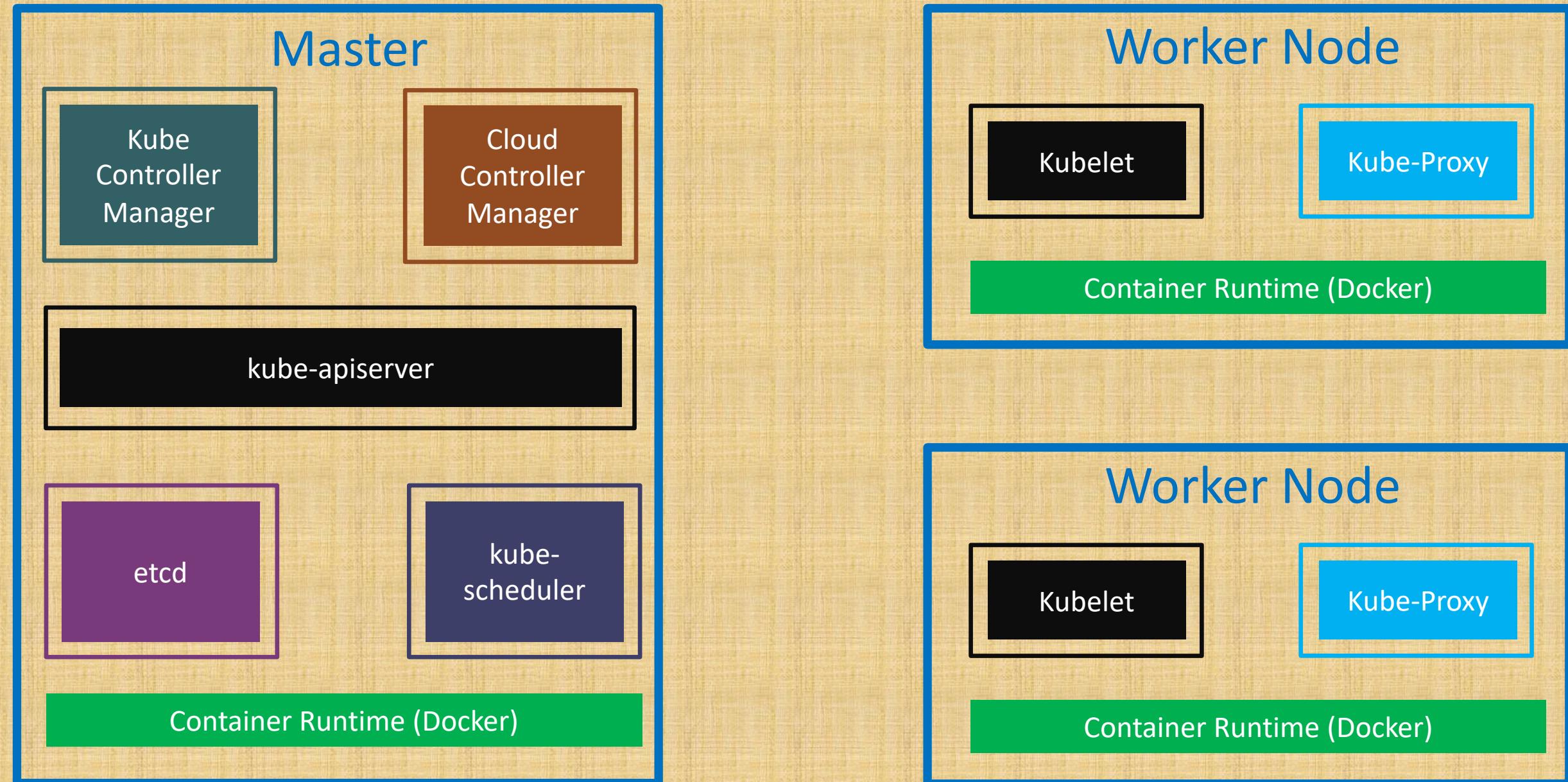
# Kubernetes Architecture – Worker Nodes



- Container Runtime
  - Container Runtime is the **underlying software** where we run all these Kubernetes components.
  - We are using Docker, but we have other runtime options like rkt, container-d etc.

- Kubelet
  - Kubelet is the **agent** that runs on every node in the cluster
  - This agent is **responsible** for making sure that containers are running in a Pod on a node.
- Kube-Proxy
  - It is a **network proxy** that runs on each node in your cluster.
  - It maintains **network rules** on nodes
  - In short, these network rules allow network communication to your Pods from network sessions inside or outside of your cluster.

# Kubernetes - Architecture

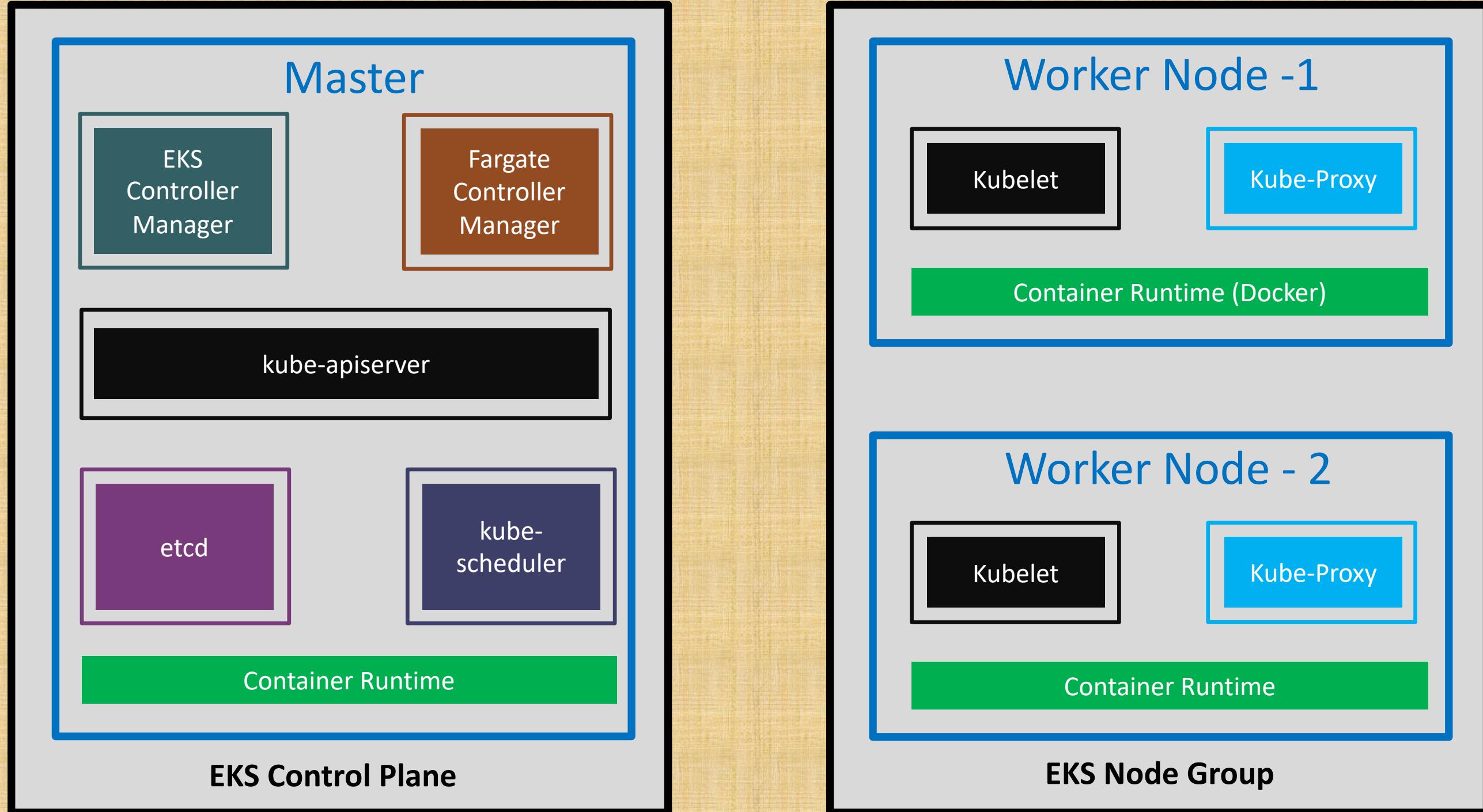




# AWS EKS Cluster



# EKS Kubernetes - Architecture

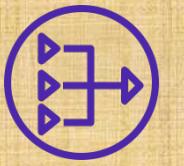




Amazon Virtual Private Cloud  
(Amazon VPC)



Internet gateway



NAT gateway



Elastic IP  
address



Route table



Public Subnet



Private Subnet

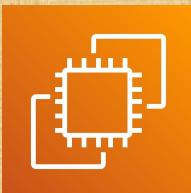
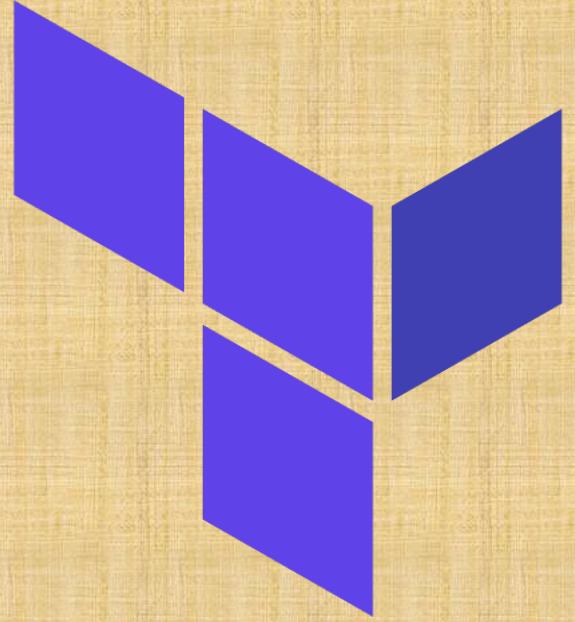


EKS  
Cluster

# AWS EKS

## Cluster & Node Groups

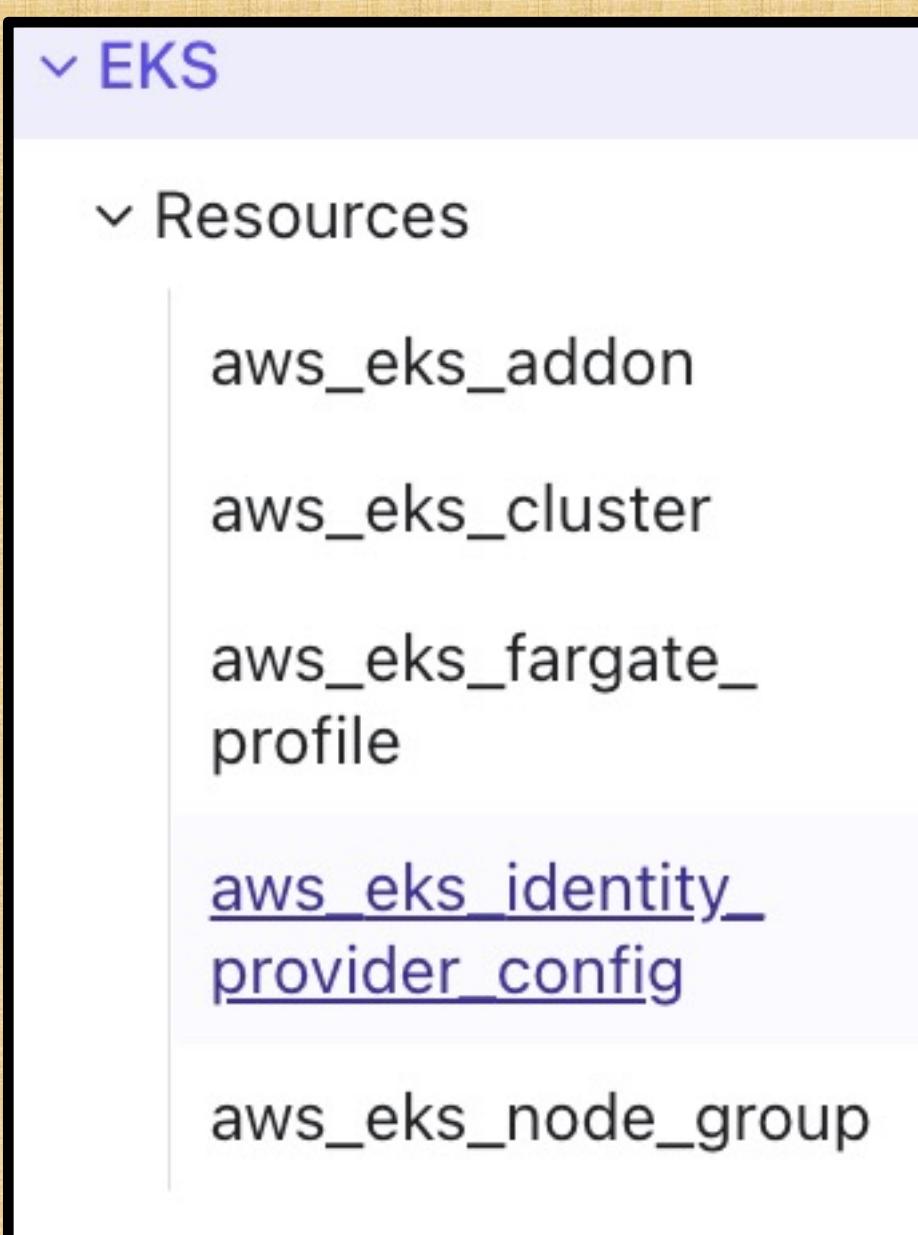
### using Terraform



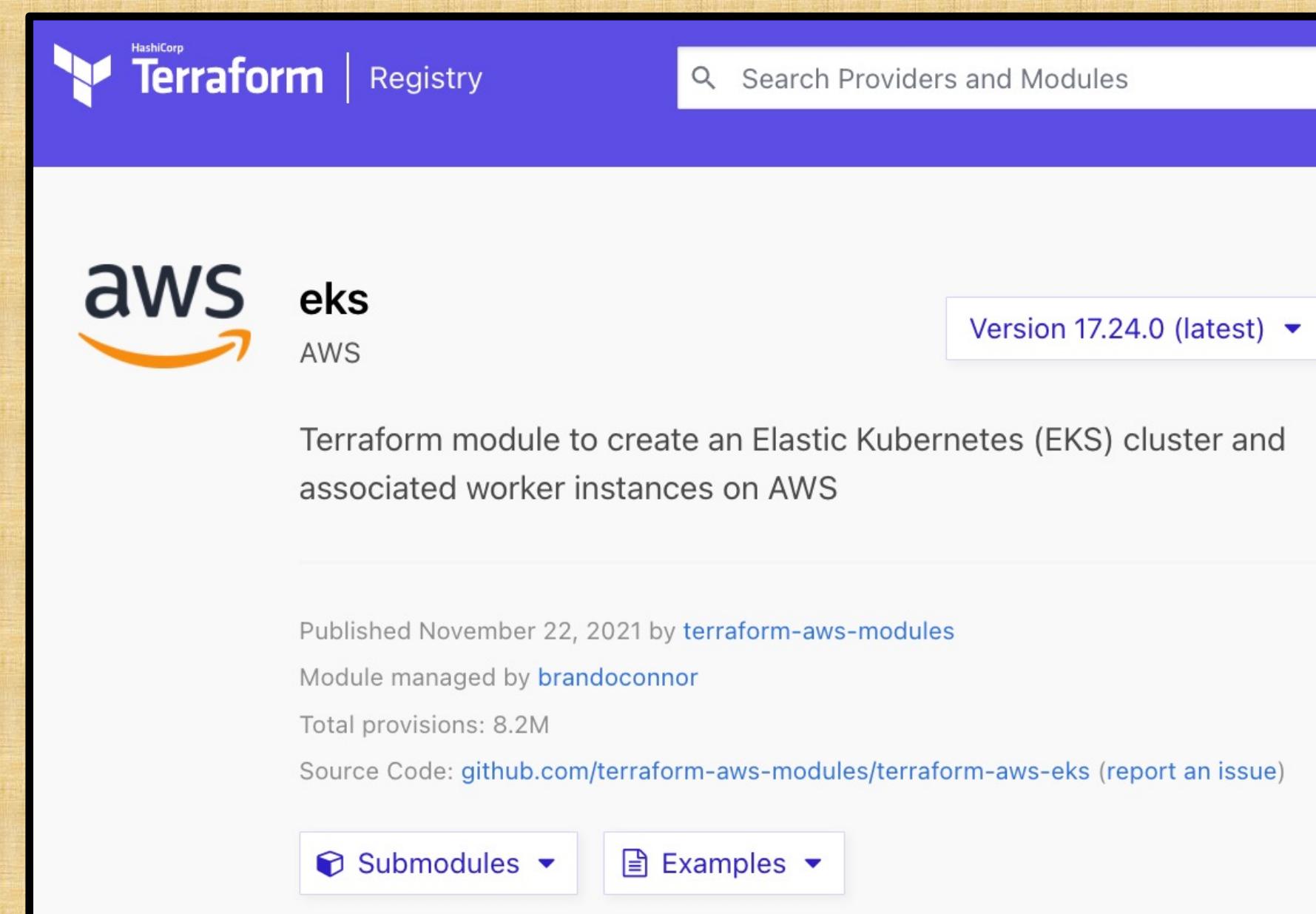
EC2 VM

# EKS Cluster Creation Options using Terraform

Terraform Resources



Terraform Module from Terraform Public Registry



# AWS EKS Resources - High Level Breakdown

VPC

Subnets (Public /  
Private)

Route Tables

NAT Gateway + Elastic  
IP

Internet Gateway

Bastion Host  
(Optional)

Bastion Host  
EC2 Instance

Bastion Host  
Security Group

Bastion Host  
Elastic IP

EKS Cluster

EKS Cluster  
IAM Role

EKS Cluster  
Security Group

EKS Cluster  
Network Interfaces

EKS Cluster

EKS Node  
Group

EKS Node Group  
IAM Role

EKS Node Group  
Security Group

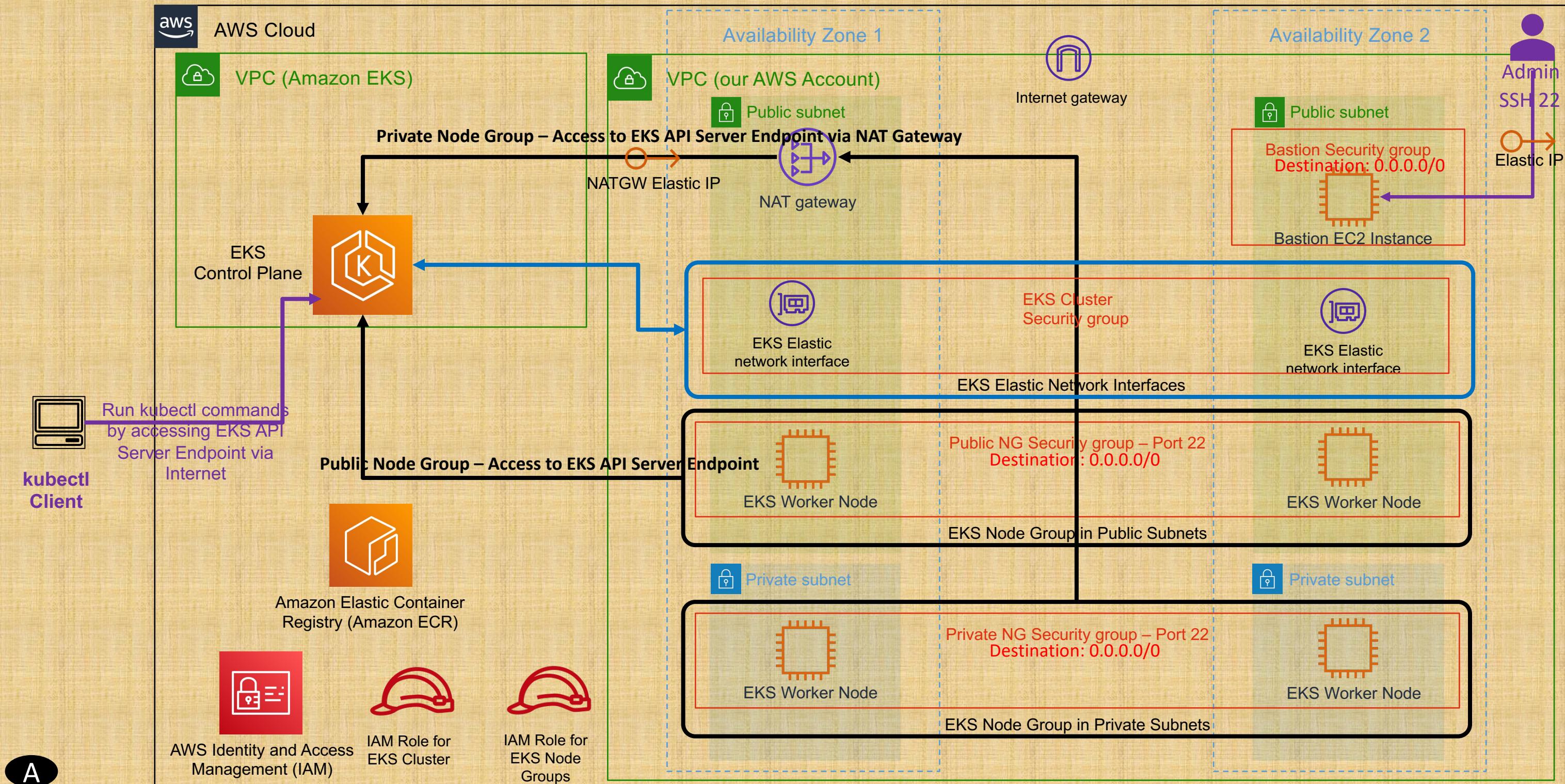
EKS Node Group  
Network Interfaces

EKS Worker Nodes  
EC2 Instances

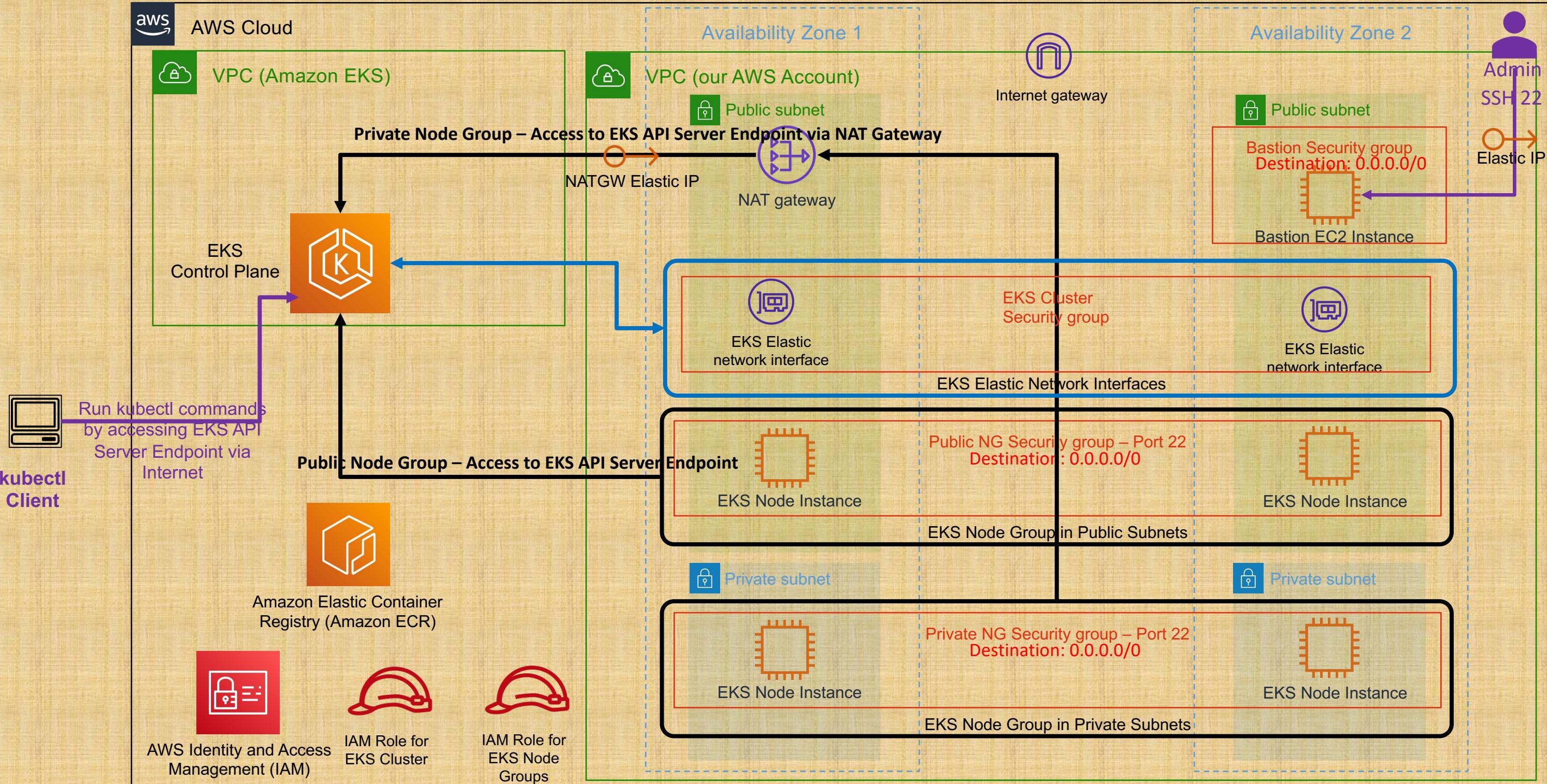
Pre-requisite Resources

EKS Resources

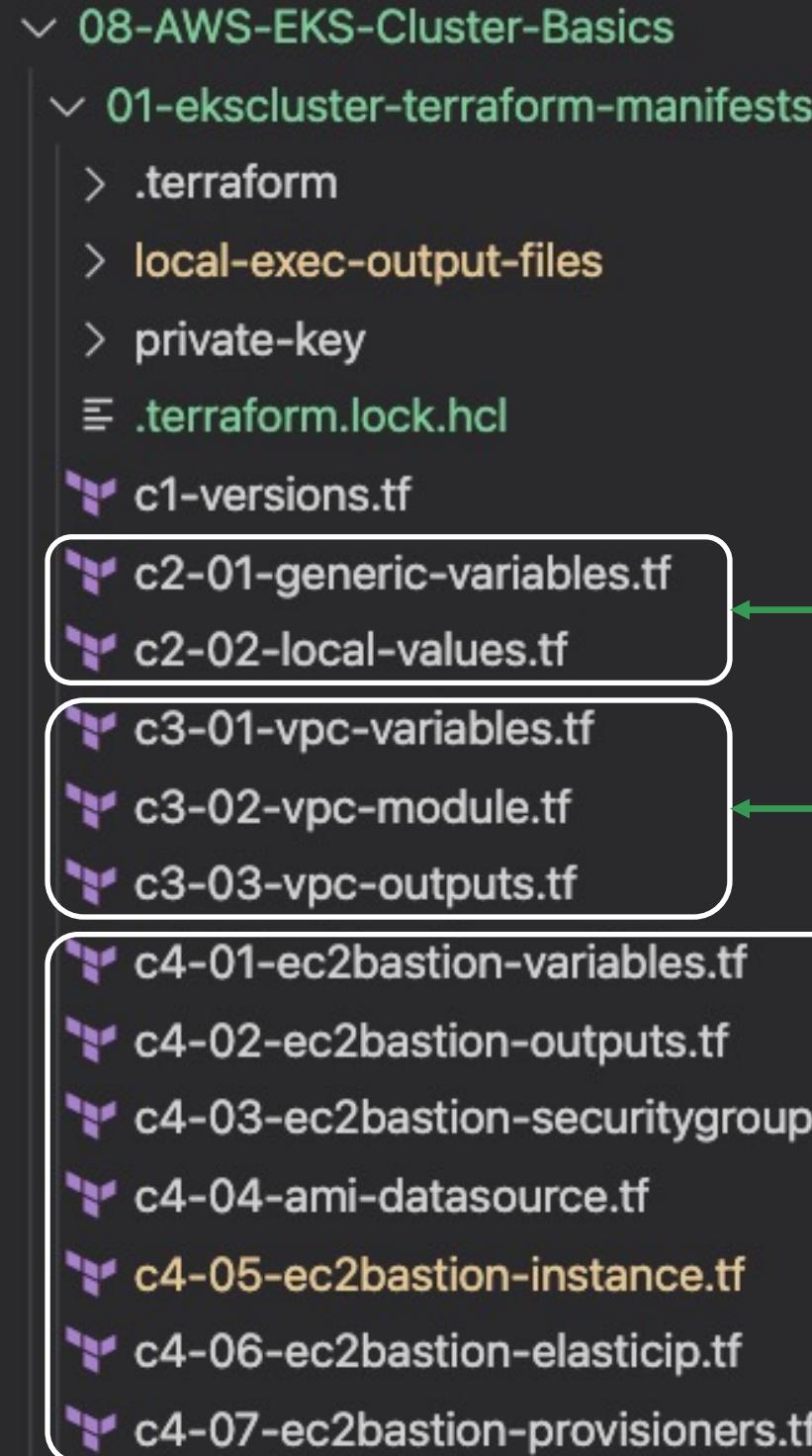
# EKS Cluster & EKS Node Groups in Public and Private Subnets



# EKS Cluster & EKS Node Groups in Public and Private Subnets



# What are we going to learn ?



C1 to C4-07 same as previous  
Section-07

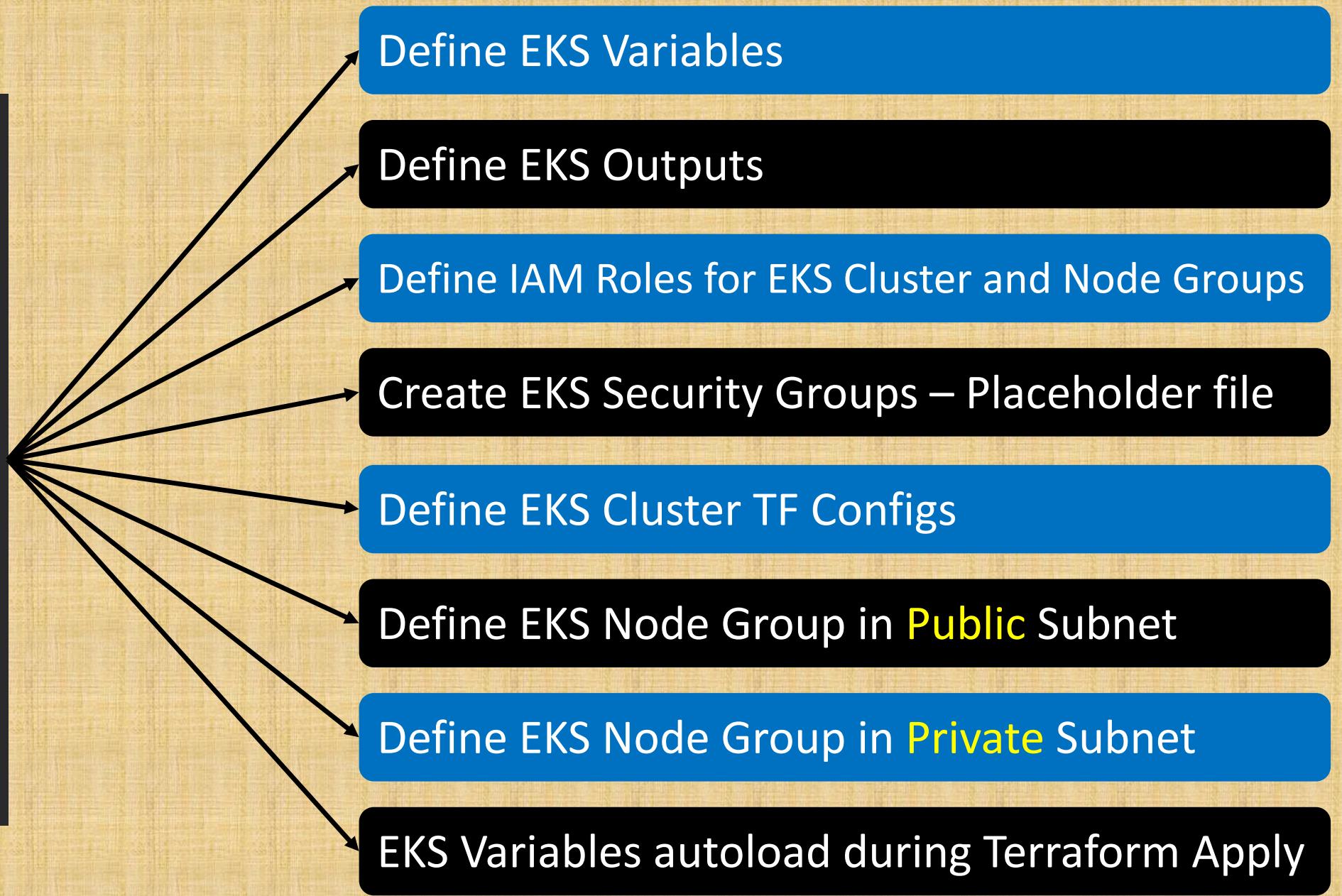
Define generic variables and Local Values

Create AWS VPC using Terraform Modules

Create AWS EC2 Bastion Host

# What are we going to learn ?

- c5-01-eks-variables.tf
- c5-02-eks-outputs.tf
- c5-03-iamrole-for-eks-cluster.tf
- c5-04-iamrole-for-eks-nodegroup.tf
- c5-05-securitygroups-eks.tf
- c5-06-eks-cluster.tf
- c5-07-eks-node-group-public.tf
- c5-08-eks-node-group-private.tf
- ec2bastion.auto.tfvars
- eks.auto.tfvars
- terraform.tfvars
- vpc.auto.tfvars



## ^ AWS EKS Cluster, Public and Private Node Groups using Terraform

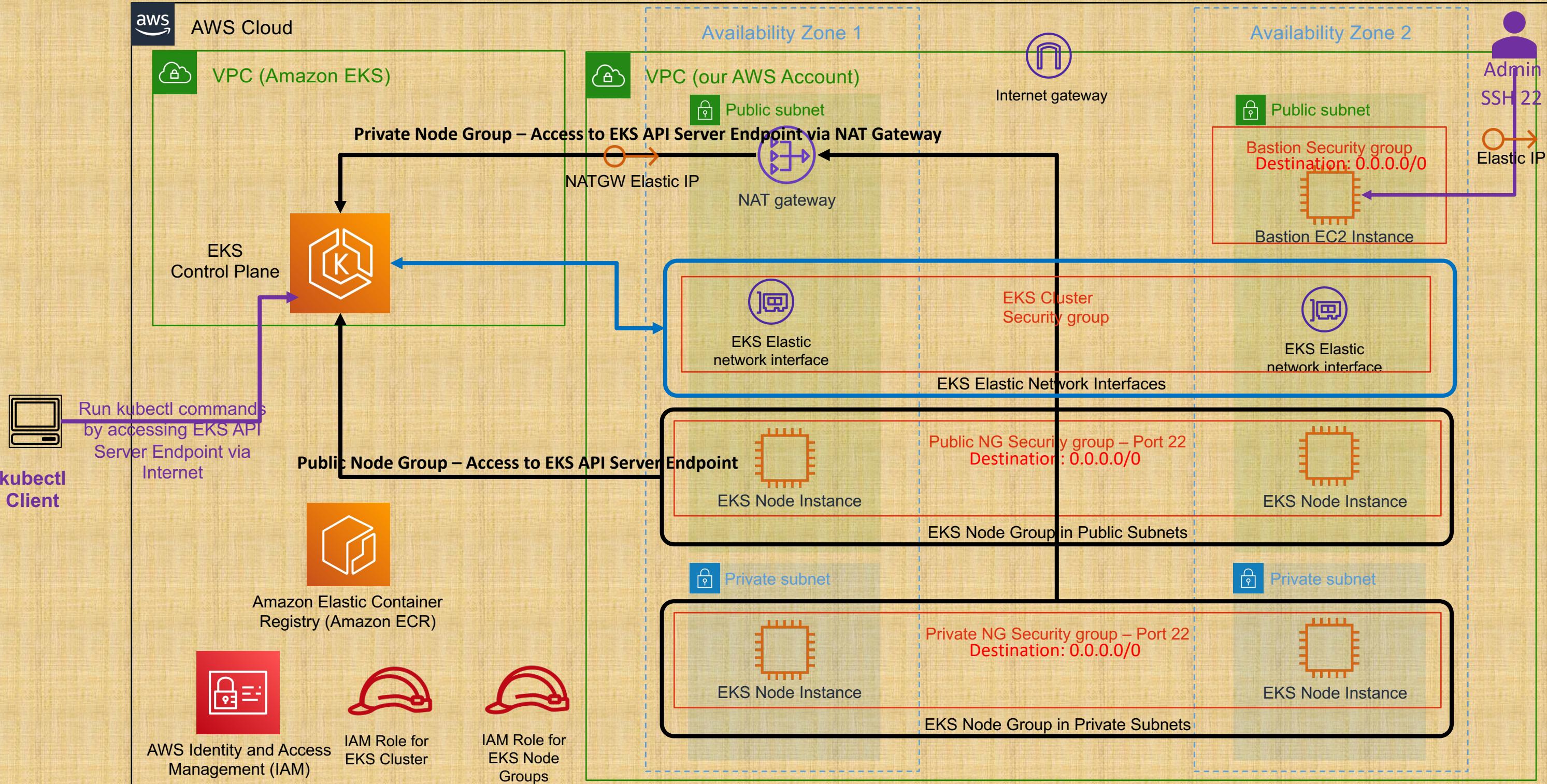
- Step-00: Introduction to EKS Cluster Basics
- Step-01: Review EKS Variables, EKS Cluster IAM Role 12:17
- Step-02: Review EKS Node Group IAM Role 05:12
- Step-03: Create EKS Cluster Terraform Resource 14:08
- Step-04: Create EKS Node Groups in Public and Private Subnets 10:50
- Step-05: Review EKS Variables and EKS Outputs 06:53
- Step-06: Execute TF Commands, Verify EKS Cluster from AWS Mgmt Console 16:48
- Step-07: Install kubectl, configure kubeconfig and verify 07:41
- Step-08: SSH to Worker nodes from Bastion Host and Verify 11:25
- Step-09: Run fundamental kubectl commands to explore EKS Cluster 09:55
- Step-10: Understand EKS Cluster Network Interfaces 08:43
- Step-11: Understand EKS Security Groups 06:57

11 lectures • 1hr 51min

Time it takes to complete this Demo

Real-World  
Demo 3

# EKS Cluster & EKS Node Groups in Public and Private Subnets



# AWS EKS Resources - High Level Breakdown

- **Elastic Network Interfaces (ENI)**

- When you create the cluster, Amazon EKS creates and manages network interfaces in your account that have [Amazon EKS <cluster name>](#) in their description.
- These network interfaces allow [AWS Fargate](#) and [Amazon EC2 instances](#) to communicate with the [EKS control plane](#) present in Amazon VPC in Amazon Account (Not our AWS Account).
- The [Amazon EKS created cluster security group](#) and any additional security groups that you specify when you create your cluster are applied to these network interfaces.
- **Very Important Note:** These Network Interfaces were created [in our EKS VPC](#) [in our AWS account](#) but attached to EKS Control Plane Master Node instances of [Amazon VPC](#).

# EKS Control Plane - Network Interfaces

Network interfaces (7) [Info](#)

[C](#) Actions ▾ [Create network interface](#)

[Filter network interfaces](#)

< 1 > [⚙️](#)

Network interface ID	Subnet ID	VPC ID	Availability Zone	Description	Instance ID	Status
eni-042b0dac3475c7759	subnet-0b7251328886b971b	vpc-0bfd05ddf2d2fb57f	us-east-1b	-	i-0ad17ca7c2a4d6e26	<span>✓ In-use</span>
eni-0f2a2529e115e5399	subnet-0ddd92a77db0424b4	vpc-0bfd05ddf2d2fb57f	us-east-1a	-	i-0040b7f14f125d2fa	<span>✓ In-use</span>
eni-00b08f0acab5d82fb	subnet-0ddd92a77db0424b4	vpc-0bfd05ddf2d2fb57f	us-east-1a	-	i-0656b0b44cd31e6d0	<span>✓ In-use</span>
eni-0bb2f010028ba99ae	subnet-0c972dd0a21bcaadb	vpc-0bfd05ddf2d2fb57f	us-east-1b	Amazon EKS hr-stag-eksdemo1	-	<span>✓ In-use</span>
eni-02af4db12f7cb8e8c	subnet-0ddd92a77db0424b4	vpc-0bfd05ddf2d2fb57f	us-east-1a	Amazon EKS hr-stag-eksdemo1	-	<span>✓ In-use</span>
eni-034ed591b137c3292	subnet-0ddd92a77db0424b4	vpc-0bfd05ddf2d2fb57f	us-east-1a	aws-K8S-i-0040b7f14f125d2fa	i-0040b7f14f125d2fa	<span>✓ In-use</span>
eni-0fcf645adaf105417	subnet-0ddd92a77db0424b4	vpc-0bfd05ddf2d2fb57f	us-east-1a	Interface for NAT Gateway nat-0494...	-	<span>✓ In-use</span>

Network Interfaces created by EKS Control Plane in our EKS VPC. These are created and managed by EKS

Network Interface description is outlined as Amazon EKS <CLUSTER-NAME>

# EKS ENI

## ▼ Network interface details

Network interface ID  
[eni-02af4db12f7cb8e8c](#)

Network interface status  
In-use

VPC ID  
[vpc-0bfd05ddf2d2fb57f](#)

Our EKS VPC ID

Owner  
[180789647333](#)

Our Amazon Account ID

Source/dest. check  
True

## ▼ IP addresses

Private IPv4 address  
[10.0.101.191](#)

Private IP of from our EKS VPC Public Subnet

Public IPv4 address  
-

Secondary private IPv4 addresses  
-

MAC address  
[02:d2:e6:22:43:3f](#)

## ▼ Instance details

Instance ID  
-

Instance owner  
[588289678924](#)

Allocation ID  
-

Name  
-

Description  
[Amazon EKS hr-stag-eksdemo1](#)

Interface type  
[Elastic network interface](#)

Security groups  
[sg-02dcf98251a20d356 \(eks-cluster-sg-hr-stag-eksdemo1-1568012412\)](#)

Subnet ID  
[subnet-0ddd92a77db0424b4](#)

Availability Zone  
[us-east-1a](#)

Requester ID  
[588289678924](#)

Requester-managed  
False

Private IPv4 DNS  
[ip-10-0-101-191.ec2.internal](#)

Elastic Fabric Adapter  
False

Public IPv4 DNS  
-

IPv6 addresses  
-

Association ID  
-

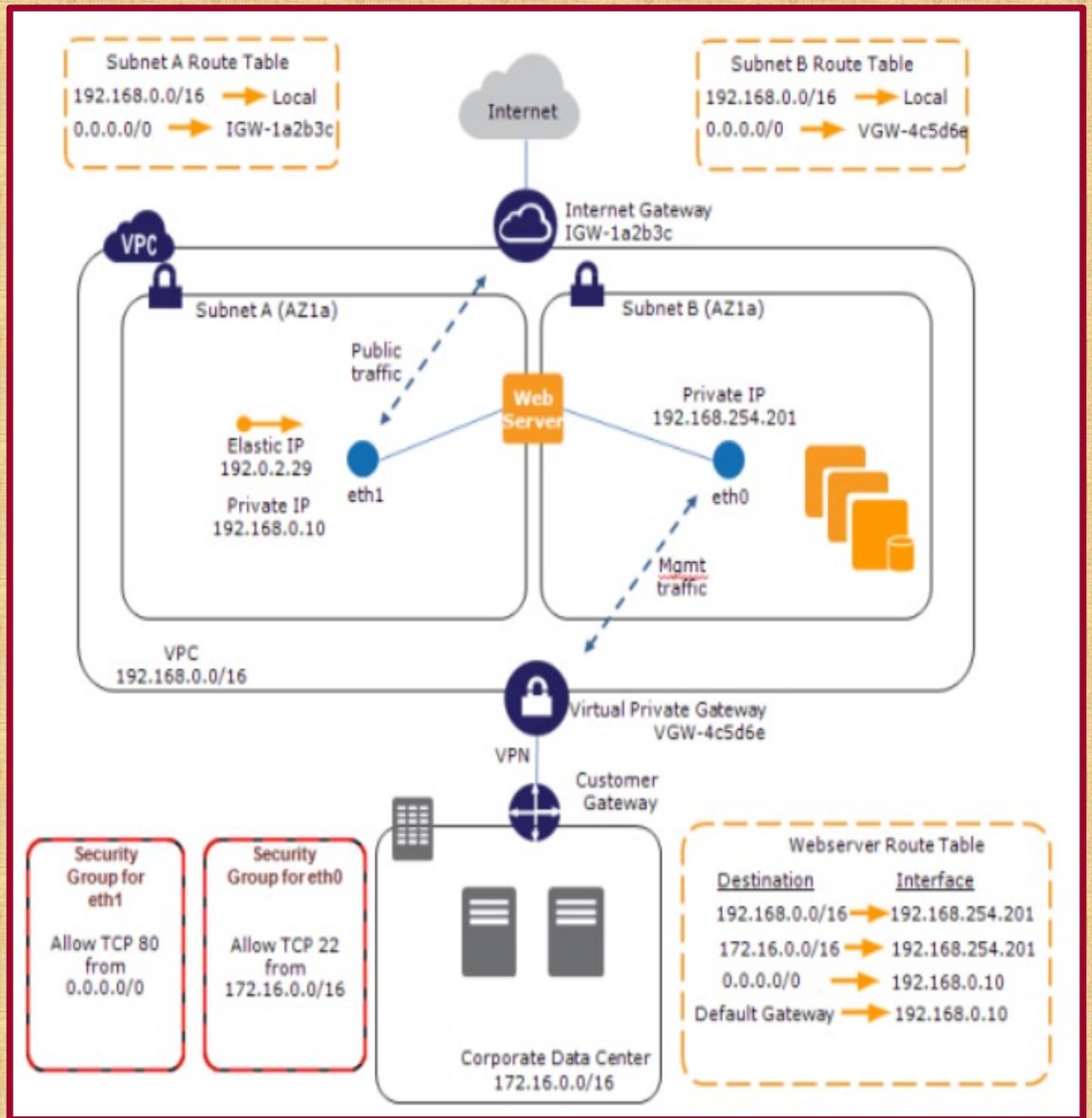
Elastic IP address owner  
-

IPv4 Prefix Delegation  
-

IPv6 Prefix Delegation  
-

Amazon VPC Account ID where EKS Control Plane is hosted

# AWS Elastic Network Interface Basic Understanding



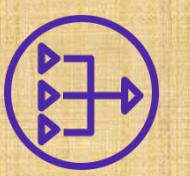
Just to understand AWS  
Elastic Network Interface  
concept



Amazon Virtual Private Cloud  
(Amazon VPC)



Internet gateway



NAT gateway



Elastic IP  
address



Route table



Public Subnet



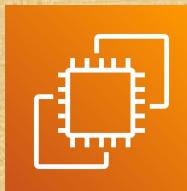
Private Subnet



# AWS EKS Deployment & Service

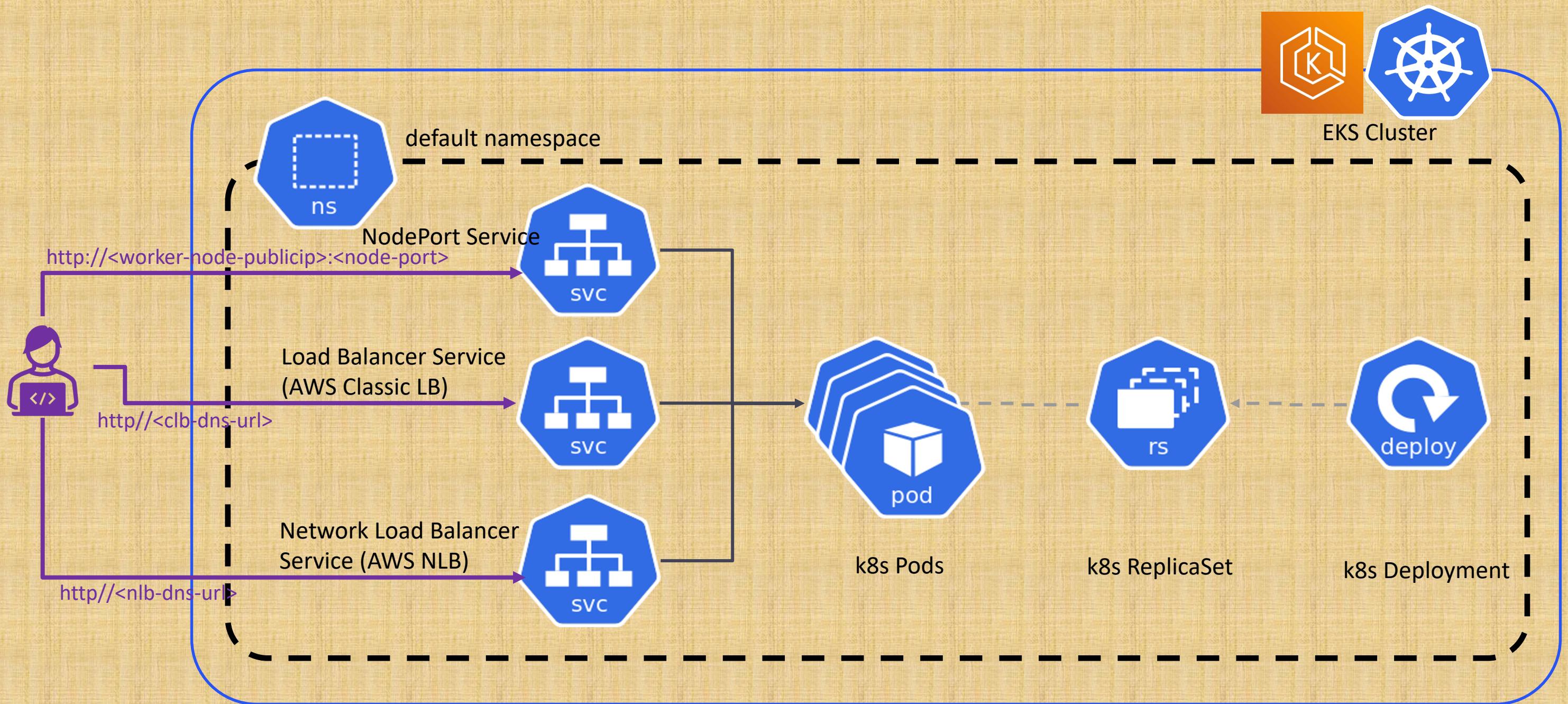


EKS  
Cluster

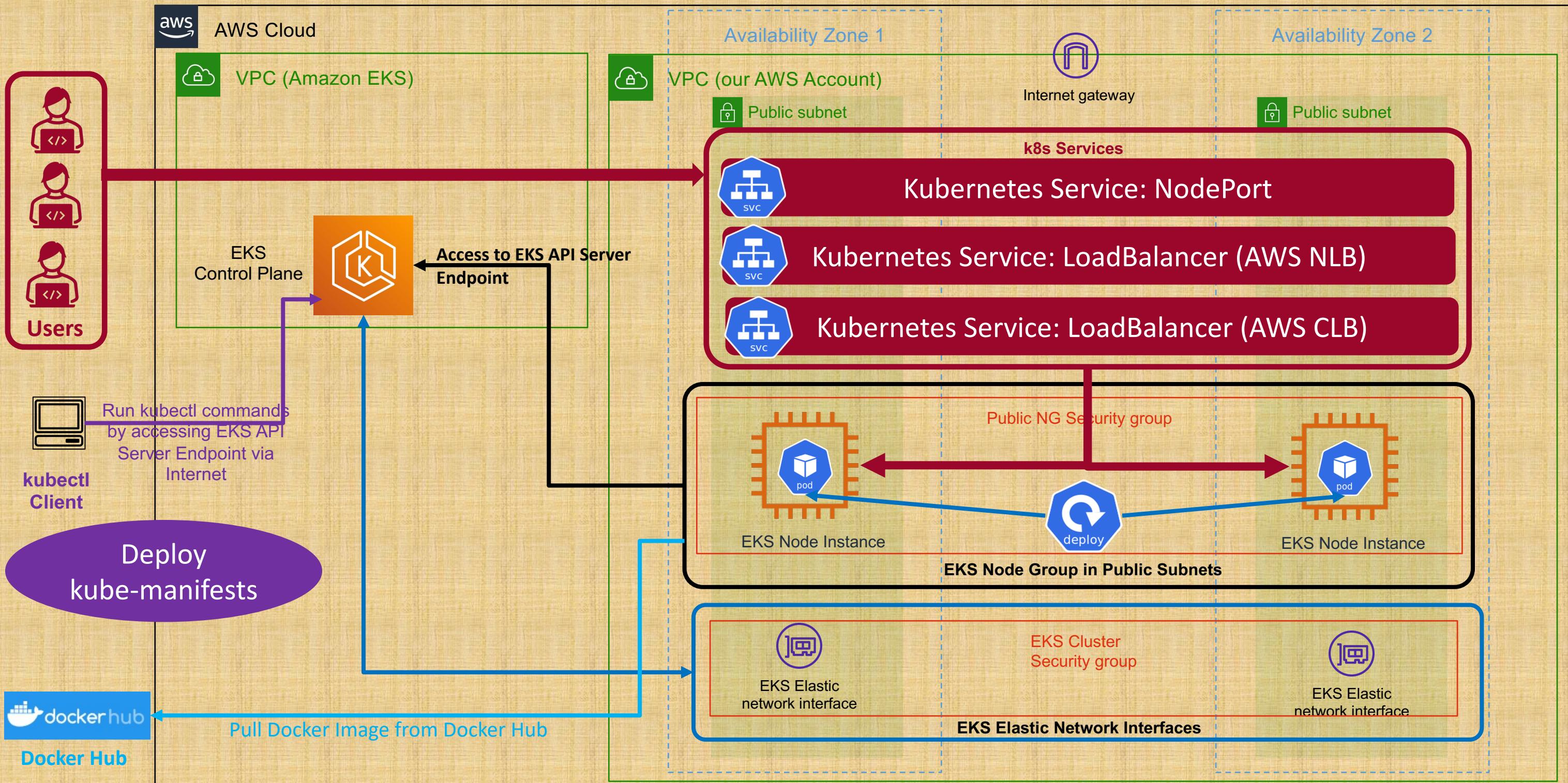


EC2 VM

# Kubernetes Sample Application



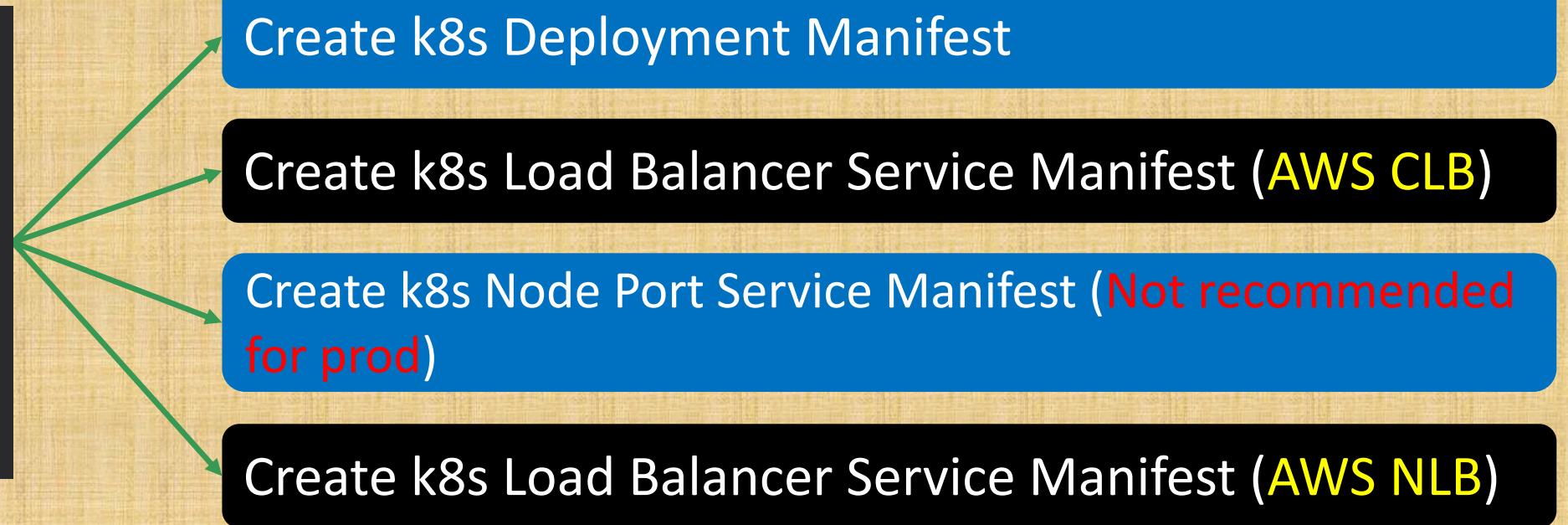
# Kubernetes Deployment & Services - Architecture



# What are we going to learn ?

Kubernetes Manifests using  
YAML

```
✓ 10-Kubernetes-Deployment-and-Service
  ✓ kube-manifests
    ! 01-Deployment.yaml
    ! 02-CLB-LoadBalancer-Service.yaml
    ! 03-NodePort-Service.yaml
    ! 04-NLB-LoadBalancer-Service.yaml
```





Amazon Virtual Private Cloud  
(Amazon VPC)



Internet gateway



NAT gateway



Elastic IP  
address



Route table



Public Subnet



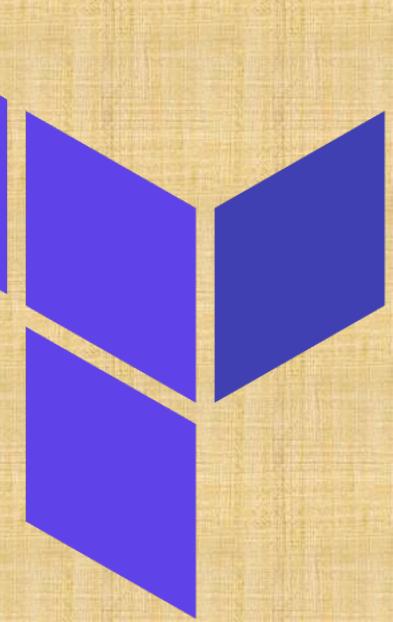
Private Subnet



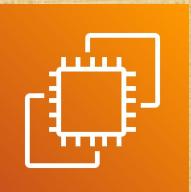
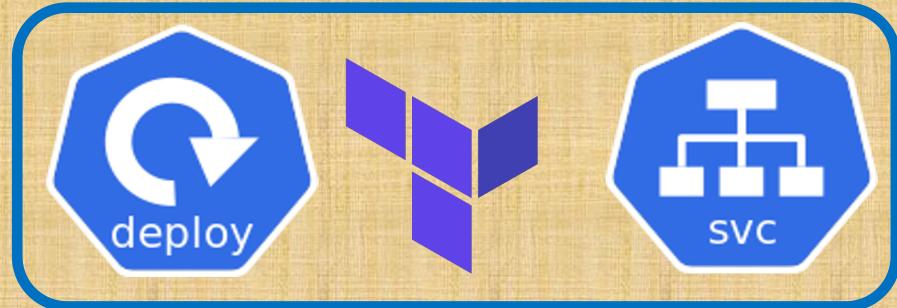
# AWS EKS

## Deployment & Service

### using Terraform Provider

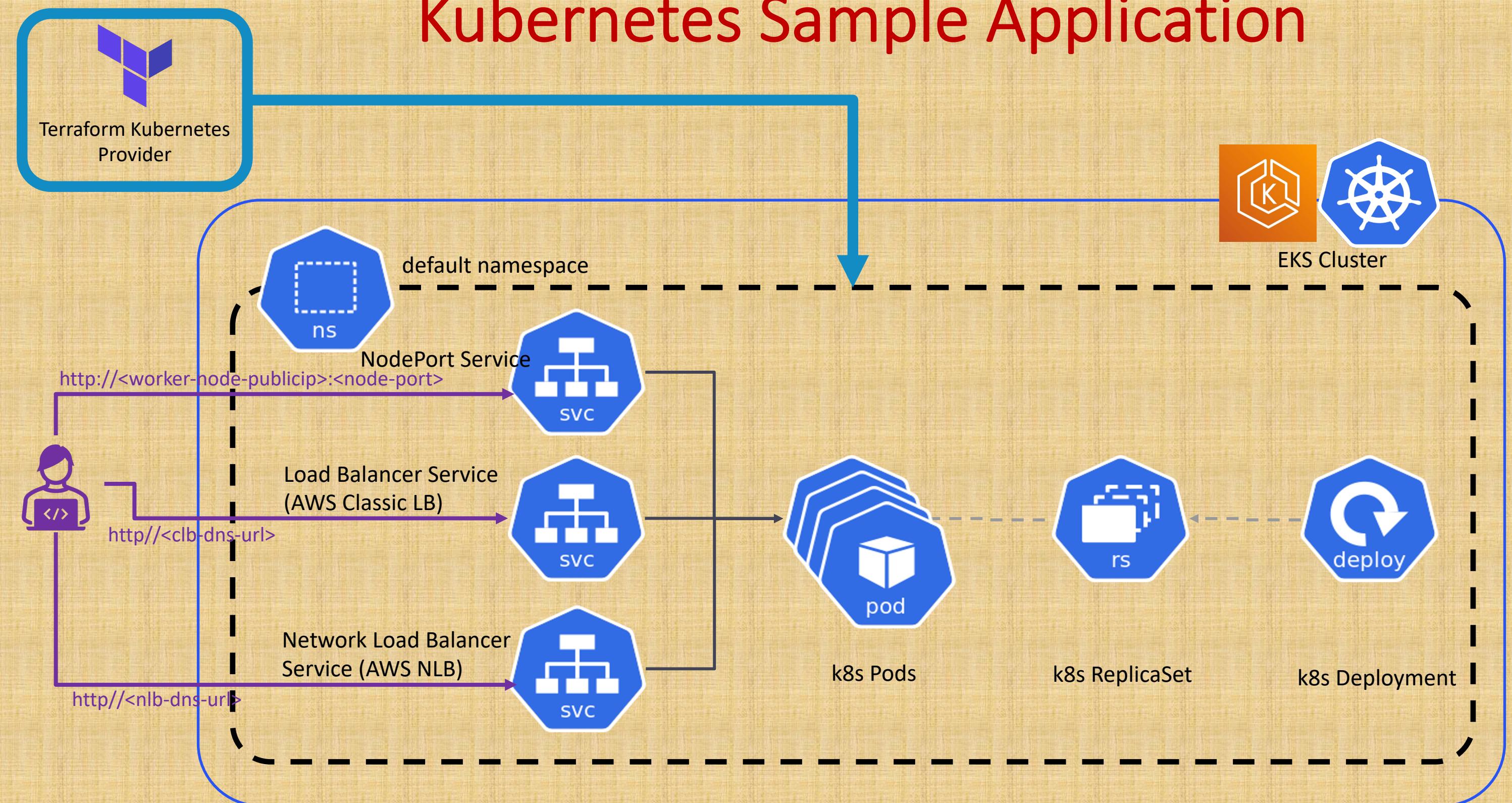


EKS  
Cluster



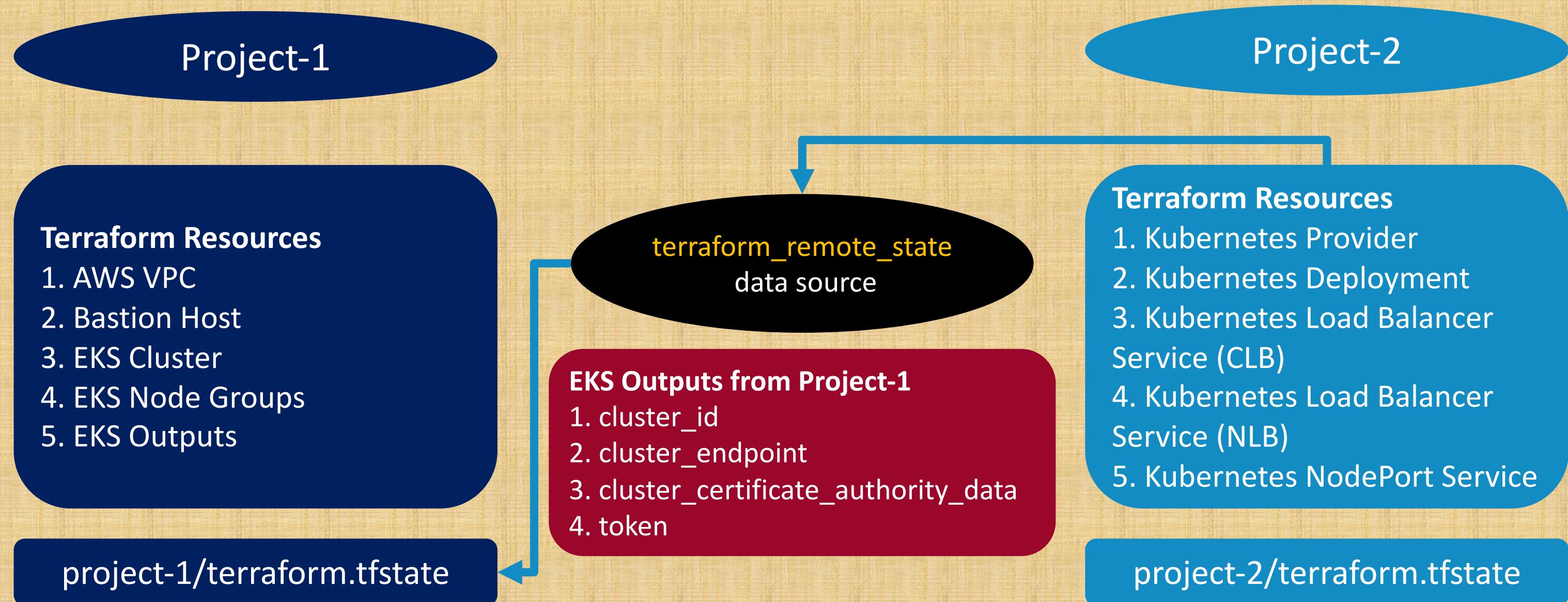
EC2 VM

# Kubernetes Sample Application



# Terraform Remote State Datasource

The `terraform_remote_state` data source retrieves the root module output values from project-1 Terraform configuration, using the latest state snapshot from the remote backend.



# What are we going to learn ?

## Kubernetes Manifests using Terraform

```
▽ 11-Kubernetes-Resources-via-Terraform
  ▽ 02-k8sresources-terraform-manifests
    c1-versions.tf
    c2-remote-state-datasource.tf
    c3-providers.tf
    c4-kubernetes-deployment.tf
    c5-kubernetes-loadbalancer-service-clb.tf
    c6-kubernetes-nodeport-service.tf
    c7-kubernetes-loadbalancer-service-nlb.tf
```

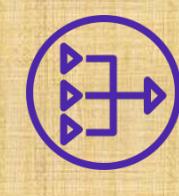
- C1 Define Terraform Settings - AWS and Kubernetes Provider Versions
- C2 Create Remote State Datasource
- C3 Define AWS and Kubernetes Providers & Datasources
- C4 Create k8s Deployment Manifest
- C5 Create k8s Load Balancer Service Manifest (AWS CLB)
- C6 Create k8s Node Port Service Manifest (Not recommended for prod)
- C7 Create k8s Load Balancer Service Manifest (AWS NLB)



VPC



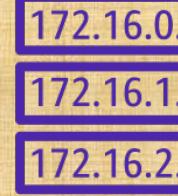
Internet gateway



NAT gateway



Elastic IP address



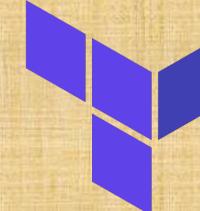
Route table



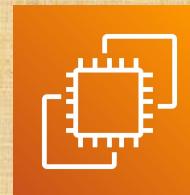
Public Subnet



Private Subnet



EKS Cluster



EC2 VM



Classic Load Balancer



Autoscaling



AWS IAM



Network Load Balancer



Amazon Simple Storage Service (Amazon S3)



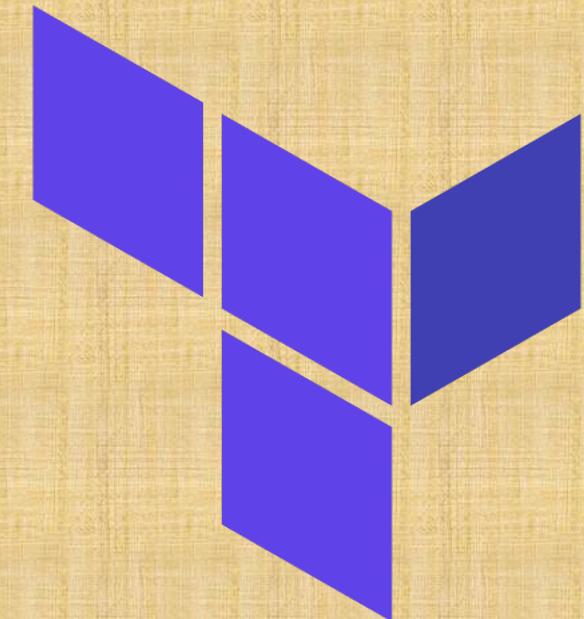
Amazon DynamoDB

# Terraform

## Remote State Storage & State Locking

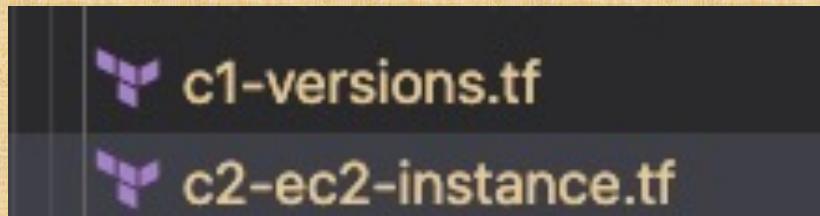
using

## AWS S3 & DynamoDB



# Desired & Current Terraform States

Terraform Configuration Files



Desired State



Real World Resource – EC2 Instance

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
web	i-0663449fef49e9cf5	Running	t2.micro	2/2 checks ...	No alarms +	us-east-1b

Instance: i-0663449fef49e9cf5 (web)

Details Security Networking Storage Status checks Monitoring Tags

Instance summary Info

Instance ID: i-0663449fef49e9cf5 (web) Public IPv4 address: 54.144.73.100 | open address

Instance state: Running Public IPv4 DNS: ec2-54-144-73-100.compute-1.amazonaws.com | open address

Instance type: t2.micro Elastic IP addresses: -

AWS Compute Optimizer finding: Opt-in to AWS Compute Optimizer for recommendations.

IAM Role: -

VPC ID: vpc-54972d2e (default-vpc)

Private IPv4 addresses: 172.31.94.137

Private IPv4 DNS: ip-172-31-94-137.ec2.internal

Subnet ID: subnet-d2e590fc



Current State

# Terraform State

A diagram illustrating Terraform State storage options. It features two large blue circles side-by-side. The left circle contains the text "Terraform Local State Storage" in white and yellow. The right circle contains the text "Terraform Remote State Storage" in white and yellow.

Terraform  
Local  
State  
Storage

Terraform  
Remote  
State  
Storage

# What is Terraform Backend ?

Backends are responsible for storing state and providing an API for state locking.

Terraform  
State Storage



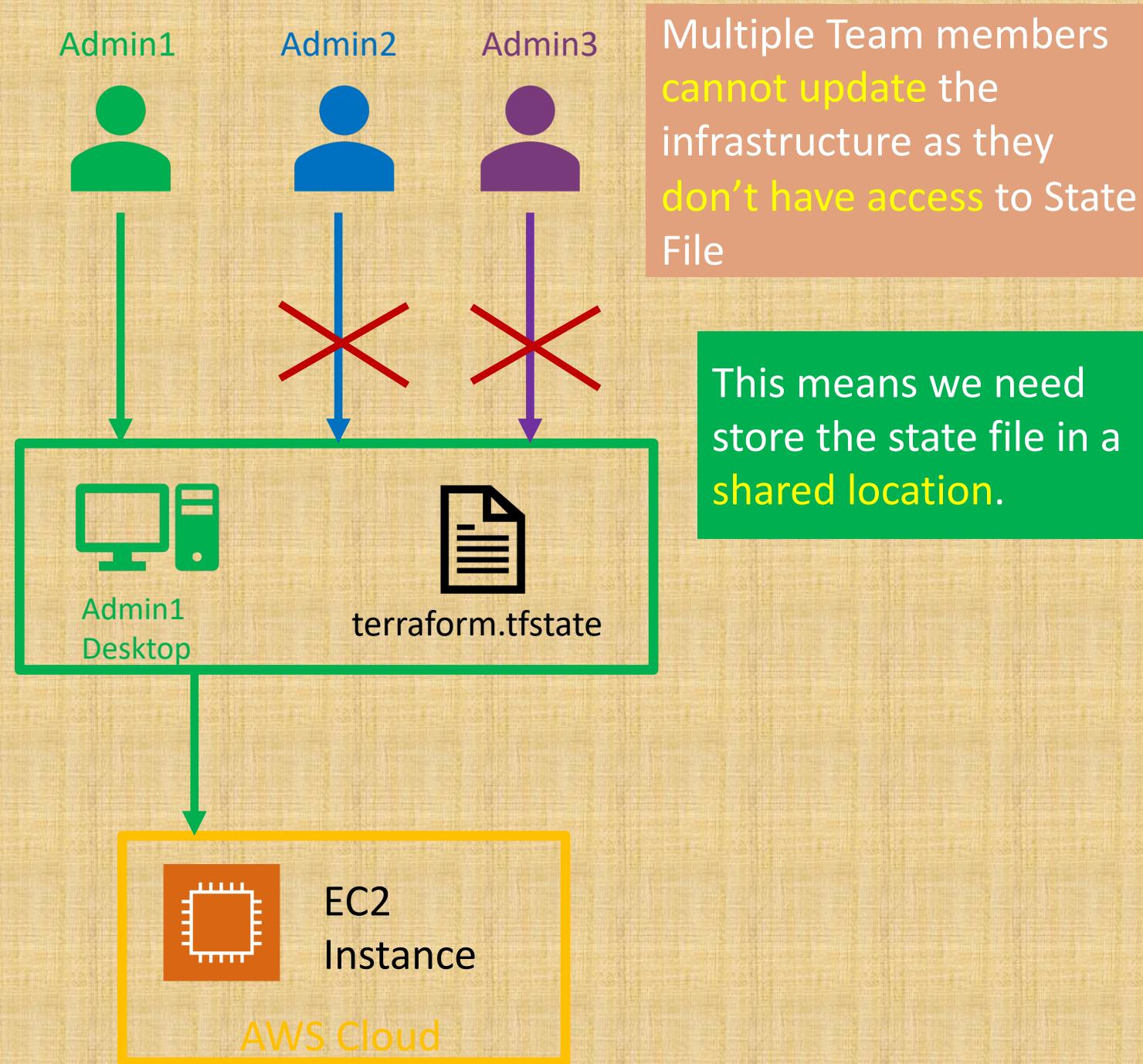
AWS S3 Bucket

Terraform  
State Locking

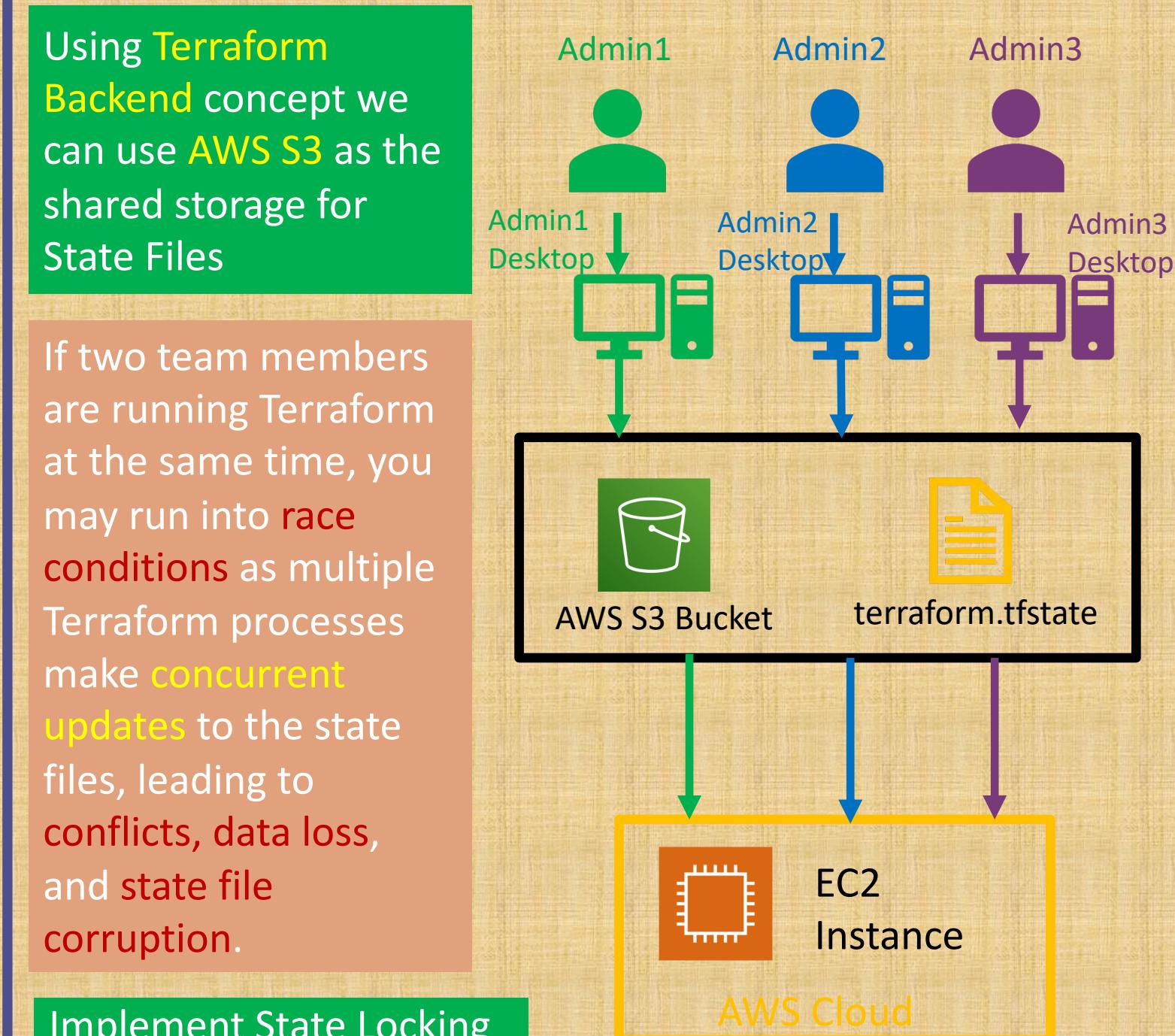


AWS DynamoDB

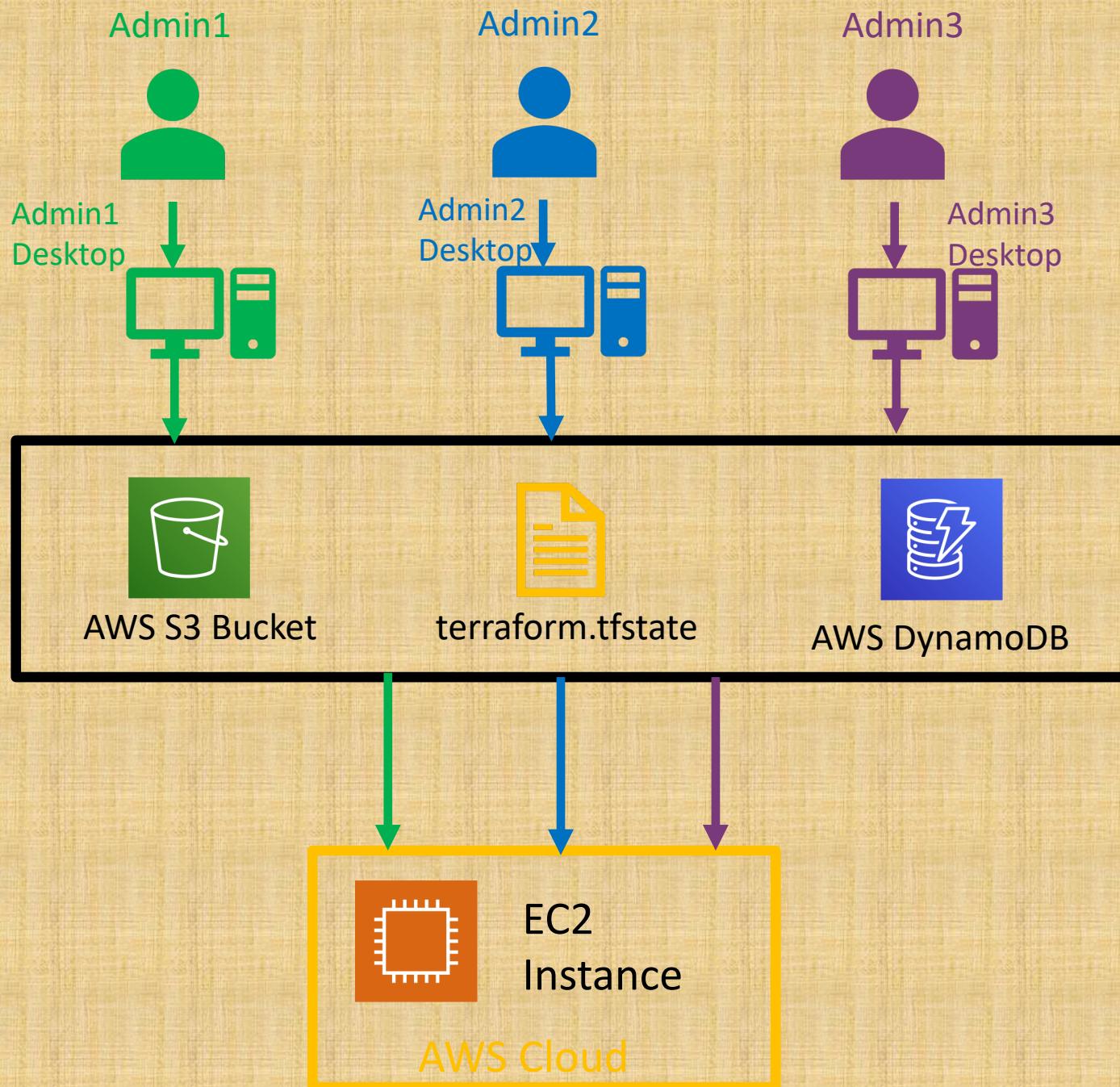
# Local State File



# Remote State File



# Terraform Remote State File with State Locking



Not all backends support State Locking. AWS S3 supports State Locking

State locking happens automatically on all operations that could write state.

If state locking fails, Terraform will not continue.

You can disable state locking for most commands with the `-lock` flag but it is not recommended.

If acquiring the lock is taking longer than expected, Terraform will output a status message.

If Terraform doesn't output a message, state locking is still occurring if your backend supports it.

Terraform has a force-unlock command to manually unlock the state if unlocking failed.

# Terraform Remote State File with State Locking

Terraform State Storage using Remote Backend AWS S3

Terraform State Locking

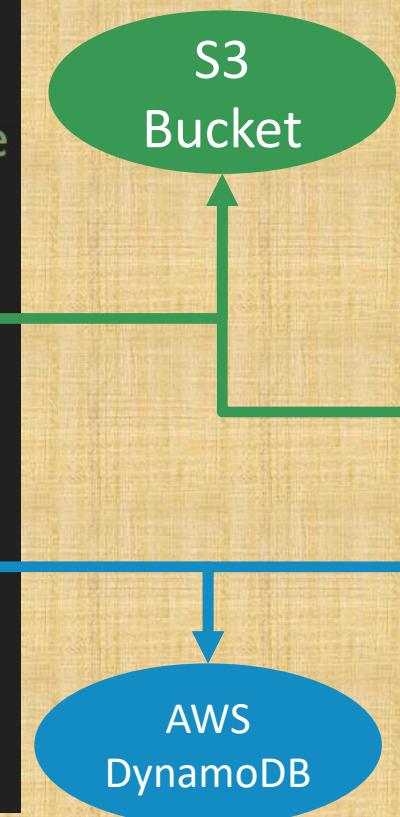
```
# Terraform Block
terraform {
    required_version = "~> 0.14" # which means any version
    required_providers {
        aws = {
            source  = "hashicorp/aws"
            version = "~> 3.0"
        }
    }
    # Adding Backend as S3 for Remote State Storage
    backend "s3" {
        bucket = "terraform-stack-simplify"
        key    = "dev/terraform.tfstate"
        region = "us-east-1"
    }
    # Enable during Step-09
    # For State Locking
    dynamodb_table = "terraform-dev-state-table"
}
```

# EKS Cluster

```
# Terraform Settings Block
terraform {
  required_version = ">= 1.0.0"
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "~> 3.63"
    }
  }
}

# Adding Backend as S3 for Remote State Storage
backend "s3" {
  bucket = "terraform-on-aws-eks"
  key    = "dev/eks-cluster/terraform.tfstate"
  region = "us-east-1"
}

# For State Locking
dynamodb_table = "dev-ekscluster"
}
```



# EKS k8s Resources

```
# Terraform Settings Block
terraform {
  required_version = ">= 1.0.0"
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "~> 3.63"
    }
    kubernetes = string
      source = "hashicorp/kubernetes"
      version = "~> 2.6.1"
    }
  }

# Adding Backend as S3 for Remote State Storage
backend "s3" {
  bucket = "terraform-on-aws-eks"
  key    = "dev/app1k8s/terraform.tfstate"
  region = "us-east-1"
}

# For State Locking
dynamodb_table = "dev-app1k8s"
}
```

# Terraform Remote State Datasource

The `terraform_remote_state` data source retrieves the root module output values from project-1 Terraform configuration, using the latest state snapshot from the remote backend.

Project-1

## Terraform Resources

1. AWS VPC
2. Bastion Host
3. EKS Cluster
4. EKS Node Groups
5. EKS Outputs

project-1/terraform.tfstate

In Section-11  
Demo

`terraform_remote_state`  
data source

## EKS Outputs from Project-1

1. cluster\_id
2. cluster\_endpoint
3. cluster\_certificate\_authority\_data
4. token

State Files in project local  
working directory

Project-2

## Terraform Resources

1. Kubernetes Provider
2. Kubernetes Deployment
3. Kubernetes Load Balancer Service (CLB)
4. Kubernetes Load Balancer Service (NLB)
5. Kubernetes NodePort Service

project-2/terraform.tfstate

# Terraform Remote State Datasource

Project-1

## Terraform Resources

1. AWS VPC
2. Bastion Host
3. EKS Cluster
4. EKS Node Groups
5. EKS Outputs

Project-1: `terraform.tfstate`



AWS S3



AWS DynamoDB

`terraform_remote_state`  
data source

## EKS Outputs from Project-1

1. `cluster_id`
2. `cluster_endpoint`
3. `cluster_certificate_authority_data`
4. `token`

Project-2

## Terraform Resources

1. Kubernetes Provider
2. Kubernetes Deployment
3. Kubernetes Load Balancer Service (CLB)
4. Kubernetes Load Balancer Service (NLB)
5. Kubernetes NodePort Service

Project-2: `terraform.tfstate`



AWS S3



AWS DynamoDB

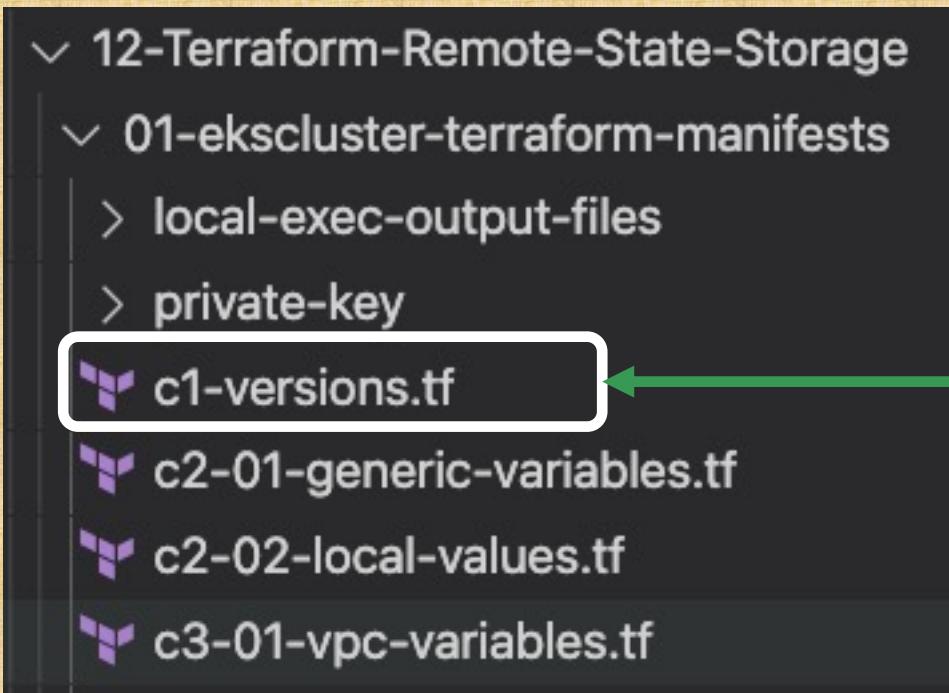
# Terraform Remote State Datasource

```
# Terraform Remote State Datasource - Remote Backend AWS S3
data "terraform_remote_state" "eks" {
    backend = "s3"
    config = {
        bucket = "terraform-on-aws-eks"
        key    = "dev/eks-cluster/terraform.tfstate"
        region = "us-east-1"
    }
}
```



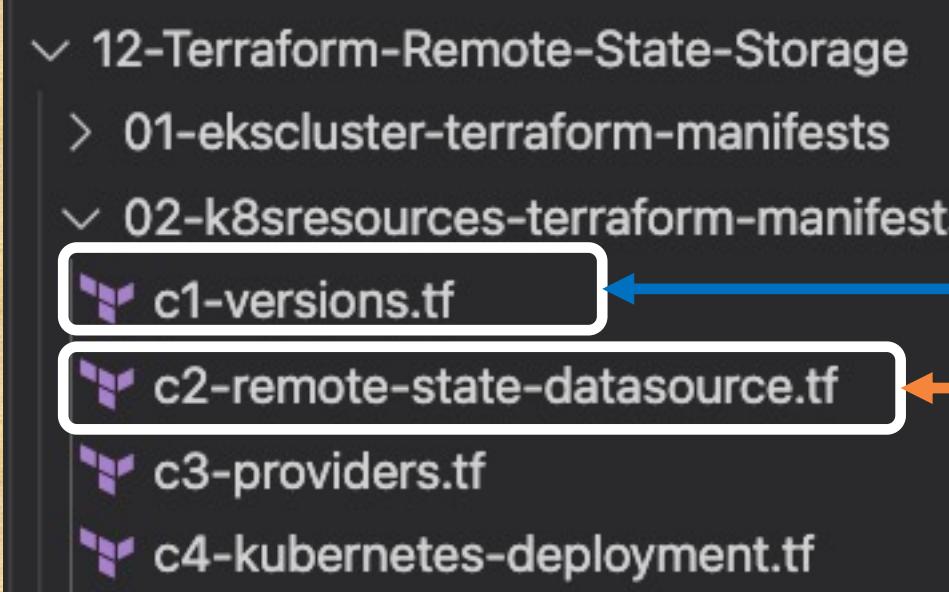
In Project-2, Terraform getting EKS Cluster information **from project-1** **terraform state file stored in AWS S3 Bucket** using Terraform Remote State Datasource.

# What are we going to learn ?



C1

Update Terraform Remote Backend Information for Project-1: EKS Cluster



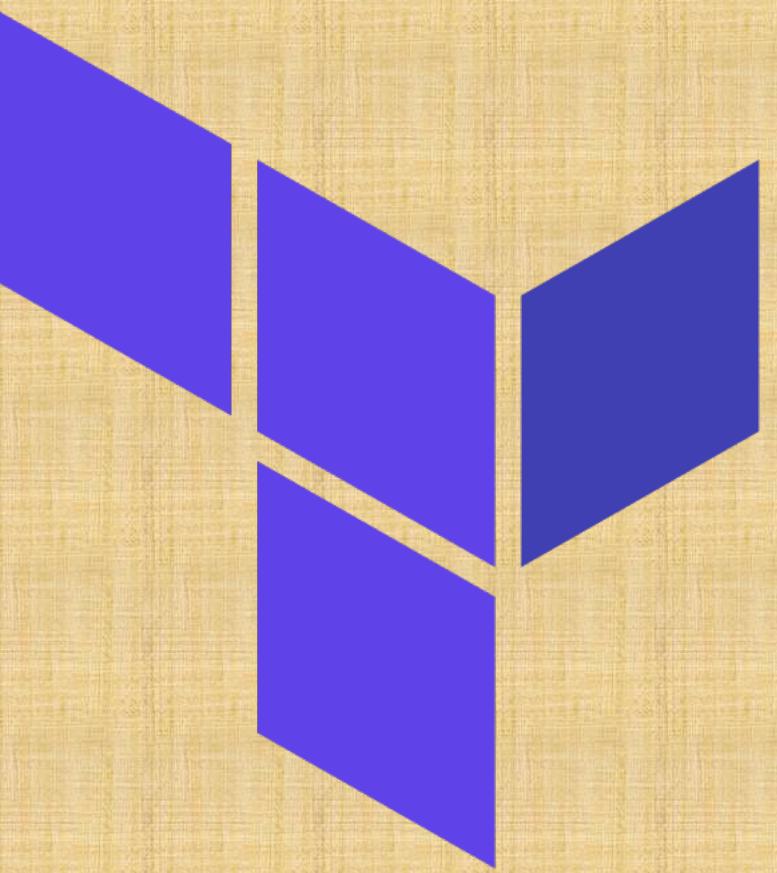
C1

Update Terraform Remote Backend Information for Project-2: Kubernetes Resources (Deployment & Services)

C2

Update Terraform Remote State Datasource from local to remote backend (AWS S3 Bucket) to access Project-1 EKS Cluster Outputs needed for Kubernetes Provider in Project-02

# Terraform Backends



# Terraform Backends

Each Terraform configuration can specify a **backend**, which defines where and how operations are performed, where **state** snapshots are stored, etc.

## Where Backends are Used

Backend configuration is only used by Terraform CLI.

Terraform Cloud and Terraform Enterprise always use their **own state storage** when performing **Terraform runs**, so they ignore any **backend block** in the configuration.

For Terraform Cloud users also it is always recommended to use **backend block** in Terraform configuration for commands like **terraform taint** which can be executed only using Terraform CLI

# Terraform Backends

## What Backends Do

1. Where state is stored
2. Where operations are performed.

## Store State

Terraform uses persistent state data to keep track of the resources it manages.

Everyone working with a given collection of infrastructure resources must be able to access the same state data (shared state storage).

## State Locking

State Locking is to prevent conflicts and inconsistencies when the operations are being performed

## Operations

"Operations" refers to performing API requests against infrastructure services in order to create, read, update, or destroy resources.

Not every terraform subcommand performs API operations; many of them only operate on state data.

Only two backends actually perform operations: local and remote.

The remote backend can perform API operations remotely, using Terraform Cloud or Terraform Enterprise.

What are Operations ?  
terraform apply  
terraform destroy

# Terraform Backends

## Backend Types

### Enhanced Backends

**Enhanced** backends can both **store state** and **perform operations**. There are only two enhanced backends: **local** and **remote**

Example for Remote Backend  
**Performing Operations** : Terraform Cloud, Terraform Enterprise

### Standard Backends

**Standard** backends **only store state**, and **rely** on the local backend for performing operations.

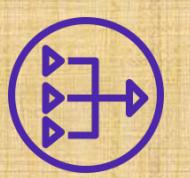
Example: AWS S3, Azure RM, Consul, etcd, gcs http and many more



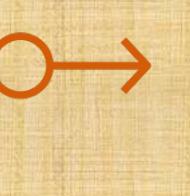
VPC



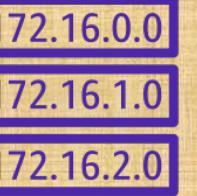
Internet gateway



NAT gateway



Elastic IP address



Route table



Public Subnet



Private Subnet



AWS S3



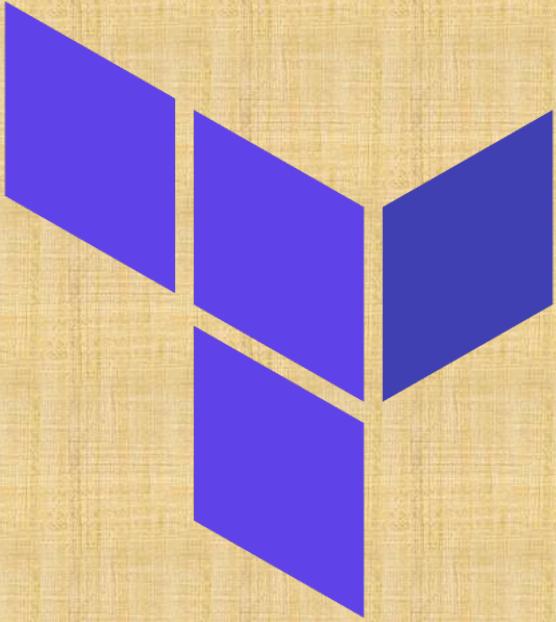
DynamoDB



# AWS EKS

## IRSA

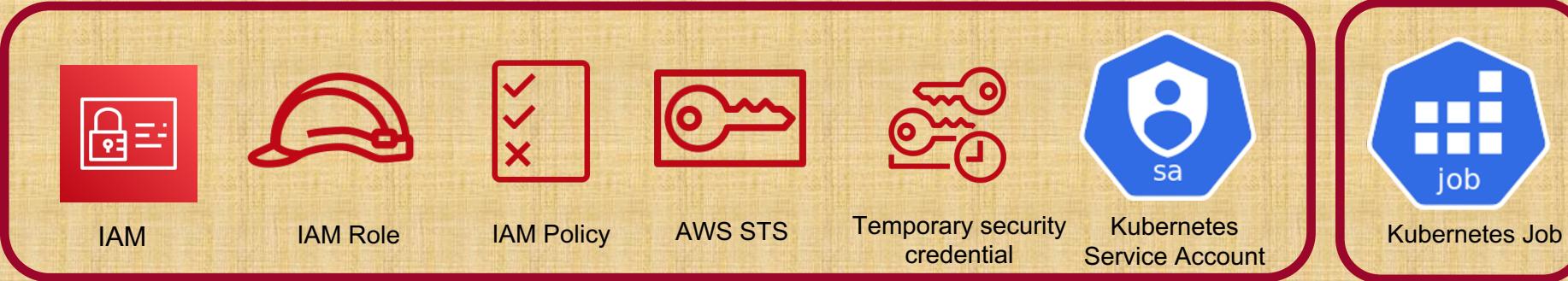
### IAM Roles for Service Accounts



EKS Cluster



EC2 VM

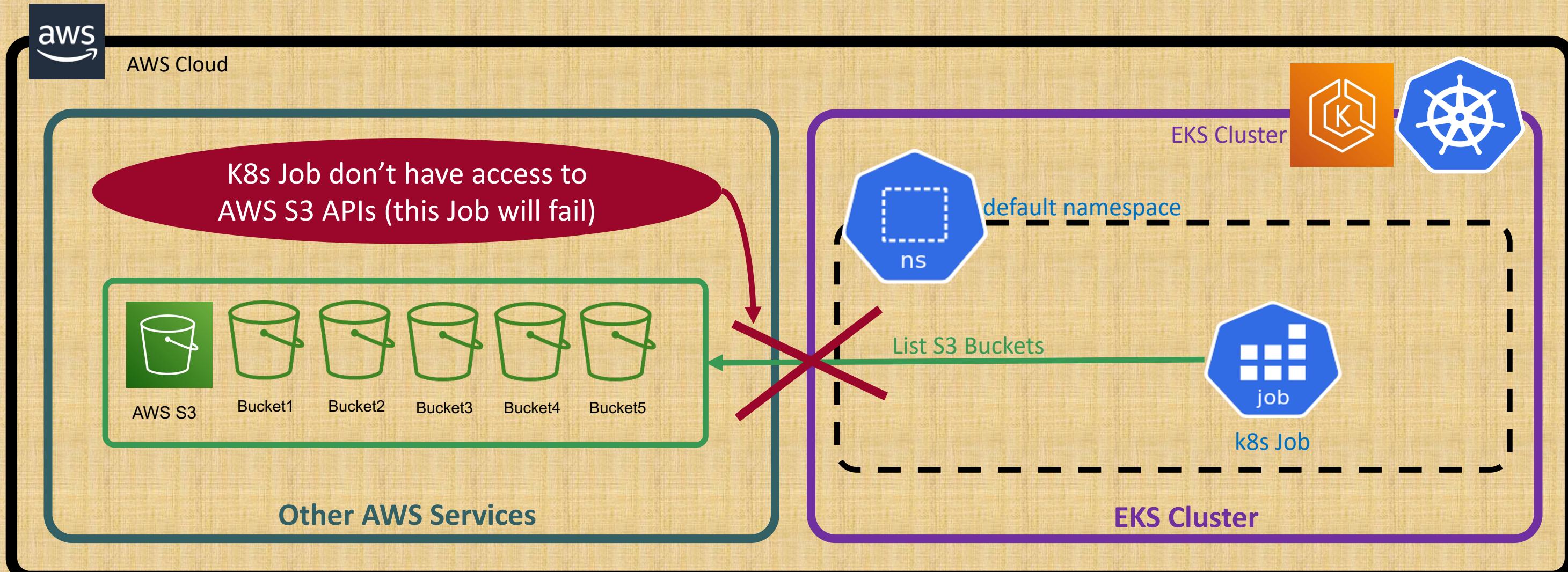


Autoscaling

VERY VERY IMPORTANT CONCEPT FOR AWS EKS

# How to access AWS Services from Workloads running in EKS Cluster?

**Demo Usecase:** Kubernetes Workload (k8s Job with AWS CLI Container) when run with a command “`s3 ls`”, it should list the S3 Buckets. In short, Kubernetes Workload in EKS Cluster should be able to access AW Services.



