# AWS EKS Kubernetes - Masterclass | DevOps, Microservices

Kalyan Reddy Daida

# AWS EKS
# Course Outline

STACKSIMPLIFY

# Kubernetes On Cloud Roadmap

- Kubernetes for Beginners On Cloud
- AWS EKS Kubernetes - Masterclass | DevOps, Microservices
- Azure AKS Kubernetes - Masterclass | DevOps, Microservices
- Google GKE Kubernetes - Masterclass | DevOps, Microservices
- Docker & Kubernetes for Java Spring Boot Developers on AWS
- Master HELM3 with Kubernetes on AWS & Azure

Kubernetes Concepts

AWS Services Integration with EKS

DevOps

Microservices

Architecture
Pods
ReplicaSets
Deployments
Node Port Service
Cluster IP Service
External Name Service
Ingress Service
Ingress SSL
Ingress & External DNS

kubectl - Imperative
Declarative with YAML
Secrets
Init Containers
Probes
Requests & Limits
Namespaces
Limit Range
Resource Quota
Storage Classes

Persistent Volumes
PVC
Load Balancers
Annotations
Canary Deployments
HPA
VPA
DaemonSets
Fluentd for logs
ConfigMaps

KUBERNETES

AWS SERVICES

DevOps

AWS Developer Tools


AWS CodeCommit


AWS CodeBuild


AWS CodePipeline

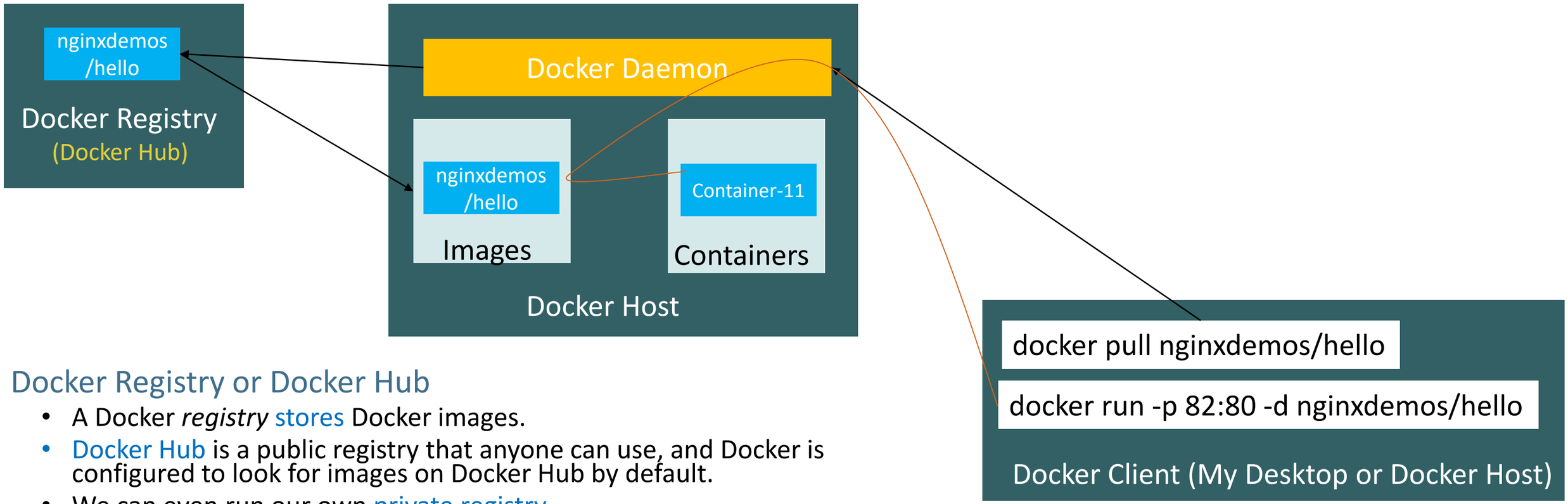
Microservices

Service Discovery

Distributed Tracing

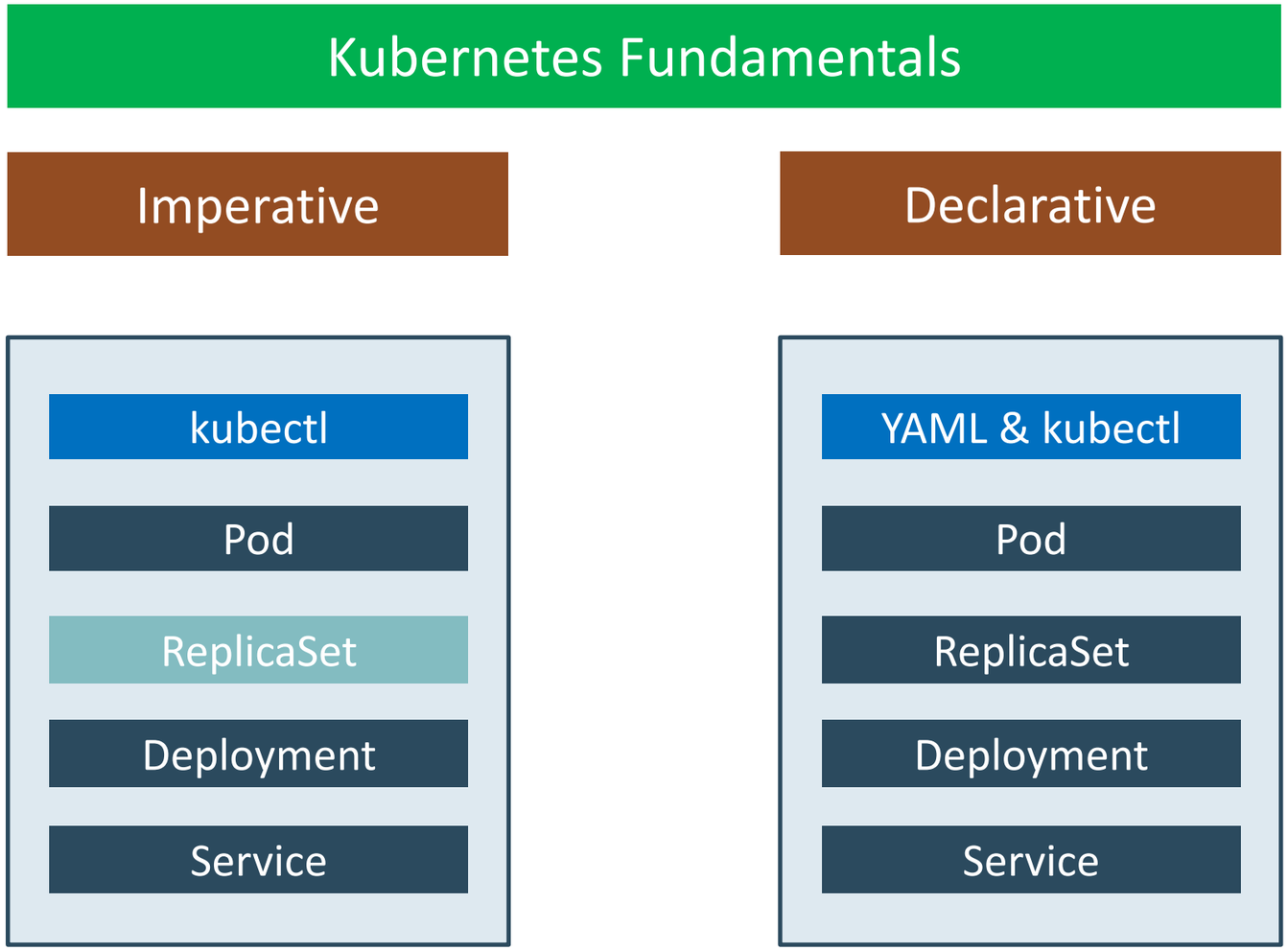
Canary Deployments

# Docker - Fundamentals




- Docker Registry or Docker Hub
  - A Docker *registry* stores Docker images.
  - Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default.
  - We can even run our own private registry.
  - When we use the docker pull or docker run commands, the required images are pulled from our configured registry.
  - When we use the docker push command, our image is pushed to our configured registry.

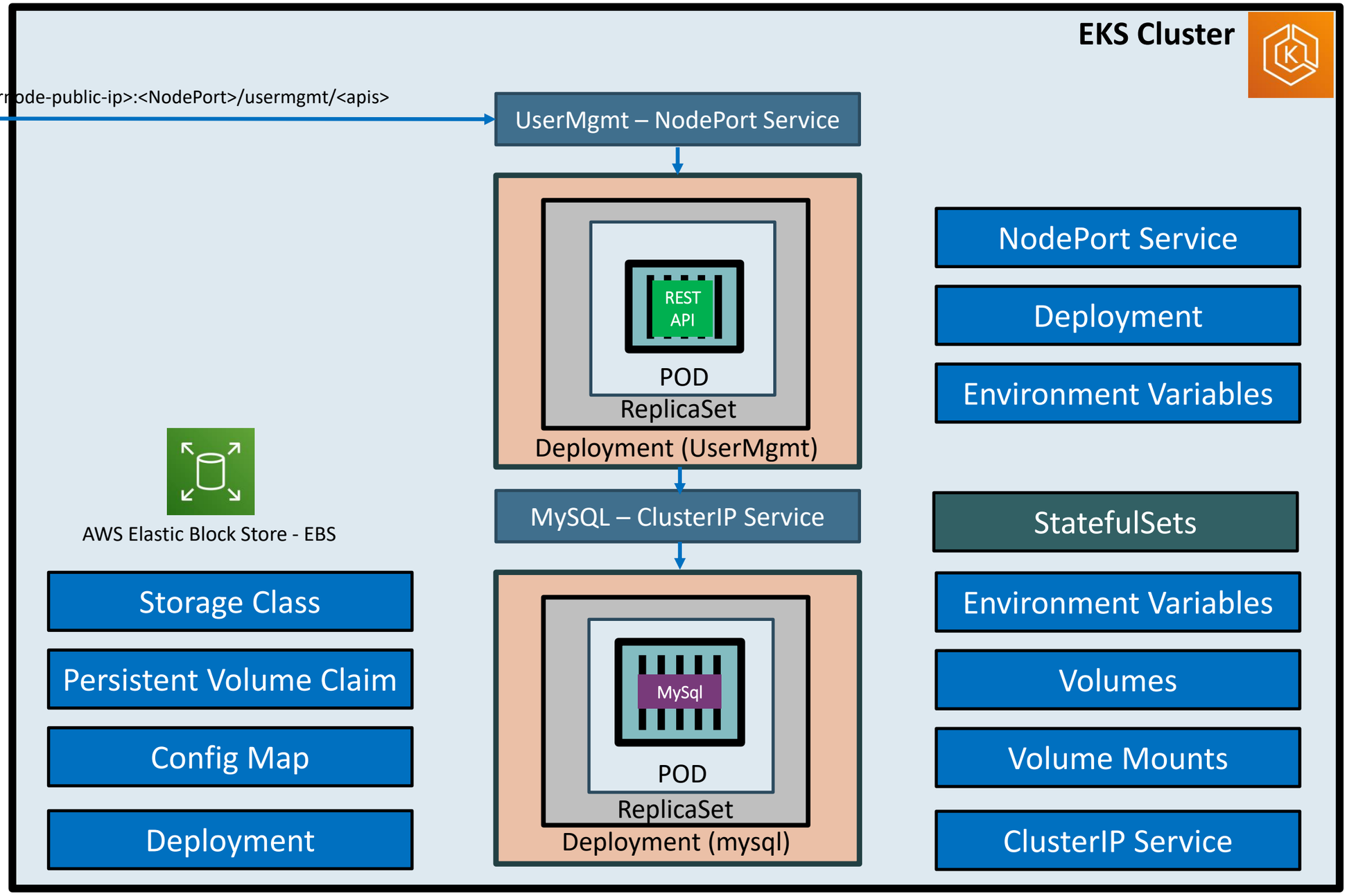
# Kubernetes - Imperative & Declarative

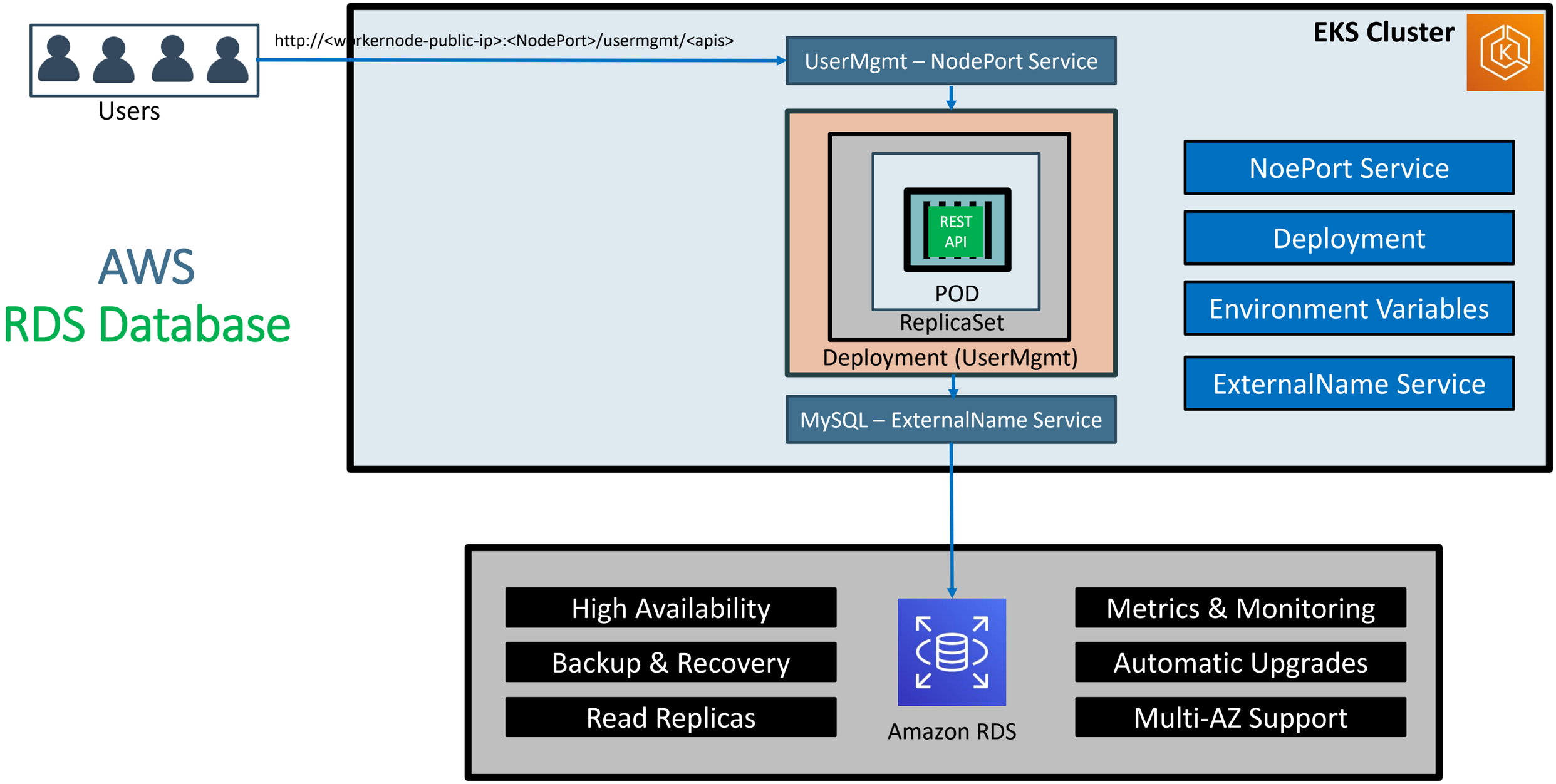


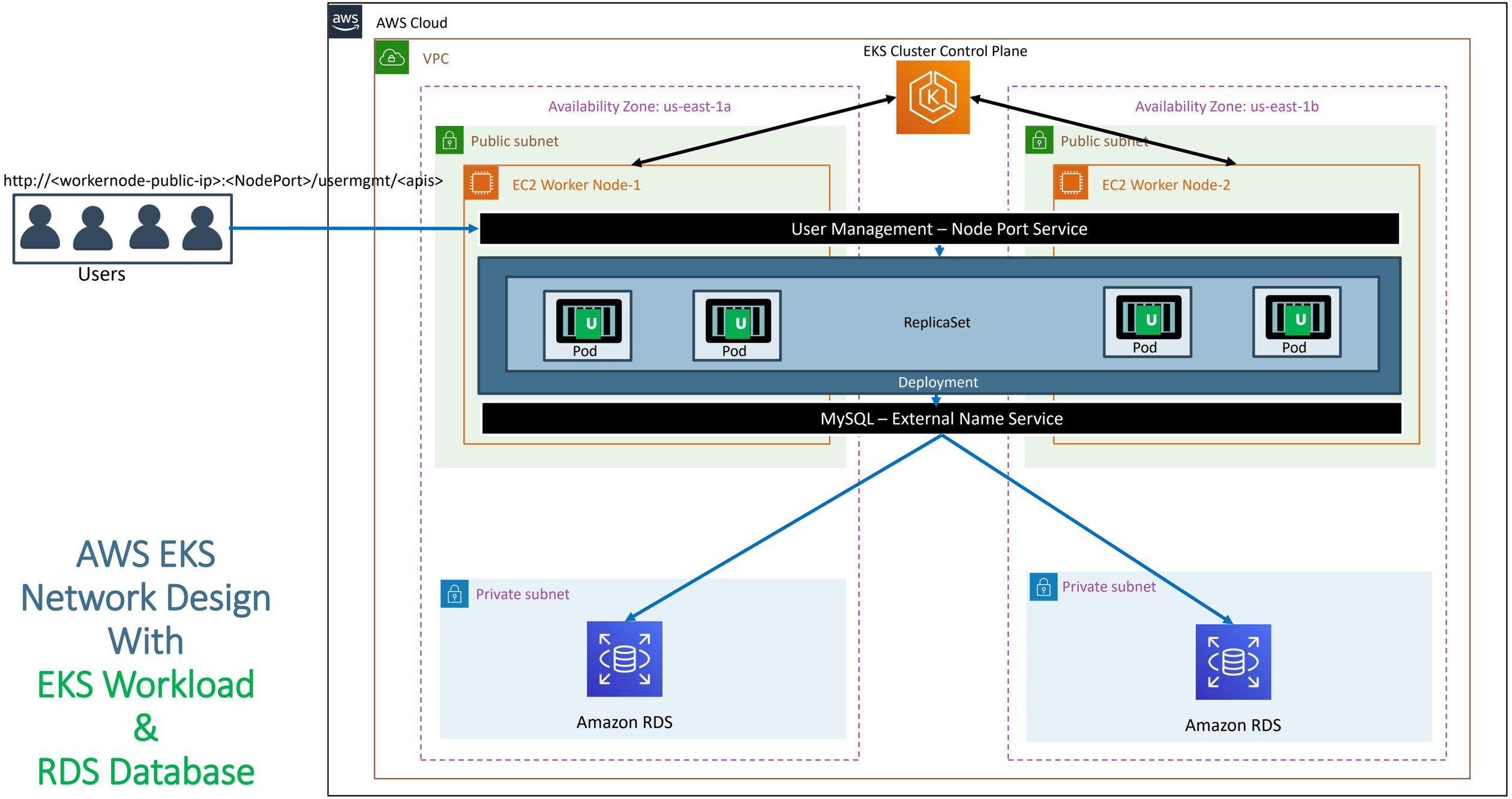
# EKS Storage
# EBS CSI Driver

- Drawbacks of EBS CSI for MySQL DB
- Complex setup to achieve HA
- Complex Multi-Az support for EBS
- Complex Master-Master MySQL setup
- Complex Master-Slave MySQL setup
- No Automatic Backup & Recovery
- No Auto-Upgrade MySQL