# Why do Kubernetes Workloads need to access AWS Services ?

EBS CSI Controller +  
Kubernetes Storage Class,  
PVC, PV

EFS CSI Controller +  
Kubernetes Storage Class +  
PVC + PV

AWS Load Balancer  
Controller + Kubernetes  
Ingress Resources

External DNS Controller

We can create / resize / delete / retain AWS EBS Volumes from  
Kubernetes Resources in EKS Cluster

We can map EFS File system to our Kubernetes Pods in our EKS  
Cluster.

We can create AWS Application, Network Load Balancer with all  
the settings from our EKS Cluster using k8s Ingress Resources

We can create / update / delete AWS Route53 DNS records using  
External DNS

For all the above, we need to understand the  
IRSA (IAM Roles for Service Accounts concept)  
in detail.

Same applies if our  
Application Pods want to  
access AWS Services from  
EKS Cluster

# Key Items for IRSA Implementation

AWS IAM Identity Provider

AWS STS AssumeRoleWithWebIdentity API operation

AWS IAM Temporary Role Credentials

IRSA IAM Roles for Service Accounts

EKS Cluster OpenID Connect Provider

Kubernetes Service Accounts

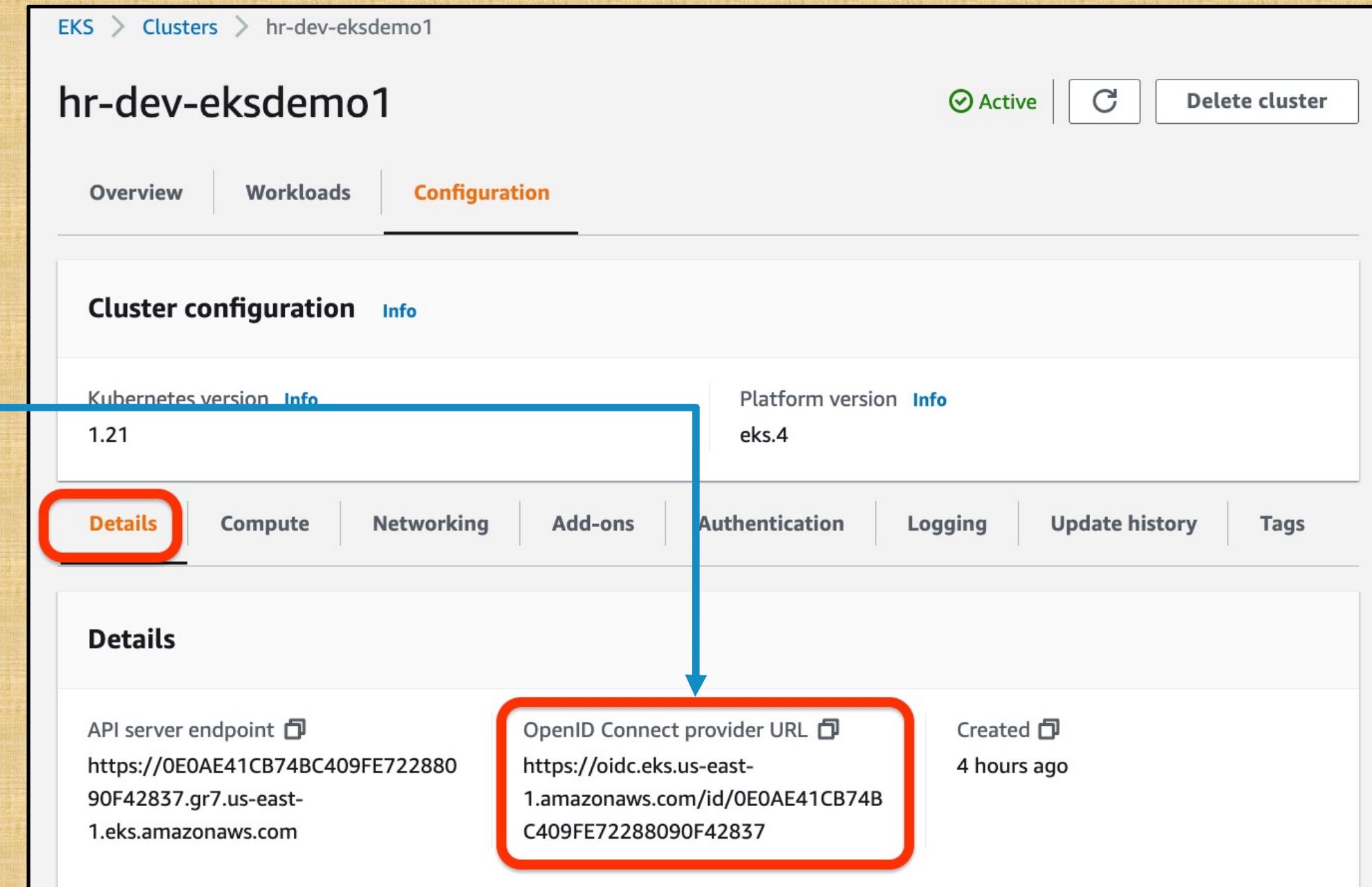
Kubernetes ProjectedServiceAccountToken feature (OIDC JSON Web Token)

# EKS Cluster OpenID Connect Provider URL

EKS Cluster acts as an OpenID Connect Provider  
(It can act as External Identity Provider to AWS IAM)

EKS Cluster contains the OpenID Connect Provider Issuer URL

We can configure this OpenID Connect Issuer URL in AWS IAM Identity Provider



# EKS Cluster OpenID Connect Configuration Endpoint

<https://<EKS Cluster OpenID Connect provider URL>/.well-known/openid-configuration>



The screenshot shows a browser window displaying the JSON configuration for an EKS cluster's OpenID Connect endpoint. The URL in the address bar is `oidc.eks.us-east-1.amazonaws.com/id/0E0AE41CB74BC409FE72288090F42837/.well-known/openid-configuration`. The JSON content is as follows:

```
// 20220107143403
// https://oidc.eks.us-east-1.amazonaws.com/id/0E0AE41CB74BC409FE72288090F42837/.well-known/openid-configuration

{
  "issuer": "https://oidc.eks.us-east-1.amazonaws.com/id/0E0AE41CB74BC409FE72288090F42837",
  "jwks_uri": "https://oidc.eks.us-east-1.amazonaws.com/id/0E0AE41CB74BC409FE72288090F42837/keys",
  "authorization_endpoint": "urn:kubernetes:programmatic_authorization",
  "response_types_supported": [
    "id_token"
  ],
  "subject_types_supported": [
    "public"
  ],
  "claims_supported": [
    "sub",
    "iss"
  ],
  "id_token_signing_alg_values_supported": [
    "RS256"
  ]
}
```

# AWS IAM Identity Provider

## Add EKS OpenID Connect Provider as Identity Provider in AWS IAM

The screenshot shows the AWS IAM Identity Providers page. On the left, the IAM navigation pane is visible with the 'Identity providers' option highlighted by a red box. The main content area displays the 'Identity providers (1)' section. A single provider entry is listed:

Provider	Type	Creation time
oidc.eks.us-east-1.amazonaws.com/id/0E0AE41CB74BC409FE72288090F42837	OpenID Connect	4 hours ago

A red box highlights the provider URL in the first column. The 'Add provider' button is located at the top right of the list.

# AWS IAM Identity Provider

## Add EKS OpenID Connect Provider as Identity Provider in AWS IAM

The screenshot shows the AWS IAM Identity Providers page. The left sidebar has a 'Identity providers' section highlighted with a red box. The main content area shows a summary of an identity provider named 'oidc.eks.us-east-1.amazonaws.com/id/0E0AE41CB74BC409FE72288090F42837'. The provider type is listed as 'OpenID Connect'. Below this, there's a table with columns for 'Provider', 'Provider Type', 'Creation Time', and 'ARN'. The ARN value is also highlighted with a red box. On the right, there are sections for 'Audiences' and 'Thumbprints', each with a single item listed and a 'Manage' button.

Provider	Provider Type	Creation Time	ARN
oidc.eks.us-east-1.amazonaws.com/id/0E0AE41CB74BC409FE72288090F42837	OpenID Connect	January 07, 2022, 09:40 (UTC+05:30)	arn:aws:iam::180789647333:oidc-provider/oidc.eks.us-east-1.amazonaws.com/id/0E0AE41CB74BC409FE72288090F42837

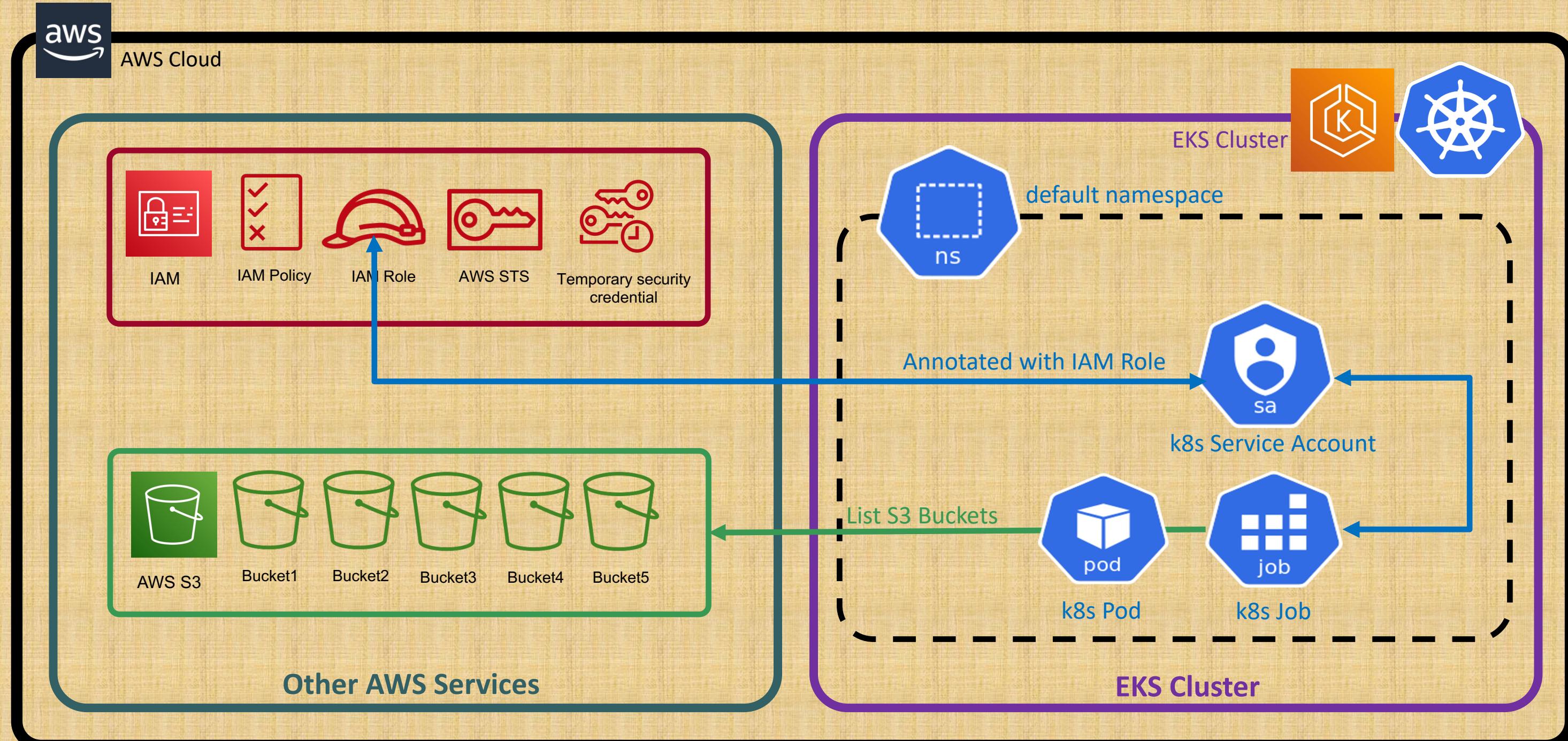
**Audiences (1)**  
Also known as client ID, audience is a value that identifies the application that is registered with an OpenID Connect provider.

Audience
sts.amazonaws.com

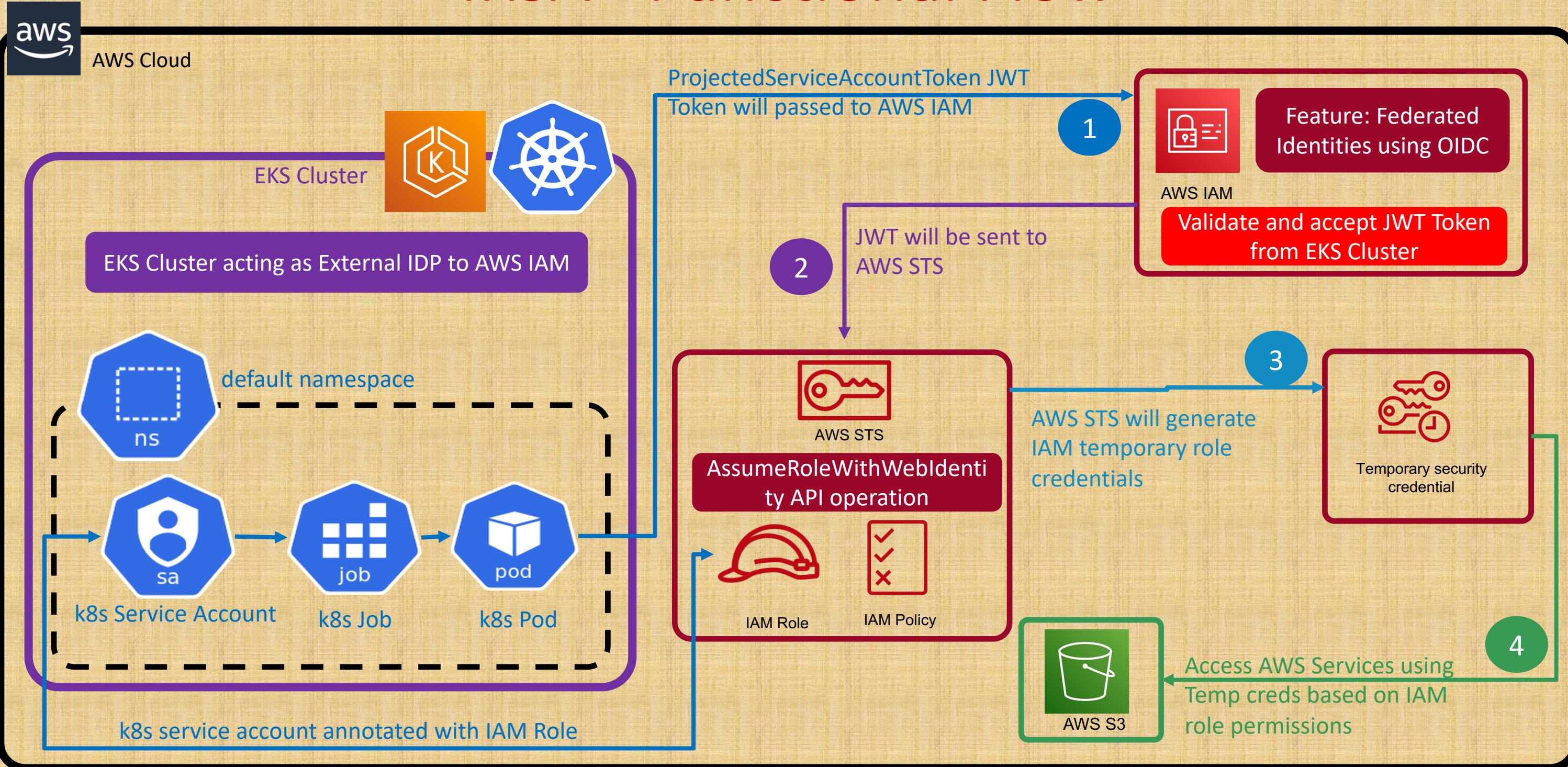
**Thumbprints (1)**  
Server certificate thumbprint is the hex-encoded SHA-1 hash value of the X.509 certificate used by the domain where the OpenID Connect provider makes its keys available.

9e99a48a9960b14926bb7f3b02e22da2b0ab7280
--

# AWS IAM Roles for Kubernetes Service Accounts



# IRSA - Functional Flow



# Kubernetes Service Account Mounted Secret - JWT Token

```
  et irsa-demo-sa-token-k2t6w
  Name:      irsa-demo-sa-token-k2t6w
  Namespace: default
  Labels:    <none>
  Annotations: kubernetes.io/service-account.name: irsa-demo-sa
                kubernetes.io/service-account.uid: 0a22696a-5aae-4af2-ae0c-eeb0eb5311fc
  Type:     kubernetes.io/service-account-token

  Data
  ====
  ca.crt:    1066 bytes
  namespace: 7 bytes
```

# Kubernetes Service Account Mounted Secret - JWT Token

The screenshot shows the jwt.io interface. At the top, it says "Crafted by auth0". Below that, there's a dropdown menu set to "RS256". The "Encoded" section contains a long base64 string of the JWT token. The "Decoded" section shows the token structure:

```
HEADER: ALGORITHM & TOKEN TYPE
{
  "alg": "RS256",
  "kid": "nwZR2-H101kUZGYT0U4rk1k00K3oPi_TEkvaTXg3cVY"
}

PAYLOAD: DATA
{
  "iss": "kubernetes/serviceaccount",
  "kubernetes.io/serviceaccount/namespace": "default",
  "kubernetes.io/serviceaccount/secret.name": "irsa-demo-sa-token-k2t6w",
  "kubernetes.io/serviceaccount/service-account.name": "irsa-demo-sa",
  "kubernetes.io/serviceaccount/service-account.uid": "0a22696a-5aae-4af2-ae0c-eeb0eb5311fc",
  "sub": "system:serviceaccount:default:irsa-demo-sa"
}

VERIFY SIGNATURE
RSASHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  Public Key in SPKI, PKCS #1,
  X.509 Certificate, or JWK string format.
  ,
  Private Key in PKCS #8, PKCS #
```

A blue box highlights the "sub" field in the payload section.

Decoded  
JWT Token

Website: <https://jwt.io/>

✓ 13-EKS-IRSA

✓ 01-ekscluster-terraform-manifests

- > .terraform
- > local-exec-output-files
- > private-key
- ≡ .terraform.lock.hcl
- └ c1-versions.tf
- └ c2-01-generic-variables.tf
- └ c2-02-local-values.tf
- └ c3-01-vpc-variables.tf
- └ c3-02-vpc-module.tf
- └ c3-03-vpc-outputs.tf

• •

- └ c6-01-iam-oidc-connect-provider-variables.tf
- └ c6-02-iam-oidc-connect-provider.tf
- └ ec2bastion.auto.tfvars
- └ eks.auto.tfvars
- └ terraform.tfvars
- └ vpc.auto.tfvars

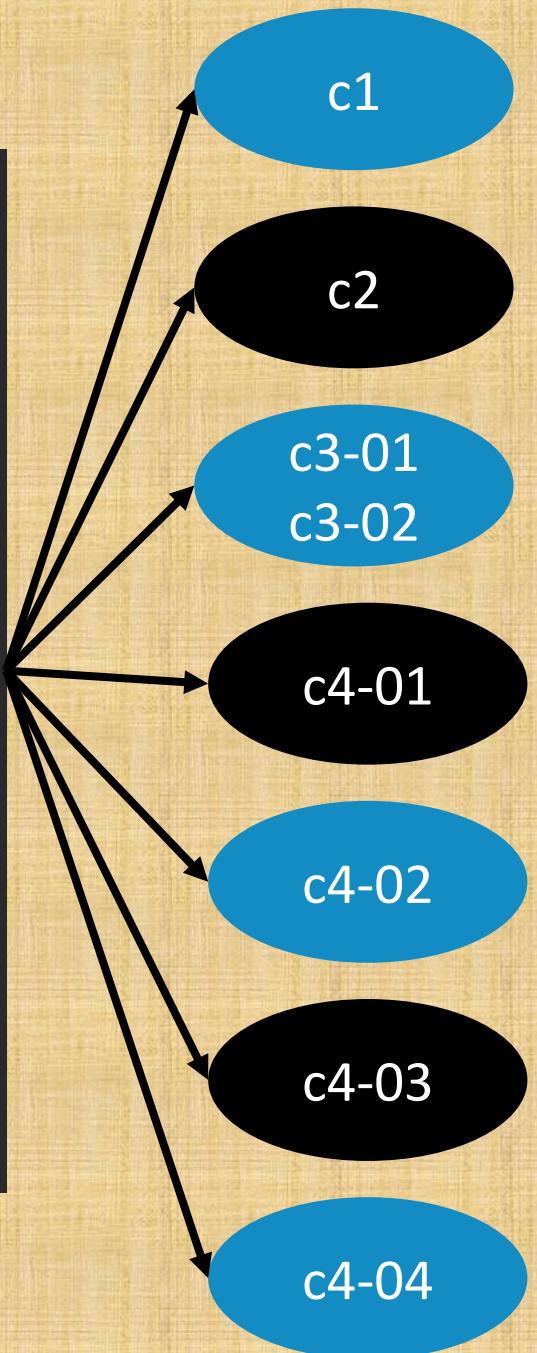
# What are we going to learn ?

Add EKS Cluster as OpenID Connect Provider in AWS IAM Service



# What are we going to learn ?

```
✓ 13-EKS-IRSA
  > 01-ekscluster-terraform-manifests
  ✓ 02-eks-irsa-demo-terraform-manifests
    ✓ c1-versions.tf
    ✓ c2-remote-state-datasource.tf
    ✓ c3-01-generic-variables.tf
    ✓ c3-02-local-values.tf
    ✓ c4-01-providers.tf
    ✓ c4-02-irsa-iam-policy-and-role.tf
    ✓ c4-03-irsa-k8s-service-account.tf
    ✓ c4-04-irsa-k8s-job.tf
    ✓ terraform.tfvars
```



Update Terraform Remote Backend information –  
AWS S3 Bucket Key & DynamoDB Table

Define Terraform Remote State Datasource for  
Project-1 EKS Cluster

Define generic variables and local values

Define AWS Provider and Kubernetes Provider

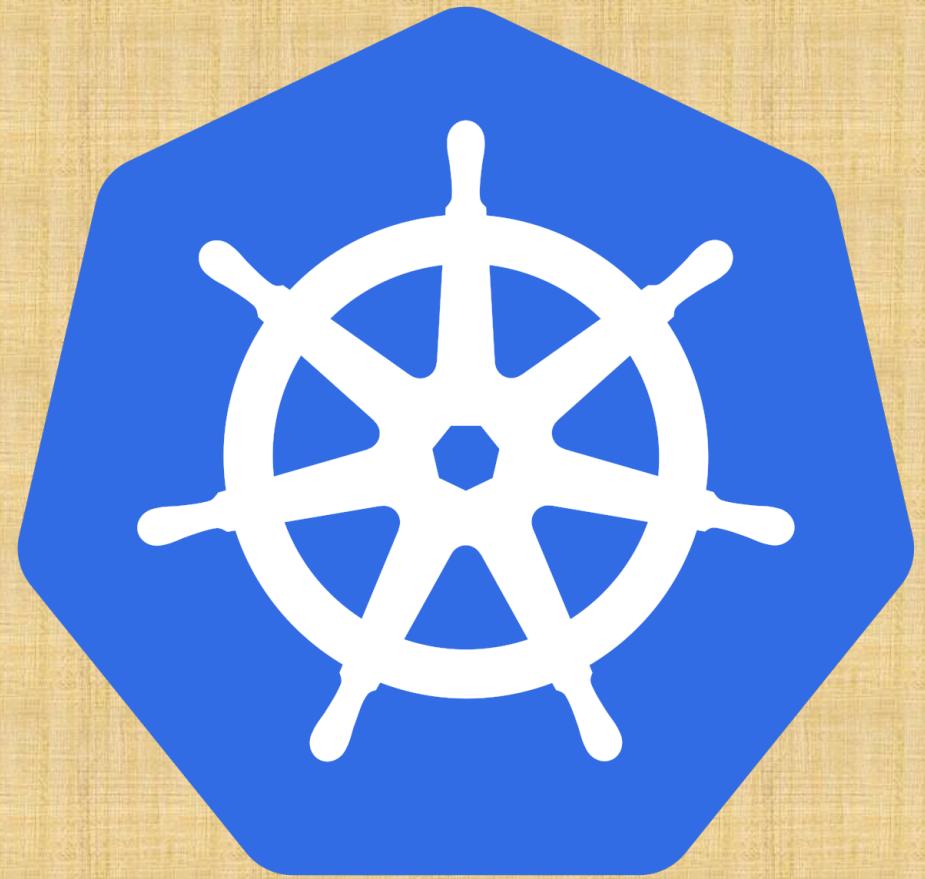
Create IAM Policy and IAM Role with  
Action: **AssumeRoleWithWebIdentity**

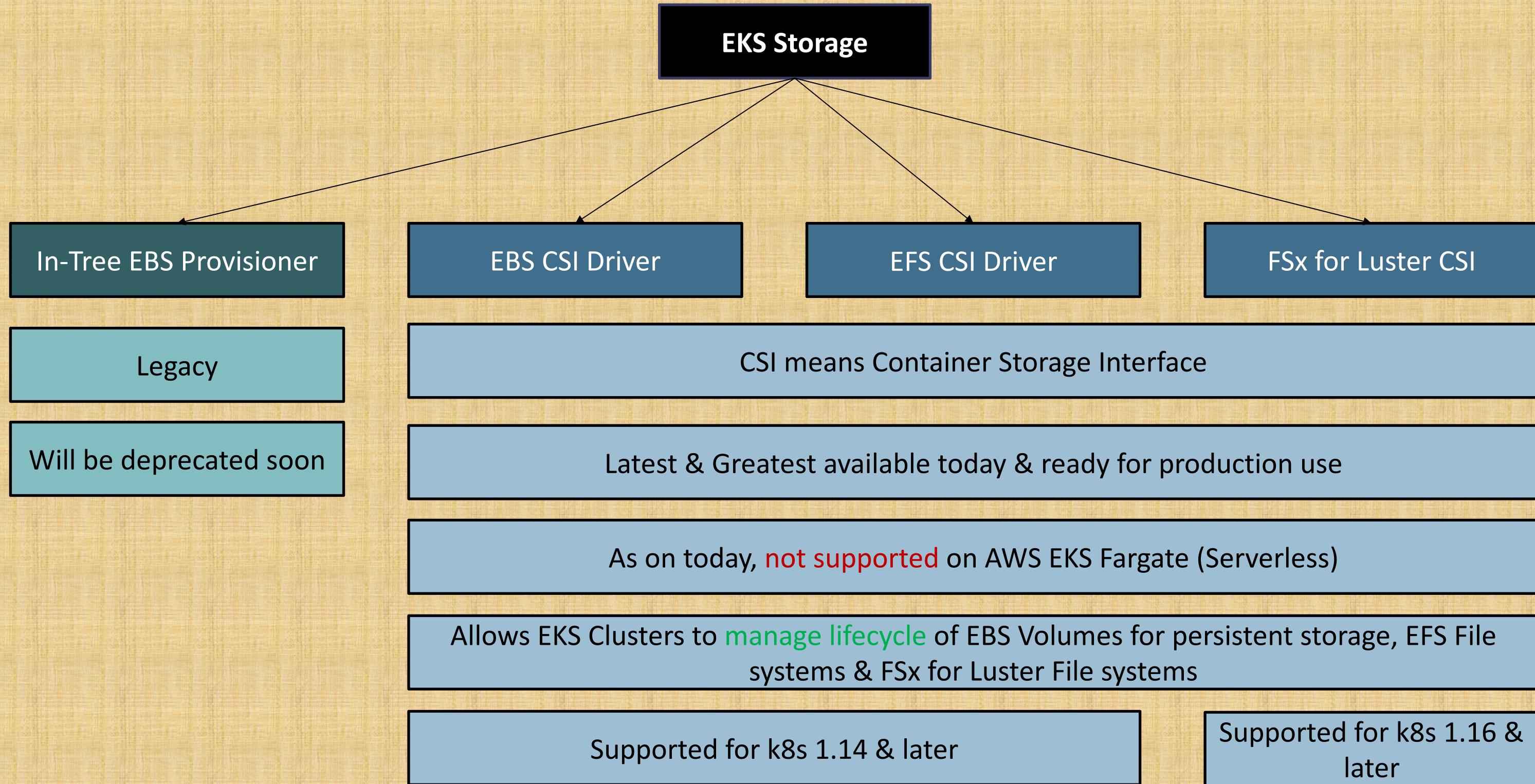
Create Kubernetes Service Account in EKS Cluster and  
annotate with IAM Role

Create **Kubernetes Job** with Kubernetes Service  
Account and AWS CLI Container



# AWS EKS Storage





# AWS Elastic Block Store - Introduction

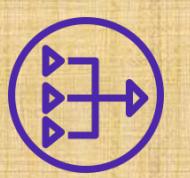
- EBS provides **block level storage volumes** for use with **EC2 & Container instances**.
- We can mount these **volumes as devices** on our EC2 & Container instances.
- EBS volumes that are attached to an instance are **exposed as storage volumes that persist independently** from the life of the EC2 or Container instance.
- We can **dynamically change** the configuration of a volume attached to an instance.
- AWS recommends EBS for data that must be **quickly accessible** and requires **long-term persistence**.
- EBS is well suited to both **database-style applications** that rely on random reads and writes, and to **throughput-intensive applications** that perform long, continuous reads and writes.



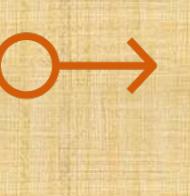
VPC



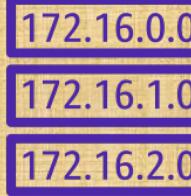
Internet gateway



NAT gateway



Elastic IP address



Route table



Public Subnet



Private Subnet



AWS S3



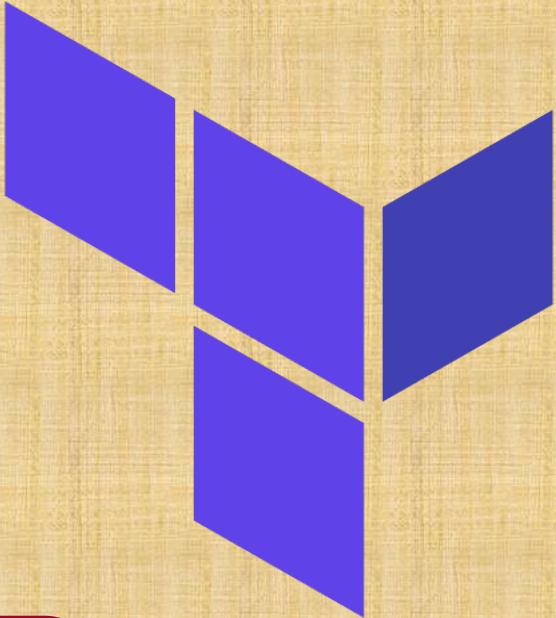
DynamoDB



# AWS EKS

## Storage

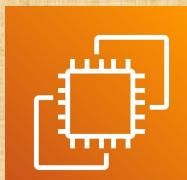
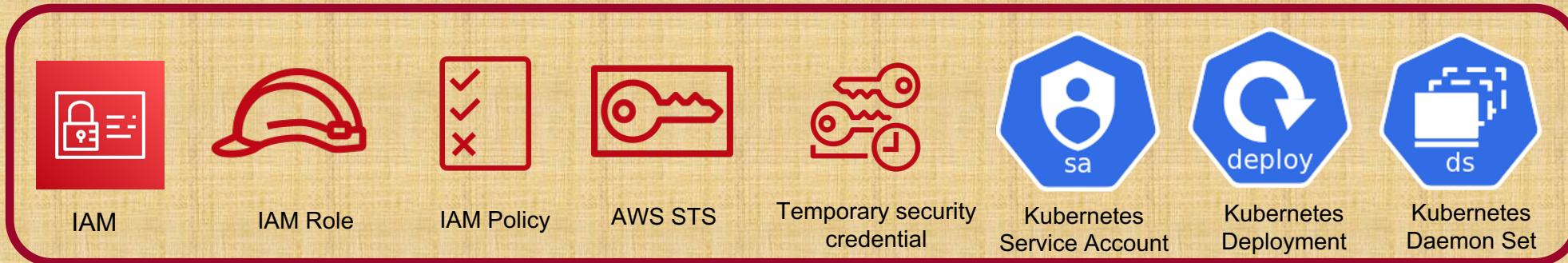
### Elastic Block Store



EKS Cluster



AWS EBS



EC2 VM



Autoscaling

**AWS EBS CSI Driver Install**

# AWS EBS CSI Driver

Why do we need to use EBS CSI Driver in EKS Cluster?

AWS EBS Container Storage Interface (CSI) driver allows EKS clusters to manage the lifecycle of EBS volumes for persistent volumes in k8s (Create / Resize / Delete Volumes).

# AWS EBS CSI Driver

## EBS CSI Driver - Deployment Options

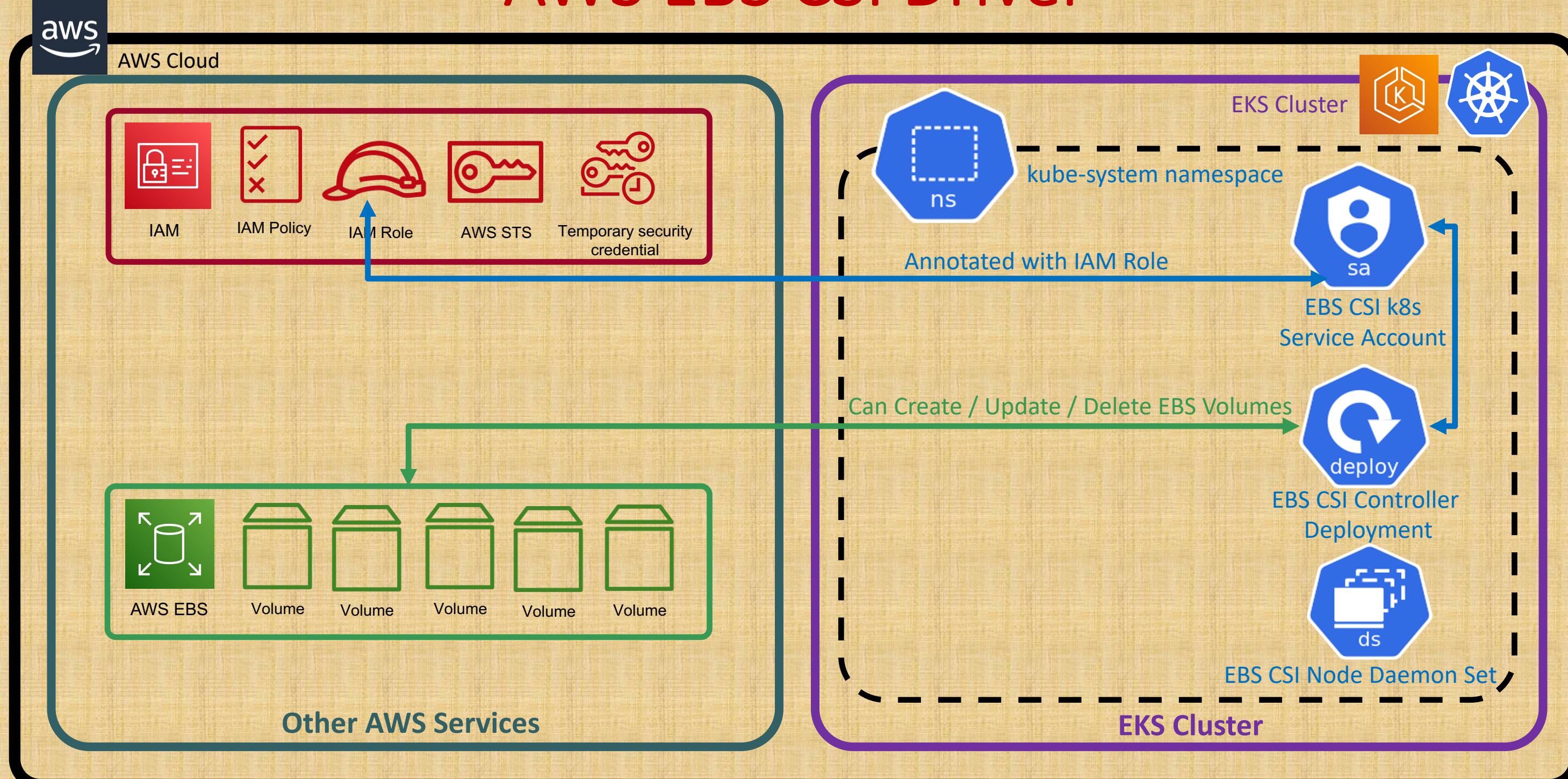
Managing the EBS  
CSI Driver self-  
managed add-on

Terraform Helm Provider

Managing the EBS  
CSI driver as an  
EKS add-on

Terraform EKS Add On  
Resource

# AWS EBS CSI Driver



✓ 13-EKS-IRSA

✓ 01-ekscluster-terraform-manifests

- > .terraform
- > local-exec-output-files
- > private-key
- ≡ .terraform.lock.hcl
- └ c1-versions.tf
- └ c2-01-generic-variables.tf
- └ c2-02-local-values.tf
- └ c3-01-vpc-variables.tf
- └ c3-02-vpc-module.tf
- └ c3-03-vpc-outputs.tf

• •

- └ c6-01-iam-oidc-connect-provider-variables.tf
- └ c6-02-iam-oidc-connect-provider.tf
- └ ec2bastion.auto.tfvars
- └ eks.auto.tfvars
- └ terraform.tfvars
- └ vpc.auto.tfvars

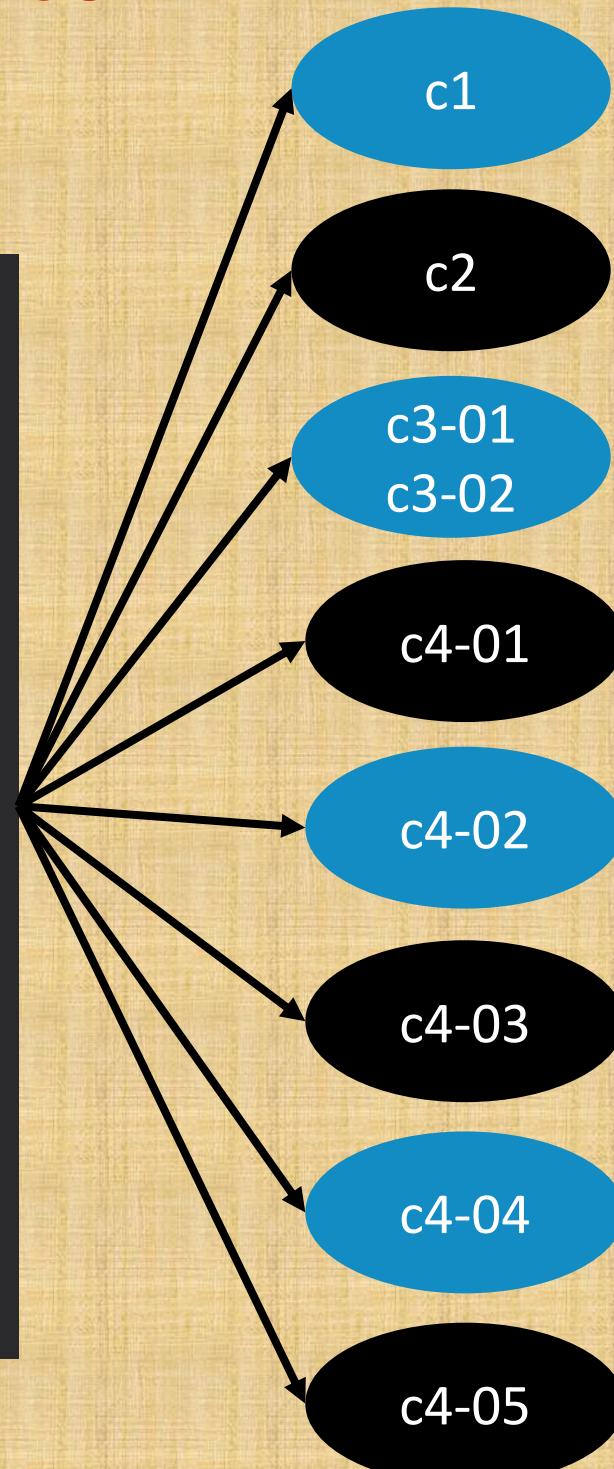
# What are we going to learn ?

Add EKS Cluster as OpenID Connect Provider in AWS IAM Service



# What are we going to learn ?

```
✓ 14-EBS-CSI-Install-Kubernetes-Storage
  > 01-ekscluster-terraform-manifests
  ✓ 02-ebs-terraform-manifests
    ✓ c1-versions.tf
    ✓ c2-remote-state-datasource.tf
    ✓ c3-01-generic-variables.tf
    ✓ c3-02-local-values.tf
    ✓ c4-01-ebs-csi-datasources.tf
    ✓ c4-02-ebs-csi-iam-policy-and-role.tf
    ✓ c4-03-ebs-csi-helm-provider.tf
    ✓ c4-04-ebs-csi-install-using-helm.tf
    ✓ c4-05-ebs-csi-outputs.tf
    ✓ terraform.tfvars
```



Update Terraform Remote Backend information – AWS S3 Bucket Key & DynamoDB Table

Define Terraform Remote State Datasource for Project-1 EKS Cluster

Define generic variables and local values

Define Terraform HTTP Datasource to download the EBS CSI IAM Role from Github

Create IAM Policy and IAM Role with AssumeRoleWithWebIdentity

Define Helm provider

Install EBS CSI Driver using Helm Resource

Create EBS CSI Outputs



VPC



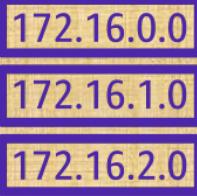
Internet gateway



NAT gateway



Elastic IP address



Route table



Public Subnet



Private Subnet



AWS S3



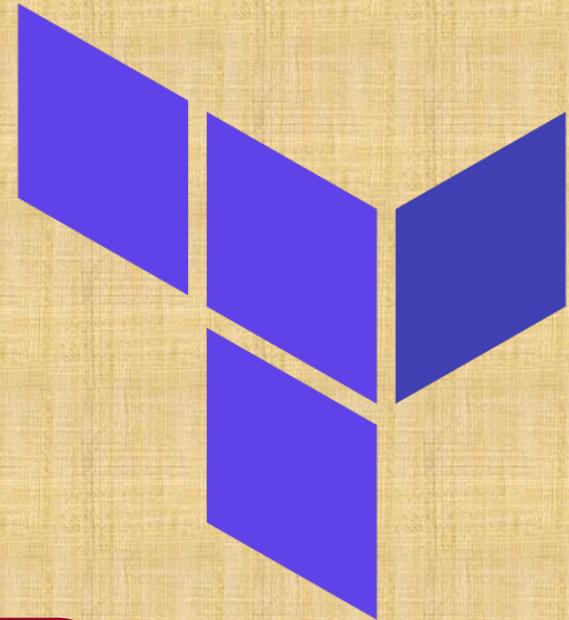
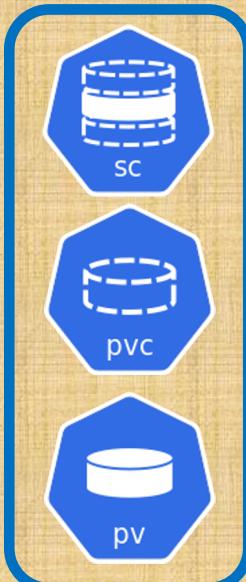
DynamoDB



# AWS EKS

## Storage

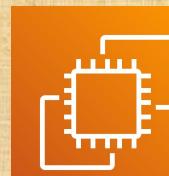
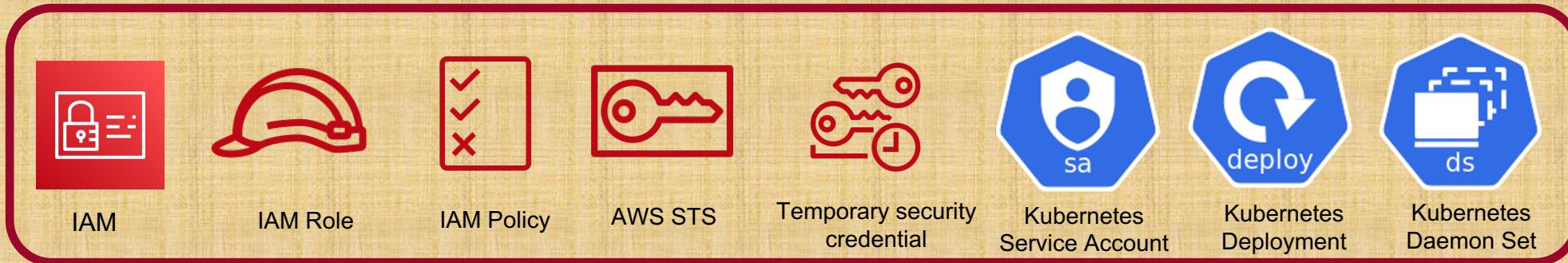
### Elastic Block Store



EKS Cluster



AWS EBS



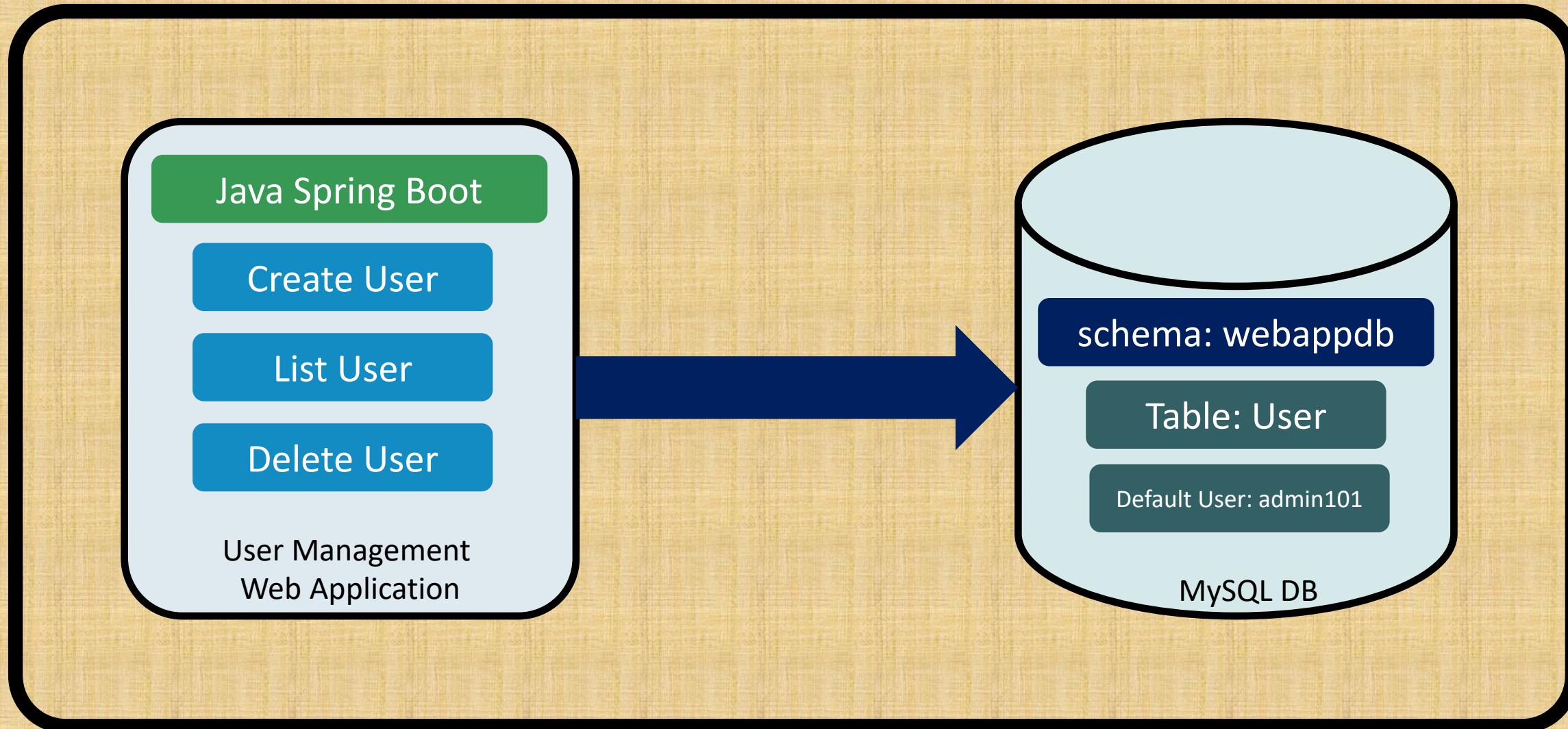
EC2 VM



Autoscaling

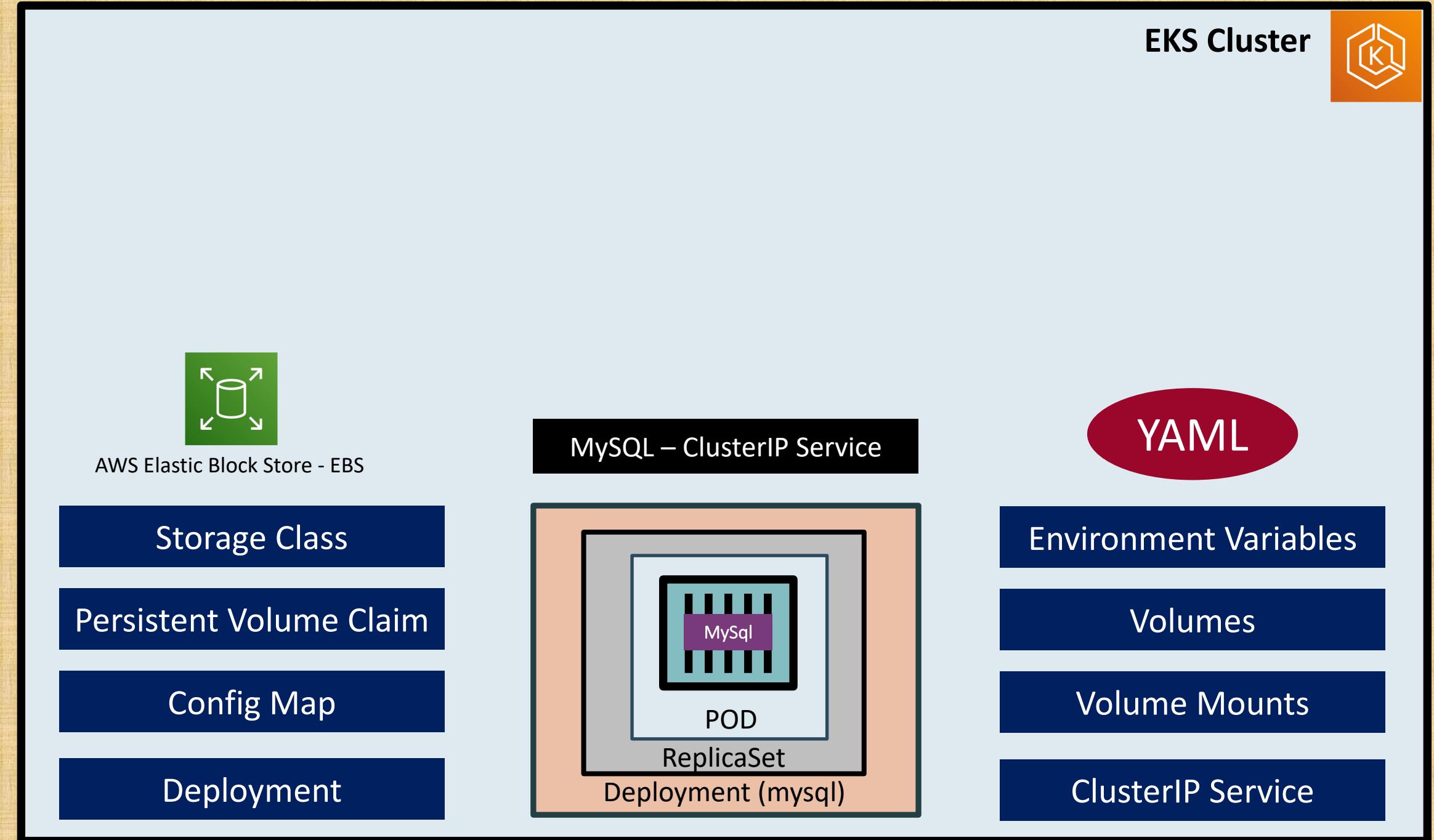
**AWS EBS CSI Driver – Sample App using YAML**

# User Management Web Application - Architecture

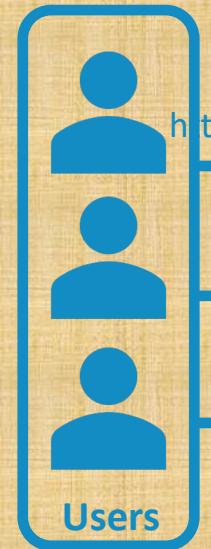


# EKS Storage with EBS CSI Driver - Demo

## EKS Storage EBS CSI Driver

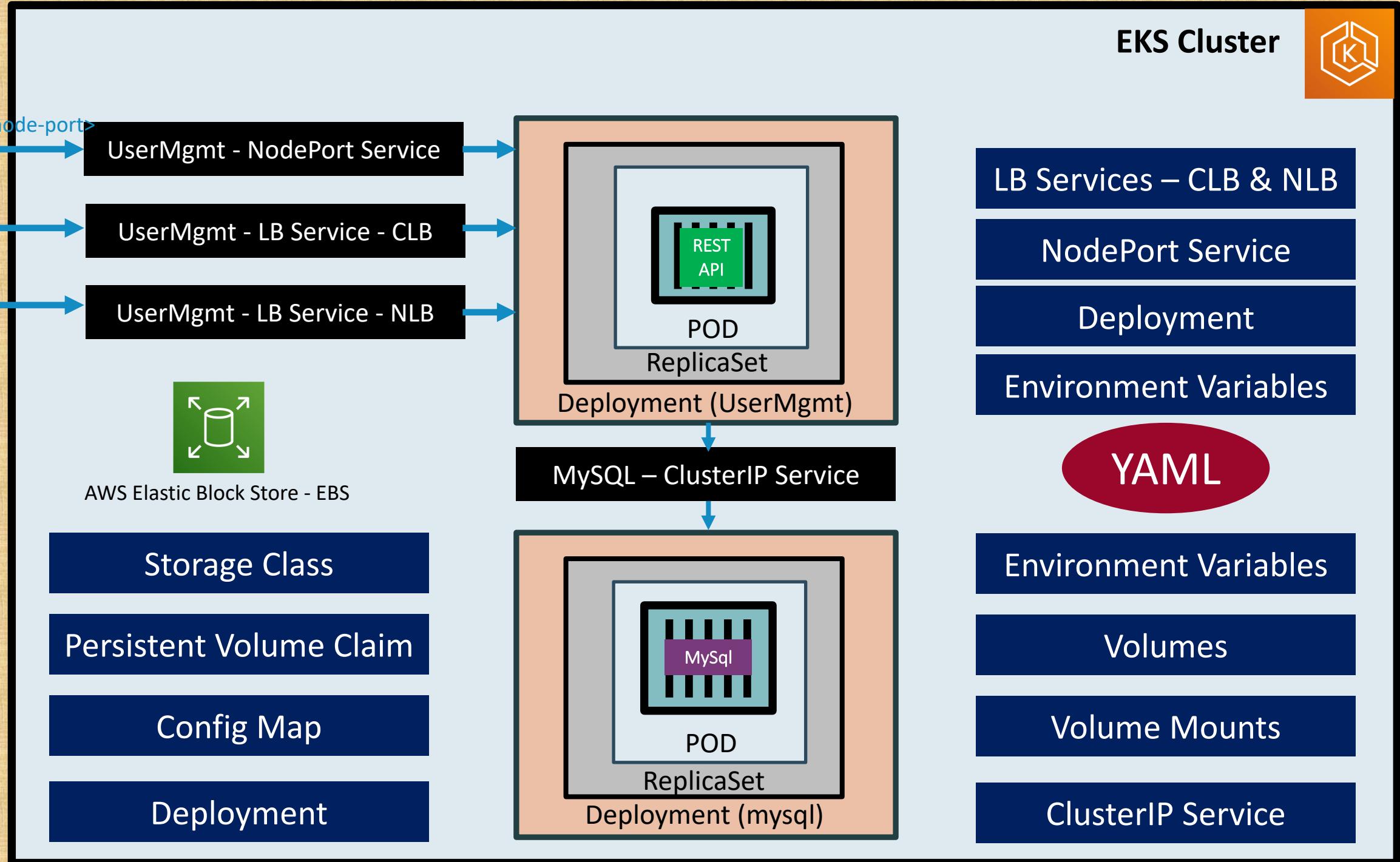


# EKS Storage with EBS CSI Driver - Demo

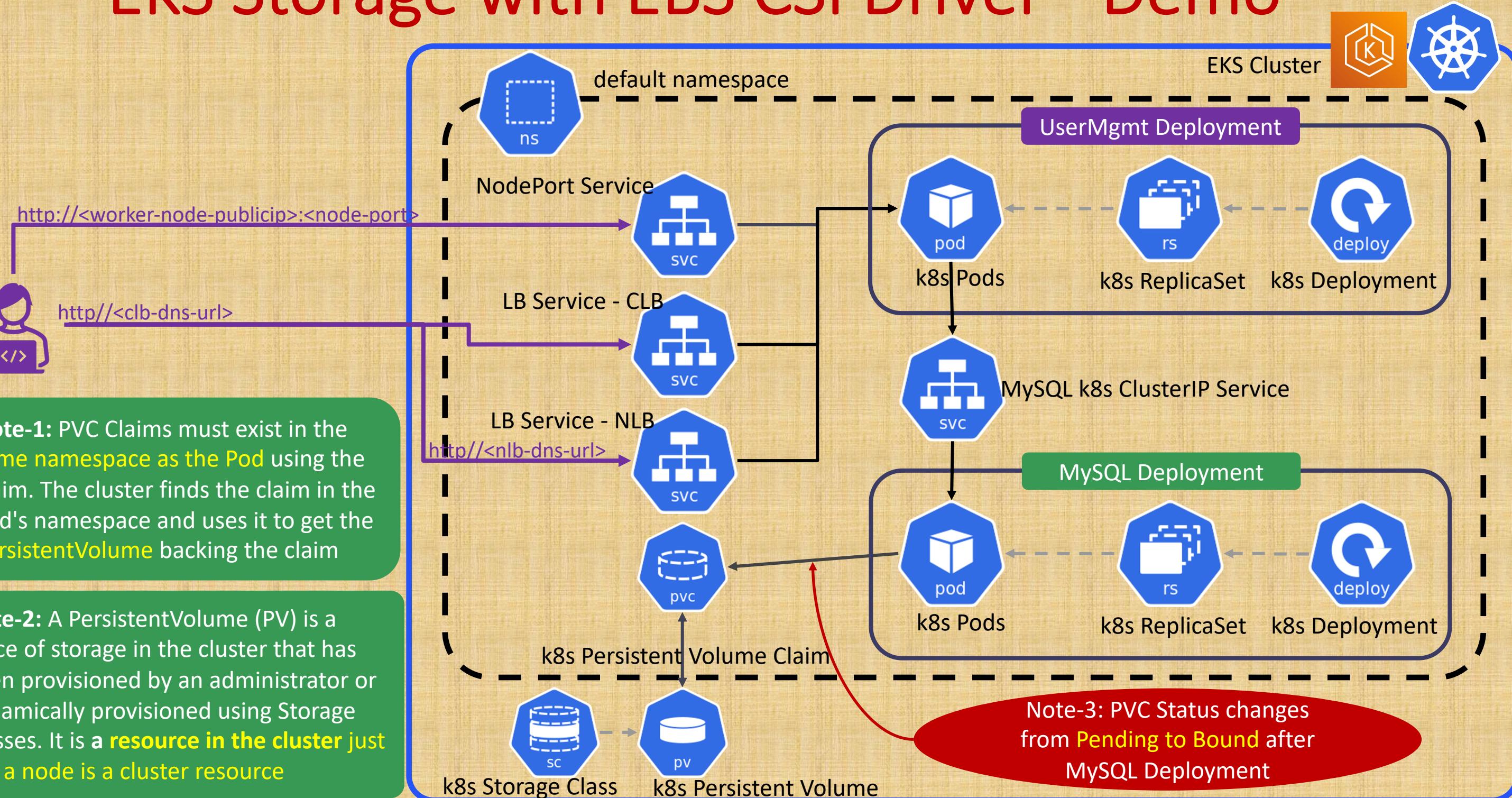


http://<workernode-publicip>:<node-port>  
http://<CLB-DNS-URL>  
http://<NLB-DNS-URL>

## EKS Storage EBS CSI Driver



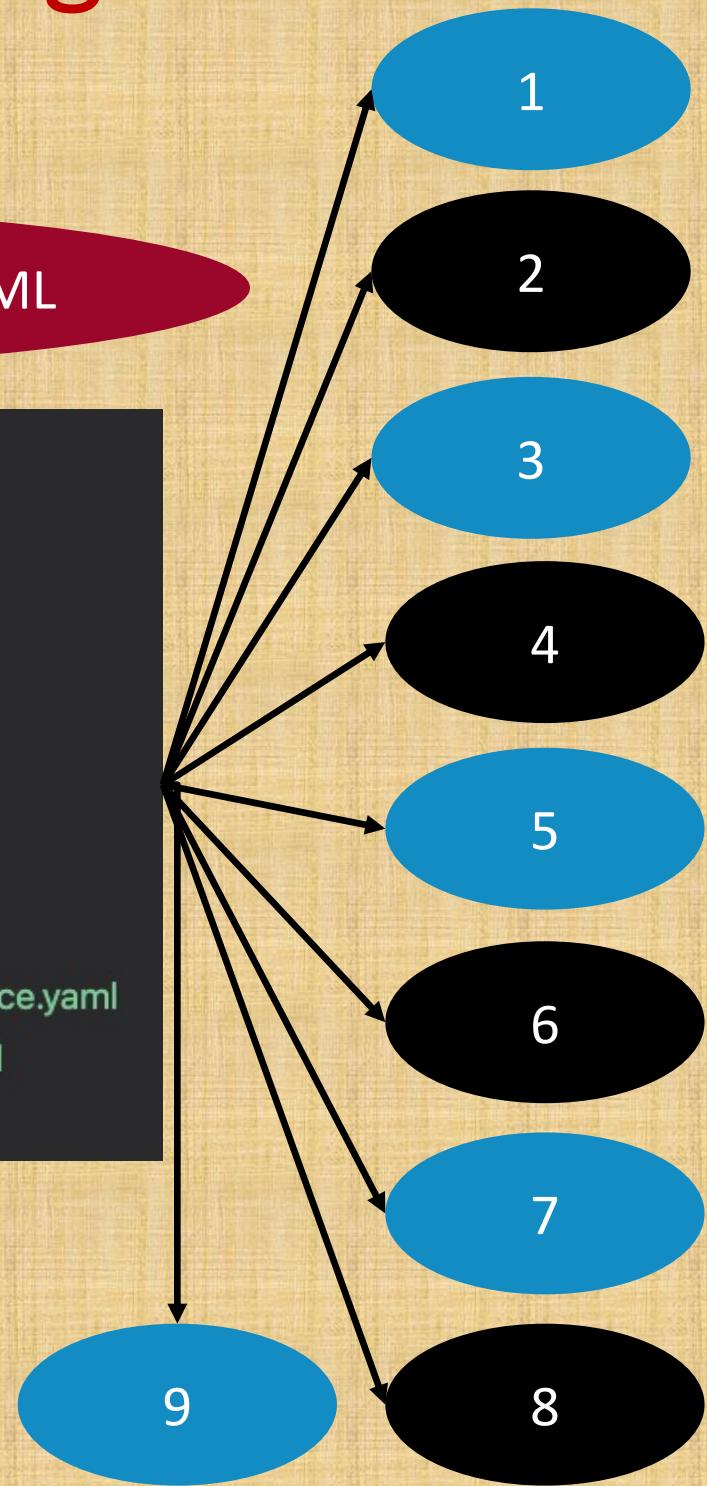
# EKS Storage with EBS CSI Driver - Demo



# What are we going to learn ?

## Kubernetes Manifests using YAML

```
✓ 15-EBS-Kubernetes-SampleApp-YAML
  > 01-ekscluster-terraform-manifests
  ✓ 02-kube-manifests-UMS-WebApp
    ! 01-storage-class.yaml
    ! 02-persistent-volume-claim.yaml
    ! 03-UserManagement-ConfigMap.yaml
    ! 04-mysql-deployment.yaml
    ! 05-mysql-clusterip-service.yaml
    ! 06-UserMgmtWebApp-Deployment.yaml
    ! 07-UserMgmtWebApp-Classic-LoadBalancer-Service.yaml
    ! 08-UserMgmtWebApp-Network-LoadBalancer.yaml
    ! 09-UserMgmtWebApp-NodePort-Service.yaml
```



Create Kubernetes Node Port Service

Create Kubernetes Storage Class

Create Kubernetes Persistent Volume Claim

Create Kubernetes Config Map which will create the database schema in MySQL DB

Create Kubernetes Deployment for MySQL DB

Create Cluster IP Service for MySQL DB

Create Kubernetes Deployment for UserMgmt WebApp

Create Kubernetes Service of type Load Balancer (AWS Classic Load Balancer)

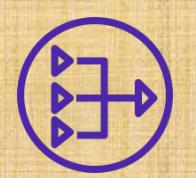
Create Kubernetes Service of type Load Balancer (AWS Network Load Balancer)



VPC



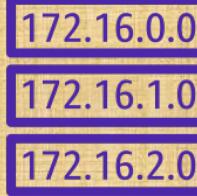
Internet gateway



NAT gateway



Elastic IP address



Route table



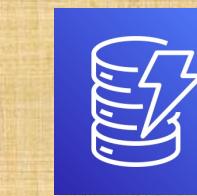
Public Subnet



Private Subnet



AWS S3



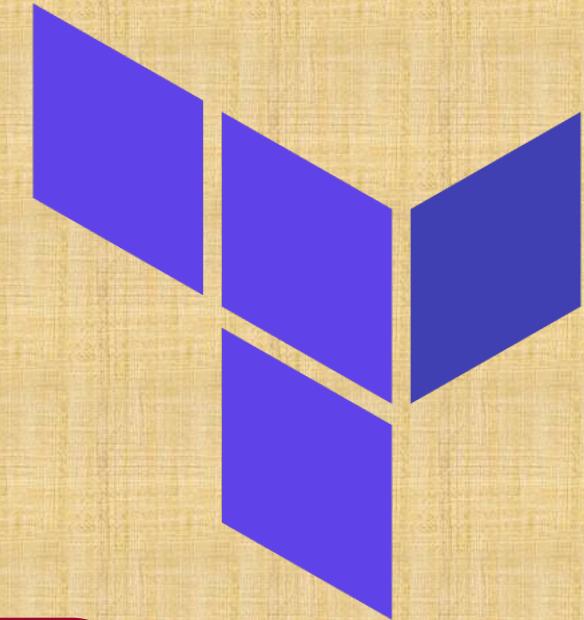
DynamoDB



# AWS EKS

## Storage

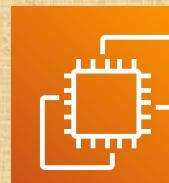
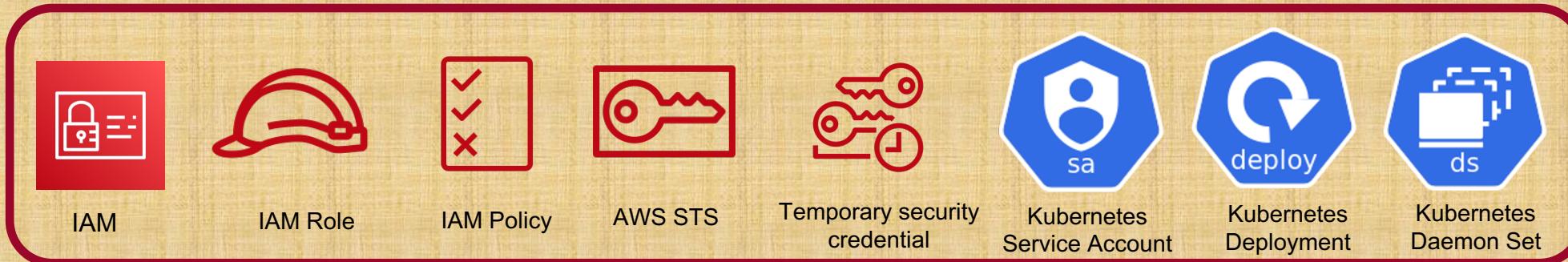
### Elastic Block Store



EKS Cluster



AWS EBS



EC2 VM

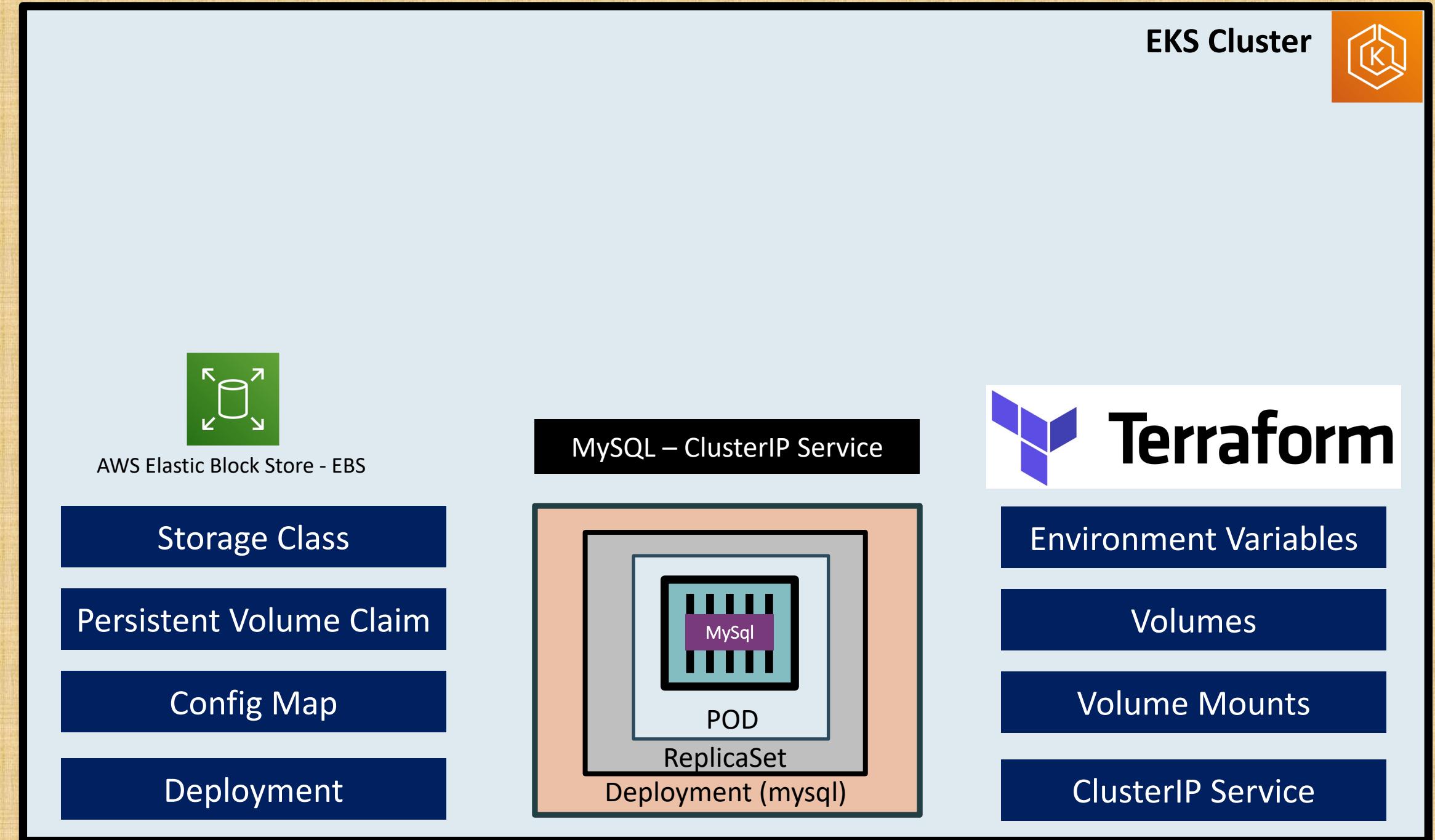


Autoscaling

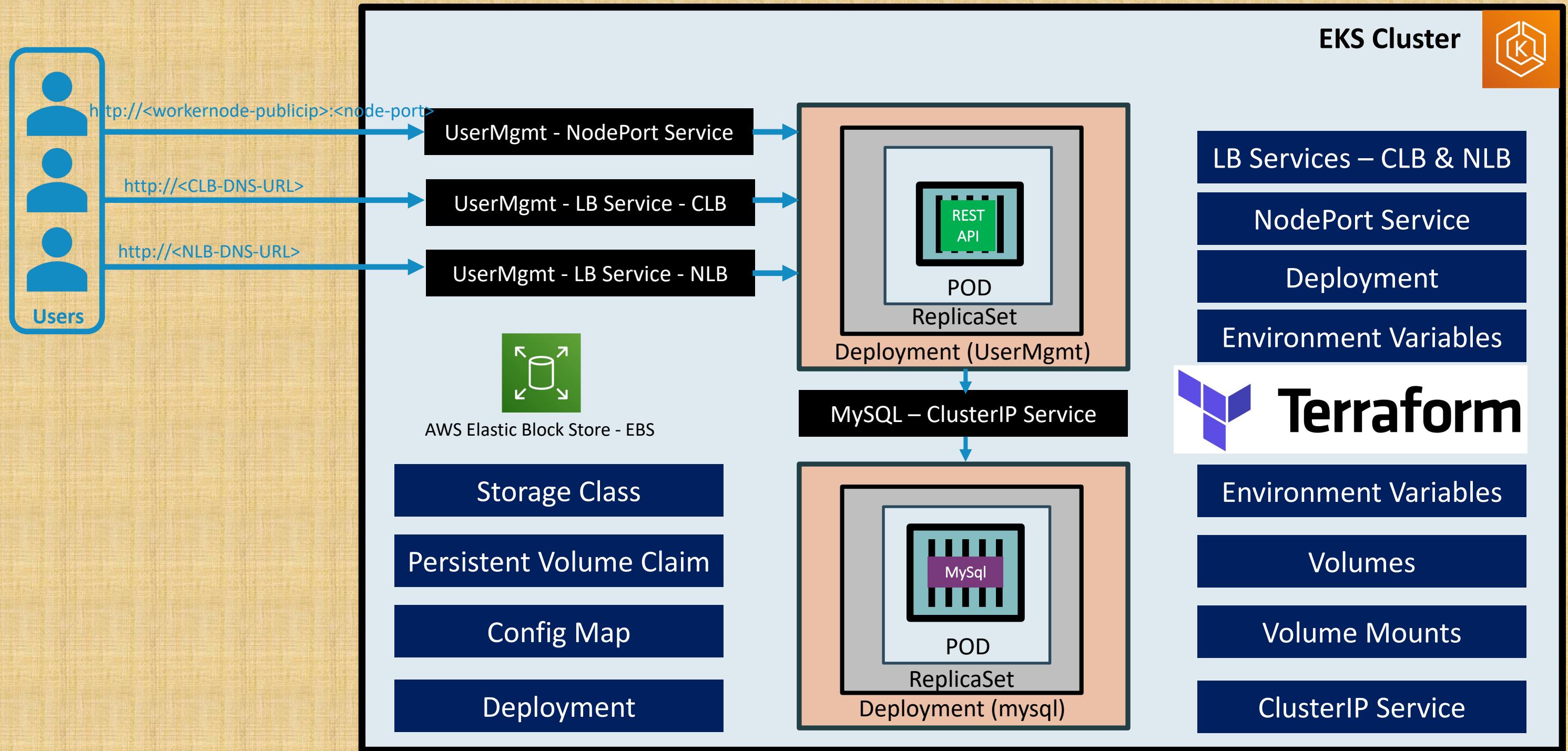
**AWS EBS CSI Driver – Sample App using Terraform**

# EKS Storage with EBS CSI Driver - Demo

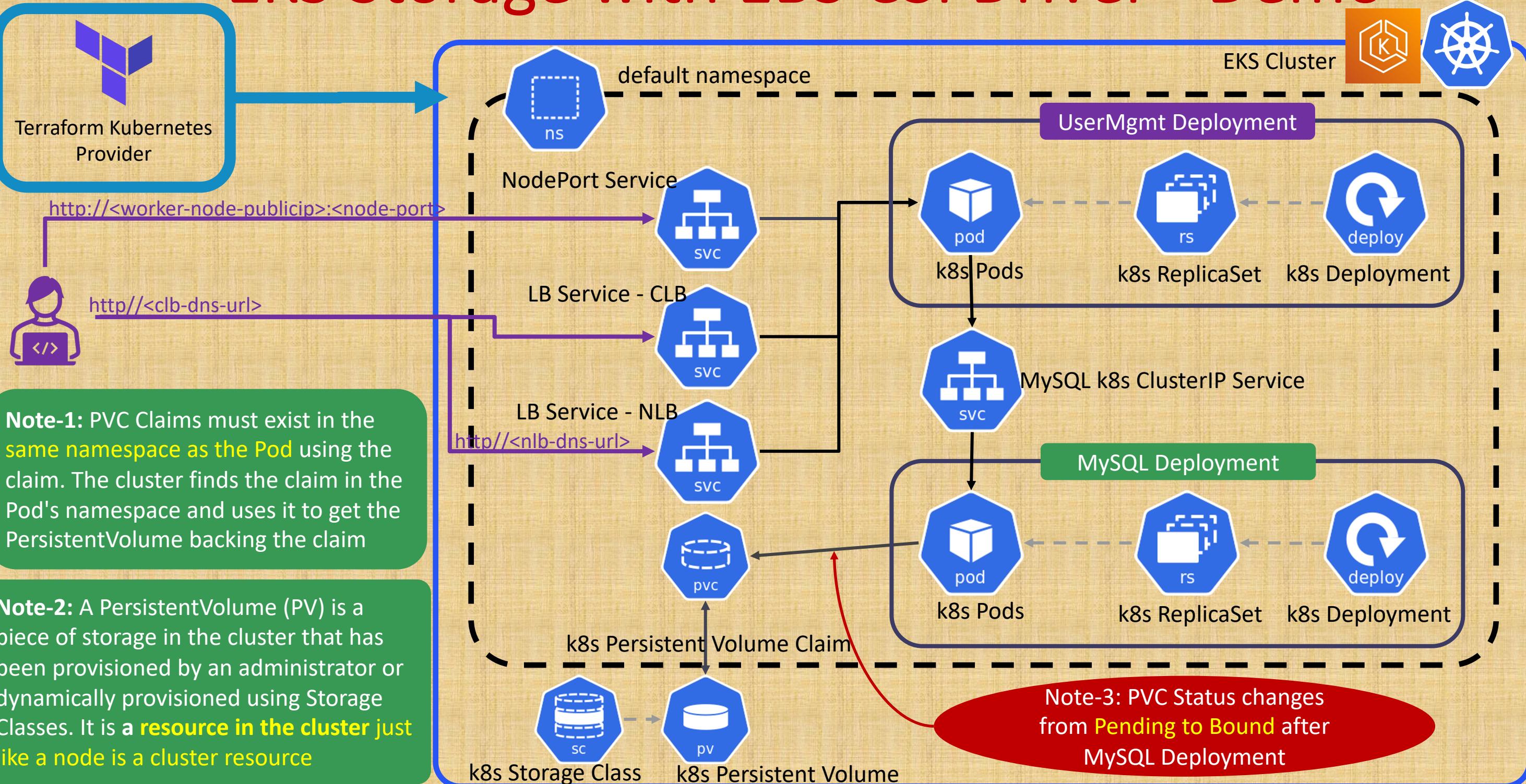
## EKS Storage EBS CSI Driver



# EKS Storage with EBS CSI Driver - Demo



# EKS Storage with EBS CSI Driver - Demo



# What are we going to learn ?

## Kubernetes Manifests using Terraform

- ✓ 16-EBS-Kubernetes-SampleApp-Terraform
  - > 01-ekscluster-terraform-manifests
  - > 02-ebs-terraform-manifests
  - ✓ 03-terraform-manifests-UMS-WebApp
    - c1-versions.tf
    - c2-remote-state-datasource.tf
    - c3-providers.tf
    - c4-01-storage-class.tf
    - c4-02-persistent-volume-claim.tf
    - c4-03-UserMgmtWebApp-ConfigMap.tf
    - c4-04-mysql-deployment.tf
    - c4-05-mysql-clusterip-service.tf
    - c4-06-UserMgmtWebApp-deployment.tf
    - c4-07-UserMgmtWebApp-loadbalancer-service.tf
    - c4-08-UserMgmtWebApp-network-loadbalancer-service.tf
    - c4-09-UserMgmtWebApp-nodeport-service.tf
    - webappdb.sql

c1

c2

c3

c4-01

c4-02

c4-03

c4-04

c4-05

Update Terraform Remote Backend information – AWS S3 Bucket Key & DynamoDB Table

Define Terraform Remote State Datasource for Project-1 EKS Cluster

Define AWS & Kubernetes Terraform Providers

Create Kubernetes Storage Class

Create Kubernetes Persistent Volume Claim

Create Kubernetes Config Map which will create the database schema in MySQL DB

Create Kubernetes Deployment for MySQL DB

Create Kubernetes Cluster IP Service for MySQL DB

# What are we going to learn ?

## Kubernetes Manifests using Terraform

```
✓ 16-EBS-Kubernetes-SampleApp-Terraform
  > 01-ekscluster-terraform-manifests
  > 02-ebs-terraform-manifests
  ✓ 03-terraform-manifests-UMS-WebApp
    ↴ c1-versions.tf
    ↴ c2-remote-state-datasource.tf
    ↴ c3-providers.tf
    ↴ c4-01-storage-class.tf
    ↴ c4-02-persistent-volume-claim.tf
    ↴ c4-03-UserMgmtWebApp-ConfigMap.tf
    ↴ c4-04-mysql-deployment.tf
    ↴ c4-05-mysql-clusterip-service.tf
    ↴ c4-06-UserMgmtWebApp-deployment.tf
    ↴ c4-07-UserMgmtWebApp-loadbalancer-service.tf
    ↴ c4-08-UserMgmtWebApp-network-loadbalancer-service.tf
    ↴ c4-09-UserMgmtWebApp-nodeport-service.tf
  ⚡ webappdb.sql
```

c4-06

c4-07

c4-08

c4-09

sql file

Create Kubernetes Deployment for UserMgmt WebApp

Create Kubernetes Service of type Load Balancer (AWS Classic Load Balancer)

Create Kubernetes Service of type Load Balancer (AWS Network Load Balancer)

Create Kubernetes Node Port Service

Create webappdb.sql file which contains the DB Schema creation queries used in k8s configMap c4-03



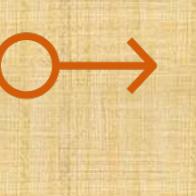
VPC



Internet gateway



NAT gateway



Elastic IP address



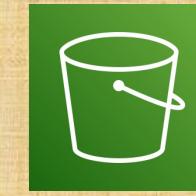
Route table



Public Subnet



Private Subnet



AWS S3



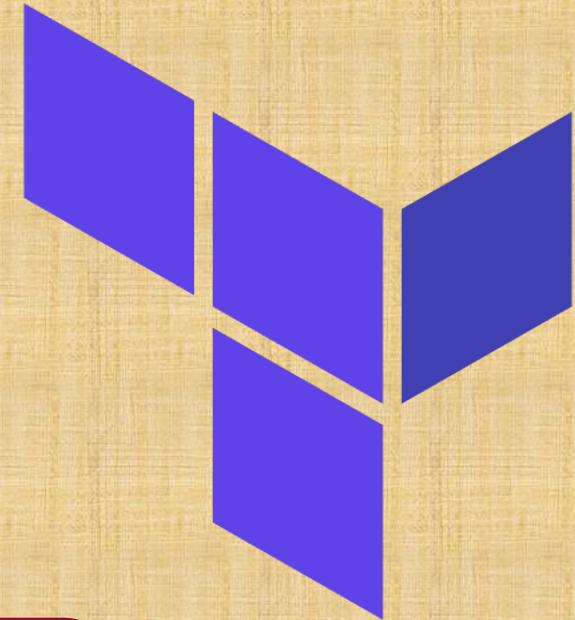
DynamoDB



# AWS EKS

## Storage

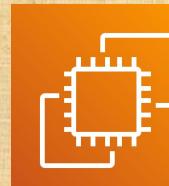
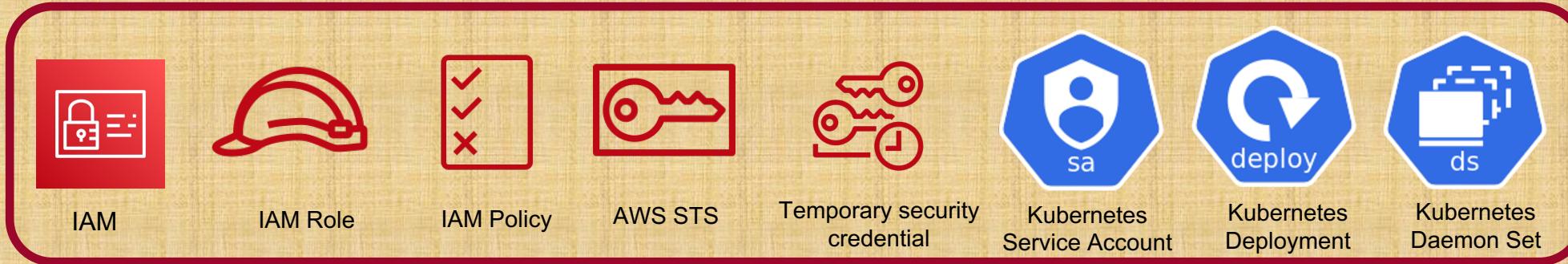
### Elastic Block Store



EKS Cluster



AWS EBS



EC2 VM



Autoscaling

**EBS Volumes – Resize and Retain Features**

# EBS Volumes - Resize & Retain Features

```
c4-01-storage-class.tf ×
terraform-on-aws-eks > 17-EBS-Resizing-on-EKS > 03-terraform-manifests-UMS-WebApp > c4-01-storage-class.tf

1 # Resource: Kubernetes Storage Class
2 resource "kubernetes_storage_class_v1" "ebs_sc" {
3     metadata {
4         name = "ebs-sc"
5     }
6     storage_provisioner = "ebs.csi.aws.com"
7     volume_binding_mode = "WaitForFirstConsumer"
8     allow_volume_expansion = "true"
9     reclaim_policy = "Retain" # Additional Reference:
10 }
```

# What are we going to learn ?

```
✓ 17-EBS-Resizing-on-EKS
  > 01-ekscluster-terraform-manifests
  > 02-ebs-terraform-manifests
  ✓ 03-terraform-manifests-UMS-WebApp
    > .terraform
    ≡ .terraform.lock.hcl
    ▾ c1-versions.tf
    ▾ c2-remote-state-datasource.tf
    ▾ c3-providers.tf
    ▾ c4-01-storage-class.tf
    ▾ c4-02-persistent-volume-claim.tf
    ▾ c4-03-UserMgmtWebApp-ConfigMap.tf
    ▾ c4-04-mysql-deployment.tf
    ▾ c4-05-mysql-clusterip-service.tf
    ▾ c4-06-UserMgmtWebApp-deployment.tf
    ▾ c4-07-UserMgmtWebApp-loadbalancer-service.tf
    ▾ c4-08-UserMgmtWebApp-network-loadbalancer-service.tf
    ▾ c4-09-UserMgmtWebApp-nodeport-service.tf
  ┌─ webappdb.sql
```

Update Kubernetes Storage Class arguments to support Volume Expansion (**Resizing Volumes**) and Reclaim Policy as **Retain**.

Default Reclaim Policy is **Delete** if not defined when PVC is deleted, volume will be deleted automatically

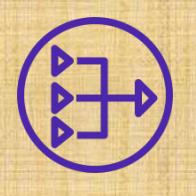
**Very Important Note:** As part of Volume Resizing, we can only increase the size of EBS Volume. We cannot **reduce** the Volume size



VPC



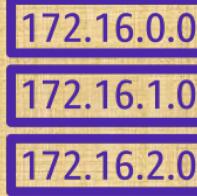
Internet gateway



NAT gateway



Elastic IP address



Route table



Public Subnet



Private Subnet



AWS S3



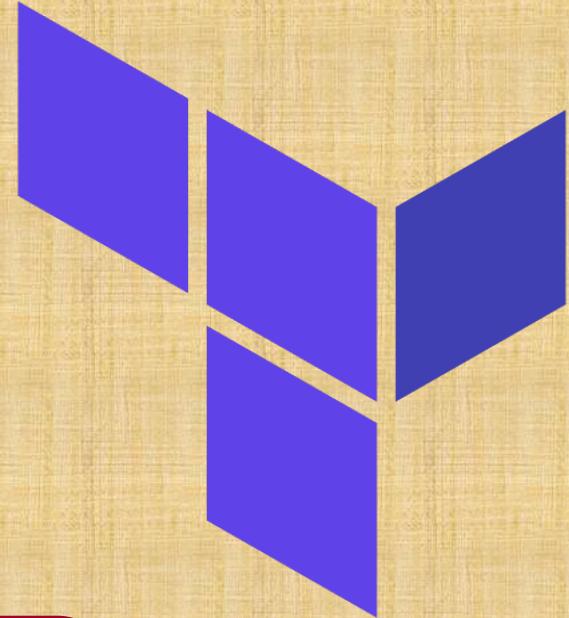
DynamoDB



# AWS EKS

## Storage

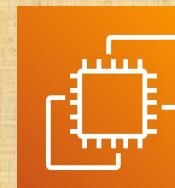
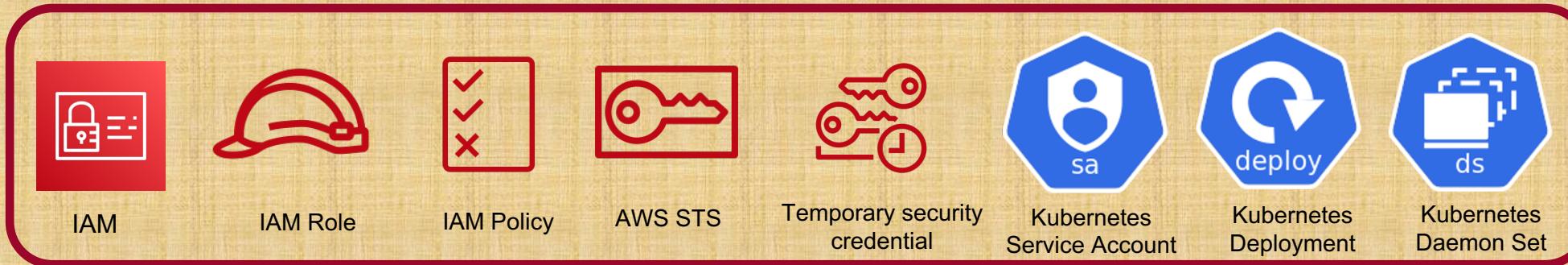
### Elastic Block Store



EKS Cluster



AWS EBS



EC2 VM



Autoscaling

Managing the EBS CSI driver as an EKS add-on

# AWS EBS CSI Driver

## EBS CSI Driver - Deployment Options

Managing the EBS  
CSI Driver self-  
managed add-on  
using HELM

Managing the EBS  
CSI driver as an  
EKS add-on

# EBS CSI driver as an EKS add-on

Has good number of **limitations** as on today unlike **Self-Managed Add-On** approach using HELM (Section-14).

In **future**, this might become the **default deployment** option for EBS CSI Driver

Currently it is in **preview mode**

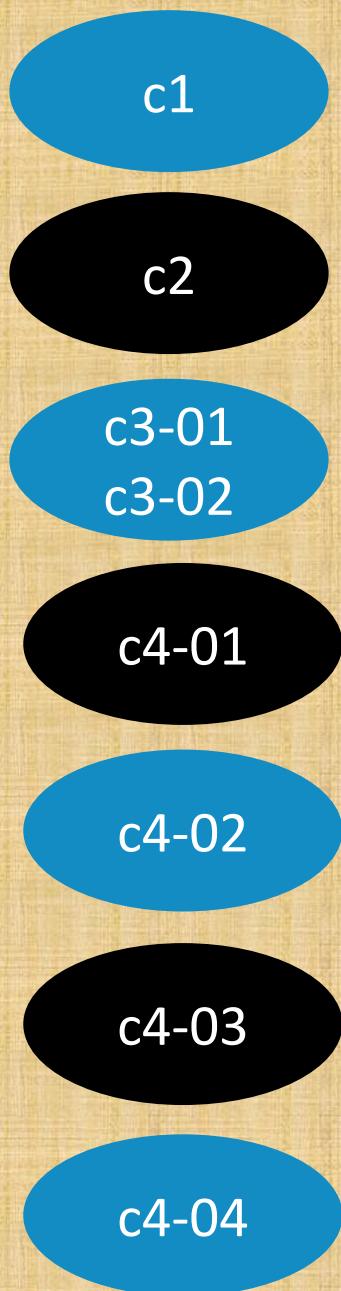
EBS CSI Driver as EKS Add-On

That said we will also **automate this using Terraform now**

Additional Reference: <https://docs.aws.amazon.com/eks/latest/userguide/managing-ebs-csi.html>

# What are we going to learn ?

```
✓ 18-EBS-CSI-Install-using-EKS-AddOn
  > 01-ekscluster-terraform-manifests
  ✓ 02-ebs-addon-terraform-manifests
    ✓ c1-versions.tf
    ✓ c2-remote-state-datasource.tf
    ✓ c3-01-generic-variables.tf
    ✓ c3-02-local-values.tf
    ✓ c4-01-ebs-csi-datasources.tf
    ✓ c4-02-ebs-csi-iam-policy-and-role.tf
    ✓ c4-03-ebs-csi-addon-install.tf
    ✓ c4-04-ebs-csi-outputs.tf
    ✓ terraform.tfvars
  > 03-terraform-manifests-UMS-WebApp
```



Update Terraform Remote Backend information –  
AWS S3 Bucket Key & DynamoDB Table

Define Terraform Remote State Datasource for  
Project-1 EKS Cluster

Define generic variables and local values

Define Terraform HTTP Datasource to download the  
EBS CSI IAM Role from Github

Create IAM Policy and IAM Role with  
AssumeRoleWithWebIdentity

Install EBS CSI Driver using EKS Add-On option

Define EBS CSI Driver Outputs

# What are we going to learn ?

- ✓ 18-EBS-CSl-Install-using-EKS-AddOn
  - > 01-ekscluster-terraform-manifests
  - > 02-ebs-addon-terraform-manifests
  - ✓ 03-terraform-manifests-UMS-WebApp
    - ↳ c1-versions.tf
    - ↳ c2-remote-state-datasource.tf
    - ↳ c3-providers.tf
    - ↳ c4-01-storage-class.tf
    - ↳ c4-02-persistent-volume-claim.tf
    - ↳ c4-03-UserMgmtWebApp-ConfigMap.tf
    - ↳ c4-04-mysql-deployment.tf
    - ↳ c4-05-mysql-clusterip-service.tf
    - ↳ c4-06-UserMgmtWebApp-deployment.tf
    - ↳ c4-07-UserMgmtWebApp-loadbalancer-service.tf
    - ↳ c4-08-UserMgmtWebApp-network-loadbalancer-service.tf
    - ↳ c4-09-UserMgmtWebApp-nodeport-service.tf
    - ↳ webappndb.sql

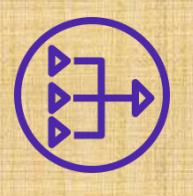
Once the EBS CSI Driver is installed using EKS Add-On option, Deploy our **User Management Sample Application** and test and ensure everything good from EBS Volumes **excluding** the **limitations** with this approach.



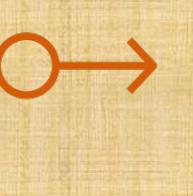
VPC



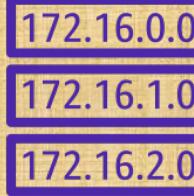
Internet gateway



NAT gateway



Elastic IP address



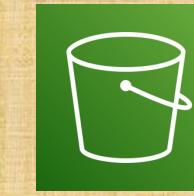
Route table



Public Subnet



Private Subnet



AWS S3



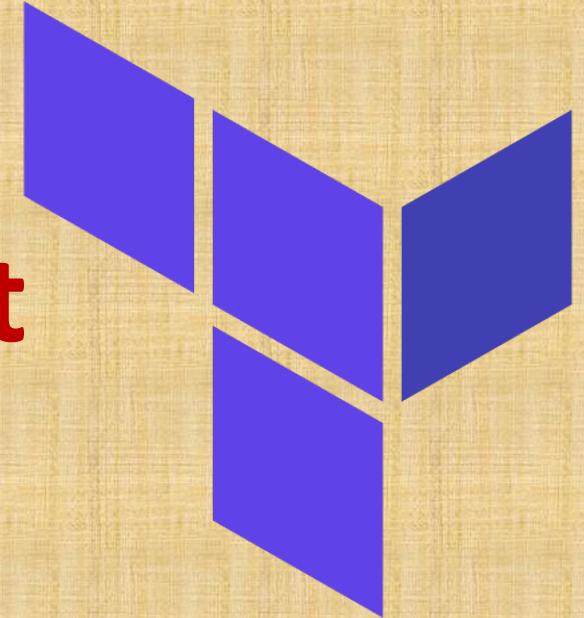
DynamoDB



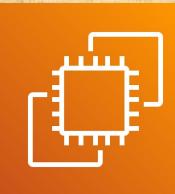
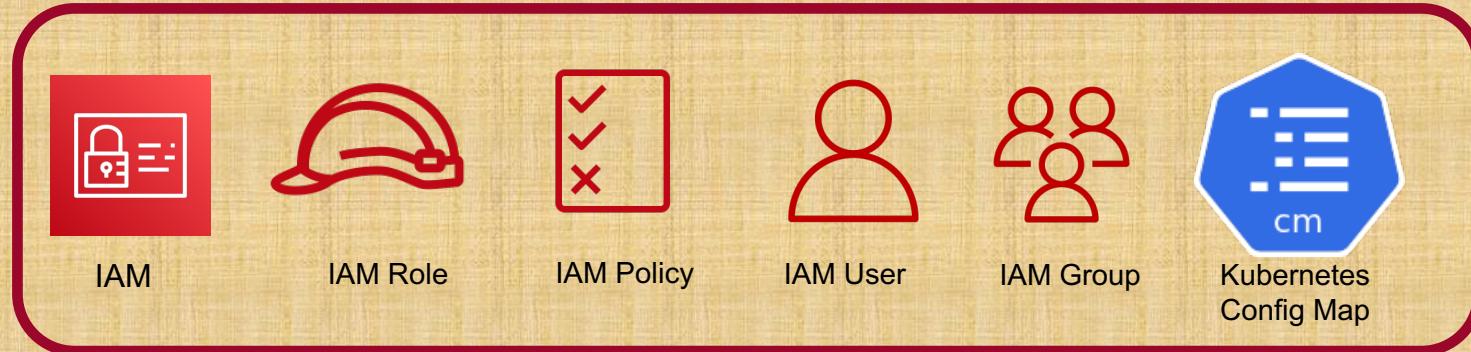
# AWS EKS

## Identity & Access Management

### using AWS IAM



EKS Cluster



EC2 VM



Autoscaling

## Introduction

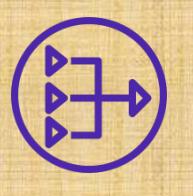
EKS IAM



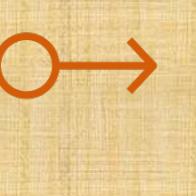
VPC



Internet gateway



NAT gateway



Elastic IP address



Route table



Public Subnet



Private Subnet



AWS S3



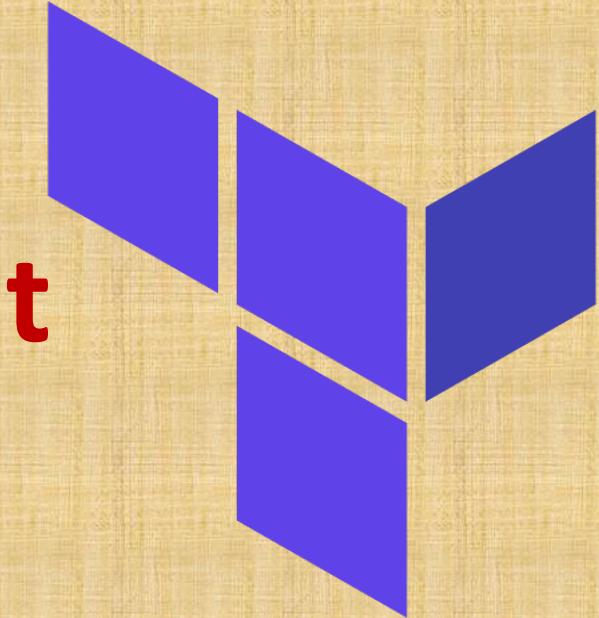
DynamoDB



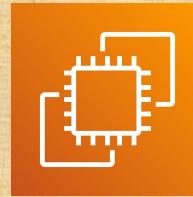
# AWS EKS

## Identity & Access Management

### using AWS IAM



EKS Cluster



EC2 VM

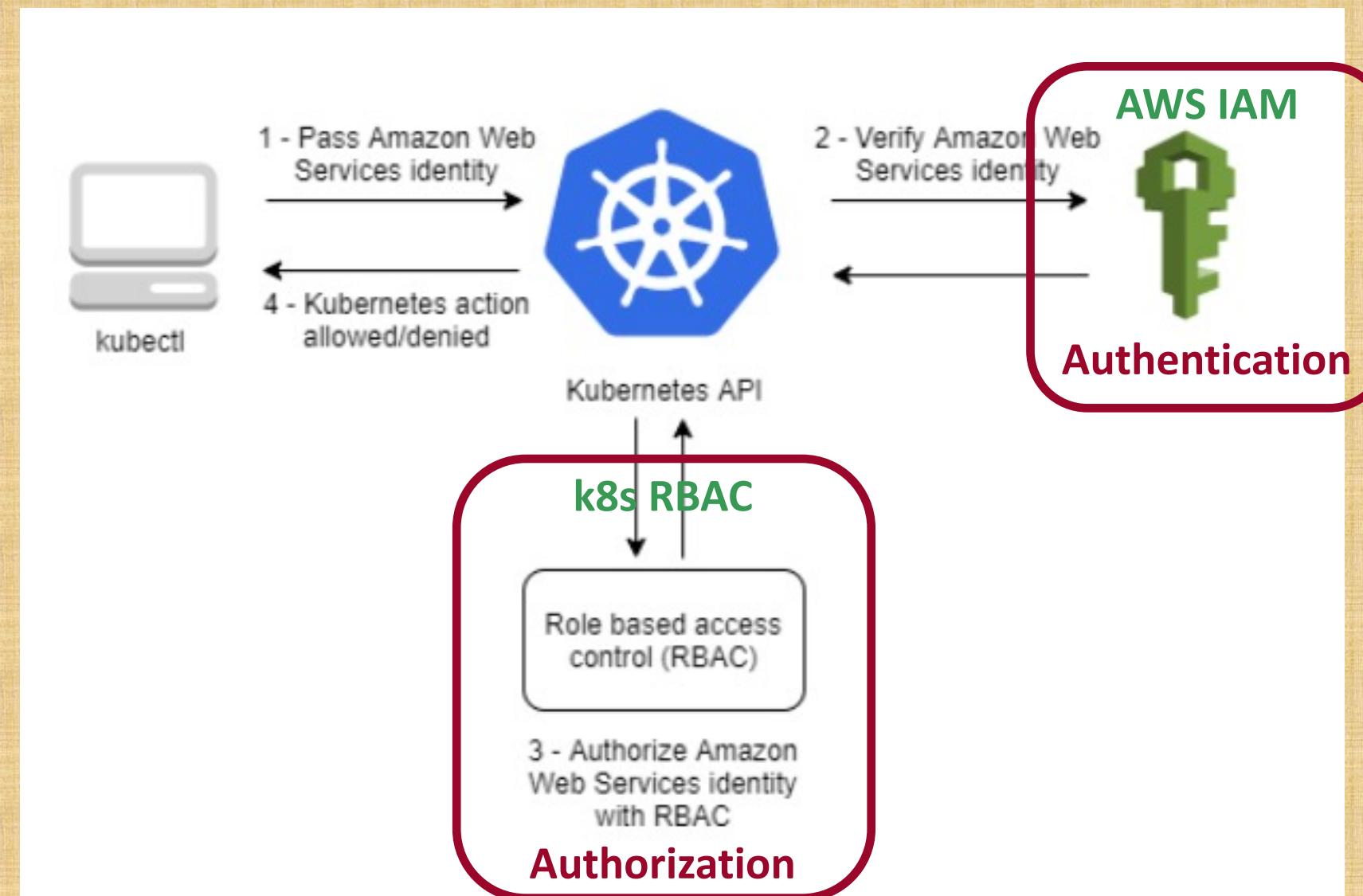


Autoscaling

**AWS EKS Authentication and Authorization**

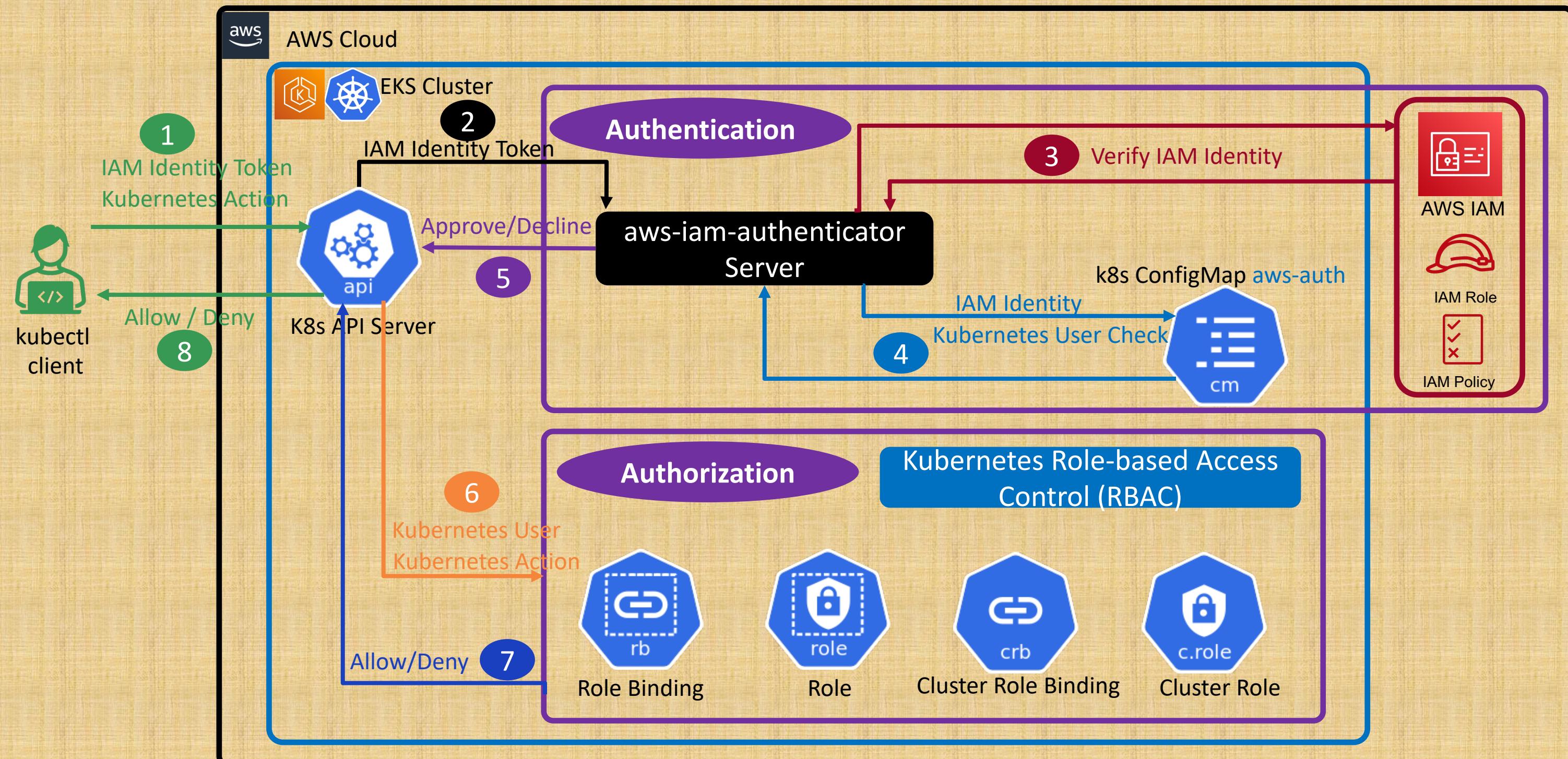
**EKS IAM**

# EKS Authentication & Authorization



Reference: <https://docs.aws.amazon.com/eks/latest/userguide/cluster-auth.html>

# EKS Authentication & Authorization



# EKS Authentication & Authorization

0

**Kubernetes Action:** From kubectl cli, run the command like `kubectl get pods`

1

Kubectl client makes a request to k8s API server passing an **access-token** with the user's information

2

k8s API server passes this token to EKS Control Plane's service: **aws-iam-authenticator**

3

aws-iam-authenticator calls AWS IAM service and passes the **access-token** to check if this is valid user and whether that user has permissions to access the EKS cluster

3-1

**Authentication Check:** AWS IAM makes **internal authentication** check by using access-token

3-2

**Authorization Check:** AWS IAM makes **internal authorization** check by checking IAM policies tied to this user (User with API calls to `eks::*` resources should be allowed)

4

aws-iam-authenticator checks with **aws-auth ConfigMap** whether this user has permissions to access this EKS cluster resources

# EKS Authentication & Authorization

5

aws-iam-authenticator returns **approve/decline** response to the k8s API server

6

k8s API Server will pass a user who already was validated with an **authentication module** to an **authorization module** to check for its permissions (Role Based Access Control – RBAC)

7

Authorization Module will send the **Allow / Deny** response to k8s API Server

8

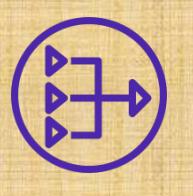
k8s API Server will perform **(Allow)** the Kubernetes action (Example: kubectl get pods) or **Deny** based on response from Authorization Module



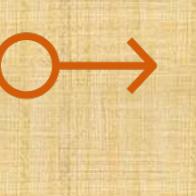
VPC



Internet gateway



NAT gateway



Elastic IP address



Route table



Public Subnet



Private Subnet



AWS S3



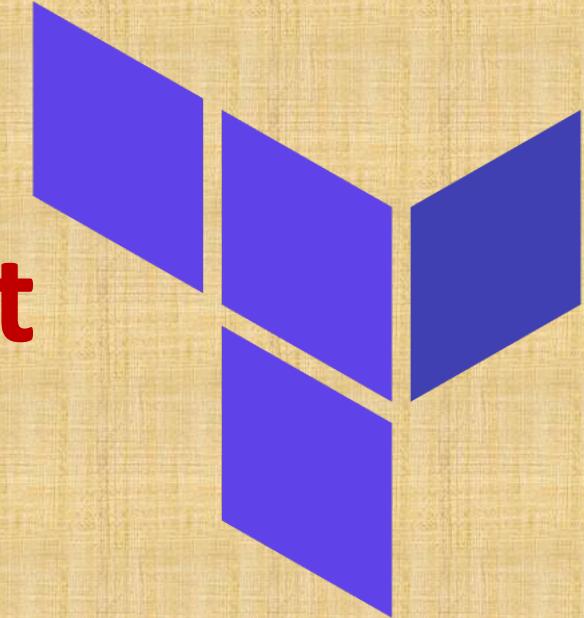
DynamoDB



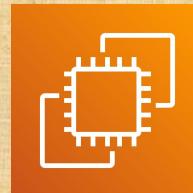
# AWS EKS

## Identity & Access Management

### using AWS IAM



EKS Cluster



EC2 VM



Autoscaling

**AWS EKS IAM UseCases**

**EKS IAM**

# AWS EKS IAM UseCases



Section-19

Provision AWS IAM Admins as EKS Admins

Section-20

Provision AWS IAM Basic User as EKS Admin

Section-21

Automate Section-19,20 with Terraform

Section-22

Provision EKS Admins with AWS IAM Roles & IAM Groups

Section-23

Automate Section-22 with Terraform

Section-24

Provision EKS Read-Only Users with IAM Roles, IAM Groups, Kubernetes Cluster Role and Cluster Role Binding

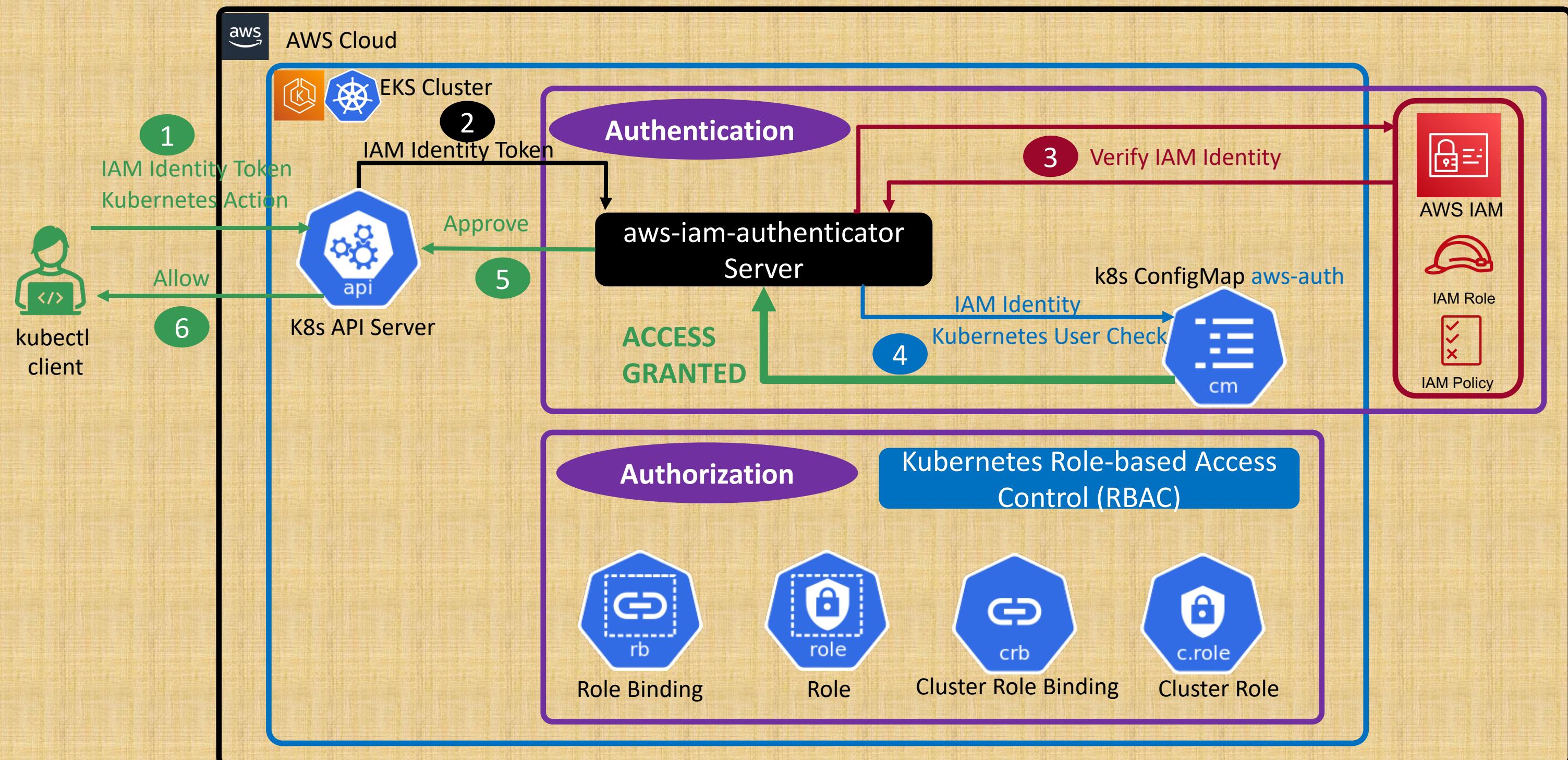
Section-25

A Developer requesting full access to a namespace (dev) in EKS Cluster and in addition, also asking for read-only access to EKS Cluster

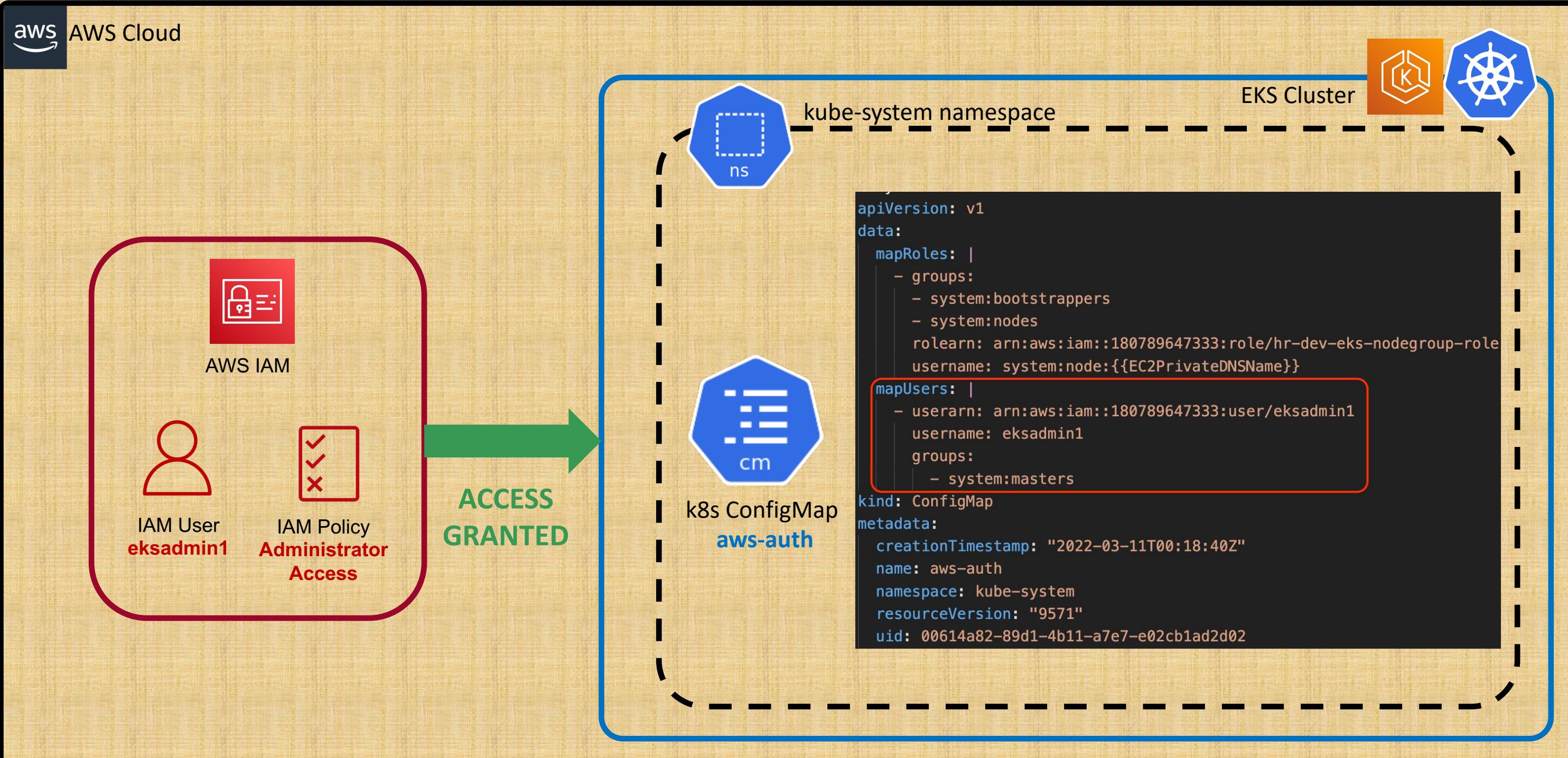
1. Kubernetes Role
2. Kubernetes Role Binding
3. Kubernetes Cluster Role
4. Kubernetes Cluster Role Binding



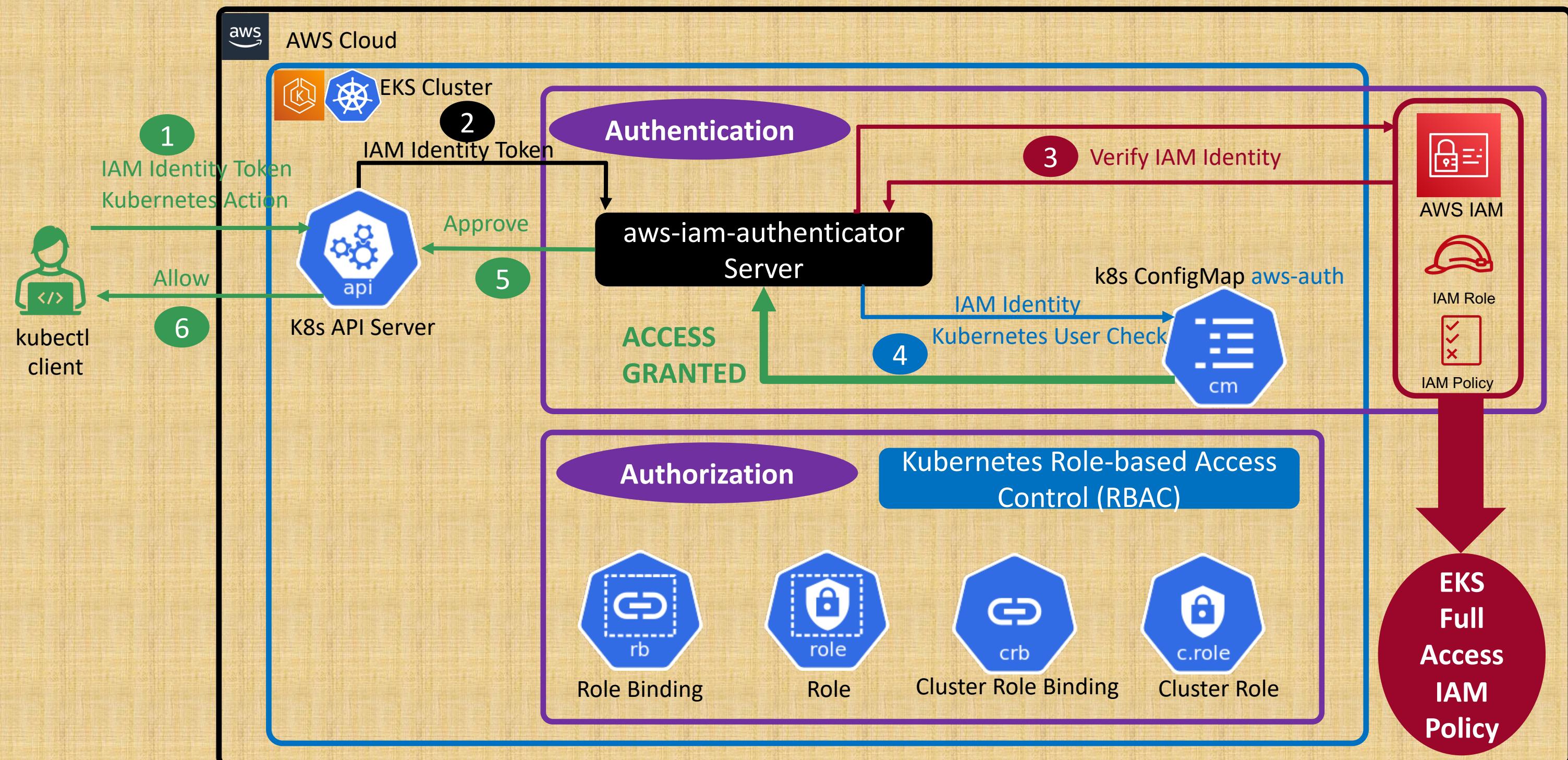
# EKS Authentication & Authorization - Admin User



# Provision AWS Admins as EKS Admins



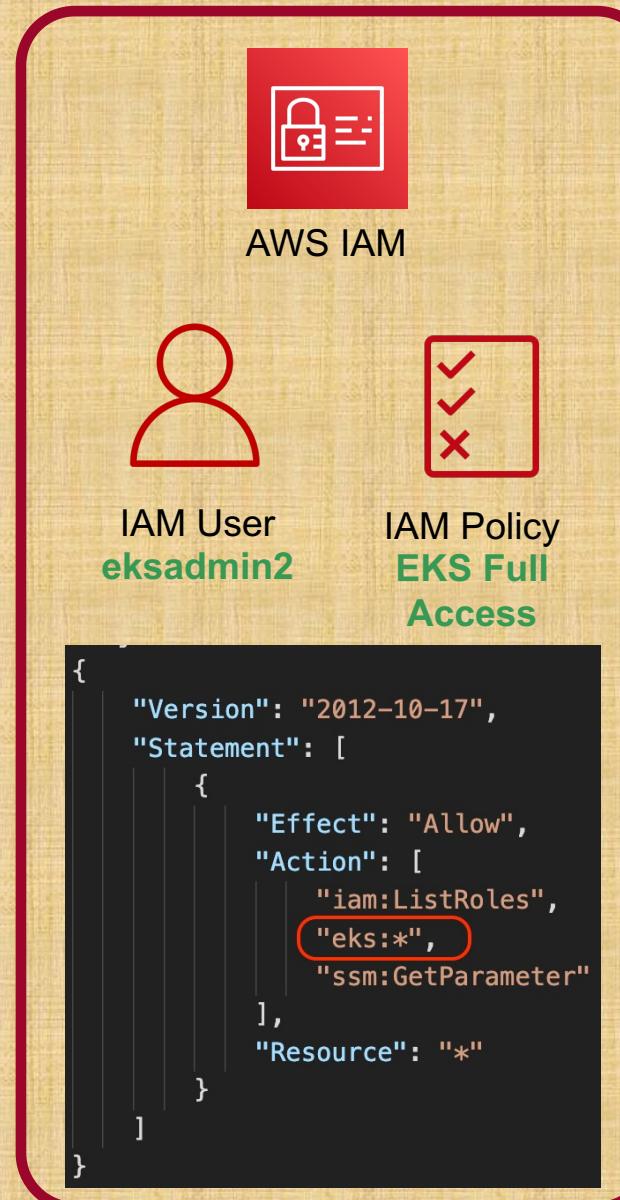
# EKS Authentication & Authorization - Basic User



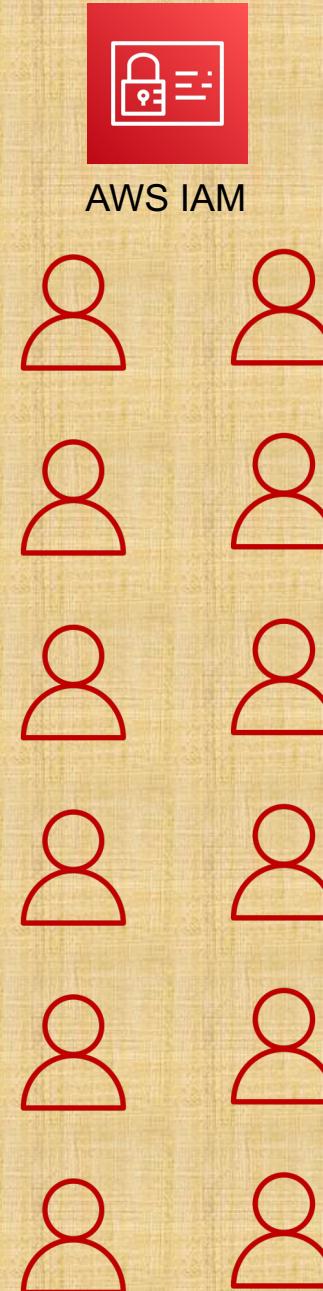
# Provision AWS Basic User as EKS Admin



AWS Cloud



# Usecase - Multiple EKS Admin Users



If we need to add many IAM Users as EKS Admins (20, 30, 40, 50) what should we do ?

**Option-1:** Add those User ARNs in mapUsers section of aws-auth ConfigMap

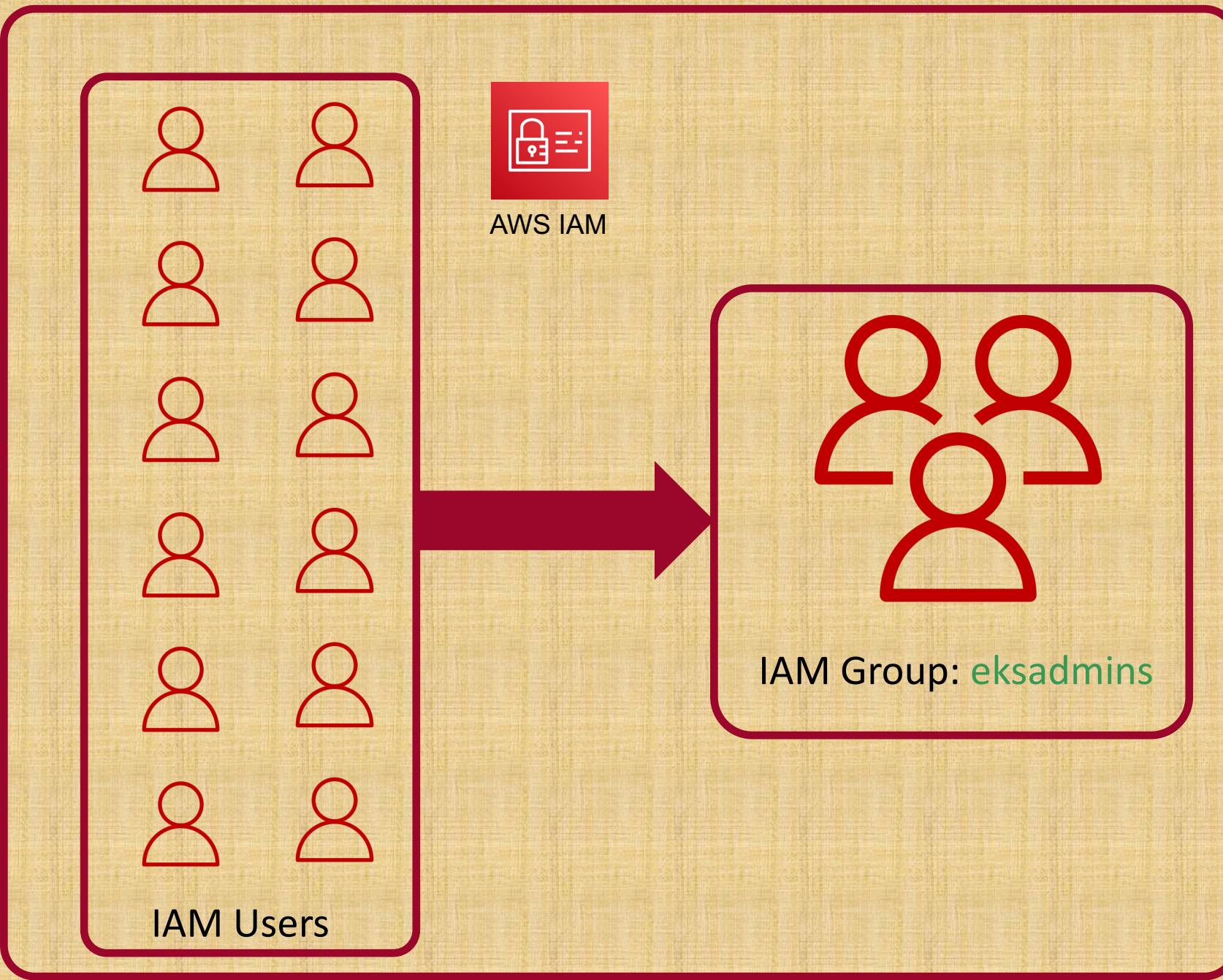
Adding all these users in aws-auth ConfigMap is not a feasible Option from maintenance perspective

Complexity increases

```
apiVersion: v1
data:
  mapRoles: |
    - groups:
      - system:bootstrappers
      - system:nodes
    rolearn: arn:aws:iam::180789647333:role/hr-dev-eks-nodegroup-role
    username: system:node:{{EC2PrivateDNSName}}
  mapUsers: |
    - userarn: arn:aws:iam::180789647333:user/eksadmin1
      username: eksadmin1
      groups:
        - system:masters
kind: ConfigMap
metadata:
  creationTimestamp: "2022-03-11T00:18:40Z"
  name: aws-auth
  namespace: kube-system
  resourceVersion: "9571"
  uid: 00614a82-89d1-4b11-a7e7-e02cb1ad2d02
```

Every time we add a new user, we need to update the Kubernetes aws-auth ConfigMap in EKS Cluster

# Usecase - Multiple EKS Admin Users



Can we use **IAM Groups** to simplify this EKS Admin Access Process ? **YES**

Setting up the whole base for this is **complex**, many things like below are involved

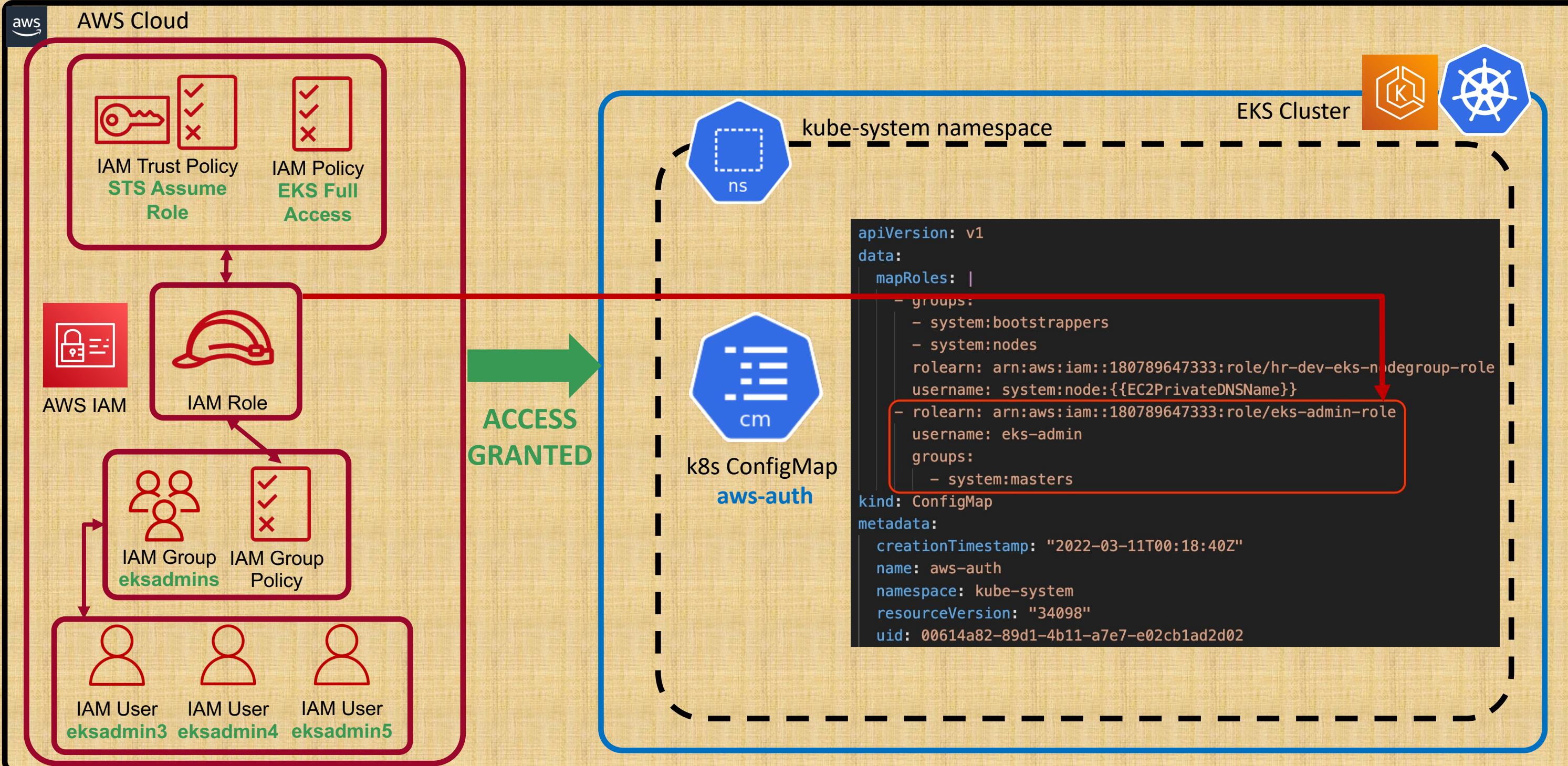
1. STS Assume Role Trust Policy
2. IAM Policy
3. IAM Role
4. IAM Group Policy
5. IAM Groups

After all steps are completed, provisioning EKS Admins becomes **so so easy**

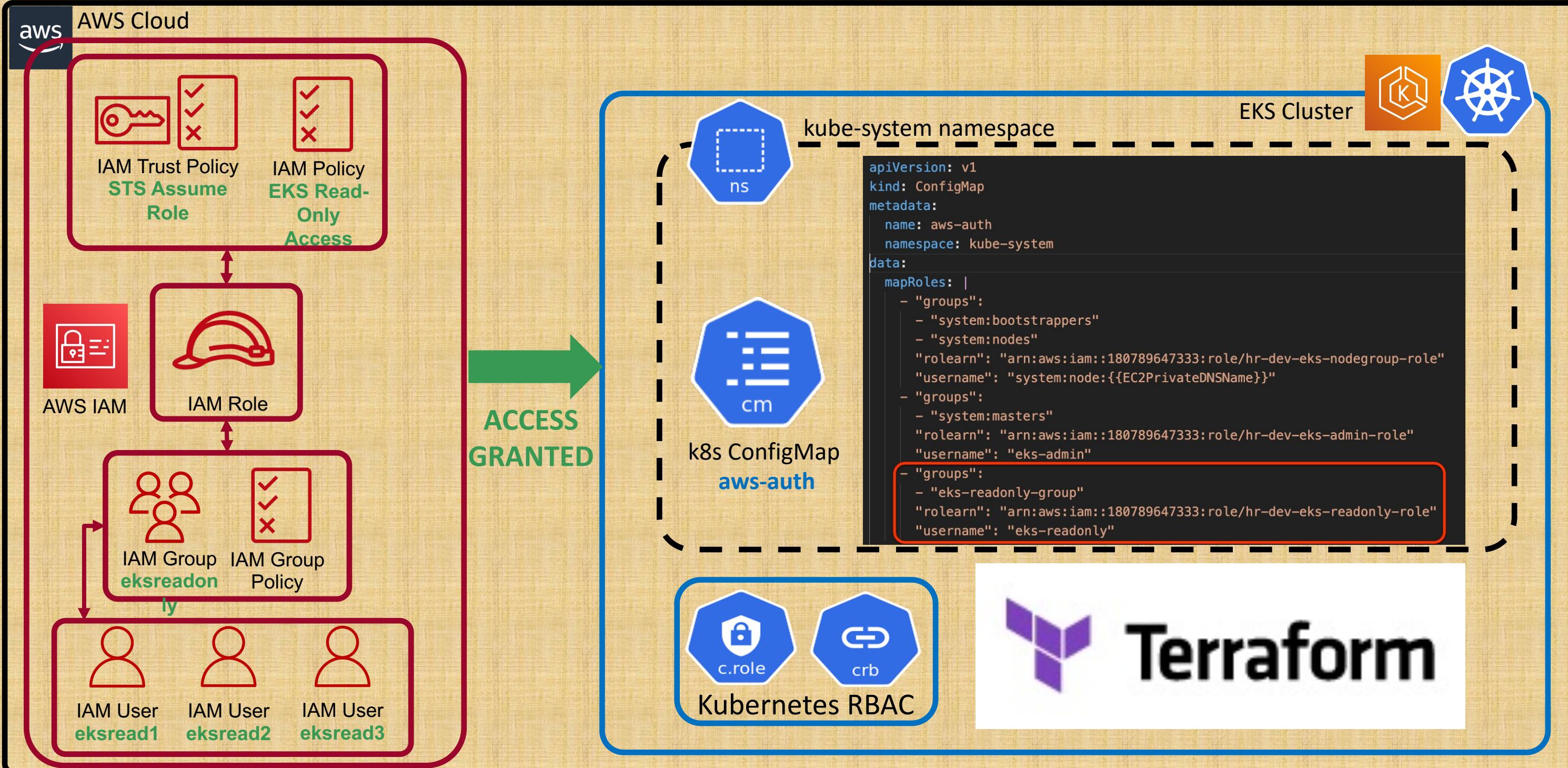
**Step-1:** Create **IAM User**

**Step-2:** Associate that User to **IAM Group: eksadmins**. (User gets access to EKS Cluster)

# Provision EKS Admins using IAM Roles & IAM Groups



# Provision EKS ReadOnly Users using IAM Roles & IAM Groups



```
# Resource: Cluster Role
resource "kubernetes_cluster_role_v1" "eksreadonly_clusterrole" {
  metadata {
    name = "${local.name}-eksreadonly-clusterrole"
  }
  rule {
    api_groups = [""] # These come under core APIs
    resources  = ["nodes", "namespaces", "pods", "events", "services"]
    #resources  = ["nodes", "namespaces", "pods", "events", "services", "config"]
    verbs       = ["get", "list"]
  }
  rule {
    api_groups = ["apps"]
    resources  = ["deployments", "daemonsets", "statefulsets", "replicasets"]
    verbs       = ["get", "list"]
  }
  rule {
    api_groups = ["batch"]
    resources  = ["jobs"]
    verbs       = ["get", "list"]
  }
}
```

# Cluster Role

# ClusterRoleBinding & aws-auth ConfigMap

```
# Resource: Cluster Role Binding
resource "kubernetes_cluster_role_binding_v1" "eksreadonly"
  metadata {
    name = "${local.name}-eksreadonly-clusterrolebinding"
  }
  role_ref {
    api_group = "rbac.authorization.k8s.io"
    kind      = "ClusterRole"
    name      = kubernetes_cluster_role_v1.eksreadonly_clu
  }
  subject {
    kind      = "Group"
    name      = "eks-readonly-group"
    api_group = "rbac.authorization.k8s.io"
  }
}
```

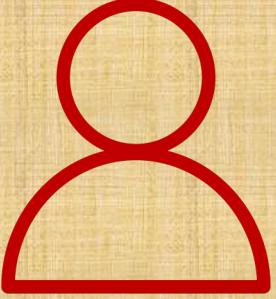
```
apiVersion: v1
kind: ConfigMap
metadata:
  name: aws-auth
  namespace: kube-system
data:
  mapRoles: |
    - "groups":
      - "system:bootstrappers"
      - "system:nodes"
      "rolearn": "arn:aws:iam::180789647333:role/hr-dev-eks-nodegroup-role"
      "username": "system:node:{EC2PrivateDNSName}"
    - "groups":
      - "system:masters"
      "rolearn": "arn:aws:iam::180789647333:role/hr-dev-eks-admin-role"
      "username": "eks-admin"
    - "groups":
      - "eks-readonly-group"
      "rolearn": "arn:aws:iam::180789647333:role/hr-dev-eks-readonly-role"
      "username": "eks-readonly"
```

How does the **aws-auth ConfigMap** enforce the **Kubernetes RBAC permissions** defined in **ClusterRole & ClusterRoleBinding**?

Subject Name in **ClusterRoleBinding** should match the **groups value** in **aws-auth ConfigMap**

# Usecase

A Developer requesting full access to a namespace (**dev**) in EKS Cluster and in addition, also asking for **read-only** access to EKS Cluster

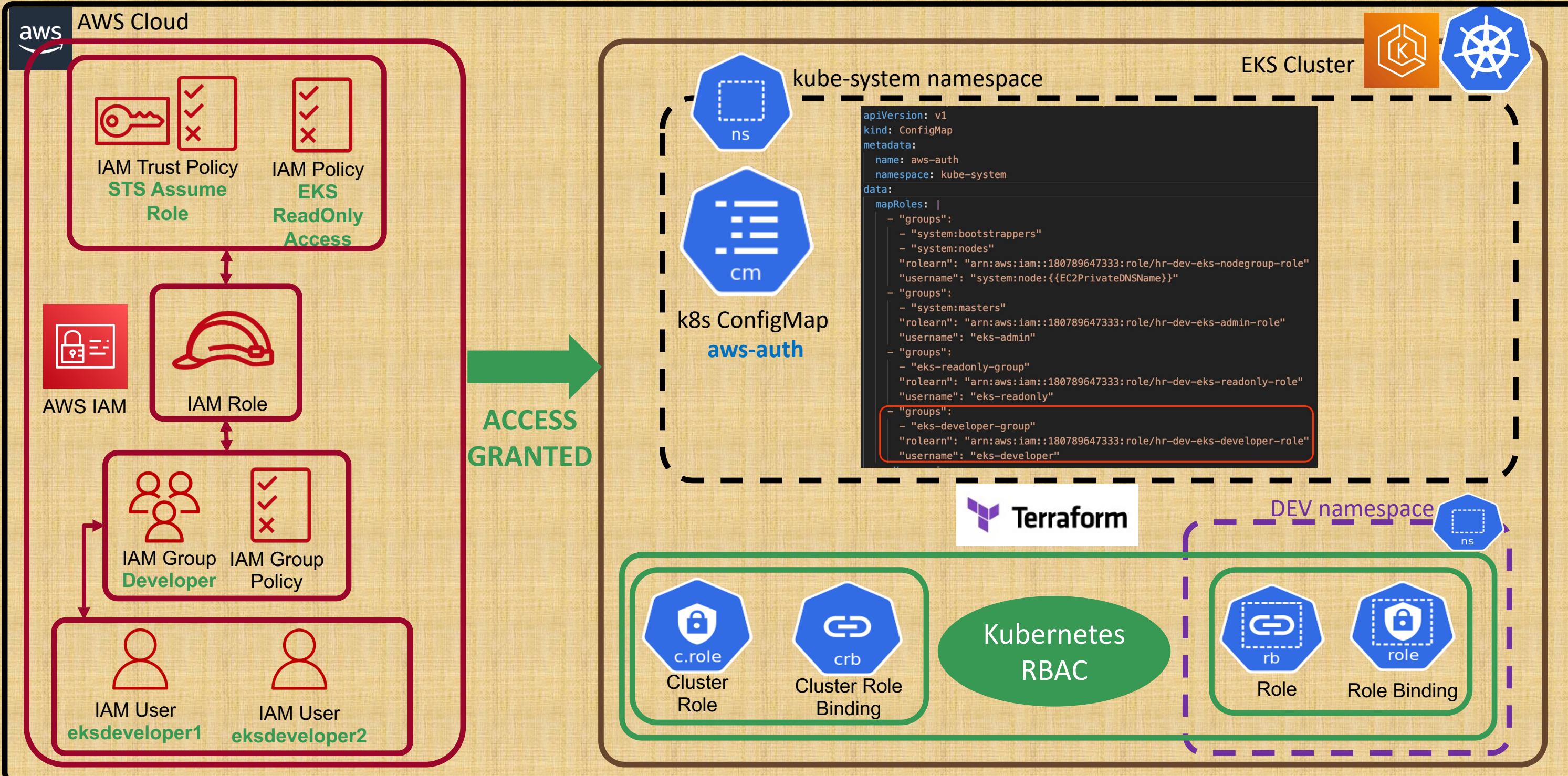
  
**EKS Developer**  
**IAM User**

EKS Cluster Read-Only Access

EKS Dev Namespace Full Access  
(All Verbs – Create , Apply, Get, List, Delete Resources)



# Provision EKS Developer Users using IAM Roles & IAM Groups



# aws-auth ConfigMap

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: aws-auth
  namespace: kube-system
data:
  mapRoles: |
    - "groups":
      - "system:bootstrappers"
      - "system:nodes"
      "rolearn": "arn:aws:iam::180789647333:role/hr-dev-eks-nodegroup-role"
      "username": "system:node:{EC2PrivateDNSName}"
    - "groups":
      - "system:masters"
      "rolearn": "arn:aws:iam::180789647333:role/hr-dev-eks-admin-role"
      "username": "eks-admin"
    - "groups":
      - "eks-readonly-group"
      "rolearn": "arn:aws:iam::180789647333:role/hr-dev-eks-readonly-role"
      "username": "eks-readonly"
    - "groups":
      - "eks-developer-group"
      "rolearn": "arn:aws:iam::180789647333:role/hr-dev-eks-developer-role"
      "username": "eks-developer"
  mapUsers: |
    - "groups":
      - "system:masters"
      "userarn": "arn:aws:iam::180789647333:user/hr-dev-eksadmin2"
      "username": "hr-dev-eksadmin2"
    - "groups":
      - "system:masters"
      "userarn": "arn:aws:iam::180789647333:user/hr-dev-eksadmin1"
      "username": "hr-dev-eksadmin1"
```

Section-  
22, 23

EKS Admins with IAM Users,  
Groups and Roles

Section-24

EKS ReadOnly Access with IAM Users,  
Groups, Roles & k8s ClusterRole &  
ClusterRoleBinding

Section-25

EKS Developer Access with IAM Users,  
Groups, Roles & k8s ClusterRole,  
ClusterRoleBinding & Role, RoleBinding

Section-  
19, 20, 21

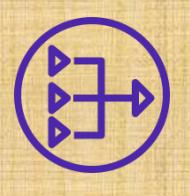
IAM Users as EKS Admins



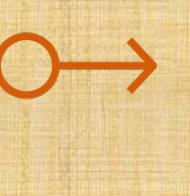
VPC



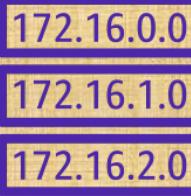
Internet gateway



NAT gateway



Elastic IP address



Route table



Public Subnet



Private Subnet



AWS S3



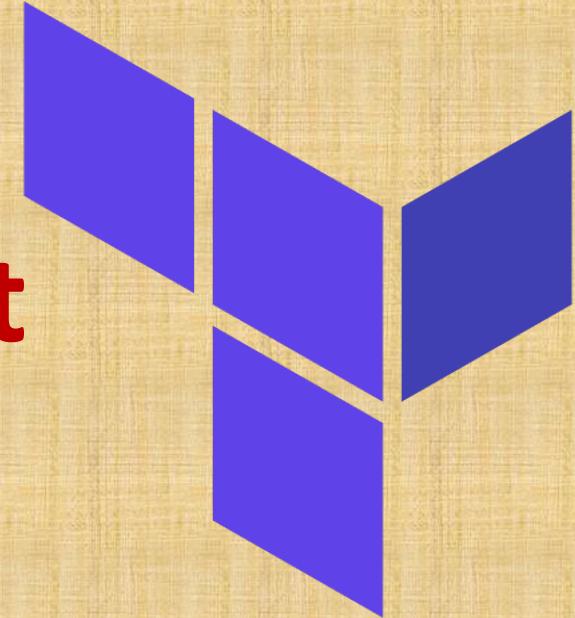
DynamoDB



# AWS EKS

## Identity & Access Management

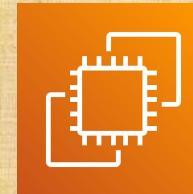
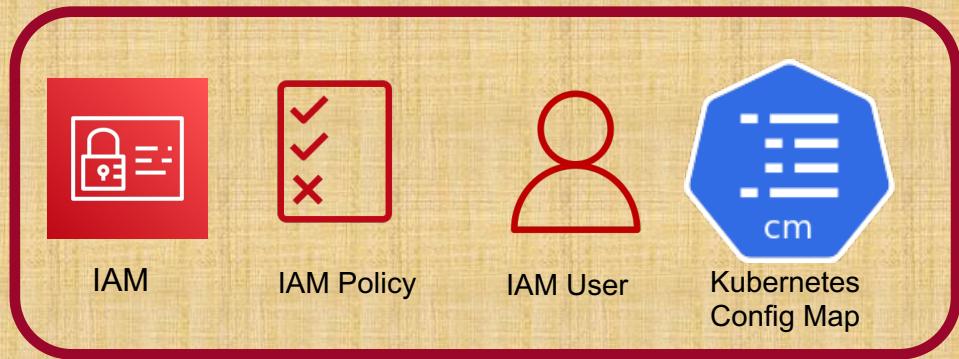
### using AWS IAM



EKS Cluster



Autoscaling



EC2 VM

**EKS IAM**

User: kalyandev



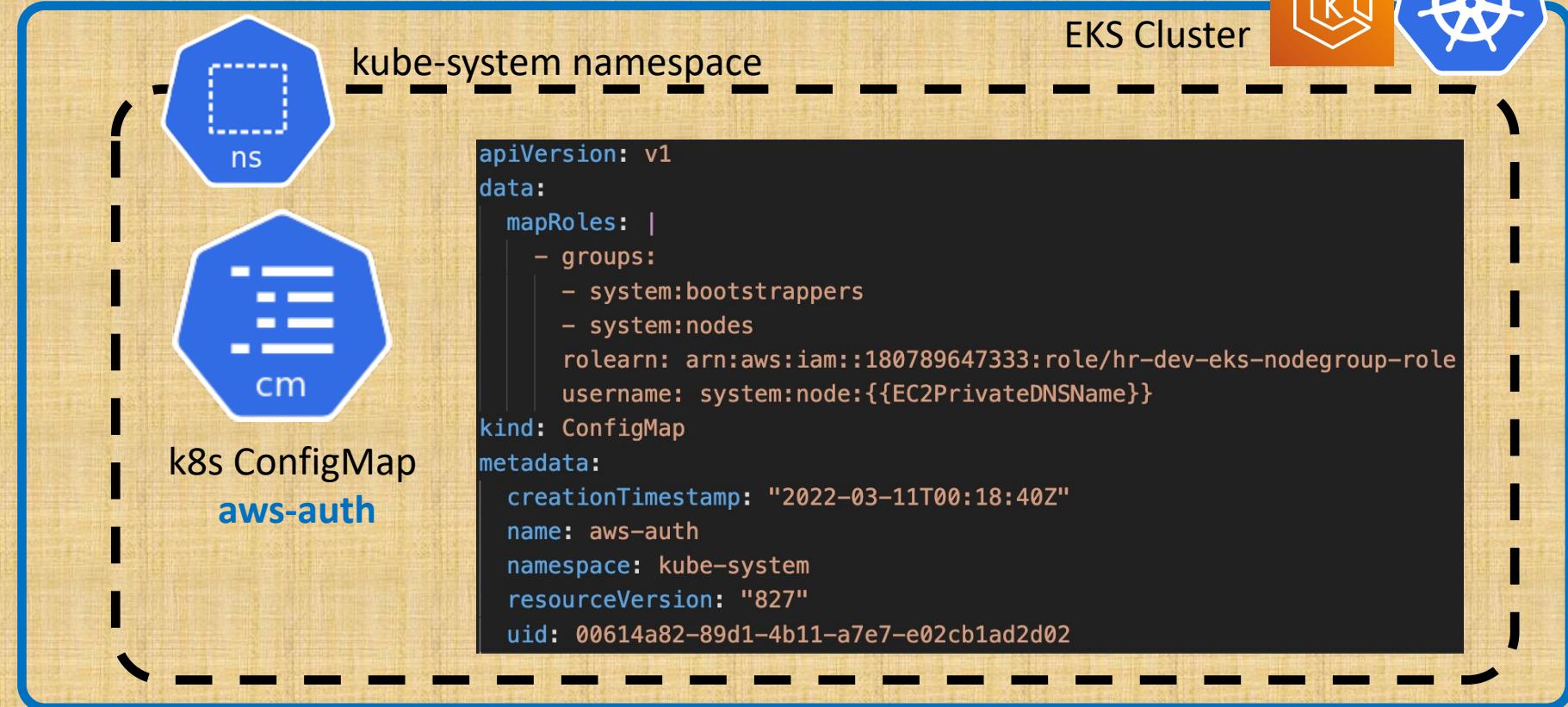
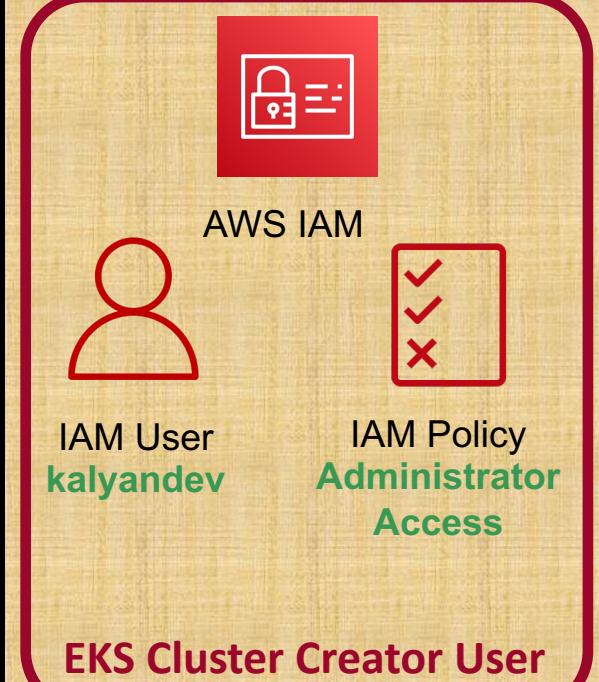
AWS CLI

aws configure  
with Security  
Credentials

Terraform to  
create EKS cluster

kalyandev  
becomes the  
EKS Cluster  
Creator User

AWS Cloud



The user with which we create the EKS Cluster is called **Cluster\_Creator\_User**

This user information is **not stored** in AWS EKS Cluster aws-auth configmap

If we face any issues with `k8s aws-auth configmap` and if we **lost access** to EKS Cluster we need the `cluster\_creator` user to restore the stuff

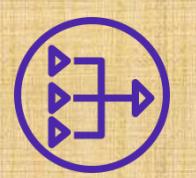
**ALWAYS REMEMBER  
WITH WHICH USER  
EKS CLUSTER IS  
CREATED**



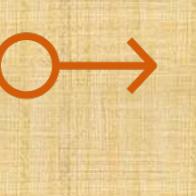
VPC



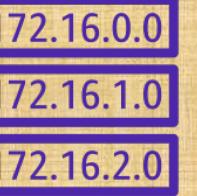
Internet gateway



NAT gateway



Elastic IP address



Route table



Public Subnet



Private Subnet



AWS S3



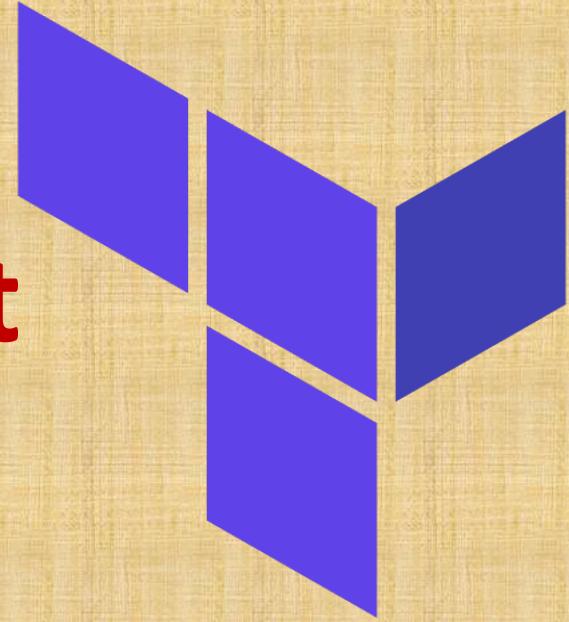
DynamoDB



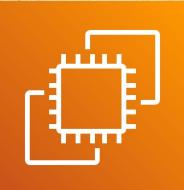
# AWS EKS

## Identity & Access Management

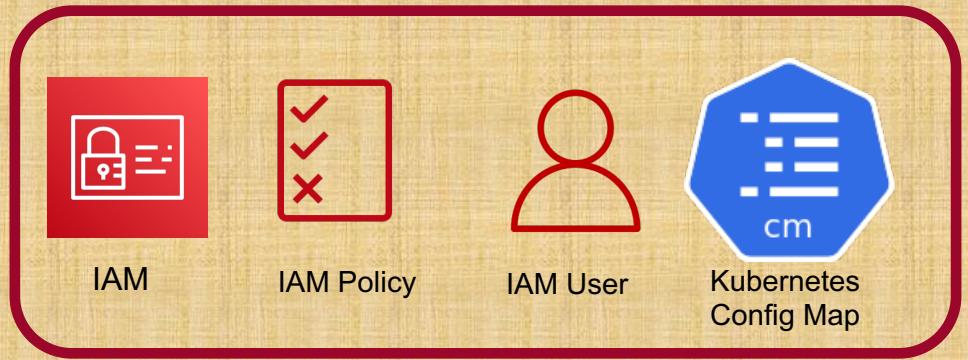
### using AWS IAM



EKS Cluster



EC2 VM



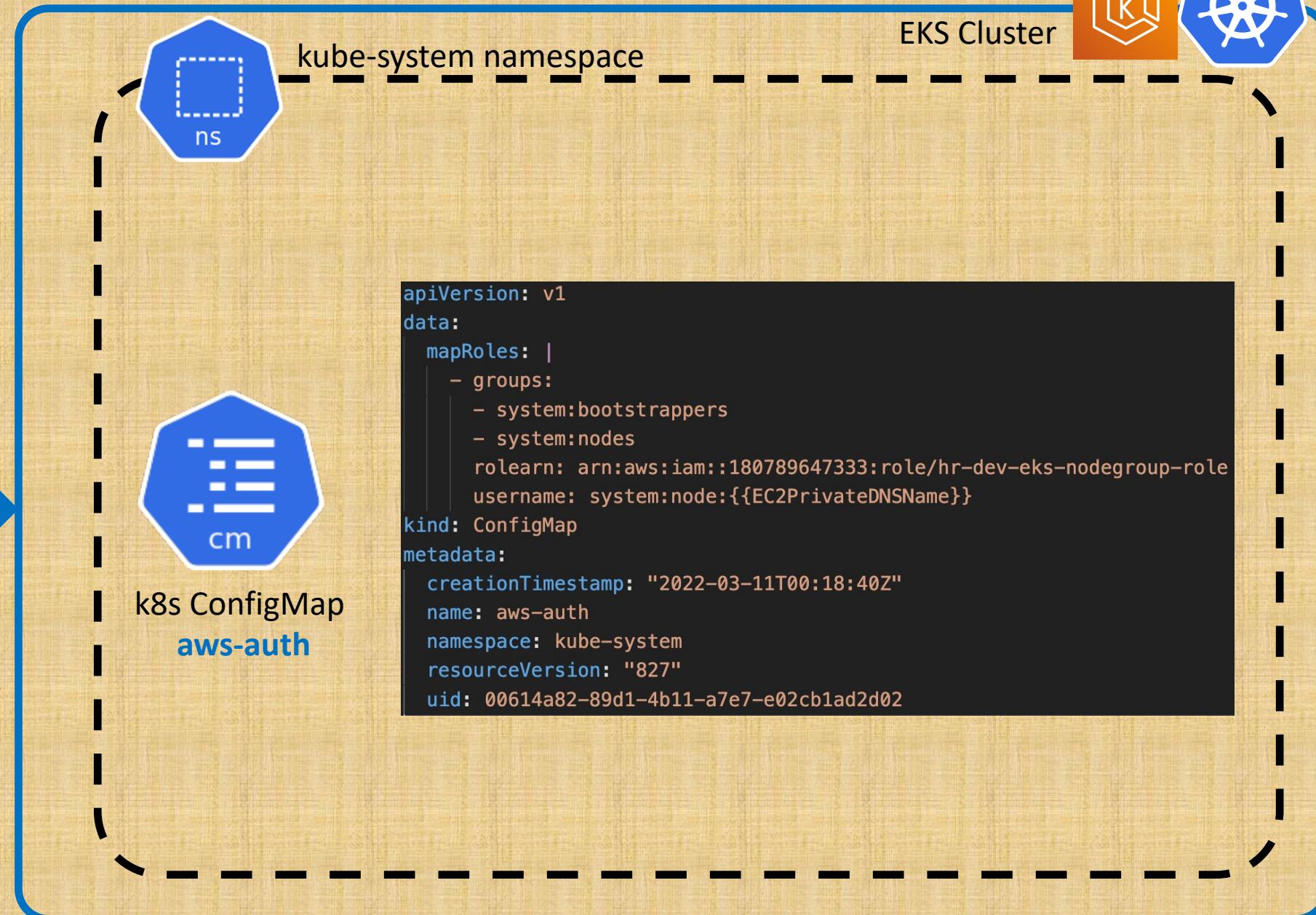
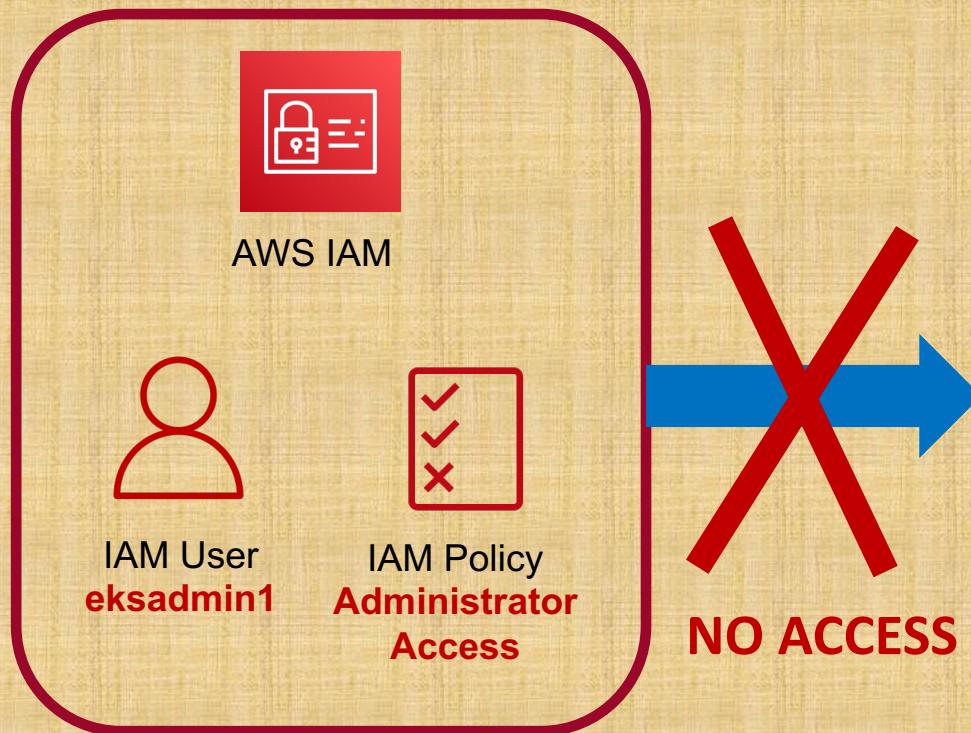
Autoscaling

Provision AWS Admins as EKS Admins

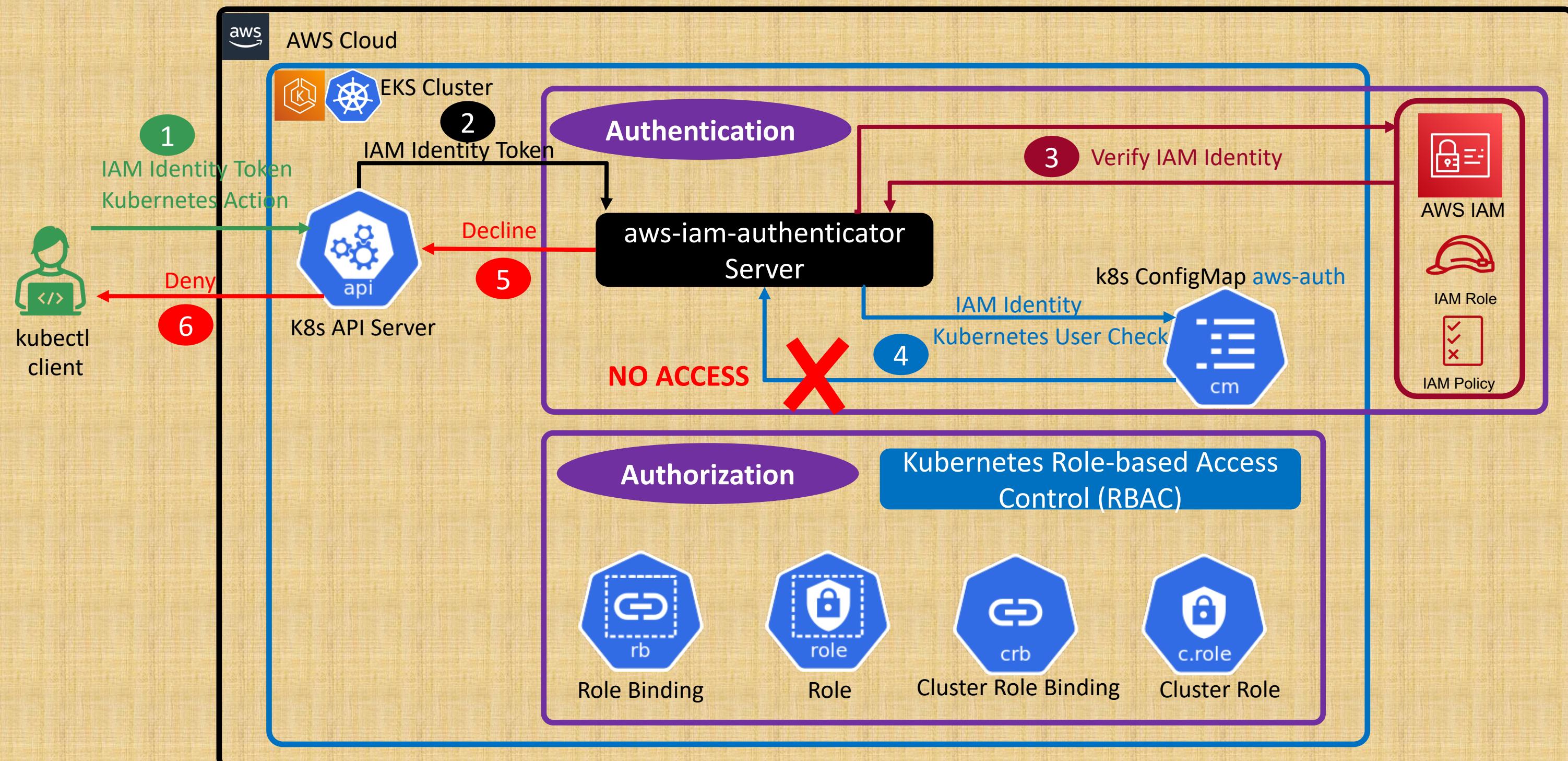
# Provision AWS Admins as EKS Admins



AWS Cloud



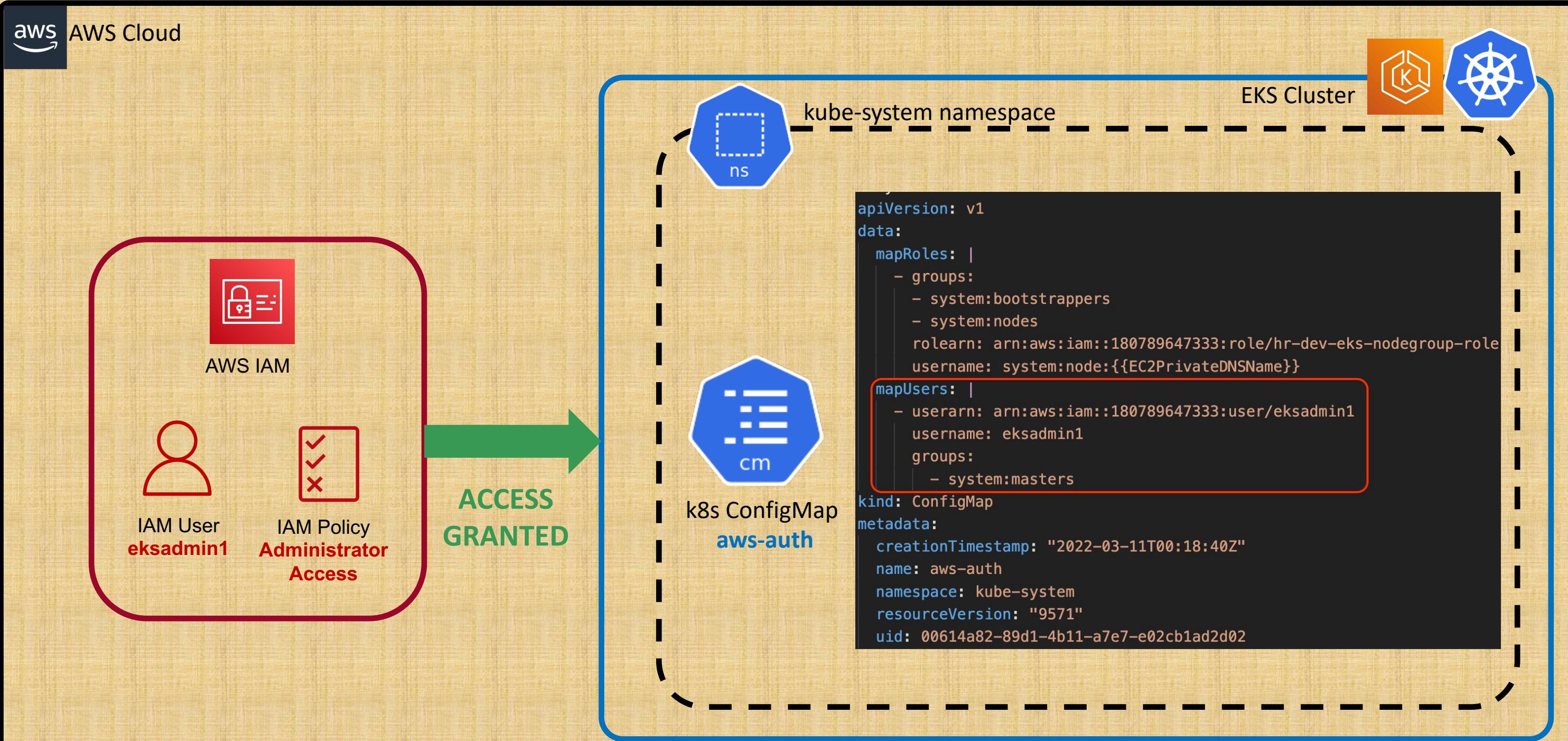
# EKS Authentication & Authorization - Admin User



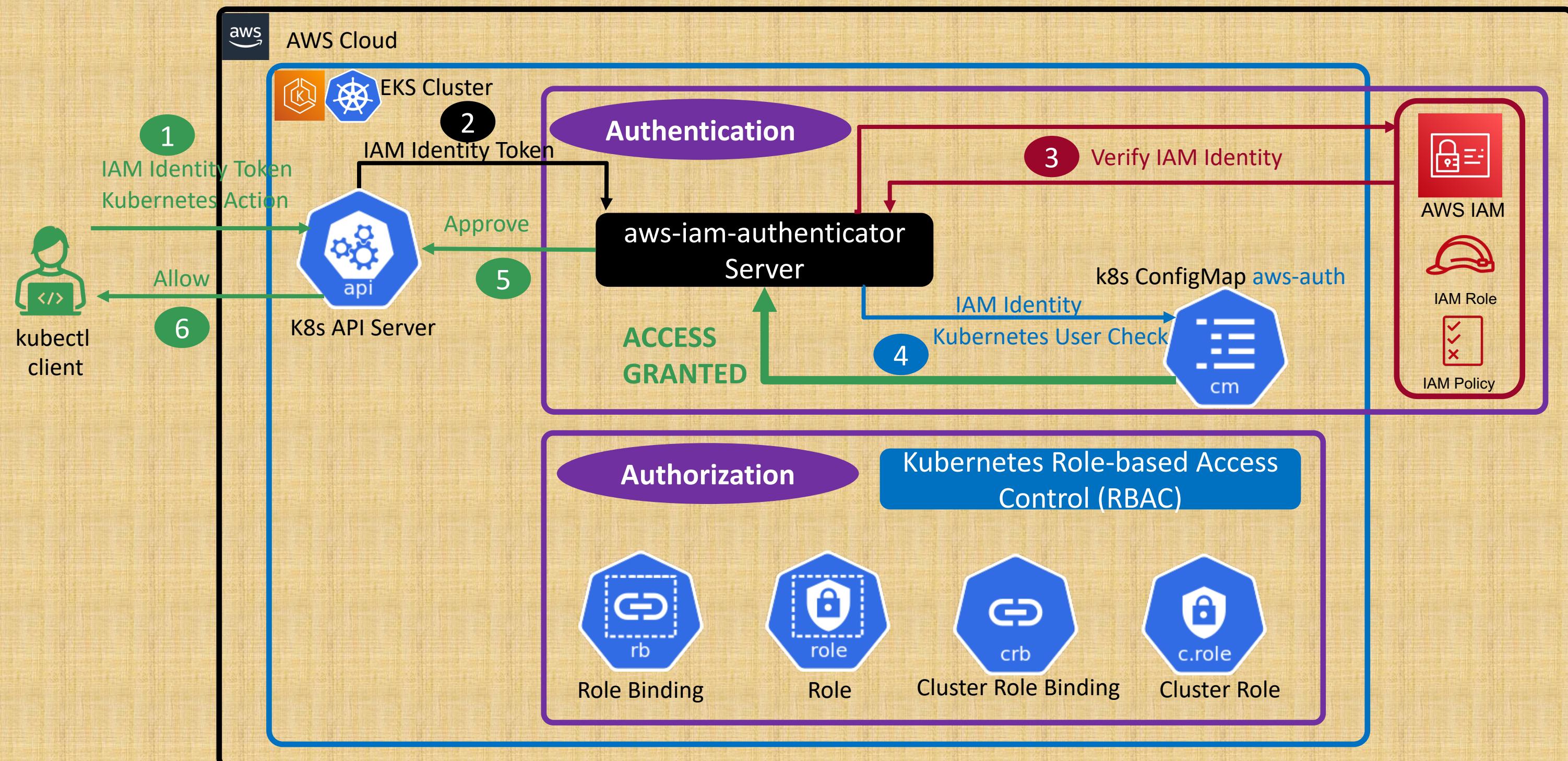
# AWS EKS - Default ConfigMap

```
apiVersion: v1
data:
  mapRoles: | 
    - groups:
      - system:bootstrappers
      - system:nodes
    rolearn: arn:aws:iam::180789647333:role/hr-dev-eks-nodegroup-role
    username: system:node:{{EC2PrivateDNSName}}
kind: ConfigMap
metadata:
  creationTimestamp: "2022-03-11T00:18:40Z"
  name: aws-auth
  namespace: kube-system
  resourceVersion: "827"
  uid: 00614a82-89d1-4b11-a7e7-e02cb1ad2d02
```

# Provision AWS Admins as EKS Admins



# EKS Authentication & Authorization - Admin User



# AWS EKS - ConfigMap with mapUsers

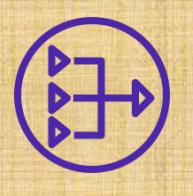
```
apiVersion: v1
data:
  mapRoles: |
    - groups:
        - system:bootstrappers
        - system:nodes
      rolearn: arn:aws:iam::180789647333:role/hr-dev-eks-nodegroup-role
      username: system:node:{{EC2PrivateDNSName}}
  mapUsers: |
    - userarn: arn:aws:iam::180789647333:user/eksadmin1
      username: eksadmin1
      groups:
        - system:masters
kind: ConfigMap
metadata:
  creationTimestamp: "2022-03-11T00:18:40Z"
  name: aws-auth
  namespace: kube-system
  resourceVersion: "9571"
  uid: 00614a82-89d1-4b11-a7e7-e02cb1ad2d02
```



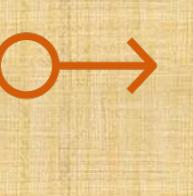
VPC



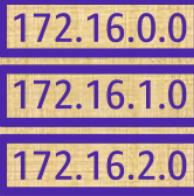
Internet gateway



NAT gateway



Elastic IP address



Route table



Public Subnet



Private Subnet



AWS S3



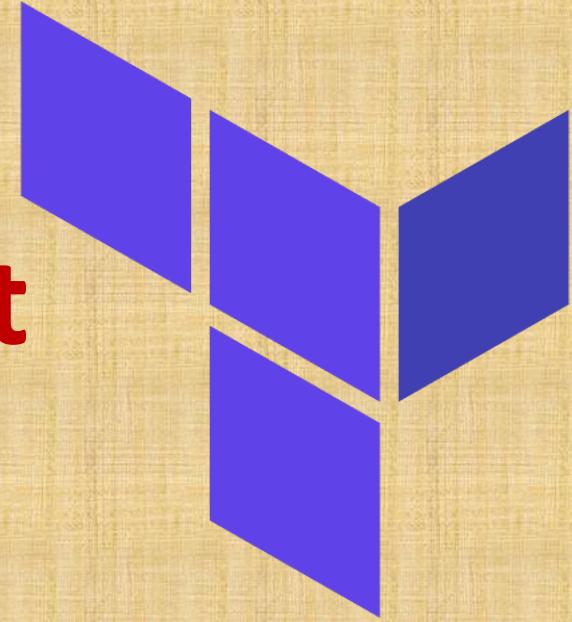
DynamoDB



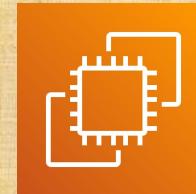
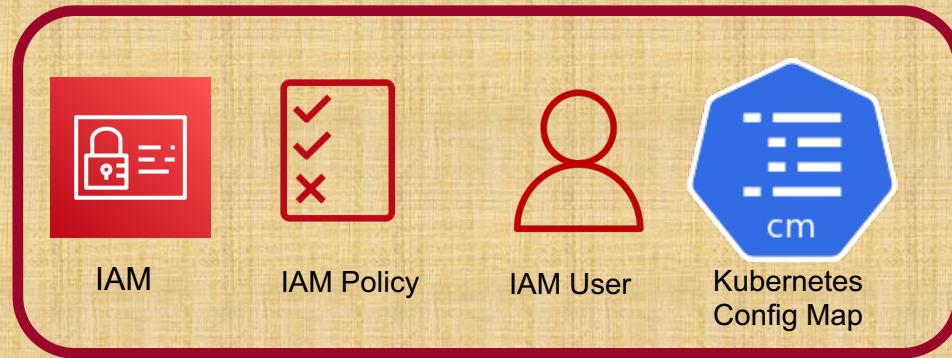
# AWS EKS

## Identity & Access Management

### using AWS IAM



EKS Cluster



EC2 VM



Autoscaling

Provision AWS Basic User as EKS Admin using  
IAM EKS Full Access Policy

# Provision AWS Basic User as EKS Admin

Usecase

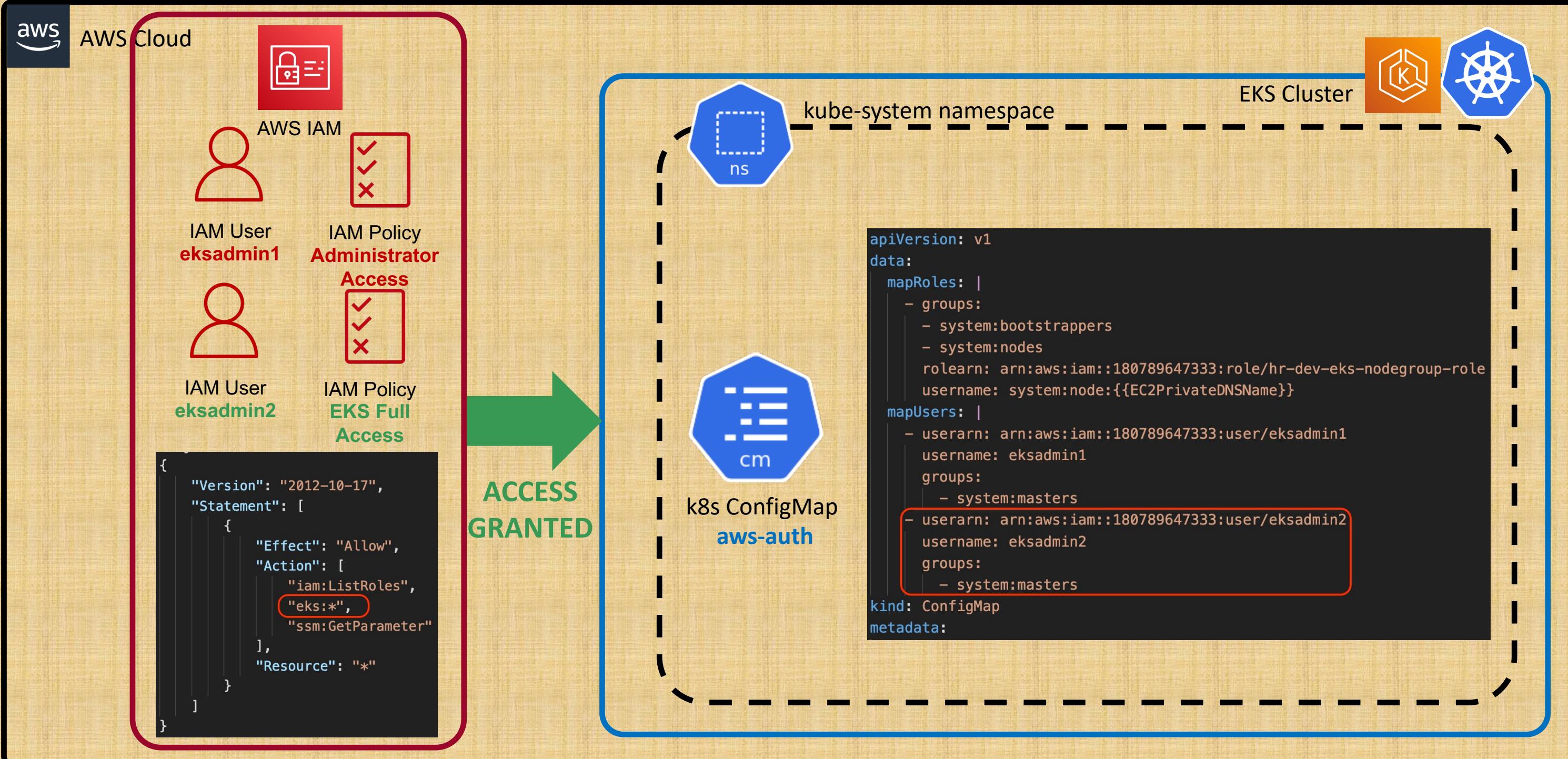
You want to give  
**FULL ACCESS** to AWS  
EKS Cluster but for  
other AWS Services  
**NO ACCESS**

**Step-1:** Create IAM **Basic User**

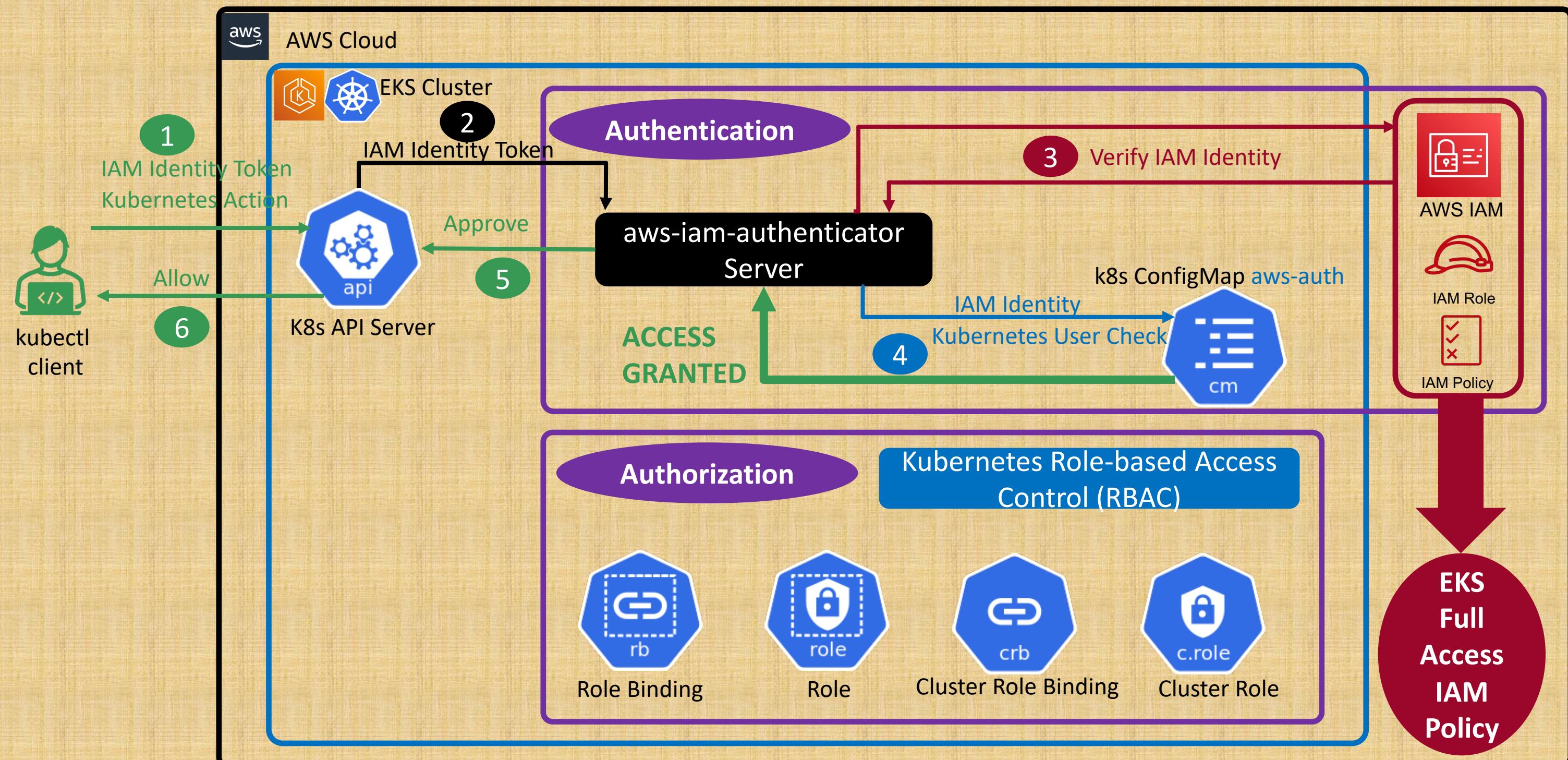
**Step-2:** Create IAM Policy with **EKS Full Access** and associate to IAM User

**Step-3:** Update user information in EKS Cluster **aws-auth ConfigMap**

# Provision AWS Basic User as EKS Admin



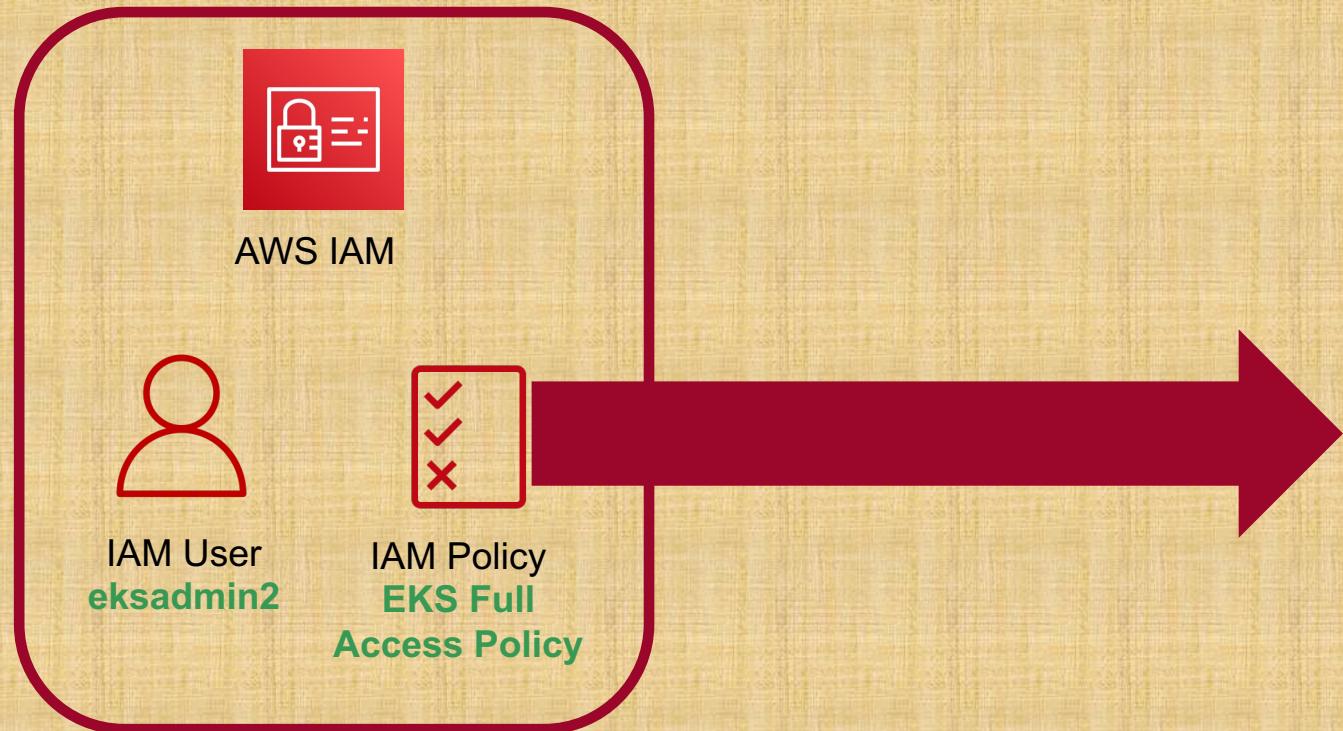
# EKS Authentication & Authorization - Basic User



# AWS EKS - ConfigMap with mapUsers