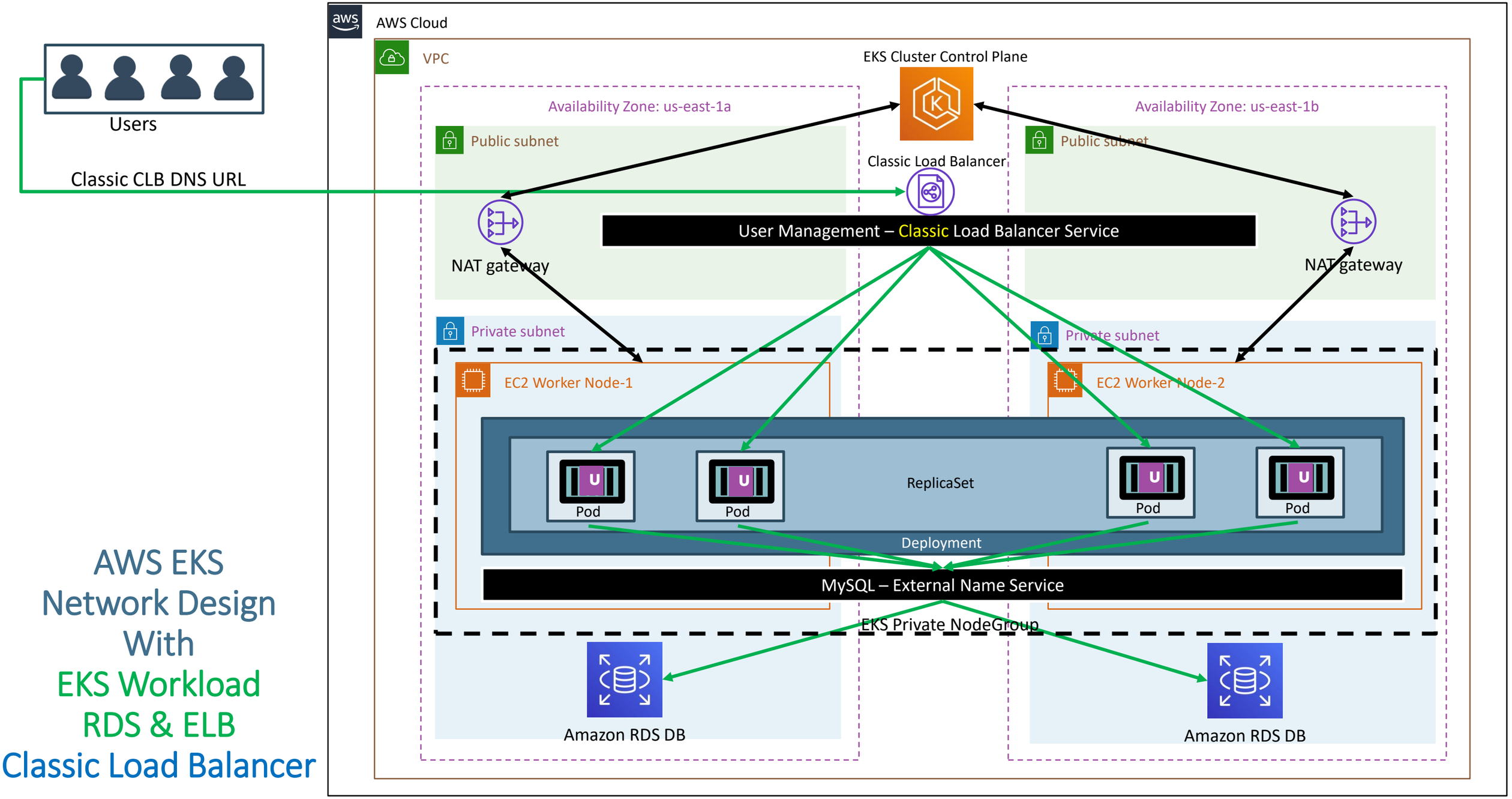

http://<workernode-public-ip>:<NodePort>/usermgmt/<apis>



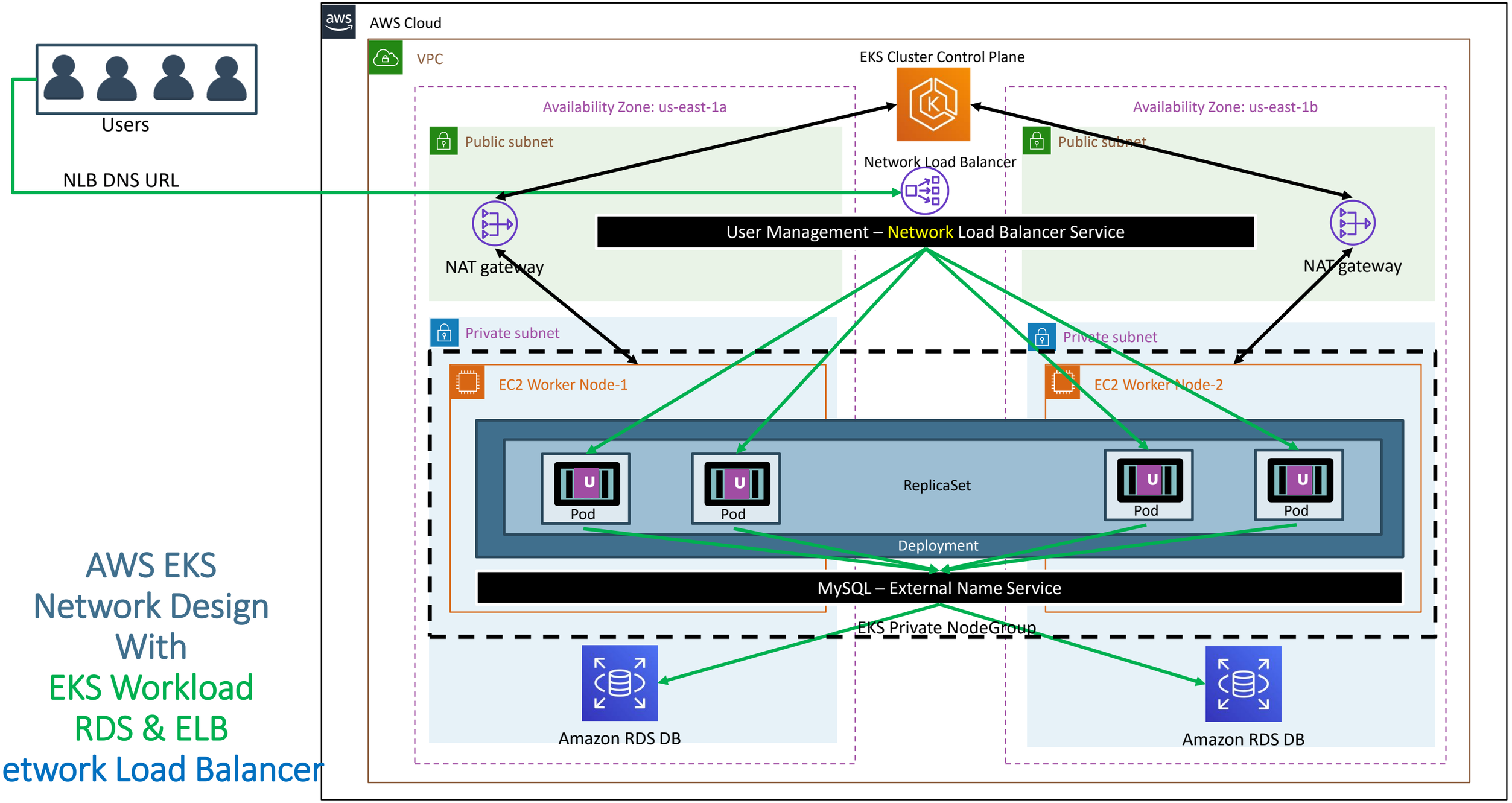




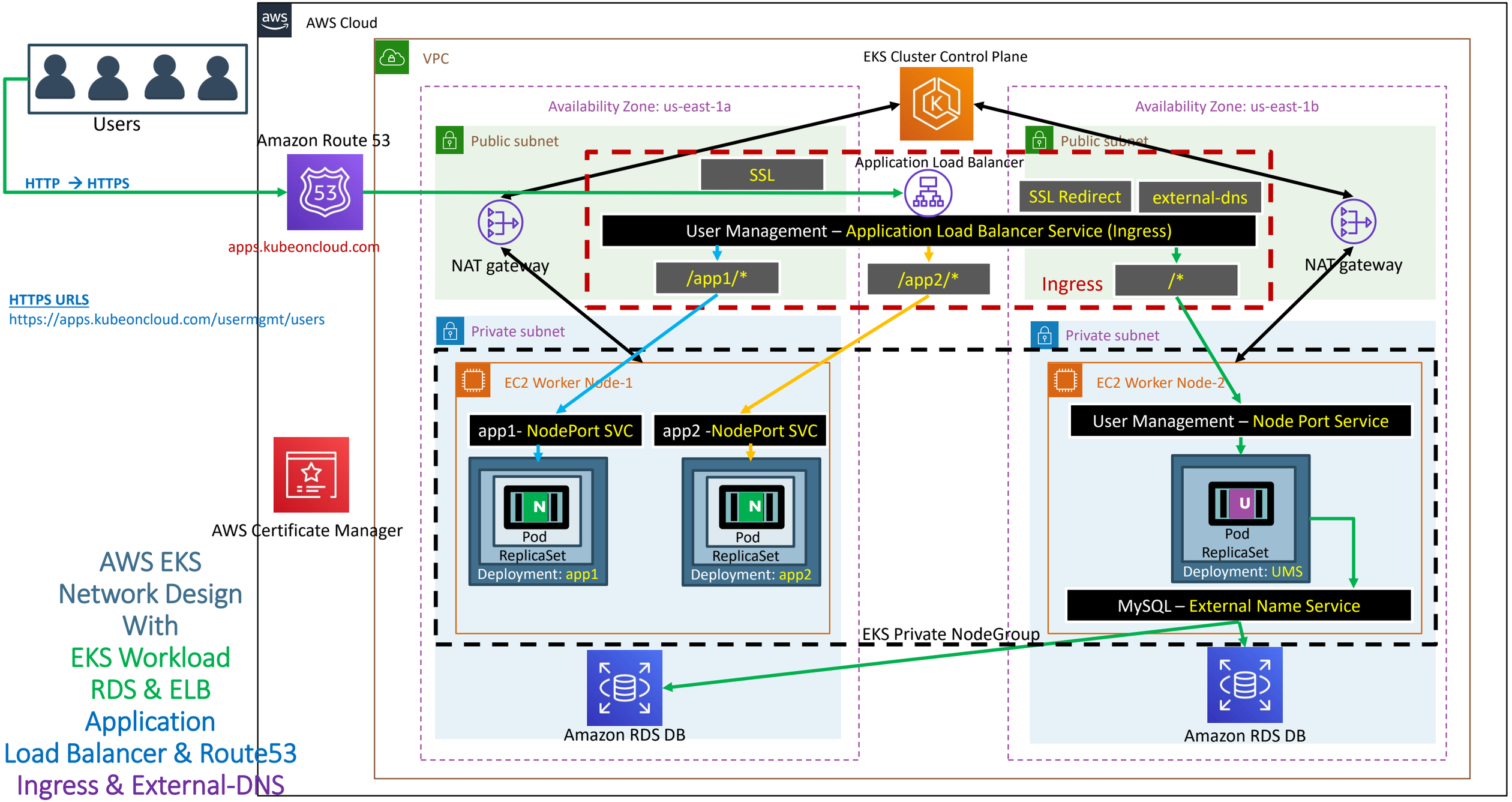

# AWS EKS Network Design With EKS Workload & RDS Database



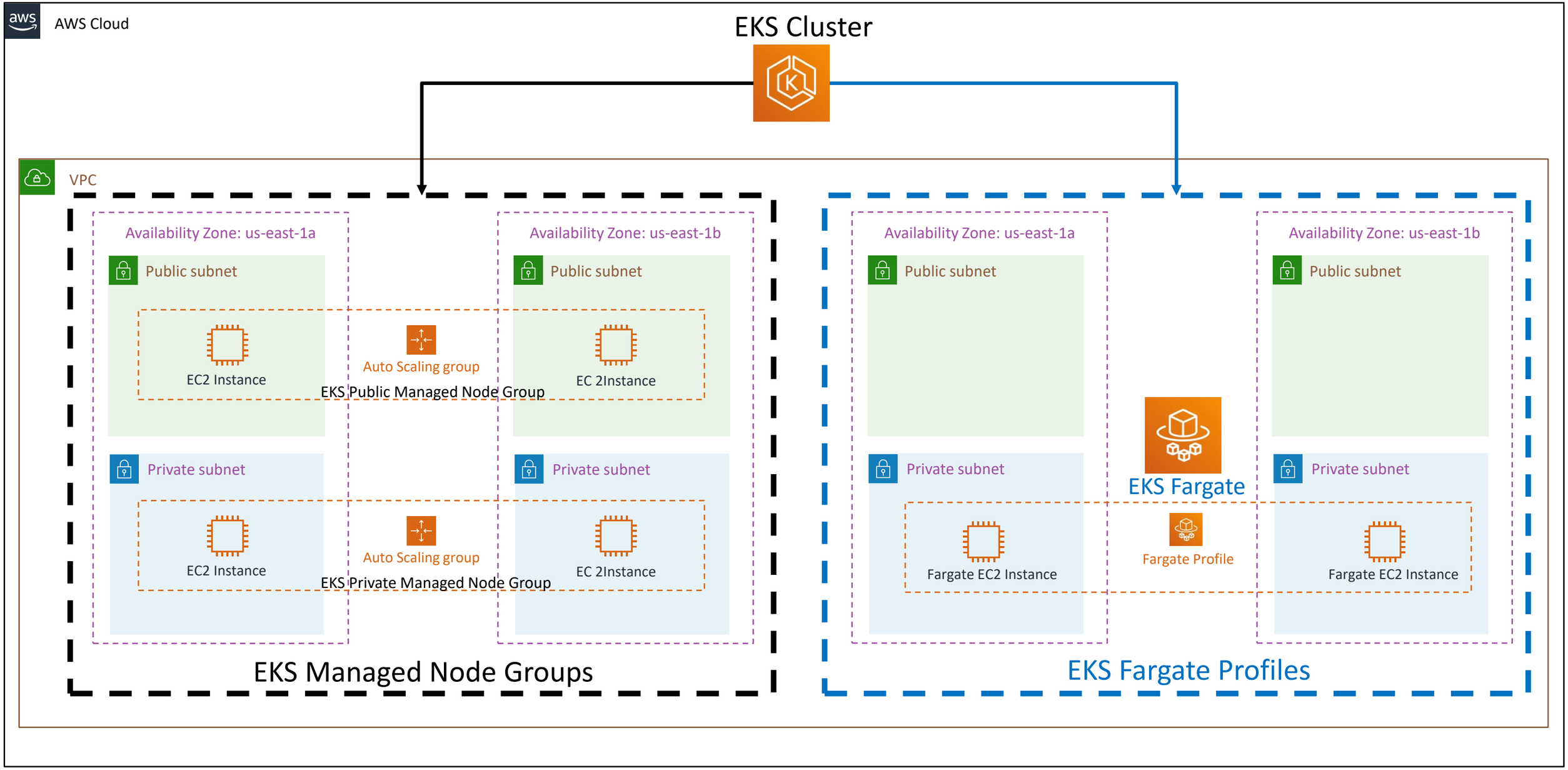

# AWS EKS Network Design With EKS Workload RDS & ELB Classic Load Balancer



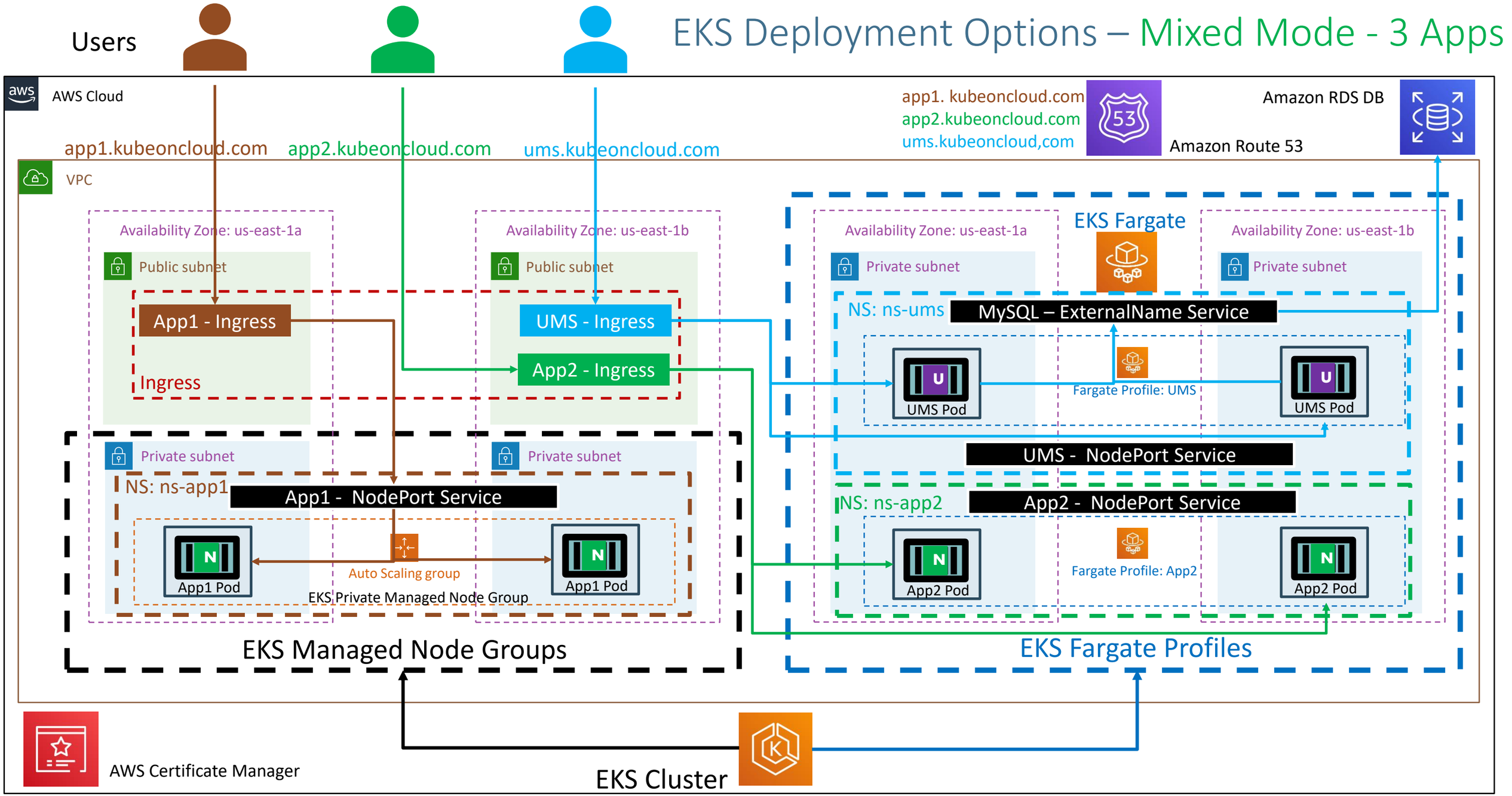

# AWS EKS Network Design With EKS Workload RDS & ELB Network Load Balancer


AWS Cloud
VPC
EKS Cluster Control Plane
Users
Amazon Route 53
HTTP → HTTPS
apps.kubeoncloud.com
Availability Zone: us-east-1a
Availability Zone: us-east-1b
Public subnet
Public subnet
Application Load Balancer
SSL
SSL Redirect
external-dns
User Management – Application Load Balancer Service (Ingress)
NAT gateway
NAT gateway
/app1/*
/app2/*
Ingress
/*
HTTPS URLS
https://apps.kubeoncloud.com/usermgmt/users
Private subnet
Private subnet
EC2 Worker Node-1
EC2 Worker Node-2
app1- NodePort SVC
app2 -NodePort SVC
User Management – Node Port Service
Pod
ReplicaSet
Deployment: app1
Pod
ReplicaSet
Deployment: app2
Pod
ReplicaSet
Deployment: UMS
MySQL – External Name Service
AWS Certificate Manager
EKS Private NodeGroup
Amazon RDS DB
Amazon RDS DB
AWS EKS
Network Design
With
EKS Workload
RDS & ELB
Application
Load Balancer & Route53
Ingress & External-DNS