```
apiVersion: v1
data:
  mapRoles: |
    - groups:
      - system:bootstrappers
      - system:nodes
      rolearn: arn:aws:iam::180789647333:role/hr-dev-eks-nodegroup-role
      username: system:node:{{EC2PrivateDNSName}}
  mapUsers: |
    - userarn: arn:aws:iam::180789647333:user/eksadmin1
      username: eksadmin1
      groups:
        - system:masters
    - userarn: arn:aws:iam::180789647333:user/eksadmin2
      username: eksadmin2
      groups:
        - system:masters
kind: ConfigMap
metadata:
```

# AWS IAM Policy - EKS Full Access



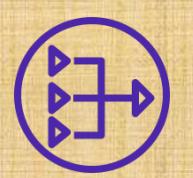
```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": [  
        "iam>ListRoles",  
        "eks:*",  
        "ssm:GetParameter"  
      ],  
      "Resource": "*"  
    }  
  ]  
}
```



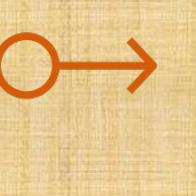
VPC



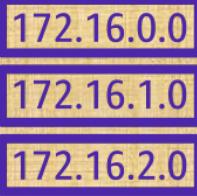
Internet gateway



NAT gateway



Elastic IP address



Route table



Public Subnet



Private Subnet



AWS S3



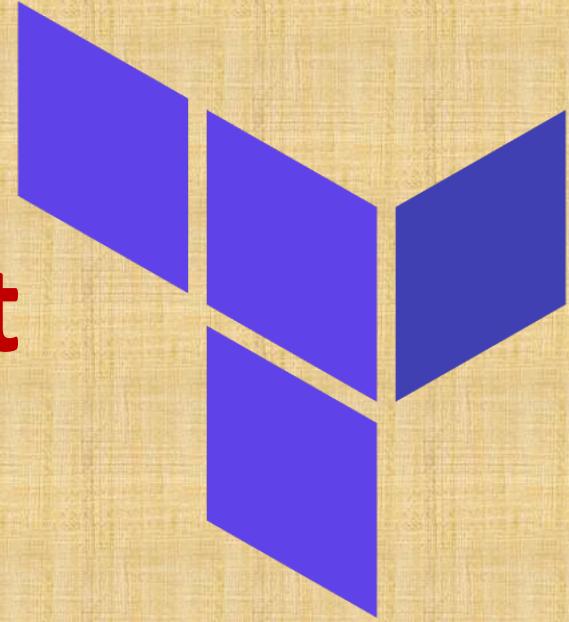
DynamoDB



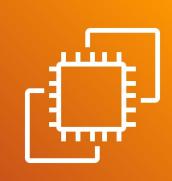
# AWS EKS

## Identity & Access Management

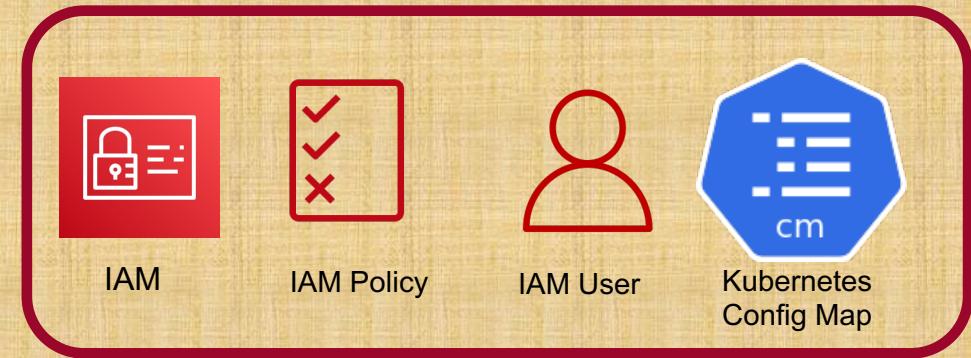
### using AWS IAM



EKS Cluster



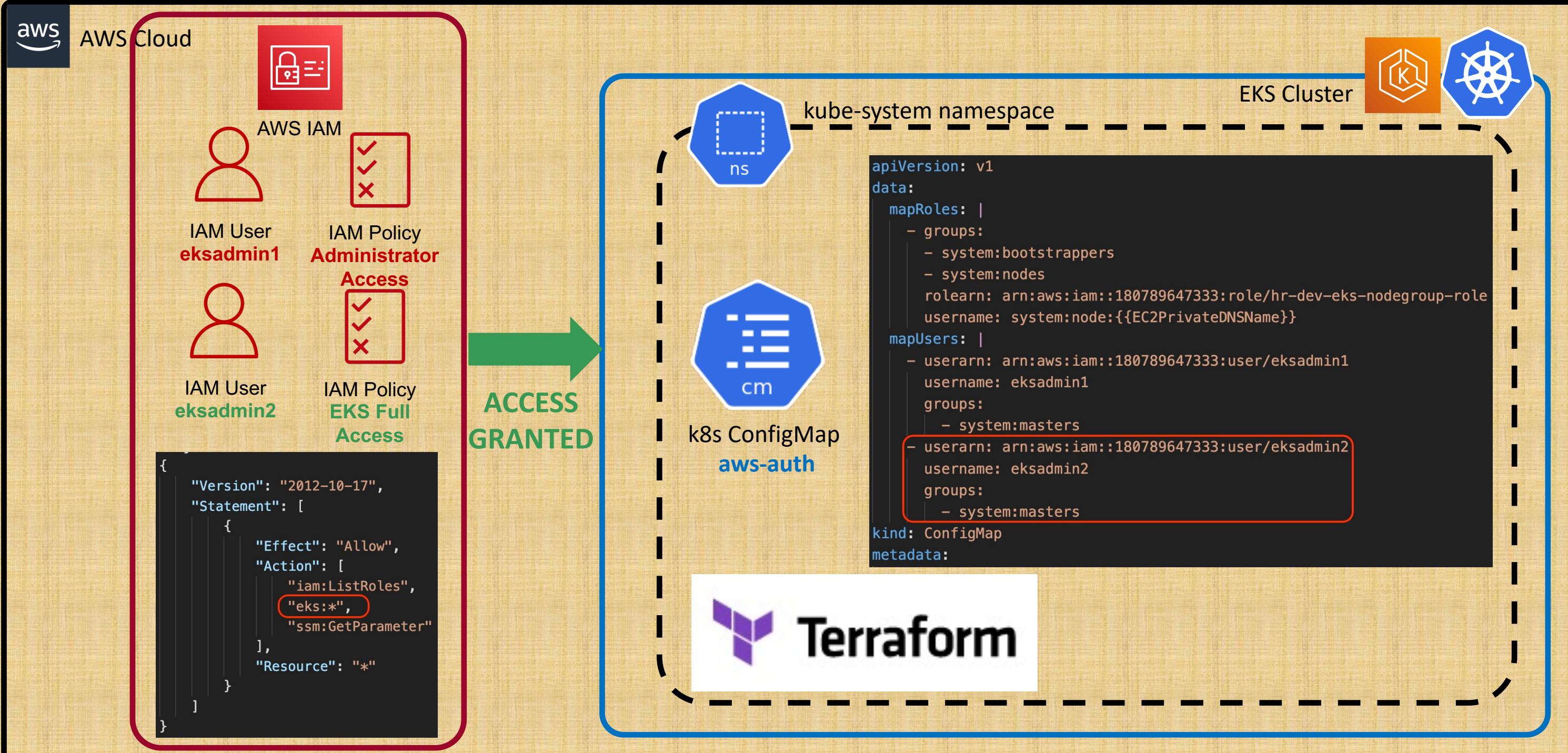
EC2 VM



Autoscaling

Section-19 & Section-20  
Automate using Terraform

# Provision AWS Basic User as EKS Admin



# Provision EKS Admins with IAM Users

**Task-1:** Create IAM User with Admin Access Policy

**Task-2:** Create IAM User with EKS Full Access policy

**Task-3:** Define Terraform Kubernetes Provider

EKS Admins with IAM Users

**Task-7:** Update EKS Node Group resource to get created only after Config Map aws-auth is created

**Task-4:** Define aws\_caller\_identity to get AWS Account ID

**Task-5:** Create locals block to define k8s config map roles and users

**Task-6:** Create Kubernetes Config Map Resource aws-auth

# What are we going to learn ?

- ✓ 21-EKS-Admins-as-AWS-IAM-Users
  - ✓ 01-ekscluster-terraform-manifests
    - > local-exec-output-files
    - > private-key
    - c1-versions.tf
    - c2-01-generic-variables.tf
    - c2-02-local-values.tf

.....

➤ c5-07-eks-node-group-public.tf

.....

- c7-01-kubernetes-provider.tf
- c7-02-kubernetes-configmap.tf
- c8-01-iam-admin-user.tf
- c8-02-iam-basic-user.tf

c7-01

c7-02

c8-01

c8-02

c5-07

Terraform Manifests

Create Terraform Kubernetes Provider

Create Kubernetes ConfigMap Resource

Create IAM Admin User with Admin Access Policy

Create IAM Basic User with EKS Full Access Policy

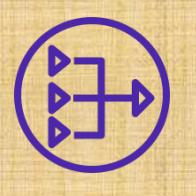
Add depends\_on Meta-argument to ensure EKS Node Group gets created only after aws-auth ConfigMap is created



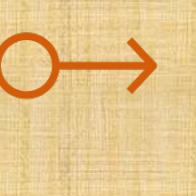
VPC



Internet gateway



NAT gateway



Elastic IP address



Route table



Public Subnet



Private Subnet



AWS S3



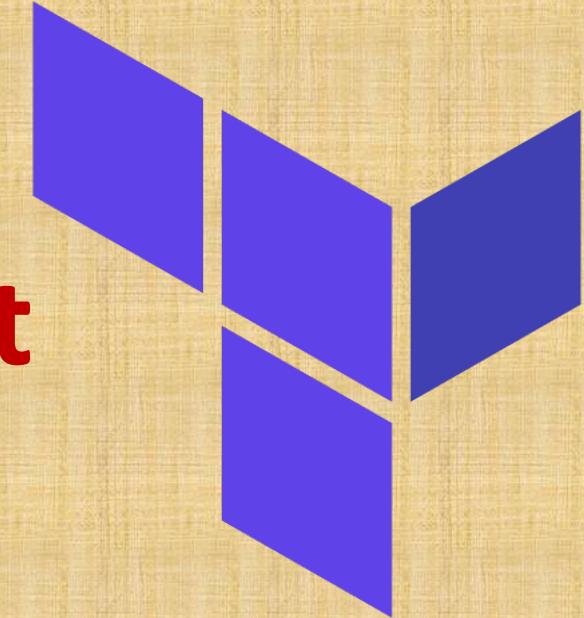
DynamoDB



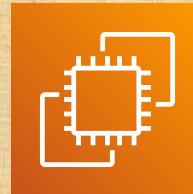
# AWS EKS

## Identity & Access Management

### using AWS IAM



EKS Cluster



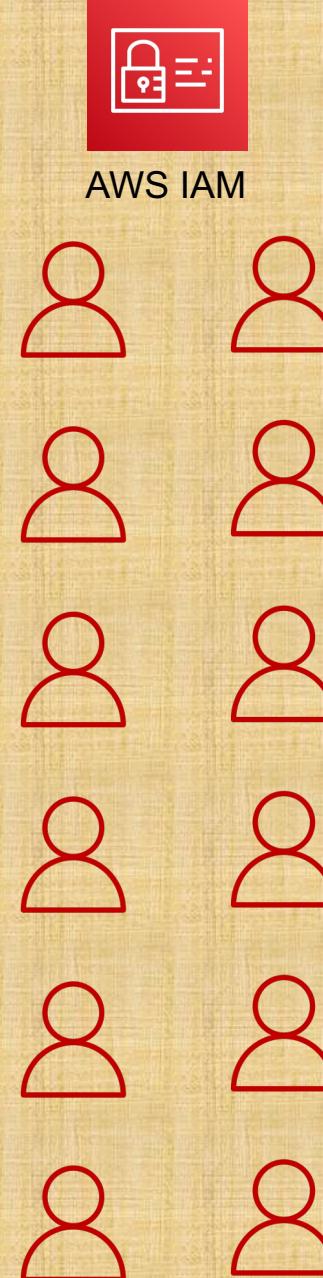
EC2 VM



Autoscaling

Provision EKS Admins with IAM Roles & IAM Groups

# Usecase - Multiple EKS Admin Users



If we need to add many IAM Users as EKS Admins (20, 30, 40, 50) what should we do ?

**Option-1:** Add those User ARNs in mapUsers section of aws-auth ConfigMap

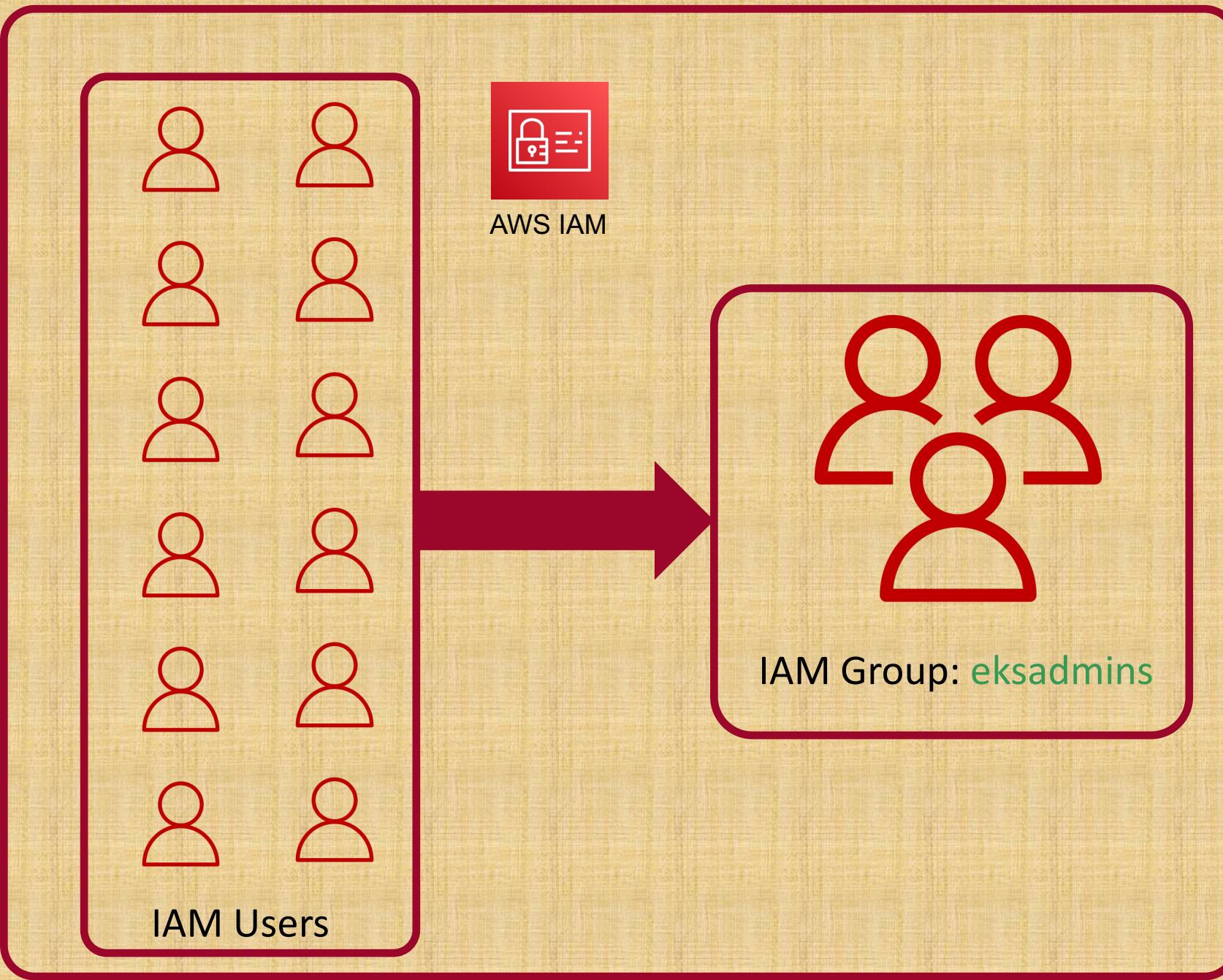
Adding all these users in aws-auth ConfigMap is not a feasible Option from maintenance perspective

Complexity increases

```
apiVersion: v1
data:
  mapRoles: |
    - groups:
      - system:bootstrappers
      - system:nodes
    rolearn: arn:aws:iam::180789647333:role/hr-dev-eks-nodegroup-role
    username: system:node:{{EC2PrivateDNSName}}
  mapUsers: |
    - userarn: arn:aws:iam::180789647333:user/eksadmin1
      username: eksadmin1
      groups:
        - system:masters
kind: ConfigMap
metadata:
  creationTimestamp: "2022-03-11T00:18:40Z"
  name: aws-auth
  namespace: kube-system
  resourceVersion: "9571"
  uid: 00614a82-89d1-4b11-a7e7-e02cb1ad2d02
```

Every time we add a new user, we need to update the Kubernetes aws-auth ConfigMap in EKS Cluster

# Usecase - Multiple EKS Admin Users



Can we use **IAM Groups** to simplify this EKS Admin Access Process ? **YES**

Setting up the whole base for this is **complex**, many things like below are involved

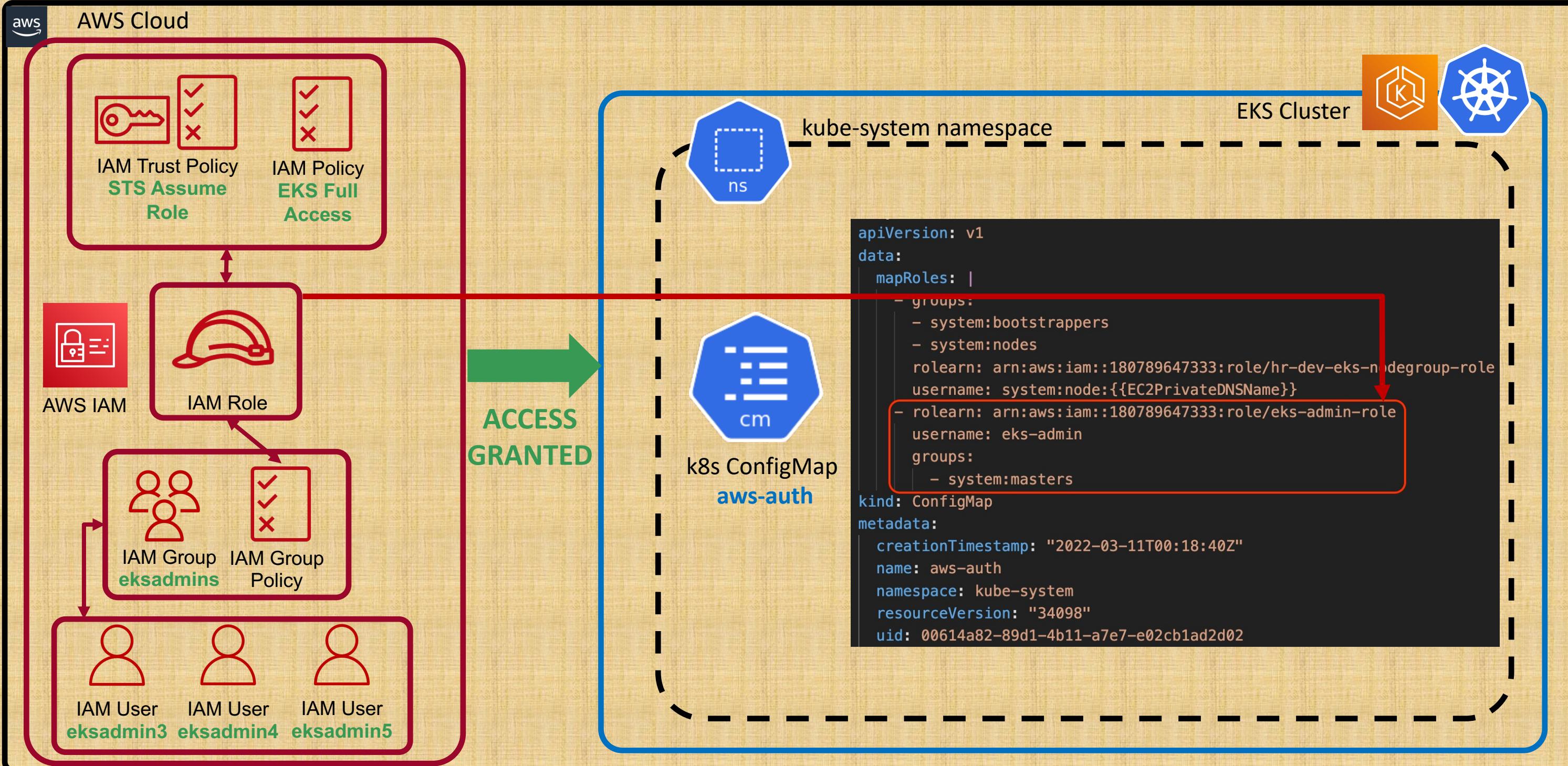
1. STS Assume Role Trust Policy
2. IAM Policy
3. IAM Role
4. IAM Group Policy
5. IAM Groups

After all steps are completed, provisioning EKS Admins becomes **so so easy**

**Step-1:** Create **IAM User**

**Step-2:** Associate that User to **IAM Group: eksadmins**. (User gets access to EKS Cluster)

# Provision EKS Admins using IAM Roles & IAM Groups



# AWS EKS - ConfigMap with mapRoles

```
apiVersion: v1
data:
  mapRoles: |
    - groups:
        - system:bootstrappers
        - system:nodes
      rolearn: arn:aws:iam::180789647333:role/hr-dev-eks-nodegroup-role
      username: system:node:{{EC2PrivateDNSName}}
    - rolearn: arn:aws:iam::180789647333:role/eks-admin-role
      username: eks-admin
      groups:
        - system:masters
kind: ConfigMap
metadata:
  creationTimestamp: "2022-03-11T00:18:40Z"
  name: aws-auth
  namespace: kube-system
  resourceVersion: "34098"
  uid: 00614a82-89d1-4b11-a7e7-e02cb1ad2d02
```

# Provision EKS Admins using IAM Roles & IAM Groups

**Section-22**  
Implement  
with manual  
steps

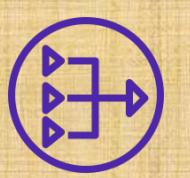
**Section-23**  
Automate  
using  
Terraform



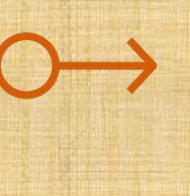
VPC



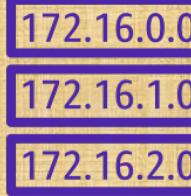
Internet gateway



NAT gateway



Elastic IP address



Route table



Public Subnet



Private Subnet



AWS S3



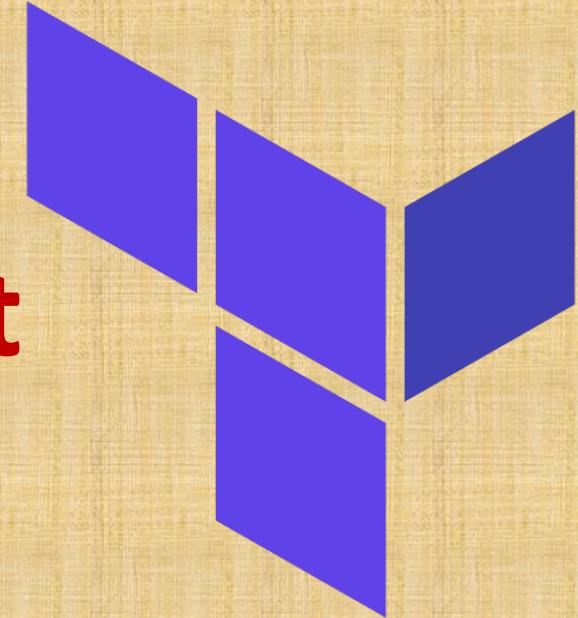
DynamoDB



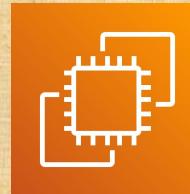
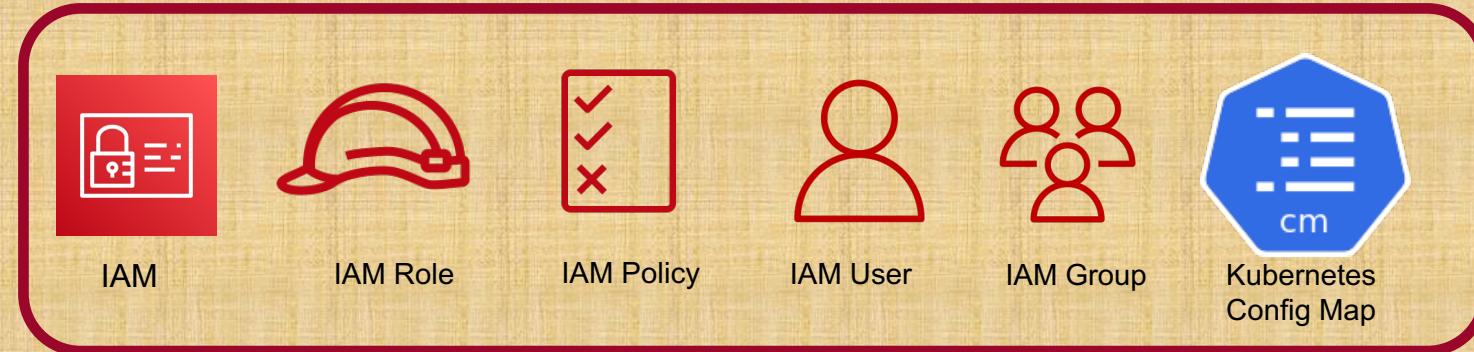
# AWS EKS

## Identity & Access Management

### using AWS IAM



EKS Cluster



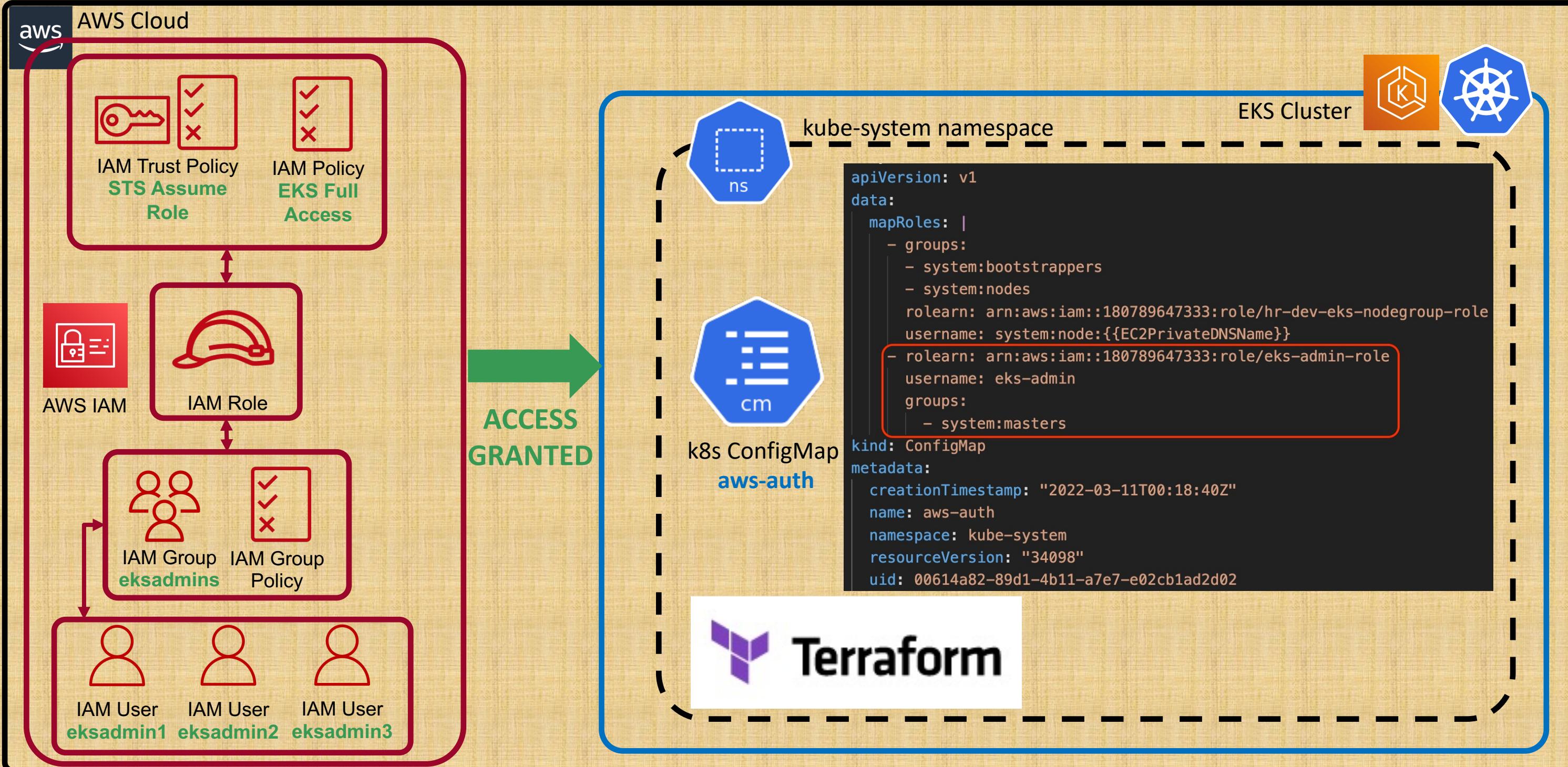
EC2 VM



Autoscaling

Provision EKS Admins with IAM Roles & IAM Groups  
Automate with Terraform

# Provision EKS Admins using IAM Roles & IAM Groups



# Provision EKS Admins using IAM Roles & IAM Groups

**Task-1:** Define Terraform Kubernetes Provider

**Task-2:** Create locals block to define k8s config map roles and users

**Task-3:** Define aws\_caller\_identity to get AWS Account ID

EKS Admins with IAM Roles & IAM Groups

**Task-4:** Create IAM Role with STS Assume Role Policy and EKS full access policy

**Task-5:** Create IAM Group Policy and IAM Group

**Task-6:** Create IAM users and associate to IAM Group

# What are we going to learn ?

- ✓ 23-EKS-Admins-with-AWS-IAM-Roles-TF
  - ✓ 01-ekscluster-terraform-manifests
    - > local-exec-output-files
    - > private-key
    - └ c1-versions.tf
    - └ c2-01-generic-variables.tf
    - └ c2-02-local-values.tf
    - └ c3-01-vpc-variables.tf
    - └ c3-02-vpc-module.tf
- .....

- └ c7-01-kubernetes-provider.tf
- └ c7-02-kubernetes-configmap.tf
- └ c8-01-iam-admin-user.tf
- └ c8-02-iam-basic-user.tf
- └ c9-01-iam-role-eksadmins.tf
- └ c9-02-iam-group-and-user-eksadmins.tf

c7-01

c7-02

c8-01

c8-02

c9-01

c9-02

Terraform Manifests

Create Terraform Kubernetes Provider

Create Kubernetes ConfigMap Resource + UPDATE  
with mapRoles changes

Create IAM Admin User with Admin Access Policy

Create IAM Basic User with EKS Full Access Policy

Create IAM Role with STS Assume Role Policy and EKS  
Full Access Policy

Create IAM Group and IAM Users and associate IAM  
Users to IAM Group

# Sample Output of aws-auth ConfigMap after creating EKS Cluster using Terraform

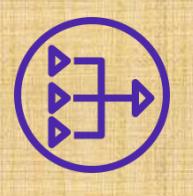
```
apiVersion: v1
data:
  mapRoles: |
    - "groups":
      - "system:bootstrappers"
      - "system:nodes"
      "rolearn": "arn:aws:iam::180789647333:role/hr-dev-eks-nodegroup-role"
      "username": "system:node:{{EC2PrivateDNSName}}"
    - "groups": Added as part of Section-23
      - "system:masters"
      "rolearn": "arn:aws:iam::180789647333:role/hr-dev-eks-admin-role"
      "username": "eks-admin"
  mapUsers: |
    - "groups": Added as part of Section-21
      - "system:masters"
      "userarn": "arn:aws:iam::180789647333:user/hr-dev-eksadmin2"
      "username": "hr-dev-eksadmin2"
    - "groups":
      - "system:masters"
      "userarn": "arn:aws:iam::180789647333:user/hr-dev-eksadmin1"
      "username": "hr-dev-eksadmin1"
kind: ConfigMap
metadata:
  creationTimestamp: "2022-04-23T10:38:00Z"
  name: aws-auth
  namespace: kube-system
```



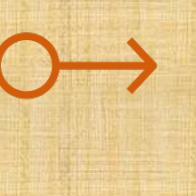
VPC



Internet gateway



NAT gateway



Elastic IP address



Route table



Public Subnet



Private Subnet



AWS S3



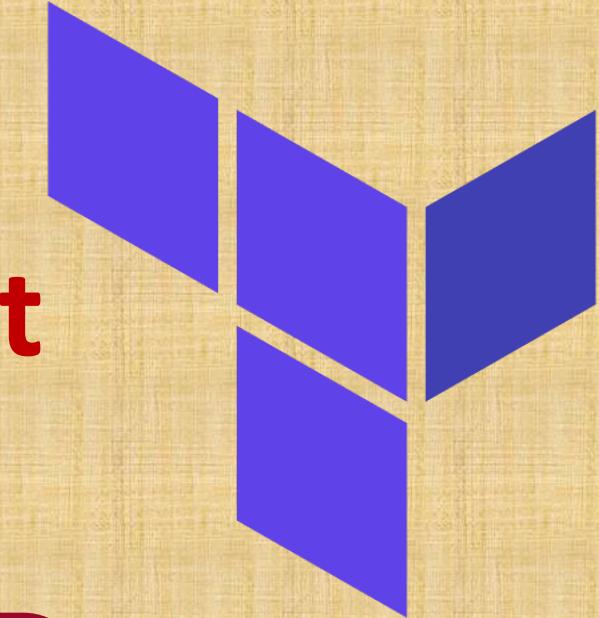
DynamoDB



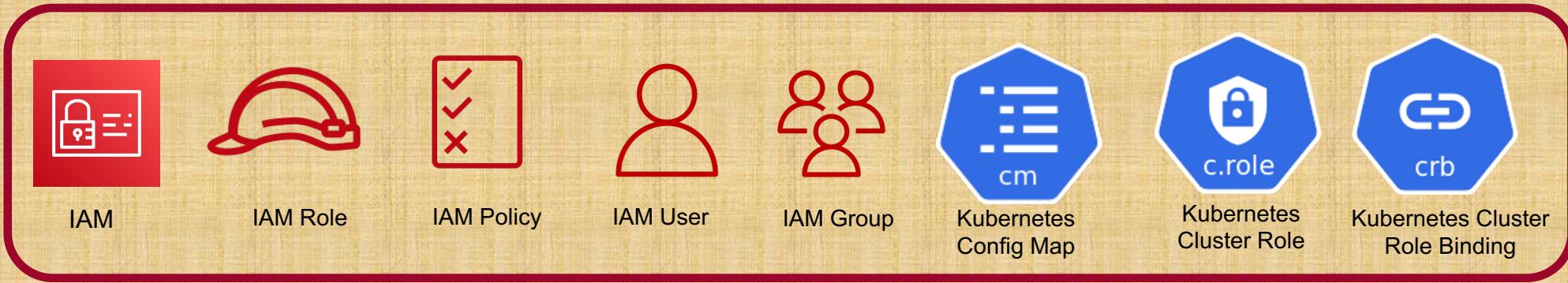
# AWS EKS

## Identity & Access Management

### using AWS IAM



EKS Cluster



Autoscaling

## Kubernetes RBAC Fundamentals

# Kubernetes RBAC

## Kubernetes RBAC - Fundamentals

### Subjects

Users or processes that need access to the Kubernetes API

Kind: Group, User

Kind: Service Account

### API Groups

The k8s API objects that we grant access to

core

extensions

apps

batch

### Resources

Pods

Deployments

Services

StatefulSets

### Verbs

List of actions that can be taken on a resource

Create

Patch

List

get

Watch

Replace

Delete

Read

**Kubernetes API Reference:** <https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.22/>

**API Groups Reference:** <https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.22/#-strong-api-groups-strong->

# Kubernetes RBAC



A **ClusterRole** can be used to grant the same permissions as a **Role**, but because they are **cluster-scoped**, access will be granted across cluster



A **Cluster Role Binding** is used to tie the **Cluster Role** and **Subject** together

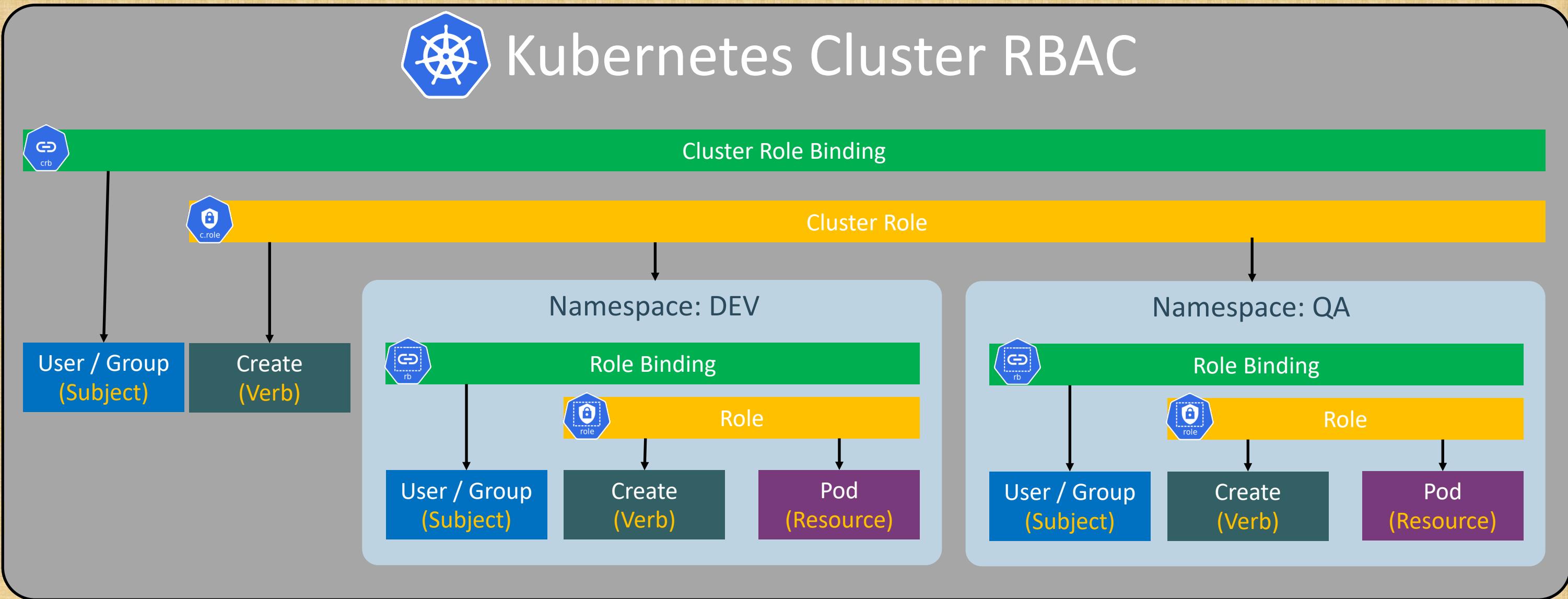


A **Role** can only be used to grant access to resources within a single namespace.



A **Role Binding** is used to tie the **Role** and **Subject** together

# Kubernetes RBAC



```
# Resource: Cluster Role
resource "kubernetes_cluster_role_v1" "eksreadonly_clusterrole" {
  metadata {
    name = "${local.name}-eksreadonly-clusterrole"
  }
  rule {
    api_groups = [""] # These come under core APIs
    resources  = ["nodes", "namespaces", "pods", "events", "services"]
    #resources  = ["nodes", "namespaces", "pods", "events", "services", "config"]
    verbs       = ["get", "list"]
  }
  rule {
    api_groups = ["apps"]
    resources  = ["deployments", "daemonsets", "statefulsets", "replicasets"]
    verbs       = ["get", "list"]
  }
  rule {
    api_groups = ["batch"]
    resources  = ["jobs"]
    verbs       = ["get", "list"]
  }
}
```

# Cluster Role

# Cluster Role Binding

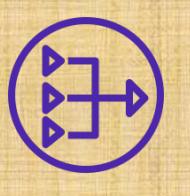
```
# Resource: Cluster Role Binding
resource "kubernetes_cluster_role_binding_v1" "eksreadonly_clusterrolebinding" {
  metadata {
    name = "${local.name}-eksreadonly-clusterrolebinding"
  }
  role_ref {
    api_group = "rbac.authorization.k8s.io"
    kind      = "ClusterRole"
    name      = kubernetes_cluster_role_v1.eksreadonly_clusterrole.metadata.0.name
  }
  subject {
    kind      = "Group"
    name      = "eks-readonly-group"
    api_group = "rbac.authorization.k8s.io"
  }
}
```



VPC



Internet gateway



NAT gateway



Elastic IP address



Route table



Public Subnet



Private Subnet



AWS S3



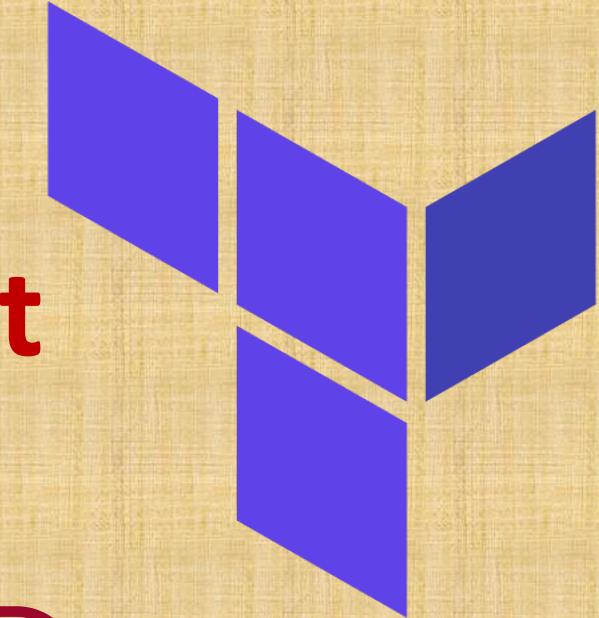
DynamoDB



# AWS EKS

## Identity & Access Management

### using AWS IAM



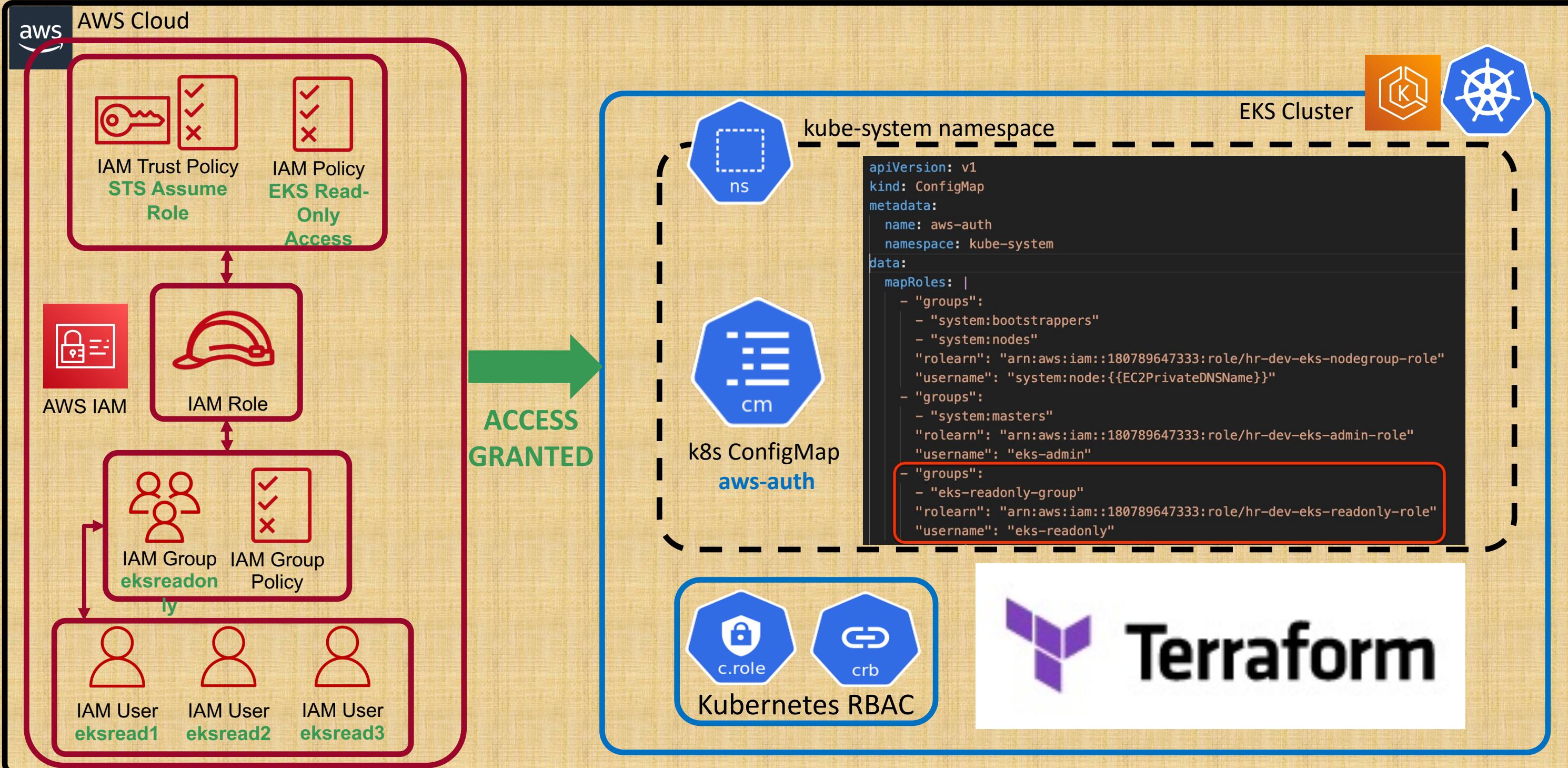
EKS Cluster



Autoscaling

Provision EKS Read-Only Users with IAM Roles & IAM Groups  
Automate with Terraform

# Provision EKS ReadOnly Users using IAM Roles & IAM Groups



```
apiVersion: v1
kind: ConfigMap
metadata:
  name: aws-auth
  namespace: kube-system
data:
  mapRoles: |
    - "groups":
      - "system:bootstrappers"
      - "system:nodes"
      "rolearn": "arn:aws:iam::180789647333:role/hr-dev-eks-nodegroup-role"
      "username": "system:node:{{EC2PrivateDNSName}}"
    - "groups":
      - "system:masters"
      "rolearn": "arn:aws:iam::180789647333:role/hr-dev-eks-admin-role"
      "username": "eks-admin"
    - "groups":
      - "eks-readonly-group"
      "rolearn": "arn:aws:iam::180789647333:role/hr-dev-eks-readonly-role"
      "username": "eks-readonly"
  mapUsers: |
    - "groups":
      - "system:masters"
      "userarn": "arn:aws:iam::180789647333:user/hr-dev-eksadmin2"
      "username": "hr-dev-eksadmin2"
    - "groups":
      - "system:masters"
      "userarn": "arn:aws:iam::180789647333:user/hr-dev-eksadmin1"
      "username": "hr-dev-eksadmin1"
```

# aws-auth ConfigMap

```
# Resource: Cluster Role
resource "kubernetes_cluster_role_v1" "eksreadonly_clusterrole" {
  metadata {
    name = "${local.name}-eksreadonly-clusterrole"
  }
  rule {
    api_groups = [""] # These come under core APIs
    resources  = ["nodes", "namespaces", "pods", "events", "services"]
    #resources = ["nodes", "namespaces", "pods", "events", "services", "config"]
    verbs      = ["get", "list"]
  }
  rule {
    api_groups = ["apps"]
    resources  = ["deployments", "daemonsets", "statefulsets", "replicasets"]
    verbs      = ["get", "list"]
  }
  rule {
    api_groups = ["batch"]
    resources  = ["jobs"]
    verbs      = ["get", "list"]
  }
}
```

# Cluster Role

# Cluster Role Binding

```
# Resource: Cluster Role Binding
resource "kubernetes_cluster_role_binding_v1" "eksreadonly_clusterrolebinding" {
  metadata {
    name = "${local.name}-eksreadonly-clusterrolebinding"
  }
  role_ref {
    api_group = "rbac.authorization.k8s.io"
    kind      = "ClusterRole"
    name      = kubernetes_cluster_role_v1.eksreadonly_clusterrole.metadata.0.name
  }
  subject {
    kind      = "Group"
    name      = "eks-readonly-group"
    api_group = "rbac.authorization.k8s.io"
  }
}
```

# ClusterRoleBinding & aws-auth ConfigMap

```
# Resource: Cluster Role Binding
resource "kubernetes_cluster_role_binding_v1" "eksreadonly"
  metadata {
    name = "${local.name}-eksreadonly-clusterrolebinding"
  }
  role_ref {
    api_group = "rbac.authorization.k8s.io"
    kind      = "ClusterRole"
    name      = kubernetes_cluster_role_v1.eksreadonly_clu
  }
  subject {
    kind      = "Group"
    name      = "eks-readonly-group"
    api_group = "rbac.authorization.k8s.io"
  }
}
```

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: aws-auth
  namespace: kube-system
data:
  mapRoles: |
    - "groups":
      - "system:bootstrappers"
      - "system:nodes"
      "rolearn": "arn:aws:iam::180789647333:role/hr-dev-eks-nodegroup-role"
      "username": "system:node:{EC2PrivateDNSName}"
    - "groups":
      - "system:masters"
      "rolearn": "arn:aws:iam::180789647333:role/hr-dev-eks-admin-role"
      "username": "eks-admin"
    - "groups":
      - "eks-readonly-group"
      "rolearn": "arn:aws:iam::180789647333:role/hr-dev-eks-readonly-role"
      "username": "eks-readonly"
```

How does the **aws-auth ConfigMap** enforce the **Kubernetes RBAC permissions** defined in **ClusterRole & ClusterRoleBinding**?

Subject Name in **ClusterRoleBinding** should match the **groups value** in **aws-auth ConfigMap**

# Provision EKS ReadOnly Users using IAM Roles, IAM Groups with Kubernetes RBAC (Cluster Role & Cluster Role Binding)

**Task-1:** Define Terraform Kubernetes Provider

**Task-2:** Create locals block to define k8s config map roles and users

**Task-3:** Define aws\_caller\_identity to get AWS Account ID

**Task-8:** update aws-auth ConfigMap

EKS Read-Only Users with IAM Roles & IAM Groups

**Task-7:** Create Cluster Role and Cluster Role Binding k8s RBAC Resources

**Task-4:** Create IAM Role with STS Assume Role Policy and EKS Read-Only access policy

**Task-5:** Create IAM Group Policy and IAM Group (eksreadonly)

**Task-6:** Create IAM users and associate to IAM Group (eksreaduser1)

```
✓ 24-EKS-ReadOnly-IAM-Users
  ✓ 01-ekscluster-terraform-manifests
    > local-exec-output-files
    > private-key
    ✓ c1-versions.tf
    ✓ c2-01-generic-variables.tf
    ✓ c2-02-local-values.tf
    ✓ c3-01-vpc-variables.tf
    ✓ c3-02-vpc-module.tf
    ✓ c3-03-vpc-outputs.tf
```

```
.....
```

```
✓ c7-01-kubernetes-provider.tf
✓ c7-02-kubernetes-configmap.tf
✓ c8-01-iam-admin-user.tf
✓ c8-02-iam-basic-user.tf
✓ c9-01-iam-role-eksadmins.tf
✓ c9-02-iam-group-and-user-eksadmins.tf
✓ c10-01-iam-role-eksreadonly.tf
✓ c10-02-iam-group-and-user-eksreadonly.tf
✓ c10-03-k8s-clusterrole-clusterrolebinding.tf
```

# What are we going to learn ?

Terraform Manifests

c7-01

Create Terraform Kubernetes Provider

c7-02

Create Kubernetes ConfigMap Resource + UPDATE  
with mapRoles changes

c8 & c9

Create IAM Admin User with mapUsers in aws-auth  
Create EKS Admins with IAM Roles & mapRoles

c10-01

Create IAM Role with STS Assume Role Policy and EKS  
READONLY Access Policy

c10-02

Create IAM Group and IAM Users and associate IAM  
Users to IAM Group

c10-03

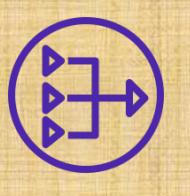
Create Kubernetes RBAC ClusterRole &  
ClusterRoleBinding Resources



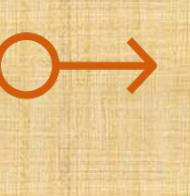
VPC



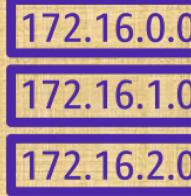
Internet gateway



NAT gateway



Elastic IP address



Route table



Public Subnet



Private Subnet



AWS S3



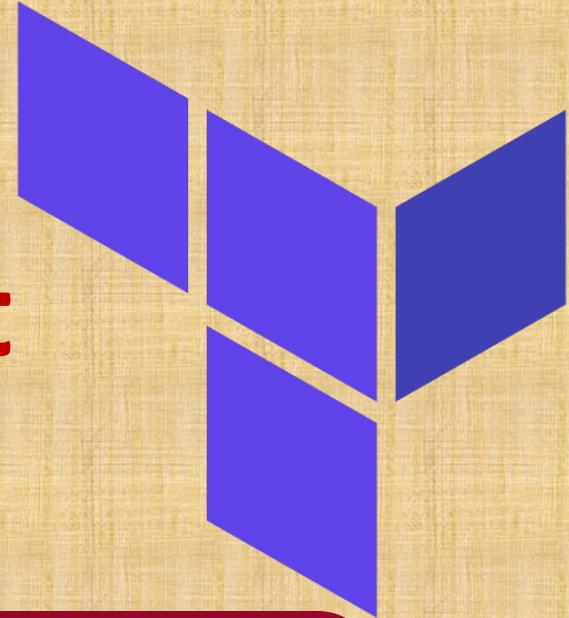
DynamoDB



# AWS EKS

## Identity & Access Management

### using AWS IAM



EKS Cluster



IAM



IAM Role



IAM Policy



IAM User



IAM Group



Kubernetes Config Map



c.role



Kubernetes Cluster Role Binding



crb



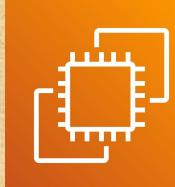
Kubernetes Role Binding



Kubernetes Role



Kubernetes Role Binding



EC2 VM

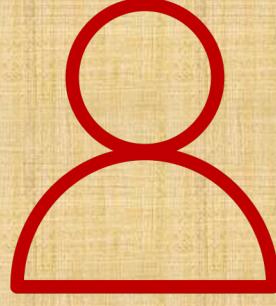


Autoscaling

Provision EKS Developer Users with IAM Roles & IAM Groups  
Automate with Terraform

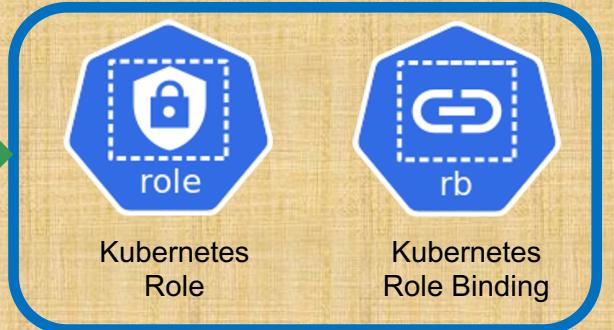
# Usecase

A Developer requesting **full access** to a namespace (**dev**) in EKS Cluster and in addition, also asking for **read-only** access to EKS Cluster

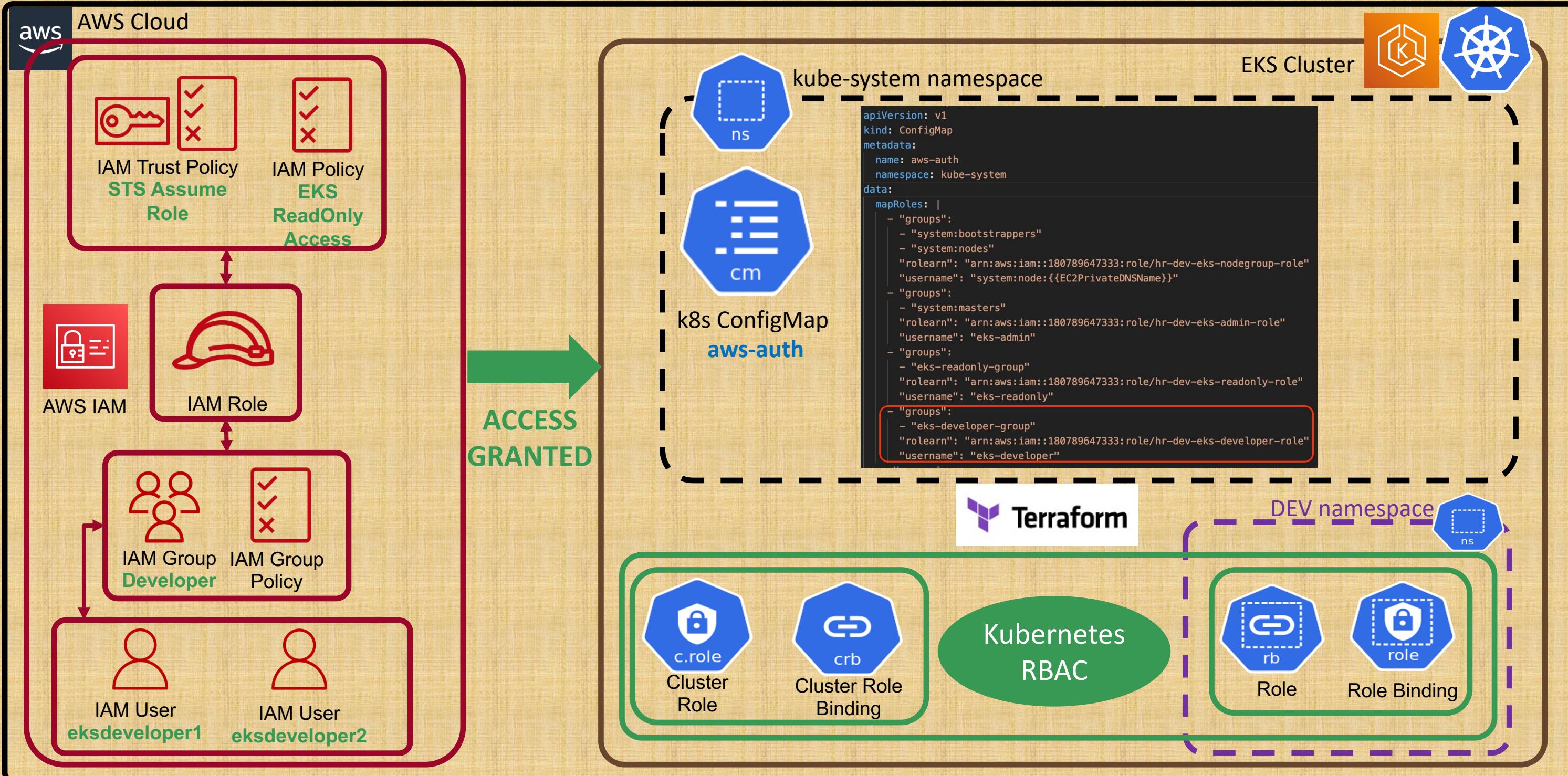
  
**EKS Developer**  
**IAM User**

EKS Cluster Read-Only Access

EKS Dev Namespace Full Access  
(All Verbs – Create , Apply, Get, List, Delete Resources)



# Provision EKS Developer Users using IAM Roles & IAM Groups



# Kubernetes Namespace Dev

```
# Resource: k8s namespace
resource "kubernetes_namespace_v1" "k8s_dev" {
  metadata {
    name = "dev"
  }
}
```

# aws-auth ConfigMap

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: aws-auth
  namespace: kube-system
data:
  mapRoles: |
    - "groups":
      - "system:bootstrappers"
      - "system:nodes"
      "rolearn": "arn:aws:iam::180789647333:role/hr-dev-eks-nodegroup-role"
      "username": "system:node:{EC2PrivateDNSName}"
    - "groups":
      - "system:masters"
      "rolearn": "arn:aws:iam::180789647333:role/hr-dev-eks-admin-role"
      "username": "eks-admin"
    - "groups":
      - "eks-readonly-group"
      "rolearn": "arn:aws:iam::180789647333:role/hr-dev-eks-readonly-role"
      "username": "eks-readonly"
    - "groups":
      - "eks-developer-group"
      "rolearn": "arn:aws:iam::180789647333:role/hr-dev-eks-developer-role"
      "username": "eks-developer"
  mapUsers: |
    - "groups":
      - "system:masters"
      "userarn": "arn:aws:iam::180789647333:user/hr-dev-eksadmin2"
      "username": "hr-dev-eksadmin2"
    - "groups":
      - "system:masters"
      "userarn": "arn:aws:iam::180789647333:user/hr-dev-eksadmin1"
      "username": "hr-dev-eksadmin1"
```

Section-23

EKS Admins with IAM Users,  
Groups and Roles

Section-24

EKS ReadOnly Access with IAM Users,  
Groups, Roles & k8s ClusterRole &  
ClusterRoleBinding

Section-25

EKS Developer Access with IAM Users,  
Groups, Roles & k8s ClusterRole,  
ClusterRoleBinding & Role, RoleBinding

Section-21

IAM Users as EKS Admins

# Kubernetes Role

```
# Resource: k8s Role
resource "kubernetes_role_v1" "eksdeveloper_role" {
  metadata {
    name = "${local.name}-eksdeveloper-role"
    namespace = kubernetes_namespace_v1.k8s_dev.metadata[0].name
  }

  rule {
    api_groups      = [ "", "extensions", "apps" ]
    resources       = [ "*" ]
    verbs           = [ "*" ]
  }

  rule {
    api_groups = [ "batch" ]
    resources  = [ "jobs", "cronjobs" ]
    verbs       = [ "*" ]
  }
}
```

```
# Resource: k8s Role Binding
resource "kubernetes_role_binding_v1" "eksdeveloper_rolebinding" {
  metadata {
    name      = "${local.name}-eksdeveloper-rolebinding"
    namespace = kubernetes_namespace_v1.k8s_dev.metadata[0].name
  }
  role_ref {
    api_group = "rbac.authorization.k8s.io"
    kind      = "Role"
    name      = kubernetes_role_v1.eksdeveloper_role.metadata.0.name
  }
  subject {
    kind      = "Group"
    name      = "eks-developer-group"
    api_group = "rbac.authorization.k8s.io"
  }
}
```

# Kubernetes Role Binding

# aws-auth ConfigMap

```
c7-02-kubernetes-configmap.tf ×  
terraform-on-aws-eks > 25-EKS-DeveloperAccess-IAM-Users > 01-ekscluster-terraform-manifests > c7-02-kubernetes-configmap.tf > locals > [ ] configmap_roles > 0 > rolearn  
11 configmap_roles = [  
12   {  
13     #rolearn  = "arn:aws:iam::${data.aws_caller_identity.current.account_id}:role/${aws_iam_role.eks_nodegroup_"  
14     rolearn = "${aws_iam_role.eks_nodegroup_role.arn}"  
15     username = "system:node:${EC2PrivateDNSName}"  
16     groups   = ["system:bootstrappers", "system:nodes"]  
17   },  
18   {  
19     rolearn  = "${aws_iam_role.eks_admin_role.arn}"  
20     username = "eks-admin" # Just a place holder name  
21     groups   = ["system:masters"]  
22   },  
23   {  
24     rolearn  = "${aws_iam_role.eks_READONLY_ROLE.arn}"  
25     username = "eks-readonly" # Just a place holder name  
26     #groups  = [ "eks-readonly-group" ]  
27     # Important Note: The group name specified in clusterrolebinding and in aws-auth configmap groups should be  
28     groups   = [ "${kubernetes_cluster_role_binding_v1.eksreadonly_clusterrolebinding.subject[0].name}" ]  
29   },  
30   {  
31     rolearn  = "${aws_iam_role.eks_developer_role.arn}"  
32     username = "eks-developer" # Just a place holder name  
33     #groups  = [ "eks-developer-group" ]  
34     # Important Note: The group name specified in clusterrolebinding and in aws-auth configmap groups should be  
35     groups   = [ "${kubernetes_role_binding_v1.eksdeveloper_rolebinding.subject[0].name}" ]  
36   },
```

# Provision EKS ReadOnly Users using IAM Roles, IAM Groups with Kubernetes RBAC (Cluster Role & Cluster Role Binding)

**Task-1:** Define Terraform Kubernetes Provider

**Task-2:** Create locals block to define k8s config map roles and users

**Task-3:** Define aws\_caller\_identity to get AWS Account ID

**Task-8:** update aws-auth ConfigMap

EKS Read-Only Users with IAM Roles & IAM Groups

**Task-7:** Create Cluster Role, Cluster Role Binding, Role, Role Binding k8s RBAC Resources

**Task-4:** Create IAM Role with STS Assume Role Policy and EKS Developer access policy

**Task-5:** Create IAM Group Policy and IAM Group (eksdeveloper)

**Task-6:** Create IAM users and associate to IAM Group (eksdevuser1)

```
✓ 25-EKS-DeveloperAccess-IAM-Users  
  ✓ 01-ekscluster-terraform-manifests  
    > local-exec-output-files  
    > private-key  
    ✓ c1-versions.tf  
.....
```

```
✓ c7-01-kubernetes-provider.tf  
✓ c7-02-kubernetes-configmap.tf  
✓ c8-01-iam-admin-user.tf  
✓ c8-02-iam-basic-user.tf  
✓ c9-01-iam-role-eksadmins.tf  
✓ c9-02-iam-group-and-user-eksadmins.tf  
✓ c10-01-iam-role-eksreadonly.tf  
✓ c10-02-iam-group-and-user-eksreadonly.tf  
✓ c10-03-k8s-clusterrole-clusterrolebinding.tf  
✓ c11-01-iam-role-eksdeveloper.tf  
✓ c11-02-iam-group-and-user-eksdeveloper.tf  
✓ c11-03-k8s-clusterrole-clusterrolebinding.tf  
✓ c11-04-namespaces.tf  
✓ c11-05-k8s-role-rolebinding.tf
```

# What are we going to learn ?

## Terraform Manifests

c7

Create Terraform Kubernetes Provider

c7-02

Create Kubernetes ConfigMap Resource + **UPDATE**  
with **mapRoles** changes

c8, c9  
& c10

Create IAM Admin User with **mapUsers** in aws-auth  
Create EKS Admins with IAM Roles & **mapRoles**

c11-01

Create IAM Role with STS Assume Role Policy and EKS  
READONLY Access Policy for **DEVELOPERS**

c11-02

Create IAM Group and IAM Users and associate IAM  
Users to IAM Group – **DEVELOPER ACCESS**

c11-03

Create Kubernetes RBAC **ClusterRole** &  
**ClusterRoleBinding** Resources

c11-04

Create Dev Namespace

c11-05

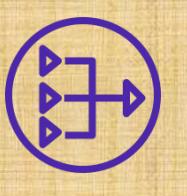
Create Kubernetes RBAC **Role** & **RoleBinding**  
Resources



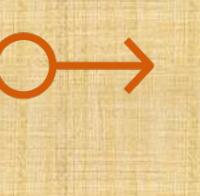
VPC



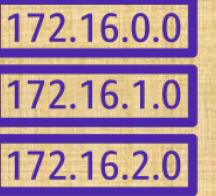
Internet gateway



NAT gateway



Elastic IP address



Route table



Public Subnet



Private Subnet



AWS S3

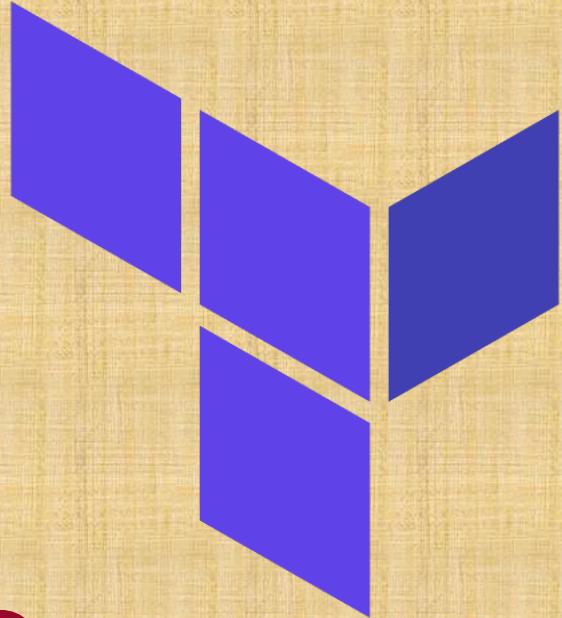


DynamoDB

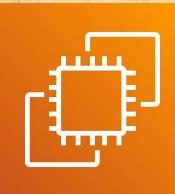


# AWS EKS

## Load Balancer Controller



EKS Cluster



EC2 VM



Autoscaling

17 Demos



AWS Load Balancer  
Controller Install

Ingress Basics

Ingress Context Path  
Routing

Ingress SSL & SSL Redirect

Ingress Cross Namespaces

External DNS Install

Ingress + External DNS

D1

D2

D3

D4

D5

D6

D7

AWS Load  
Balancer  
Controller

AWS  
Application  
Load  
Balancer

Kubernetes  
Ingress  
Service

Terraform  
Kubernetes  
Provider

D8

D9

D10

D11

D12

D13

D14

k8s Service + External DNS

Ingress Name based Virtual  
Host Routing

SSL Discovery - Host

SSL Discovery - TLS

Ingress Groups

Ingress Target Type - IP

Ingress Internal ALB



AWS Load  
Balancer  
Controller

AWS  
Network  
Load  
Balancer

Kubernetes  
Service

Terraform  
Kubernetes  
Provider

D15

Network Load Balancer  
Basics

D16

Network Load Balancer  
TLS + External DNS

D17

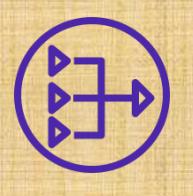
Network Load Balancer  
Internal LB



VPC



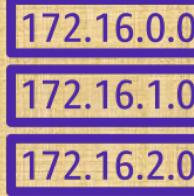
Internet gateway



NAT gateway



Elastic IP address



Route table



Public Subnet



Private Subnet



AWS S3

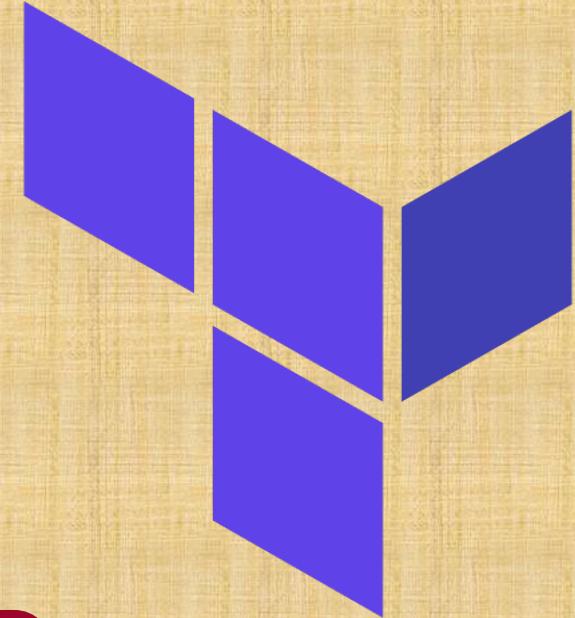


DynamoDB

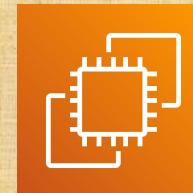


# AWS EKS

## Load Balancer Controller



EKS Cluster



EC2 VM



Autoscaling

Install AWS Load Balancer Controller on EKS Cluster  
Automate with Terraform

# Kubernetes Ingress

AWS ALB Ingress Controller

AWS Application Load Balancer Only

Doesn't support latest Ingress API Version  
(`apiVersion: networking.k8s.io/v1`) from  
Kubernetes Version 1.22 onwards

No Development releases, Soon it will be  
deprecated

Renamed & Redesigned

AWS Load  
Balancer  
Controller

AWS Application and Network Load  
Balancers

K8s Ingress  
Resource

K8s Service  
Resource

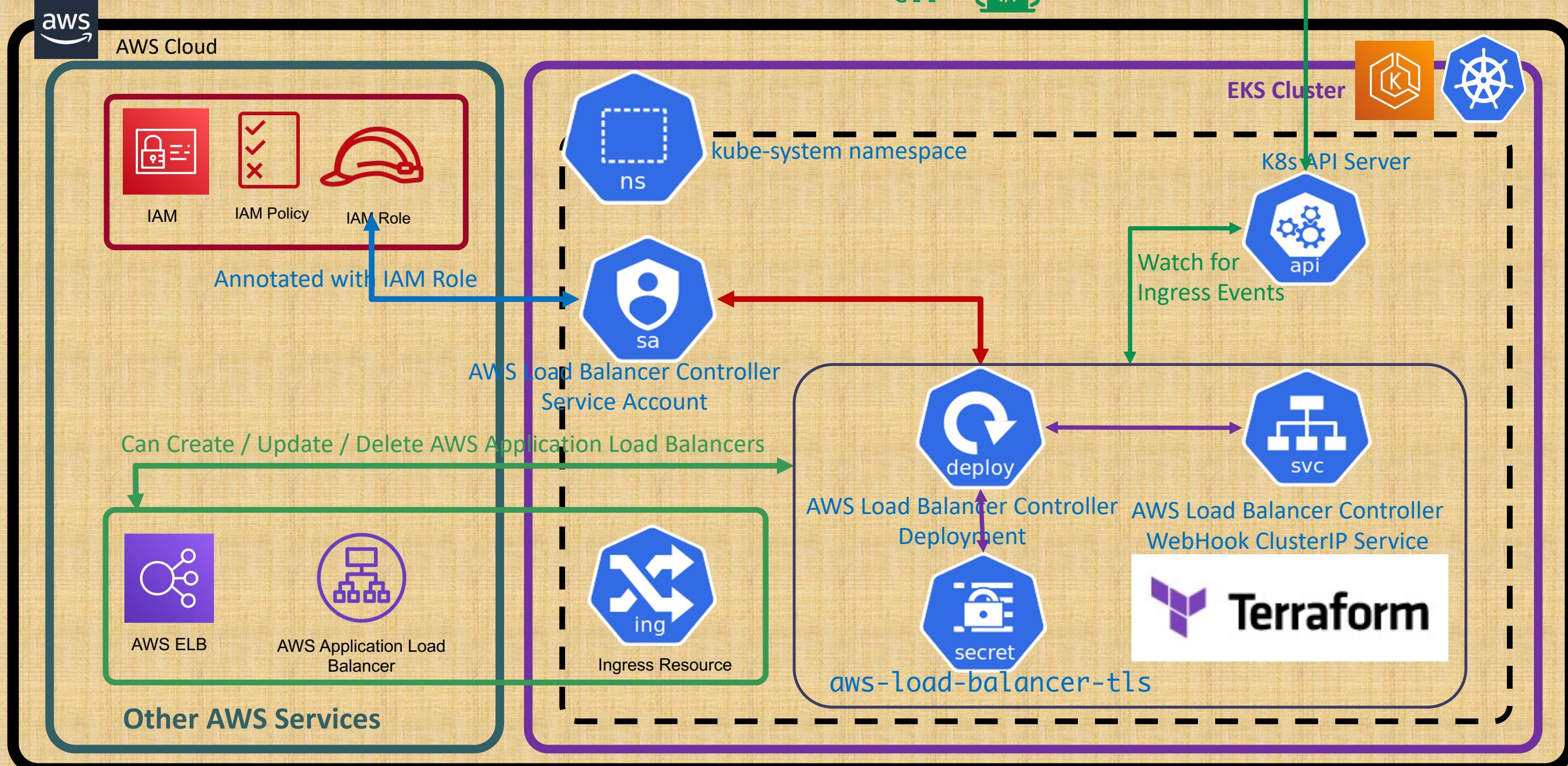
AWS Application  
Load Balancer

AWS Network  
Load Balancer

Supports latest and greatest from Ingress  
perspective

# AWS Load Balancer Controller

kubectl cli Deploy Ingress k8s manifest



- ✓ 26-EKS-with-LoadBalancer-Controller
  - ✓ 01-ekscluster-terraform-manifests
    - > local-exec-output-files
    - > private-key
    - c1-versions.tf
    - c2-01-generic-variables.tf
    - c2-02-local-values.tf

• •

- c5-01-eks-variables.tf
- c5-02-eks-outputs.tf
- c5-03-iamrole-for-eks-cluster.tf
- c5-04-iamrole-for-eks-nodegroup.tf
- c5-05-securitygroups-eks.tf
- c5-06-eks-cluster.tf
- c5-07-eks-node-group-public.tf
- c5-08-eks-node-group-private.tf
- c6-01-iam-oidc-connect-provider-variables.tf
- c6-02-iam-oidc-connect-provider.tf

• •

# What are we going to learn ?

## Terraform Manifests

Comment Public Node Group Outputs & uncomment Private Node Group Outputs

Comment the Code related to EKS Public Node Group

Uncomment the Code related to EKS Private Node Group

Add EKS Cluster as OpenID Connect Provider in AWS IAM Service

c5-02

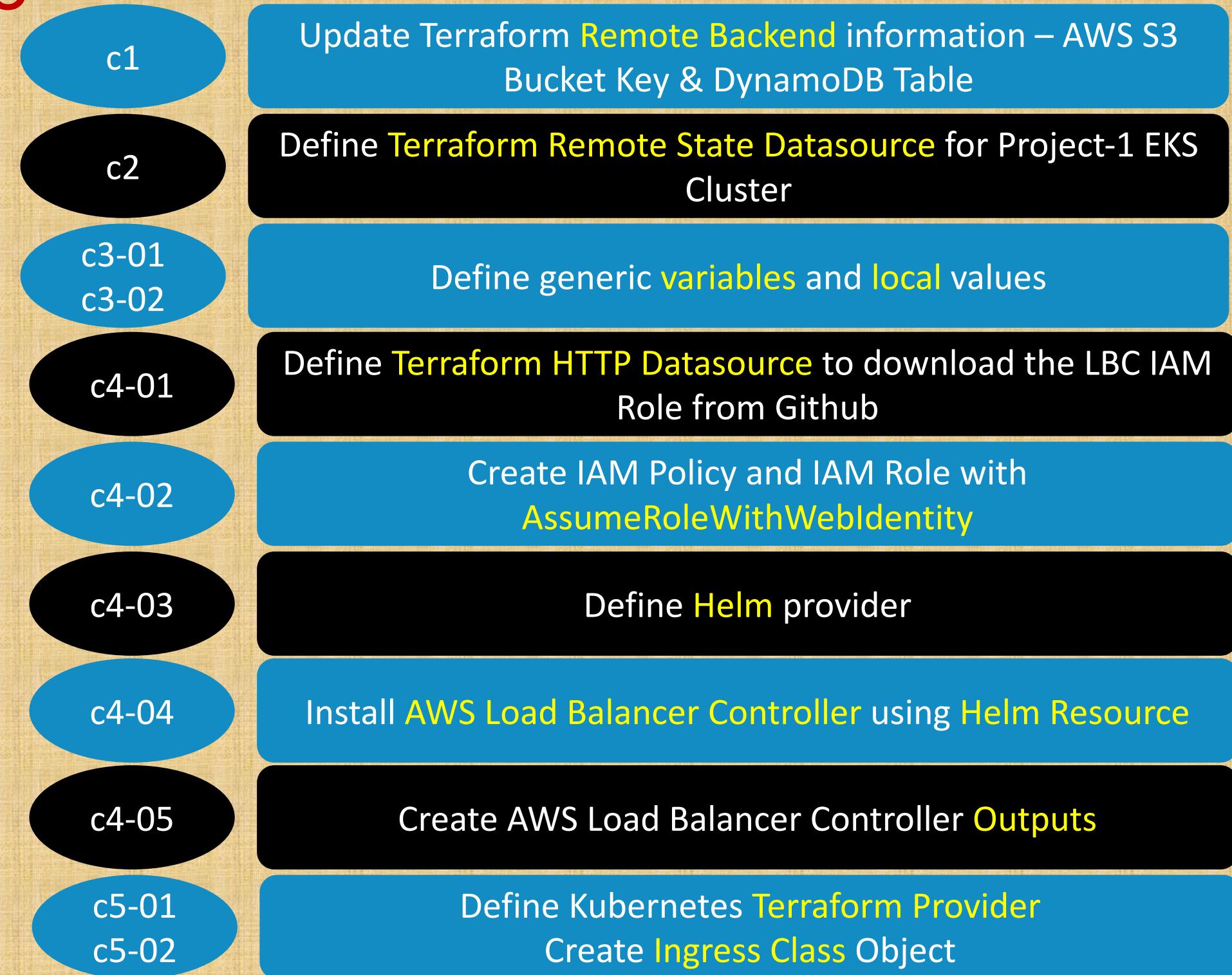
c5-07

c5-08

c6-01  
c6-02

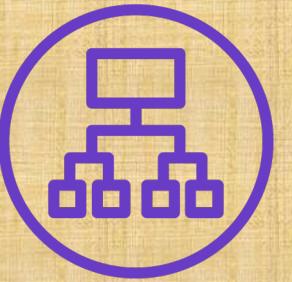
# What are we going to learn ?

- ✓ 26-EKS-with-LoadBalancer-Controller
  - > 01-ekscluster-terraform-manifests
  - ✓ 02-lbc-install-terraform-manifests
    - └ c1-versions.tf
    - └ c2-remote-state-datasource.tf
    - └ c3-01-generic-variables.tf
    - └ c3-02-local-values.tf
    - └ c4-01-lbc-datasources.tf
    - └ c4-02-lbc-iam-policy-and-role.tf
    - └ c4-03-lbc-helm-provider.tf
    - └ c4-04-lbc-install.tf
    - └ c4-05-lbc-outputs.tf
    - └ c5-01-kubernetes-provider.tf
    - └ c5-02-ingress-class.tf
    - └ terraform.tfvars

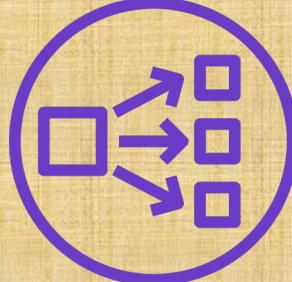




Elastic Load  
Balancing



Application  
Load Balancer



Network  
Load Balancer



Kubernetes  
Ingress



# AWS EKS

## Load Balancer Controller

### Kubernetes Ingress Class



# Kubernetes Ingress Class

AWS EKS ALB  
Ingress Controller

NGINX Ingress  
Controller

AKS Application  
Gateway Ingress  
Controller

If we have multiple Ingress Controllers running in our Kubernetes cluster, how to identify to which Ingress Controller our Ingress Resource/Service should be associated to ?

Kubernetes Object  
Kind: IngressClass

**More Ingress Controllers:** <https://kubernetes.io/docs/concepts/services-networking/ingress-controllers/>

# Kubernetes Ingress Class

```
apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  name: my-aws-ingress-class
  annotations:
    ingressclass.kubernetes.io/is-default-class: "true"
spec:
  controller: ingress.k8s.aws/alb
```

# Ingress Class

```
apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  name: my-aws-ingress-class
  #annotations:
    #ingressclass.kubernetes.io/is-default-class: "true"
spec:
  controller: ingress.k8s.aws/alb
```

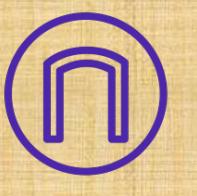
If **IngressClass** is not defined as **is-default-class:true** then in Ingress Resource we need to define the **spec.ingressClassName** in Ingress Resource

# Ingress Resource

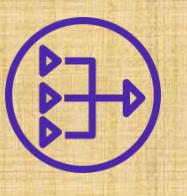
```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-nginxapp1
  labels:
    app: app1-nginx
  annotations:
    alb.ingress.kubernetes.io/load-balancer-name: app1ingress
    alb.ingress.kubernetes.io/scheme: internet-facing
spec:
  ingressClassName: my-aws-ingress-class # Ingress Class
  defaultBackend:
    service:
      name: app1-nginx-nodeport-service
      port:
        number: 80
```



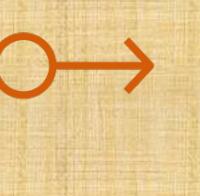
VPC



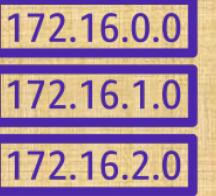
Internet gateway



NAT gateway



Elastic IP address



Route table



Public Subnet



Private Subnet



AWS S3

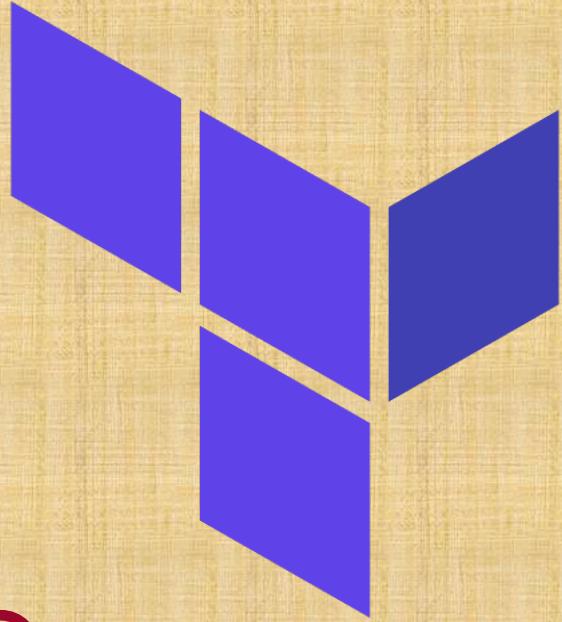


DynamoDB

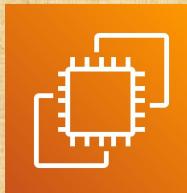
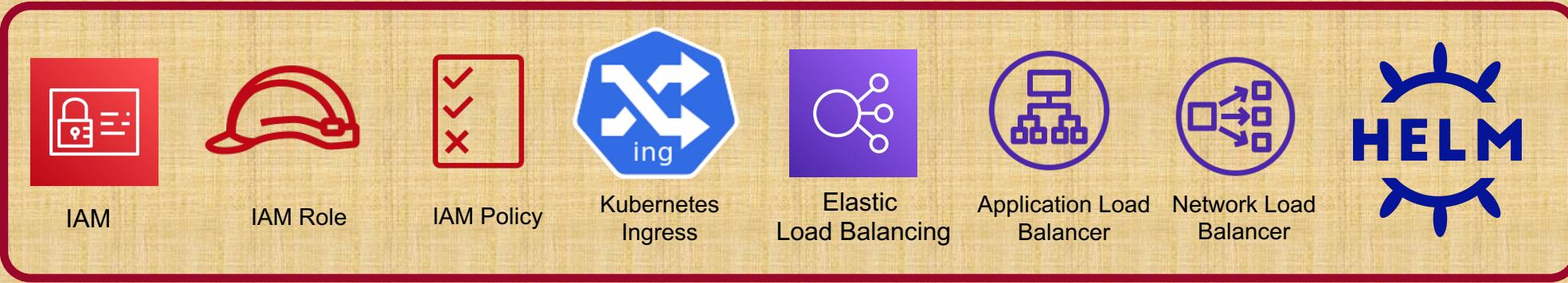


# AWS EKS

## Load Balancer Controller



EKS Cluster



EC2 VM



Autoscaling

Kubernetes Ingress Basics  
Automate with Terraform

# Ingress Manifest – Key Items

**Ingress Annotations**  
(Load Balancer Settings)

**Ingress Class Name**  
(Which Ingress Controller to use)

**Ingress Spec**  
(Define Ingress Routing Rules, Default Backend)

**Annotations Reference:** <https://kubernetes-sigs.github.io/aws-load-balancer-controller/v2.4/guide/ingress/annotations/#annotations>



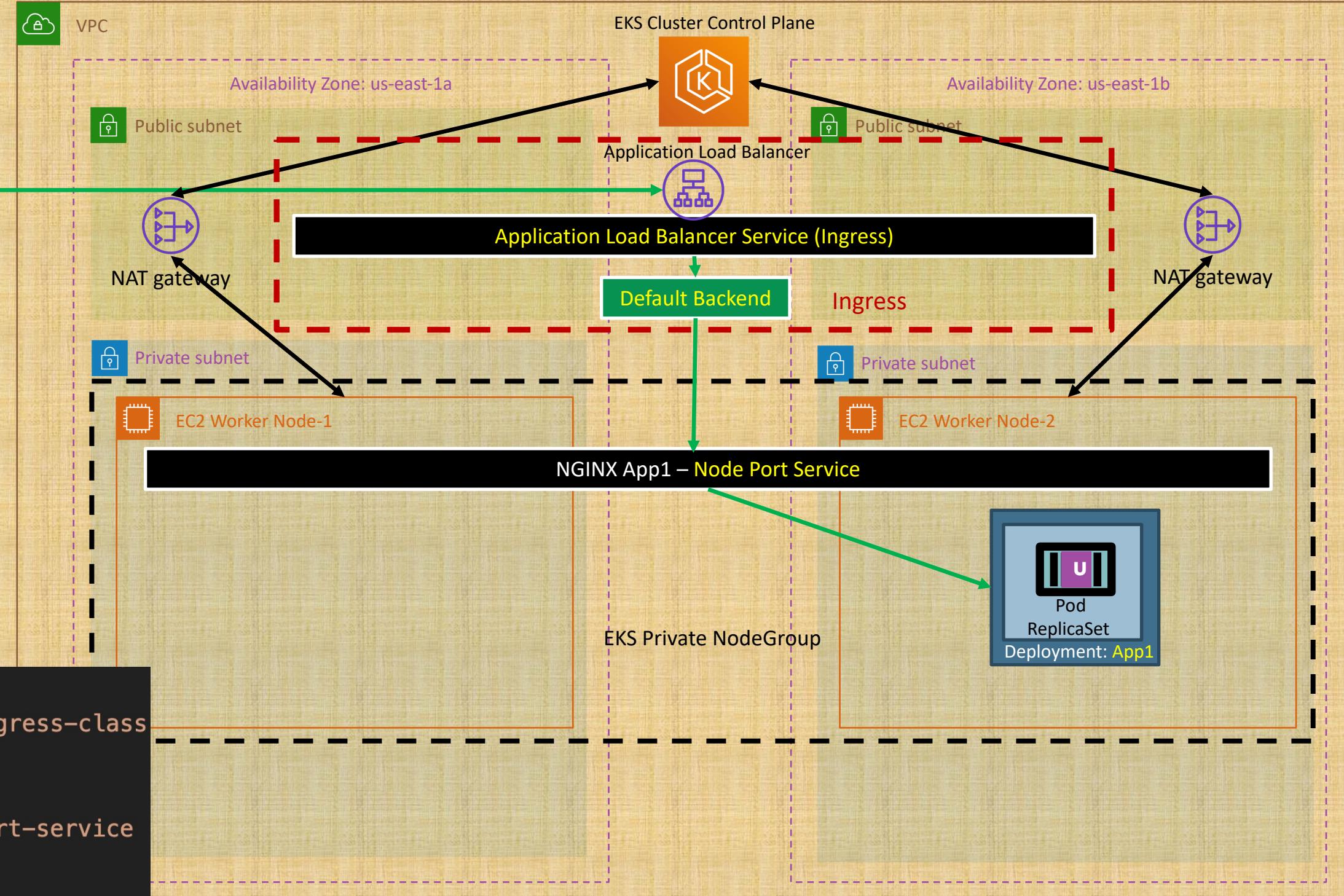
Users

http://ALB-DNS-URL

## AWS EKS Network Design With EKS Workload Application Load Balancer

**Default Backend**

```
spec:
  ingressClassName: my-aws-ingress-class
  defaultBackend:
    service:
      name: app1-nginx-nodeport-service
      port:
        number: 80
```



# What are we going to learn ?

- ✓ 27-EKS-Ingress-Basics
  - > 01-ekscluster-terraform-manifests
  - > 02-lbc-install-terraform-manifests
  - > 03-kube-manifests-ingress-basics
  - > 04-ingress-basics-terraform-manifests

- ✓ 27-EKS-Ingress-Basics
  - > 01-ekscluster-terraform-manifests
  - > 02-lbc-install-terraform-manifests
  - ✓ 03-kube-manifests-ingress-basics
    - ! 01-Nginx-App1-Deployment-and-NodePortService.yml
    - ! 02-ALB-Ingress-Basic.yml
  - ✓ 04-ingress-basics-terraform-manifests
    - ⚡ c1-versions.tf
    - ⚡ c2-remote-state-datasource.tf
    - ⚡ c3-providers.tf
    - ⚡ c4-kubernetes-app3-deployment.tf
    - ⚡ c5-kubernetes-app3-nodeport-service.tf
    - ⚡ c6-kubernetes-ingress-service.tf

01

02

03

04

## Project Folders

EKS Cluster Terraform Manifests

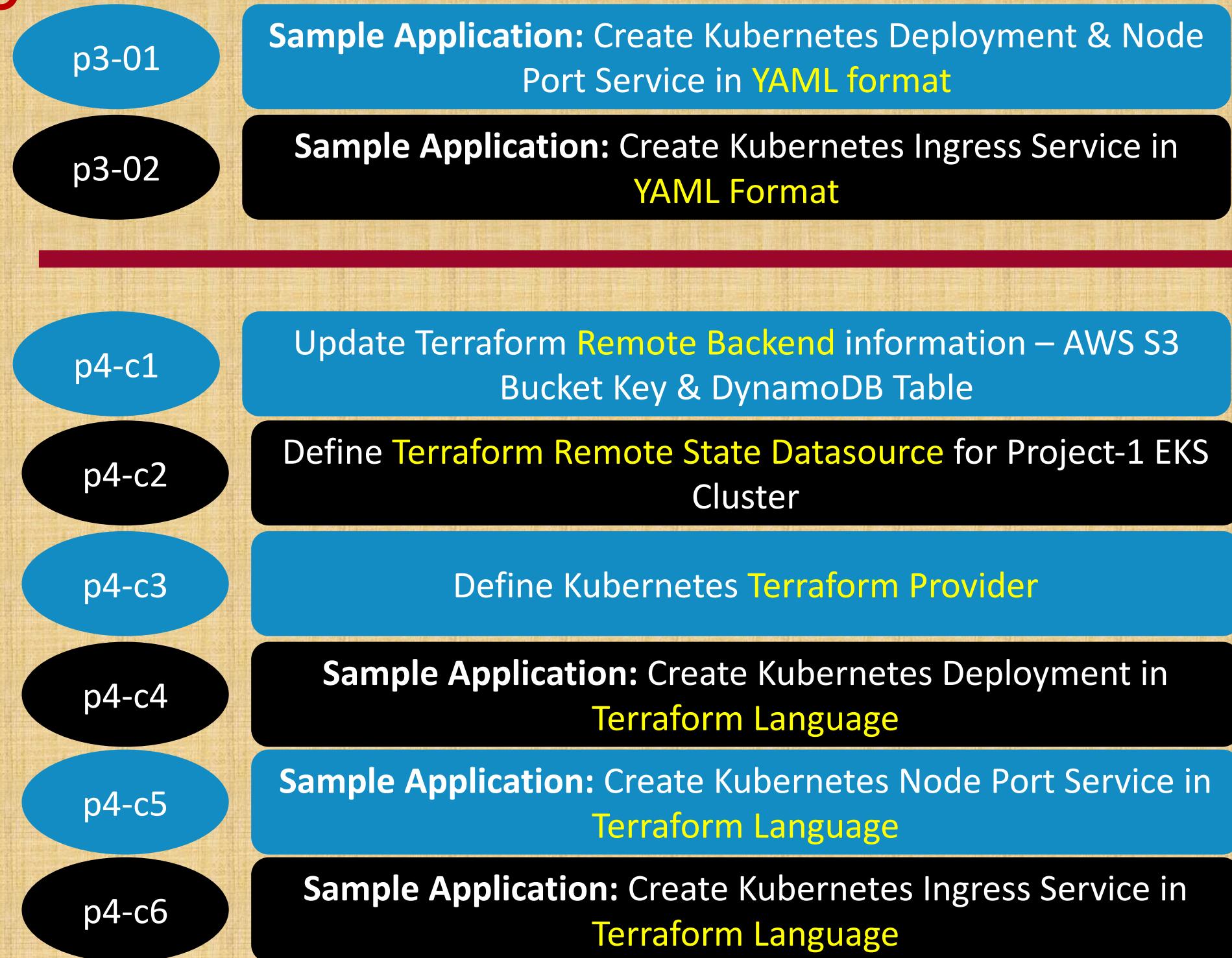
AWS Load Balancer Controller Terraform Manifests

Kubernetes Manifests in **YAML** format  
**Sample Application:** Kubernetes Deployment, Node Port Service & Ingress Service

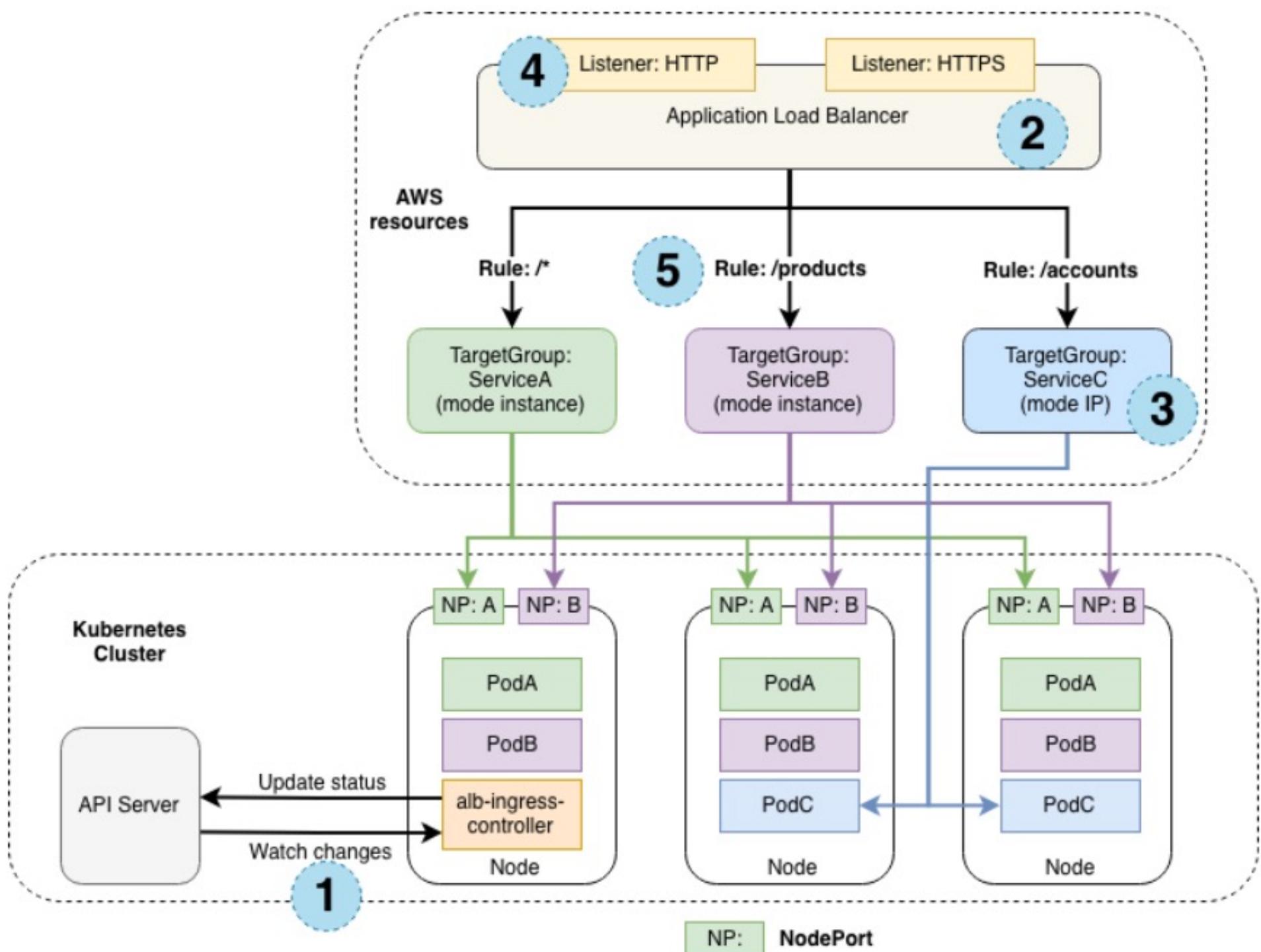
Kubernetes Manifests in **Terraform** format  
**Sample Application:** Kubernetes Deployment, Node Port Service & Ingress Service

# What are we going to learn ?