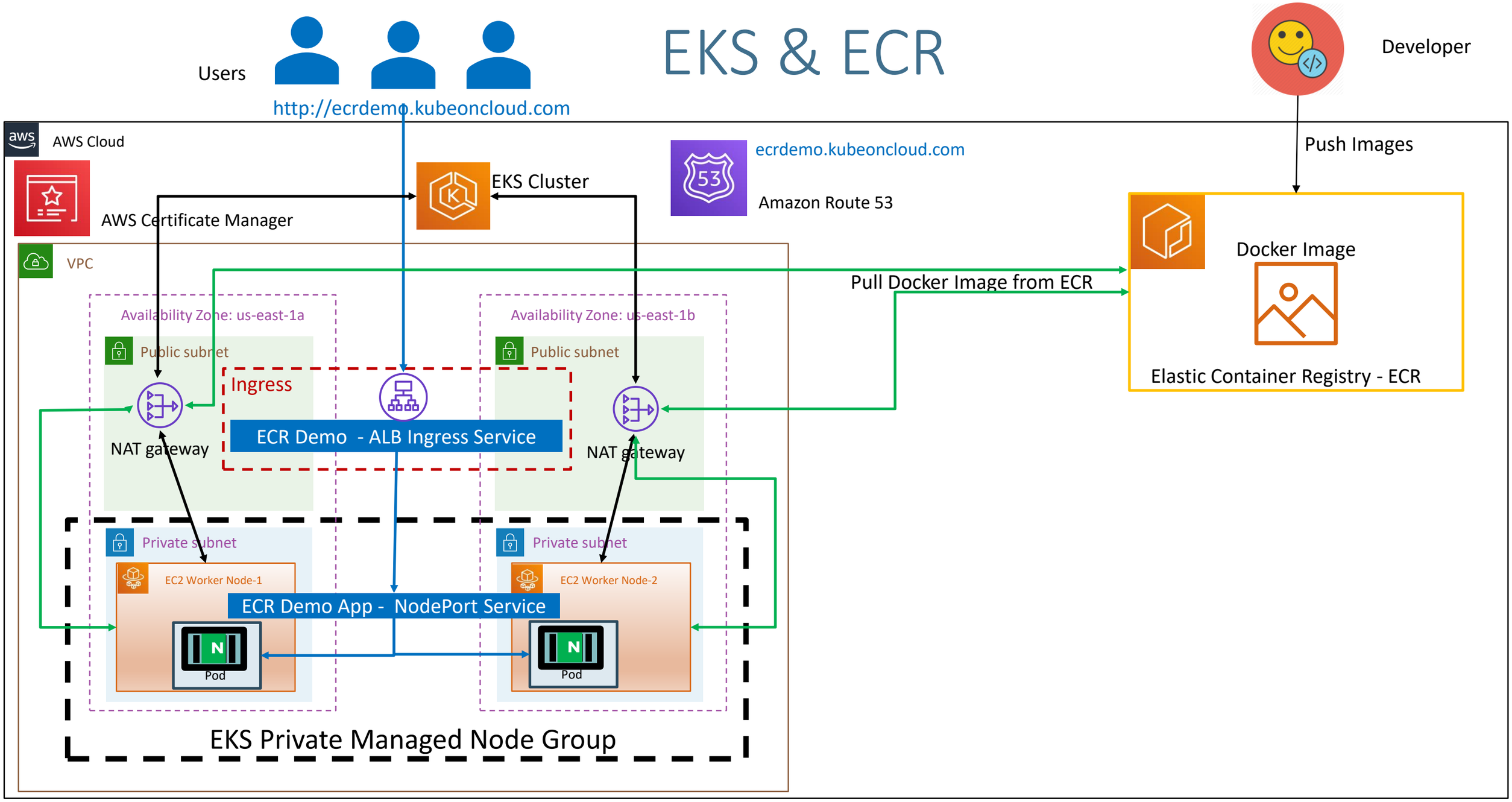
# EKS Deployment Options - Mixed



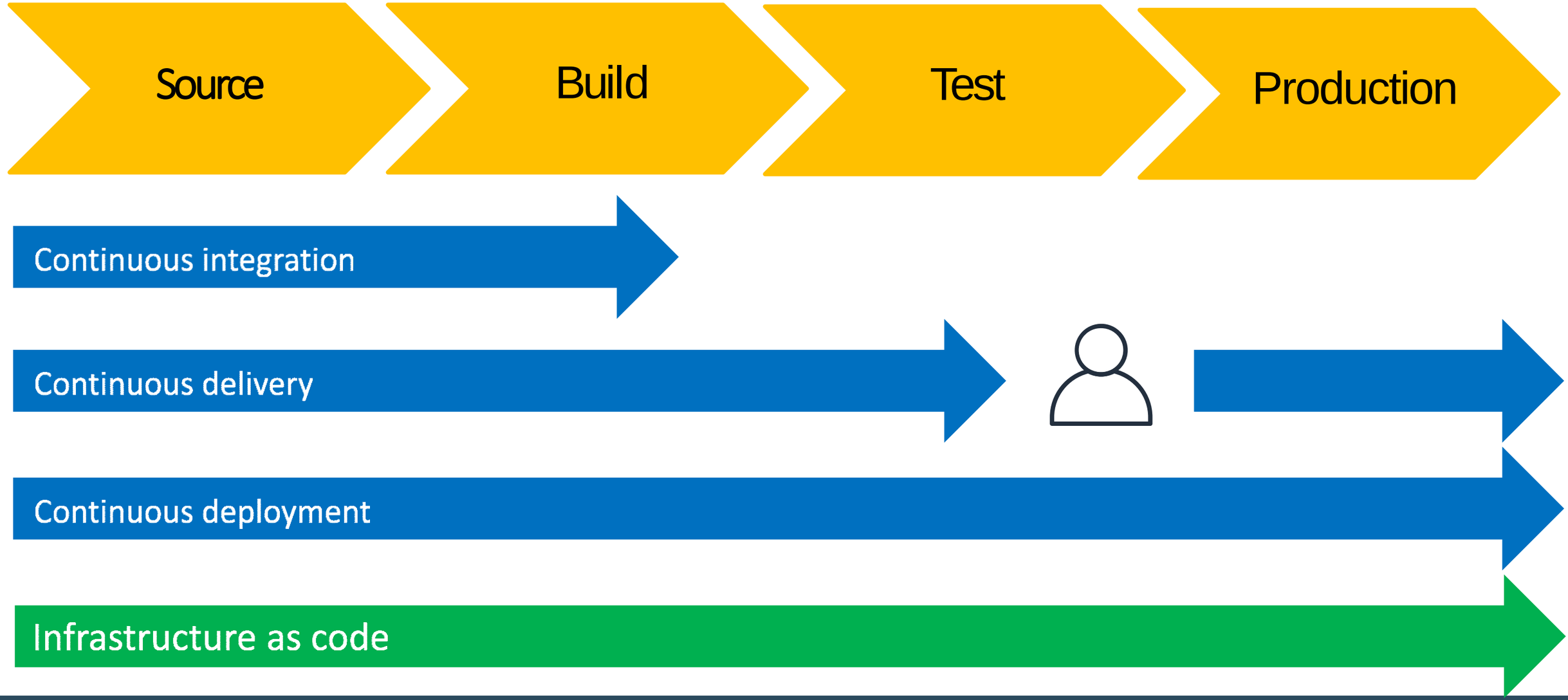
# EKS Deployment Options – Mixed Mode - 3 Apps



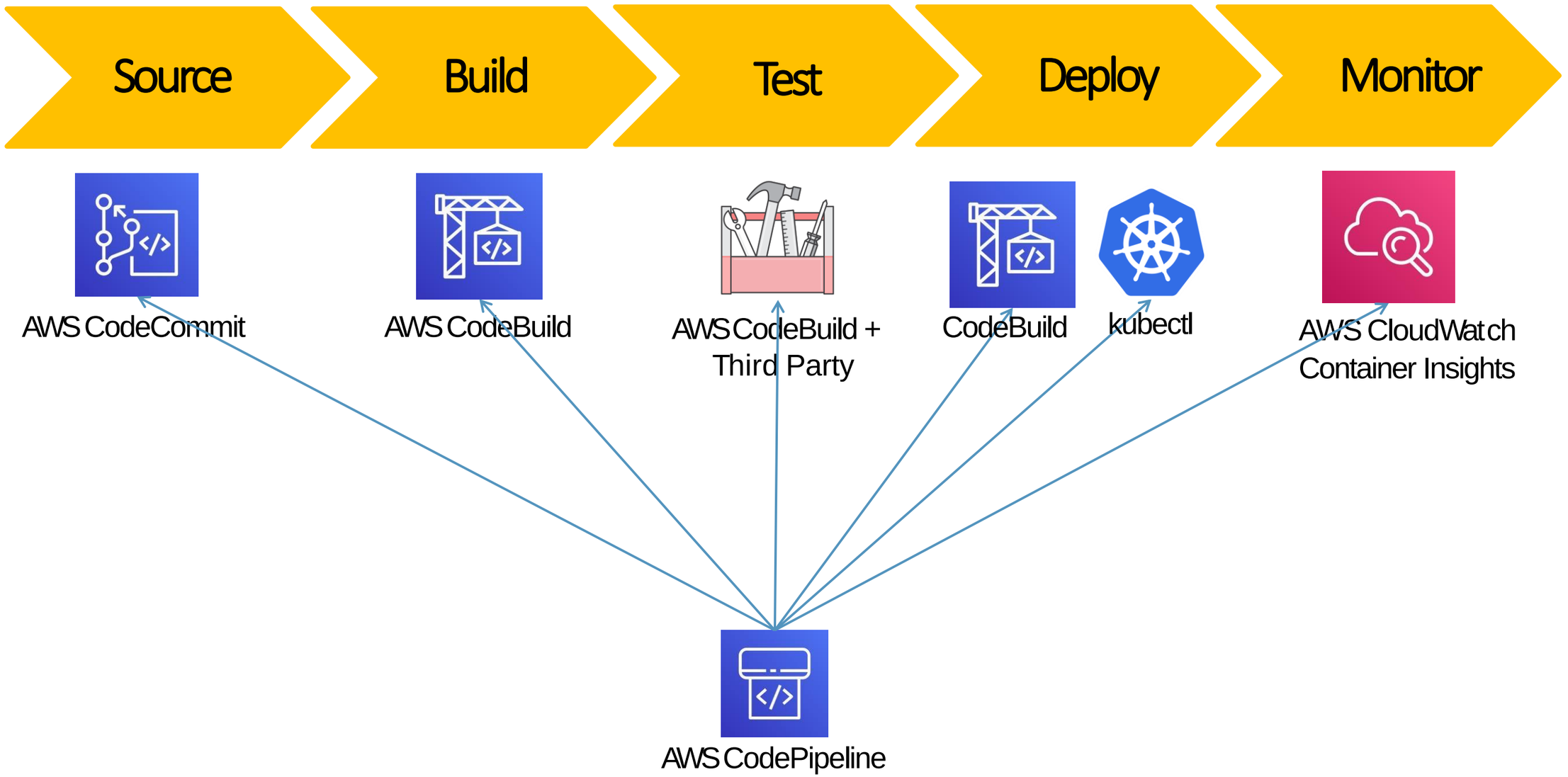
# EKS & ECR



# Stages in Release Process



# AWS Developer Tools or Code Services



# AWS Developer Tools or Code Services


AWS CodeCommit

AWS CodeBuild

AWS CodeBuild + Third Party

CodeBuild

kubectl

AWS CloudWatch Container Insights

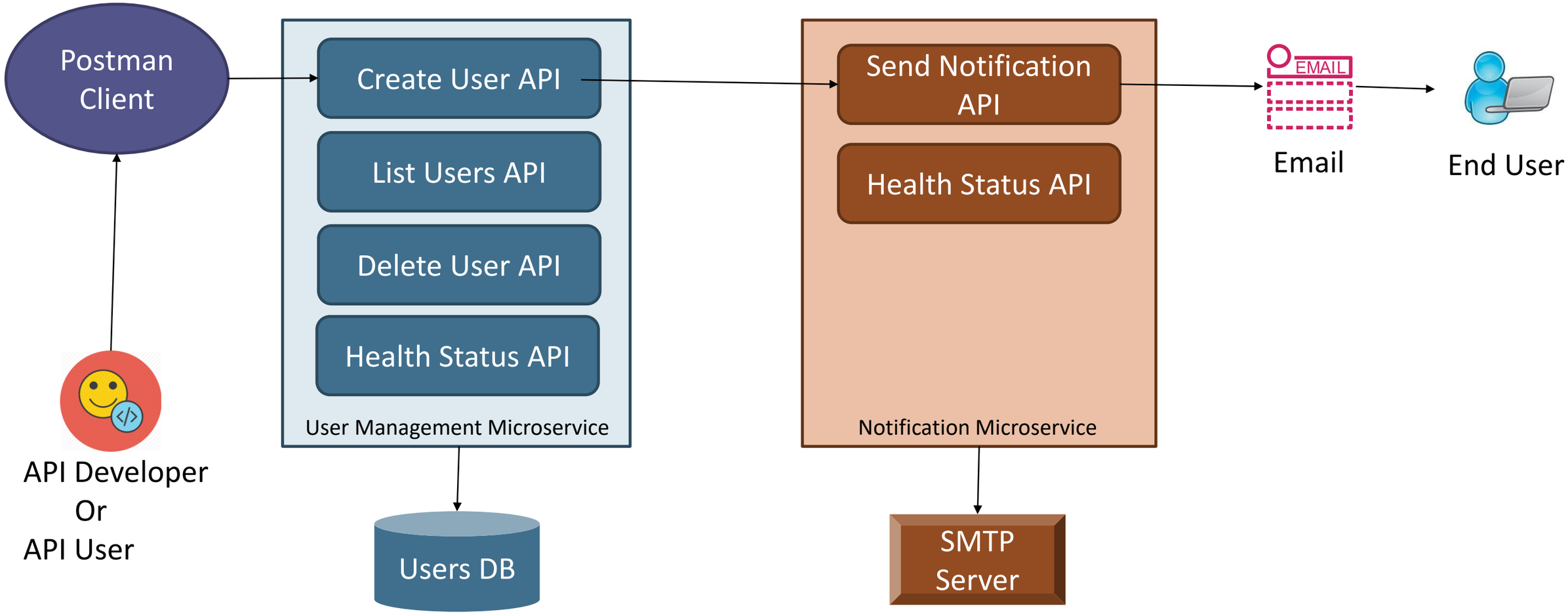
Continuous integration

Continuous delivery

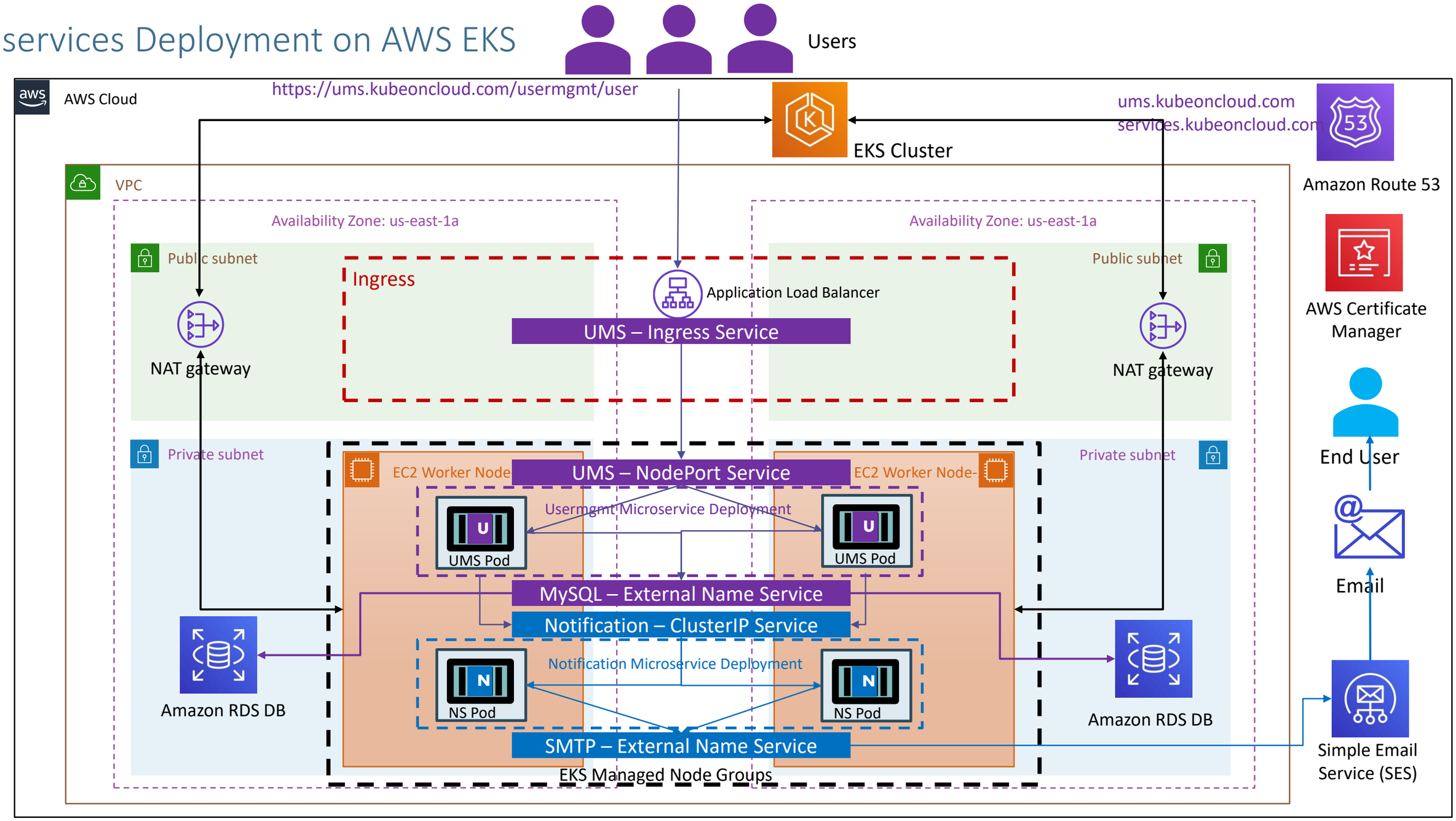
Continuous deployment

AWS CodePipeline

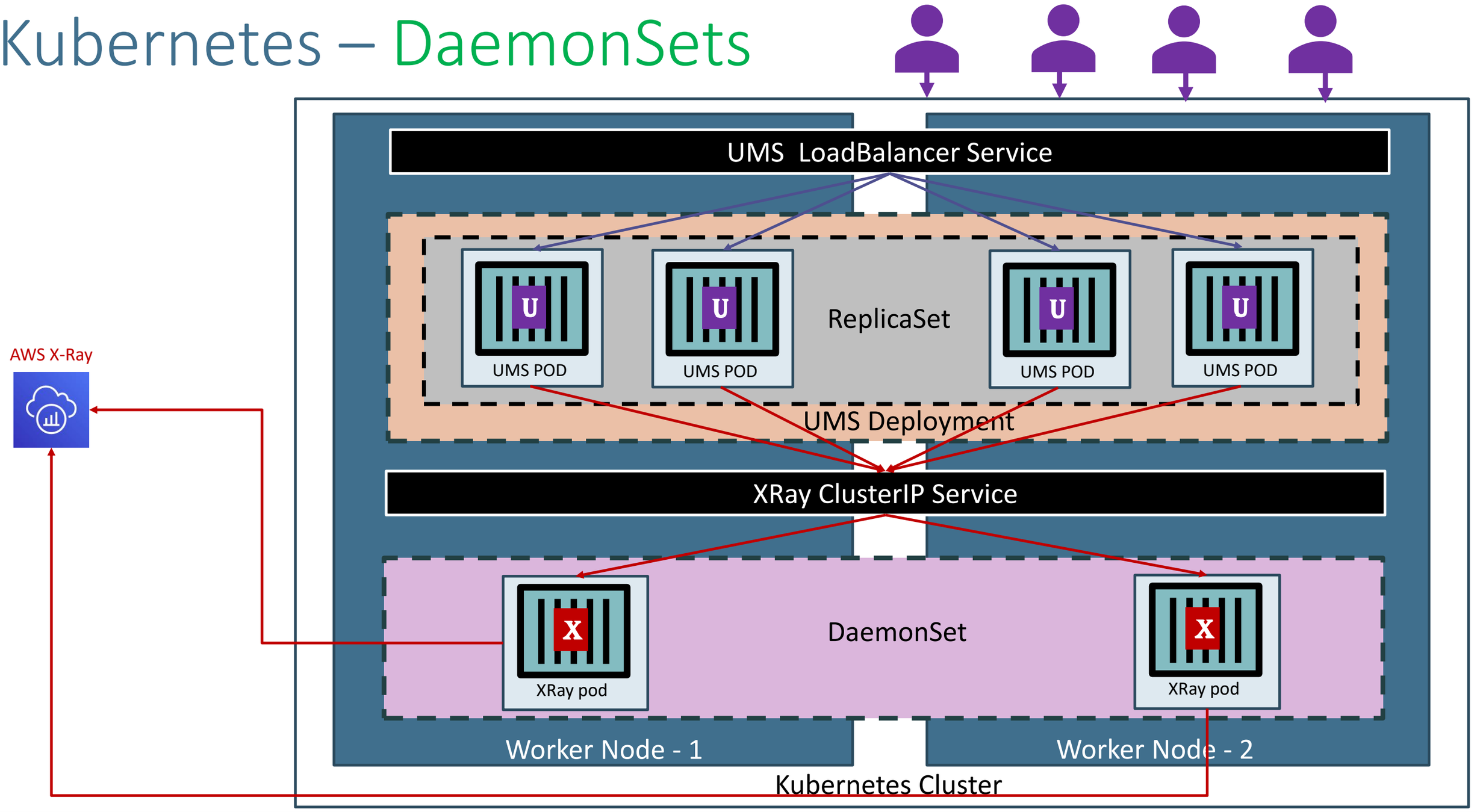
# Microservices



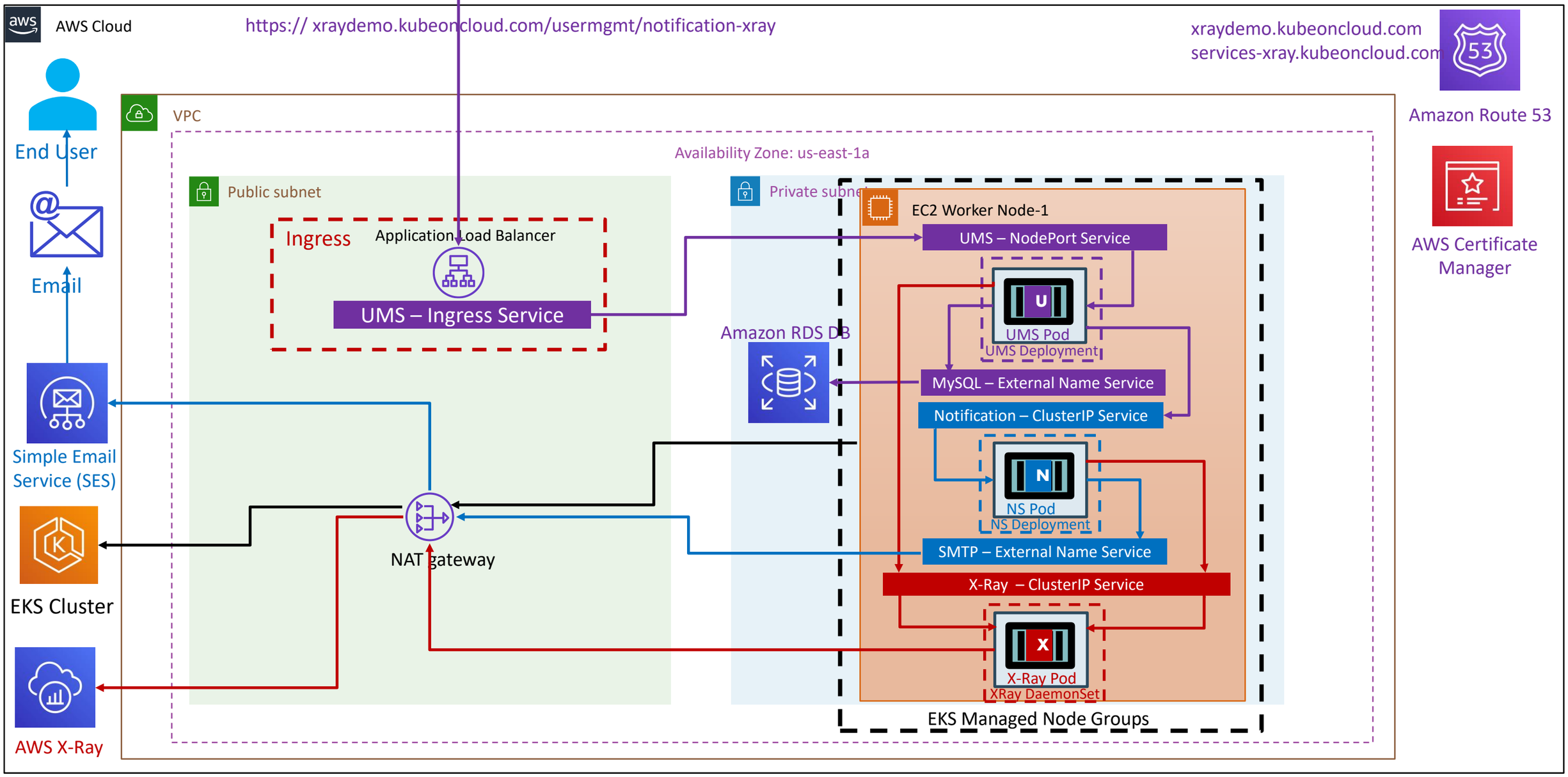
# Microservices Deployment on AWS EKS



# Kubernetes – DaemonSets

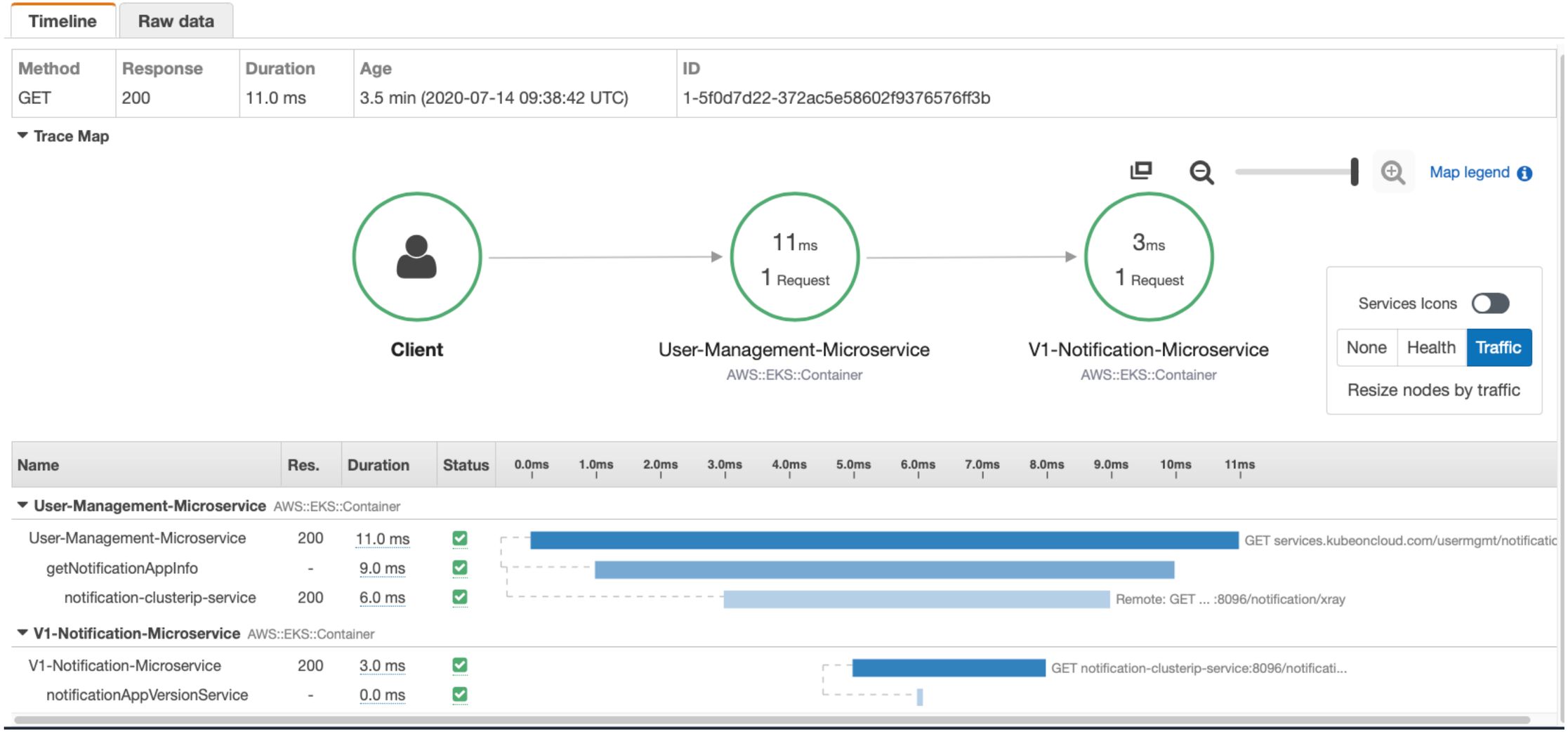


# Microservices Distributed Tracing with AWS X-Ray

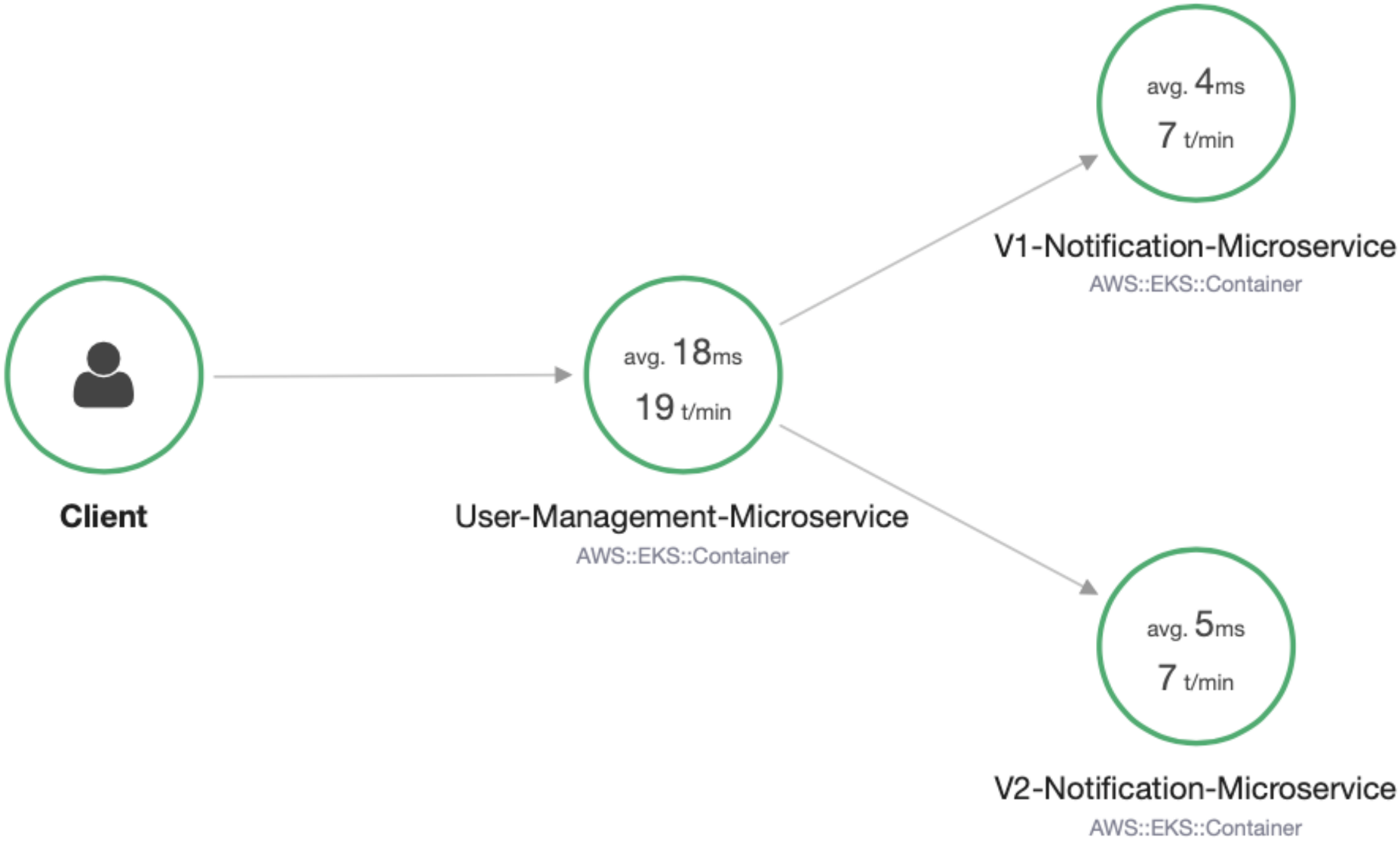


# AWS X-Ray – Service Map

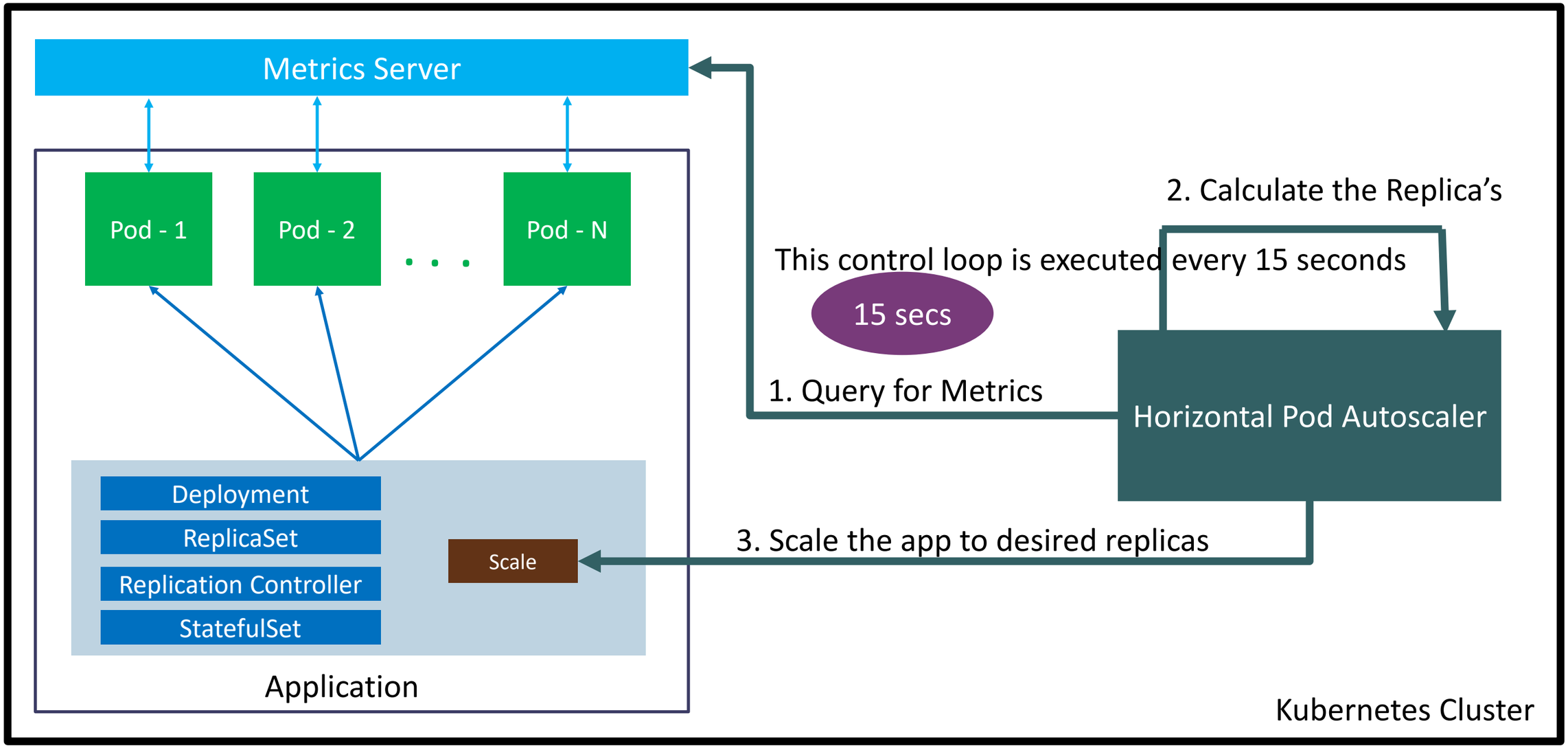
# AWS X-Ray - Traces



# Microservices – Canary Deployments



# How HPA works?



Container Map