```
✓ 27-EKS-Ingress-Basics
  > 01-ekscluster-terraform-manifests
  > 02-lbc-install-terraform-manifests
✓ 03-kube-manifests-ingress-basics
  ! 01-Nginx-App1-Deployment-and-NodePortService.yml
  ! 02-ALB-Ingress-Basic.yml
✓ 04-ingress-basics-terraform-manifests
  ↘ c1-versions.tf
  ↘ c2-remote-state-datasource.tf
  ↘ c3-providers.tf
  ↘ c4-kubernetes-app3-deployment.tf
  ↘ c5-kubernetes-app3-nodeport-service.tf
  ↘ c6-kubernetes-ingress-service.tf
```



# How AWS Load Balancer controller works?



Reference: <https://kubernetes-sigs.github.io/aws-load-balancer-controller/v2.4/how-it-works/>

# What are we going to learn ?



**YAML**

Ingress Manifest – Live Template writing using **YAML** for first demo



Ingress Manifest – Live Template writing using **Terraform** for first demo

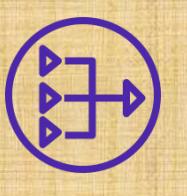
Remaining Ingress Demos are going to be just adding some code to base Ingress template either in **Annotations** section or in **Ingress Spec** section. So those we are going to **review** them during the implementation



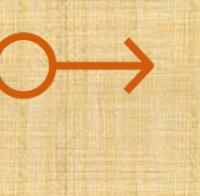
VPC



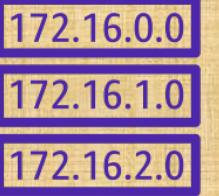
Internet gateway



NAT gateway



Elastic IP address



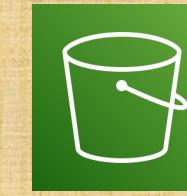
Route table



Public Subnet



Private Subnet



AWS S3

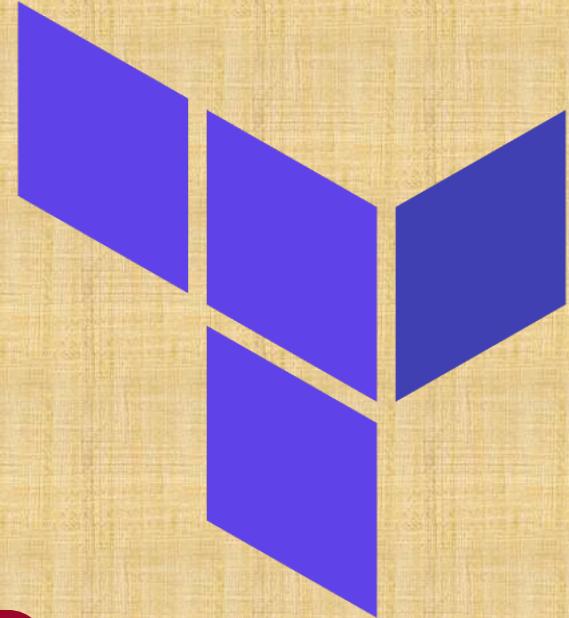


DynamoDB

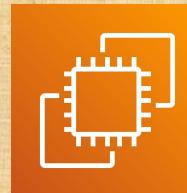


# AWS EKS

## Load Balancer Controller



EKS Cluster

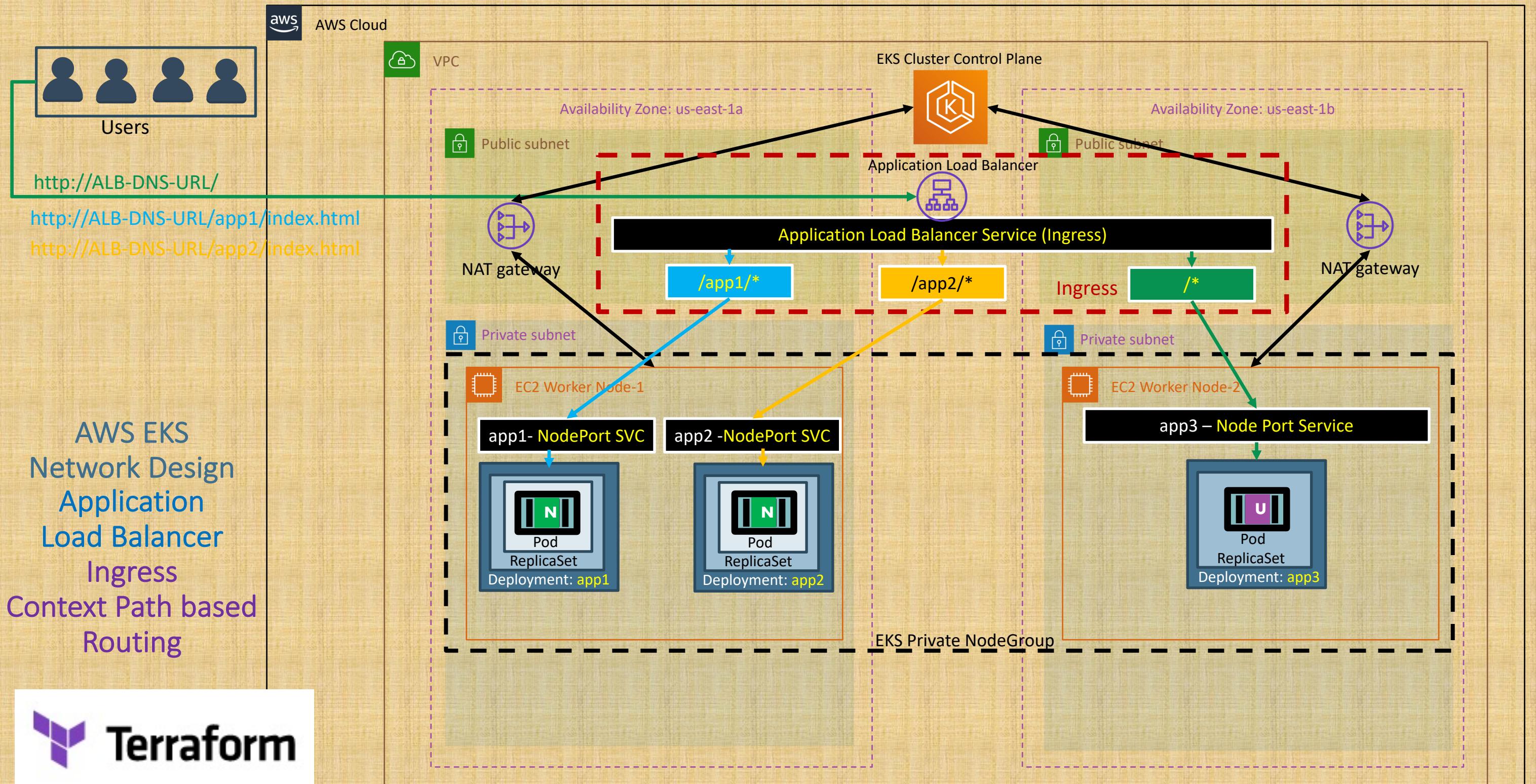


EC2 VM

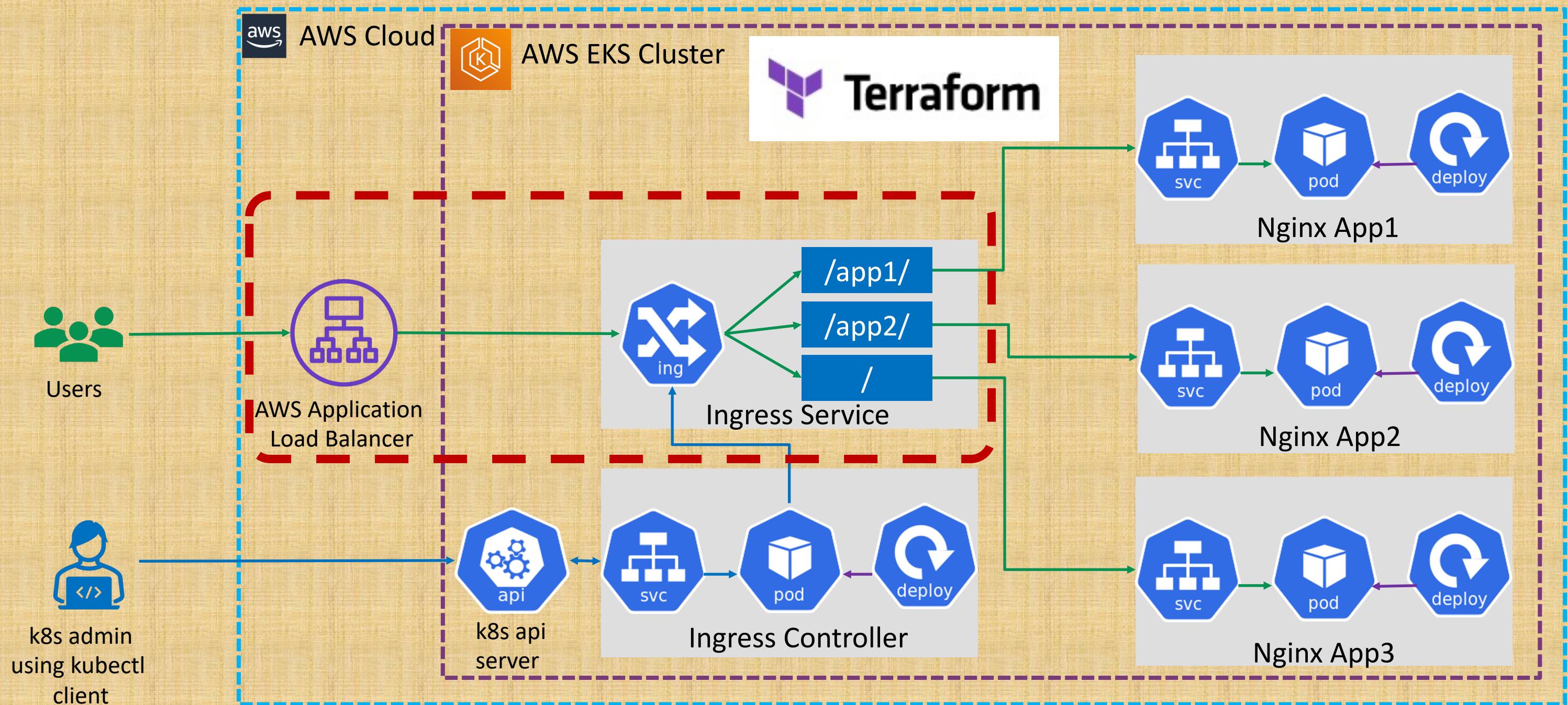


Autoscaling

Kubernetes Ingress Context Path Based Routing  
Automate with Terraform



# AWS EKS ALB Ingress – Context Path Based Routing



# What are we going to learn ?

```
✓ 28-EKS-Ingress-Context-Path-Routing
  > 01-ekscluster-terraform-manifests
  > 02-lbc-install-terraform-manifests
  ✓ 03-kube-manifests-ingress-cpr
    ! 01-Nginx-App1-Deployment-and-NodePortService.yml
    ! 02-Nginx-App2-Deployment-and-NodePortService.yml
    ! 03-Nginx-App3-Deployment-and-NodePortService.yml
    ! 04-ALB-Ingress-ContextPath-Based-Routing.yml
  ✓ 04-ingress-cpr-terraform-manifests
    ✓ c1-versions.tf
    ✓ c2-remote-state-datasource.tf
    ✓ c3-providers.tf
    ✓ c4-kubernetes-app1-deployment.tf
    ✓ c5-kubernetes-app2-deployment.tf
    ✓ c5-kubernetes-app3-deployment.tf
    ✓ c7-kubernetes-app1-nodeport-service.tf
    ✓ c8-kubernetes-app2-nodeport-service.tf
    ✓ c9-kubernetes-app3-nodeport-service.tf
    ✓ c10-kubernetes-ingress-service.tf
```

## Project Folders

01

EKS Cluster Terraform Manifests

02

AWS Load Balancer Controller Terraform Manifests

03

Kubernetes Manifests in **YAML** format  
**Sample Applications (3 Apps):** Kubernetes Deployment, Node Port Service & Ingress Service

04

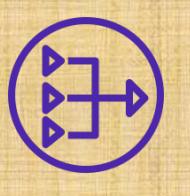
Kubernetes Manifests in **Terraform** format  
**Sample Applications (3 Apps):** Kubernetes Deployment, Node Port Service & Ingress Service



VPC



Internet gateway



NAT gateway



Elastic IP address



Route table



Public Subnet



Private Subnet



AWS S3

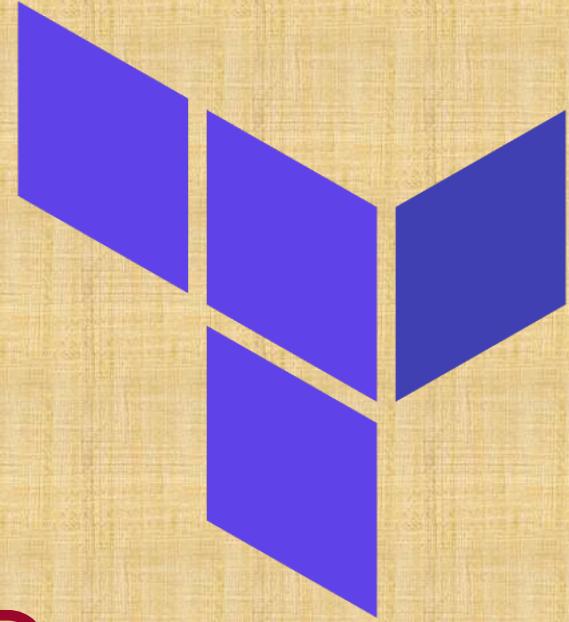


DynamoDB

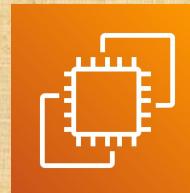
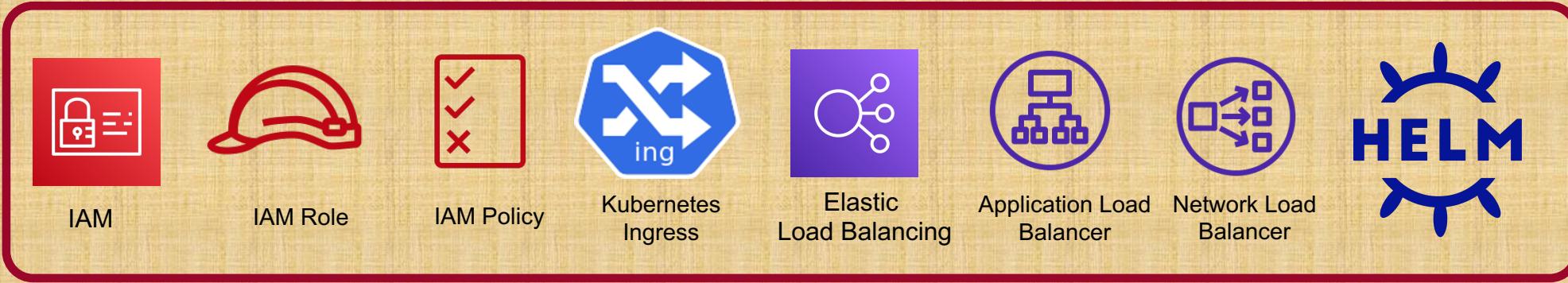


# AWS EKS

## Load Balancer Controller



EKS Cluster



EC2 VM



Autoscaling

Kubernetes Ingress SSL & SSL Redirect  
Automate with Terraform

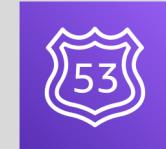
# Ingress SSL & SSL Redirect

**Task-1:** Register AWS Route53 DNS Domain

**Task-2:** Create SSL Certificate in AWS Certificate Manager

**Task-3:** Update SSL & SSL Redirect Annotations in Ingress Service

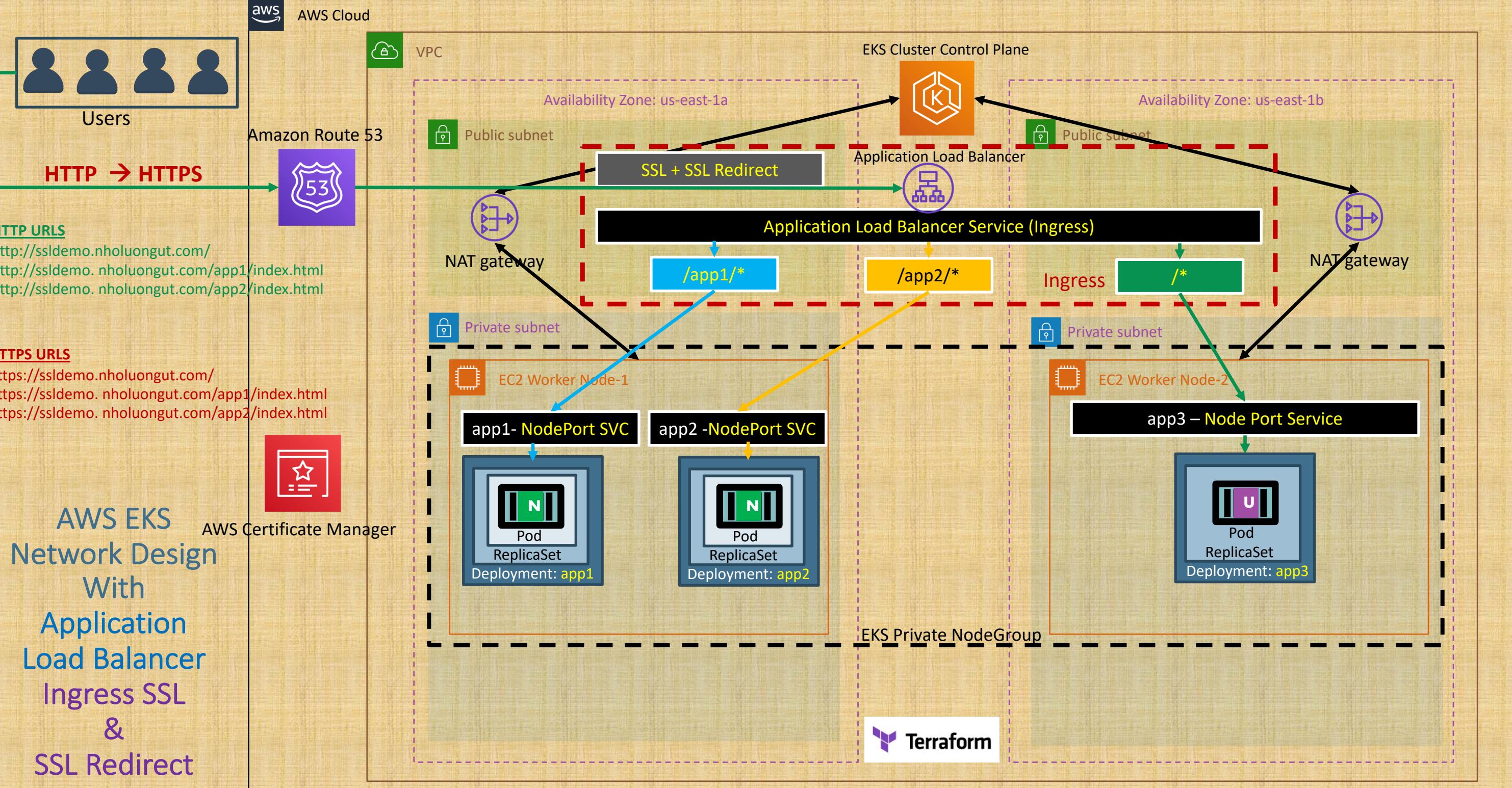
**Task-4:** Deploy k8s manifests and Test



AWS  
Route 53



AWS  
Certificate Manager



# What are we going to learn ?

```
✓ 29-EKS-Ingress-SSL-SSLRedirect
  > 01-ekscluster-terraform-manifests
  > 02-lbc-install-terraform-manifests
  ✓ 03-kube-manifests-Ingress-SSL
    ! 01-Nginx-App1-Deployment-and-NodePortService.yml
    ! 02-Nginx-App2-Deployment-and-NodePortService.yml
    ! 03-Nginx-App3-Deployment-and-NodePortService.yml
    ! 04-ALB-Ingress-SSL-Redirect.yml
  ✓ 04-ingress-ssl-terraform-manifests
    > listen-ports
    ↴ c1-versions.tf
    ↴ c2-remote-state-datasource.tf
    ↴ c3-providers.tf
    ↴ c4-kubernetes-app1-deployment.tf
    ↴ c5-kubernetes-app2-deployment.tf
    ↴ c5-kubernetes-app3-deployment.tf
    ↴ c7-kubernetes-app1-nodeport-service.tf
    ↴ c8-kubernetes-app2-nodeport-service.tf
    ↴ c9-kubernetes-app3-nodeport-service.tf
    ↴ c10-kubernetes-ingress-service.tf
    ↴ c11-acm-certificate.tf
```

## Project Folders

01

EKS Cluster Terraform Manifests

02

AWS Load Balancer Controller Terraform Manifests

03

Kubernetes Manifests in **YAML** format  
**Sample Applications (3 Apps)**: Kubernetes Deployment, Node Port Service & Ingress Service

04

Kubernetes Manifests in **Terraform** format  
**Sample Applications (3 Apps)**: Kubernetes Deployment, Node Port Service & Ingress Service

p4-c11

SSL Certificate created using **aws\_acm\_certificate** resource in Terraform

# Ingress Annotations

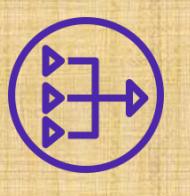
```
## SSL Settings
# Option-1: Using Terraform jsonencode Function
"alb.ingress.kubernetes.io/listen-ports" = jsonencode([{"HTTPS" = 443}, {"HTTP" = 80}])
# Option-2: Using Terraform File Function
#"alb.ingress.kubernetes.io/listen-ports" = file("${path.module}/listen-ports/listen-ports.json")
"alb.ingress.kubernetes.io/certificate-arn" = "${aws_acm_certificate.acm_cert.arn}"
#"alb.ingress.kubernetes.io/ssl-policy" = "ELBSecurityPolicy-TLS-1-1-2017-01" #Optional (Picks defa
# SSL Redirect Setting
"alb.ingress.kubernetes.io/ssl-redirect" = 443
```



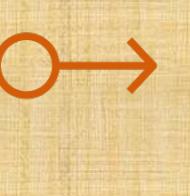
VPC



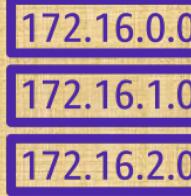
Internet gateway



NAT gateway



Elastic IP address



Route table



Public Subnet



Private Subnet



AWS S3

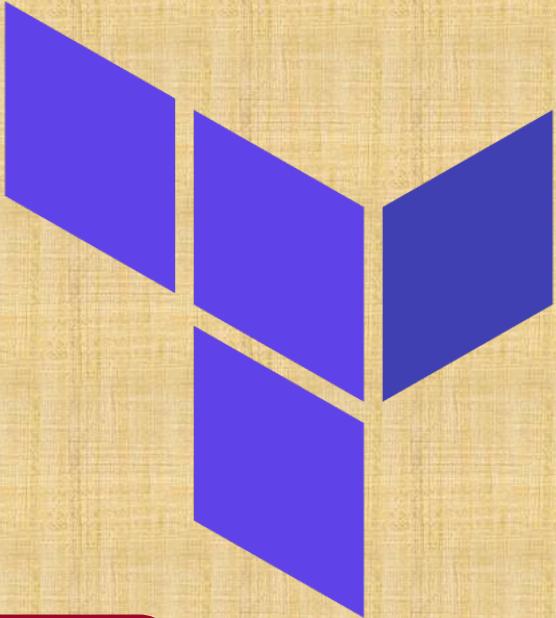


DynamoDB

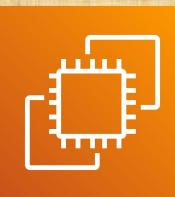
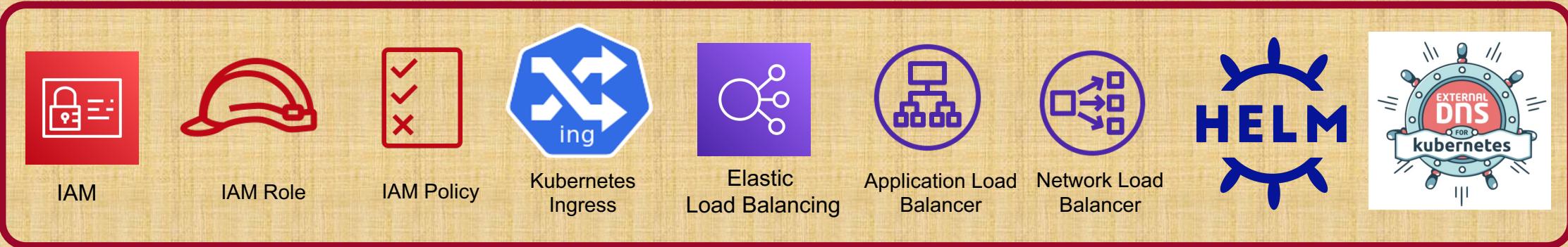


# AWS EKS

## Load Balancer Controller



EKS Cluster



EC2 VM



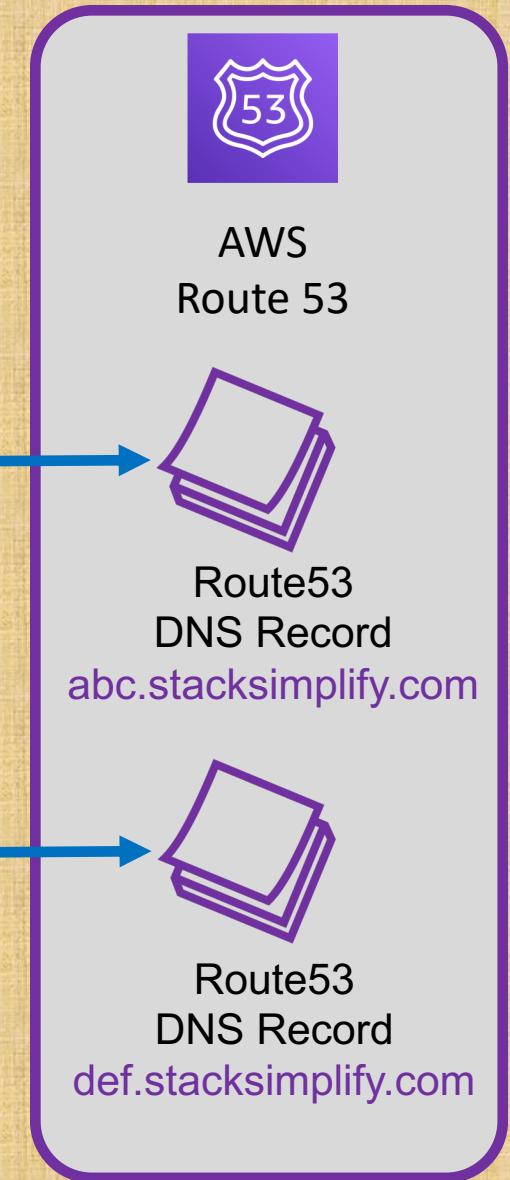
Autoscaling

**Kubernetes External DNS Install  
Automate with Terraform**

# Kubernetes External DNS

In previous SSL Demos, we added AWS Route53 DNS Record manually (ssldemo101.stack simplify.com)

With External DNS we can automatically add it for a Kubernetes Ingress Service or Kubernetes Service by defining it as Annotation

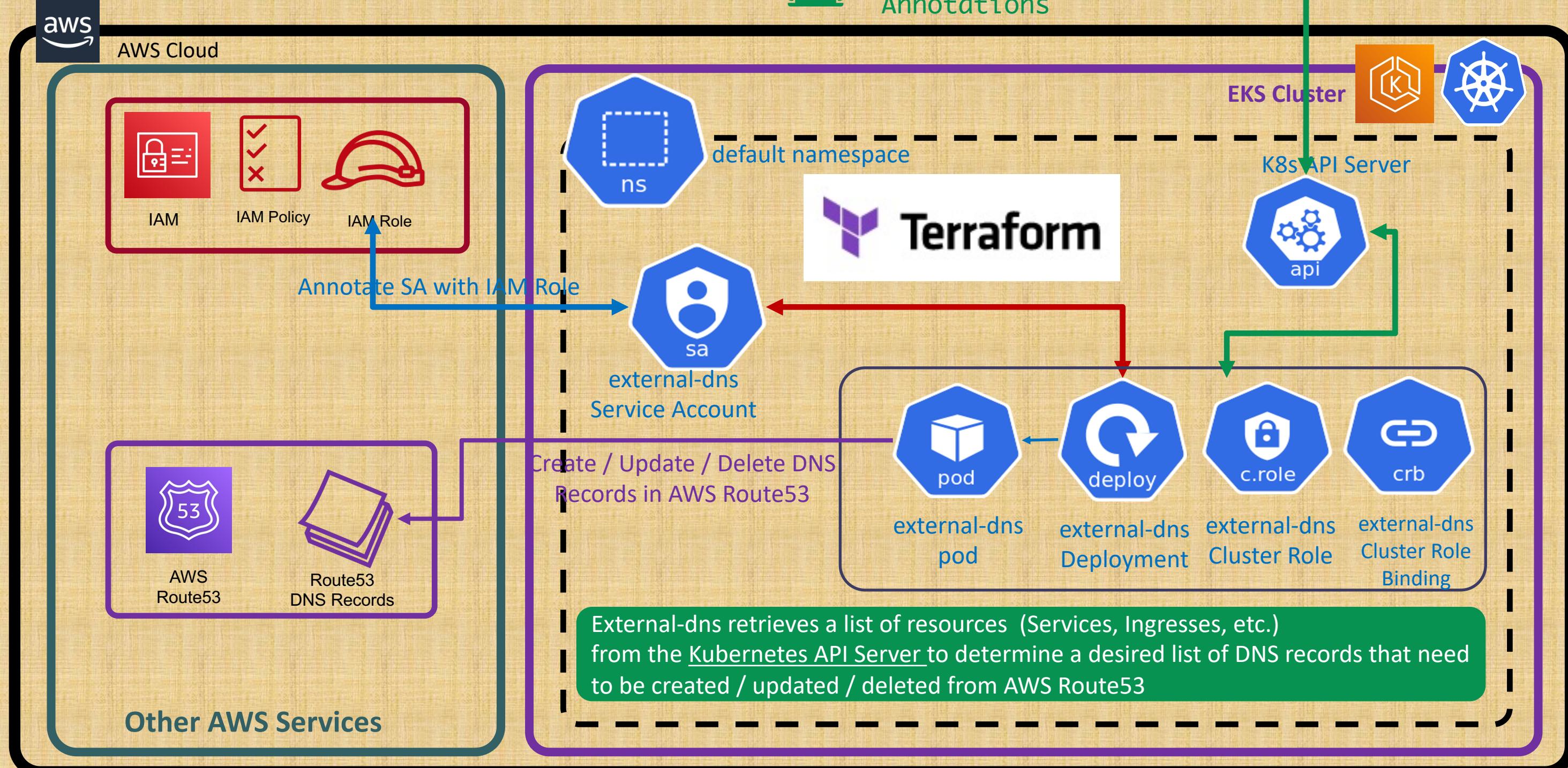


```
# External DNS – For creating a Record Set in Route53  
external-dns.alpha.kubernetes.io/hostname: dnstest901.
```

# AWS EKS + External DNS



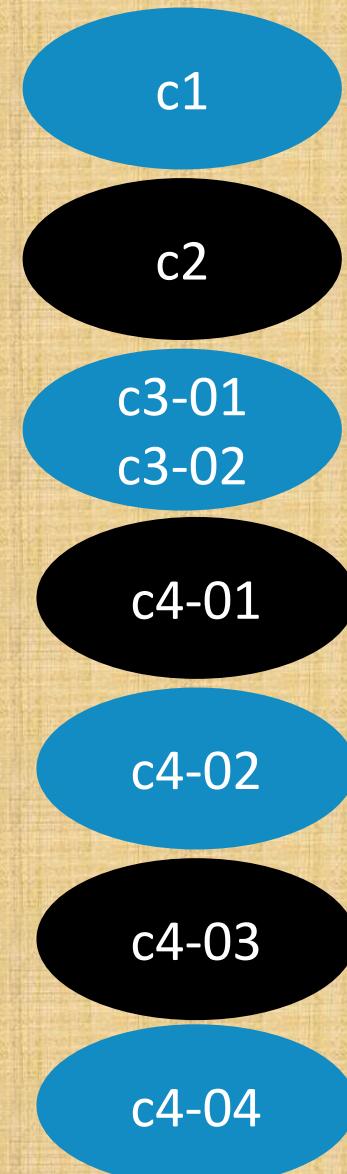
Deploy Ingress /service k8s Manifest with external-dns Annotations



# What are we going to learn ?

## Terraform Manifests

- ✓ 30-EKS-ExternalDNS-Install
  - > 01-ekscluster-terraform-manifests
  - > 02-lbc-install-terraform-manifests
  - ✓ 03-externaldns-install-terraform-manifests
    - ⚡ c1-versions.tf
    - ⚡ c2-remote-state-datasource.tf
    - ⚡ c3-01-generic-variables.tf
    - ⚡ c3-02-local-values.tf
    - ⚡ c4-01-externaldns-iam-policy-and-role.tf
    - ⚡ c4-02-externaldns-helm-provider.tf
    - ⚡ c4-03-externaldns-install.tf
    - ⚡ c4-04-externaldns-outputs.tf
    - ⚡ terraform.tfvars



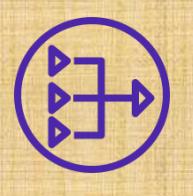
- Update Terraform **Remote Backend** information – AWS S3 Bucket Key & DynamoDB Table
- Define Terraform **Remote State Datasource** for Project-1 EKS Cluster
- Define generic **variables** and **local values**
- Create IAM Policy and IAM Role with **AssumeRoleWithWebIdentity**
- Define **Helm** provider
- Install External DNS using **Helm Resource** on EKS Cluster
- Create External DNS **Outputs**



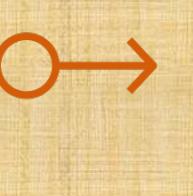
VPC



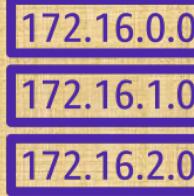
Internet gateway



NAT gateway



Elastic IP address



Route table



Public Subnet



Private Subnet



AWS S3

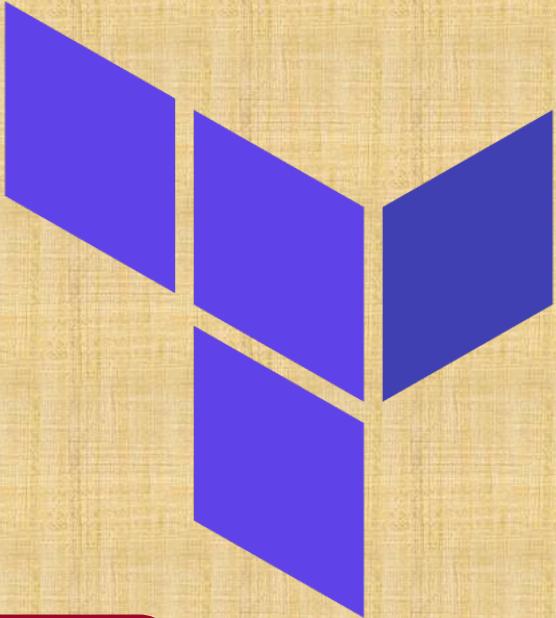


DynamoDB

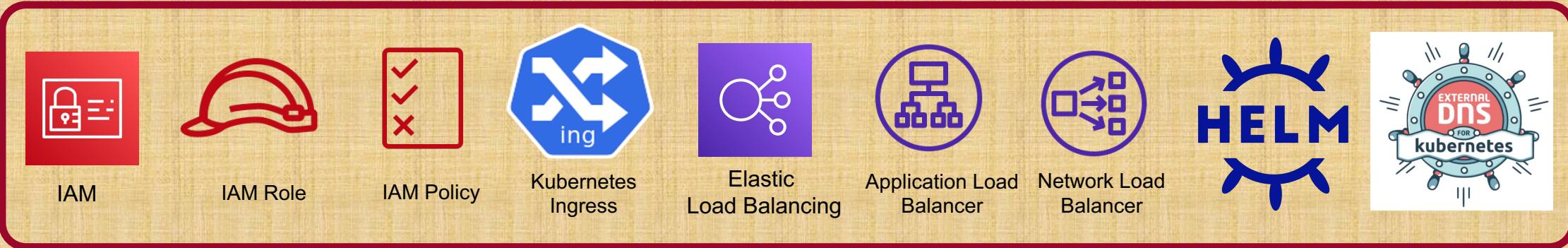


# AWS EKS

## Load Balancer Controller

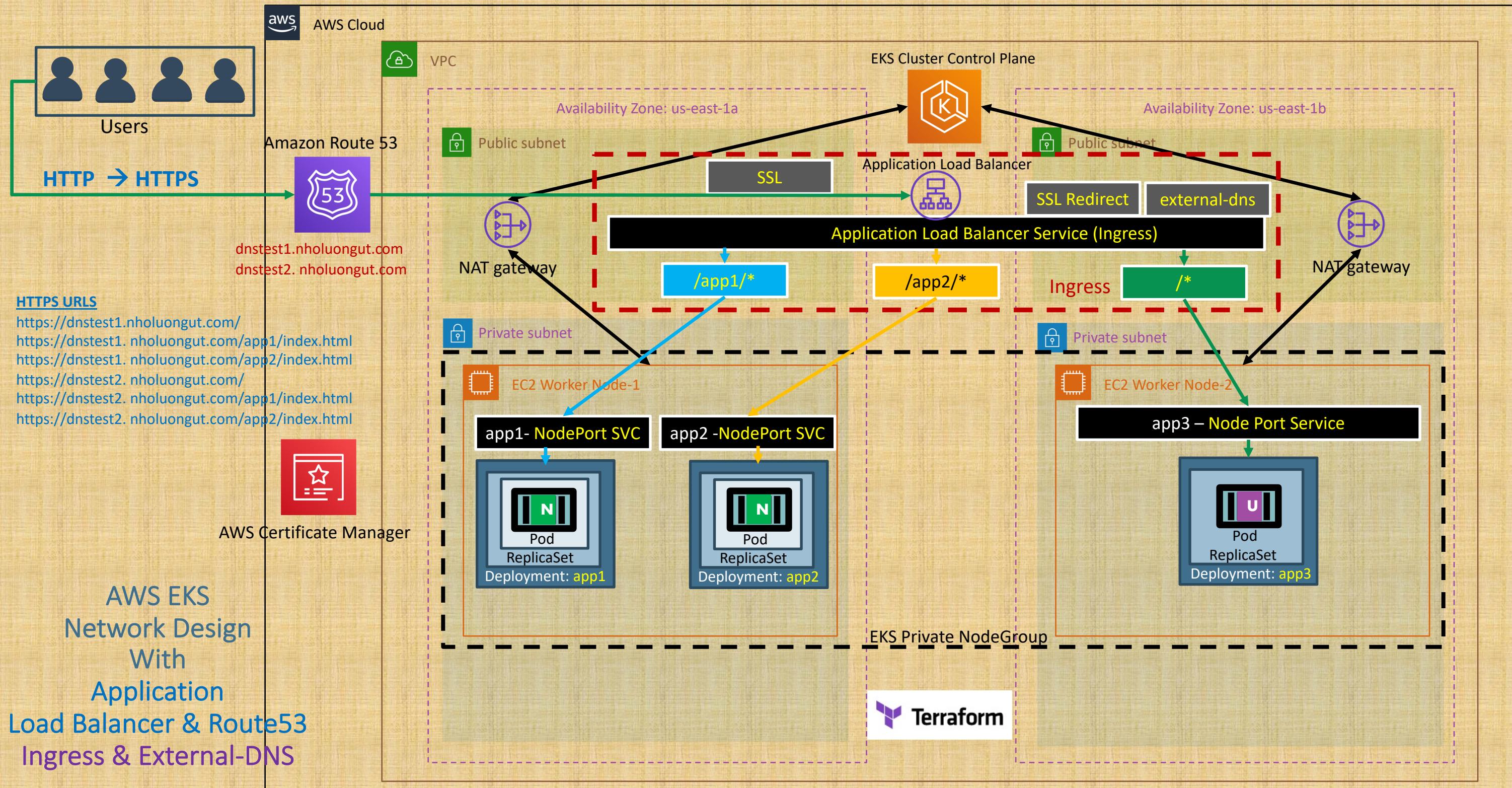


EKS Cluster



Autoscaling

**Kubernetes External DNS with Ingress Service  
Automate with Terraform**



# What are we going to learn ?

- ✓ 31-EKS-ExternalDNS-with-Ingress-Service
  - > 01-ekscluster-terraform-manifests
  - > 02-lbc-install-terraform-manifests
  - > 03-externaldns-install-terraform-manifests
- ✓ 04-kube-manifests-ingress-externaldns
  - ! 01-Nginx-App1-Deployment-and-NodePortService.yml
  - ! 02-Nginx-App2-Deployment-and-NodePortService.yml
  - ! 03-Nginx-App3-Deployment-and-NodePortService.yml
  - ! 04-ALB-Ingress-SSL-Redirect-ExternalDNS.yml
- ✓ 05-ingress-externaldns-terraform-manifests
  - > listen-ports
  - ↳ c1-versions.tf
  - ↳ c2-remote-state-datasource.tf
  - ↳ c3-providers.tf
  - ↳ c4-kubernetes-app1-deployment.tf
  - ↳ c5-kubernetes-app2-deployment.tf
  - ↳ c5-kubernetes-app3-deployment.tf
  - ↳ c7-kubernetes-app1-nodeport-service.tf
  - ↳ c8-kubernetes-app2-nodeport-service.tf
  - ↳ c9-kubernetes-app3-nodeport-service.tf
  - ↳ c10-kubernetes-ingress-service.tf
  - ↳ c11-acm-certificate.tf

## Project Folders

01

EKS Cluster Terraform Manifests

02

AWS Load Balancer Controller Terraform Manifests

03

External DNS Install Terraform Manifests

04

Kubernetes Manifests in **YAML** format  
**Sample Applications (3 Apps):** Kubernetes Deployment, Node Port Service & Ingress Service

05

Kubernetes Manifests in **Terraform** format  
**Sample Applications (3 Apps):** Kubernetes Deployment, Node Port Service & Ingress Service

# Ingress Annotations

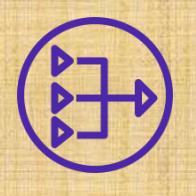
```
# External DNS – For creating a Record Set in  
"external-dns.alpha.kubernetes.io/hostname" =
```



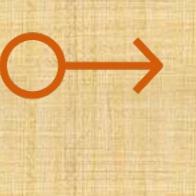
VPC



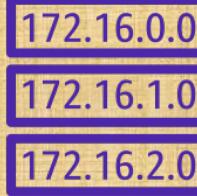
Internet gateway



NAT gateway



Elastic IP address



Route table



Public Subnet



Private Subnet



AWS S3

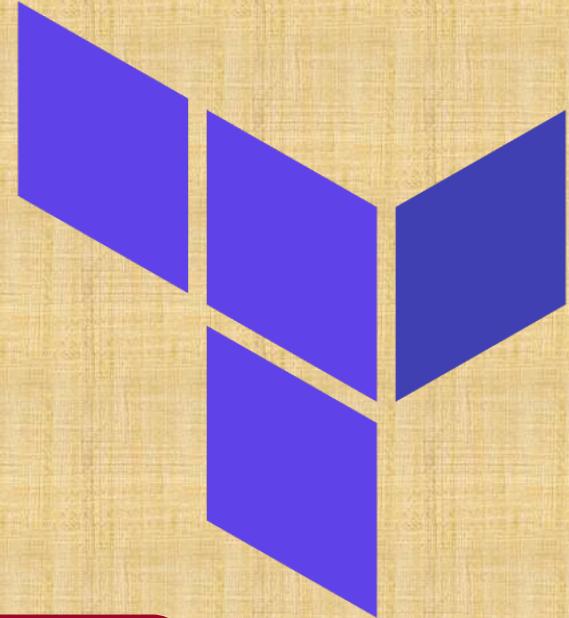


DynamoDB

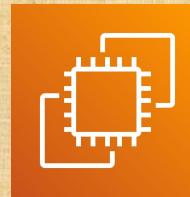


# AWS EKS

## Load Balancer Controller



EKS Cluster



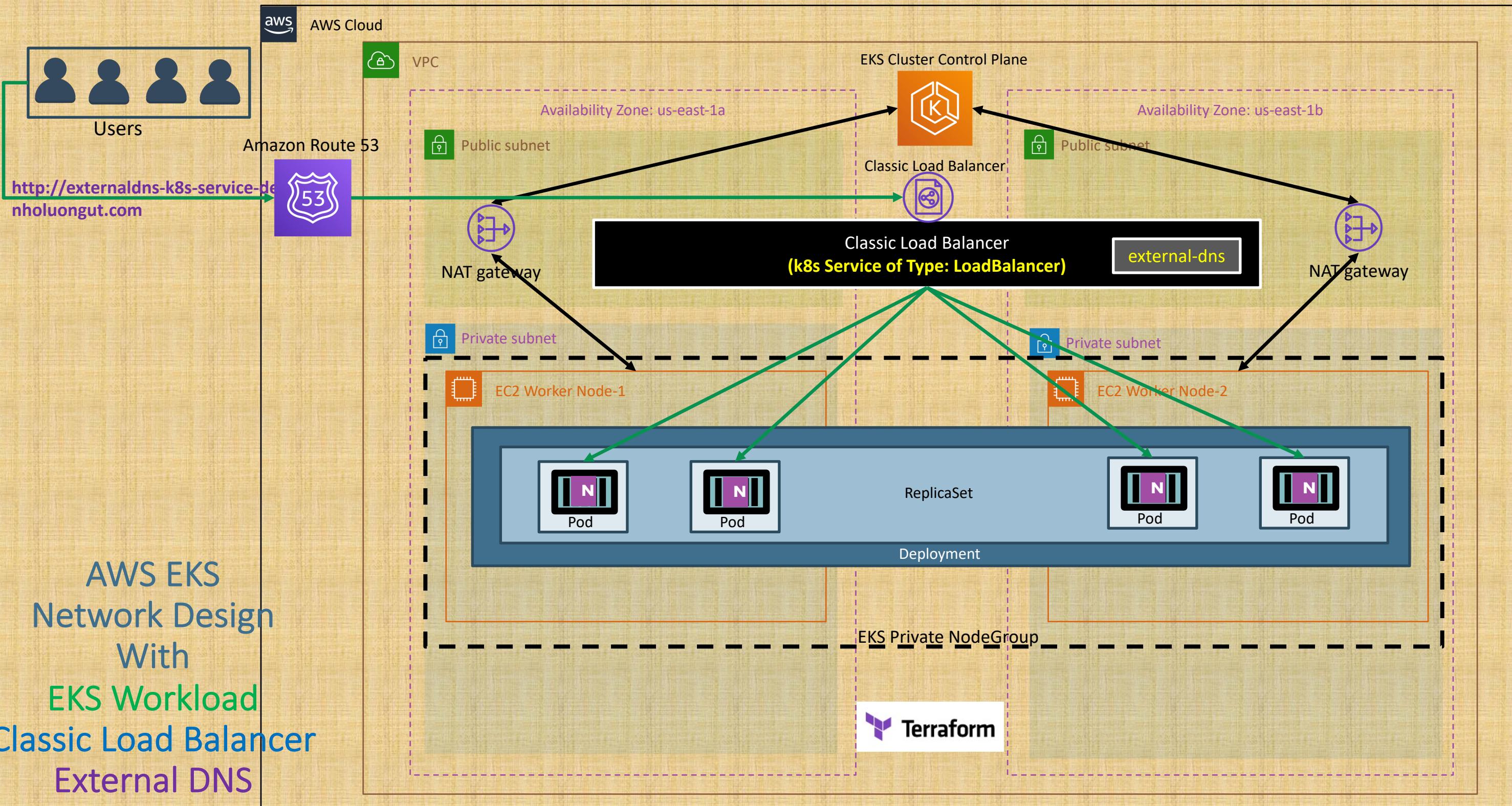
EC2 VM



Autoscaling

**Kubernetes External DNS with Kubernetes Load Balancer Service**  
**Automate with Terraform**

# AWS EKS Network Design With EKS Workload Classic Load Balancer External DNS



# Kubernetes Service with External DNS

```
apiVersion: v1
kind: Service
metadata:
  name: app1-nginx-loadbalancer-service
  labels:
    app: app1-nginx
  annotations:
    external-dns.alpha.kubernetes.io/hostname:
spec:
  type: LoadBalancer
  selector:
    app: app1-nginx
  ports:
    - port: 80
      targetPort: 80
```

YAML  
Manifest

# Kubernetes Service with External DNS

```
# Kubernetes Service Manifest (Type: Node Port Service)
resource "kubernetes_service_v1" "myapp1_np_service" {
  metadata {
    name = "app1-nginx-loadbalancer-service"
    annotations = {
      "alb.ingress.kubernetes.io/healthcheck-path" = "/app1/index.html"
      "external-dns.alpha.kubernetes.io/hostname" = "tfextdns-k8s-service-demo101."
    }
  }
  spec {
    selector = {
      app = kubernetes_deployment_v1.myapp1.spec.0.selector.0.match_labels
    }
    port {
      name        = "http"
      port        = 80
      target_port = 80
    }
    type = "LoadBalancer"
  }
}
```

Terraform  
Manifest

# What are we going to learn ?

```
✓ 32-EKS-ExternalDNS-with-k8s-Service
  > 01-ekscluster-terraform-manifests
  > 02-lbc-install-terraform-manifests
  > 03-externaldns-install-terraform-manifests
  ✓ 04-kube-manifests-k8sService-externaldns
    ! 01-Nginx-App1-Deployment.yml
    ! 02-Nginx-App1-LoadBalancer-Service.yml
  ✓ 05-k8sService-externaldns-terraform-manifests
    ✓ c1-versions.tf
    ✓ c2-remote-state-datasource.tf
    ✓ c3-providers.tf
    ✓ c4-kubernetes-app1-deployment.tf
    ✓ c5-kubernetes-app1-loadbalancer-service.tf
```

## Project Folders

01

EKS Cluster Terraform Manifests

02

AWS Load Balancer Controller Terraform Manifests

03

External DNS Install Terraform Manifests

04

Kubernetes Manifests in **YAML** format  
**Sample Applications (1 App):** Kubernetes Deployment & Load Balancer Service

05

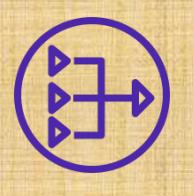
Kubernetes Manifests in **Terraform** format  
**Sample Applications (1 App):** Kubernetes Deployment & Load Balancer Service



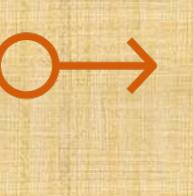
VPC



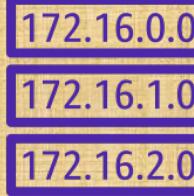
Internet gateway



NAT gateway



Elastic IP address



Route table



Public Subnet



Private Subnet



AWS S3

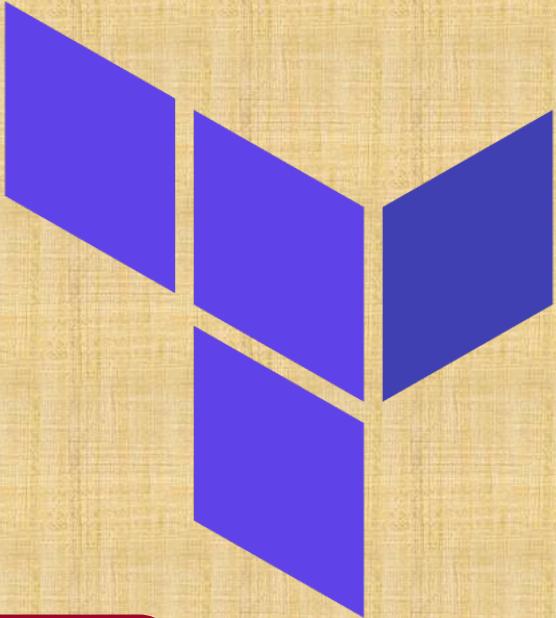


DynamoDB

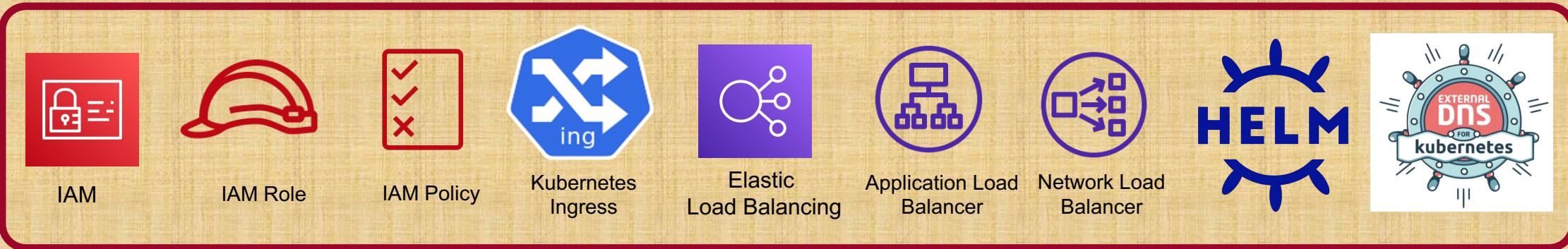


# AWS EKS

## Load Balancer Controller

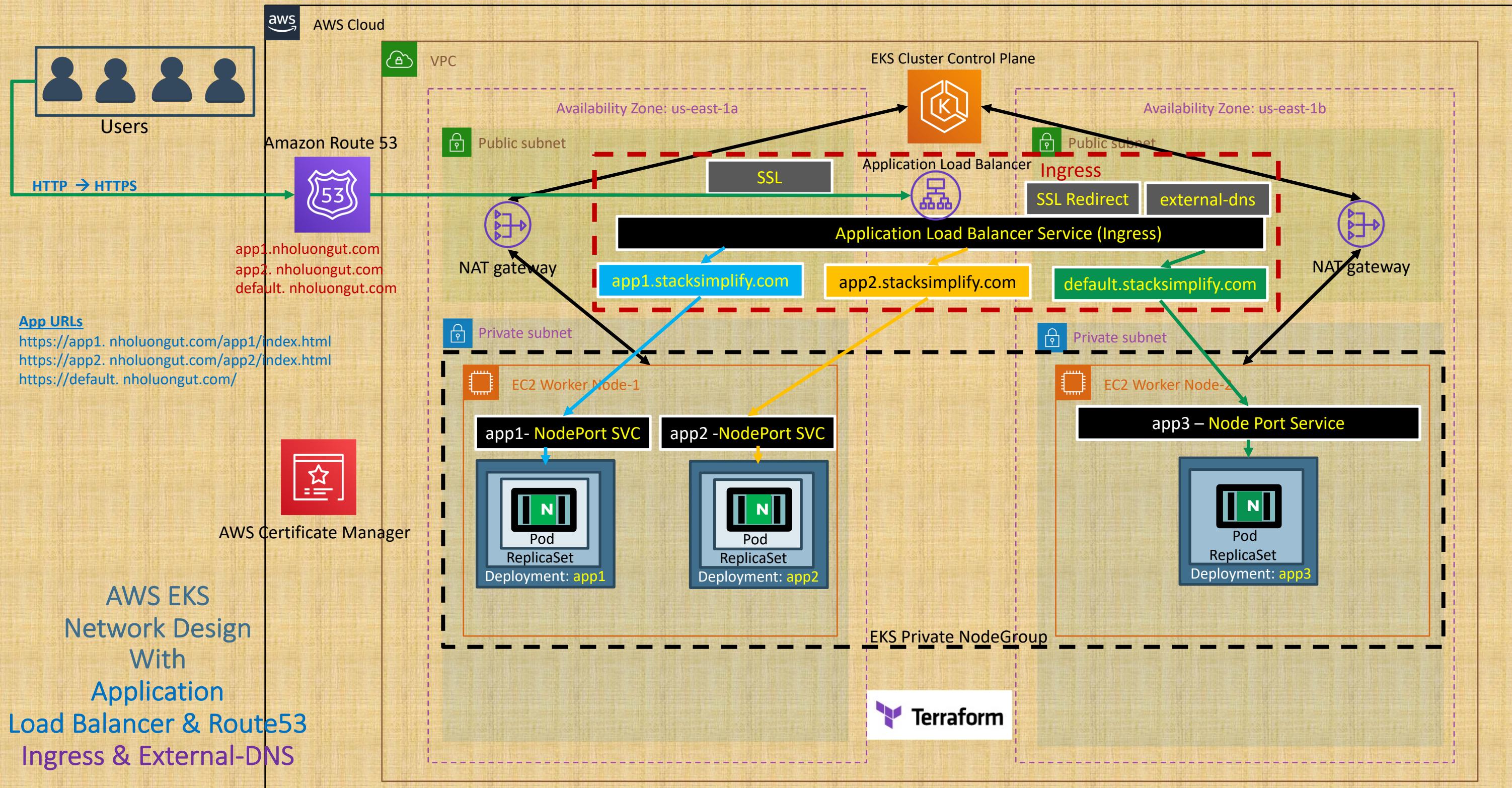


EKS Cluster



Autoscaling

**Kubernetes Ingress Name Based Virtual Host Routing**  
**Automate with Terraform**



# Ingress Name based Virtual Host Routing

```
http:  
  paths:  
    - path: /  
      pathType: Prefix  
      backend:  
        service:  
          name: app1-nginx-nodeport-service  
          port:  
            number: 80
```

```
http:  
  paths:  
    - path: /  
      pathType: Prefix  
      backend:  
        service:  
          name: app2-nginx-nodeport-service  
          port:  
            number: 80
```

Requests with **app101.nholuongut.com**  
will go to **App1 Node Port Service**

Requests with **app201.nholuongut.com**  
will go to **App2 Node Port Service**

YAML  
Manifest

# Ingress Named based Virtual Host Routing

```
# External DNS – For creating a Record Set
external-dns.alpha.kubernetes.io/hostname:←
spec:
  ingressClassName: my-aws-ingress-class    # I
  defaultBackend:
    service:
      name: app3-nginx-nodeport-service
      port:
        number: 80
```

YAML  
Manifest

Requests with **default101.nholuongut.com** will go to App3 Node Port Service (Default Backend)

# Ingress Name based Virtual Host Routing

```
http {  
  path {  
    backend {  
      service {  
        name = kubernetes_service_v1.myapp1_np_service.metadata[0].name  
        port {  
          number = 80  
        }  
      }  
    }  
    path = "/"  
    path_type = "Prefix"  
  }  
}
```

```
http {  
  path {  
    backend {  
      service {  
        name = kubernetes_service_v1.myapp3_np_service.metadata[0].name  
        port {  
          number = 80  
        }  
      }  
    }  
    path = "/"  
    path_type = "Prefix"  
  }  
}
```

Requests with **tfapp101.nholuongut.com**  
will go to **App1 Node Port Service**

Requests with **tfapp201.nholuongut.com**  
will go to **App2 Node Port Service**

Terraform  
Manifest

# Ingress Named based Virtual Host Routing

```
    "external-dns.alpha.kubernetes.io/hostname" =  
    }  
}  
  
spec {  
  ingress_class_name = "my-aws-ingress-class" # Ingress Class  
  # Default Rule: Route requests to App3 if the DNS is "tfdefault101.  
  default_backend {  
    service {  
      name = kubernetes_service_v1.myapp3_np_service.metadata[0].name  
      port {  
        number = 80  
      }  
    }  
  }  
}
```

The diagram illustrates the flow of configuration from a Terraform Manifest to an Ingress Controller. A blue oval labeled "Terraform Manifest" has a green arrow pointing upwards to a dark grey rectangular box containing the Terraform code. From this box, another green arrow points upwards to a black rectangular box containing the Ingress Controller configuration.

Terraform  
Manifest

Requests with **tfdefault101.nholuongut.com** will go to **App3 Node Port Service (Default Backend)**

# What are we going to learn ?

```
✓ 33-EKS-Ingress-NameBasedVirtualHost-Routing
  > 01-ekscluster-terraform-manifests
  > 02-lbc-install-terraform-manifests
  > 03-externaldns-install-terraform-manifests
  ✓ 04-kube-manifests-ingress-nvhr
    ! 01-Nginx-App1-Deployment-and-NodePortService.yml
    ! 02-Nginx-App2-Deployment-and-NodePortService.yml
    ! 03-Nginx-App3-Deployment-and-NodePortService.yml
    ! 04-ALB-Ingress-HostHeader-Routing.yml
  ✓ 05-ingress-nvhr-terraform-manifests
    > listen-ports
    ✓ c1-versions.tf
    ✓ c2-remote-state-datasource.tf
    ✓ c3-providers.tf
    ✓ c4-kubernetes-app1-deployment.tf
    ✓ c5-kubernetes-app2-deployment.tf
    ✓ c5-kubernetes-app3-deployment.tf
    ✓ c7-kubernetes-app1-nodeport-service.tf
    ✓ c8-kubernetes-app2-nodeport-service.tf
    ✓ c9-kubernetes-app3-nodeport-service.tf
    ✓ c10-kubernetes-ingress-service.tf
    ✓ c11-acm-certificate.tf
```

## Project Folders

01

EKS Cluster Terraform Manifests

02

AWS Load Balancer Controller Terraform Manifests

03

External DNS Install Terraform Manifests

04

Kubernetes Manifests in **YAML** format  
**Sample Applications (3 Apps)**: Kubernetes Deployment, Node Port Service & Ingress Service

05

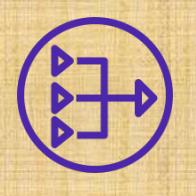
Kubernetes Manifests in **Terraform** format  
**Sample Applications (3 Apps)**: Kubernetes Deployment, Node Port Service & Ingress Service



VPC



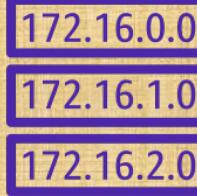
Internet gateway



NAT gateway



Elastic IP address



Route table



Public Subnet



Private Subnet



AWS S3

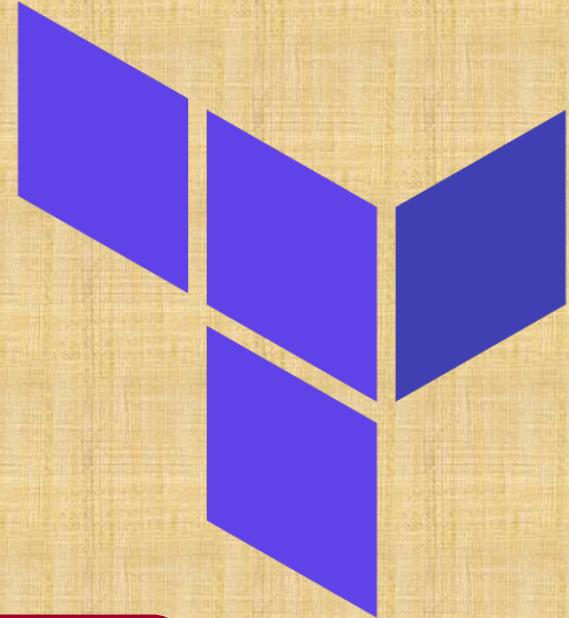


DynamoDB



# AWS EKS

## Load Balancer Controller



EKS Cluster



Autoscaling

**Kubernetes Ingress SSL Discovery Host & TLS Options**  
**Automate with Terraform**

# Ingress SSL Certificate Discovery using Host

## Demo-1: SSL Discovery – Host

```
http:  
  paths:  
    - path: /  
      pathType: Prefix  
      backend:  
        service:  
          name: app1-nginx-nodeport-service  
          port:  
            number: 80  
  
http:  
  paths:  
    - path: /  
      pathType: Prefix  
      backend:  
        service:  
          name: app2-nginx-nodeport-service  
          port:  
            number: 80
```

YAML  
Manifest

TLS certificates for ALB Listeners can be automatically discovered with **hostnames** from Ingress resources if the **alb.ingress.kubernetes.io/certificate-arn** annotation is not specified.

The controller will attempt to discover **TLS certificates** from the **host field** in Ingress rules ([Example: Name based Virtual Host Demo](#)) and **tls field** in Ingress Spec ([Example: Context Path based Routing Demo](#)).

# Ingress SSL Certificate Discovery using Host

```
http {
  path {
    backend {
      service {
        name = kubernetes_service_v1.myapp1_np_service.metadata[0].name
        port {
          number = 80
        }
      }
    }
    path = "/"
    path_type = "Prefix"
  }
}
```

```
http {
  path {
    backend {
      service {
        name = kubernetes_service_v1.myapp3_np_service.metadata[0].name
        port {
          number = 80
        }
      }
    }
    path = "/"
    path_type = "Prefix"
  }
}
```

```
## SSL Settings
# Option-1: Using Terraform jsonencode Function
"alb.ingress.kubernetes.io/listen-ports" = jsonencode([{"HTTPS" = 443}, {"HTTP" = 80}])
# Option-2: Using Terraform File Function
#"alb.ingress.kubernetes.io/listen-ports" = file("${path.module}/listen-ports/listen-ports.json")
#"alb.ingress.kubernetes.io/certificate-arn" = "${aws_acm_certificate.acm_cert.arn}"
#"alb.ingress.kubernetes.io/ssl-policy" = "ELBSecurityPolicy-TLS-1-1-2017-01" #Optional (Picks defa
# SSL Redirect Setting
"alb.ingress.kubernetes.io/ssl-redirect" = 443
```

Certificate ARN Commented

Terraform Manifest

# Ingress SSL Certificate Discovery using tls

```
spec:  
  ingressClassName: my-aws-ingress-class  # Ingress Class  
  defaultBackend:  
    service:  
      name: app3-nginx-nodeport-service  
      port:  
        number: 80  
  
rules:  
  - http:  
    paths:  
      - path: /app1  
        pathType: Prefix  
        backend:  
          service:  
            name: app1-nginx-nodeport-service
```

Demo-2: SSL  
Discovery - TLS

YAML  
Manifest

# Ingress SSL Certificate Discovery using TLS

```
"alb.ingress.kubernetes.io/listen-ports" = jsonencode([{"HTTPS" = 443}, {"HTTP" = 80}])  
# Option-2: Using Terraform File Function  
#"alb.ingress.kubernetes.io/listen-ports" = file("${path.module}/listen-ports/listen-ports.json")  
#"alb.ingress.kubernetes.io/certificate-arn" = "${aws_acm_certificate.acm_cert.arn}"  
#"alb.ingress.kubernetes.io/ssl-policy" = "ELBSecurityPolicy-TLS-1-1-2017-01" #Optional  
# SSL Redirect Setting  
"alb.ingress.kubernetes.io/ssl-redirect" = 443  
  
}  
spec {  
    ingress_class_name = "my-aws-ingress-class" # Ingress Class  
    default_backend {  
        service {  
            name = kubernetes_service_v1.myapp3_np_service.metadata[0].name  
            port {  
                number = 80  
            }  
        }  
    }  
}
```

Certificate ARN  
Commented

Terraform  
Manifest

Demo-2:  
SSL Discovery - TLS

# What are we going to learn ?

- ✓ 34-EKS-Ingress-SSLDiscovery-Host
  - > 01-ekscluster-terraform-manifests
  - > 02-lbc-install-terraform-manifests
  - > 03-externaldns-install-terraform-manifests
  - > 04-kube-manifests-SSLDiscoveryHost
    - NVHR Demo
  - > 05-ingress-SSLDiscoveryHost-terraform-manifests

(i) README.md

- ✓ 35-EKS-Ingress-SSLDiscovery-TLS
  - > 01-ekscluster-terraform-manifests
  - > 02-lbc-install-terraform-manifests
  - > 03-externaldns-install-terraform-manifests
  - > 04-kube-manifests-SSLDiscoveryTLS
    - CPR Demo
  - > 05-ingress-SSLDiscoveryTLS-terraform-manifests

(i) README.md

## Project Folders

01

EKS Cluster Terraform Manifests

02

AWS Load Balancer Controller Terraform Manifests

03

External DNS Install Terraform Manifests

04

Kubernetes Manifests in **YAML** format  
**Sample Applications (3 Apps)**: Kubernetes Deployment, Node Port Service & Ingress Service

05

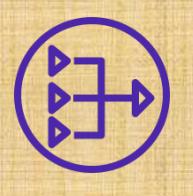
Kubernetes Manifests in **Terraform** format  
**Sample Applications (3 Apps)**: Kubernetes Deployment, Node Port Service & Ingress Service



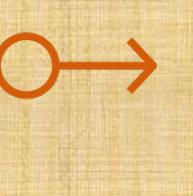
VPC



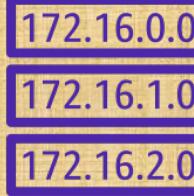
Internet gateway



NAT gateway



Elastic IP address



Route table



Public Subnet



Private Subnet



AWS S3

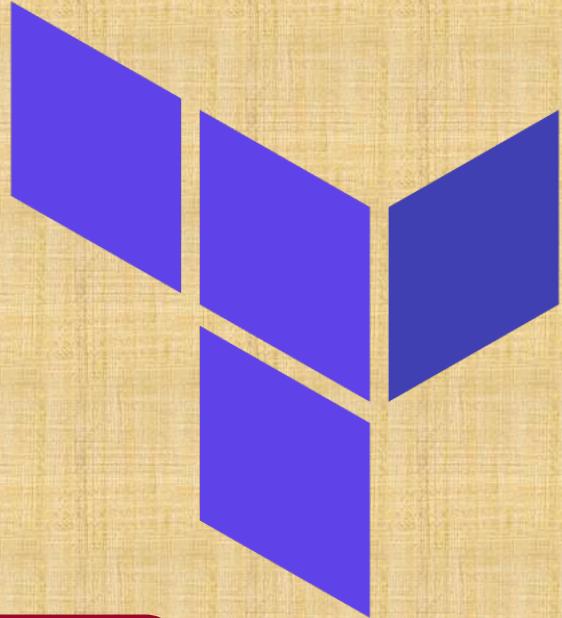


DynamoDB

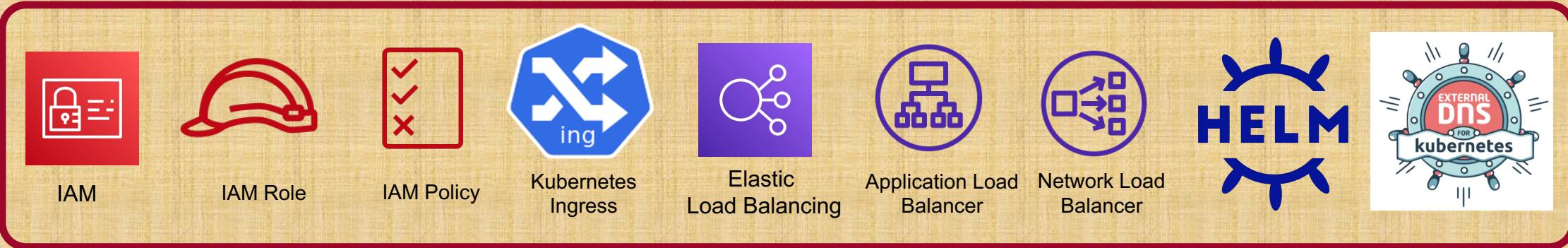


# AWS EKS

## Load Balancer Controller



EKS Cluster



Autoscaling

**Kubernetes Ingress Groups**  
Automate with Terraform

# Without Ingress Groups

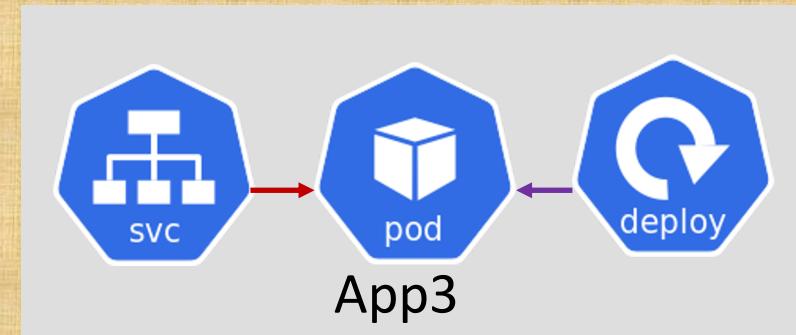
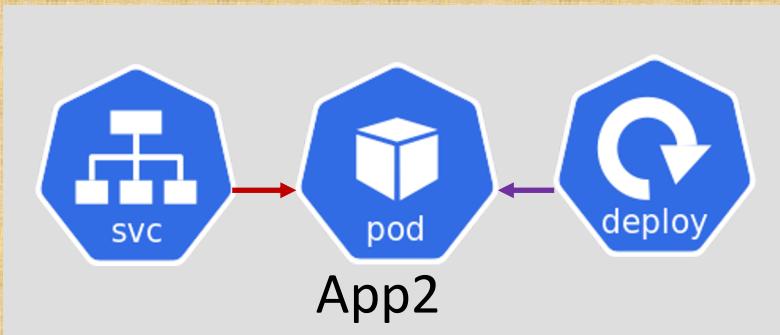
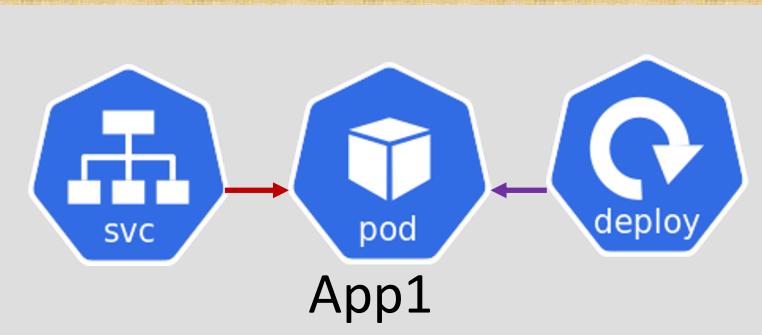
Single Ingress Resource for all three Apps

**Example:** If we have 50 apps which needs different rules in them and all should be part of single ALB, maintaining such huge k8s manifest file with configs becomes tedious and confusing.



Any changes to App3 we need to update the “my-apps” ingress resource

With **Ingress Groups** we can simplify our Kubernetes Ingress Manifest when dealing with **multiple apps** requiring single ALB



# With Ingress Groups

alb.ingress.kubernetes.io/load-balancer-name: ingress-groups-demo

Ingress Group: myapps.web

Priority: 10



Ingress Service: app1-ingress

Priority: 20

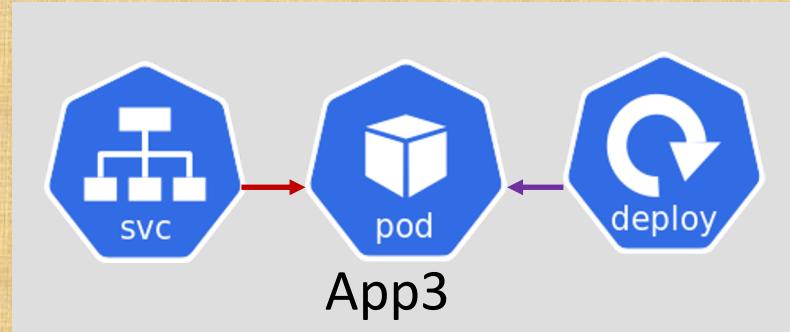
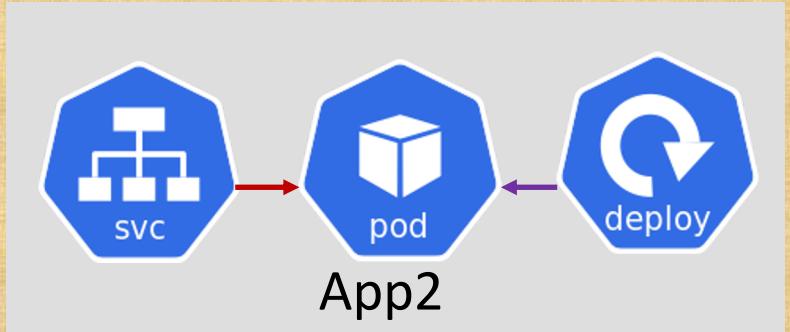
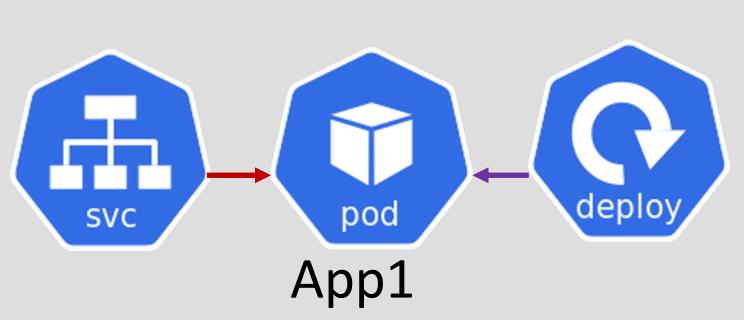


Ingress Service: app2-ingress

Priority: 30



Ingress Service: app3-ingress



# Ingress Groups

1

IngressGroup feature enables you to group multiple Ingress resources together.

2

The controller will automatically merge Ingress rules for all Ingresses within IngressGroup and support them with a single ALB.

3

**VERY VERY IMPORTANT NOTE:** In addition, most annotations defined on an Ingress only applies to the paths defined by that Ingress.

Sample

```
# Ingress Groups
alb.ingress.kubernetes.io/group.name: myapps.web
alb.ingress.kubernetes.io/group.order: '10'
```

- ✓ 36-EKS-Ingress-Groups
  - > 01-ekscluster-terraform-manifests
  - > 02-lbc-install-terraform-manifests
  - > 03-externaldns-install-terraform-manifests
  - > 04-kube-manifests-ingress-groups
  - ✓ 05-ingress-groups-terraform-manifests
    - > listen-ports
    - └ c1-versions.tf
    - └ c2-remote-state-datasource.tf
    - └ c3-providers.tf
    - └ c4-kubernetes-app1-deployment.tf
    - └ c5-kubernetes-app2-deployment.tf
    - └ c5-kubernetes-app3-deployment.tf
    - └ c7-kubernetes-app1-nodeport-service.tf
    - └ c8-kubernetes-app2-nodeport-service.tf
    - └ c9-kubernetes-app3-nodeport-service.tf
    - └ c10-kubernetes-app1-ingress-service.tf
    - └ c11-kubernetes-app2-ingress-service.tf
    - └ c12-kubernetes-app3-ingress-service.tf
    - └ c13-acm-certificate.tf

# What are we going to learn ?

## Project Folders

01

EKS Cluster Terraform Manifests

02

AWS Load Balancer Controller Terraform Manifests

03

External DNS Install Terraform Manifests

04

Kubernetes Manifests in **YAML** format  
**Sample Applications (3 Apps)**: Kubernetes Deployment, Node Port Service & **3 Ingress Services**

05

Kubernetes Manifests in **Terraform** format  
**Sample Applications (3 Apps)**: Kubernetes Deployment, Node Port Service & **3 Ingress Services**

```
✓ 04-kube-manifests-ingress-groups
  ✓ app1
    YAML Manifests
      ! 01-Nginx-App1-Deployment-and-NodePortService.yml
      ! 02-App1-Ingress.yml
  ✓ app2
    ! 01-Nginx-App2-Deployment-and-NodePortService.yml
    ! 02-App2-Ingress.yml
  ✓ app3
    ! 01-Nginx-App3-Deployment-and-NodePortService.yml
    ! 03-App3-Ingress-default-backend.yml
✓ 05-ingress-groups-terraform-manifests
  > listen-ports
    Terraform Manifests
      ✓ c1-versions.tf
      ✓ c2-remote-state-datasource.tf
      ✓ c3-providers.tf
      ✓ c4-kubernetes-app1-deployment.tf
      ✓ c5-kubernetes-app2-deployment.tf
      ✓ c5-kubernetes-app3-deployment.tf
      ✓ c7-kubernetes-app1-nodeport-service.tf
      ✓ c8-kubernetes-app2-nodeport-service.tf
      ✓ c9-kubernetes-app3-nodeport-service.tf
      ✓ c10-kubernetes-app1-ingress-service.tf
      ✓ c11-kubernetes-app2-ingress-service.tf
      ✓ c12-kubernetes-app3-ingress-service.tf
      ✓ c13-acm-certificate.tf
```

# Ingress Groups

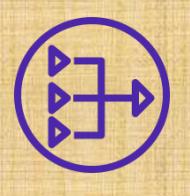
3 Ingress Kubernetes  
Manifests creates  
3 Ingress Resources  
and merge to create a  
Single AWS ALB.



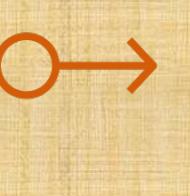
VPC



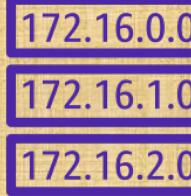
Internet gateway



NAT gateway



Elastic IP address



Route table



Public Subnet



Private Subnet



AWS S3

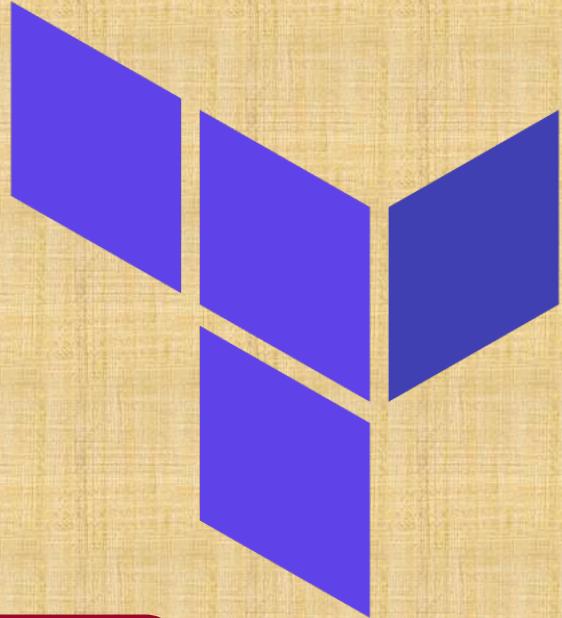


DynamoDB

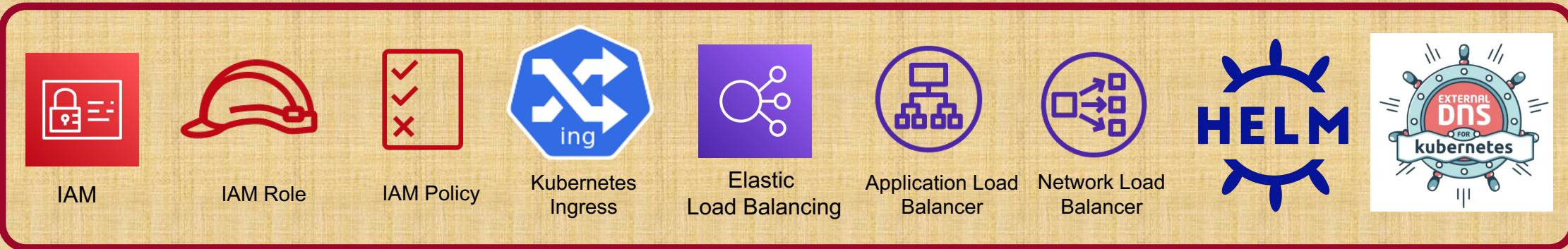


# AWS EKS

## Load Balancer Controller



EKS Cluster



Autoscaling

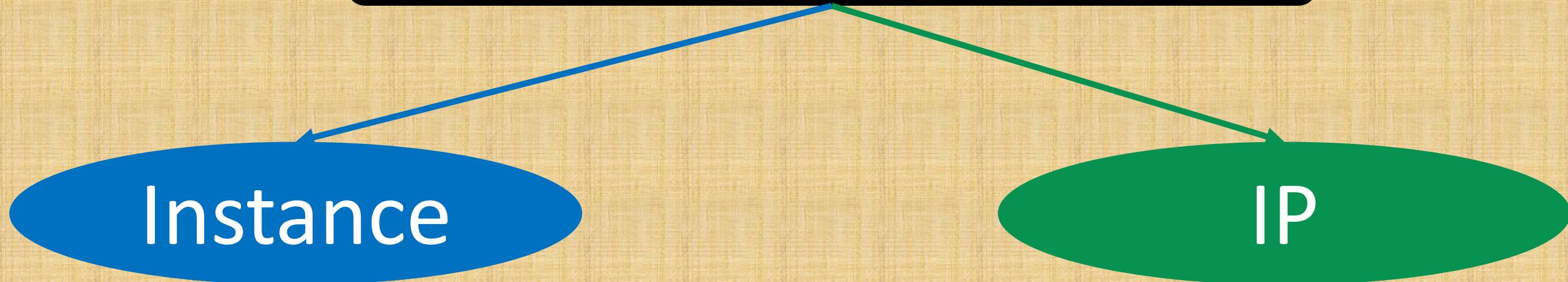
**Kubernetes Ingress - Target Type IP**  
**Automate with Terraform**

# AWS ALB Ingress Target Types

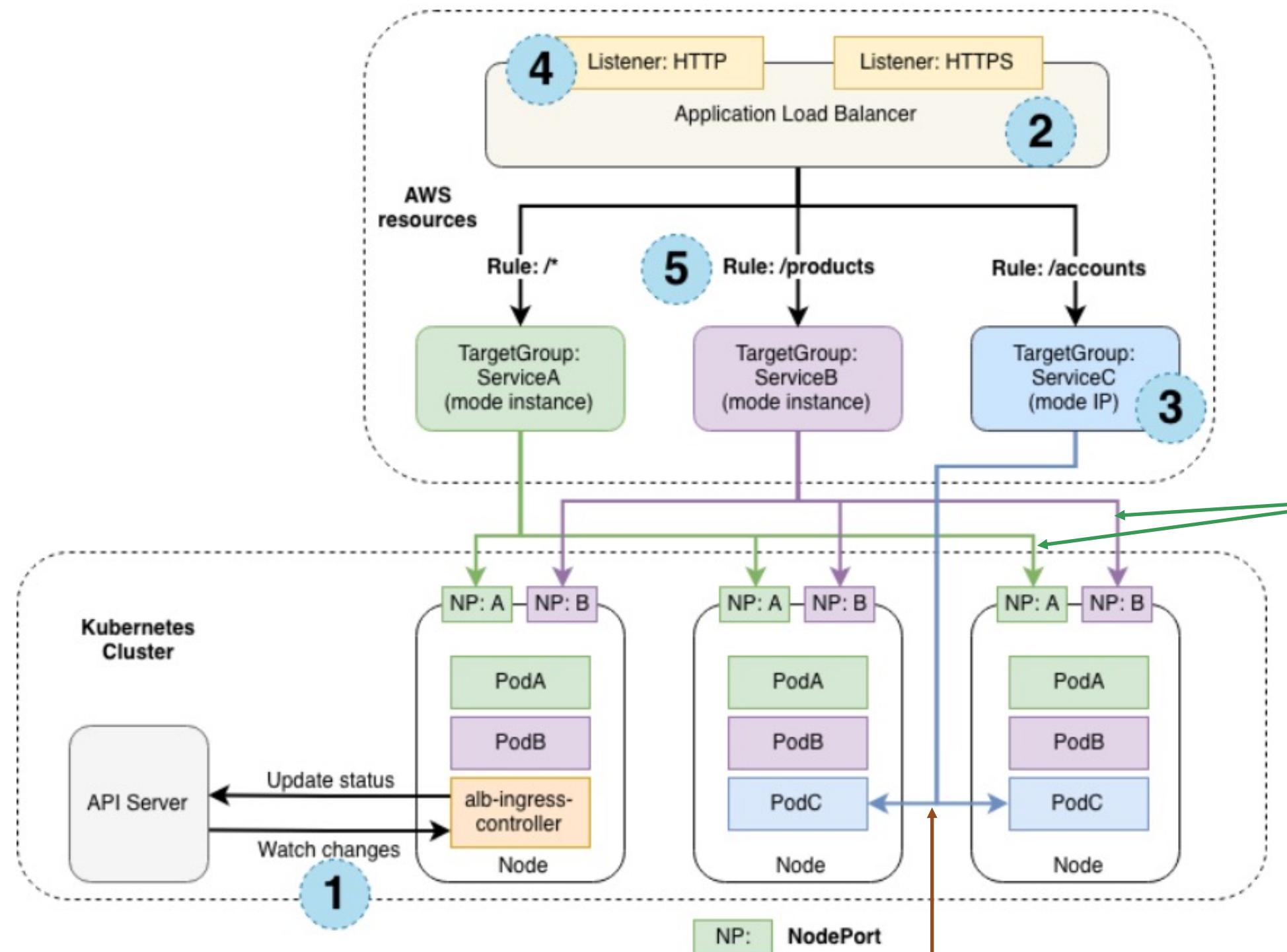
What is  
alb.ingress.kubernetes.io/target-type  
Annotation ?

target-type specifies how to route  
traffic to pods.

## ALB Ingress Target Types



# Ingress Target Types



Reference: <https://kubernetes-sigs.github.io/aws-load-balancer-controller/v2.4/how-it-works/>

# ALB Ingress Target Types

Instance

instance mode will route traffic to all ec2 instances within cluster on NodePort opened for your service.

service must be of type "NodePort" or "LoadBalancer" to use instance mode

Default Target Type is Instance Mode which means no need to define this annotation in Ingress, by default it takes Instance Mode

```
# Target Type: Instance  
alb.ingress.kubernetes.io/target-type: instance
```

IP

ip mode will route traffic directly to the pod IP.

ip mode is required for sticky sessions to work with Application Load Balancers.

Need to define the annotation explicitly as target-type: ip

```
# Target Type: IP  
alb.ingress.kubernetes.io/target-type: ip
```

# Ingress Annotation

```
# Target Type: IP (Defaults to Instance if not specified)  
alb.ingress.kubernetes.io/target-type: ip
```



YAML Format

Terraform Format



```
# Target Type: IP (Defaults to Instance if not specified)  
"alb.ingress.kubernetes.io/target-type" = "ip"
```

- ✓ 37-EKS-Ingress-TargetType-IP
  - > 01-ekscluster-terraform-manifests
  - > 02-lbc-install-terraform-manifests
  - > 03-externaldns-install-terraform-manifests
  - ✓ 04-kube-manifests-ingress-TargetType-IP
    - ! 01-Nginx-App1-Deployment-and-ClusterIPService.yml
    - ! 02-Nginx-App2-Deployment-and-ClusterIPService.yml
    - ! 03-Nginx-App3-Deployment-and-ClusterIPService.yml
    - ! 04-ALB-Ingress-target-type-ip.yml
- ✓ 05-ingress-TargetType-IP-terraform-manifests
  - > listen-ports
  - └ c1-versions.tf
  - └ c2-remote-state-datasource.tf
  - └ c3-providers.tf
  - └ c4-kubernetes-app1-deployment.tf
  - └ c5-kubernetes-app2-deployment.tf
  - └ c5-kubernetes-app3-deployment.tf
  - └ c7-kubernetes-app1-nodeport-service.tf
  - └ c8-kubernetes-app2-nodeport-service.tf
  - └ c9-kubernetes-app3-nodeport-service.tf
  - └ c10-kubernetes-ingress-service.tf
  - └ c11-acm-certificate.tf

# What are we going to learn ?

## Project Folders

01

EKS Cluster Terraform Manifests

02

AWS Load Balancer Controller Terraform Manifests

03

External DNS Install Terraform Manifests

04

Kubernetes Manifests in **YAML** format  
**Sample Applications (3 Apps):** Kubernetes Deployment, Node Port Service & Ingress Service

05

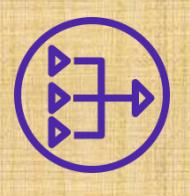
Kubernetes Manifests in **Terraform** format  
**Sample Applications (3 Apps):** Kubernetes Deployment, Node Port Service & Ingress Service



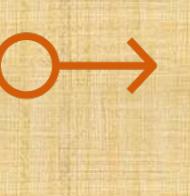
VPC



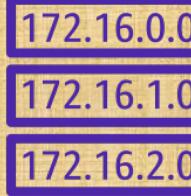
Internet gateway



NAT gateway



Elastic IP address



Route table



Public Subnet



Private Subnet



AWS S3

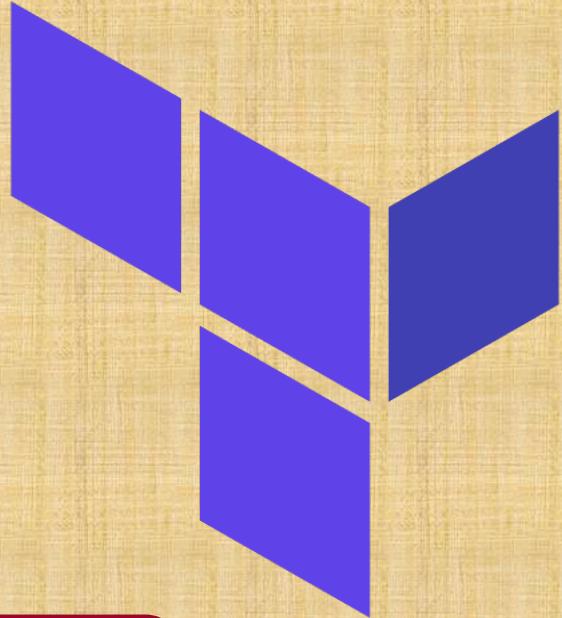


DynamoDB

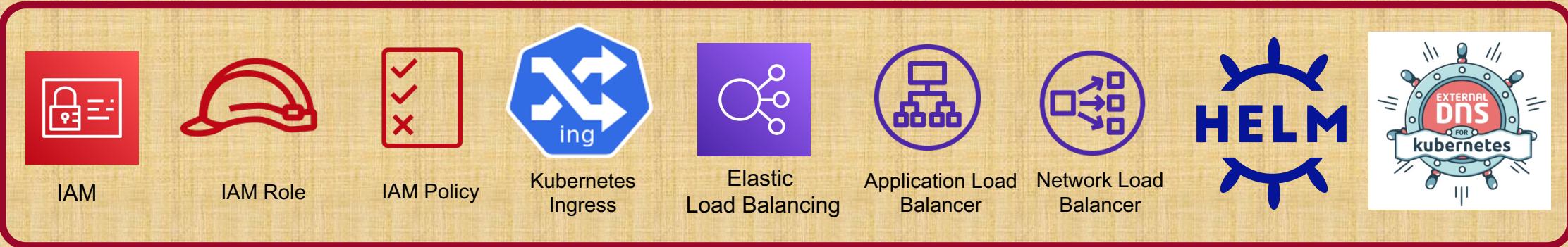


# AWS EKS

## Load Balancer Controller



EKS Cluster



Autoscaling

**Kubernetes Ingress - Internal Load Balancer**  
**Automate with Terraform**

# Ingress ALB Internal Load Balancer

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-internal-lb-demo
annotations:
  # Load Balancer Name
  alb.ingress.kubernetes.io/load-balancer-name: ingress-internal-lb
  # Ingress Core Settings
  #kubernetes.io/ingress.class: "alb" (OLD INGRESS CLASS NOTATION -
  # Creates External Application Load Balancer
  #alb.ingress.kubernetes.io/scheme: internet-facing
  # Creates Internal Application Load Balancer
  #alb.ingress.kubernetes.io/scheme: internal
  # Health Check Settings
  alb.ingress.kubernetes.io/healthcheck-protocol: HTTP
  alb.ingress.kubernetes.io/healthcheck-port: traffic-port
```

Creates AWS  
ALB Internal  
Load Balancer

YAML  
Manifest

# Ingress ALB Internal Load Balancer

```
# Kubernetes Service Manifest (Type: Load Balancer)
resource "kubernetes_ingress_v1" "ingress" {
  metadata {
    name = "ingress-internal-lb-demo"
    annotations = {
      # Load Balancer Name
      "alb.ingress.kubernetes.io/load-balancer-name" = "ingress-internal-lb-demo"
      # Ingress Core Settings
      # Creates External Application Load Balancer
      #"alb.ingress.kubernetes.io/scheme" = "internet-facing"
      # Creates Internal Application Load Balancer
      "alb.ingress.kubernetes.io/scheme" = "internal" ←
      # Health Check Settings
      "alb.ingress.kubernetes.io/healthcheck-protocol" = "HTTP"
      "alb.ingress.kubernetes.io/healthcheck-port" = "traffic-port"
    }
  }
}
```

Creates AWS  
ALB Internal  
Load Balancer

Terraform  
Manifest

# Curl Pod

```
apiVersion: v1
kind: Pod
metadata:
  name: curl-pod
spec:
  containers:
  - name: curl
    image: curlimages/curl
    command: [ "sleep", "600" ]
```

We **cannot** connect to Internal LB directly from internet to test it.

We will deploy a **curl pod** in EKS Cluster so we can connect to curl pod in EKS Cluster and test the **Internal LB Endpoint** using **curl** command.

# Curl Pod

```
# Kubernetes Curl Pod for Internal LB Testing
resource "kubernetes_pod_v1" "curl_pod" {
  metadata {
    name = "curl-pod"
  }
  spec {
    container {
      image = "curlimages/curl"
      name  = "curl"
      command = [ "sleep", "600" ]
    }
  }
}
```

Terraform  
Manifest

- ✓ 38-EKS-Ingress-InternalLB
  - > 01-ekscluster-terraform-manifests
  - > 02-lbc-install-terraform-manifests
  - > 03-externaldns-install-terraform-manifests
  - ✓ 04-kube-manifests-ingress-InternalLB
    - ! 01-Nginx-App1-Deployment-and-NodePortService.yml
    - ! 02-Nginx-App2-Deployment-and-NodePortService.yml
    - ! 03-Nginx-App3-Deployment-and-NodePortService.yml
    - ! 04-ALB-Ingress-Internal-LB.yml
  - ✓ 05-kube-manifests-curl
    - ! 01-curl-pod.yml
- ✓ 06-ingress-InternalLB-terraform-manifests
  - ↳ c1-versions.tf
  - ↳ c2-remote-state-datasource.tf
  - ↳ c3-providers.tf
  - ↳ c4-kubernetes-app1-deployment.tf
  - ↳ c5-kubernetes-app2-deployment.tf
  - ↳ c5-kubernetes-app3-deployment.tf
  - ↳ c7-kubernetes-app1-nodeport-service.tf
  - ↳ c8-kubernetes-app2-nodeport-service.tf
  - ↳ c9-kubernetes-app3-nodeport-service.tf
  - ↳ c10-kubernetes-ingress-service.tf
  - ↳ c11-kubernetes-curl-pod-for-testing-InternalLB.tf

**YAML Manifests**

**Terraform Manifests**

# What are we going to learn ?

## Project Folders

01

EKS Cluster Terraform Manifests

02

AWS Load Balancer Controller Terraform Manifests

03

External DNS Install Terraform Manifests

04, 05

Kubernetes Manifests in **YAML** format  
**Sample Applications (3 Apps):** Kubernetes Deployment, Node Port Service & Ingress Service

06

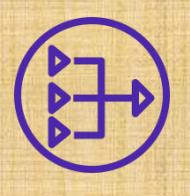
Kubernetes Manifests in **Terraform** format  
**Sample Applications (3 Apps):** Kubernetes Deployment, Node Port Service & Ingress Service



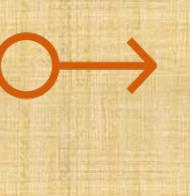
VPC



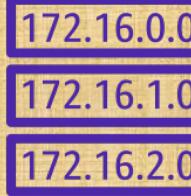
Internet gateway



NAT gateway



Elastic IP address



Route table



Public Subnet



Private Subnet



AWS S3

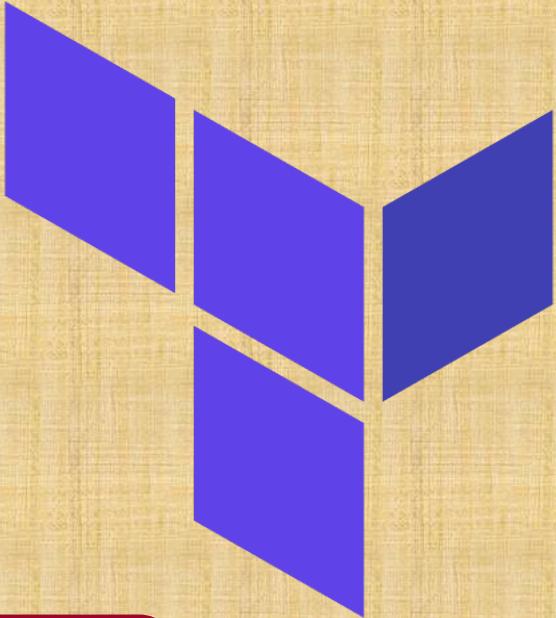


DynamoDB

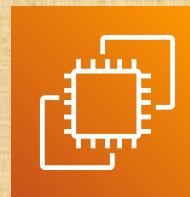
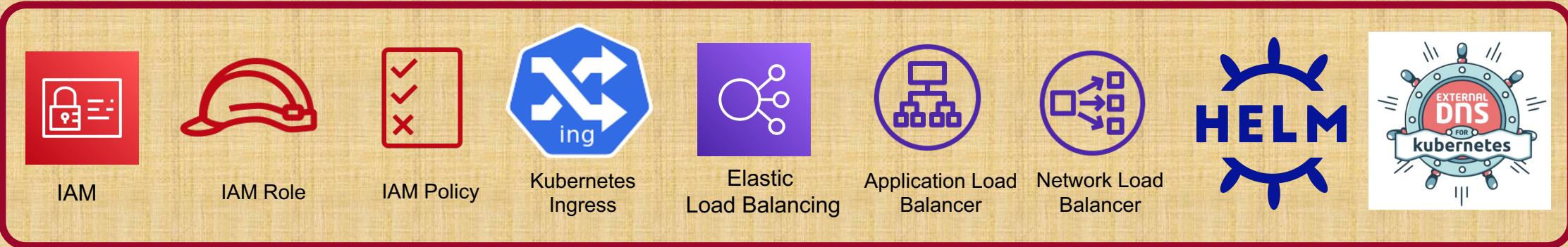


# AWS EKS

## Load Balancer Controller



EKS Cluster



EC2 VM



Autoscaling

**Kubernetes Ingress Cross Namespaces with Ingress Groups  
Automate with Terraform**

# Section 36 Demo - Ingress Groups



default Namespace

alb.ingress.kubernetes.io/load-balancer-name: ingress-groups-demo

alb.ingress.kubernetes.io/group.name: myapps.web

Priority: 10



Ingress Service: app1-ingress

Priority: 20



Ingress Service: app2-ingress

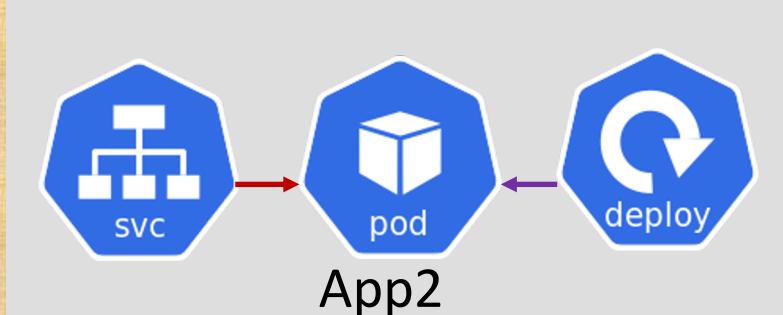
Priority: 30



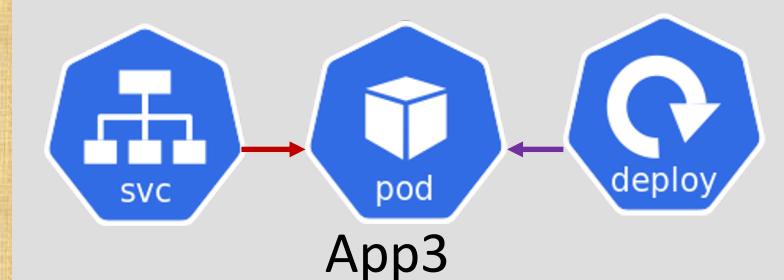
Ingress Service: app3-ingress



App1

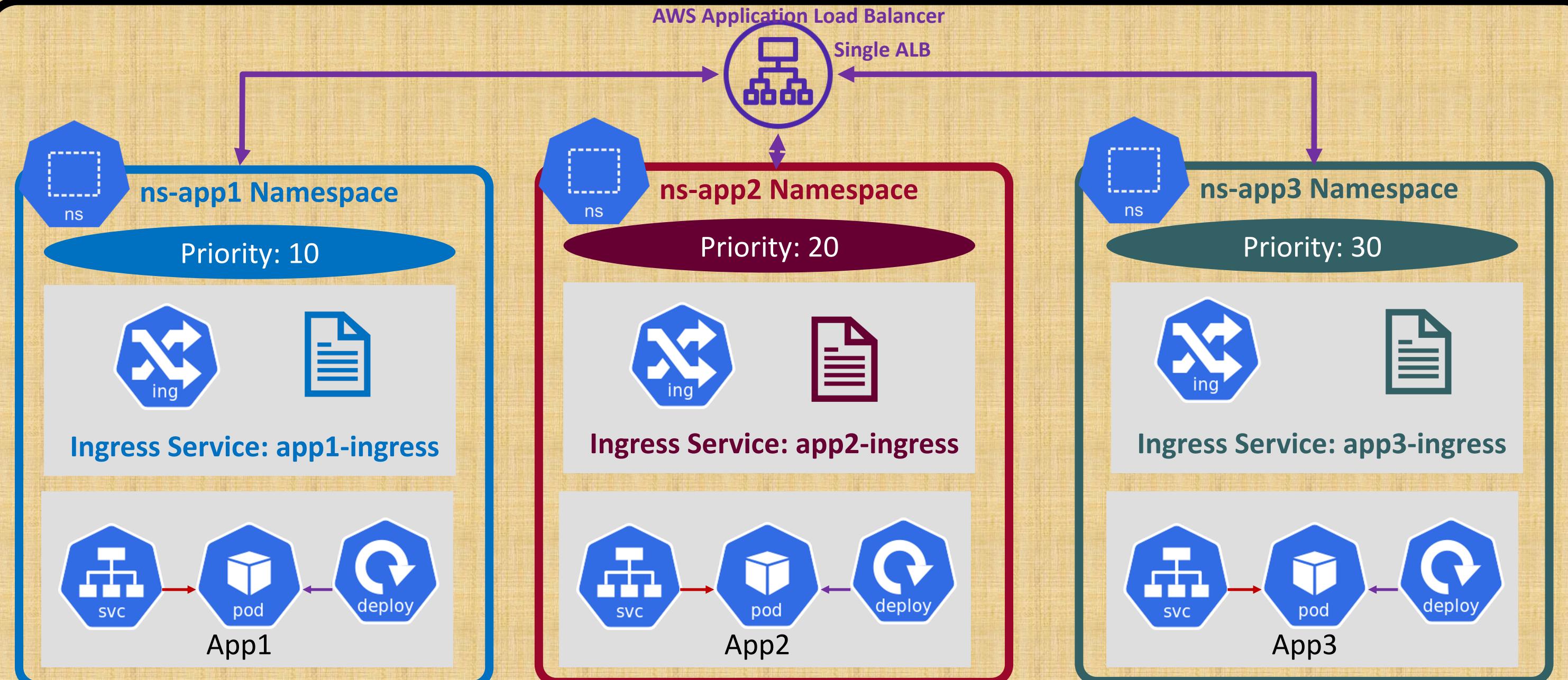


App2



App3

# Ingress Service - Cross Namespaces



alb.ingress.kubernetes.io/load-balancer-name: **ingress-crossns-demo**

alb.ingress.kubernetes.io/group.name: **myapps.web**

# Ingress Service - Cross Namespaces



**Usecase:** Single AWS Application Load Balancer for Multiple Ingress Services across Multiple Kubernetes Namespaces

## AWS ALB Ingress Controller (Legacy)

Main limitation of the ALB ingress controller is that it does not support cross-namespaces.

To work with multi-namespaces we must deploy ingress in each namespace and it will create another load balancer.

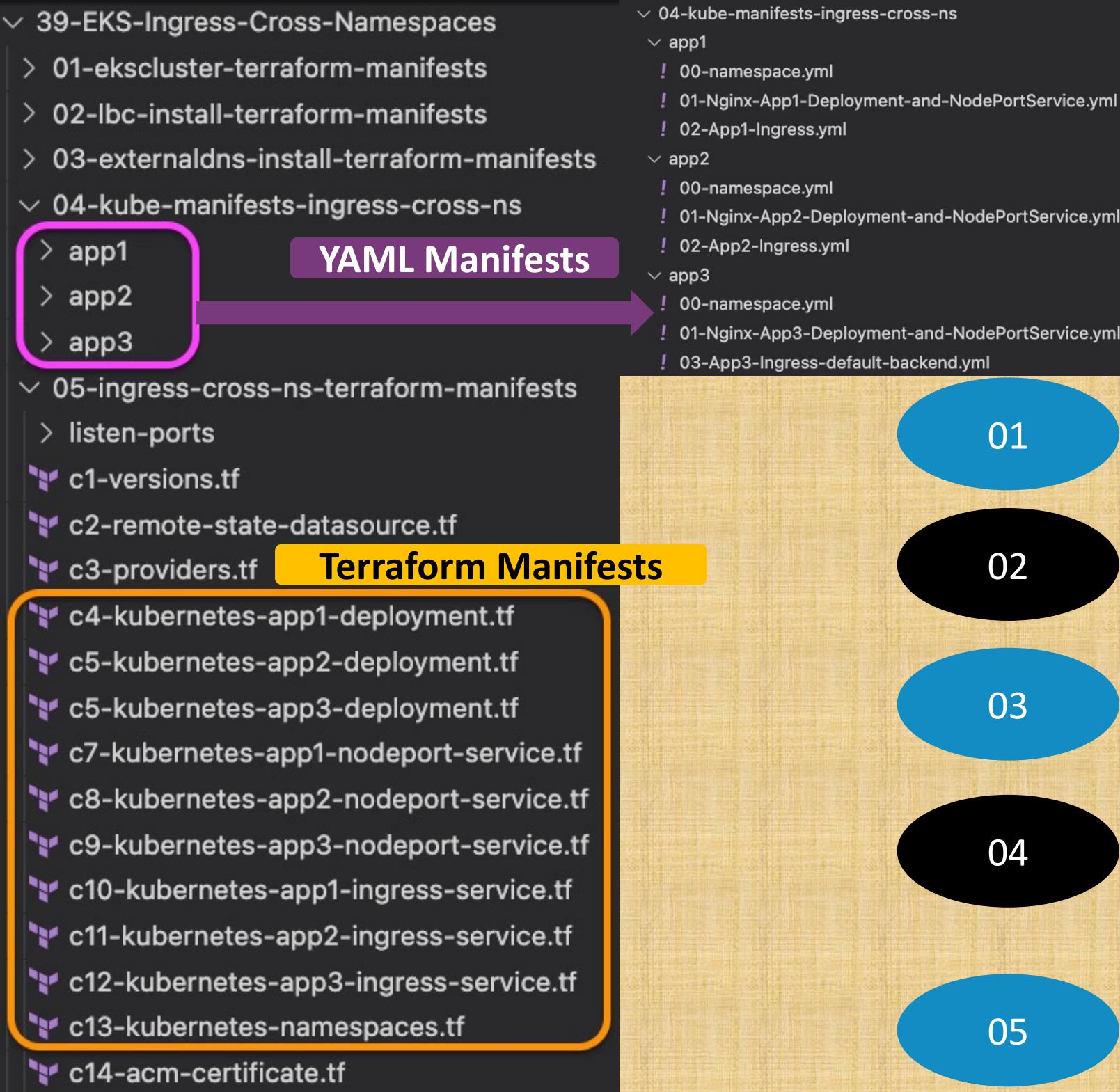
It is really very expensive if we have many namespaces.

## AWS Load Balancer Controller (Latest & Greatest)

Supports Ingress with Cross Namespaces

**Ingress Groups** concept helps us to achieve it

THIS IS A **VERY VERY CRITICAL REQUIREMENT** FROM LAST 4 YEARS FOR MANY KUBERNETES TEAMS AND IT IS **LIVE NOW** WITH AWS LOAD BALANCER CONTROLLER



# What are we going to learn ?

## Project Folders

EKS Cluster Terraform Manifests

AWS Load Balancer Controller Terraform Manifests

External DNS Install Terraform Manifests

Kubernetes Manifests in **YAML** format  
**Sample Applications (3 Apps):** Kubernetes Deployment, Node Port Service & Ingress Service

Kubernetes Manifests in **Terraform** format  
**Sample Applications (3 Apps):** Kubernetes NS, Deployment, Node Port Service & Ingress Service



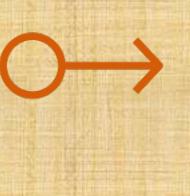
VPC



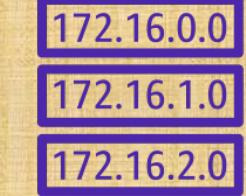
Internet gateway



NAT gateway



Elastic IP address



Route table



Public Subnet



Private Subnet



AWS S3

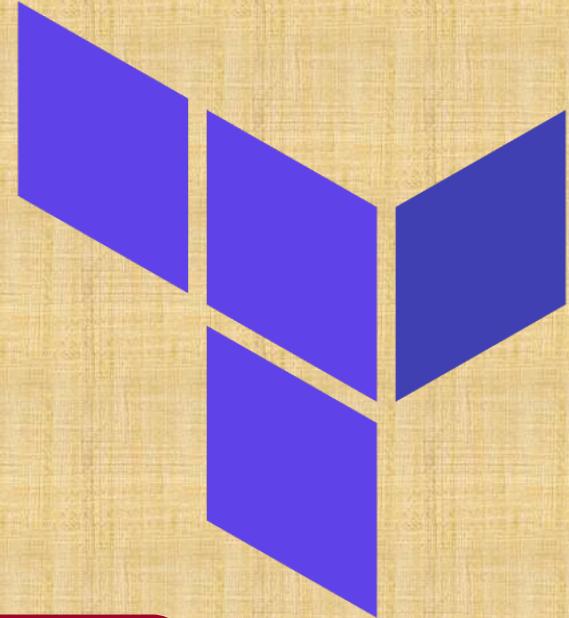


DynamoDB

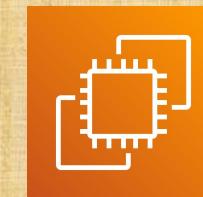


# AWS EKS

## Load Balancer Controller



EKS Cluster



EC2 VM



Autoscaling

**Kubernetes Service to Create AWS Network Load Balancer**  
**Automate with Terraform**

# Network Load Balancer

## What is Network Load Balancer ?

Network traffic is load balanced at L4 of the OSI model.

Supports TCP, UDP and TLS Protocols

**ELB Features Comparison for Additional Understanding:**  
<https://aws.amazon.com/elasticloadbalancing/features/>

# Options for creating Network Load Balancer using Kubernetes on EKS Cluster

## AWS cloud provider load balancer controller

Legacy Controller which can create AWS Classic LB and Network LB

Creates very basic Network Load Balancer

This controller will receive only critical bug fixes and in long run it will be deprecated

This controller will not have any new features added to it

- ✓ 07-ELB-Classic-and-Network-LoadBalancers
  - > 07-01-Create-EKS-Private-NodeGroup
  - > 07-02-Classic-LoadBalancer-CLB
  - > 07-03-Network-LoadBalancer-NLB

## AWS Load Balancer Controller

Latest and greatest which can create AWS ALB and NLB

Supports Protocols TCP, UDP & TLS (SSL) and for Health Check it supports TCP, HTTP & HTTPS

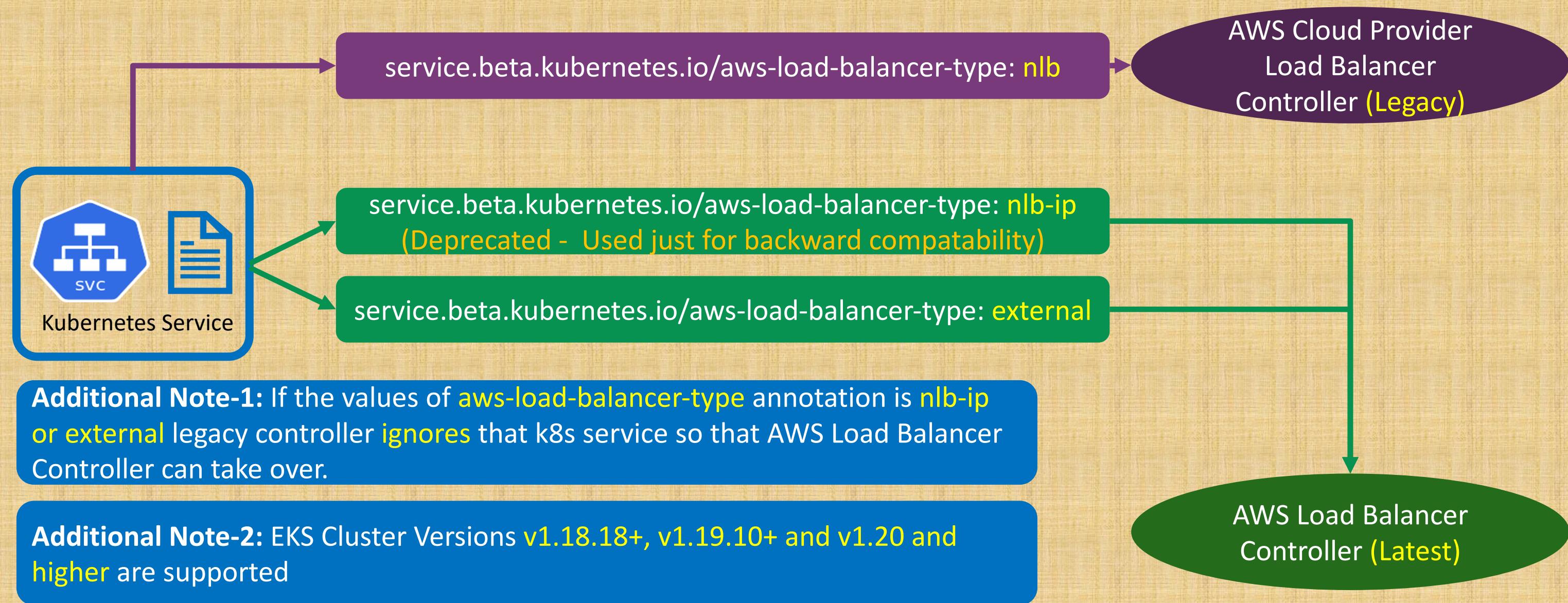
Supports Instance and IP Mode(For Fargate)

Supports Internal, External NLB and Custom Subnet Discovery

Supports Static Elastic IP and Access Control

Supports 25+ new Annotations  
<https://kubernetes-sigs.github.io/aws-load-balancer-controller/v2.4/guide/service/annotations/>

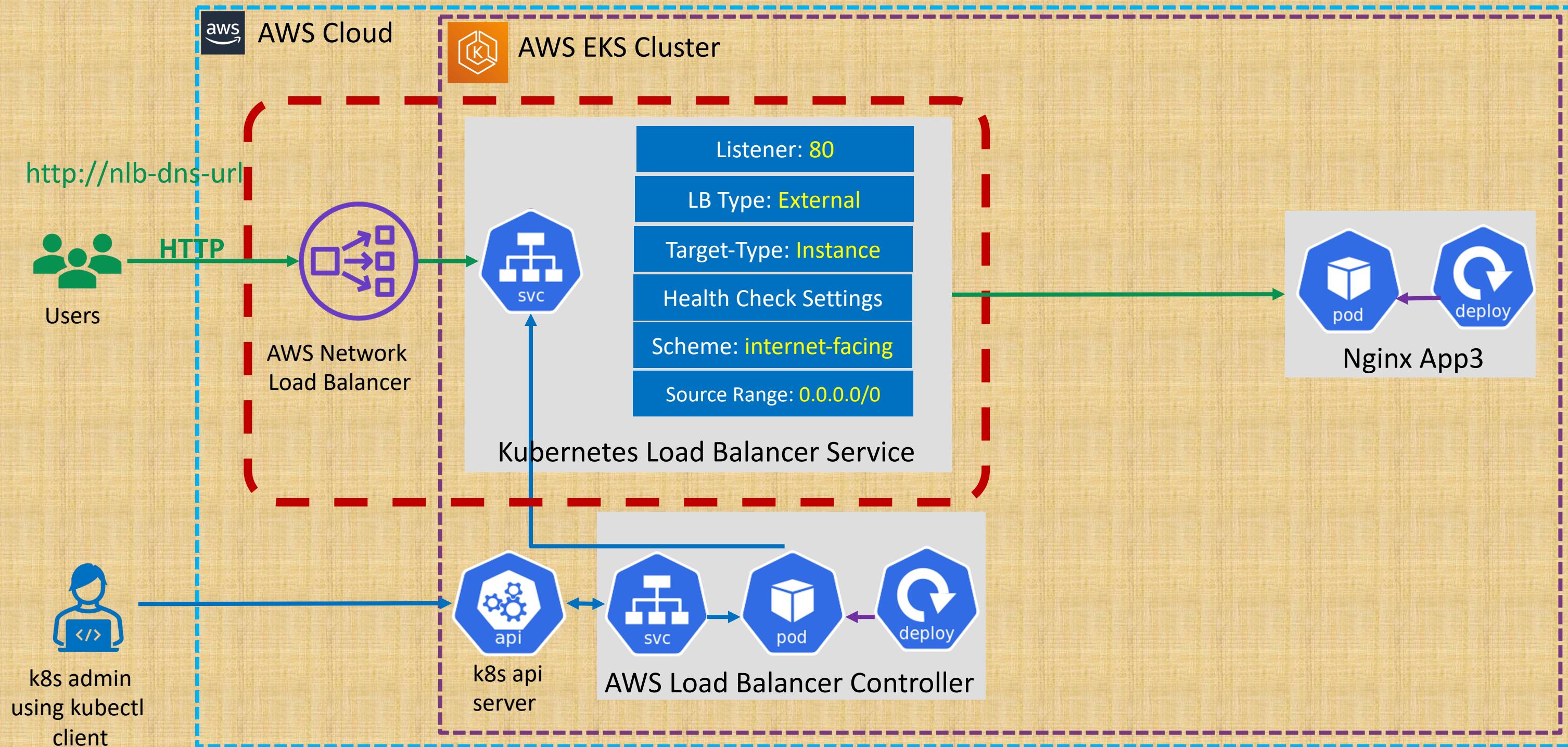
# How to ensure our Network LB created using Kubernetes Service will be associated with only latest AWS Load Balancer Controller ?

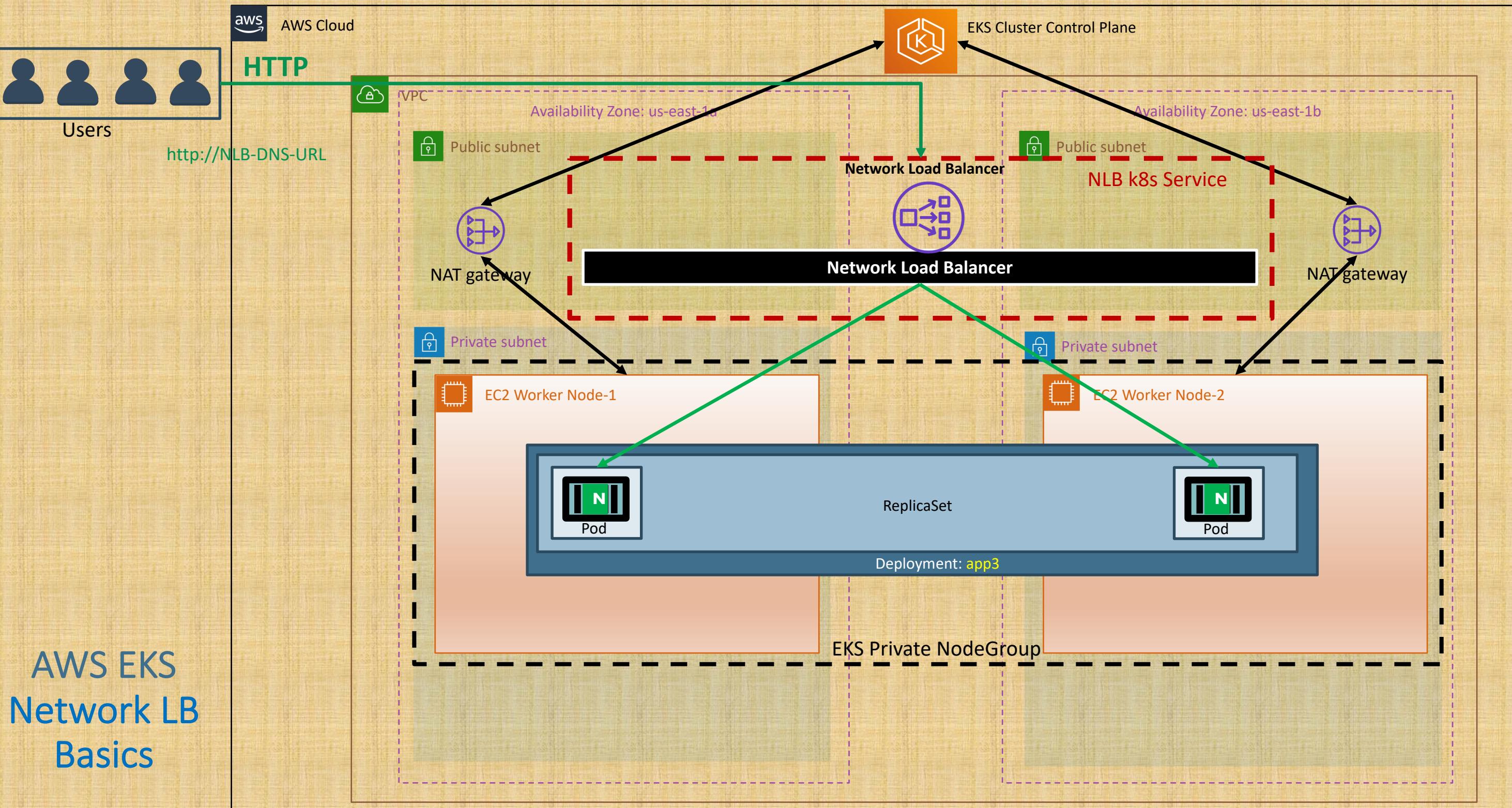


```
apiVersion: v1
kind: Service
metadata:
  name: lbc-network-lb-demo
  annotations:
    # Traffic Routing
    service.beta.kubernetes.io/aws-load-balancer-name: lbc-network-lb-demo
    service.beta.kubernetes.io/aws-load-balancer-type: external
    service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: instance
    #service.beta.kubernetes.io/aws-load-balancer-subnets: subnet-xxxx, mySubnet ## Subnets are auto-detected
    # Health Check Settings
    service.beta.kubernetes.io/aws-load-balancer-healthcheck-protocol: http
    service.beta.kubernetes.io/aws-load-balancer-healthcheck-port: traffic-port
    service.beta.kubernetes.io/aws-load-balancer-healthcheck-path: /index.html
    service.beta.kubernetes.io/aws-load-balancer-healthcheck-healthy-threshold: "3"
    service.beta.kubernetes.io/aws-load-balancer-healthcheck-unhealthy-threshold: "3"
    service.beta.kubernetes.io/aws-load-balancer-healthcheck-interval: "10" # The controller currently ignores this value
    # Access Control
    service.beta.kubernetes.io/load-balancer-source-ranges: 0.0.0.0/0
    service.beta.kubernetes.io/aws-load-balancer-scheme: "internet-facing"
    # AWS Resource Tags
    service.beta.kubernetes.io/aws-load-balancer-additional-resource-tags: Environment=dev,Team=test
spec:
  type: LoadBalancer
  selector:
    app: app3-nginx
  ports:
    - port: 80
      targetPort: 80
```

This Kubernetes Service creates Network Load Balancer which will be associated to latest AWS Load Balancer Controller

# AWS EKS NLB – Basics





## annotations:

```
# Traffic Routing
service.beta.kubernetes.io/aws-load-balancer-name: lbc-network-lb-demo
service.beta.kubernetes.io/aws-load-balancer-type: external
service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: instance
#service.beta.kubernetes.io/aws-load-balancer-subnets: subnet-xxxx, mySubnet ## Subnets

# Health Check Settings
service.beta.kubernetes.io/aws-load-balancer-healthcheck-protocol: http
service.beta.kubernetes.io/aws-load-balancer-healthcheck-port: traffic-port
service.beta.kubernetes.io/aws-load-balancer-healthcheck-path: /index.html
service.beta.kubernetes.io/aws-load-balancer-healthcheck-healthy-threshold: "3"
service.beta.kubernetes.io/aws-load-balancer-healthcheck-unhealthy-threshold: "3"
service.beta.kubernetes.io/aws-load-balancer-healthcheck-interval: "10" # The controller

# Access Control
service.beta.kubernetes.io/load-balancer-source-ranges: 0.0.0.0/0
service.beta.kubernetes.io/aws-load-balancer-scheme: "internet-facing"

# AWS Resource Tags
service.beta.kubernetes.io/aws-load-balancer-additional-resource-tags: Environment=dev,
```

Kubernetes  
Service of  
Type NLB

Annotations

# What are we going to learn ?

```
✓ 40-EKS-NLB-Basics
  > 01-ekscluster-terraform-manifests
  > 02-lbc-install-terraform-manifests
  > 03-externaldns-install-terraform-manifests
  ✓ 04-kube-manifests-nlb-basics
    ! 01-Nginx-App3-Deployment.yml
    ! 02-LBC-NLB-LoadBalancer-Service.yml
  ✓ 05-nlb-basics-terraform-manifests
    ✓ c1-versions.tf
    ✓ c2-remote-state-datasource.tf
    ✓ c3-providers.tf
    ✓ c4-kubernetes-app3-deployment.tf
    ✓ c5-kubernetes-app3-nlb-service.tf
```

01

02

03

04

05

## Project Folders

EKS Cluster Terraform Manifests

AWS Load Balancer Controller Terraform Manifests

External DNS Install Terraform Manifests

Kubernetes Manifests in **YAML** format  
**Sample Applications (1 App):** Kubernetes Deployment, Load Balancer Service

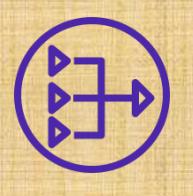
Kubernetes Manifests in **Terraform** format  
**Sample Applications (1 App):** Kubernetes Deployment, Load Balancer Service



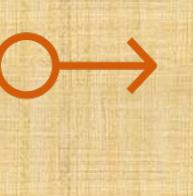
VPC



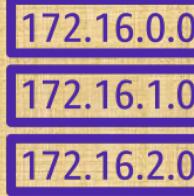
Internet gateway



NAT gateway



Elastic IP address



Route table



Public Subnet



Private Subnet



AWS S3

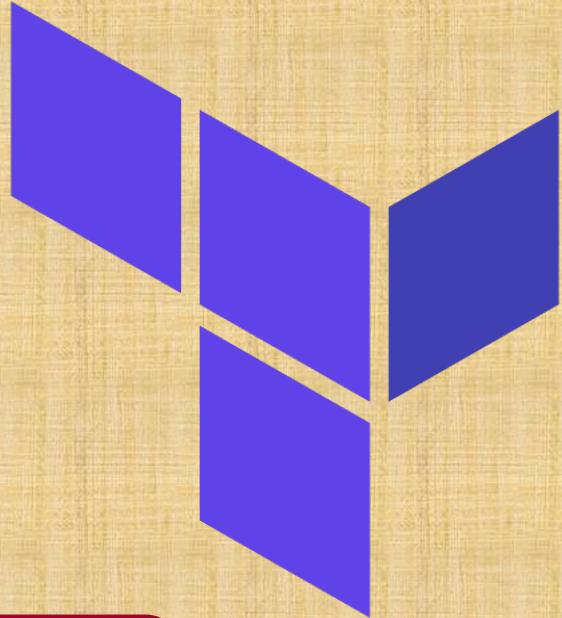


DynamoDB

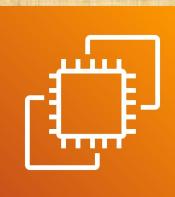
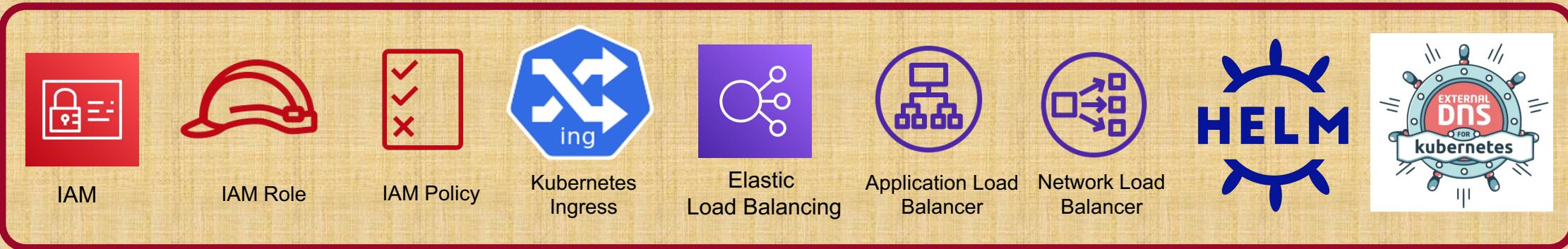


# AWS EKS

## Load Balancer Controller



EKS Cluster



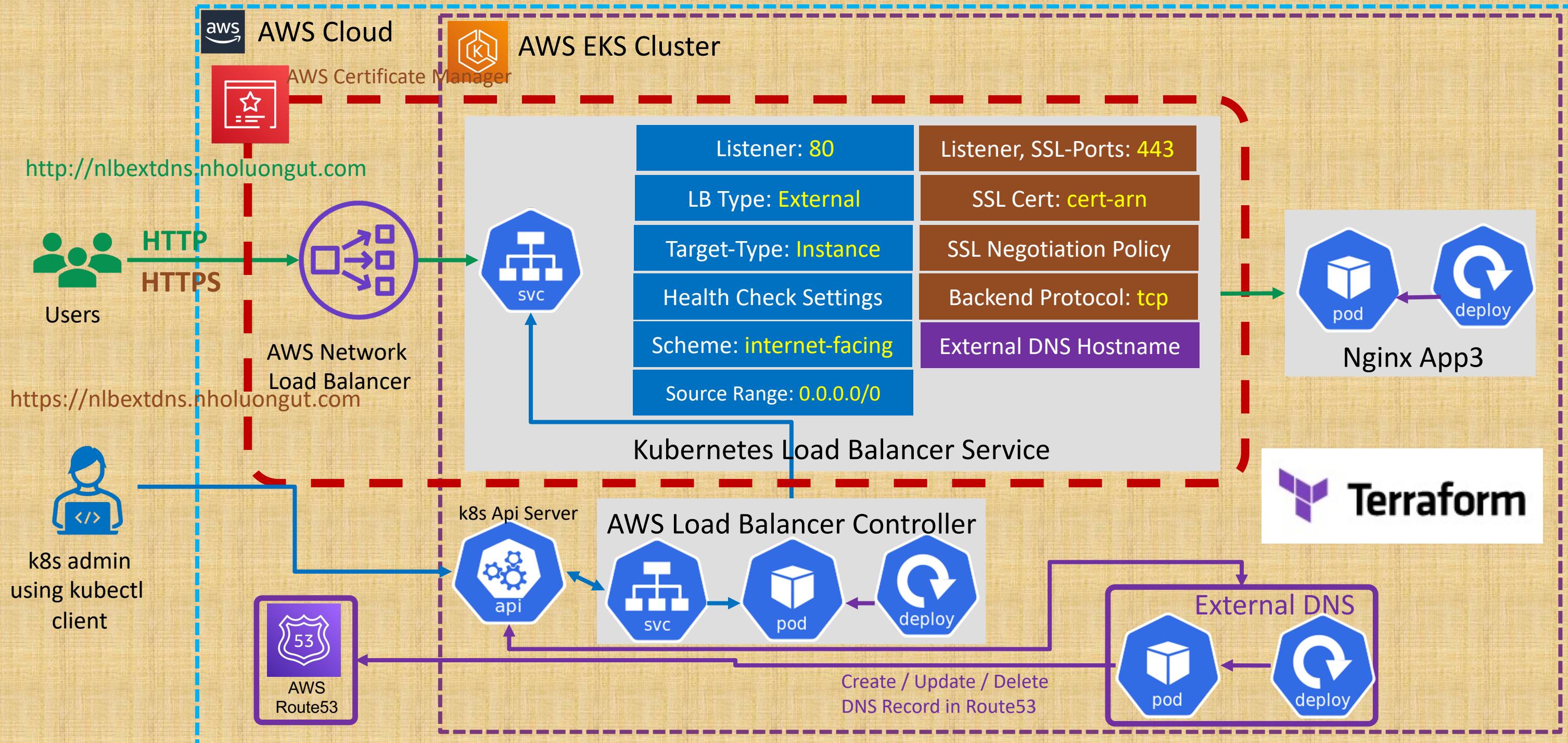
EC2 VM

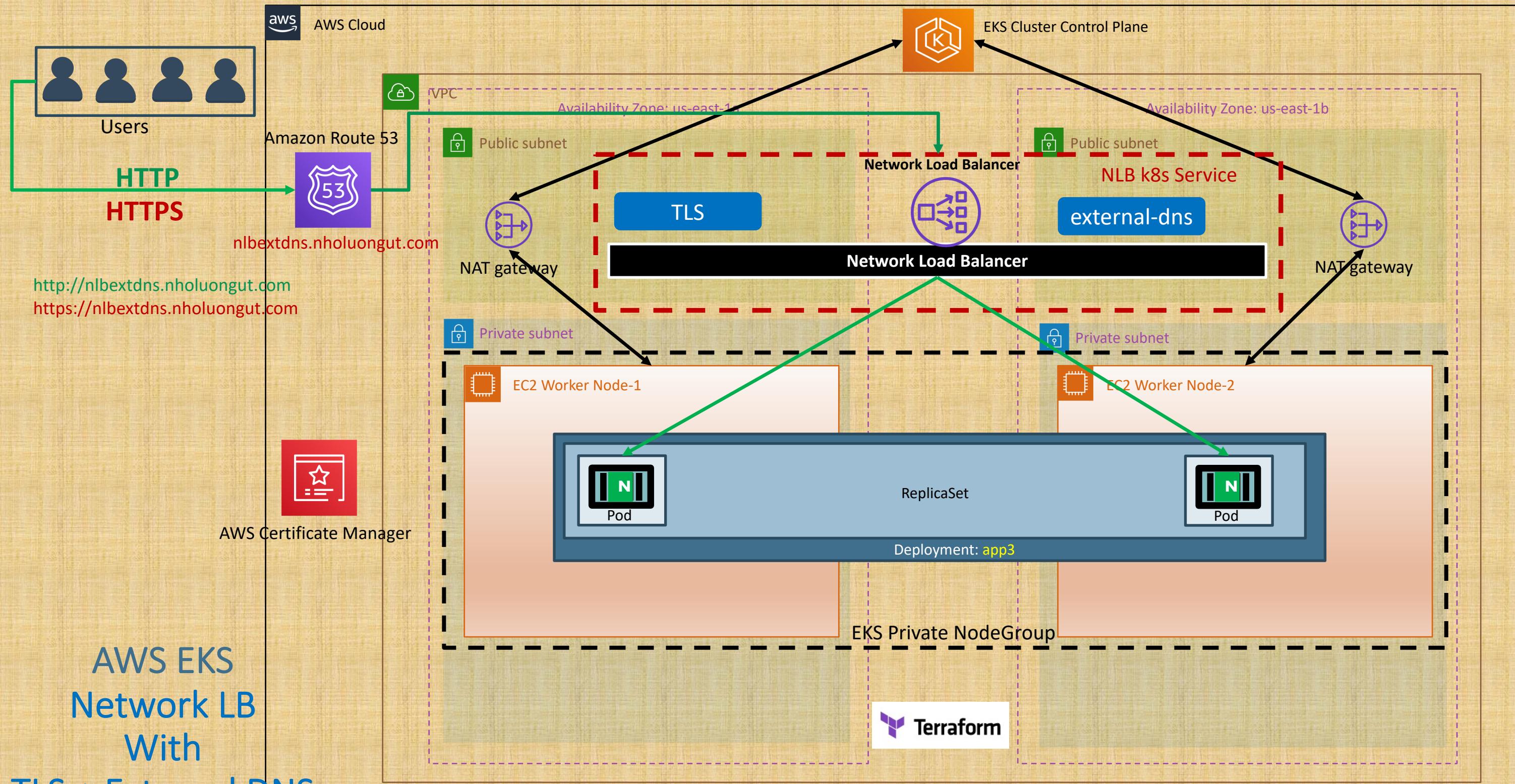


Autoscaling

**Kubernetes Service: Network Load Balancer TLS + External DNS**  
**Automate with Terraform**

# AWS EKS NLB - TLS + External DNS





AWS EKS  
Network LB  
With  
TLS + External DNS

# AWS EKS NLB - Annotations

```
# TLS
service.beta.kubernetes.io/aws-load-balancer-ssl-cert: arn:aws:acm:us-east-1:180789647333:certificate/d8e
service.beta.kubernetes.io/aws-load-balancer-ssl-ports: 443, # Specify this annotation if you need both T
service.beta.kubernetes.io/aws-load-balancer-ssl-negotiation-policy: ELBSecurityPolicy-TLS13-1-2-2021-06
service.beta.kubernetes.io/aws-load-balancer-backend-protocol: tcp
```

```
spec:
  type: LoadBalancer
  selector:
    app: app3-nginx
  ports:
    - name: http
      port: 80
      targetPort: 80
    - name: https
      port: 443
      targetPort: 80
```

# AWS EKS NLB - Annotations

```
# TLS
service.beta.kubernetes.io/aws-load-balancer-ssl-cert: arn:aws:acm:us-east-1:123456789012:certificate/12345678-1234-1234-1234-123456789012
service.beta.kubernetes.io/aws-load-balancer-ssl-ports: 443, # Specify this port if your application uses https
service.beta.kubernetes.io/aws-load-balancer-ssl-negotiation-policy: ELBSecurityPolicy-TLS-1-2-2017-1
service.beta.kubernetes.io/aws-load-balancer-backend-protocol: tcp # specifies the protocol for the target group

spec:
  type: LoadBalancer
  selector:
    app: app3-nginx
  ports:
    - name: http
      port: 80      # Creates NLB Port 80 Listener
      targetPort: 80 # Creates NLB Port 80 Target Group-1
    - name: https
      port: 443     # Creates NLB Port 443 Listener
      targetPort: 80 # Creates NLB Port 80 Target Group-2
    - name: http81
      port: 81      # Creates NLB Port 81 Listener
      targetPort: 80 # Creates NLB Port 80 Target Group-3
    - name: http82
      port: 82      # Creates NLB Port 82 Listener
      targetPort: 80 # Creates NLB Port 80 Target Group-4
```

YAML  
Manifest

# AWS EKS NLB - TLS + External DNS Annotations

Certificate ARN is dynamic

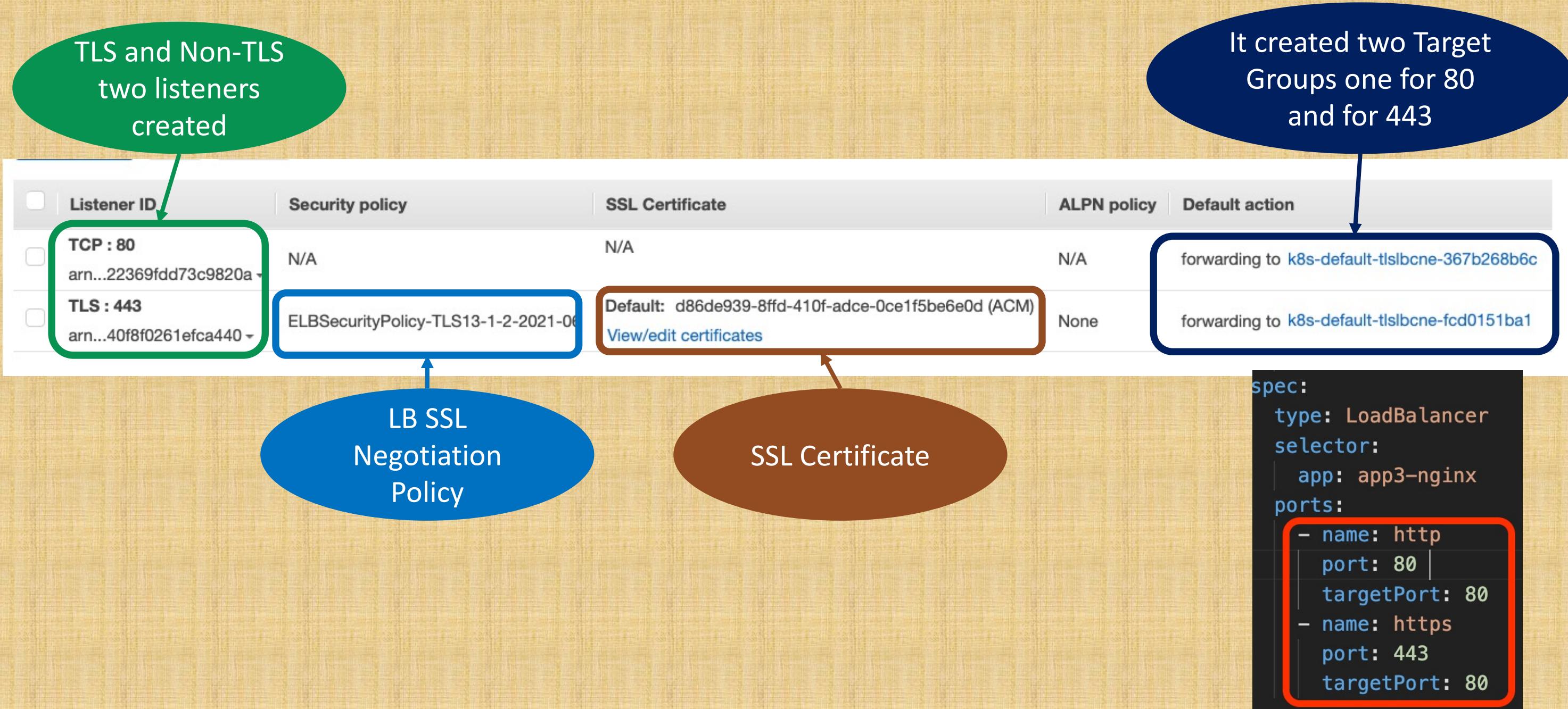
```
# TLS
"service.beta.kubernetes.io/aws-load-balancer-ssl-cert" = "${aws_acm_certificate.acm_cert.arn}"
"service.beta.kubernetes.io/aws-load-balancer-ssl-ports" = "443" # Specify this annotation if you need both TLS
"service.beta.kubernetes.io/aws-load-balancer-ssl-negotiation-policy" = "ELBSecurityPolicy-TLS13-1-2-2021-06"
"service.beta.kubernetes.io/aws-load-balancer-backend-protocol" = "tcp"
```

```
# External DNS – For creating a Record Set in Route53
"external-dns.alpha.kubernetes.io/hostname" = "tfnlbdns101.stackimplify.com"
```

```
# External DNS – For creating a Record Set in Route53
"external-dns.alpha.kubernetes.io/hostname" =
```

Terraform  
Manifest

# AWS EKS NLB - Load Balancer



# AWS EKS NLB -Target Groups

EC2 > Target groups

**Target groups (2) Info** C Actions ▾ Create target group

Search or filter target groups < 1 > ⚙️

<input type="checkbox"/>	Name	ARN	Port	Protocol	Target type	Load balancer
<input type="checkbox"/>	k8s-default-tlslbcne-367b268b6c	arn:aws:elasticloadbalancing:us-east-1:123456789012:targetgroup/k8s-default-tlslbcne-367b268b6c/7890123456789012	30810	TCP	Instance	tls-lbc-network-lb
<input type="checkbox"/>	k8s-default-tlslbcne-fcd0151ba1	arn:aws:elasticloadbalancing:us-east-1:123456789012:targetgroup/k8s-default-tlslbcne-fcd0151ba1/7890123456789012	32162	TCP	Instance	tls-lbc-network-lb

# AWS EKS NLB -Target Groups

arn:aws:elasticloadbalancing:us-east-1:180789647333:targetgroup/k8s-default-tlsLbcne-367b268b6c/e564572efca1f1aa

Details					
Target type Instance	Protocol : Port TCP: 30810	VPC vpc-0165a396e41e292a3	IP address type IPv4		
Total targets 2	Healthy <input checked="" type="radio"/> 2	Unhealthy <input type="radio"/> 0	Unused <input type="radio"/> 0	Initial <input type="radio"/> 0	Draining <input type="radio"/> 0

Targets    Monitoring    Health checks    Attributes    Tags

Registered targets (2)				
<input type="text"/> Filter resources by property or value				
<input type="checkbox"/>	Instance ID	Name	Port	Zone
<input type="checkbox"/>	i-0e06f6d06e5dfa0df	eksdemo1-eksdemo1-nginx-private1-Node	30810	us-east-1a
<input type="checkbox"/>	i-016128c9dd4835826	eksdemo1-eksdemo1-nginx-private1-Node	30810	us-east-1b

C    Deregister    Register targets    < 1 >   

Health status  
 healthy  
 healthy

# AWS EKS NLB -Target Groups

arn:aws:elasticloadbalancing:us-east-1:180789647333:targetgroup/k8s-default-tlslbcne-367b268b6c/e564572efca1f1aa

## Details

Target type Instance	Protocol : Port TCP: 30810	VPC vpc-0165a396e41e292a3	IP address type IPv4
Load balancer <a href="#">tls-lbc-network-lb</a>			
Total targets 2	Healthy 2	Unhealthy 0	Unused 0
Initial 0			Draining 0

Targets    Monitoring    **Health checks**    Attributes    Tags

## Health check settings

Protocol HTTP	Path /index.html	Port Traffic port	Healthy threshold 3 consecutive health check successes
Unhealthy threshold 3 consecutive health check failures	Timeout 6 seconds	Interval 10 seconds	Success codes 200-399

# What are we going to learn ?

```
✓ 41-EKS-NLB-TLS-externaldns
  > 01-ekscluster-terraform-manifests
  > 02-lbc-install-terraform-manifests
  > 03-externaldns-install-terraform-manifests
✓ 04-kube-manifests-nlb-tls-externaldns
  ! 01-Nginx-App3-Deployr YAML Manifests
  ! 02-LBC-NLB-LoadBalancer-Service.yml
✓ 05-nlb-tls-extdns-terraform-manifests
  ✓ c1-versions.tf
  ✓ c2-remote-state-datasource.tf
  ✓ c3-providers.tf Terraform Manifests
  ✓ c4-kubernetes-app3-deployment.tf
  ✓ c5-kubernetes-app3-nlb-service.tf
  ✓ c6-acm-certificate.tf
```

01

02

03

04

05

## Project Folders

EKS Cluster Terraform Manifests

AWS Load Balancer Controller Terraform Manifests

External DNS Install Terraform Manifests

Kubernetes Manifests in **YAML** format  
**Sample Applications (1 App):** Kubernetes Deployment, Load Balancer Service

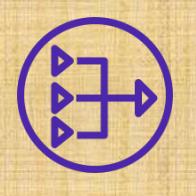
Kubernetes Manifests in **Terraform** format  
**Sample Applications (1 App):** Kubernetes Deployment, Load Balancer Service, ACM Cert



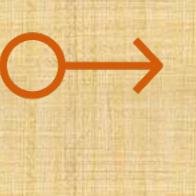
VPC



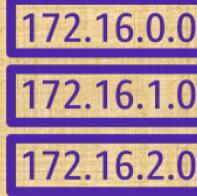
Internet gateway



NAT gateway



Elastic IP address



Route table



Public Subnet



Private Subnet



AWS S3

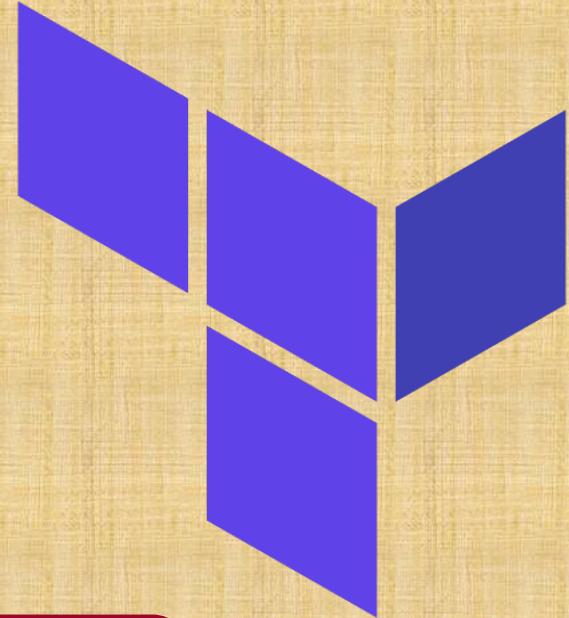


DynamoDB



# AWS EKS

## Load Balancer Controller



EKS Cluster

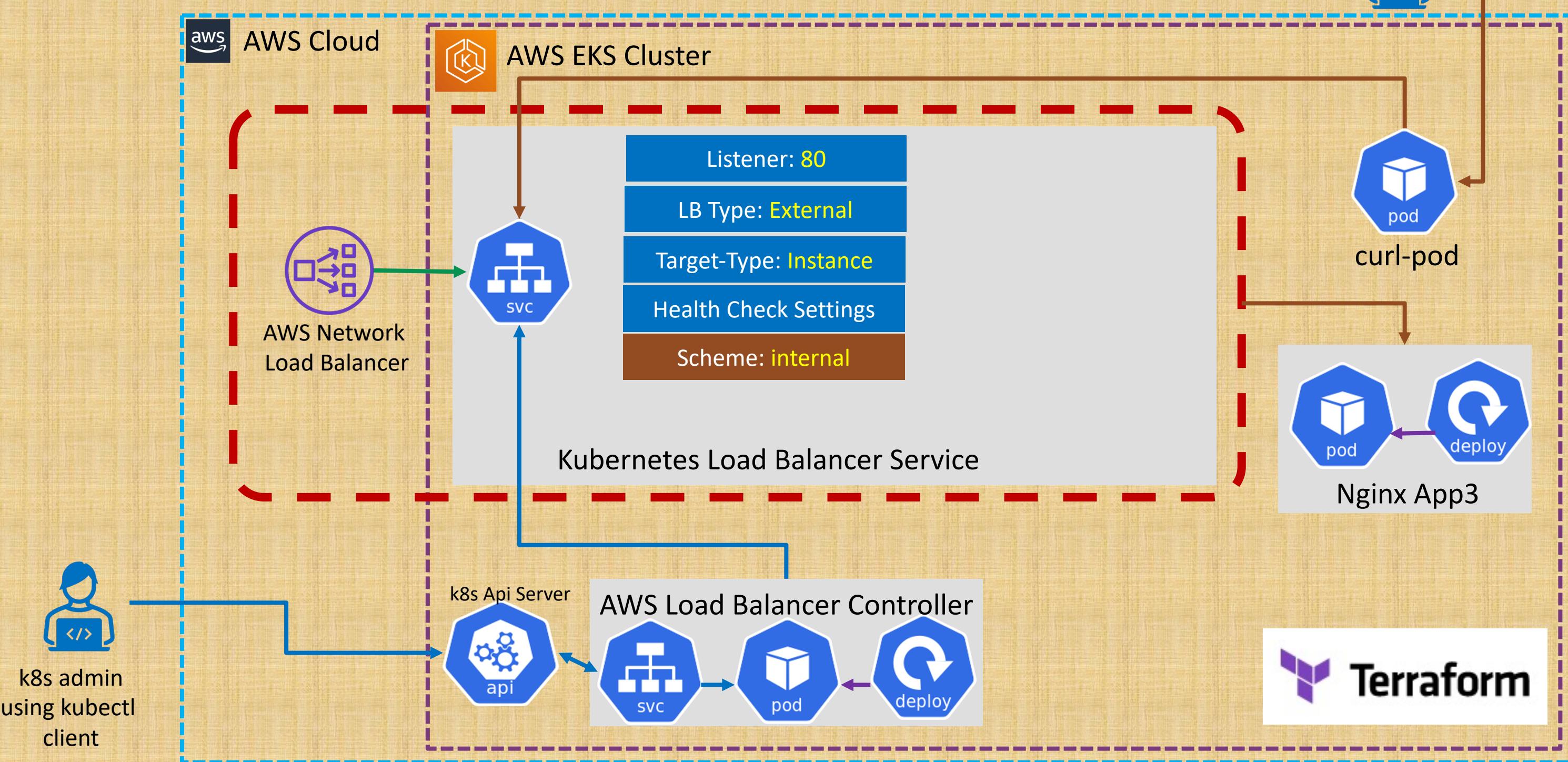


Autoscaling

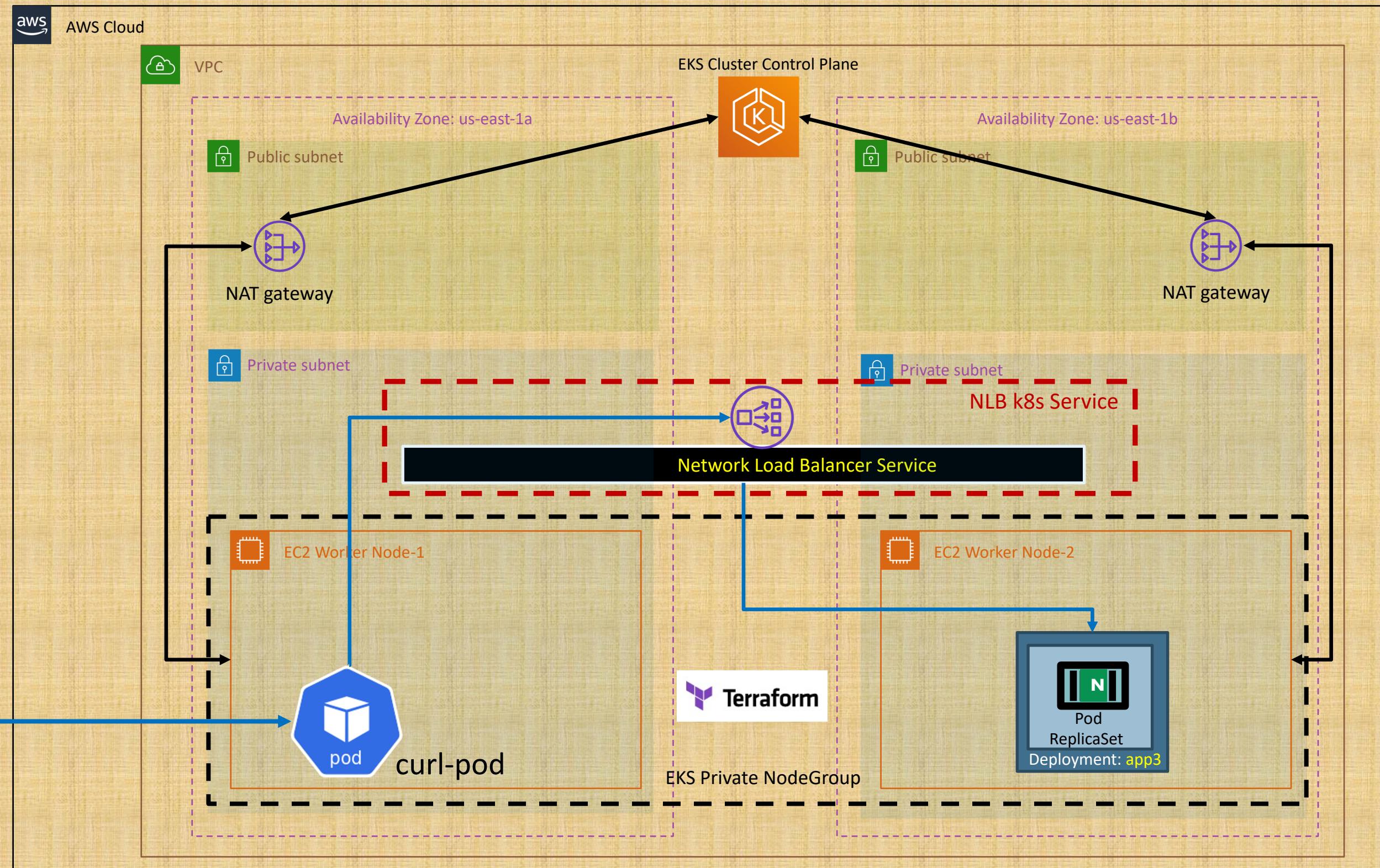
**Kubernetes Service: Internal Network Load Balancer**  
**Automate with Terraform**

# AWS EKS NLB – Internal LB

k8s admin using  
kubectl client



# AWS EKS Network Design With Network Load Balancer Internal



# AWS EKS NLB - Internal

```
# Access Control  
service.beta.kubernetes.io/aws-load-balancer-scheme: "internal"
```

# Curl Pod

```
apiVersion: v1
kind: Pod
metadata:
  name: curl-pod
spec:
  containers:
  - name: curl
    image: curlimages/curl
    command: [ "sleep", "600" ]
```

We **cannot** connect to Internal LB directly from internet to test it.

We will deploy a **curl pod** in EKS Cluster so we can connect to curl pod in EKS Cluster and test the **Internal LB Endpoint** using **curl** command.

# What are we going to learn ?

- ✓ 42-EKS-NLB-InternalLB
  - > 01-ekscluster-terraform-manifests
  - > 02-lbc-install-terraform-manifests
  - > 03-externaldns-install-terraform-manifests
  - ✓ 04-kube-manifests-nlb-internal
    - ! 01-Nginx-App3-Deployment.yml
    - ! 02-LBC-NLB-LoadBalancer-Service.yml
  - ✓ 05-kube-manifests-curl
    - ! 01-curl-pod.yml
- ✓ 06-nlb-internal-terraform-manifests
  - └ c1-versions.tf
  - └ c2-remote-state-datasource.tf
  - └ c3-providers.tf
  - └ c4-kubernetes-app3-deployment.tf
  - └ c5-kubernetes-app3-nlb-service.tf
  - └ c6-kubernetes-curl-pod-for-testing-InternalLB.tf

**YAML Manifests**

**Terraform Manifests**

## Project Folders

01

EKS Cluster Terraform Manifests

02

AWS Load Balancer Controller Terraform Manifests

03

External DNS Install Terraform Manifests

04

Kubernetes Manifests in **YAML** format  
**Sample Applications (1 App):** Kubernetes Deployment, Load Balancer Service

05

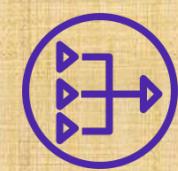
Kubernetes Manifests in **Terraform** format  
**Sample Applications (1 App):** Kubernetes Deployment, Load Balancer Service



VPC



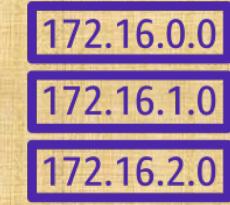
Internet gateway



NAT gateway



Elastic IP address



Route table



Public Subnet



Private Subnet



AWS S3



DynamoDB

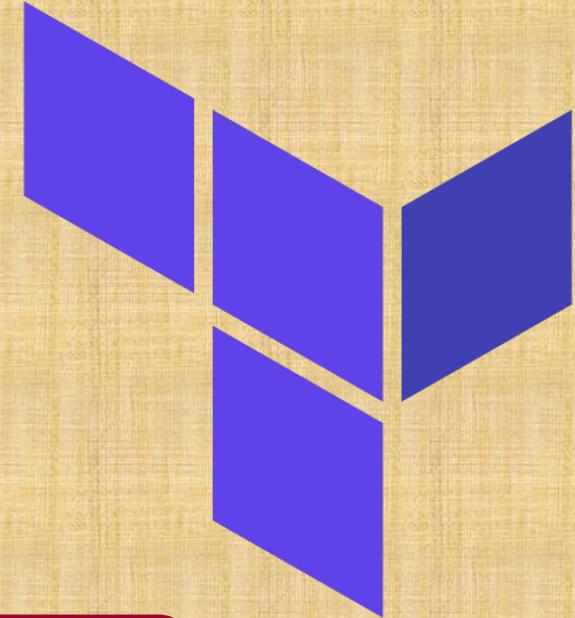


Fargate Profiles

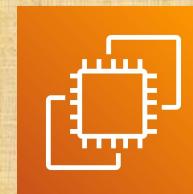
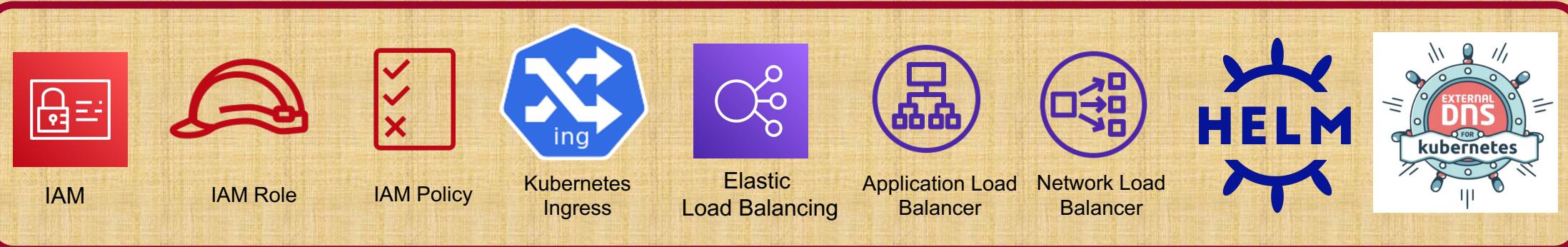


# AWS EKS

## FARGATE



EKS Cluster



EC2 VM



Autoscaling

What is AWS Fargate ?

# What is Fargate?

Fargate is a **Serverless** compute platform for containers on AWS

Fargate provides **on-demand**, right-sized **compute capacity** for containers

EKS **integrates** Kubernetes with Fargate by using **controllers** that are built by AWS

These controllers **run as part** of the EKS managed Kubernetes control plane and are **responsible** for scheduling native Kubernetes pods onto Fargate

The **Fargate controllers** include a **new scheduler** that runs alongside the **default Kubernetes scheduler**

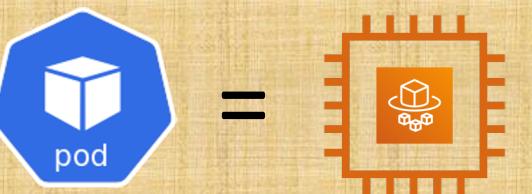
When we **start** a pod that meets the criteria for **running on Fargate**, the Fargate controllers running in the cluster **recognize** and **schedule** the pod onto Fargate.

# AWS EKS on Fargate



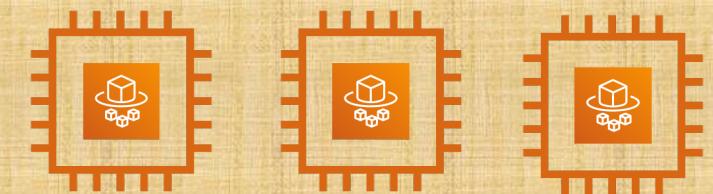
## Bring existing pods

- We **don't need** to change our existing pods
- Fargate works with **existing workflows and services** that run on kubernetes



## Production Ready

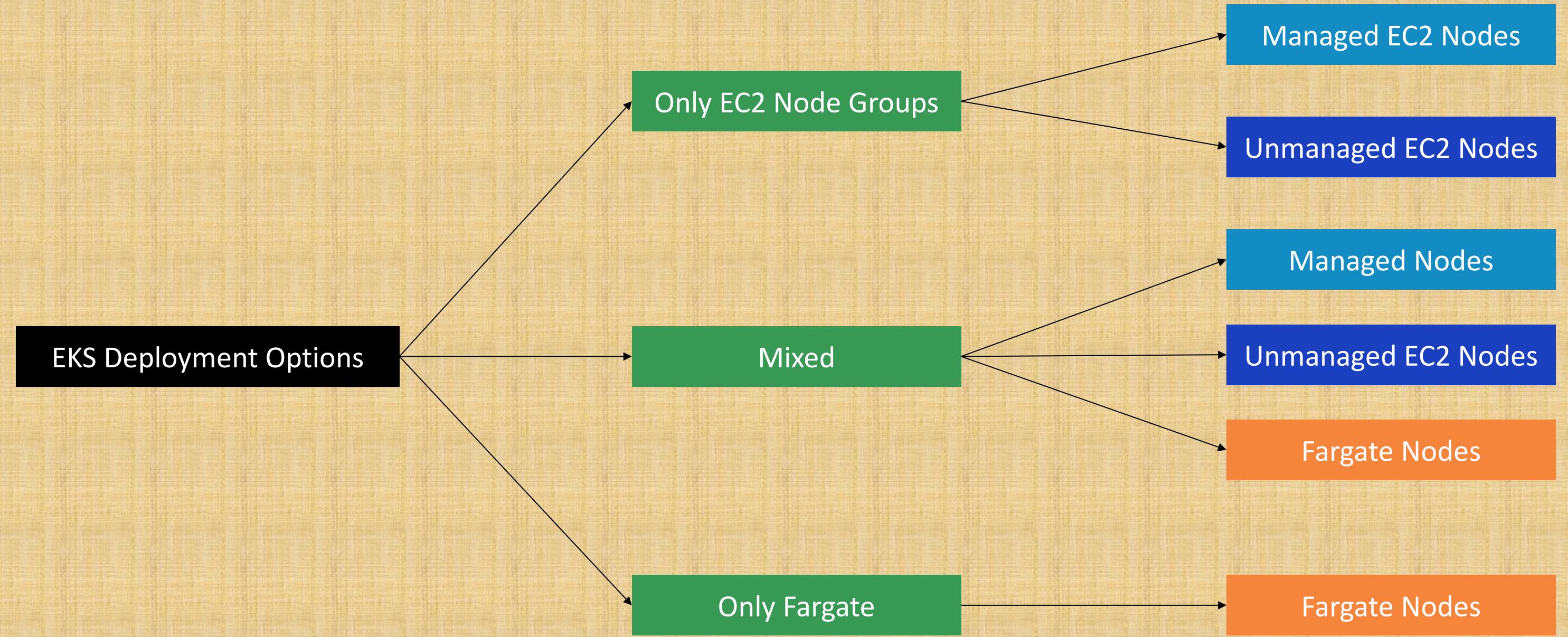
- Launch pods **easily**.
- Easily run pods across **Azs for HA**
- Each pod runs in an **isolated compute environment**



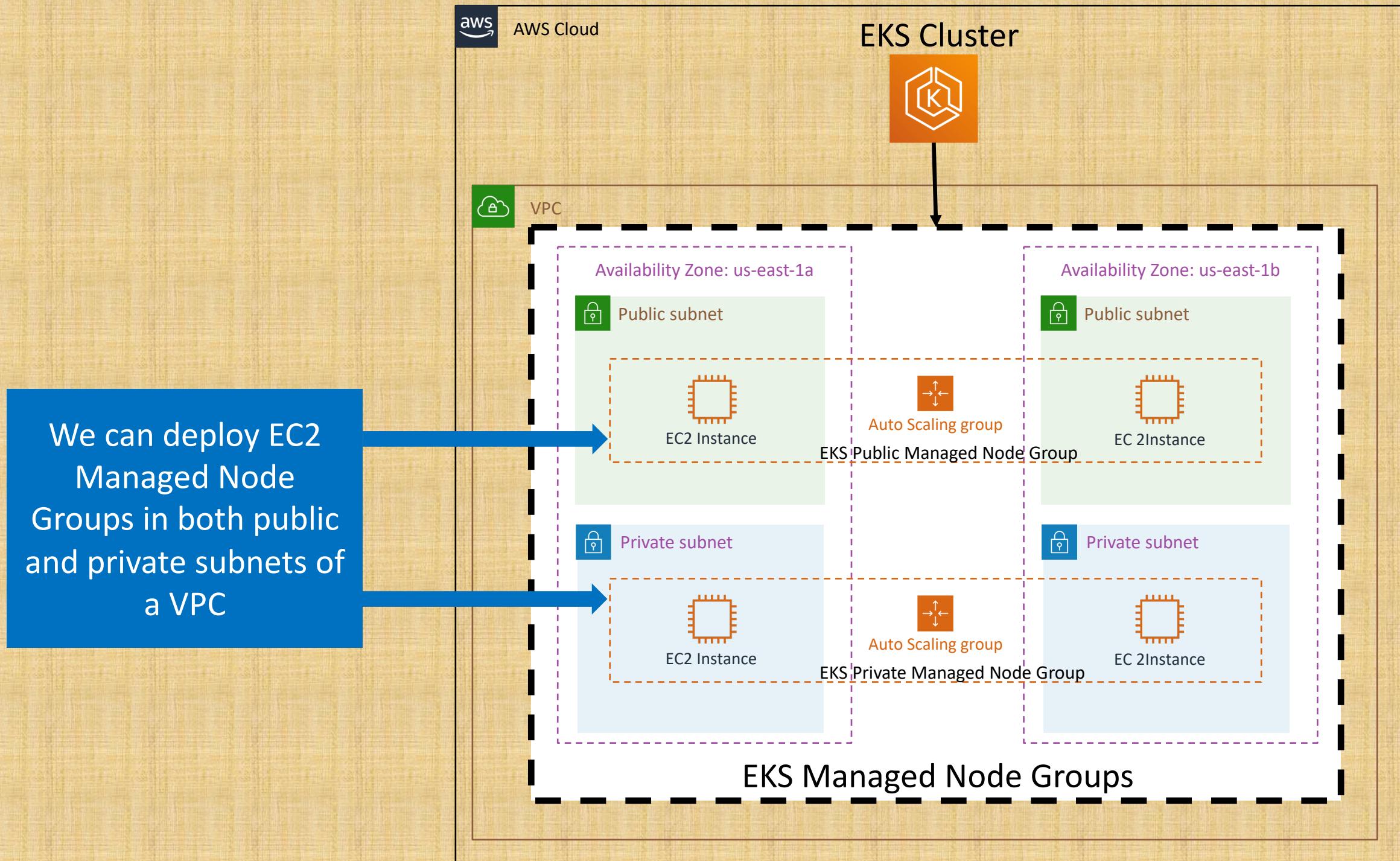
## Rightsized and Integrated

- Only **pay** for resources you need to run your pods
- Includes **native AWS integrations** for networking and security

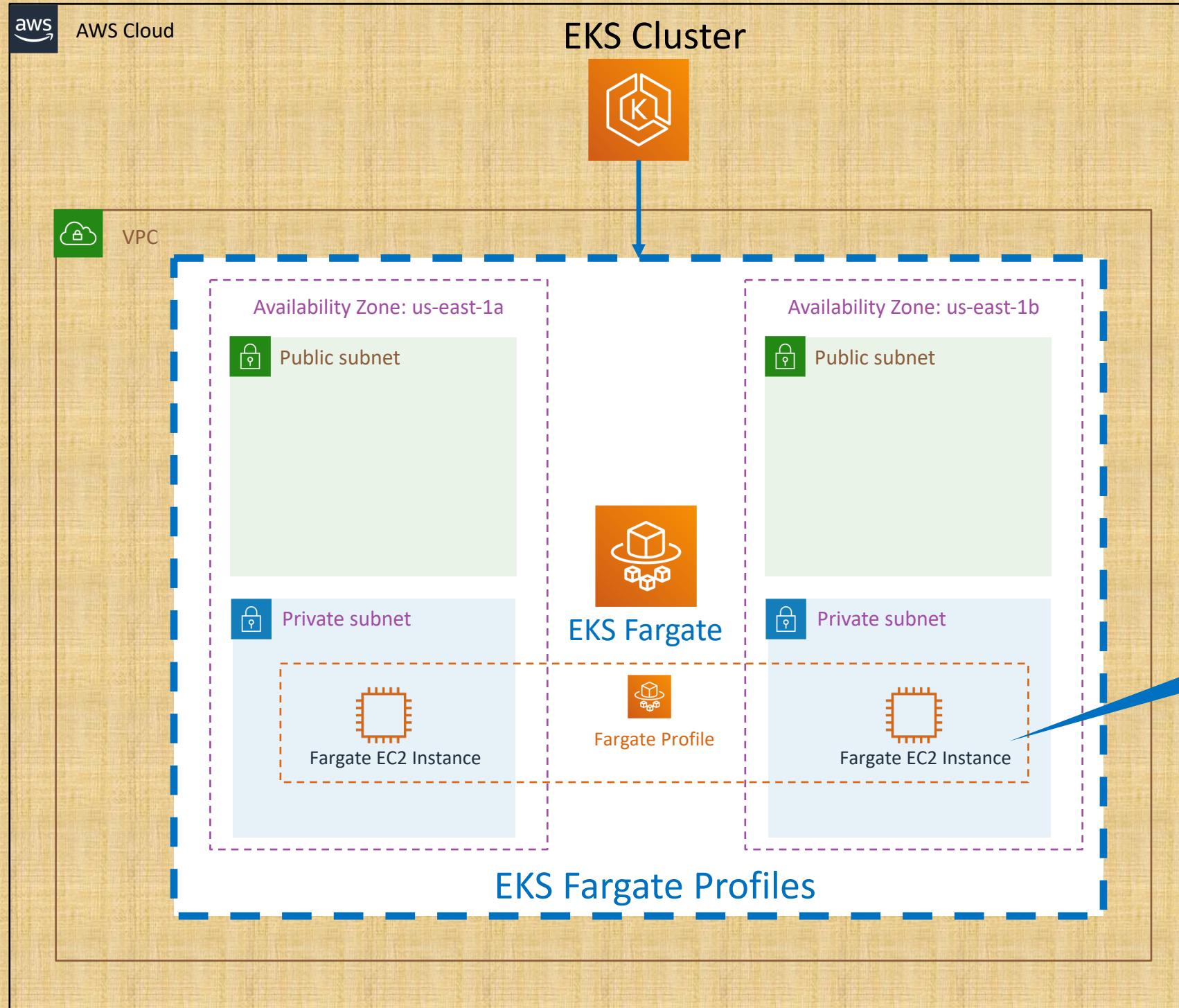
# EKS Deployment Options



# EKS Deployment Options – EC2 Node Groups

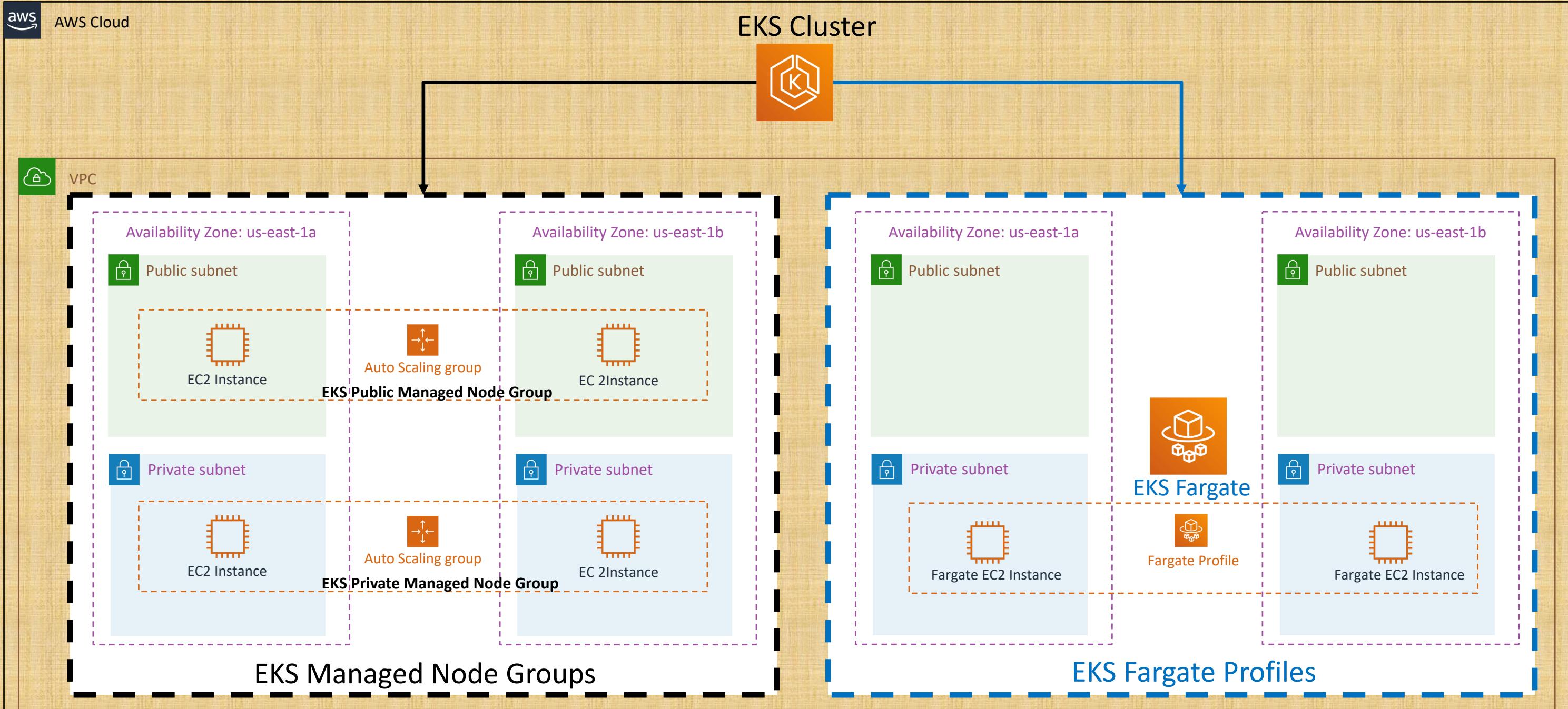


# EKS Deployment Options – Only Fargate



Pods running on  
Fargate are only  
supported on private  
subnets

# EKS Deployment Options - Mixed



# EKS Fargate vs Managed vs Unmanaged Nodes

	Fargate	Managed nodes	Unmanaged nodes
Units of work	Pod	Pod and EC2	Pod and EC2
Unit of charge	Pod	EC2	EC2
Host lifecycle	There is no visible host	AWS (SSH is allowed)	Customer
Host AMI	There is no visible host	AWS vetted AMIs	Customer BYO
Host : Pods	1 : 1	1 : many	1 : many

# EKS Fargate Considerations

There are many considerations we need to be aware of before we decide our Kubernetes workloads to run on Fargate.

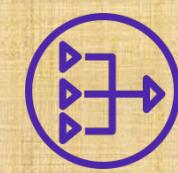
**Reference:** <https://docs.aws.amazon.com/eks/latest/userguide/fargate.html>



VPC



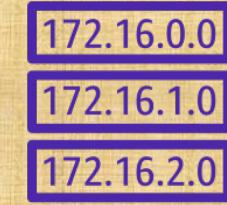
Internet gateway



NAT gateway



Elastic IP address



Route table



Public Subnet



Private Subnet



AWS S3



DynamoDB

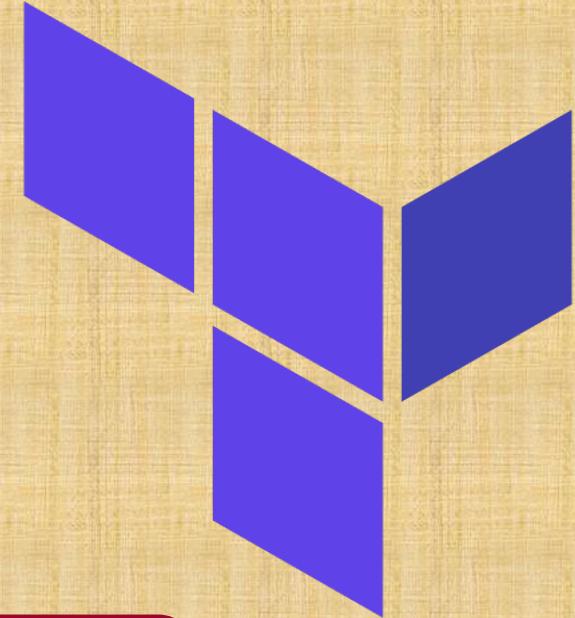


Fargate Profiles

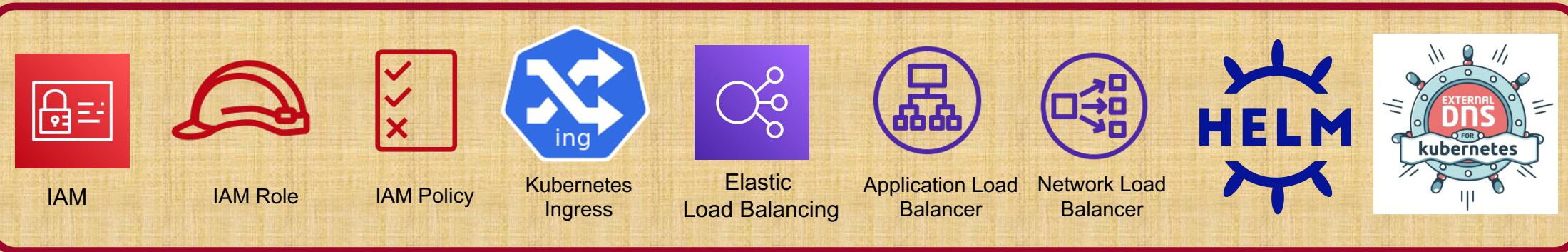


# AWS EKS

## FARGATE

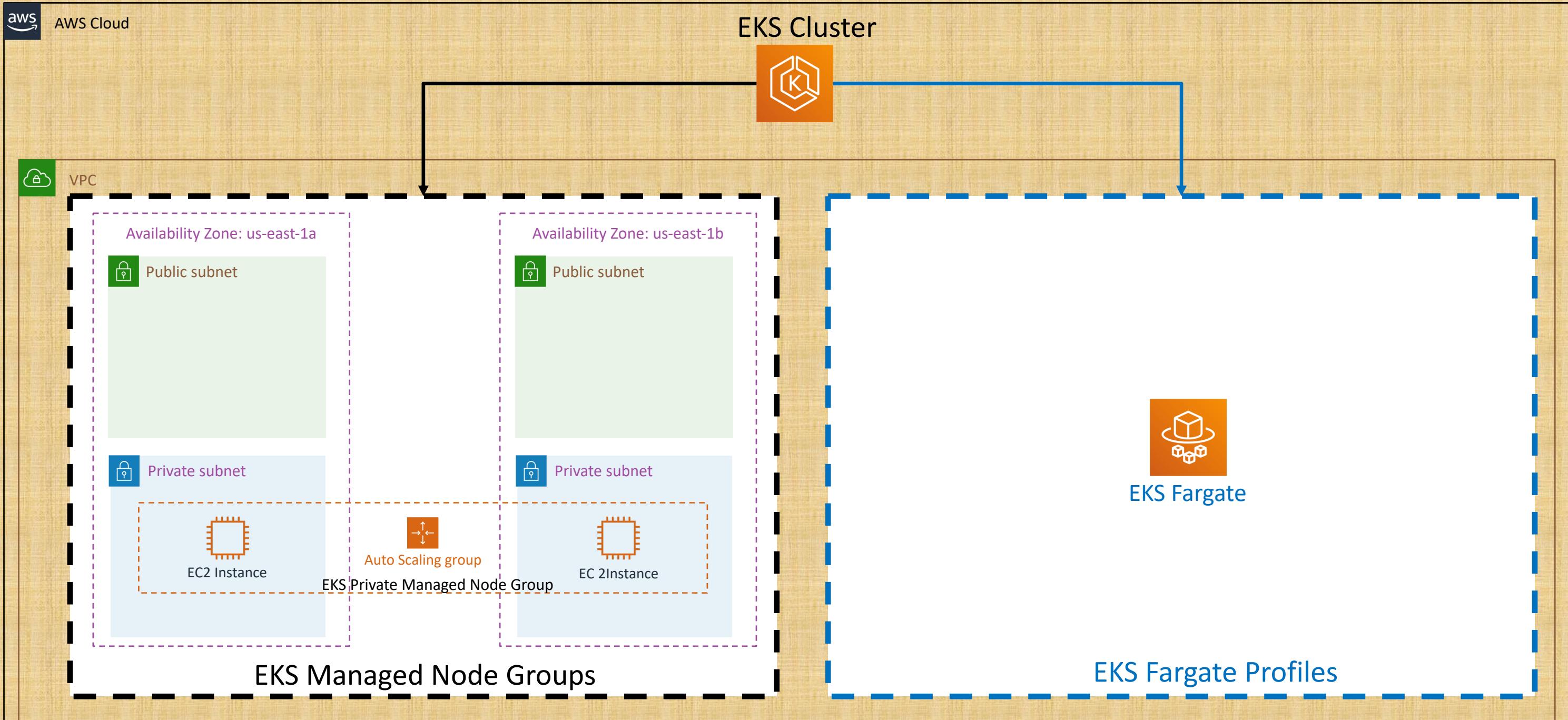


EKS Cluster



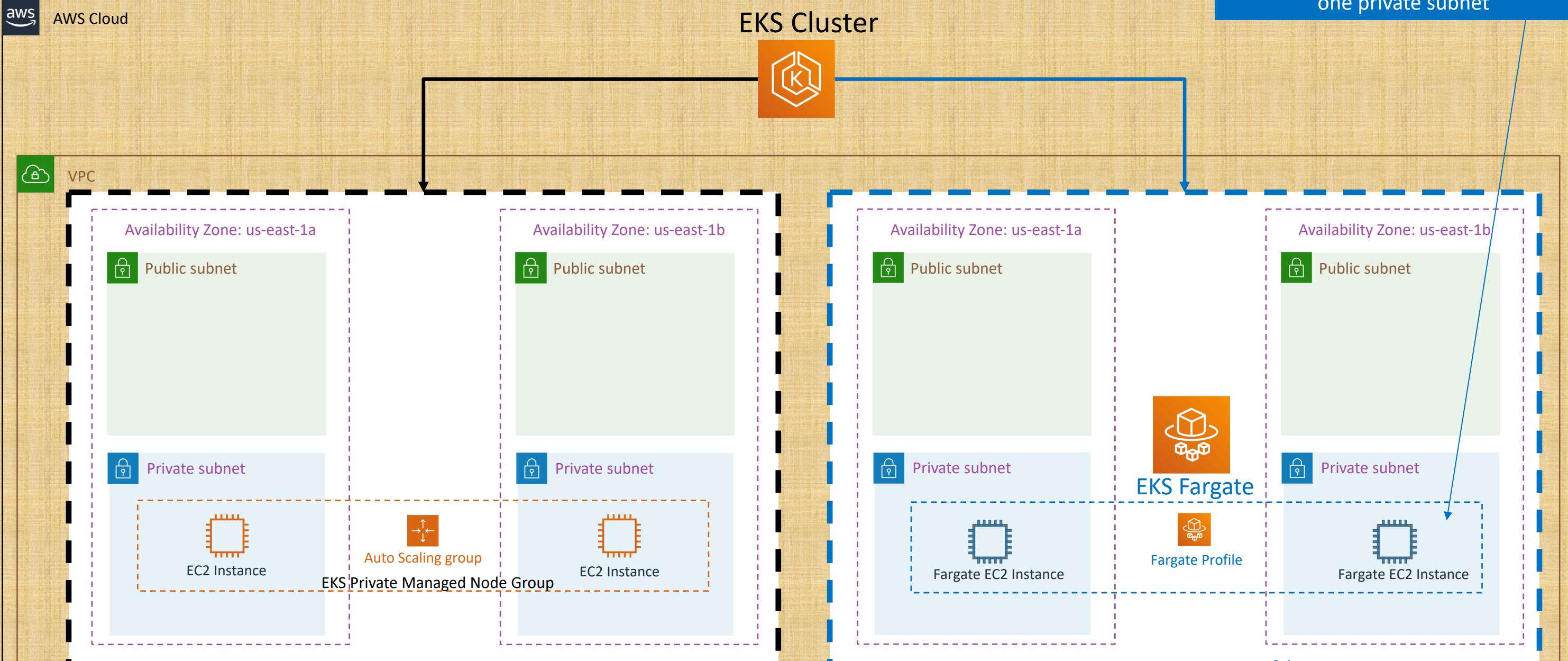
Create Fargate Profiles and Run AWS EKS Workloads on Fargate  
Automate with Terraform

# EKS with Fargate



# EKS with Fargate

Fargate Profiles can be deployed to EKS Cluster only when we have **at least one private subnet**





AWS Cloud



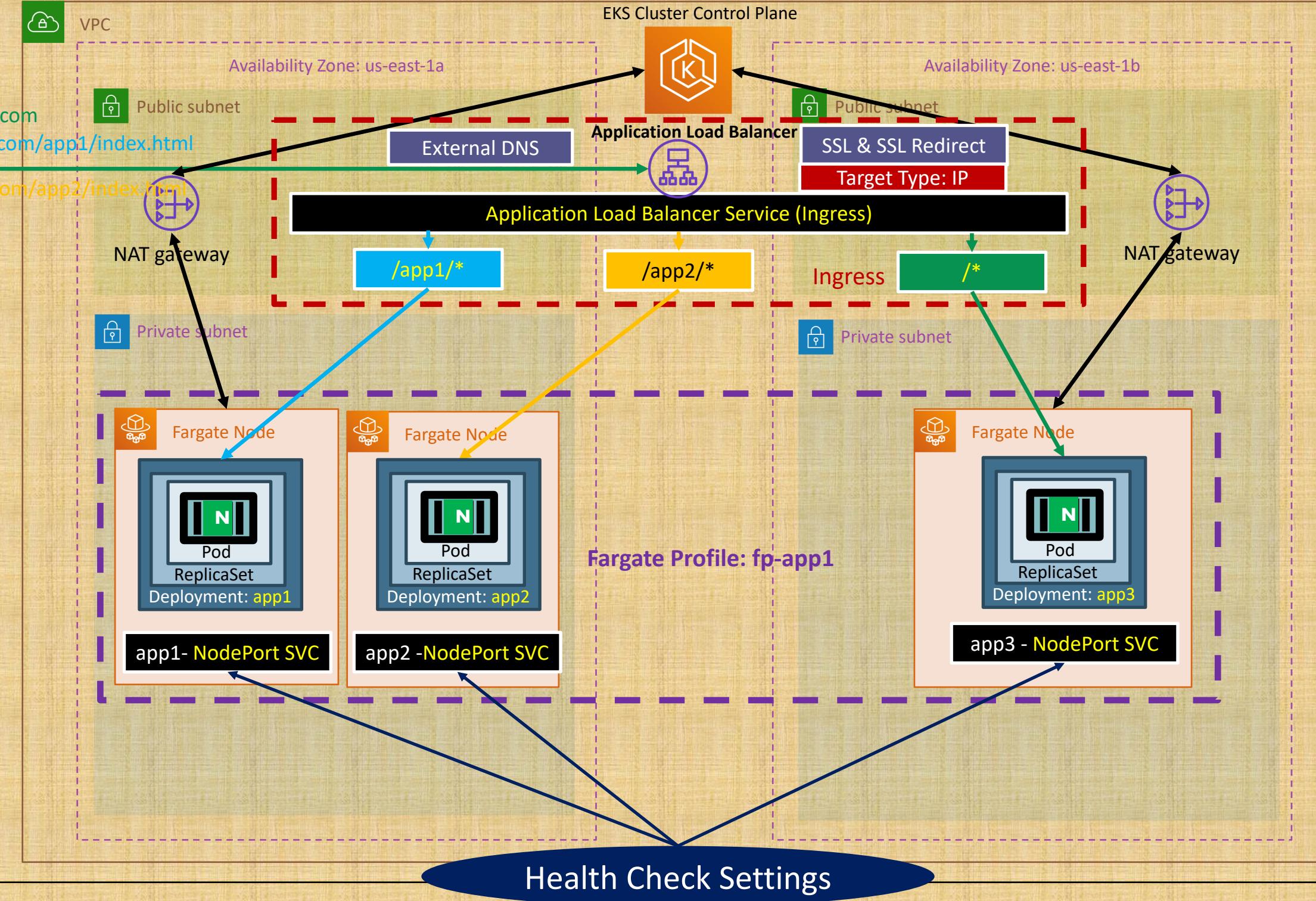
Users

<http://fargate-profile-demo-501.nholuongut.com>

<http://fargate-profile-demo-501.nholuongut.com/app1/index.html>

<http://fargate-profile-demo-501.nholuongut.com/app2/index.html>

Run AWS EKS  
Workloads on  
AWS Fargate



# What are we going to learn ?

```
✓ 43-EKS-Fargate-Profiles
  > 01-ekscluster-terraform-manifests
  > 02-lbc-install-terraform-manifests
  > 03-externaldns-install-terraform-manifests
  ✓ 04-fargate-profiles-terraform-manifests
    ✓ c1-versions.tf
    ✓ c2-remote-state-datasource.tf
    ✓ c3-01-generic-variables.tf
    ✓ c3-02-local-values.tf
    ✓ c4-01-kubernetes-provider.tf
    ✓ c4-02-kubernetes-namespace.tf
    ✓ c5-01-fargate-profile-iam-role-and-policy.tf
    ✓ c5-02-fargate-profile.tf
    ✓ c5-03-fargate-profile-outputs.tf
    ✓ terraform.tfvars
```

**Terraform Manifests**

01

02

03

04

## Project Folders

EKS Cluster Terraform Manifests

AWS Load Balancer Controller Terraform Manifests

External DNS Install Terraform Manifests

Create Fargate Profile with namespace fp-ns-app1

# What are we going to learn ?

- ✓ 43-EKS-Fargate-Profiles
  - > 01-ekscluster-terraform-manifests
  - > 02-lbc-install-terraform-manifests
  - > 03-externaldns-install-terraform-manifests
  - ✓ 04-fargate-profiles-terraform-manifests
    - ↳ c1-versions.tf
    - ↳ c2-remote-state-datasource.tf
    - ↳ c3-01-generic-variables.tf
    - ↳ c3-02-local-values.tf
    - ↳ c4-01-kubernetes-provider.tf
    - ↳ c4-02-kubernetes-namespace.tf
    - ↳ c5-01-fargate-profile-iam-role-and-policy.tf
    - ↳ c5-02-fargate-profile.tf
    - ↳ c5-03-fargate-profile-outputs.tf
    - ↳ terraform.tfvars

Terraform Manifests

## Terraform Manifests

c1 to  
c4-01

c4-02

c5-01

c5-02

c5-03

Terraform Settings, Remote State Datasource,  
Generic Variables, Local Values, Kubernetes Provider

Kubernetes Namespace: fp-ns-app1

Create Fargate Pod Execution **IAM Role** and **IAM Policy**

Create Fargate Profile

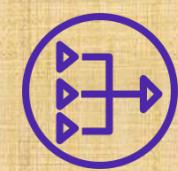
Create Fargate Profile Outputs



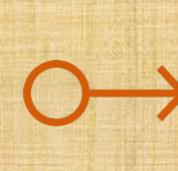
VPC



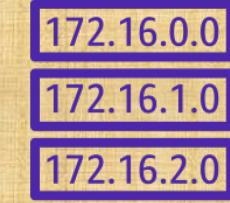
Internet gateway



NAT gateway



Elastic IP address



Route table



Public Subnet



Private Subnet



AWS S3



DynamoDB

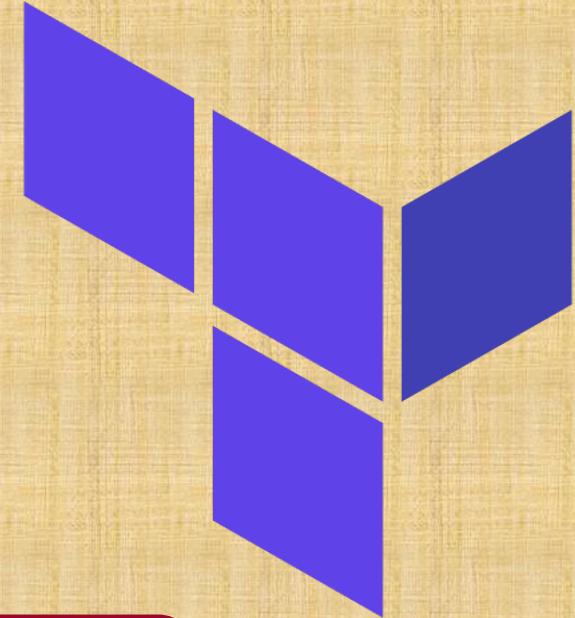


Fargate Profiles

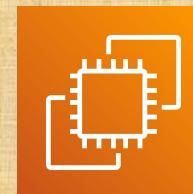


# AWS EKS

## FARGATE



EKS Cluster



EC2 VM



Autoscaling

Run AWS EKS Workloads on AWS Fargate  
Automate with Terraform



AWS Cloud



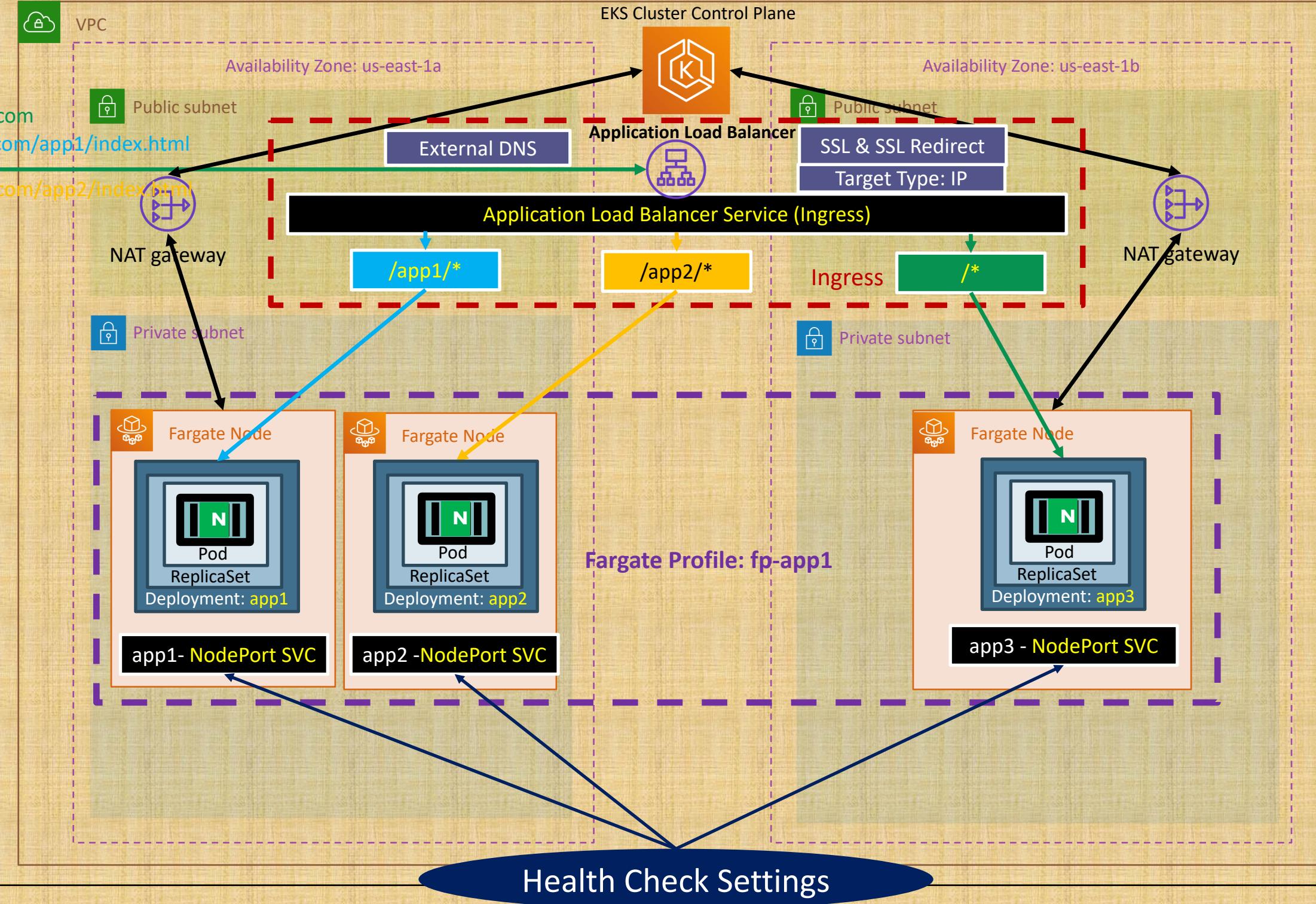
Users

<http://fargate-profile-demo-501.nhoulongut.com>

<http://fargate-profile-demo-501.nhoulongut.com/app1/index.html>

<http://fargate-profile-demo-501.stackimplify.com/app2/index.html>

Run AWS EKS  
Workloads on  
AWS Fargate



# Fargate Profile Namespace - fp-ns-app1

```
# Kubernetes Deployment Manifest
```

```
resource "kubernetes_deployment_v1" "myapp1" {  
  metadata {  
    name = "app1-nginx-deployment"  
    namespace = "fp-ns-app1"
```

```
# Kubernetes Deployment Manifest
```

```
resource "kubernetes_deployment_v1" "myapp2" {  
  metadata {  
    name = "app2-nginx-deployment"  
    namespace = "fp-ns-app1"
```

```
# Kubernetes Deployment Manifest
```

```
resource "kubernetes_deployment_v1" "myapp3" {  
  metadata {  
    name = "app3-nginx-deployment"  
    namespace = "fp-ns-app1"
```

3 k8s Deployments

1 Ingress Service

```
# Kubernetes Service Manifest (Type: Node Port Service)
```

```
resource "kubernetes_service_v1" "myapp1_np_service" {  
  metadata {  
    name = "app1-nginx-nodeport-service"  
    namespace = "fp-ns-app1"
```

```
# Kubernetes Service Manifest (Type: Node Port Service)
```

```
resource "kubernetes_service_v1" "myapp2_np_service" {  
  metadata {  
    name = "app2-nginx-nodeport-service"  
    namespace = "fp-ns-app1"
```

```
# Kubernetes Service Manifest (Type: Node Port Service)
```

```
resource "kubernetes_service_v1" "myapp3_np_service" {  
  metadata {  
    name = "app3-nginx-nodeport-service"  
    namespace = "fp-ns-app1"
```

3 k8s NodePort Svc

All created in  
Namespace fp-ns-app1

```
# Kubernetes Ingress Service Manifest  
resource "kubernetes_ingress_v1" "ingress" {  
  metadata {  
    name = "fargate-profile-demo"  
    namespace = "fp-ns-app1"
```

# Ingress Service - Annotation for Fargate Workloads

## Ingress Service Important Annotation for Fargate Workloads

```
# Target Type: IP (Defaults to Instance if not specified)
"alb.ingress.kubernetes.io/target-type" = "ip"
```

```
✓ 44-EKS-Run-k8s-workloads-on-Fargate
  > 01-ekscluster-terraform-manifests
  > 02-lbc-install-terraform-manifests
  > 03-externaldns-install-terraform-manifests
  > 04-fargate-profiles-terraform-manifests
  ✓ 05-kube-manifests-Run-On-Fargate
    ! 01-Nginx-App1-Deployment-and-ClusterIPService.yml
    ! 02-Nginx-App2-Deployment-and-ClusterIPService.yml
    ! 03-Nginx-App3-Deployment-and-ClusterIPService.yml
    ! 04-ALB-Ingress-target-type-ip.yml
  ✓ 06-run-on-fargate-terraform-manifests
    > listen-ports
    ⚡ c1-versions.tf
    ⚡ c2-remote-state-datasource.tf
    ⚡ c3-providers.tf
    ⚡ c4-kubernetes-app1-deployment.tf
    ⚡ c5-kubernetes-app2-deployment.tf
    ⚡ c6-kubernetes-app3-deployment.tf
    ⚡ c7-kubernetes-app1-nodeport-service.tf
    ⚡ c8-kubernetes-app2-nodeport-service.tf
    ⚡ c9-kubernetes-app3-nodeport-service.tf
    ⚡ c10-kubernetes-ingress-service.tf
    ⚡ c11-acm-certificate.tf
  ⚡ README.md
```

# What are we going to learn ?

## Project Folders

01

EKS Cluster Terraform Manifests

02

AWS Load Balancer Controller Terraform Manifests

03

External DNS Install Terraform Manifests

04

Fargate Profile Terraform Manifests

05

Kubernetes Manifests in **YAML** format  
**Sample Applications (3 Apps):** Kubernetes Deployment, Node Port and Ingress Service

06

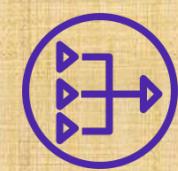
Kubernetes Manifests in **Terraform** format  
**Sample Applications (3 Apps):** Kubernetes Deployment, Node Port Service, Ingress Service and ACM Certificate



VPC



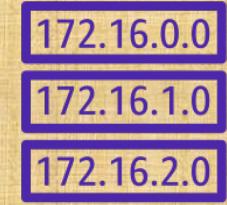
Internet gateway



NAT gateway



Elastic IP address



Route table



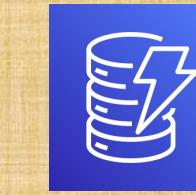
Public Subnet



Private Subnet



AWS S3



DynamoDB

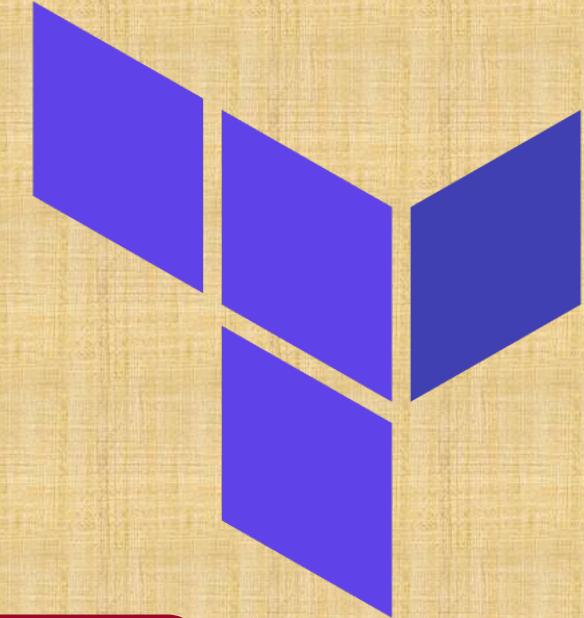


Fargate Profiles

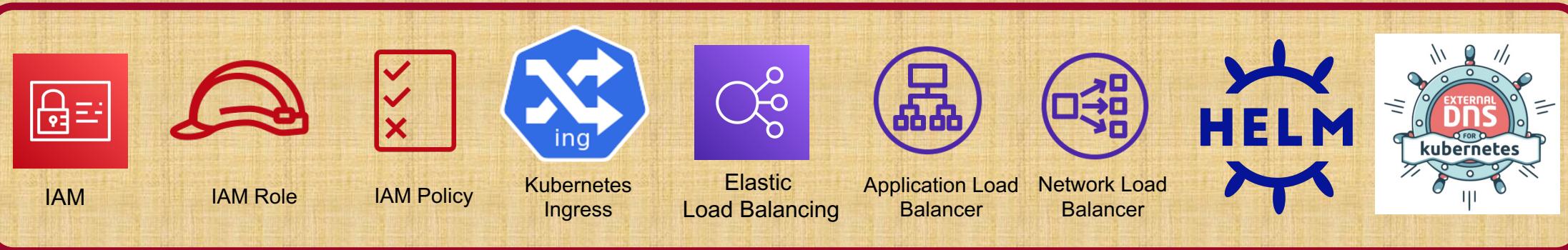


# AWS EKS

## FARGATE



EKS Cluster

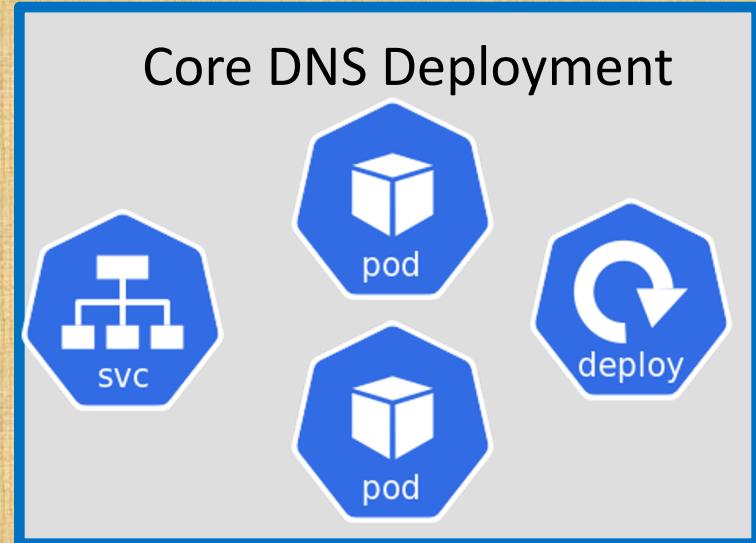


Create FARGATE ONLY EKS Cluster  
Automate with Terraform

# Kubernetes CoreDNS

CoreDNS is a **DNS server** that can serve as the Kubernetes cluster DNS.