Container Resources

Performance Dashboards

Log Groups

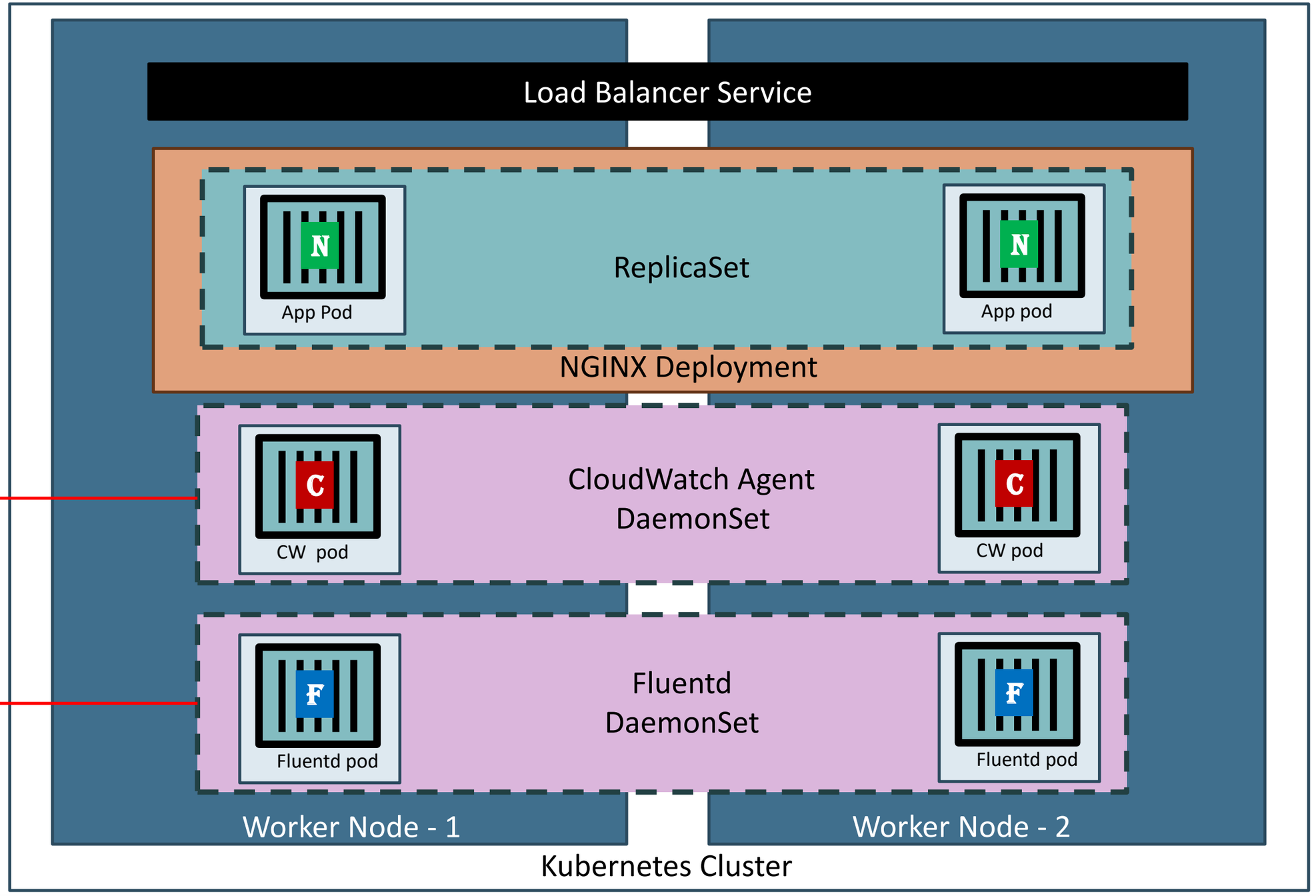
Log Insights

Alarms

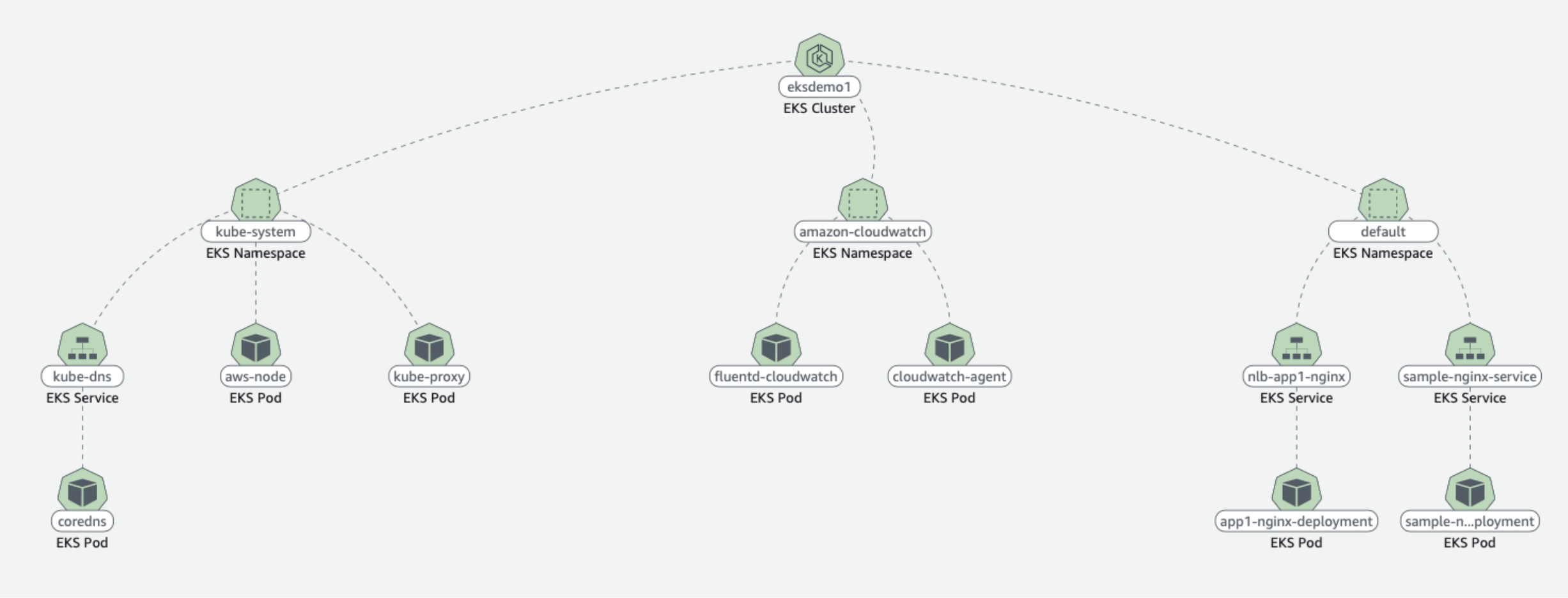
Developer or Operations User

CloudWatch

# Container Insights



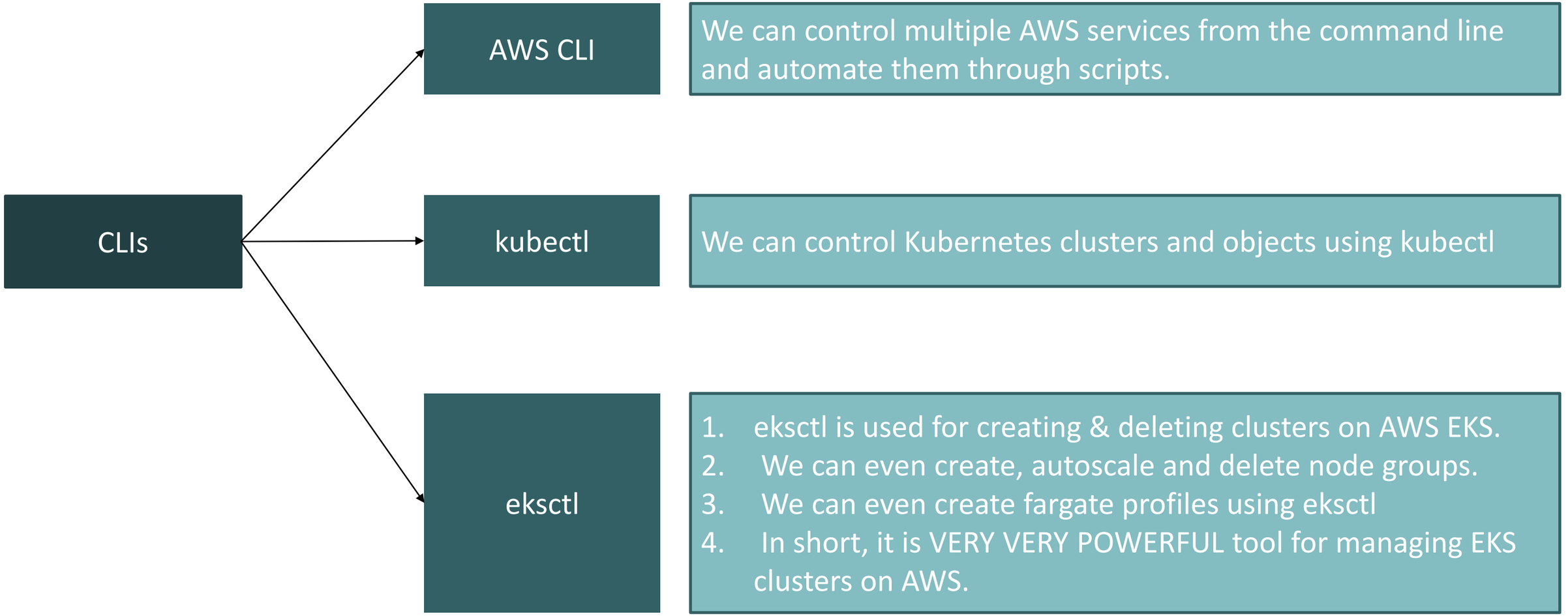
# CloudWatch Container Insights Map



# AWS EKS

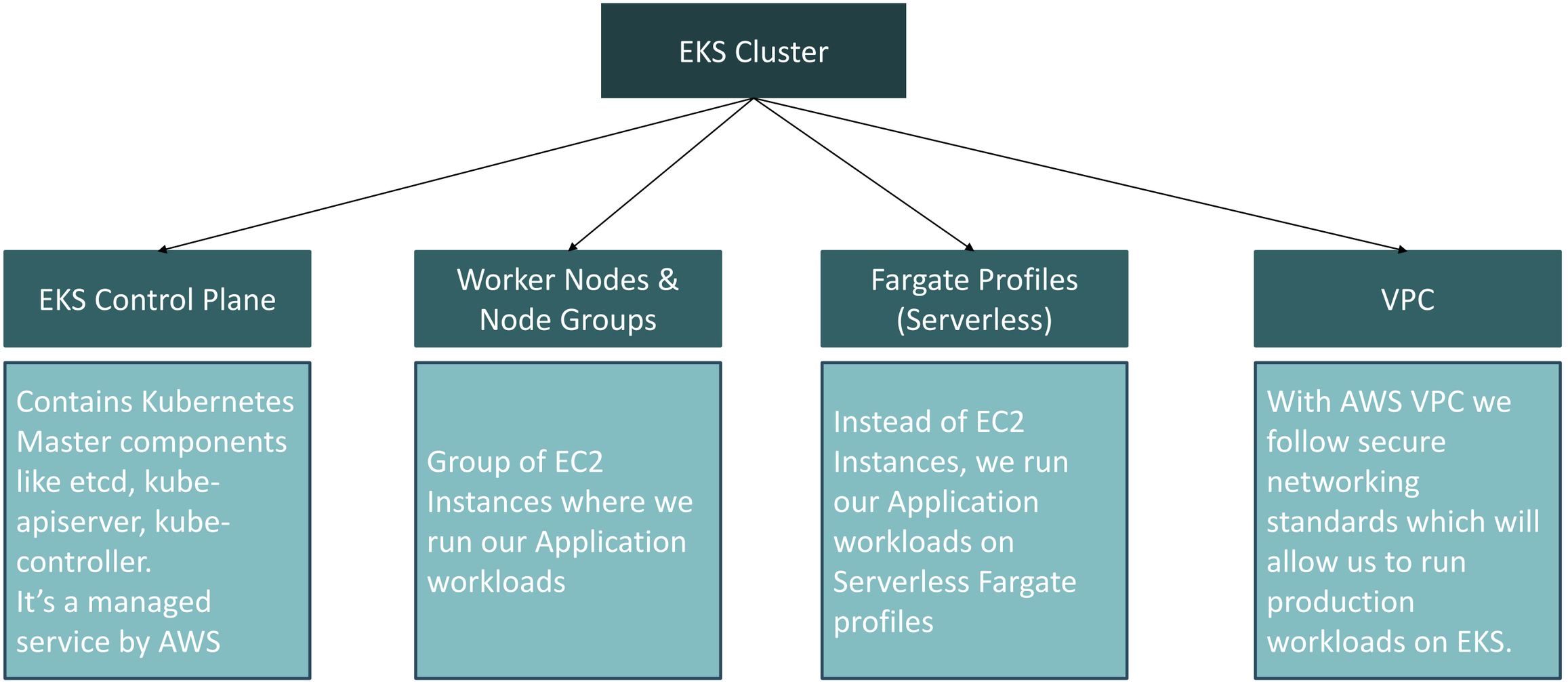
## CLIs

# AWS EKS Cluster - CLIs

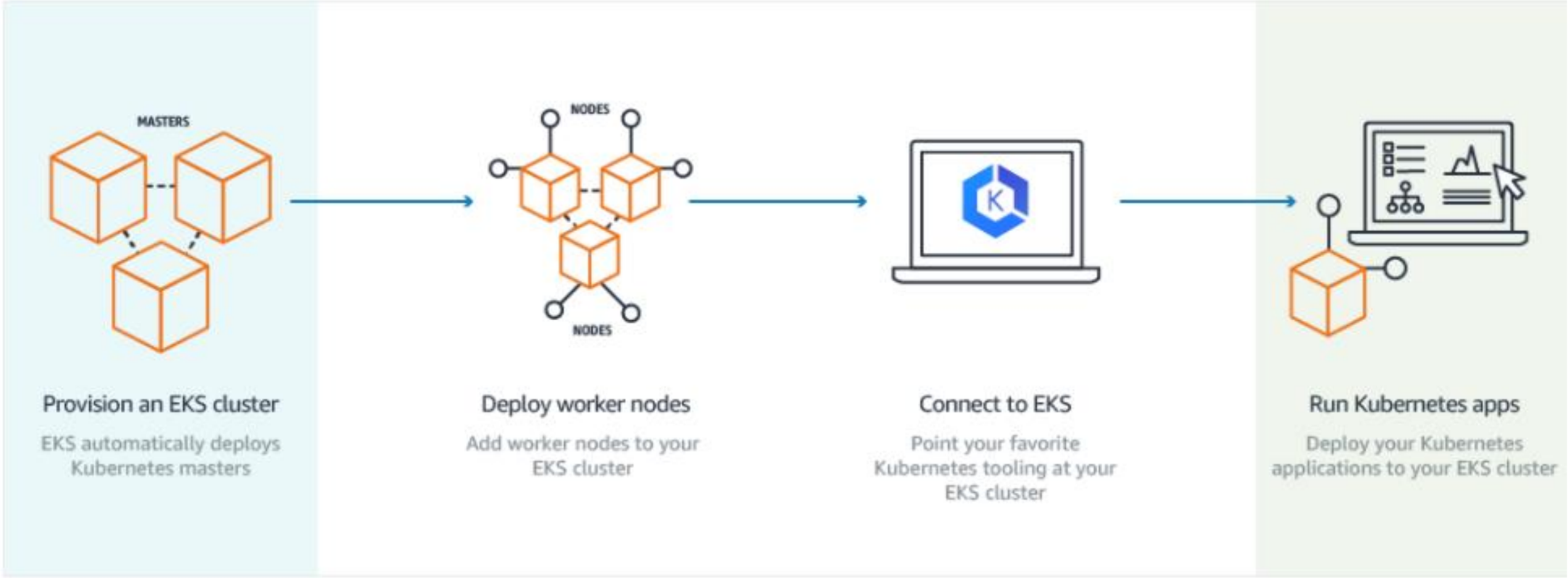


# AWS EKS
# Cluster

# 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

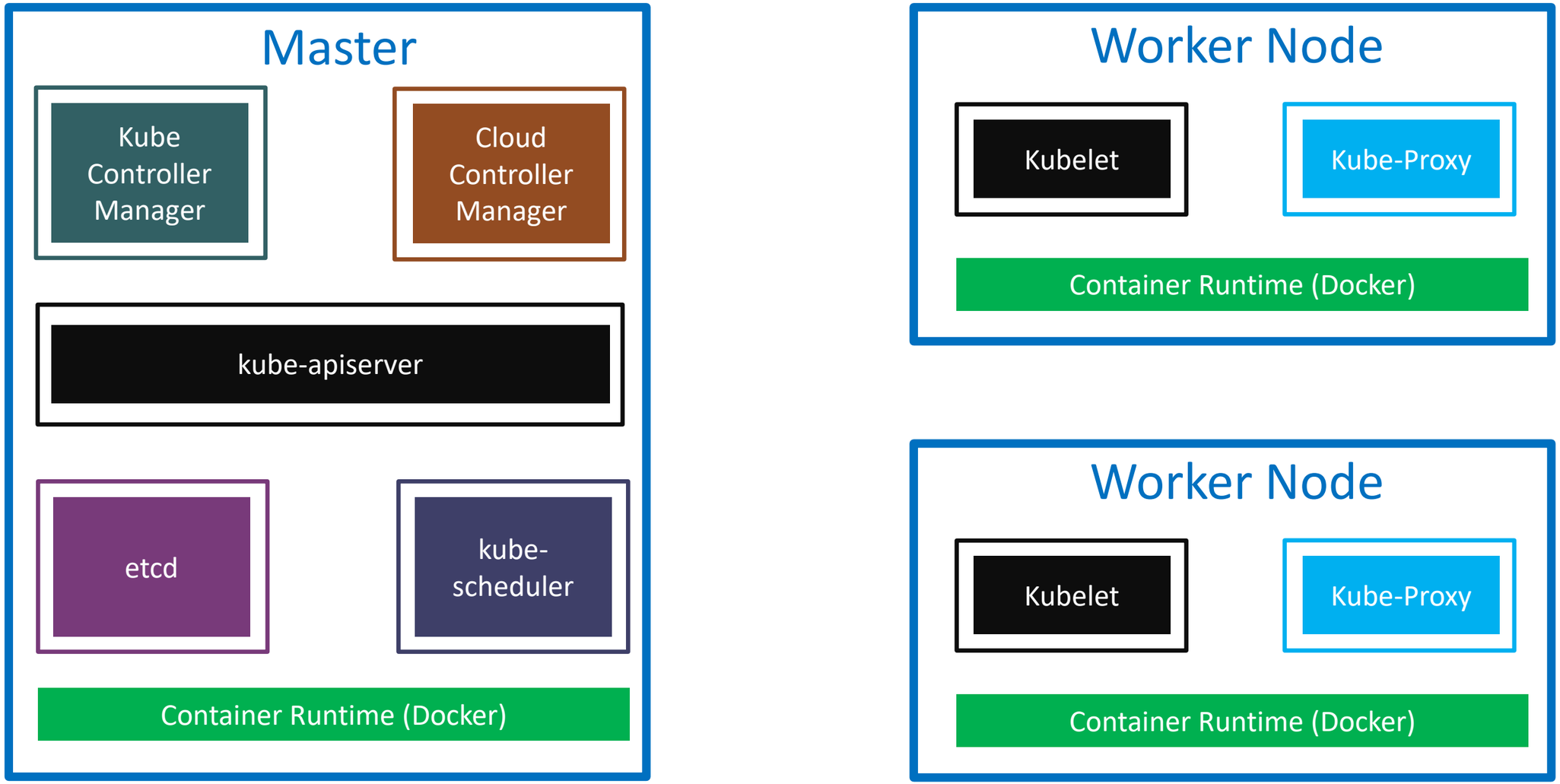
# EKS Cluster – Core Objects Detailed

| | |
|---|---|
| Fargate Profiles | 1. AWS Fargate is a technology that provides on-demand, right-sized compute capacity for containers<br>2. With Fargate, we no longer have to provision, configure, or scale groups of virtual machines to run containers.<br>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.