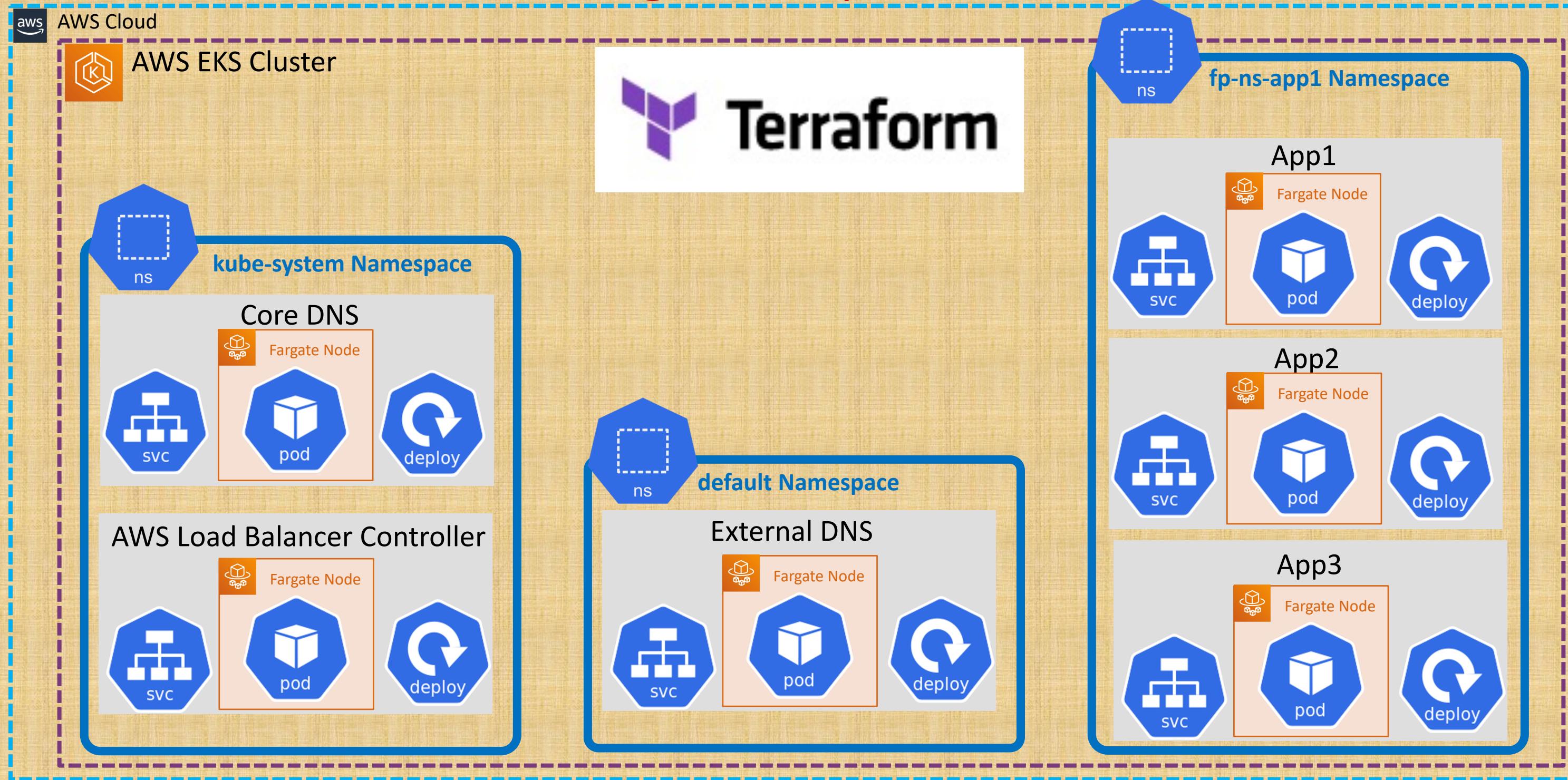
CoreDNS pods provide **name resolution** for all pods in the cluster

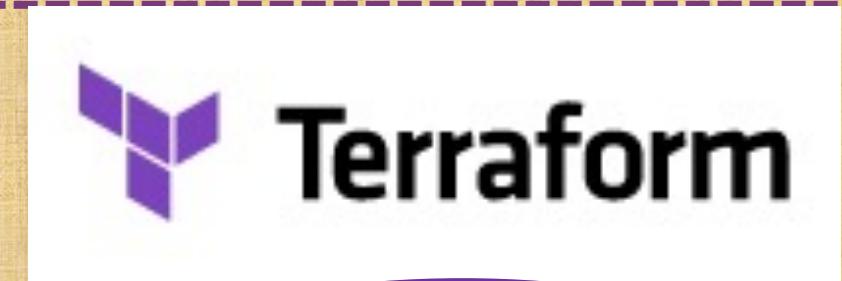


When we create an Amazon EKS cluster with at least **one worker node**, **two replicas** of the CoreDNS image are deployed by default

The CoreDNS pods can be deployed to **Fargate nodes** if your cluster includes an **AWS Fargate profile** with a **namespace** that matches the **namespace** for the CoreDNS deployment. In AWS EKS it is **kube-system namespace**

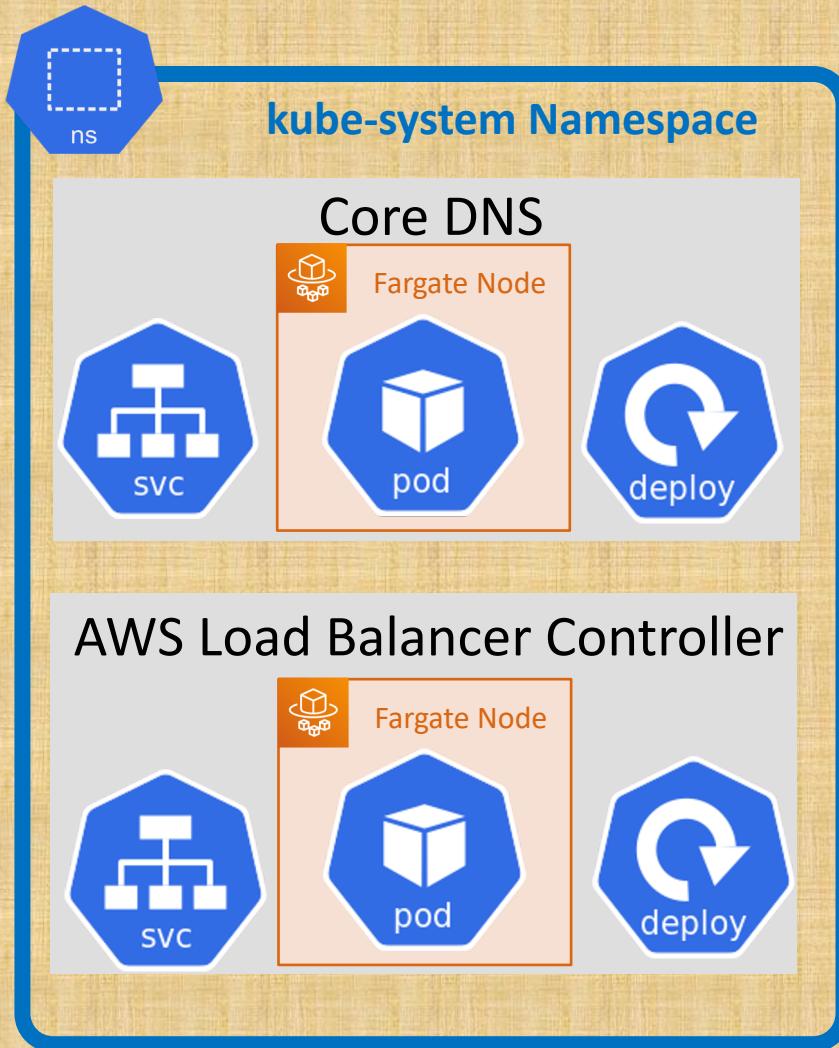
# AWS Fargate Only EKS Cluster



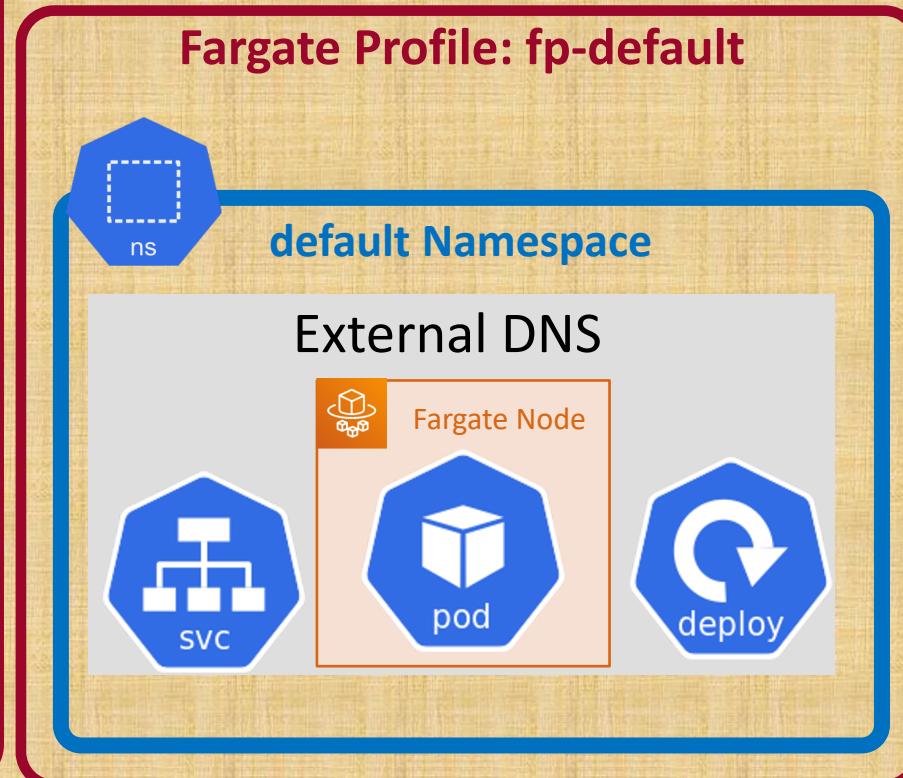


Everything for this  
usecase running on  
AWS Fargate

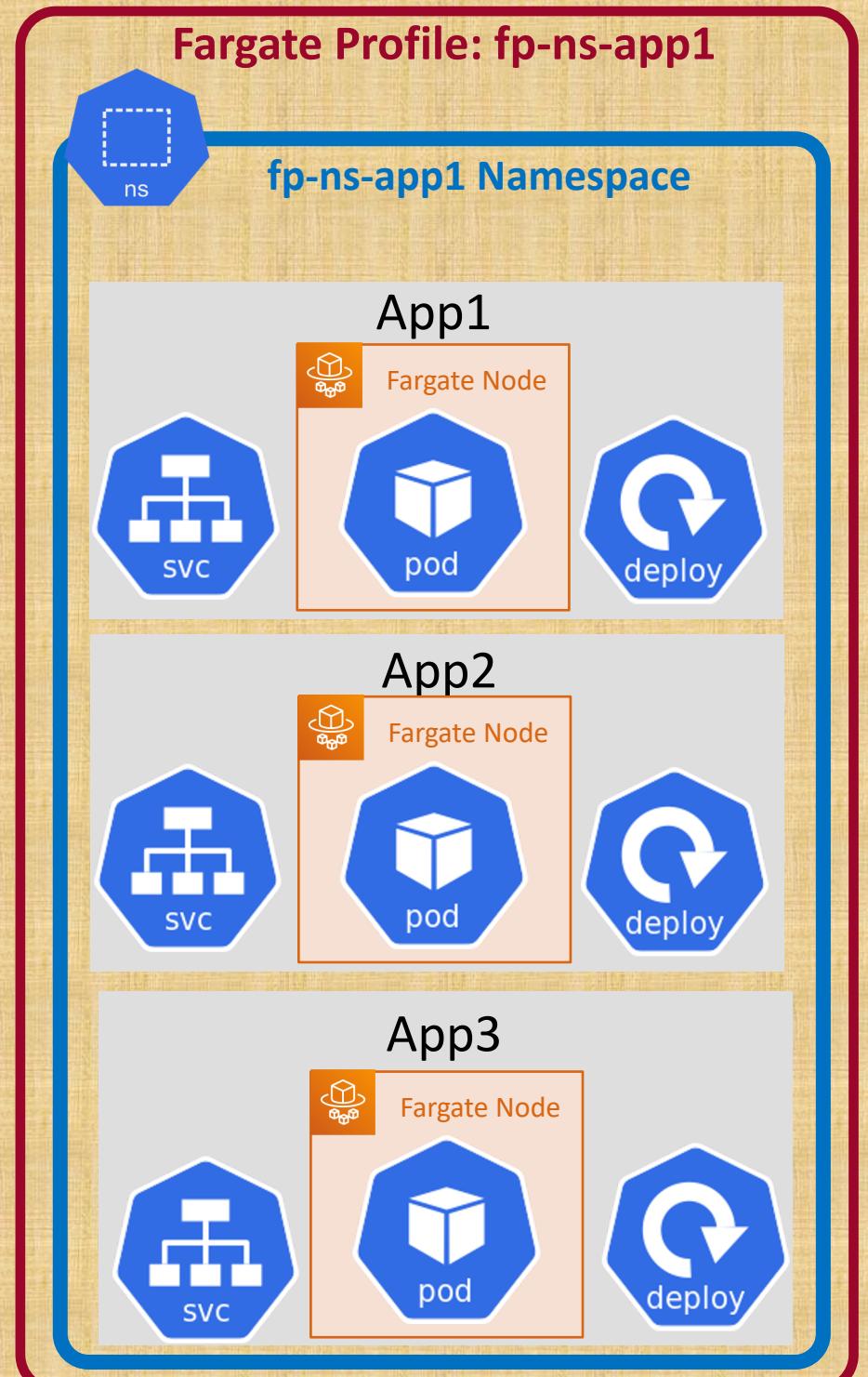
### Fargate Profile: fp-kube-system



### Fargate Profile: fp-default



### Fargate Profile: fp-ns-app1



# AWS Fargate Only EKS Cluster

Node groups (0) <a href="#">Info</a>		<a href="#">Edit</a>	<a href="#">Delete</a>	<a href="#">Add node group</a>					
Group name	▲	Desired size	▼	AMI release version	▼	Launch template	▼	Status	▼
<strong>No node groups</strong>									
This cluster does not have any node groups.									
Nodes that are not part of an Amazon EKS managed node group are not shown in the AWS console.									
<a href="#">Add node group</a>									
Fargate profiles (3) <a href="#">Info</a>		<a href="#">Edit</a>	<a href="#">Delete</a>	<a href="#">Add Fargate profile</a>					
Profile name	▲	Namespaces					Status		
<input type="radio"/> hr-dev-fp-default		default			<span style="color: green;">✓ Active</span>				
<input type="radio"/> hr-dev-fp-kube-system		kube-system			<span style="color: green;">✓ Active</span>				
<input type="radio"/> hr-dev-fp-ns-app1		fp-ns-app1			<span style="color: green;">✓ Active</span>				

# AWS Fargate Only EKS Cluster

**Resource types** X

- ▼ Workloads
  - PodTemplates
  - Pods**
  - ReplicaSets
  - Deployments
  - StatefulSets
  - DaemonSets
  - Jobs
  - CronJobs
  - PriorityClasses
  - HorizontalPodAutoscalers
- ▶ Cluster
- ▶ Service and networking
- ▶ Config and storage
- ▶ Authentication
- ▶ Authorization
- ▶ Policy
- ▶ Extensions

**Workloads: Pods (8)** View details

Pod is the smallest and simplest Kubernetes object. A Pod represents a set of running containers on your cluster.

[Learn more](#)

All Namespaces ▼  < 1 >

Name	Age
app1-nginx-deployment-777cddb9b4-lwwdd	Created 30 minutes ago
app2-nginx-deployment-577bf469f7-4kbr6	Created 30 minutes ago
app3-nginx-deployment-b54c5b6bd-szgqn	Created 30 minutes ago
aws-load-balancer-controller-b69d5c6b7-29wj4	Created 34 minutes ago
aws-load-balancer-controller-b69d5c6b7-6mjhf	Created 34 minutes ago
coredns-677d9c745f-2zjn5	Created 40 minutes ago
coredns-677d9c745f-6tr4h	Created 40 minutes ago
external-dns-6557c6c8d6-mpxwm	Created 32 minutes ago

# What are we going to learn ?

- ✓ 45-Fargate-Only-EKS-Cluster
  - > 01-ekscluster-terraform-manifests
  - > 02-lbc-install-terraform-manifests
  - > 03-externaldns-install-terraform-manifests
  - ✓ 04-run-on-fargate-terraform-manifests
    - > listen-ports
    - c1-versions.tf
    - c2-remote-state-datasource.tf
    - c3-providers.tf
    - c4-kubernetes-app1-deployment.tf
    - c5-kubernetes-app2-deployment.tf
    - c6-kubernetes-app3-deployment.tf
    - c7-kubernetes-app1-nodeport-service.tf
    - c8-kubernetes-app2-nodeport-service.tf
    - c9-kubernetes-app3-nodeport-service.tf
    - c10-kubernetes-ingress-service.tf
    - c11-acm-certificate.tf

01

02

03

04

## Project Folders

### EKS Cluster Terraform Manifests

1. EKS Cluster
2. Fargate Profile for **kube-system** Namespace
3. Fargate Profile for **default** Namespace
4. Fargate Profile for **fp-ns-app1** Namespace
5. Core DNS runs on AWS Fargate

### AWS Load Balancer Controller Terraform Manifests (Runs on AWS Fargate)

### External DNS Install Terraform Manifests (Runs on AWS Fargate)

### Kubernetes Manifests in **Terraform** format **Sample Applications (3 Apps):** Kubernetes Deployment, Node Port Service, Ingress Service and ACM Certificate (Runs on AWS Fargate)

- ✓ 45-Fargate-Only-EKS-Cluster
  - ✓ 01-ekscluster-terraform-manifests
    - > local-exec-output-files
    - > private-key
    - ✗ c1-versions.tf
    - ✗ c2-01-generic-variables.tf
    - ✗ c2-02-local-values.tf
    - ✗ c3-01-vpc-variables.tf
    - ✗ c3-02-vpc-module.tf
    - ✗ c3-03-vpc-outputs.tf
    - ✗ c4-01-eks-variables.tf
    - ✗ c4-02-eks-outputs.tf
    - ✗ c4-03-iamrole-for-eks-cluster.tf
    - ✗ c4-04-eks-cluster.tf
    - ✗ c4-05-fargate-profile-iam-role-and-policy.tf
    - ✗ c4-06-fargate-profile-kube-system-namespace.tf
    - ✗ c4-07-fargate-profile-default-namespace.tf
    - ✗ c4-08-fargate-profile-fp-ns-app1-namespace.tf
    - ✗ c5-01-iam-oidc-connect-provider-variables.tf
    - ✗ c5-02-iam-oidc-connect-provider.tf
    - ✗ eks.auto.tfvars
    - ✗ terraform.tfvars
    - ✗ vpc.auto.tfvars

# What are we going to learn ?

## Terraform Manifests

No Changes

c4-02

c4-05

c4-06

c4-07

c4-08

c1 to c4-01, c4-03, c4-04, c5-01, c5-02 (Same as regular EKS Cluster we learned so far)

Removed Outputs related to EKS Public and Private Node Groups

Create Fargate Pod Execution IAM Role and IAM Policy

Fargate Profile: fp-kube-system with selector as kube-system namespace (For CoreDNS, LBC Controller Pods)

Fargate Profile: fp-default with selector as default namespace (For external-dns pod)

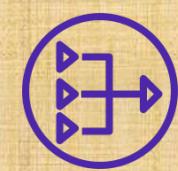
Fargate Profile: fp-ns-app1 with selector as fp-ns-app1 namespace (For 3 Sample App Pods)



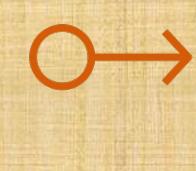
VPC



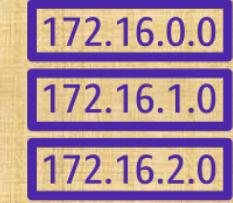
Internet gateway



NAT gateway



Elastic IP address



Route table



Public Subnet



Private Subnet



AWS S3



DynamoDB

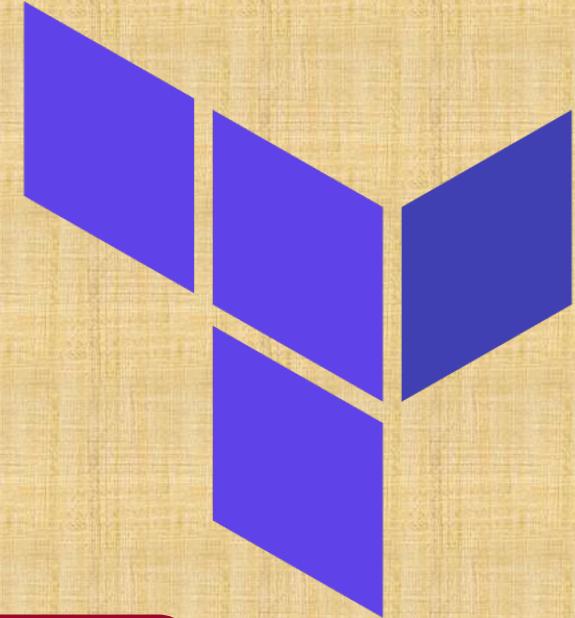


Fargate Profiles

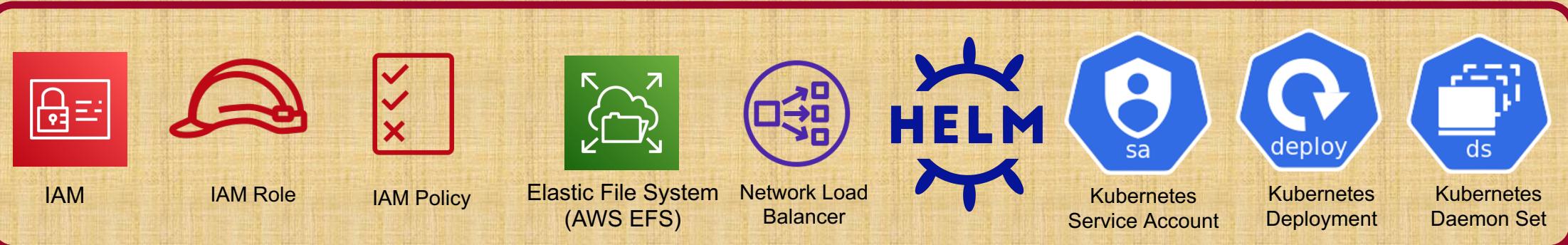


# AWS EKS

## EFS CSI Driver



EKS Cluster



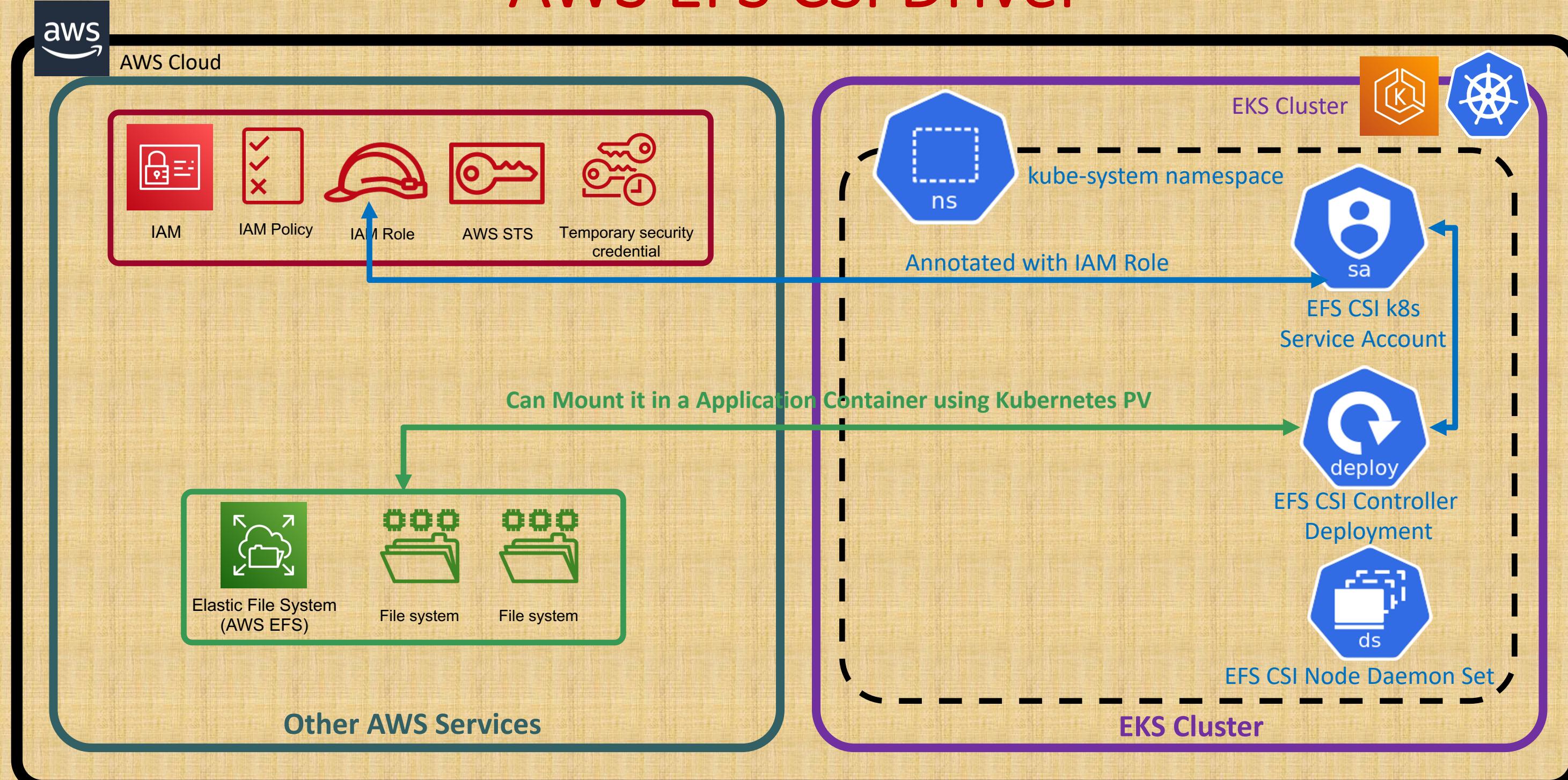
**AWS EFS CSI Driver Install  
Automate with Terraform**

# AWS EFS CSI Driver

Why do we need to use EFS CSI Driver in EKS Cluster?

AWS EFS Container Storage Interface (CSI) driver allows EKS clusters to manage the lifecycle of EFS File System in Kubernetes

# AWS EFS CSI Driver



## AWS EFS CSI Driver supports

Static Provisioning

Dynamic Provisioning

In both cases, we need to manually create the AWS EFS File System and Mount Targets first. In other words, EFS CSI Driver will not be able to create EFS File System

With Static Provisioning it can be mounted (EFS Mount Targets) inside a container as a persistent volume (PV) using the EFS CSI driver

Uses a persistent volume claim (PVC) to dynamically provision a persistent volume (PV)

On Creating a PVC, kubernetes requests EFS to create an Access Point in a file system which will be used to mount the PV

EFS File System ID should be provide as input to k8s Persistent Volume

EFS File System ID should be provide as input to k8s Storage Class

# AWS EFS CSI Driver

The following CSI interfaces are implemented in EFS CSI Driver

Controller Service

CreateVolume, DeleteVolume, ControllerGetCapabilities,  
ValidateVolumeCapabilities

Node Service

NodePublishVolume, NodeUnpublishVolume,  
NodeGetCapabilities, NodeGetInfo, NodeGetId,  
NodeGetVolumeStats

Identity Service

GetPluginInfo, GetPluginCapabilities, Probe

```
✓ 46-EKS-EFS-CS-Install  
  ✓ 01-ekscluster-terraform-manifests  
    > local-exec-output-files  
    > private-key  
    ✎ c1-versions.tf  
    ✎ c2-01-generic-variables.tf  
    ✎ c2-02-local-values.tf  
    ✎ c3-01-vpc-variables.tf  
    ✎ c3-02-vpc-module.tf  
    ✎ c3-03-vpc-outputs.tf
```

.....

```
✎ c5-08-eks-node-group-private.tf  
✎ c6-01-iam-oidc-connect-provider-variables.tf  
✎ c6-02-iam-oidc-connect-provider.tf  
✎ c7-01-kubernetes-provider.tf  
✎ c7-02-kubernetes-configmap.tf  
✎ c8-01-iam-admin-user.tf
```

# What are we going to learn ?

Add EKS Cluster as  
OpenID Connect  
Provider in AWS IAM  
Service

# What are we going to learn ?

```
✓ 46-EKS-EFS-CSI-Install
  > 01-ekscluster-terraform-manifests
  ✓ 02-efs-install-terraform-manifests
    ✓ c1-versions.tf
    ✓ c2-remote-state-datasource.tf
    ✓ c3-01-generic-variables.tf
    ✓ c3-02-local-values.tf
    ✓ c4-01-efs-csi-datasources.tf
    ✓ c4-02-efs-csi-iam-policy-and-role.tf
    ✓ c4-03-efs-helm-provider.tf
    ✓ c4-04-efs-csi-install.tf
    ✓ c4-05-efs-outputs.tf
    ✓ terraform.tfvars
```

p2-c4-05

Create EFS Outputs

p1

p2-c1

p2-c2

p2-c3

p2-c4-01

p2-c4-02

p2-c4-03

p2-c4-04

## Terraform Manifests

EKS Cluster Terraform Manifests

Create Terraform Settings Block State Storage: S3  
Backend, State Locking: DynamoDB Table

Terraform **Remote State Datasource** to access **Project-01** EKS Cluster data in Project-02

Terraform Input Variables and Local Values

Terraform **HTTP Datasource** to download the **EFS CSI IAM Policy**

Create EFS CSI **IAM Policy** and **IAM Role**

Define Terraform **Helm Provider**

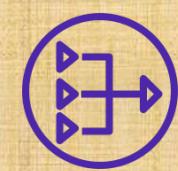
Create Terraform **Helm Release Resource** to install EFS CSI Driver



VPC



Internet gateway



NAT gateway



Elastic IP address



Route table



Public Subnet



Private Subnet



AWS S3



DynamoDB

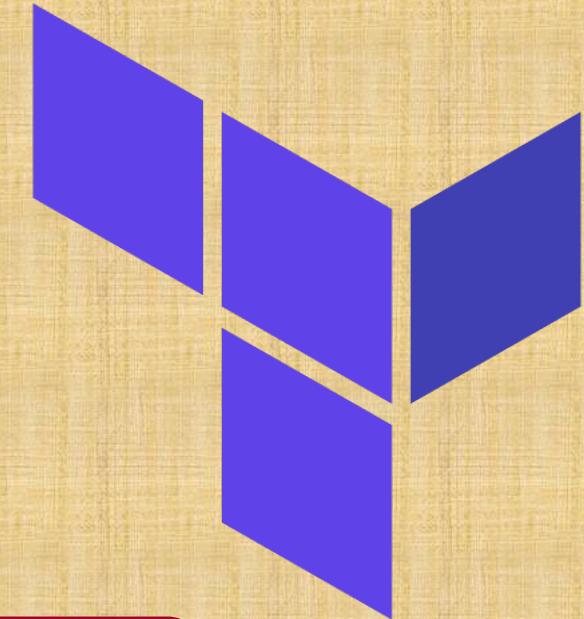


Fargate Profiles

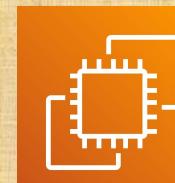
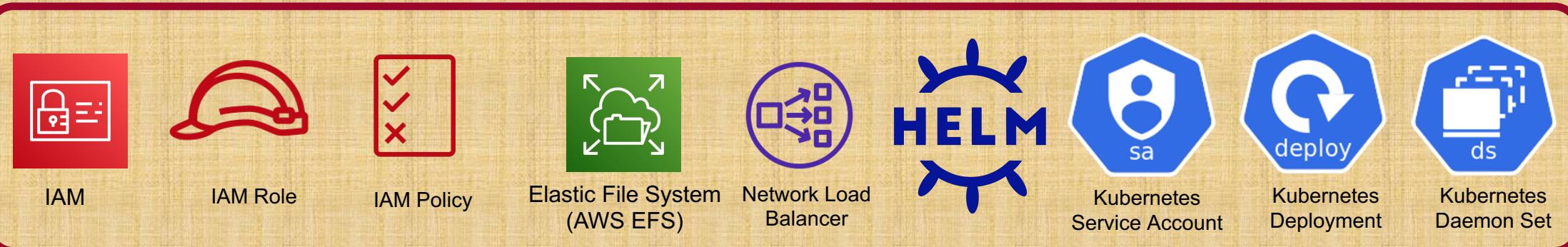


# AWS EKS

## EFS CSI Driver



EKS Cluster



EC2 VM



Autoscaling

**AWS EFS Static Provisioning**  
Automate with Terraform

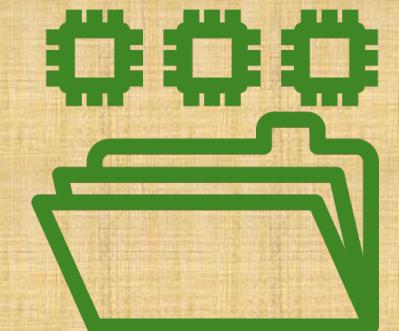
# AWS EKS Storage: EFS Static Provisioning

EFS file system needs to be created manually first, then it could be mounted inside container as a persistent volume (PV) using the EFS CSI driver.



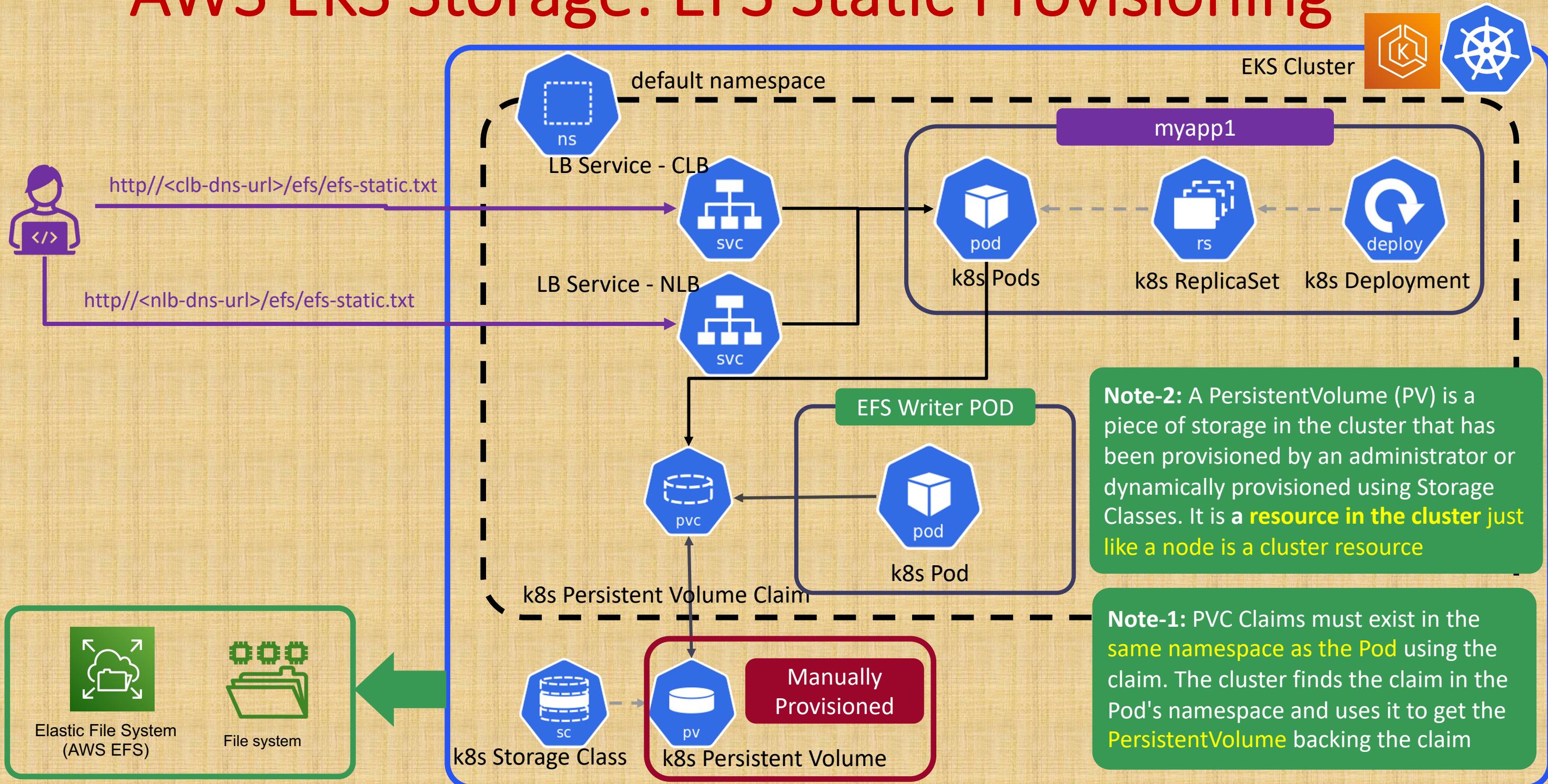
Elastic File System  
(AWS EFS)

In short, EFS CSI Driver cannot create EFS File system Resource, it can only manage a pre-created EFS File System (unlike EBS CSI Driver)

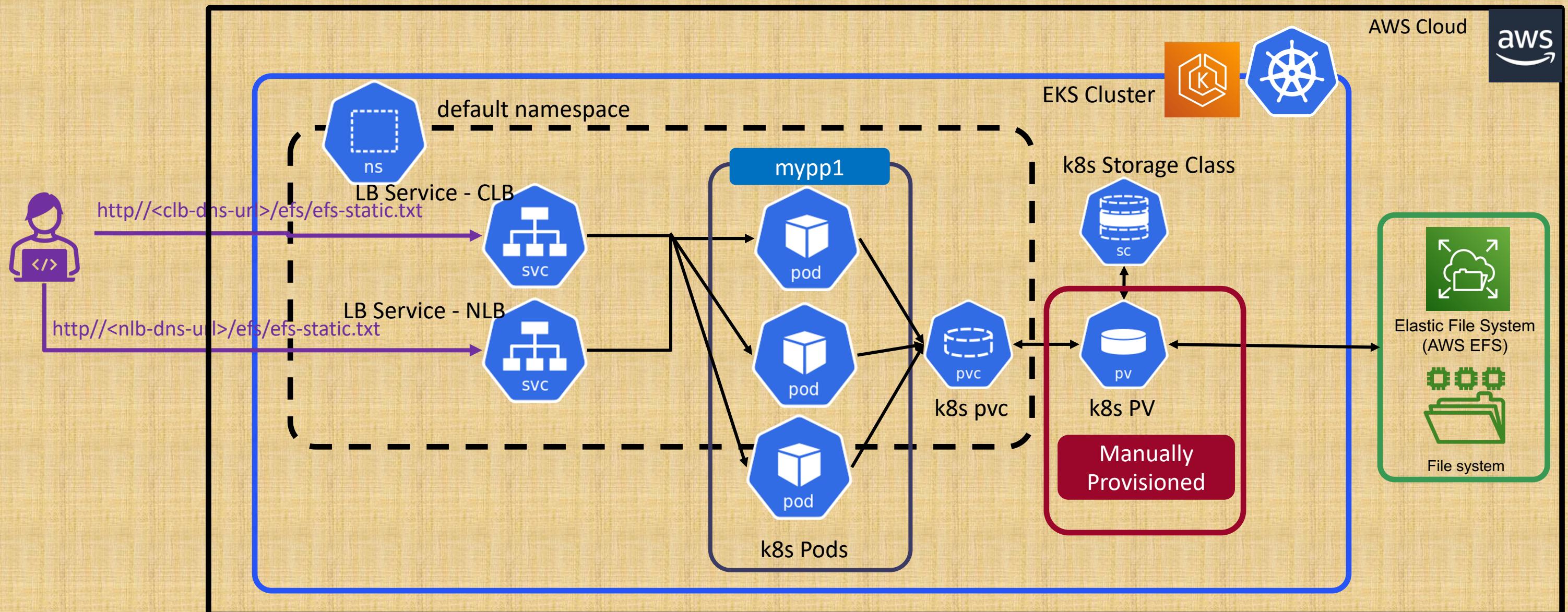


EFS File system

# AWS EKS Storage: EFS Static Provisioning



# AWS EKS Storage: EFS Static Provisioning



Request to any pod will serve the **same content** from EFS File system (Context: /efs/efs-static.txt)

# What are we going to learn ?

## Terraform Manifests

- ✓ 47-EKS-EFS-Static-Provisioning
  - > 01-ekscluster-terraform-manifests
  - > 02-efs-install-terraform-manifests
  - ✓ 03-efs-static-prov-terraform-manifests
    - └ c1-versions.tf
    - └ c2-remote-state-datasource.tf
    - └ c3-providers.tf
    - └ c4-01-storage-class.tf
    - └ c4-02-persistent-volume-claim.tf
    - └ c4-03-persistent-volume.tf
    - └ c5-write-to-efs-pod.tf
    - └ c6-01-myapp1-deployment.tf
    - └ c6-02-myapp1-loadbalancer-service.tf
    - └ c6-03-myapp1-network-loadbalancer-service.tf

p3-c6-02

Create a myapp1 k8s service of type Load Balancer

p3-c6-03

Create a myapp1 k8s service of type Network Load Balancer

p1

p2

p3-  
c1,c2,c3

p3-c4-01

p3-c4-02

p3-c4-03

p3-c5

p3-c6-01

EKS Cluster Terraform Manifests

EFS CSI Driver **Install** Terraform Manifests

Define Terraform Settings Block, Remote State Datasource, Kubernetes Provider

Create Kubernetes **Storage Class**

Create Kubernetes **Persistent Volume Claim**

Create Kubernetes **Persistent Volume** (requires EFS File System ID)

Create a **write to EFS k8s Pod** with k8s **Volume** and **Volume Mounts**

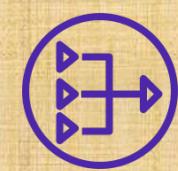
Create a **myapp1 deployment** with k8s **Volume** and **Volume Mounts**



VPC



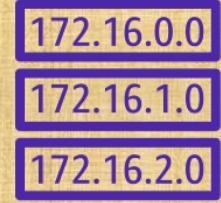
Internet gateway



NAT gateway



Elastic IP address



Route table



Public Subnet



Private Subnet



AWS S3



DynamoDB

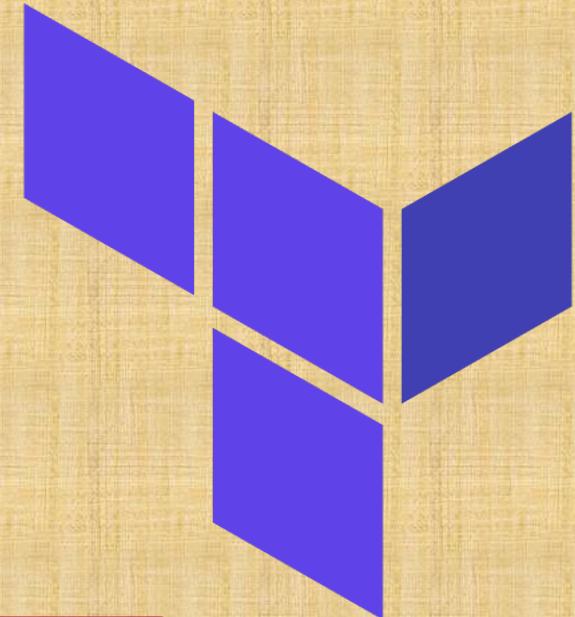


Fargate Profiles

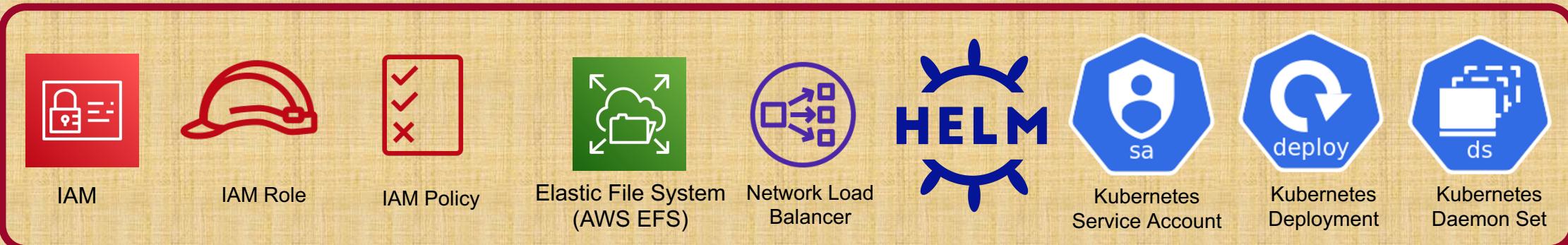


# AWS EKS

## EFS CSI Driver



EKS Cluster



Author: Nho Luong

Skill: DevOps Engineer Lead

**AWS EFS Dynamic Provisioning**  
**Automate with Terraform**

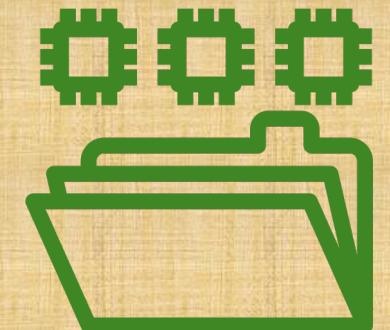
# AWS EKS Storage: EFS Dynamic Provisioning

EFS file system needs to be created manually first. Uses a **persistent volume claim (PVC)** to dynamically provision a **persistent volume (PV)**.



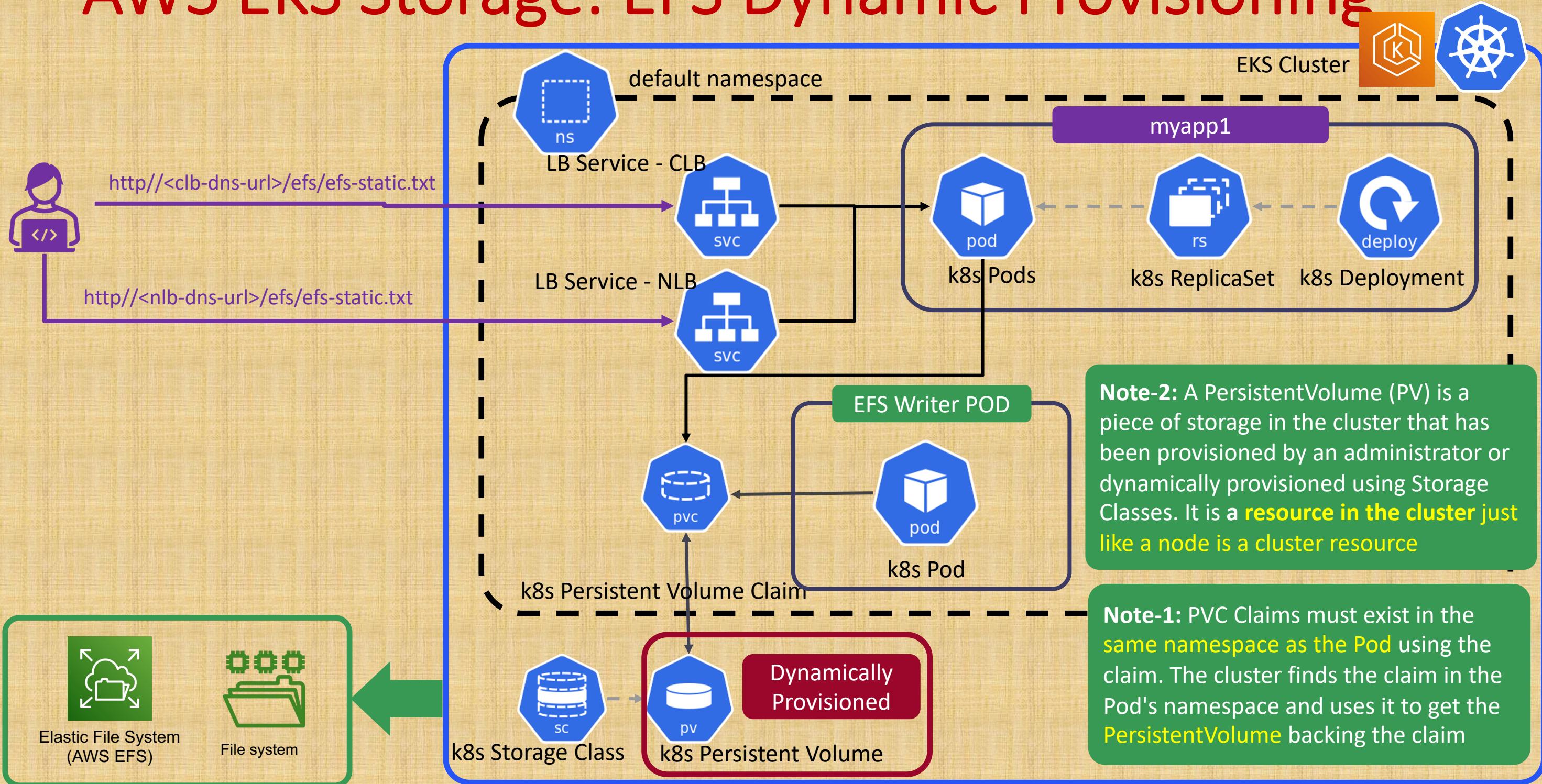
Elastic File System  
(AWS EFS)

On Creating a PVC, kubernetes requests EFS to create an Access Point in a **EFS file system** which will be used to mount the PV.

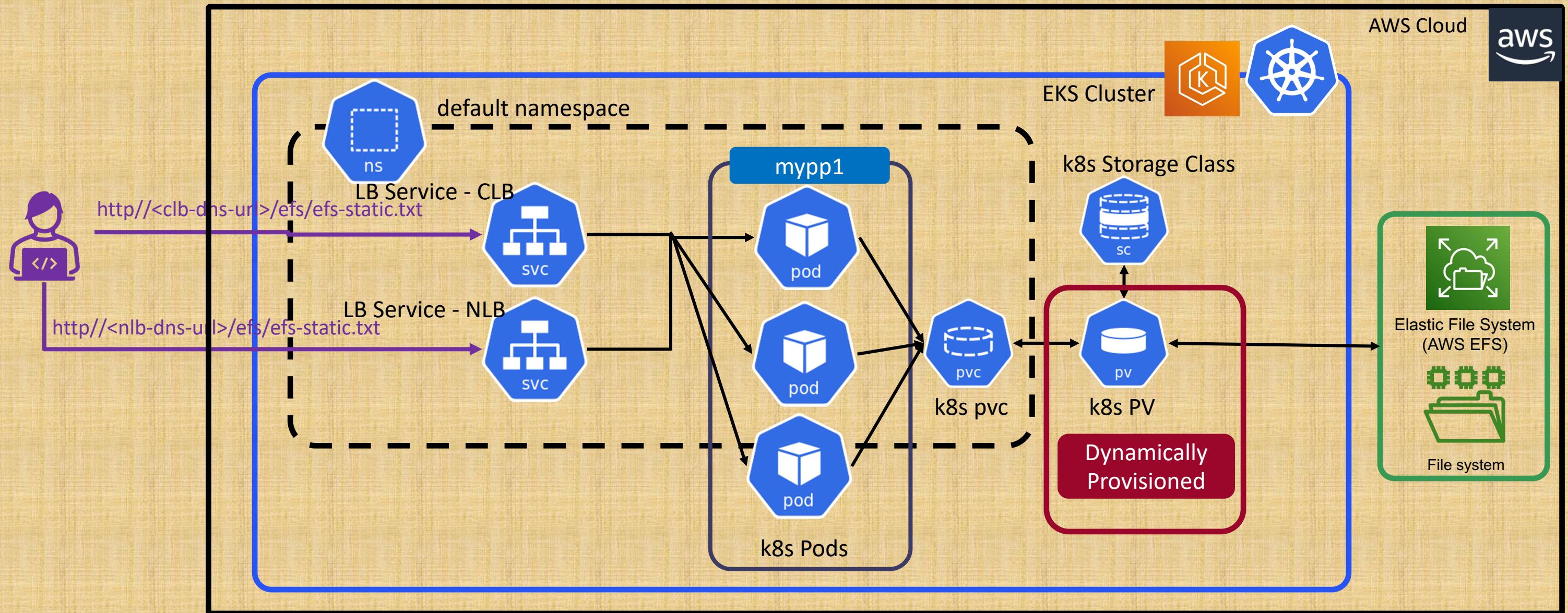


EFS File system

# AWS EKS Storage: EFS Dynamic Provisioning



# AWS EKS Storage: EFS Dynamic Provisioning

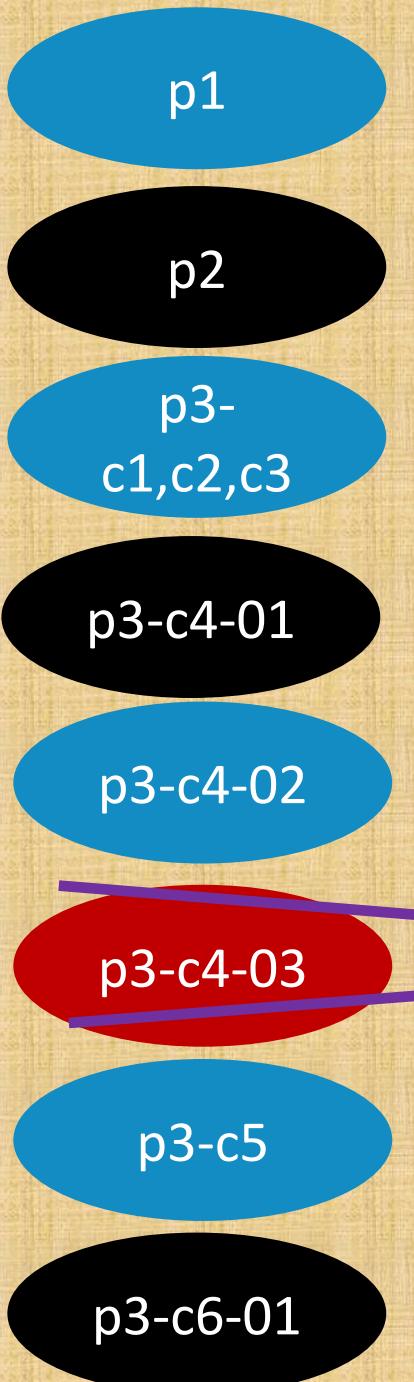


Request to any pod will serve the same content from EFS File system (Context: /efs/efs-static.txt)

# What are we going to learn ?

## Terraform Manifests

- ✓ 48-EKS-EFS-Dynamic-Provisioning
  - > 01-ekscluster-terraform-manifests
  - > 02-efs-install-terraform-manifests
  - ✓ 03-efs-dynamic-prov-terraform-manifests
    - └ c1-versions.tf
    - └ c2-remote-state-datasource.tf
    - └ c3-providers.tf
    - └ c4-01-storage-class.tf
    - └ c4-02-persistent-volume-claim.tf
    - └ c5-write-to-efs-pod.tf
    - └ c6-01-myapp1-deployment.tf
    - └ c6-02-myapp1-loadbalancer-service.tf
    - └ c6-03-myapp1-network-loadbalancer-service.tf



EKS Cluster Terraform Manifests

efs CSI Driver **Install** Terraform Manifests

Define Terraform Settings Block, Remote State Datasource, Kubernetes Provider

Create Kubernetes Storage Class (requires EFS File System ID)

Create Kubernetes Persistent Volume Claim

~~Create Kubernetes Persistent Volume (requires EFS File System ID)~~

Create a **write to EFS k8s Pod** with k8s Volume and Volume Mounts

Create a **myapp1 deployment** with k8s Volume and Volume Mounts

p3-c6-02

Create a **myapp1** k8s service of type **Load Balancer**

p3-c6-03

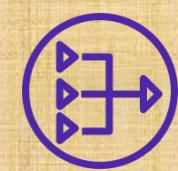
Create a **myapp1** k8s service of type **Network Load Balancer**



VPC



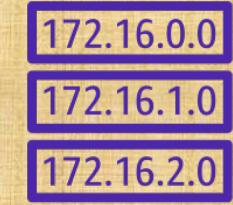
Internet gateway



NAT gateway



Elastic IP address



Route table



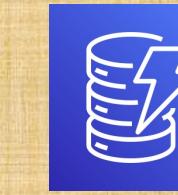
Public Subnet



Private Subnet



AWS S3



DynamoDB

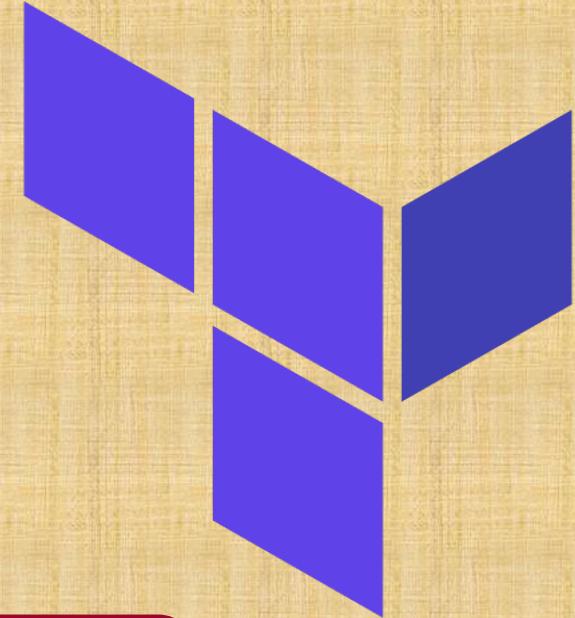


Fargate Profiles

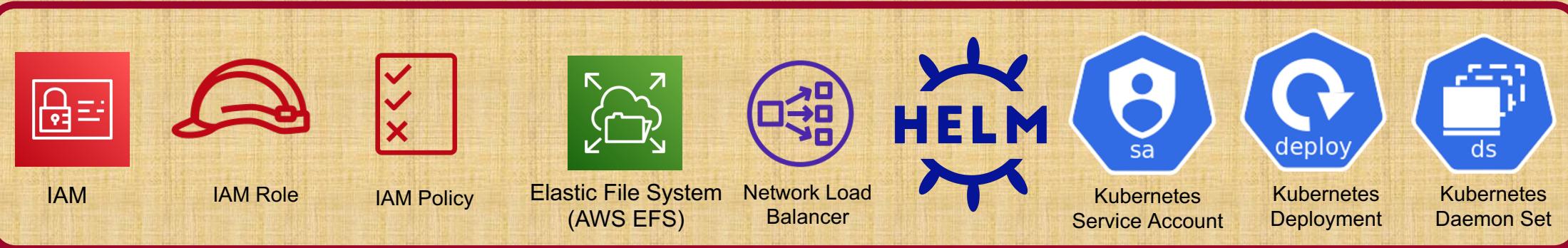


# AWS EKS

## EFS CSI Driver



EKS Cluster



Autoscaling

**AWS EFS for workloads running on AWS Fargate**  
**Automate with Terraform**

# AWS EFS File System Mount for AWS Fargate Workloads

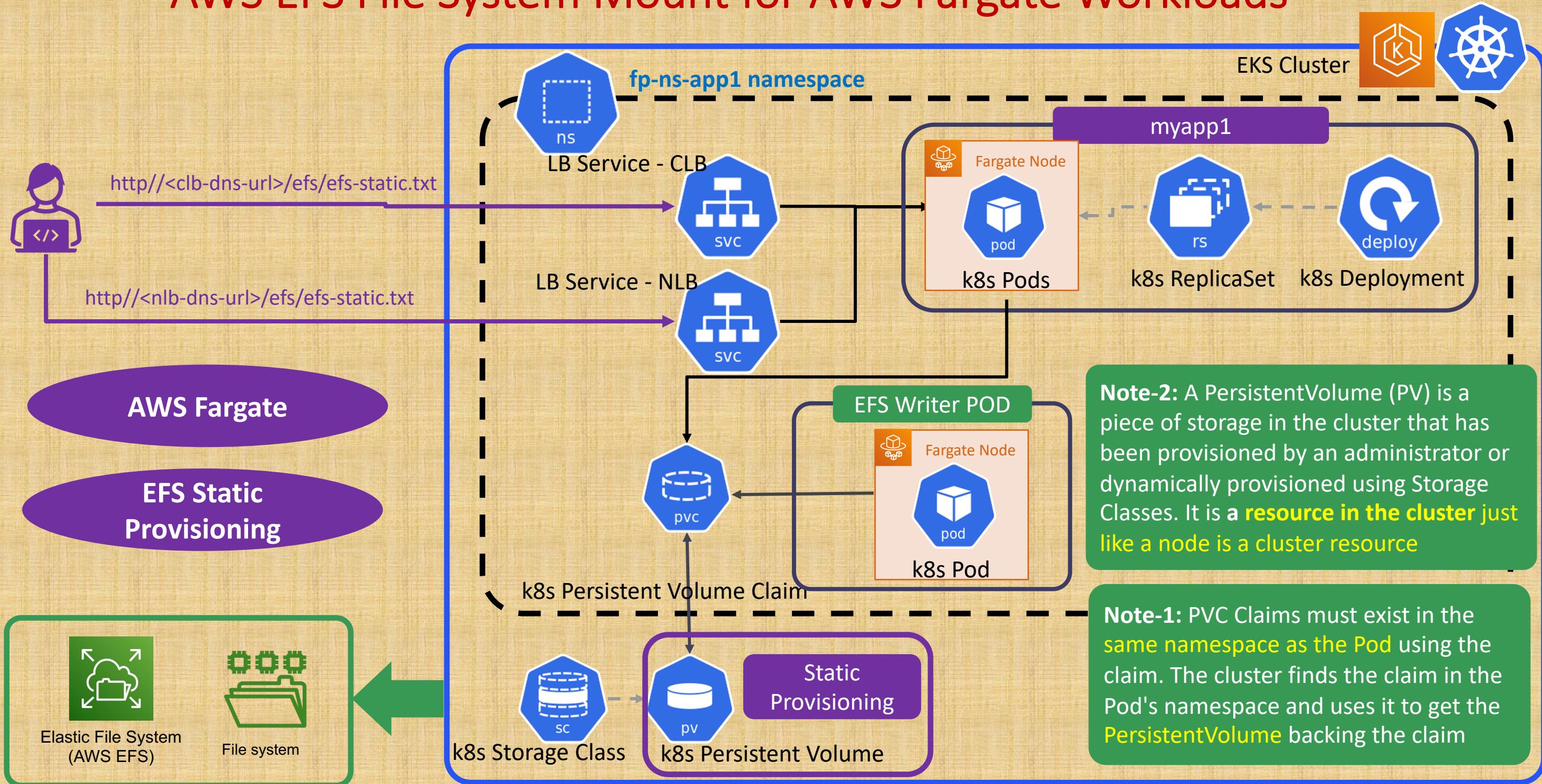
**Usecase: AWS EFS File System mount for Kubernetes Workloads  
running on AWS Fargate**

AWS EFS **Static**  
Provisioning

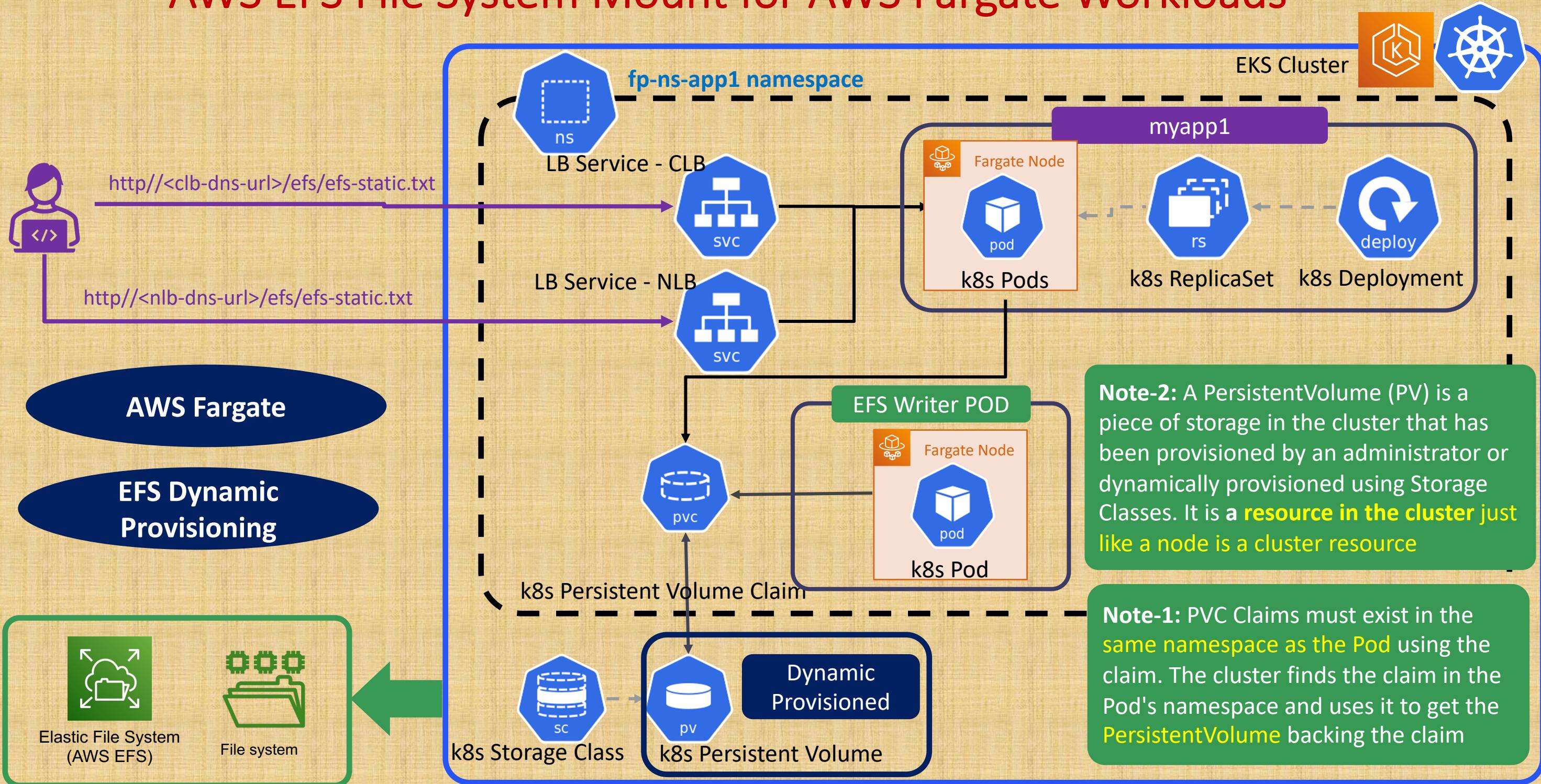
AWS EFS **Dynamic**  
Provisioning

We are going to test **both** the scenarios on AWS Fargate

# AWS EFS File System Mount for AWS Fargate Workloads

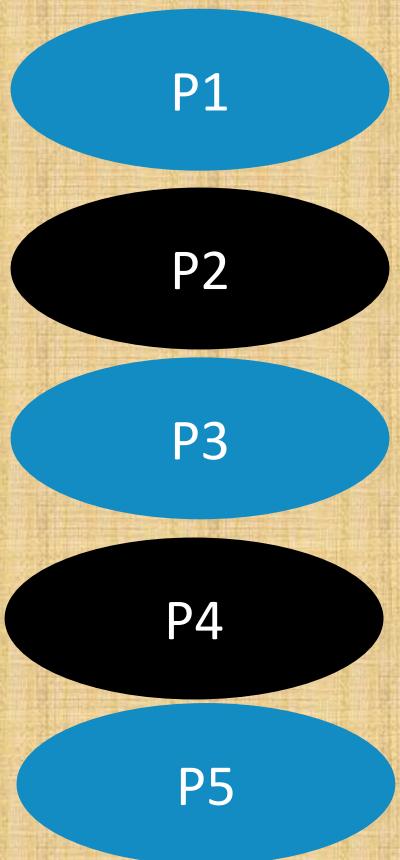


# AWS EFS File System Mount for AWS Fargate Workloads



# What are we going to learn ?

- ✓ 49-EKS-EFS-Fargate
  - > 01-ekscluster-terraform-manifests
  - > 02-efs-install-terraform-manifests
  - > 03-fargate-profiles-terraform-manifests
  - > 04-efs-static-prov-terraform-manifests
  - > 05-efs-dynamic-prov-terraform-manifests



## Terraform Manifests

EKS Cluster Terraform Manifests

EFS CSI Driver **Install** Terraform Manifests

Create a **Fargate Profile** and namespace fp-ns-app1

EFS **Static Provisioning** Terraform Manifests  
(Deploy workloads on Fargate)

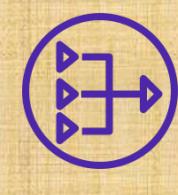
EFS **Dynamic Provisioning** Terraform Manifests  
(Deploy workloads on Fargate)



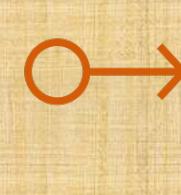
VPC



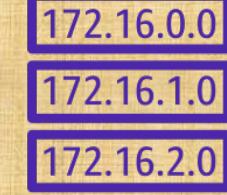
Internet gateway



NAT gateway



Elastic IP address



Route table



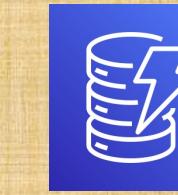
Public Subnet



Private Subnet



AWS S3



DynamoDB

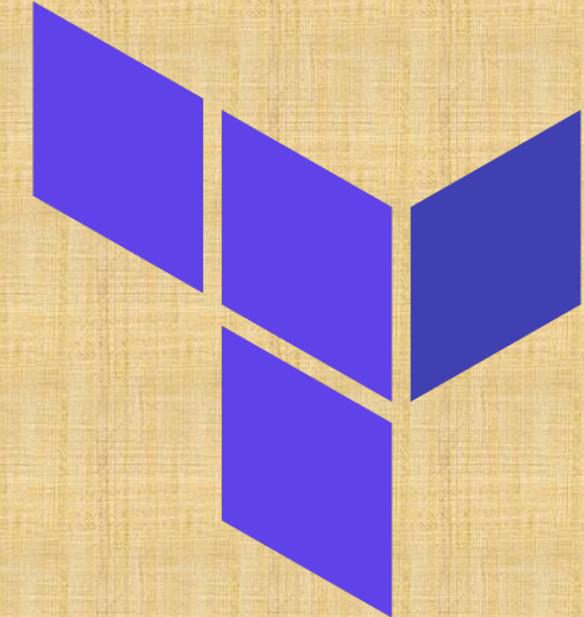


Fargate Profiles

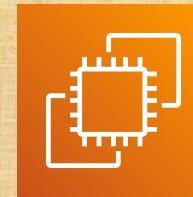


# AWS EKS

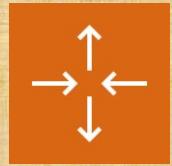
## Cluster Autoscaler



EKS Cluster



EC2 VM



Autoscaling

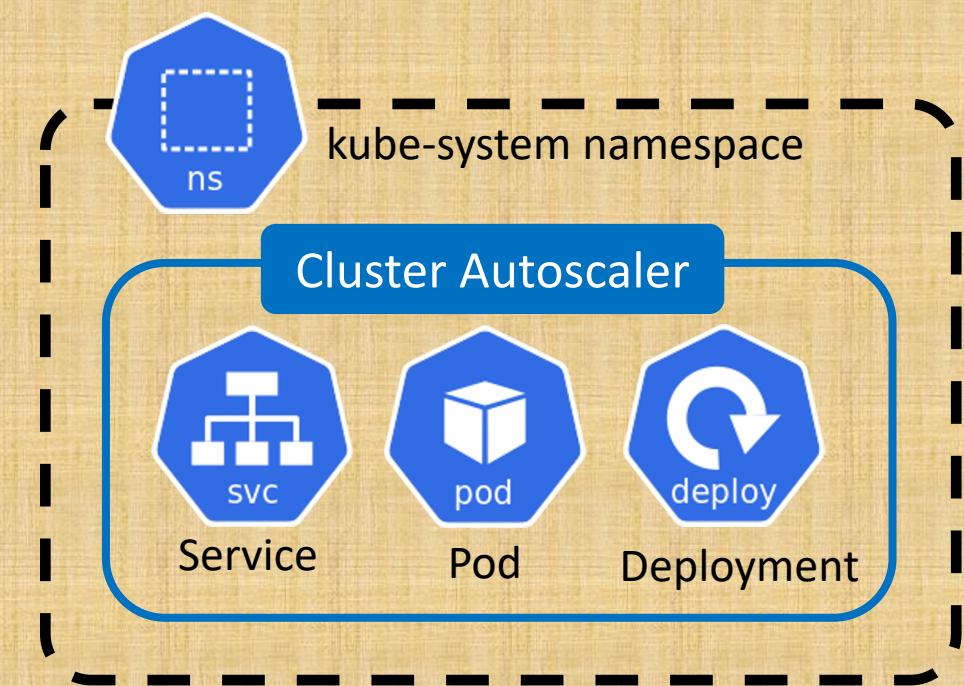
Kubernetes Cluster Autoscaler Install using Helm Provider  
Automate with Terraform

# Cluster Autoscaler

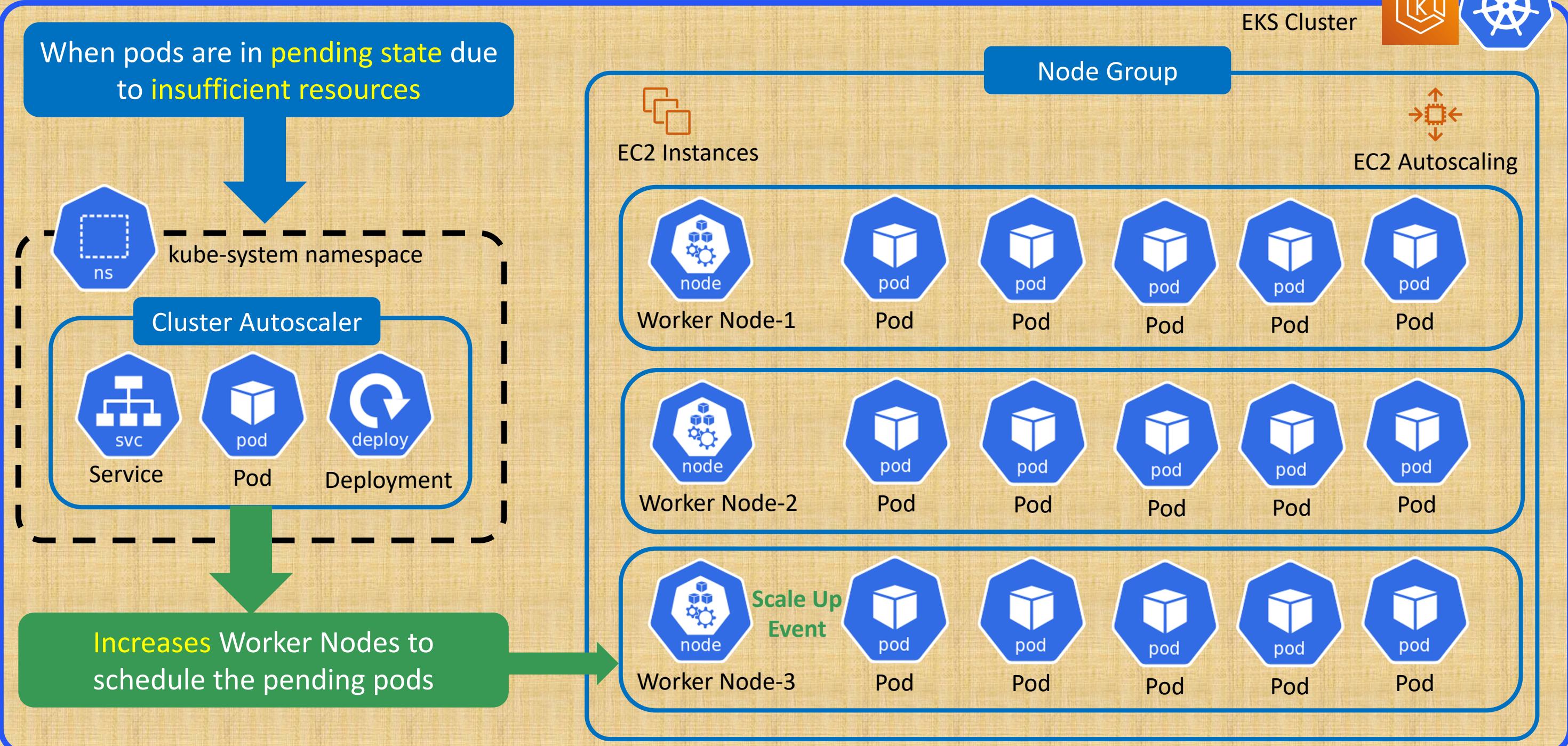
Cluster Autoscaler (CA) is responsible for ensuring that our cluster has **enough nodes to schedule your pods** without wasting resources.

**Scale Up Event:** CA watches for pods in **pending** state due to **insufficient resources** and create **new worker nodes** and **schedule** pods on those worker nodes

**Scale In Event:** CA watches for nodes which are **underutilized** and terminate those nodes. That said, it saves wastage of resources.

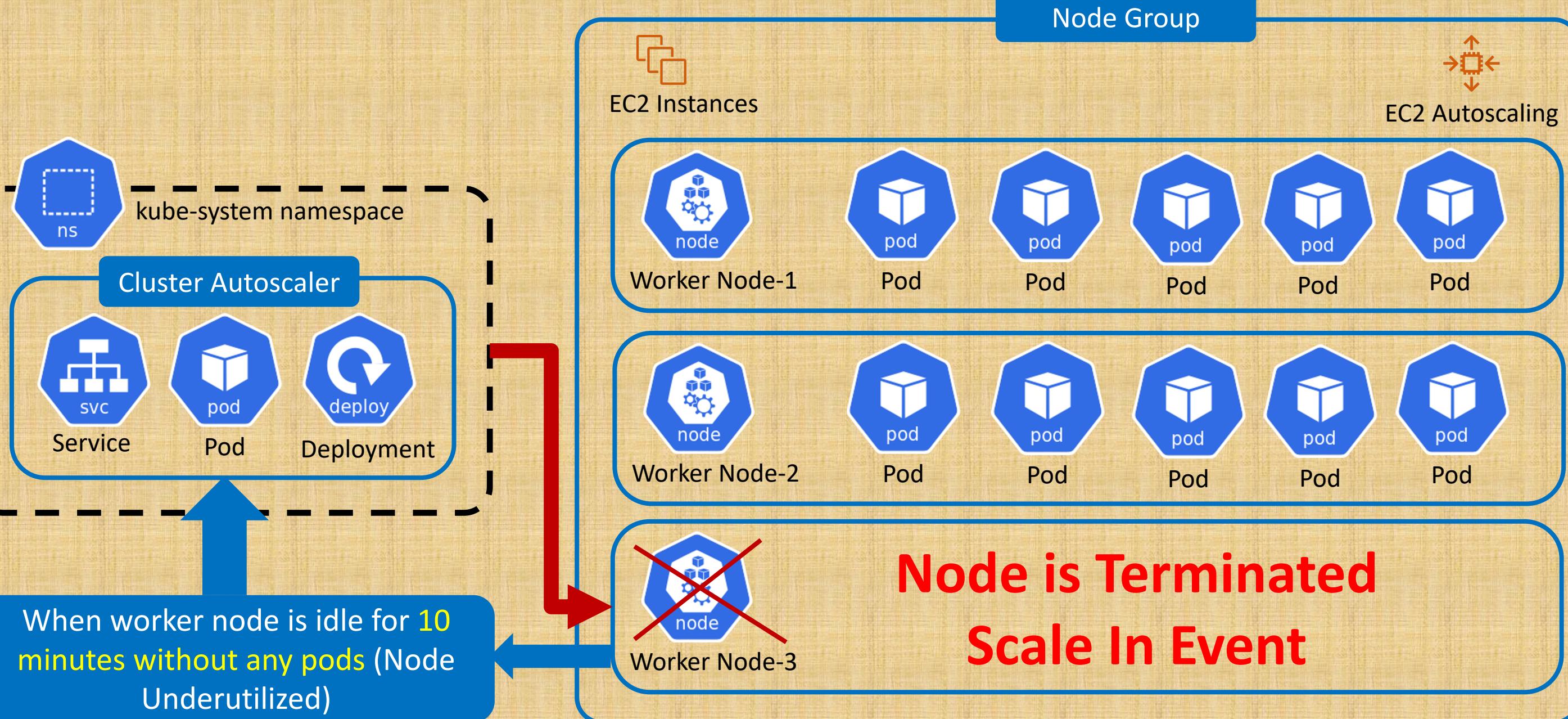


# AWS EKS Cluster Autoscaler - Scale Up

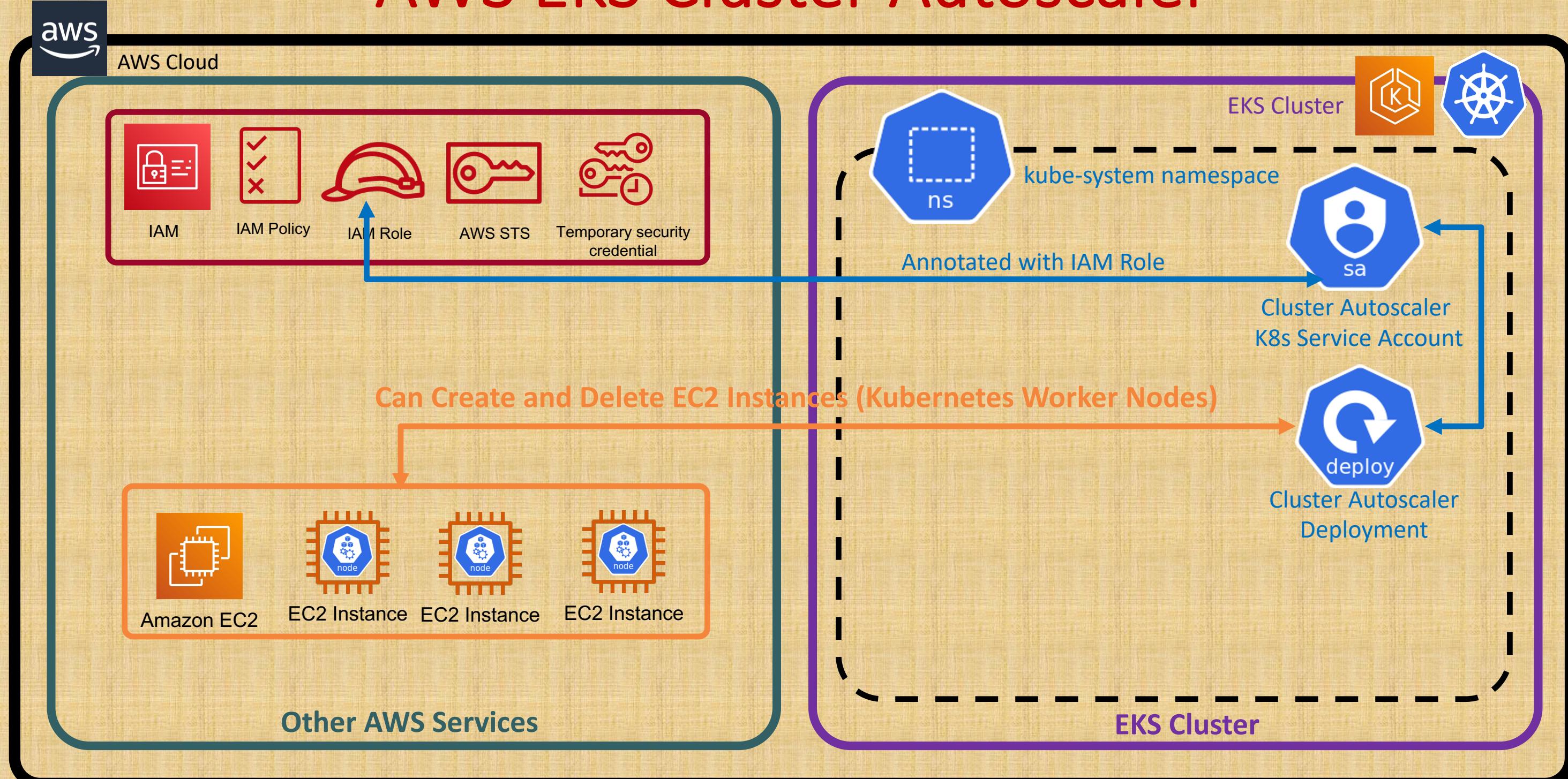


# AWS EKS Cluster Autoscaler - Scale Down

EKS Cluster



# AWS EKS Cluster Autoscaler



# What are we going to learn ?

```
✓ 50-EKS-Cluster-Autoscaler
  > 01-ekscluster-terraform-manifests
  ✓ 02-cluster-autoscaler-install-terraform-manifests
    ↴ c1-versions.tf
    ↴ c2-remote-state-datasource.tf
    ↴ c3-01-generic-variables.tf
    ↴ c3-02-local-values.tf
    ↴ c4-01-cluster-autoscaler-iam-policy-and-role.tf
    ↴ c4-02-cluster-autoscaler-helm-provider.tf
    ↴ c4-03-cluster-autoscaler-install.tf
    ↴ c4-04-cluster-autoscaler-outputs.tf
    ↴ terraform.tfvars
```

## Project Folders

01

EKS Cluster Terraform Manifests

02

Cluster Autoscaler Install Terraform Manifests

- ✓ 50-EKS-Cluster-Autoscaler
- ✓ 01-ekscluster-terraform-manifests
  - > local-exec-output-files
  - > private-key
  - c1-versions.tf
  - c2-01-generic-variables.tf
  - c2-02-local-values.tf
  - c3-01-vpc-variables.tf

- c5-01-eks-variables.tf
- c5-02-eks-outputs.tf
- c5-03-iamrole-for-eks-cluster.tf
- c5-04-iamrole-for-eks-nodegroup.tf
- c5-05-securitygroups-eks.tf
- c5-06-eks-cluster.tf
- c5-07-eks-node-group-public.tf
- c5-08-eks-node-group-private.tf
- c6-01-iam-oidc-connect-provider-variables.tf
- c6-02-iam-oidc-connect-provider.tf
- c7-01-kubernetes-provider.tf
- c7-02-kubernetes-configmap.tf
- c8-01-iam-admin-user.tf
- c8-02-iam-basic-user.tf

# What are we going to learn ?

## Terraform Manifests

01

EKS Cluster Terraform Manifests

c5-07

Add Tags related to Cluster Autoscaler for Public Node Group Resource

c5-08

Add Tags related to Cluster Autoscaler for Private Node Group Resource

c5-04

Attach Autoscaling Full Access policy to EKS Node Group IAM Role

# What are we going to learn ?

- ✓ 50-EKS-Cluster-Autoscaler
  - > 01-ekscluster-terraform-manifests
  - ✓ 02-cluster-autoscaler-install-terraform-manifests
    - ✗ c1-versions.tf
    - ✗ c2-remote-state-datasource.tf
    - ✗ c3-01-generic-variables.tf
    - ✗ c3-02-local-values.tf
    - ✗ c4-01-cluster-autoscaler-iam-policy-and-role.tf
    - ✗ c4-02-cluster-autoscaler-helm-provider.tf
    - ✗ c4-03-cluster-autoscaler-install.tf
    - ✗ c4-04-cluster-autoscaler-outputs.tf
  - ✗ terraform.tfvars

## Terraform Manifests

c1

Terraform Settings Block with Remote S3 backend for State Storage and **DynamoDB Table** for State Locking

c2

Terraform Remote State Datasource to access **Project-01 EKS Cluster Outputs**

c3-01

Terraform Input Variables

c3-02

Terraform Local Values

c4-01

Cluster Autoscaler **IAM Policy** and **IAM Role**

c4-02

Define **Terraform Helm Provider**

c4-03

Create **Terraform Helm Release Resource** to install Cluster Autoscaler on EKS Cluster

c4-04

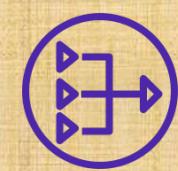
Cluster Autoscaler Helm Release **Outputs**



VPC

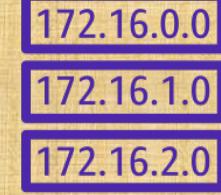


Internet gateway



NAT gateway

Elastic IP address  
→



Route table



Public Subnet



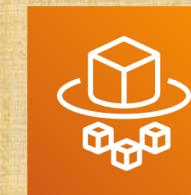
Private Subnet



AWS S3



DynamoDB

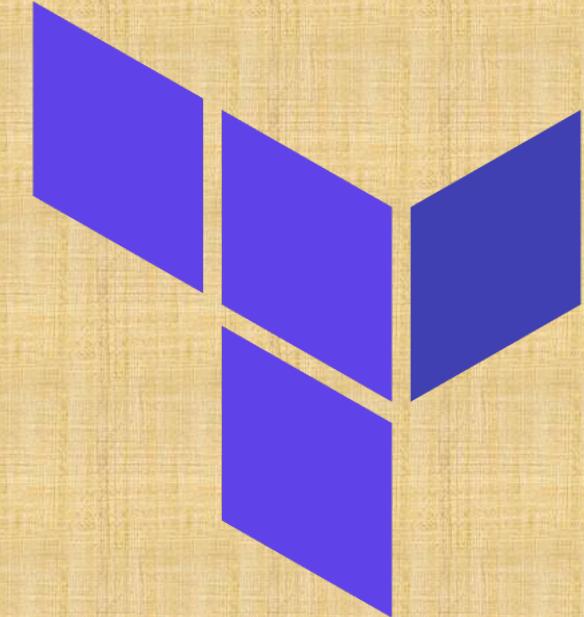


Fargate Profiles

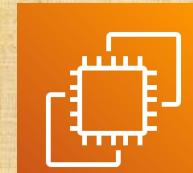
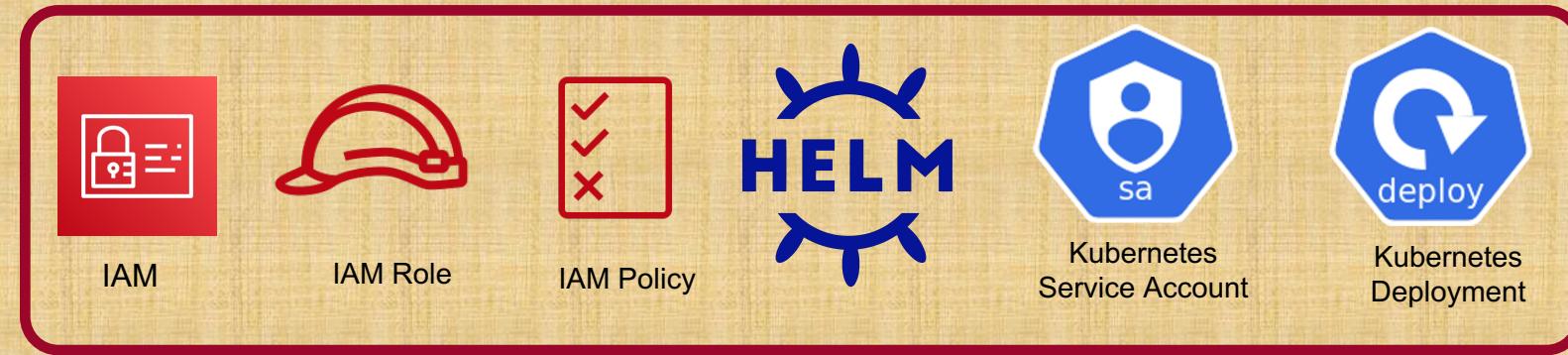


# AWS EKS

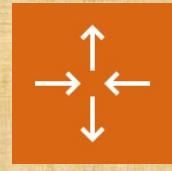
## Cluster Autoscaler



EKS Cluster



EC2 VM



Autoscaling

**Kubernetes Cluster Autoscaler - Testing**  
**Automate with Terraform**

# What are we going to learn ?

## YAML Manifests

```
✓ 51-EKS-Cluster-Autoscaler-Testing
  > 01-ekscluster-terraform-manifests
  > 02-cluster-autoscaler-install-terraform-manifests
  ✓ 03-cluster-autoscaler-sample-app
    ! cluster-autoscaler-sample-app.yaml
```

01

EKS Cluster Terraform Manifests

02

Cluster Autoscaler Install Terraform Manifests

03

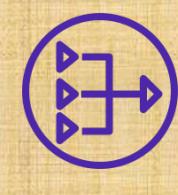
Simple k8s Deployment in YAML to Test Cluster  
Autoscaler



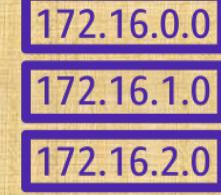
VPC



Internet gateway



NAT gateway



Route table



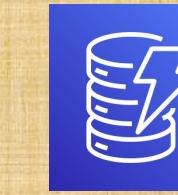
Public Subnet



Private Subnet



AWS S3

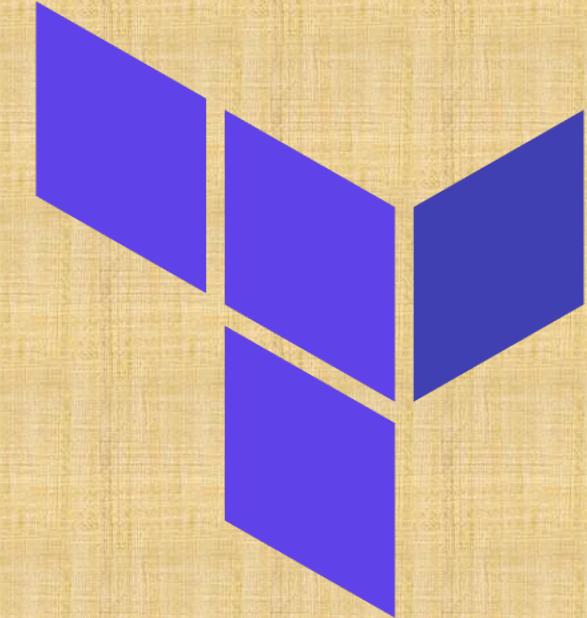
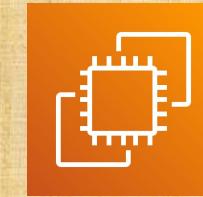
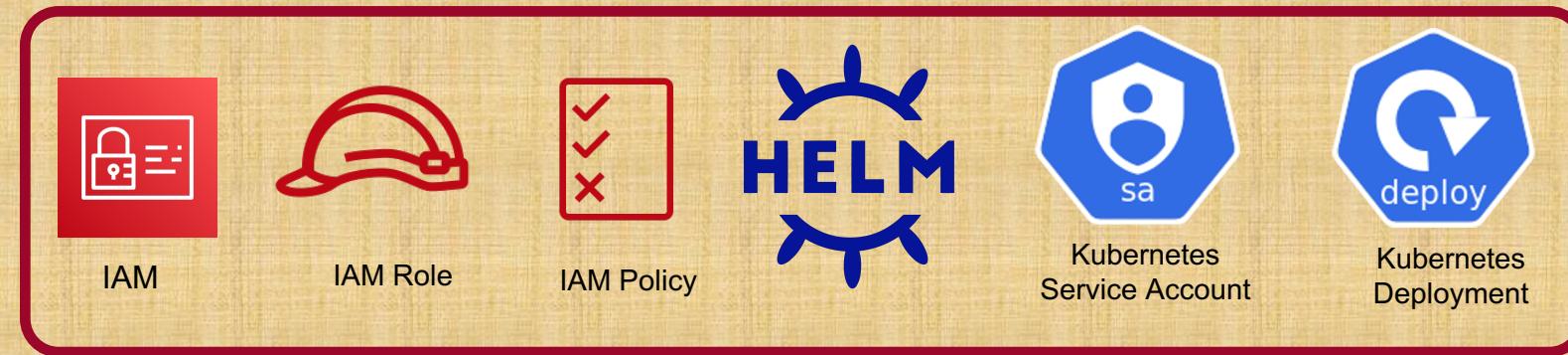


DynamoDB

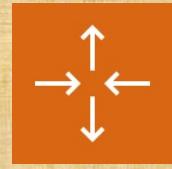
Fargate  
Profiles

# AWS EKS

## Metrics Server

EKS  
Cluster

EC2 VM



Autoscaling

**Kubernetes Horizontal Pod Autoscaler**  
**Automate with Terraform**

# Horizontal Pod Autoscaler

Horizontal Scaling means **increasing** and **decreasing** the number of Replicas (Pods)

HPA automatically **scales the number of pods** in a deployment, replication controller, or replica set, stateful set based on that resource's **CPU utilization**.