<br>4. AWS specially built Fargate controllers that recognizes the pods belonging to fargate and schedules them on Fargate profiles.<br>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.<br>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.<br>3. This secure and highly-available configuration makes EKS reliable and recommended for production workloads. |

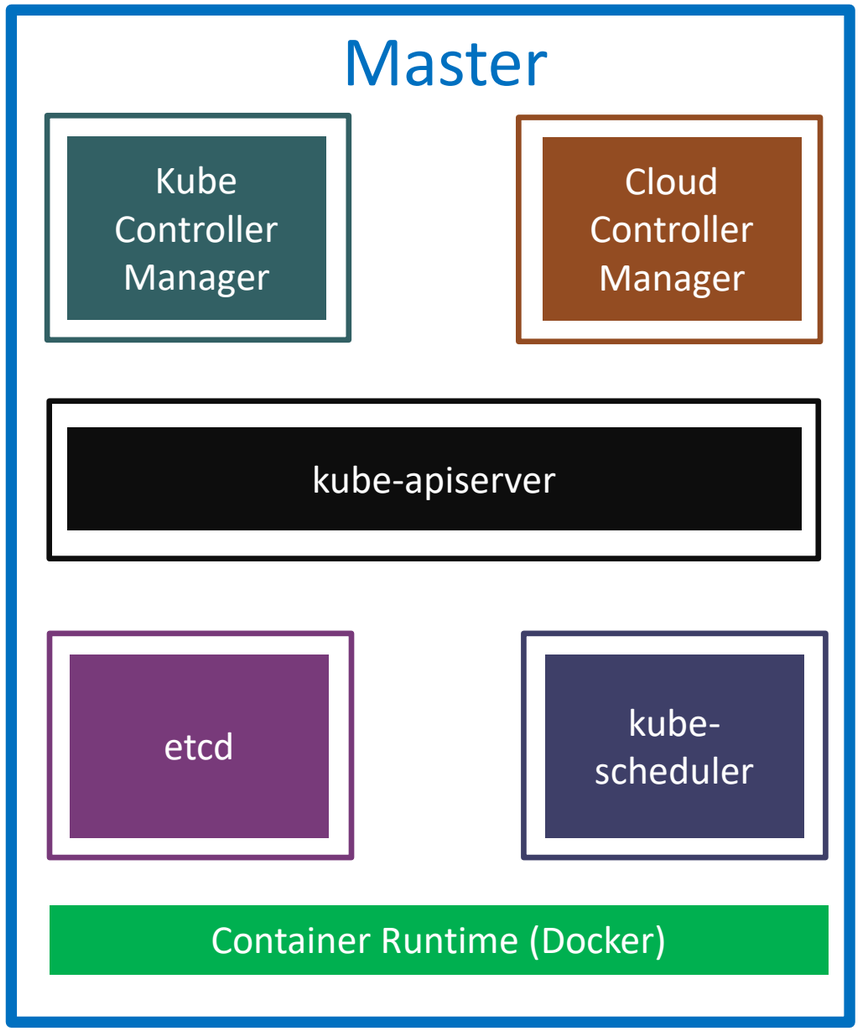
# Kubernetes
# Architecture

# Kubernetes Architecture

# Kubernetes - Architecture

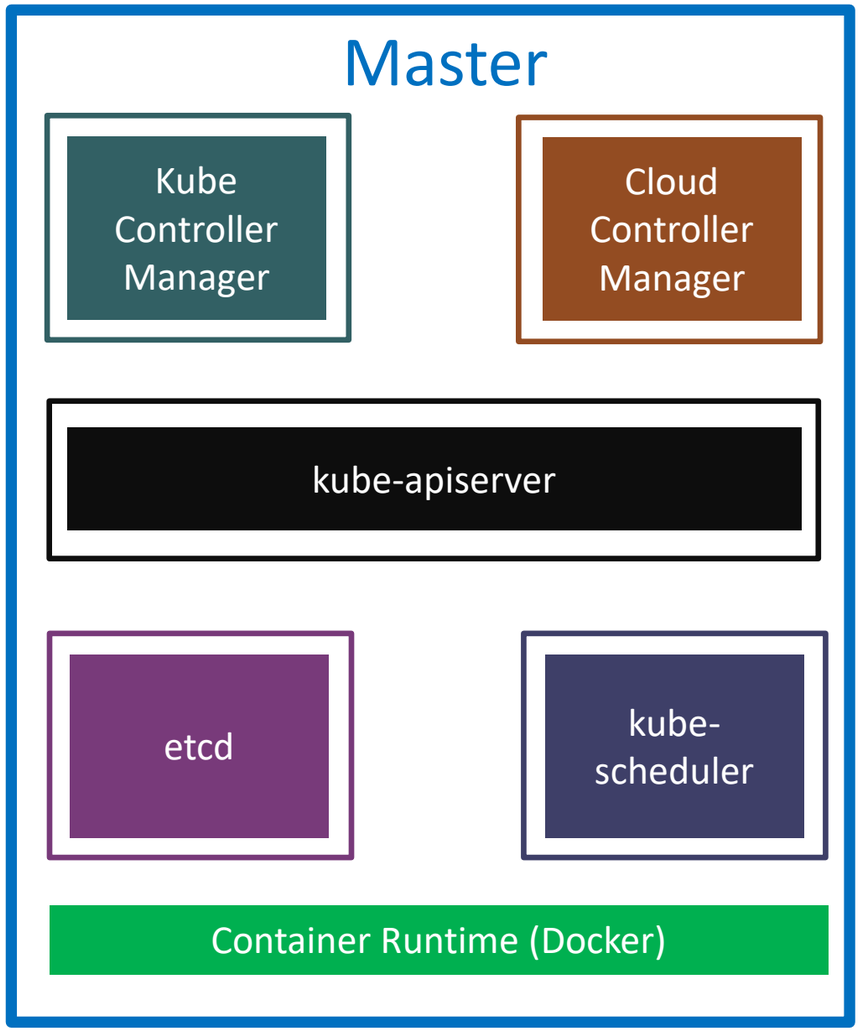


# Kubernetes Architecture - Master



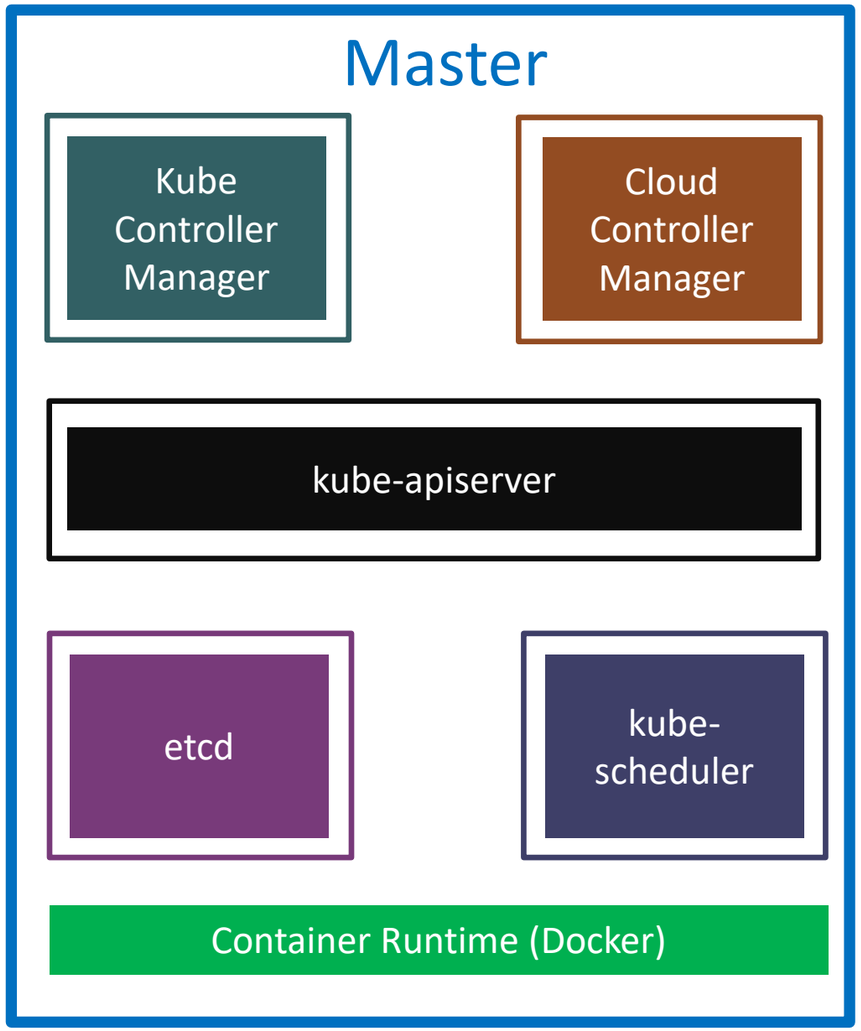

- kube-apiserver
  - It acts as front end for the Kubernetes control plane. It exposes the Kubernetes API
  - Command line tools (like kubectl), Users and even Master components (scheduler, controller manager, etcd) and Worker node components like (Kubelet) everything talk with API Server.
- etcd
  -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