Horizontal  
Pod Autoscaler

This can help our applications **scale out** to meet increased demand or **scale in** when resources are not needed, thus freeing up your worker nodes for other applications.

When we set a **target CPU utilization percentage**, the HPA scales our application in or out to try to meet that **target**.

HPA needs **Kubernetes metrics server** to verify CPU metrics of a pod.

We **do not need to deploy or install** the HPA on our cluster to begin scaling our applications, its **out of the box** available as a default Kubernetes API resource.

# Kubernetes Metrics Server

Metrics Server collects resource metrics from Kubelets and exposes them in Kubernetes apiserver through Metrics API for use by Horizontal Pod Autoscaler and Vertical Pod Autoscaler.

Metrics API can also be accessed by `kubectl top`, making it easier to debug autoscaling pipelines.

Metrics Server is not meant for non-autoscaling purposes.  
For example, don't use it to forward metrics to monitoring solutions.  
In short Metrics Server is for k8s autoscaling purpose only.

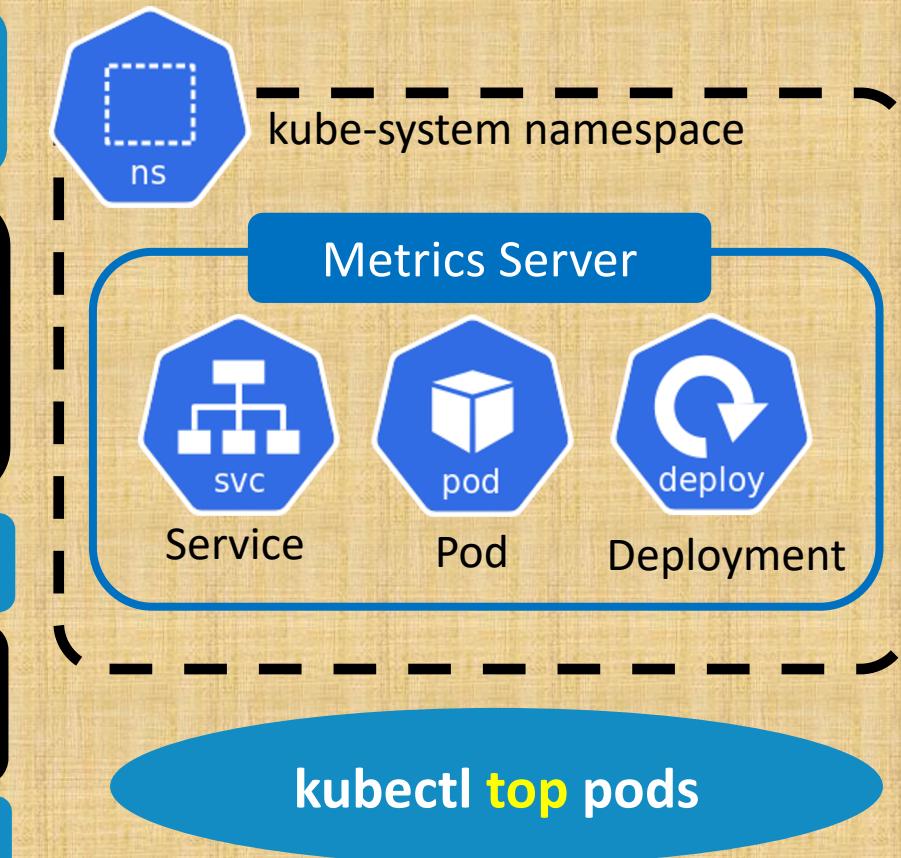
Fast Autoscaling solution, collecting metrics every 15 seconds

Resource efficiency, using 1 mili core of CPU and 2 MB of memory for each node in a cluster

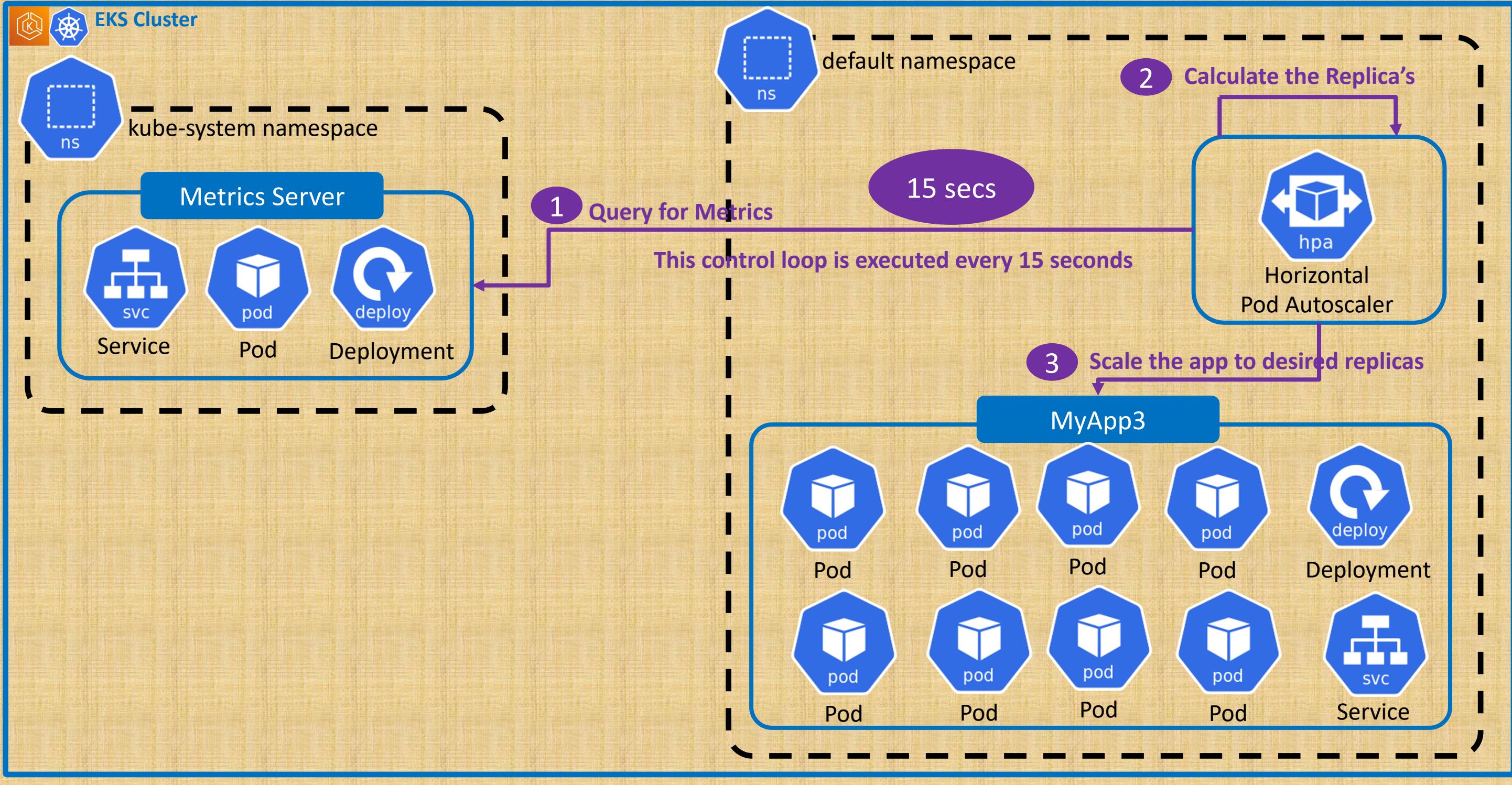
Scalable support up to 5,000 node clusters

Metrics Server used for CPU/Memory based horizontal autoscaling

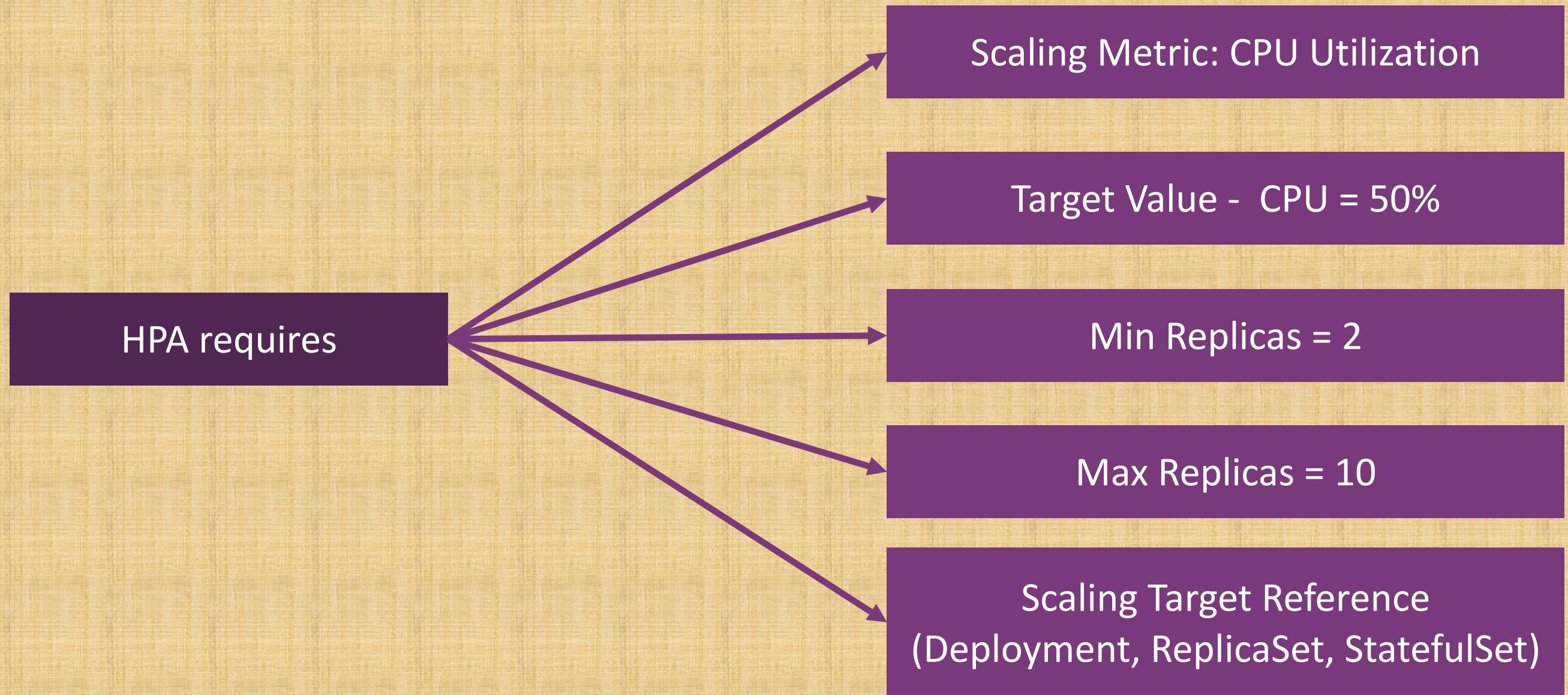
Metrics Server used for Automatically adjusting/suggesting resources needed by containers (Vertical Autoscaling)



# Kubernetes Horizontal Pod Autoscaler



# How is HPA configured?



# Horizontal Pod Autoscaler - YAML Manifest

```
# Resource: Horizontal Pod Autoscaler
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: hpa-app3
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: app3-nginx-deployment
  minReplicas: 1
  maxReplicas: 10
  targetCPUUtilizationPercentage: 50
```



Horizontal  
Pod Autoscaler

# Horizontal Pod Autoscaler - Terraform Manifest

```
# Resource: Horizontal Pod Autoscaler
resource "kubernetes_horizontal_pod_autoscaler_v1" "hpa_myapp3" {
  metadata {
    name = "hpa-app3"
  }
  spec {
    max_replicas = 10
    min_replicas = 1
    scale_target_ref {
      api_version = "apps/v1"
      kind = "Deployment"
      name = kubernetes_deployment_v1.myapp3.metadata[0].name
    }
    target_cpu_utilization_percentage = 50
  }
}
```



Horizontal  
Pod Autoscaler

```
✓ 52-EKS-Horizontal-Pod-Autoscaler
  > 01-ekscluster-terraform-manifests
    ✓ 02-k8s-metrics-server-terraform-manifests
      ✓ c1-versions.tf
      ✓ c2-remote-state-datasource.tf
      ✓ c3-01-generic-variables.tf
      ✓ c3-02-local-values.tf
      ✓ c4-01-helm-provider.tf
      ✓ c4-02-metrics-server-install.tf
      ✓ c4-03-metrics-server-outputs.tf
      ✓ terraform.tfvars
    ✓ 03-hpa-demo-yaml
      ! 01-deployment.yaml
      ! 02-service.yaml
      ! 03-hpa.yaml
    ✓ 04-hpa-demo-terraform-manifests
      ✓ c1-versions.tf
      ✓ c2-remote-state-datasource.tf
      ✓ c3-providers.tf
      ✓ c4-kubernetes-app3-deployment.tf
      ✓ c5-kubernetes-app3-clusterip-service.tf
      ✓ c6-hpa-resource.tf
```

# What are we going to learn ?

## Project Folders

01

EKS Cluster Terraform Manifests

02

Metrics Server Install Terraform Manifests

03

HPA Demo: k8s Deployment, Service and HPA  
resources using YAML

04

HPA Demo: k8s Deployment, Service and HPA  
resources using Terraform

- ✓ 52-EKS-Horizontal-Pod-Autoscaler
  - > 01-ekscluster-terraform-manifests
  - ✓ 02-k8s-metrics-server-terraform-manifests
    - └ c1-versions.tf
    - └ c2-remote-state-datasource.tf
    - └ c3-01-generic-variables.tf
    - └ c3-02-local-values.tf
    - └ c4-01-helm-provider.tf
    - └ c4-02-metrics-server-install.tf
    - └ c4-03-metrics-server-outputs.tf
    - └ terraform.tfvars
  - ✓ 03-hpa-demo-yaml
    - ! 01-deployment.yaml
    - ! 02-service.yaml
    - ! 03-hpa.yaml
  - ✓ 04-hpa-demo-terraform-manifests
    - └ c1-versions.tf
    - └ c2-remote-state-datasource.tf
    - └ c3-providers.tf
    - └ c4-kubernetes-app3-deployment.tf
    - └ c5-kubernetes-app3-clusterip-service.tf
    - └ c6-hpa-resource.tf

# What are we going to learn ?

## Terraform Manifests Project-02

c1, c2,  
c3

c4-01

c4-02

c4-03

Terraform Settings, Remote State Datasource,  
Generic Input Variables and Local Values

Define Terraform Helm Provider

Define Terraform Helm Release Resource for  
deploying Metrics Service

Helm Release Outputs

- ✓ 52-EKS-Horizontal-Pod-Autoscaler
  - > 01-ekscluster-terraform-manifests
  - ✓ 02-k8s-metrics-server-terraform-manifests
    - └ c1-versions.tf
    - └ c2-remote-state-datasource.tf
    - └ c3-01-generic-variables.tf
    - └ c3-02-local-values.tf
    - └ c4-01-helm-provider.tf
    - └ c4-02-metrics-server-install.tf
    - └ c4-03-metrics-server-outputs.tf
    - └ terraform.tfvars
  - ✓ 03-hpa-demo-yaml
    - ! 01-deployment.yaml
    - ! 02-service.yaml
    - ! 03-hpa.yaml
  - ✓ 04-hpa-demo-terraform-manifests
    - └ c1-versions.tf
    - └ c2-remote-state-datasource.tf
    - └ c3-providers.tf
    - └ c4-kubernetes-app3-deployment.tf
    - └ c5-kubernetes-app3-clusterip-service.tf
    - └ c6-hpa-resource.tf

# What are we going to learn ?

## Terraform Manifests Project-04

c1, c2,  
c3

c4

c5

c6

Terraform Settings, Remote State Datasource,  
Generic Input Variables and Local Values

Kubernetes App3 Deployment

Kubernetes App3 Cluster IP Service

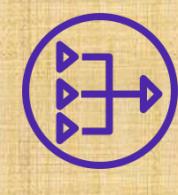
Kubernetes Horizontal Pod Autoscaler Resource (HPA)



VPC



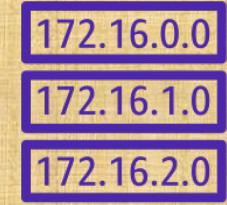
Internet gateway



NAT gateway



Elastic IP address



Route table



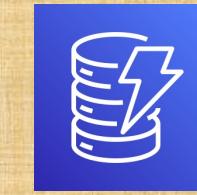
Public Subnet



Private Subnet



AWS S3



DynamoDB

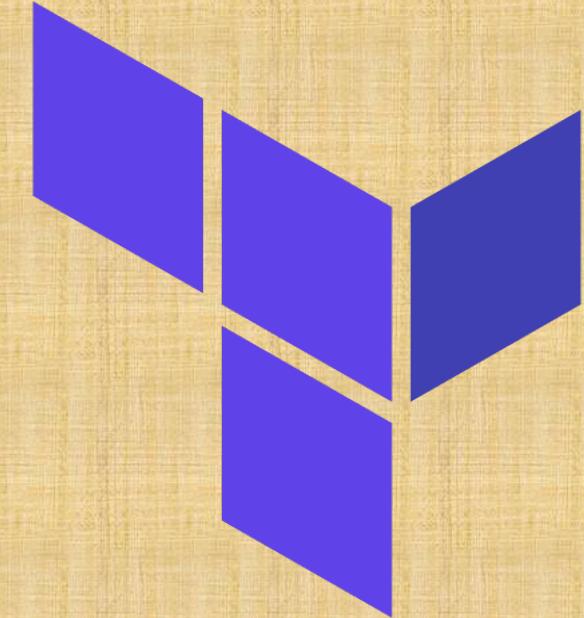


Fargate Profiles

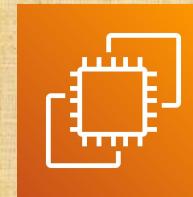
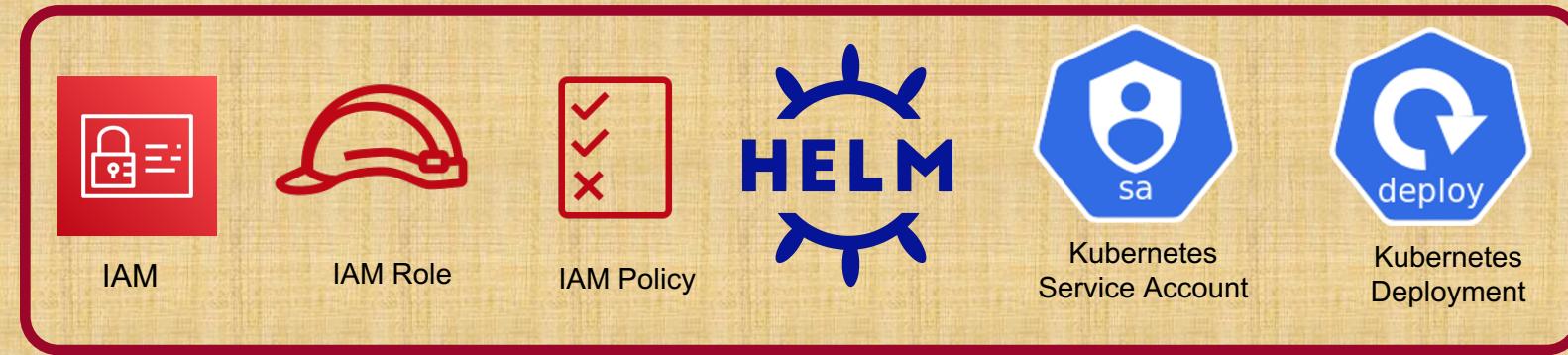


# AWS EKS

## Metrics Server



EKS Cluster



EC2 VM



Autoscaling

**Kubernetes Vertical Pod Autoscaler**  
Automate with Terraform

# Vertical Pod Autoscaler

- VPA automatically adjusts the **CPU and memory reservations** for our pods to help "**right size**" our applications.
- This adjustment can **improve cluster resource utilization** and **free up** CPU and memory for other pods.
- Benefits
  - Cluster nodes are used **efficiently**, because Pods use exactly what they need.
  - Pods are **scheduled onto nodes** that have the appropriate resources available.
  - We **don't have** to run **time-consuming benchmarking tasks** to determine the correct values for CPU and memory requests.
  - **Maintenance time is reduced**, because the VPA can adjust CPU and memory requests over time without any action on your part.

## VPA Components

### VPA Admission Hook

Every pod submitted to the k8s cluster goes through this webhook automatically which checks whether a **VerticalPodAutoscaler** object is referencing this pod or one of its parents (a ReplicaSet, a Deployment, etc).

### VPA Recommender

Connects to the **metrics-server** in the cluster, fetches historical and current usage data (CPU and memory) for each VPA-enabled pod and generates recommendations for **scaling up or down** the requests and limits of these pods.

### VPA Updater

**Runs every 1 minute.** If a pod is not running in the calculated recommendation range, **it evicts the currently running version of this pod**, so it can restart and go through the **VPA admission webhook** which will change the CPU and memory settings for it, before it can start.

- ✓ 53-EKS-Vertical-Pod-Autoscaler-Install
  - > 01-ekscluster-terraform-manifests
  - ✓ 02-k8s-metrics-server-terraform-manifests
    - └ c1-versions.tf
    - └ c2-remote-state-datasource.tf
    - └ c3-01-generic-variables.tf
    - └ c3-02-local-values.tf
    - └ c4-01-helm-provider.tf
    - └ c4-02-metrics-server-install.tf
    - └ c4-03-metrics-server-outputs.tf
    - └ terraform.tfvars
  - ✓ 03-vpa-install-terraform-manifests
    - └ c1-versions.tf
    - └ c2-vpa-install.tf
  - ✓ 04-vpa-demo-yaml
    - ! 01-vpa-demo-app.yaml
    - ! 02-vpa-resource.yaml
  - ✓ 05-vpa-demo-terraform-manifests
    - └ c1-versions.tf
    - └ c2-remote-state-datasource.tf
    - └ c3-01-generic-variables.tf
    - └ c3-02-local-values.tf
    - └ c4-01-terraform-providers.tf
    - └ c4-02-vpa-sample-app-deployment.tf

# What are we going to learn ?

## Project Folders

01

EKS Cluster Terraform Manifests

02

Metrics Server Install Terraform Manifests

03

VPA Install Terraform Manifests

04

VPA Demo: k8s Deployment, Service and VPA resources using YAML

05

VPA Demo: k8s Deployment, Service and VPA resources using Terraform

- ✓ 53-EKS-Vertical-Pod-Autoscaler-Install
  - > 01-ekscluster-terraform-manifests
  - > 02-k8s-metrics-server-terraform-manifests
  - ✓ 03-vpa-install-terraform-manifests
    - └ c1-versions.tf
    - └ c2-vpa-install.tf
  - > 04-vpa-demo-yaml
  - > 05-vpa-demo-terraform-manifests

p3-c1

Terraform Settings Block & Providers

p3-c2

#### VPA Install Process:

1. **Terraform Resources:** Null Resource + Local Exec Provisioner
2. **Resource-1:** VPA Git Repo Clone
3. **Resource-2:** Install VPA Resources
4. **Resource-3:** Uninstall VPA Resource During Destroy
5. **Resource-4:** Remove “autoscaler” folder from local Terraform Working Directory during Destroy

What are we going to learn ?

Terraform Manifests  
Project-03

# What are we going to learn ?

- ✓ 53-EKS-Vertical-Pod-Autoscaler-Install
  - > 01-ekscluster-terraform-manifests
  - > 02-k8s-metrics-server-terraform-manifests
  - > 03-vpa-install-terraform-manifests
  - > 04-vpa-demo-yaml
  - ✓ 05-vpa-demo-terraform-manifests
    - ⚡ c1-versions.tf
    - ⚡ c2-remote-state-datasource.tf
    - ⚡ c3-01-generic-variables.tf
    - ⚡ c3-02-local-values.tf
    - ⚡ c4-01-terraform-providers.tf
    - ⚡ c4-02-vpa-sample-app-deployment.tf
    - ⚡ c4-03-vpa-sample-app-service.tf
    - ⚡ c4-04-vpa-resource.tf
    - ⚡ **terraform.tfvars**

## Terraform Manifests Project-05

c1, c2,  
c3

c4-01

c4-02

c4-03

c4-04

Terraform Settings, Remote State Datasource,  
Generic Input Variables and Local Values

Terraform Providers (AWS, Kubernetes, **Kubectl**)

Kubernetes **VPA Sample App k8s Deployment**

Kubernetes **VPA Sample App k8s ClusterIP Service**

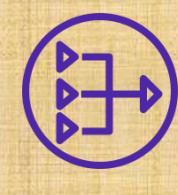
Kubernetes **Vertical Pod Autoscaler Resource (VPA)**



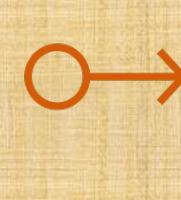
VPC



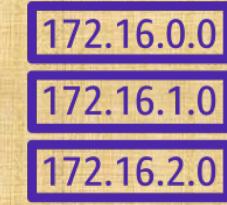
Internet gateway



NAT gateway



Elastic IP address



Route table



Public Subnet



Private Subnet



AWS S3



DynamoDB

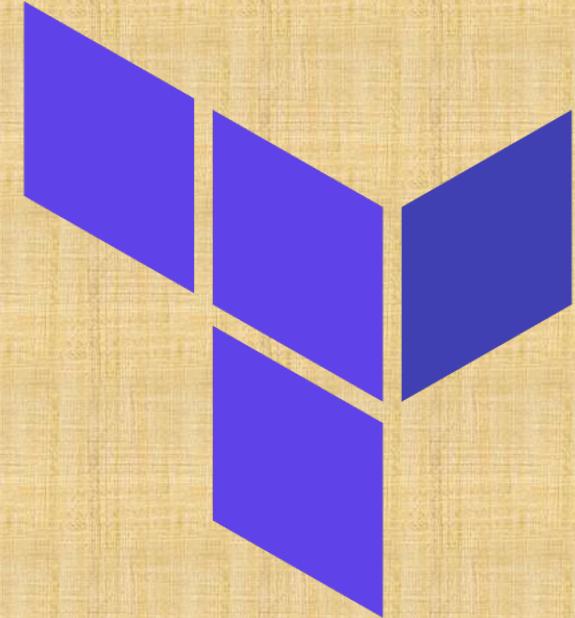


Fargate Profiles

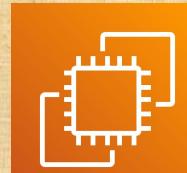
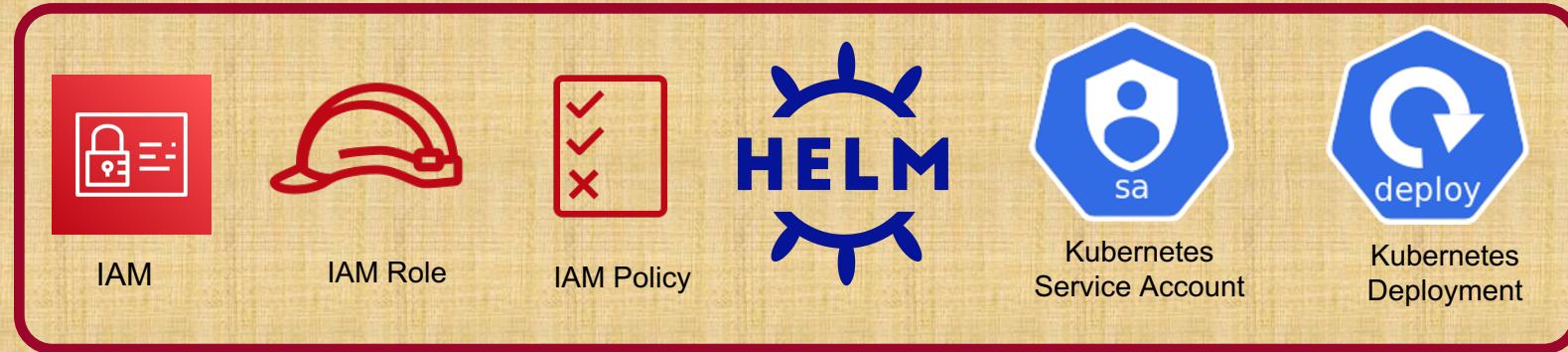


# AWS EKS

## Logging & Monitoring



EKS Cluster



EC2 VM

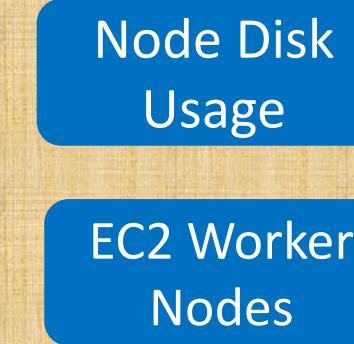
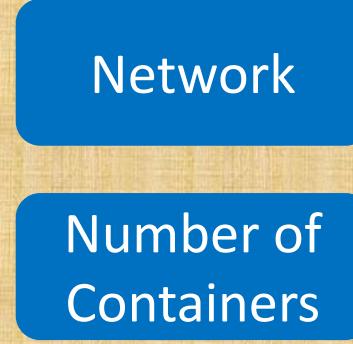
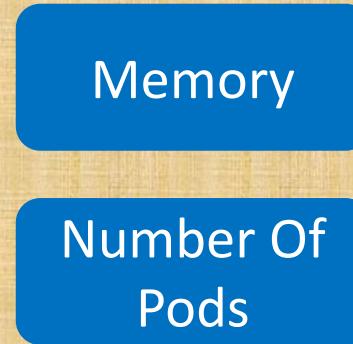
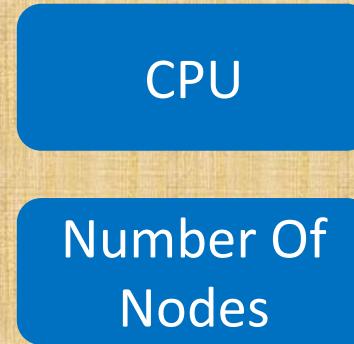


Autoscaling

**AWS CloudWatch Container Insights**

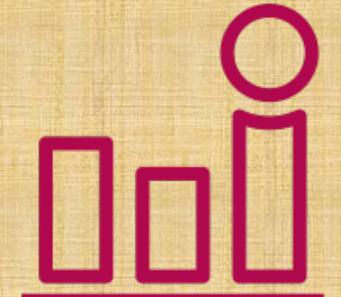
# AWS CloudWatch Container Insights

AWS CloudWatch Container Insights collects metrics and logs from containerized applications and microservices deployed on AWS EKS Clusters



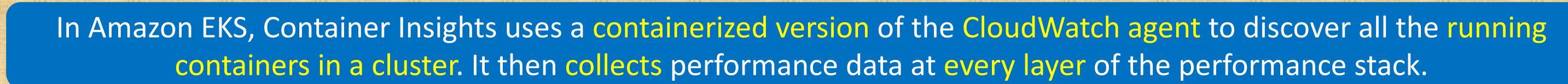
Amazon CloudWatch

Container Insights also provides diagnostic information, such as container restart failures, to help us isolate issues and resolve them quickly.



CloudWatch  
Alarm

Metrics collected by Container Insights are charged as custom metrics.



# AWS CloudWatch Container Insights

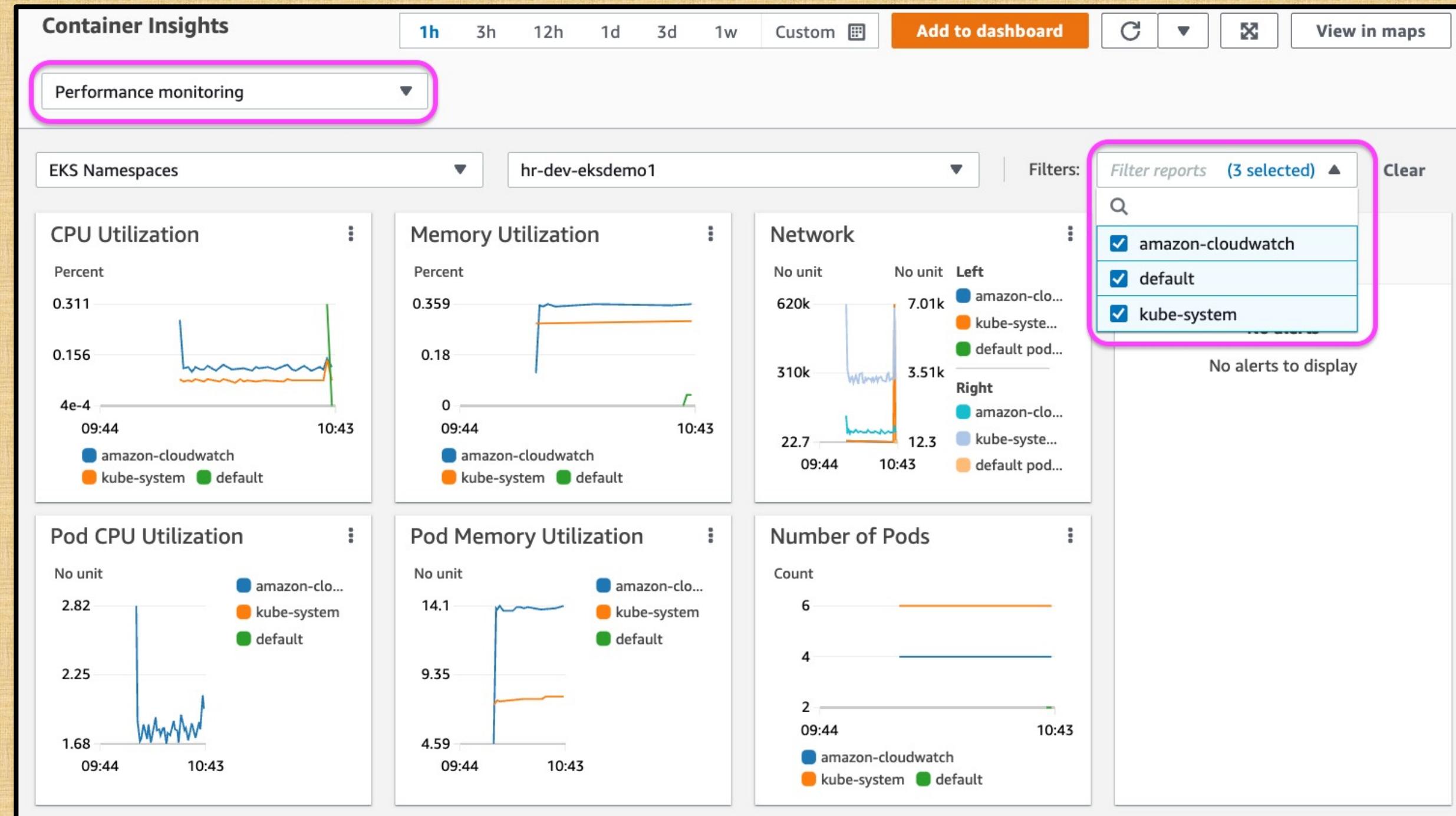
Resources (15)

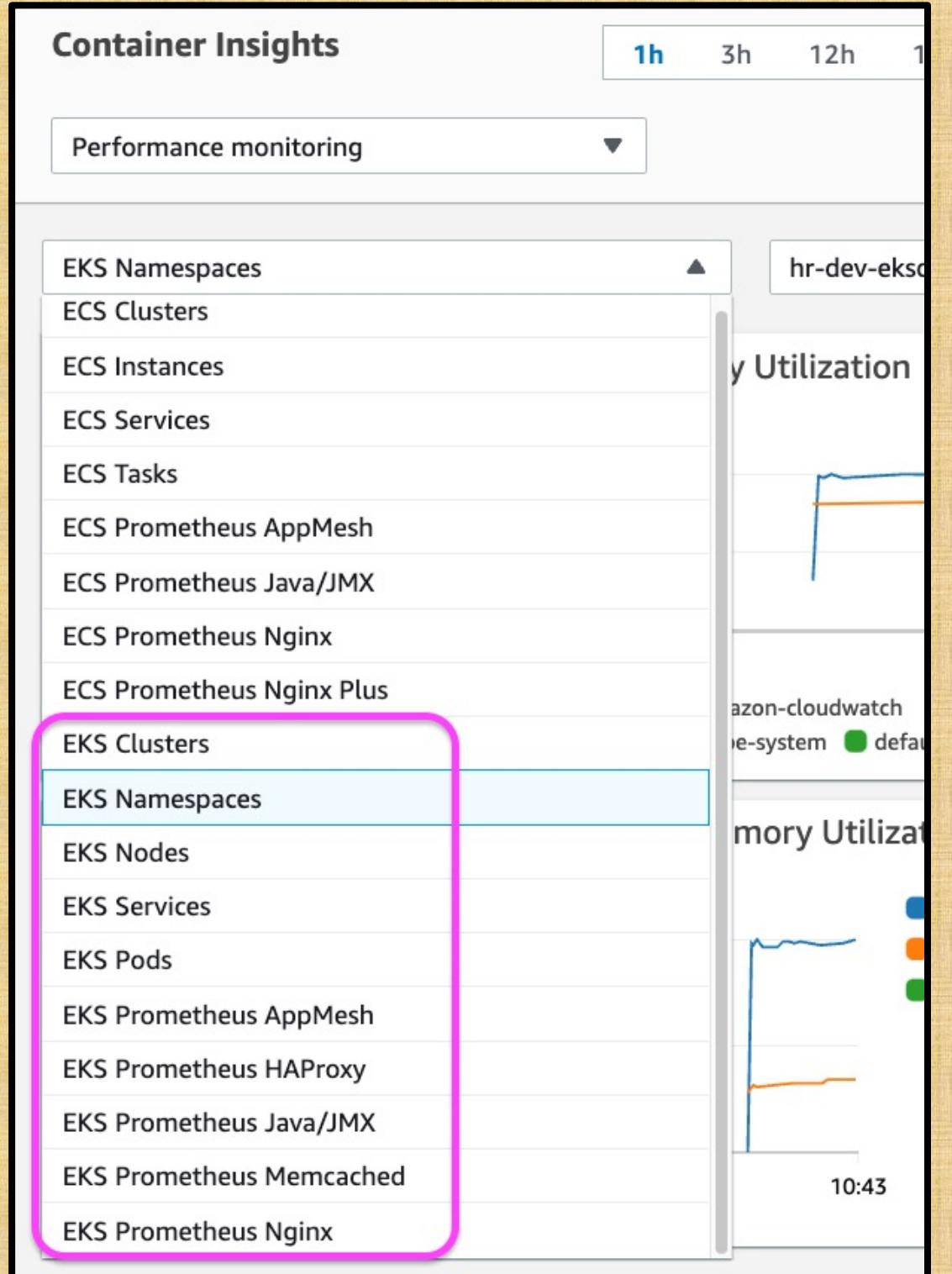
View logs  View dashboard 

Find resources   

	Name	Type	Cluster name	Alarms	Prometheus	Avg CPU (%)	Avg memory (%)
	<a href="#">amazon-cloudwatch</a>	EKS Namespace	hr-dev-eksdemo1	-	-	0.1%	0.4%
	<a href="#">aws-node</a>	EKS Pod	hr-dev-eksdemo1	-	-	0.2%	0.6%
	<a href="#">ca-demo-deployment</a>	EKS Pod	hr-dev-eksdemo1	-	-	-	-
	<a href="#">ca-demo-service-nginx</a>	EKS Service	hr-dev-eksdemo1	-	-	-	-
	<a href="#">cloudwatch-agent</a>	EKS Pod	hr-dev-eksdemo1	-	-	0.2%	0.3%
	<a href="#">coredns</a>	EKS Pod	hr-dev-eksdemo1	-	-	<0.1%	0.2%
	<a href="#">default</a>	EKS Namespace	hr-dev-eksdemo1	-	-	0.3%	<0.1%
	<a href="#">fluent-bit</a>	EKS Pod	hr-dev-eksdemo1	-	-	<0.1%	0.4%
	<a href="#">hr-dev-eksdemo1</a>	EKS Cluster	hr-dev-eksdemo1	-	-	3.7%	7.4%

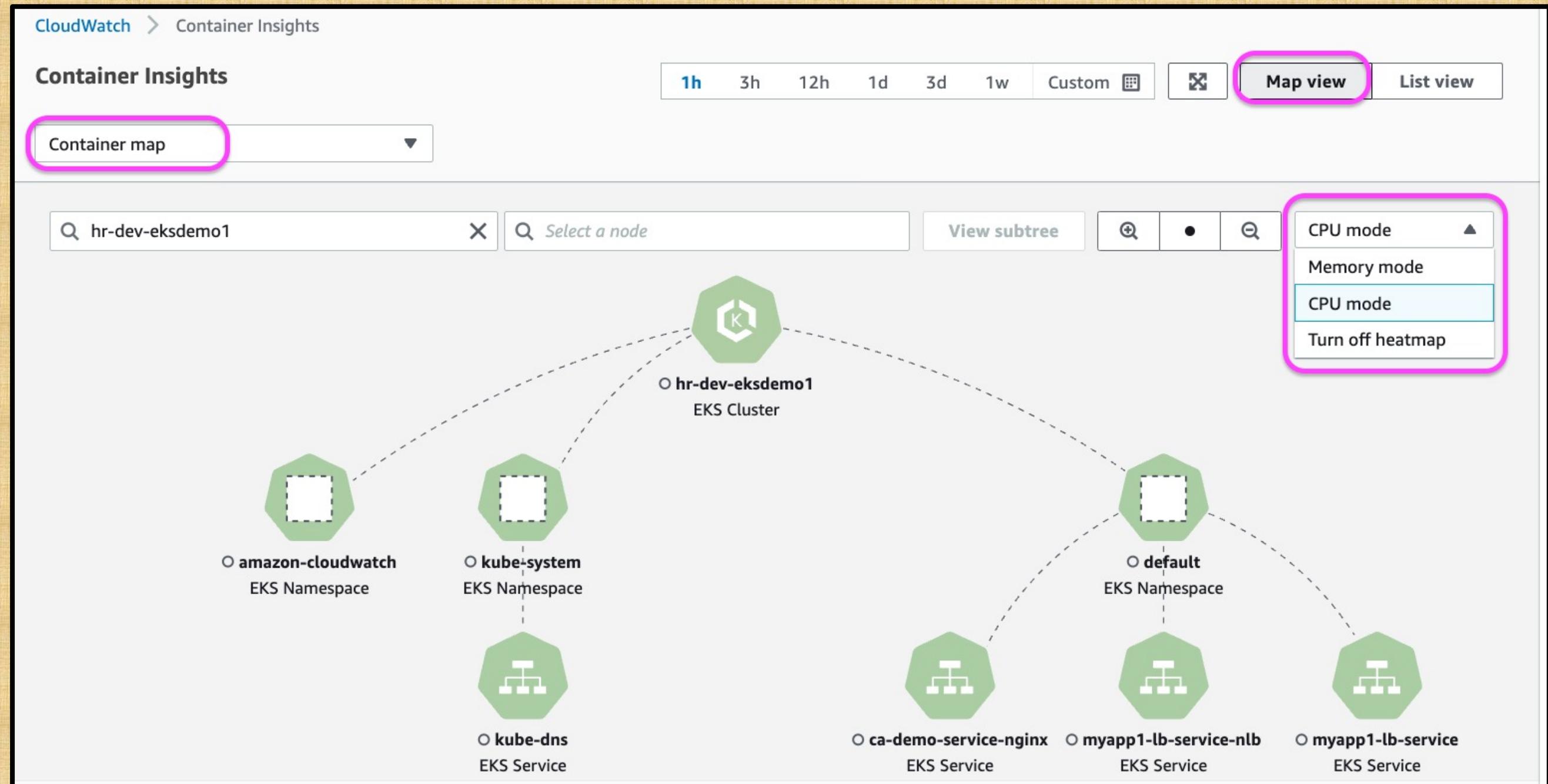
# AWS CloudWatch Container Insights



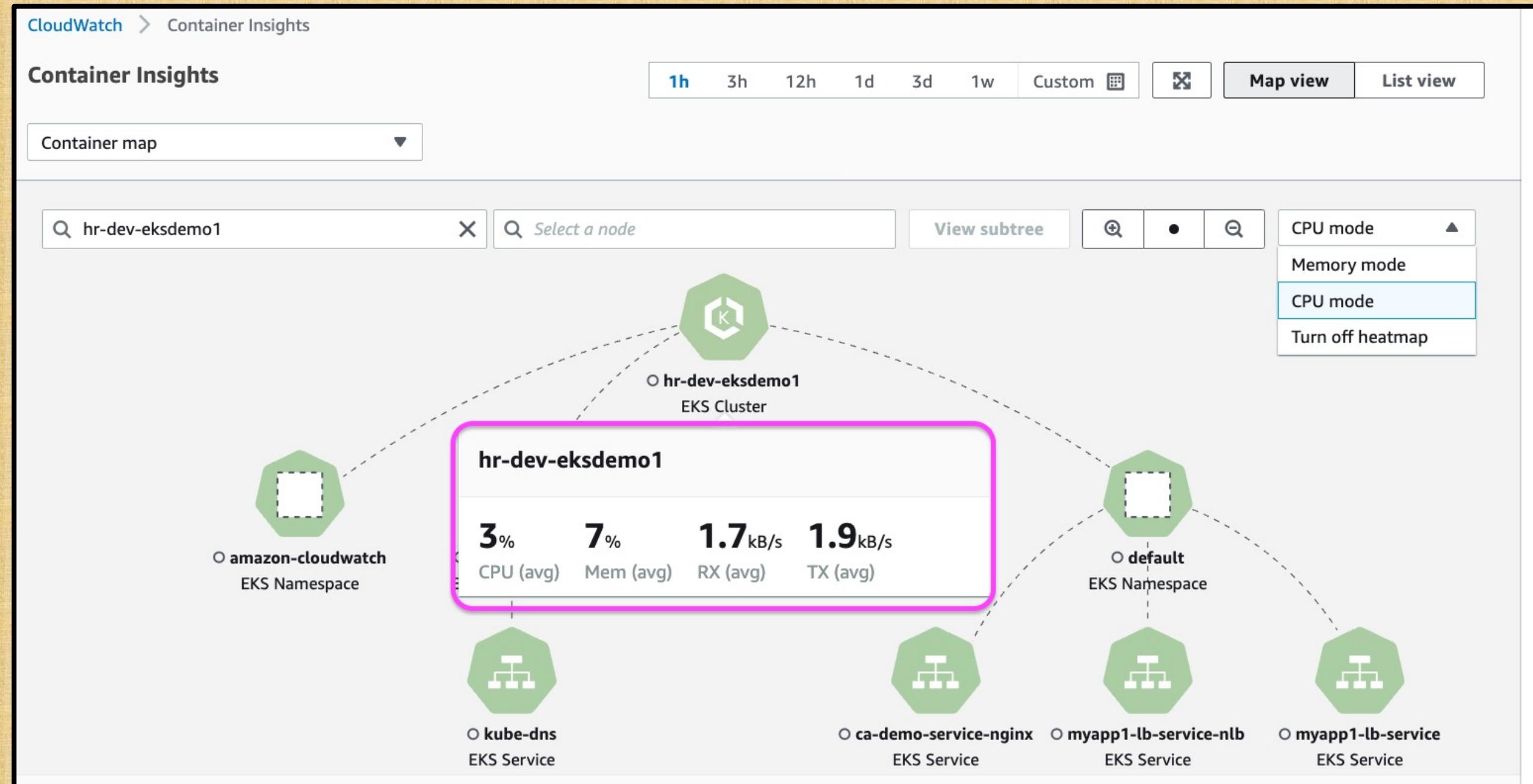


# AWS CloudWatch Container Insights

# AWS CloudWatch Container Insights



# AWS CloudWatch Container Insights



# AWS CloudWatch Container Insights

Alarms (12)				
Alarm name	State	State updated	Related nodes	
<a href="#">ApplicationInsights/ApplicationInsights-ContainerInsights-EKS_CLUSTER-hr-dev-eksdemo1/ContainerInsights/pod_memory_reserved_capacity/hr-dev-eksdemo1/</a>	<span>OK</span>	2022-05-22 10:32:30	<ul style="list-style-type: none"><li>hr-dev-eksdemo1 (EKS Cluster)</li></ul>	
<a href="#">ApplicationInsights/ApplicationInsights-ContainerInsights-EKS_CLUSTER-hr-dev-eksdemo1/ContainerInsights/pod_memory_utilization_over_pod_limit/hr-dev-eksdemo1/</a>	<span>OK</span>	2022-05-22 10:32:28	<ul style="list-style-type: none"><li>hr-dev-eksdemo1 (EKS Cluster)</li></ul>	
<a href="#">ApplicationInsights/ApplicationInsights-ContainerInsights-EKS_CLUSTER-hr-dev-eksdemo1/ContainerInsights/cluster_failed_node_count/hr-dev-eksdemo1/</a>	<span>OK</span>	2022-05-22 10:32:19	<ul style="list-style-type: none"><li>hr-dev-eksdemo1 (EKS Cluster)</li></ul>	
<a href="#">ApplicationInsights/ApplicationInsights-ContainerInsights-EKS_CLUSTER-hr-dev-eksdemo1/ContainerInsights/pod_memory_utilization/hr-dev-eksdemo1/</a>	<span>OK</span>	2022-05-22 10:32:18	<ul style="list-style-type: none"><li>hr-dev-eksdemo1 (EKS Cluster)</li></ul>	

# AWS CloudWatch Container Insights

Container Insights collects data as performance log events using embedded metric format

These performance log events are entries that use a structured JSON schema that enables high-cardinality data to be ingested and stored at scale.

From this performance data, CloudWatch creates aggregated metrics at the cluster, node, pod, task, and service level as CloudWatch metrics

CloudWatch > Log groups

**Log groups (7)**  
By default, we only load up to 10000 log groups.

Filter log groups or try prefix search

<input type="checkbox"/> Log group
/aws/containerinsights/hr-dev-eksdemo1/application
/aws/containerinsights/hr-dev-eksdemo1/dataplane
/aws/containerinsights/hr-dev-eksdemo1/host
<input type="checkbox"/> /aws/containerinsights/hr-dev-eksdemo1/performance
/aws/eks/hr-dev-eksdemo1/cluster

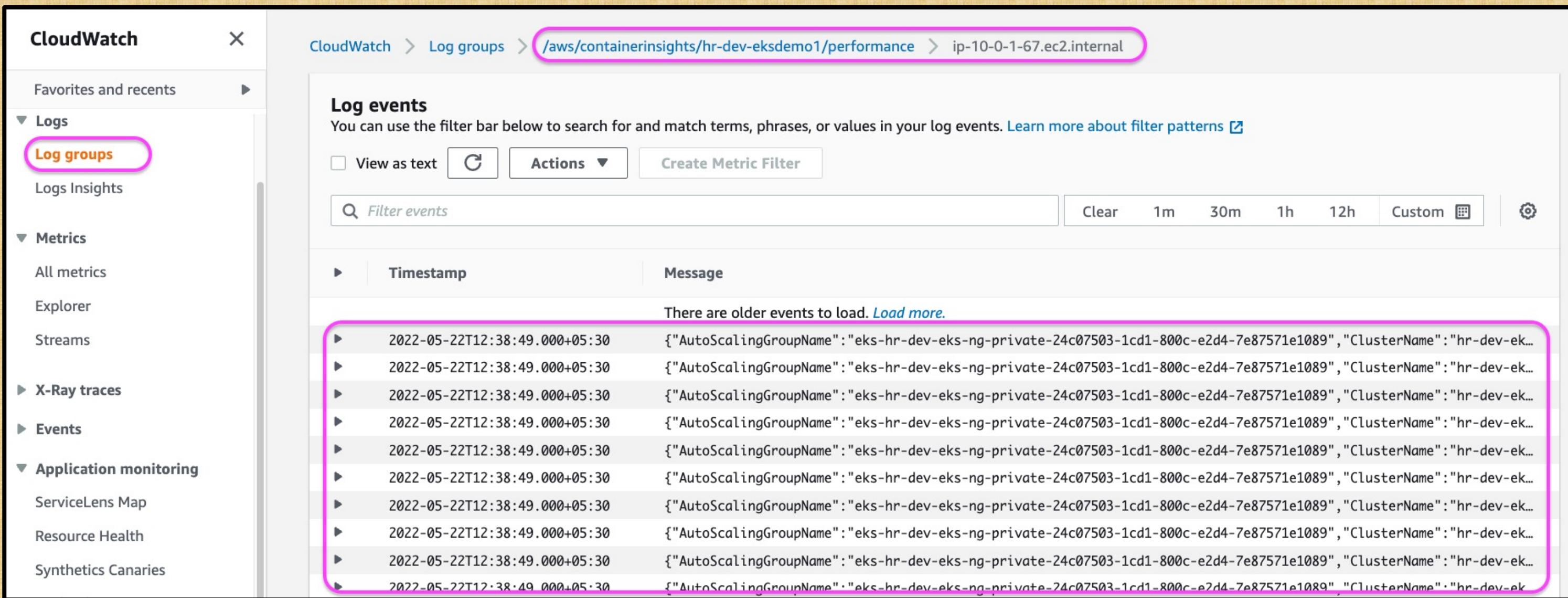
2022-05-22T12:38:49.000+05:30 {"AutoScalingGroupName": "eks-hr-dev-eks-ng-private-24c07503-1cd1-800c-e2d4-7e87571e1089", "ClusterName": "hr-dev-eksdemo1", "InstanceId": "i-0d62a...}

```
{  
    "AutoScalingGroupName": "eks-hr-dev-eks-ng-private-24c07503-1cd1-800c-e2d4-7e87571e1089",  
    "ClusterName": "hr-dev-eksdemo1",  
    "InstanceId": "i-0d62a679a61c0b766",  
    "InstanceType": "t3.large",  
    "Namespace": "kube-system",  
    "NodeName": "ip-10-0-1-67.ec2.internal",  
    "PodName": "kube-proxy",  
    "Sources": [  
        "cadvisor",  
        "calculated"  
    ],  
    "Timestamp": "1653203316568",  
    "Type": "PodNet",  
    "Version": "0",  
    "interface": "eni817a8f7676b",  
    "kubernetes": {  
        "host": "ip-10-0-1-67.ec2.internal",  
        "labels": {  
            "controller-revision-hash": "799c56dccb",  
            "k8s-app": "kube-proxy",  
            "pod-template-generation": "1"  
        },  
        "namespace_name": "kube-system",  
        "pod_id": "0e73898d-43d3-45ee-82a0-51b63380f3ad",  
        "pod_name": "kube-proxy-7h9t6",  
        "pod_owners": [  
            {  
                "owner_kind": "DaemonSet",  
                "owner_name": "kube-proxy"  
            }  
        ]  
    },  
    "pod_interface_network_rx_bytes": 27.491589949745826,  
    "pod_interface_network_rx_dropped": 0,  
    "pod_interface_network_rx_errors": 0,  
    "pod_interface_network_rx_packets": 0.4062796544297413,  
    "pod_interface_network_total_bytes": 79.563098992491,  
    "pod_interface_network_tx_bytes": 52.07150904274518,  
    "pod_interface_network_tx_dropped": 0,  
    "pod_interface_network_tx_errors": 0,  
    "pod_interface_network_tx_packets": 0.7787026709903374  
}
```

2022-05-22T12:38:49.000+05:30 {"AutoScalingGroupName": "eks-hr-dev-eks-ng-private-24c07503-1cd1-800c-e2d4-7e87571e1089", "ClusterName": "hr-dev-eksdemo1", "InstanceId": "i-0d62a..."}

## Embedded Metric Format

# AWS CloudWatch Container Insights

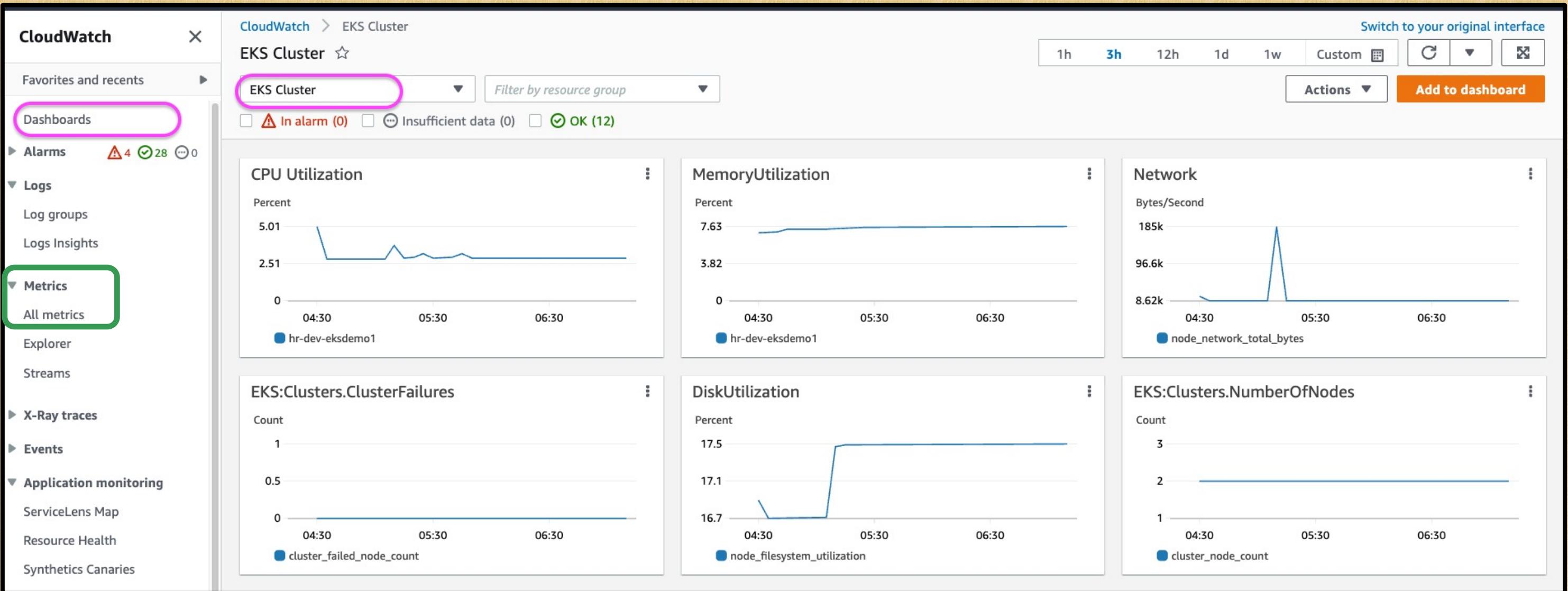


The screenshot shows the AWS CloudWatch Container Insights interface. On the left, a sidebar menu includes 'CloudWatch' (with a close button), 'Favorites and recents', 'Logs' (selected), 'Log groups' (highlighted with a pink border), 'Logs Insights', and 'Metrics'. Under 'Logs', there are sections for 'All metrics', 'Explorer', 'Streams', 'X-Ray traces', 'Events', and 'Application monitoring' (which includes 'ServiceLens Map', 'Resource Health', and 'Synthetics Canaries'). The main content area shows a breadcrumb navigation path: CloudWatch > Log groups > /aws/containerinsights/hr-dev-eksdemo1/performance > ip-10-0-1-67.ec2.internal. Below this is a section titled 'Log events' with a note: 'You can use the filter bar below to search for and match terms, phrases, or values in your log events. Learn more about filter patterns'. It features buttons for 'View as text' (unchecked), a refresh icon, 'Actions' (with a dropdown arrow), and 'Create Metric Filter'. A search bar says 'Filter events' with a magnifying glass icon. To the right are time range buttons: 'Clear', '1m', '30m', '1h', '12h', 'Custom' (with a calendar icon), and a gear icon for settings. The main table has columns 'Timestamp' and 'Message'. A message at the top of the table says 'There are older events to load. [Load more.](#)'. Below this, ten log entries are listed, each with a timestamp from '2022-05-22T12:38:49.000+05:30' and a JSON message object. The entire list of log entries is highlighted with a pink rectangular border.

Timestamp	Message
2022-05-22T12:38:49.000+05:30	{"AutoScalingGroupName": "eks-hr-dev-eks-ng-private-24c07503-1cd1-800c-e2d4-7e87571e1089", "ClusterName": "hr-dev-ek..."}

# AWS CloudWatch Container Insights

The metrics that Container Insights collects are available in CloudWatch automatic dashboards, and viewable in the **Metrics** section of the CloudWatch console.



# AWS CloudWatch Container Insights

CloudWatch X

Favorites and recents ▶

Dashboards

▶ Alarms ⚠ 4 ✓ 28 🕒 0

▼ Logs

**Log groups** (7)

Logs Insights

▼ Metrics

All metrics

Explorer

Streams

▶ X-Ray traces

CloudWatch > Log groups

FluentBit creates these 3 logs groups and pushes logs from EKS Cluster

Log groups (7) By default, we only load up to 10000 log groups.

Filter log groups or try prefix search

<input type="checkbox"/>	Log group	▲ Retention ▼
<input type="checkbox"/>	/aws/containerinsights/hr-dev-eksdemo1/application	Never expire
<input type="checkbox"/>	/aws/containerinsights/hr-dev-eksdemo1/dataplane	Never expire
<input type="checkbox"/>	/aws/containerinsights/hr-dev-eksdemo1/host	Never expire
<input type="checkbox"/>	/aws/containerinsights/hr-dev-eksdemo1/performance	Never expire
<input type="checkbox"/>	/aws/eks/hr-dev-eksdemo1/cluster	Never expire
<input type="checkbox"/>	EKS Cluster Control Plane logs enabled as part of /aws/lambda/invalidate aws_eks_cluster Terraform Resource	Never expire

# AWS CloudWatch Container Insights

The screenshot shows the AWS CloudWatch interface for Container Insights. The left sidebar navigation includes CloudWatch, Favorites and recents, Dashboards, Alarms (4 alerts), Logs (selected), Log groups (highlighted with a blue border), Logs Insights, Metrics, X-Ray traces, Events, Application monitoring (ServiceLens Map, Resource Health, Synthetics Canaries, Evidently, RUM), and CloudWatch Metrics.

The main content area displays the log group path: CloudWatch > Log groups > /aws/containerinsights/hr-dev-eksdemo1/application > ip-10-0-1-67.ec2.internal-application.var.log.containers/myapp1-deployment-58ccb86d9-cdlw8\_default\_myapp1-container-dabcb25413b75d1ae2a369048f500e3f944a3efefff3f1a2fc697affb9e7cdc7.log. The title bar also shows the full log group path.

The "Log events" section contains a search bar, filter controls (View as text, Actions, Create Metric Filter), and time range buttons (Clear, 1m, 30m, 1h, 12h, Custom). A message indicates "There are older events to load. Load more." Below this, several log entries are listed, each starting with a timestamp and a log message. One specific log entry is highlighted with a blue border:

```
2022-05-22T11:02:17.602+05:30 {"log": "10.0.1.67 - - [22/May/2022:05:32:17 +0000] \"GET / HTTP/1.1\" 200 218 \"-\" \"curl/7.79.1\" \"-\"\n", "stream": "stdout", "kubernetes": { "pod_name": "myapp1-deployment-58ccb86d9-cdlw8", "namespace_name": "default", "pod_id": "ef0f45d5-57a1-4854-94d2-87e5665f9bcc", "host": "ip-10-0-1-67.ec2.internal", "container_name": "myapp1-container", "docker_id": "dabcb25413b75d1ae2a369048f500e3f944a3efefff3f1a2fc697affb9e7cdc7", "container_hash": "stacksimplify/kubenginx@sha256:205961b09a80476af4c2379841bf6abec0022101a7e6c5585a88316f7115d17a", "container_image": "stacksimplify/kubenginx:1.0.0" }}
```

A "Copy" button is located to the right of the highlighted log entry.

# AWS CloudWatch Container Insights

CloudWatch does not automatically create all possible metrics from the log data, to help you manage your Container Insights costs.

However, you can view additional metrics and additional levels of granularity by using CloudWatch Logs Insights to analyze the raw performance log events.

# EKS Control Plane Logs in AWS CloudWatch

CloudWatch > Log groups > /aws/eks/hr-dev-eksdemo1/cluster

## /aws/eks/hr-dev-eksdemo1/cluster

[Actions](#) ▾ [View in Logs Insights](#) [Search log group](#)

▶ Log group details

[Log streams](#) [Metric filters](#) [Subscription filters](#) [Contributors](#)

**Log streams (450+)**  
By default, we only load the most recent log streams. [Load more](#).

Filter loaded log streams or try prefix search

<input type="checkbox"/> Log stream
<input type="checkbox"/> kube-apiserver-audit-1eab02f7c0c0e63e88215e3df812f6f7
<input type="checkbox"/> kube-apiserver-audit-64e47452c4f948ed924e2372536ad86c

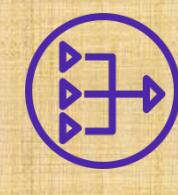
<input type="checkbox"/>	kube-scheduler-1eab02f7c0c0e63e88215e3df812f6f7
<input type="checkbox"/>	kube-controller-manager-1eab02f7c0c0e63e88215e3df812f6f7
<input type="checkbox"/>	authenticator-64e47452c4f948ed924e2372536ad86c
<input type="checkbox"/>	authenticator-1eab02f7c0c0e63e88215e3df812f6f7
<input type="checkbox"/>	kube-apiserver-1eab02f7c0c0e63e88215e3df812f6f7
<input type="checkbox"/>	kube-apiserver-64e47452c4f948ed924e2372536ad86c
<input type="checkbox"/>	kube-scheduler-64e47452c4f948ed924e2372536ad86c
<input type="checkbox"/>	cloud-controller-manager-1eab02f7c0c0e63e88215e3df812f6f7
<input type="checkbox"/>	cloud-controller-manager-64e47452c4f948ed924e2372536ad86c
<input type="checkbox"/>	kube-controller-manager-218438e5e8f42e38e56532ac2e202ee8



VPC



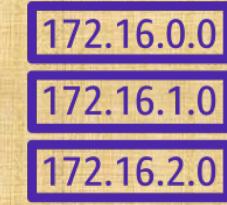
Internet gateway



NAT gateway



Elastic IP address



Route table



Public Subnet



Private Subnet



AWS S3



DynamoDB

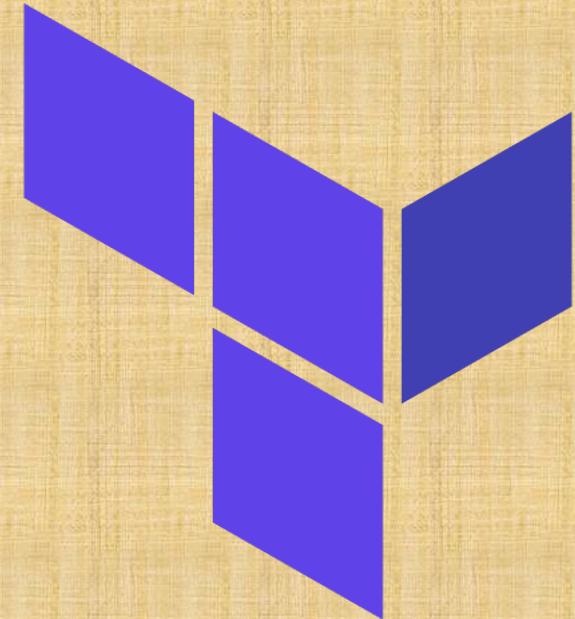


Fargate Profiles

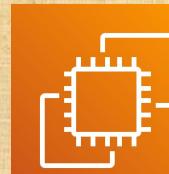
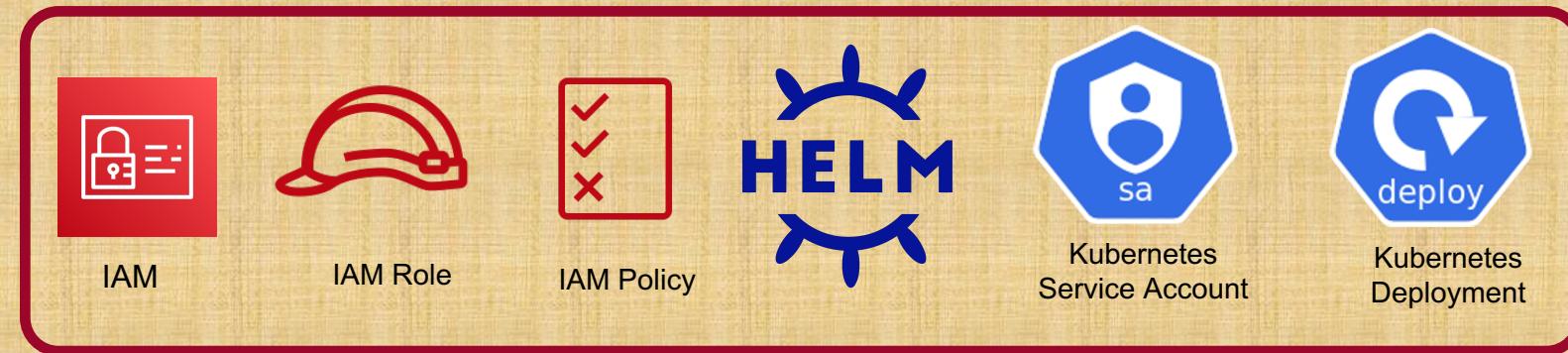


# AWS EKS

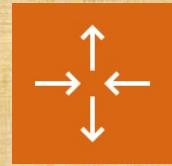
## Logging & Monitoring



EKS Cluster



EC2 VM



Autoscaling

**AWS CloudWatch Agent for Metrics and FluentBit for Logs**  
**Firstly, Implement Manually and Understand Concepts in detail**

# AWS CloudWatch Agent Deployment Modes

## AWS CloudWatch Agent Deployment Modes

### DaemonSet

CWA works at **node** level

CWA to serve for **all pods** within a Node

CWA to **collect metrics** for each Node.

CWA to be **scaled** according to the number of Nodes automatically.

### Service

CWA works at **cluster** level

CWA to serve for **all pods** within a Cluster

CWA as **centralized endpoint** within a cluster for your application to access

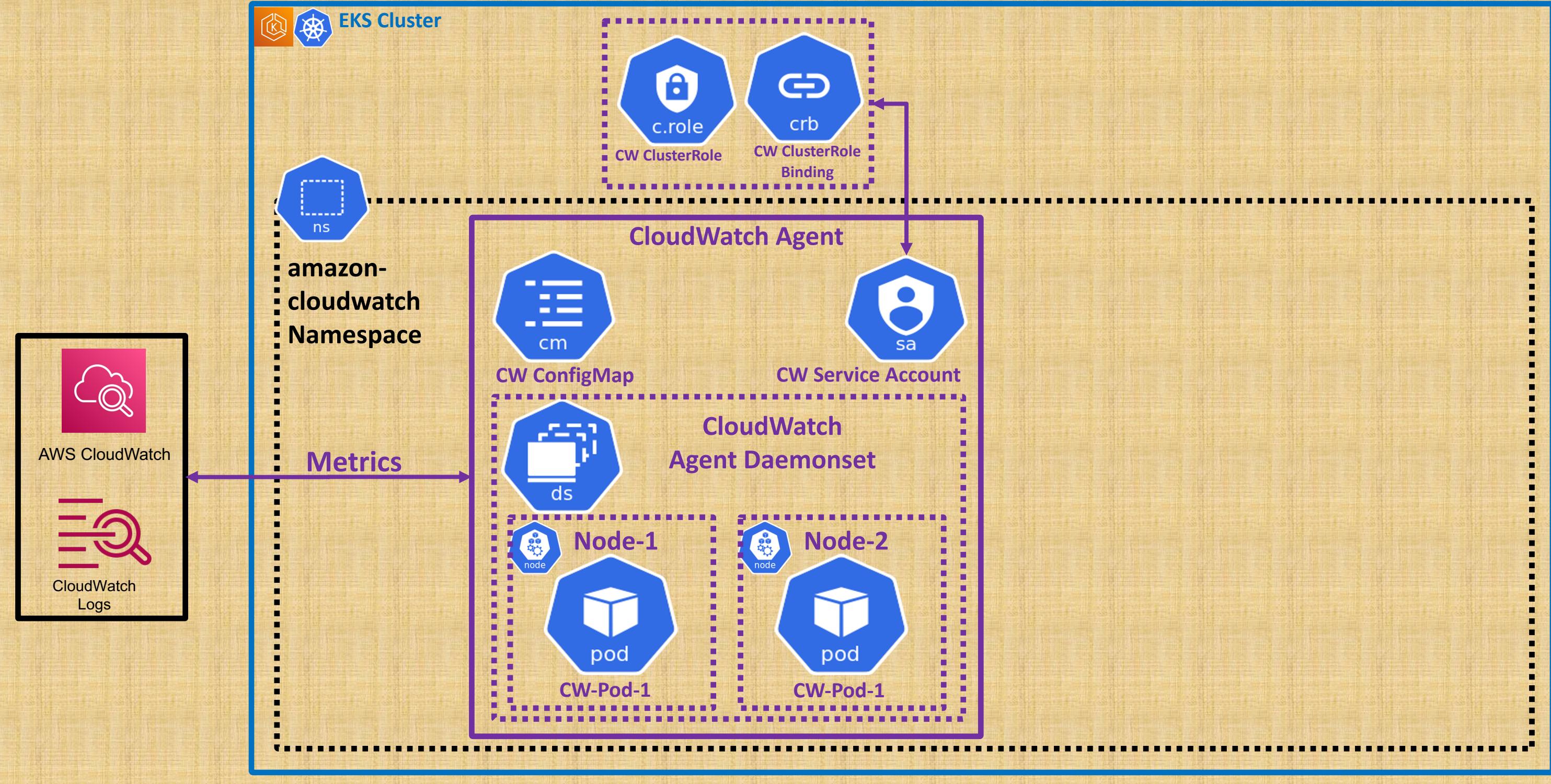
An **application outside** the cluster to be able **to access** to CWA

### Sidecar

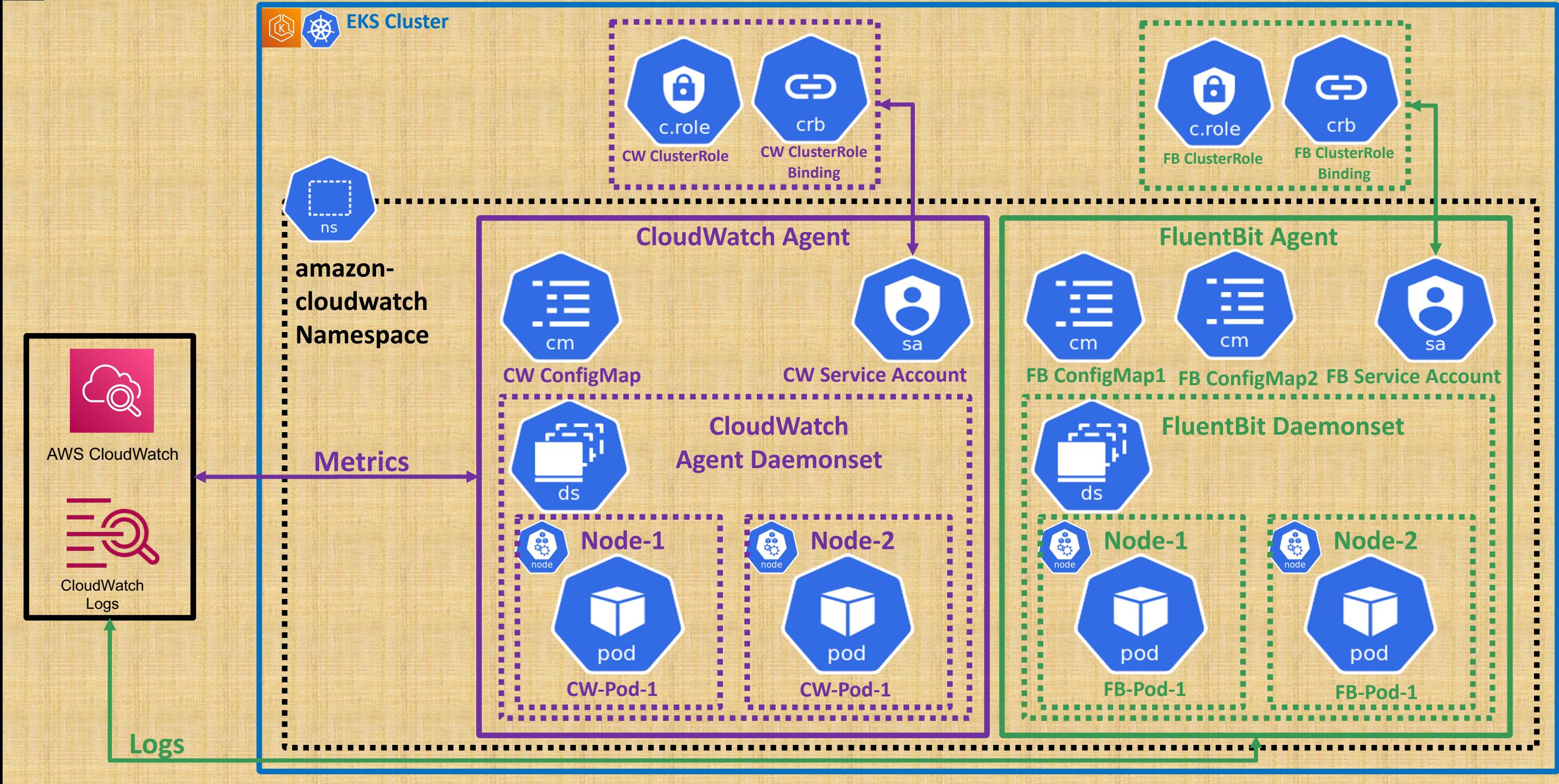
CWA works at **pod** level

**Dedicated** CWA Agent for your pod

# AWS EKS CloudWatch Container Insights



# AWS EKS CloudWatch Container Insights



# AWS CloudWatch Container Insights

Section-54

Install CloudWatch Agent

Install FluentBit Agent

Deploy using kubectl

Section-55

Install CloudWatch Agent

Install FluentBit Agent

Automate with Terraform

# What are we going to learn ?

## Terraform Manifests

```
✓ 54-EKS-Monitoring-Logging-kubectl
  > 01-ekscluster-terraform-manifests
    ✓ 02-cwagent-container-insights
      ! 01-cw-agent-configmap.yaml
      ! 02-cw-fluentbit-configmap.yaml
    ✓ 03-sample-app-test-container-insights
      ! 01-Deployment.yaml
      ! 02-CLB-LoadBalancer-Service.yaml
      ! 03-NLB-LoadBalancer-Service.yaml
```

01

EKS Cluster Terraform Manifests

02

CloudWatch Agent YAML Manifests

1. For the manifests which **needs modification** are added here in **Project-02**
2. Rest all will be directly used from **CloudWatch Agent Git Repo**

03

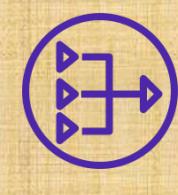
Simple **MyApp1** Deployment and Services in **YAML** format to **Test CloudWatch Container Insights**



VPC



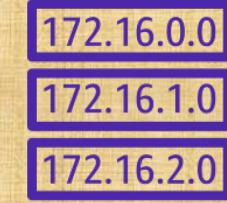
Internet gateway



NAT gateway



Elastic IP address



Route table



Public Subnet



Private Subnet



AWS S3



DynamoDB

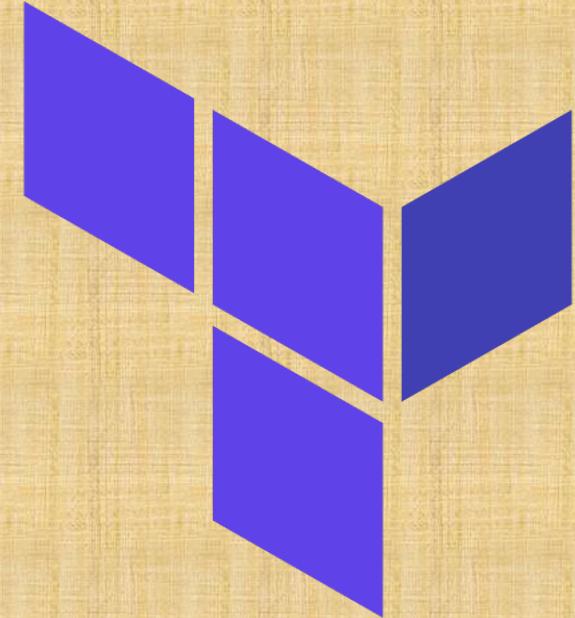


Fargate Profiles

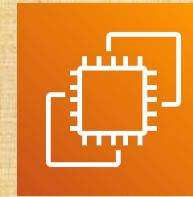
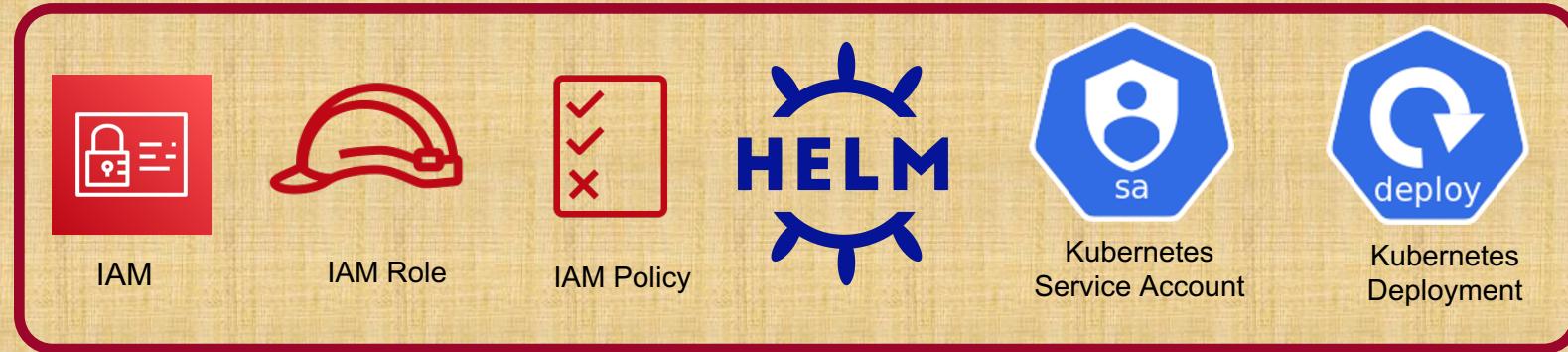


# AWS EKS

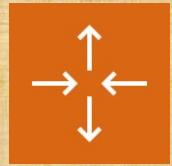
## Logging & Monitoring



EKS Cluster



EC2 VM



Autoscaling

**AWS CloudWatch Agent for Metrics and FluentBit for Logs**  
**Automate with Terraform**

# AWS CloudWatch Container Insights

Section-54

Install CloudWatch Agent

Install FluentBit Agent

Deploy using kubectl

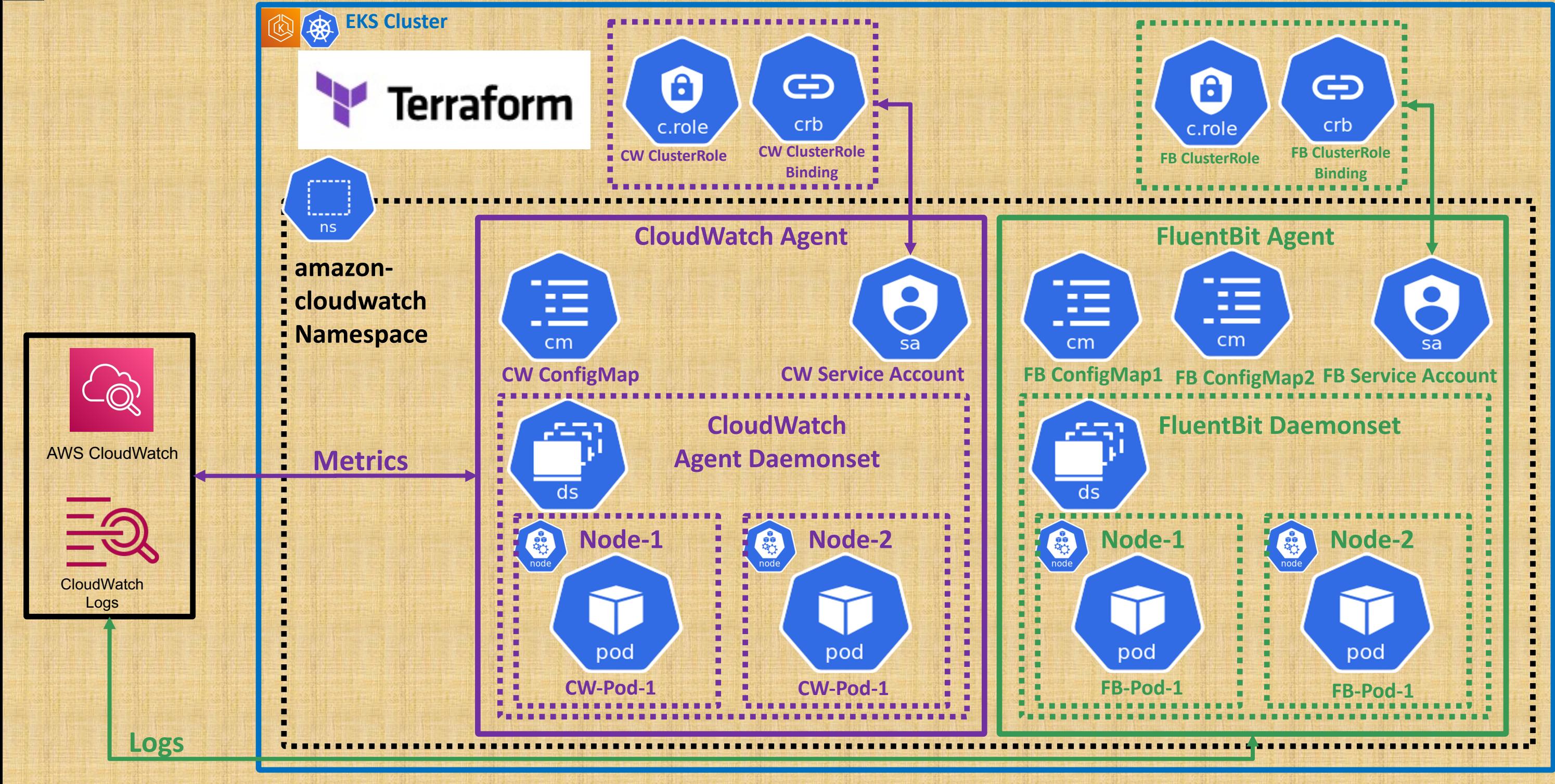
Section-55

Install CloudWatch Agent

Install FluentBit Agent

Automate with Terraform

# AWS EKS CloudWatch Container Insights



# Terraform Provider - kubectl

This provider helps us to manage Kubernetes resources in **Terraform**, by using **YAML** manifests

kubectl provider

✓ Resources

kubectl\_manifest

kubectl\_server\_version

✓ Data Sources

kubectl\_file\_documents

kubectl\_filename\_list

kubectl\_path\_documents

kubectl\_server\_version

kubectl

**kubectl apply -f my-apps/**

Terraform Provider URL:

<https://registry.terraform.io/providers/gavinbunney/kubectl/latest>

Terraform

```
resource "kubectl_manifest" "test" {  
    yaml_body = <<YAML  
    apiVersion: networking.k8s.io/v1  
    kind: Ingress  
    metadata:  
        name: test-ingress  
    annotations:  
        nginx.ingress.kubernetes.io/rewrite-target: /  
        azure/frontdoor: enabled  
    spec:  
        rules:  
        - http:  
            paths:  
            - path: /testpath  
              pathType: "Prefix"  
            backend:  
                serviceName: test  
                servicePort: 80  
YAML  
}
```

# Terraform Provider - kubectl

```
# Datasource
data "http" "get_cwagent_serviceaccount" {
  url = "https://raw.githubusercontent.com/aws-samples/aws-k8s-eks-cwagent-distro/main/cwagent-serviceaccount.yaml"
  # Optional request headers
  request_headers = {
    Accept = "text/*"
  }
}
```

1

```
# Datasource: kubectl_file_documents
# This provider provides a data resource kubectl_file_documents
data "kubectl_file_documents" "cwagent_docs" {
  content = data.http.get_cwagent_serviceaccount.body
}
```

2

```
# Resource: kubectl_manifest which will create k8s Resources from
resource "kubectl_manifest" "cwagent_serviceaccount" {
  depends_on = [kubernetes_namespace_v1.amazon_cloudwatch]
  for_each = data.kubectl_file_documents.cwagent_docs.manifests
  yaml_body = each.value
}
```

3

- ✓ 55-EKS-Monitoring-Logging-Terraform
  - > 01-ekscluster-terraform-manifests
  - ✓ 02-cloudwatchagent-fluentbit-terraform-manifests
    - └ c1-versions.tf
    - └ c2-remote-state-datasource.tf
    - └ c3-01-generic-variables.tf
    - └ c3-02-local-values.tf
    - └ c4-01-terraform-providers.tf
    - └ c4-02-cwagent-namespace.tf
    - └ c4-03-cwagent-service-accounts-cr-crb.tf
    - └ c4-04-cwagent-configmap.tf
    - └ c4-05-cwagent-daemonset.tf
    - └ c5-01-fluentbit-configmap.tf
    - └ c5-02-fluentbit-daemonset.tf
    - └ terraform.tfvars
  - ✓ 03-sample-app-test-container-insights
    - ! 01-Deployment.yaml
    - ! 02-CLB-LoadBalancer-Service.yaml
    - ! 03-NLB-LoadBalancer-Service.yaml

# What are we going to learn ?

## Project Folders

01

EKS Cluster Terraform Manifests

02

CloudWatch Agent & Fluent Bit Terraform Manifests

03

Simple MyApp1 Deployment and Services in YAML  
format to Test CloudWatch Container Insights

# What are we going to learn ?

- ✓ 55-EKS-Monitoring-Logging-Terraform
  - > 01-ekscluster-terraform-manifests
  - ✓ 02-cloudwatchagent-fluentbit-terraform-manifests
    - ⚡ c1-versions.tf
    - ⚡ c2-remote-state-datasource.tf
    - ⚡ c3-01-generic-variables.tf
    - ⚡ c3-02-local-values.tf
    - ⚡ c4-01-terraform-providers.tf
    - ⚡ c4-02-cwagent-namespace.tf
    - ⚡ c4-03-cwagent-service-accounts-cr-crb.tf
    - ⚡ c4-04-cwagent-configmap.tf
    - ⚡ c4-05-cwagent-daemonset.tf
    - ⚡ c5-01-fluentbit-configmap.tf
    - ⚡ c5-02-fluentbit-daemonset.tf
    - ⚡ terraform.tfvars
  - > 03-sample-app-test-container-insights

## Terraform Manifests

c1- c3

Terraform Settings Block, Remote State Datasource, Generic Input Variables and Local Values

c4-01

Terraform Providers (AWS, Kubernetes, http, Kubectl)

c4-02

Create Namespace **amazon-cloudwatch**

c4-03

CloudWatch Agent Service Account, Cluster Role, Cluster Role Binding

c4-04

CloudWatch Agent **ConfigMap**  
(Needs some edits from git repo version)

c4-05

CloudWatch Agent DaemonSet

c5-01

FluentBit **ConfigMap**  
(Needs some edits from git repo version)

c5-02

FluentBit DaemonSet and Other Resources



Thank You