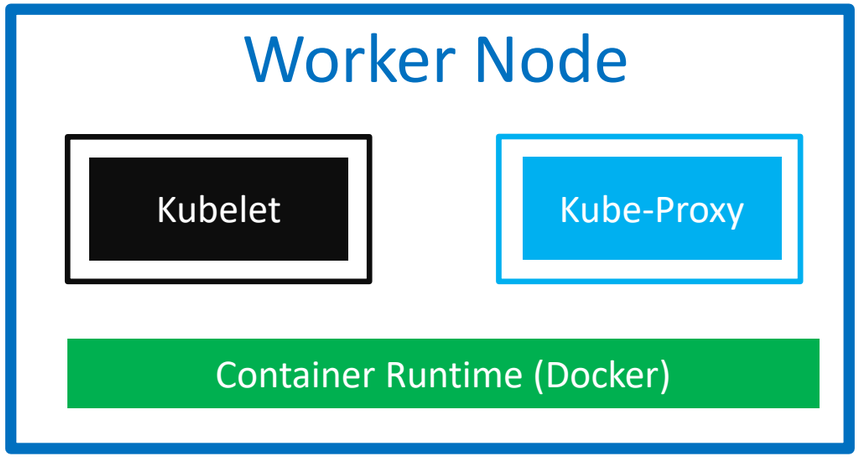

- kube-controller-manager
  - Controllers are responsible for noticing and responding when nodes, containers or endpoints go down. They make decisions to bring up new containers in such cases.
  - Node Controller: Responsible for noticing and responding when nodes go down.
  - Replication Controller: 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

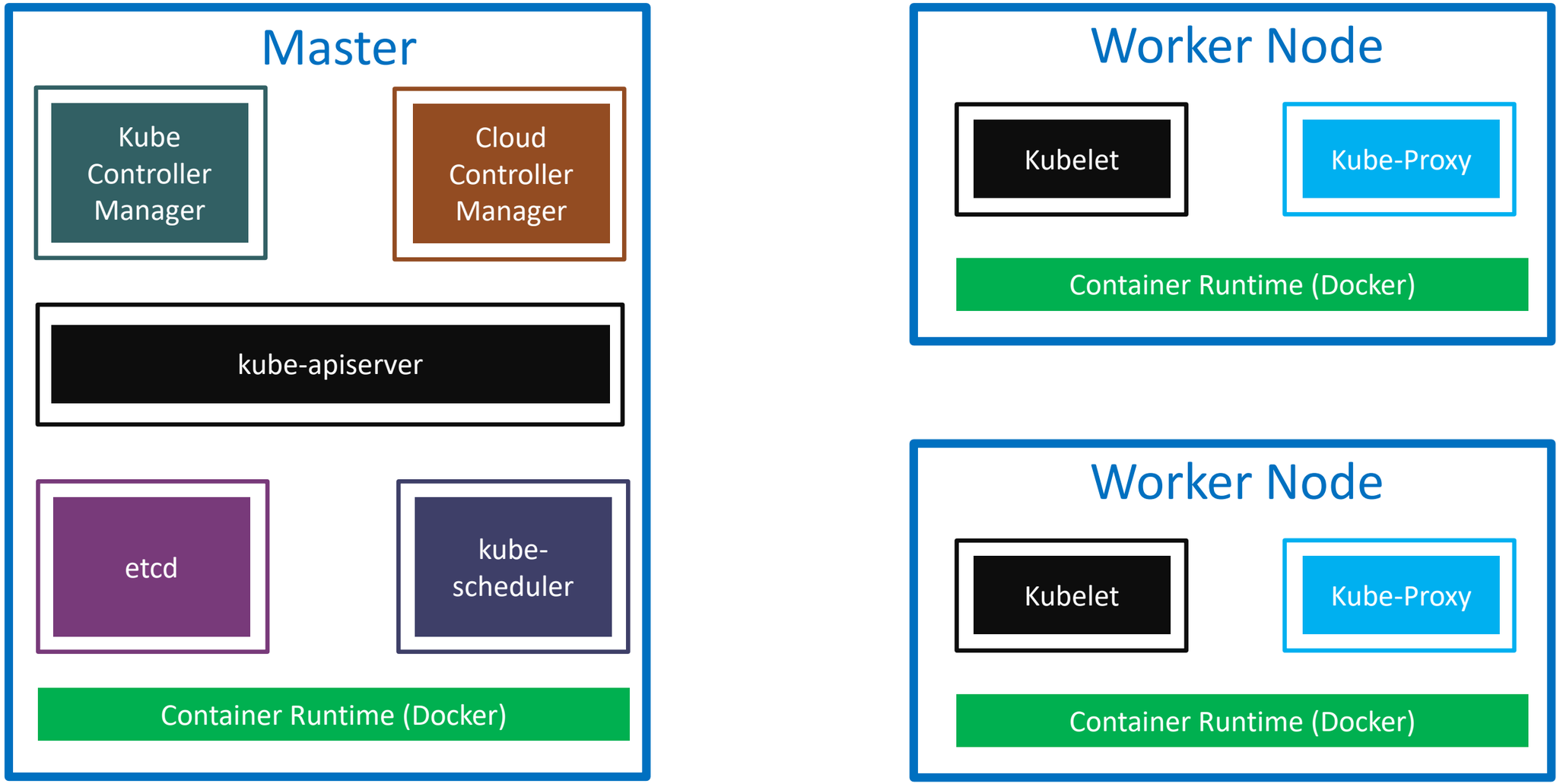
# Kubernetes Architecture – Worker Nodes




- 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.

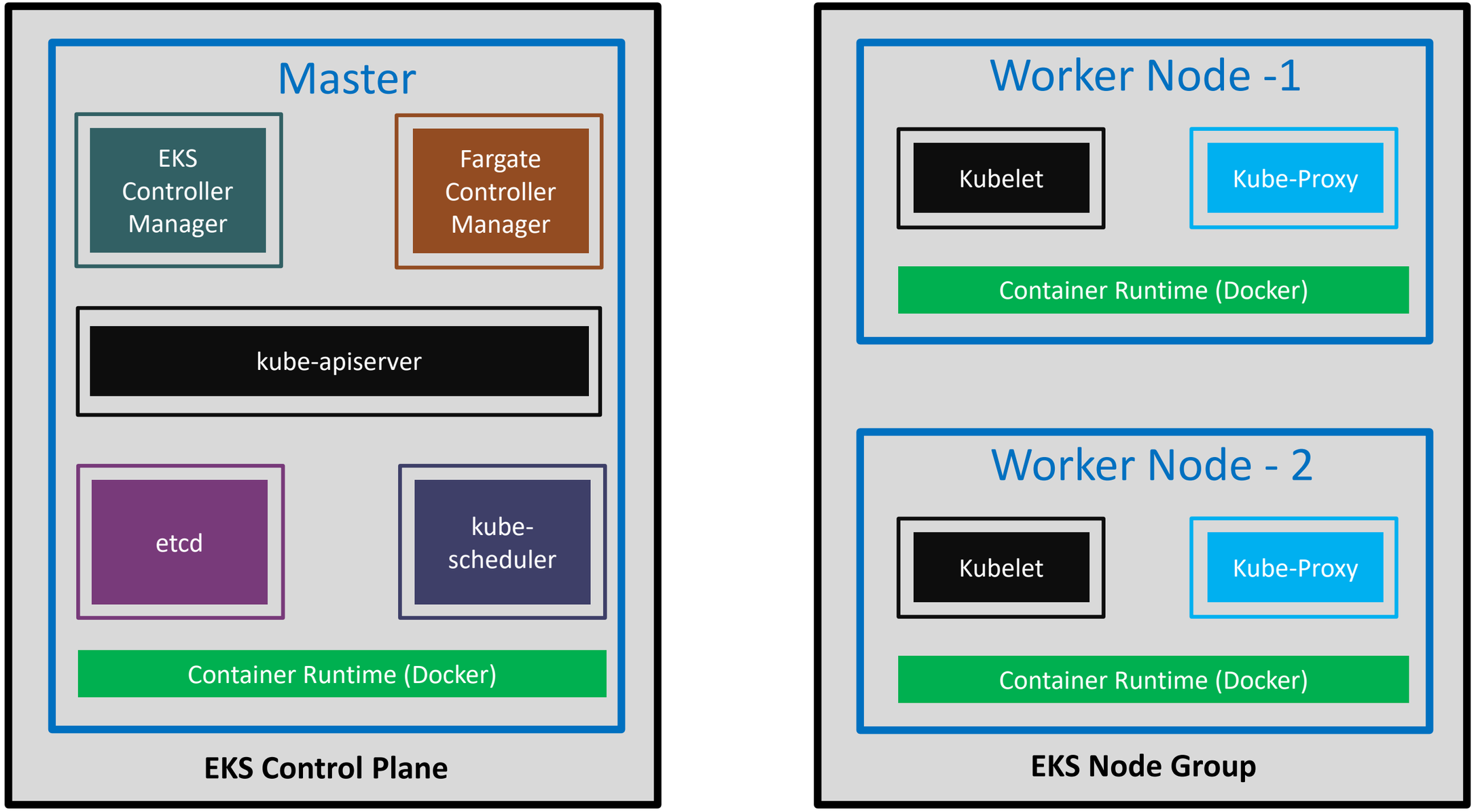
- Kubelet
  - Kubelet is the agent that runs on every node in the cluster
  - This agent is responsible for making sure that containers are running in a Pod on a node.
- Kube-Proxy
  - It is a network proxy that runs on each node in your cluster.
  - It maintains network rules on nodes
  - In short,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

# Kubernetes - Architecture



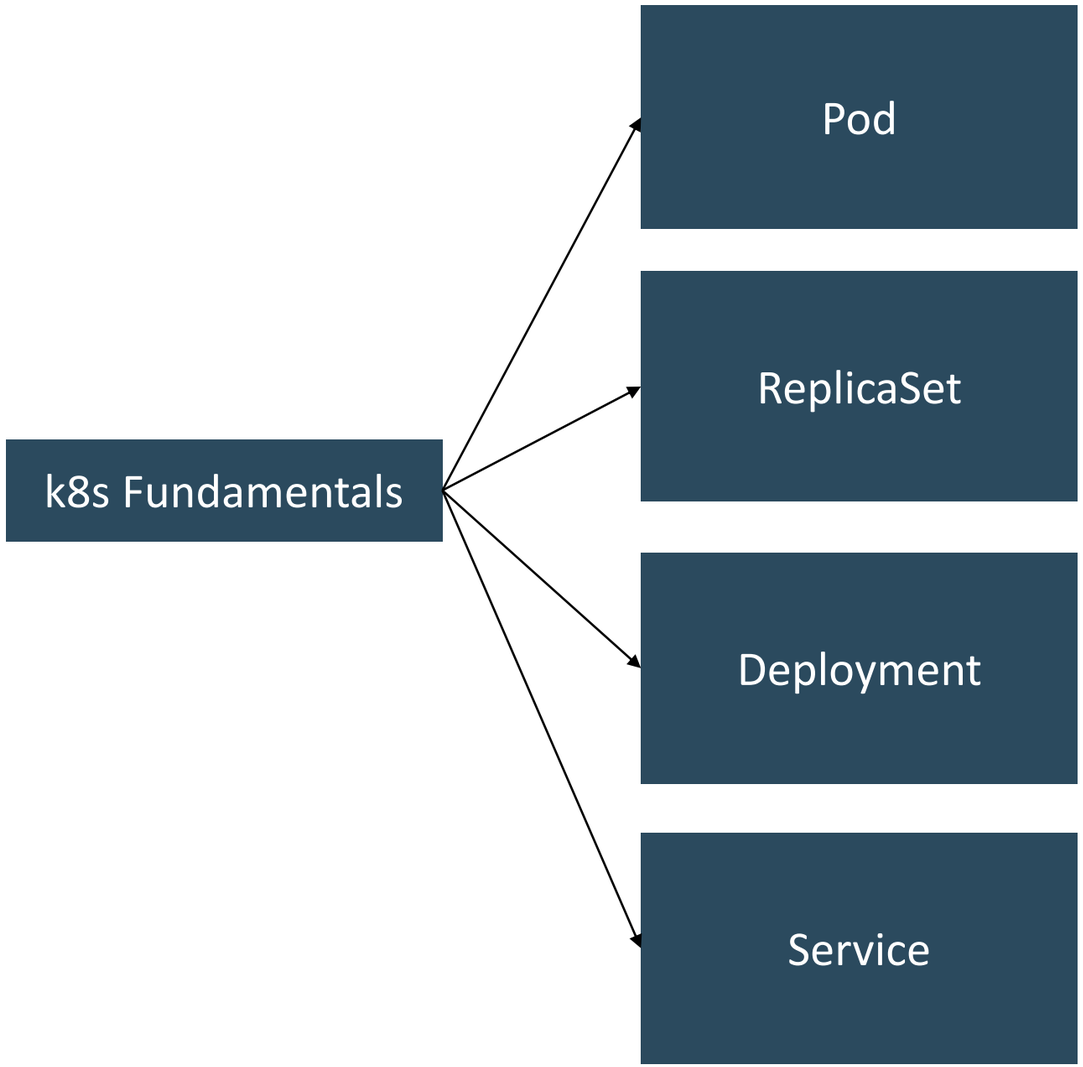
# AWS EKS
# Cluster

# EKS Kubernetes - Architecture



# Kubernetes
# Fundamentals
## Pod, ReplicaSet, Deployment & Service

# Kubernetes - Fundamentals



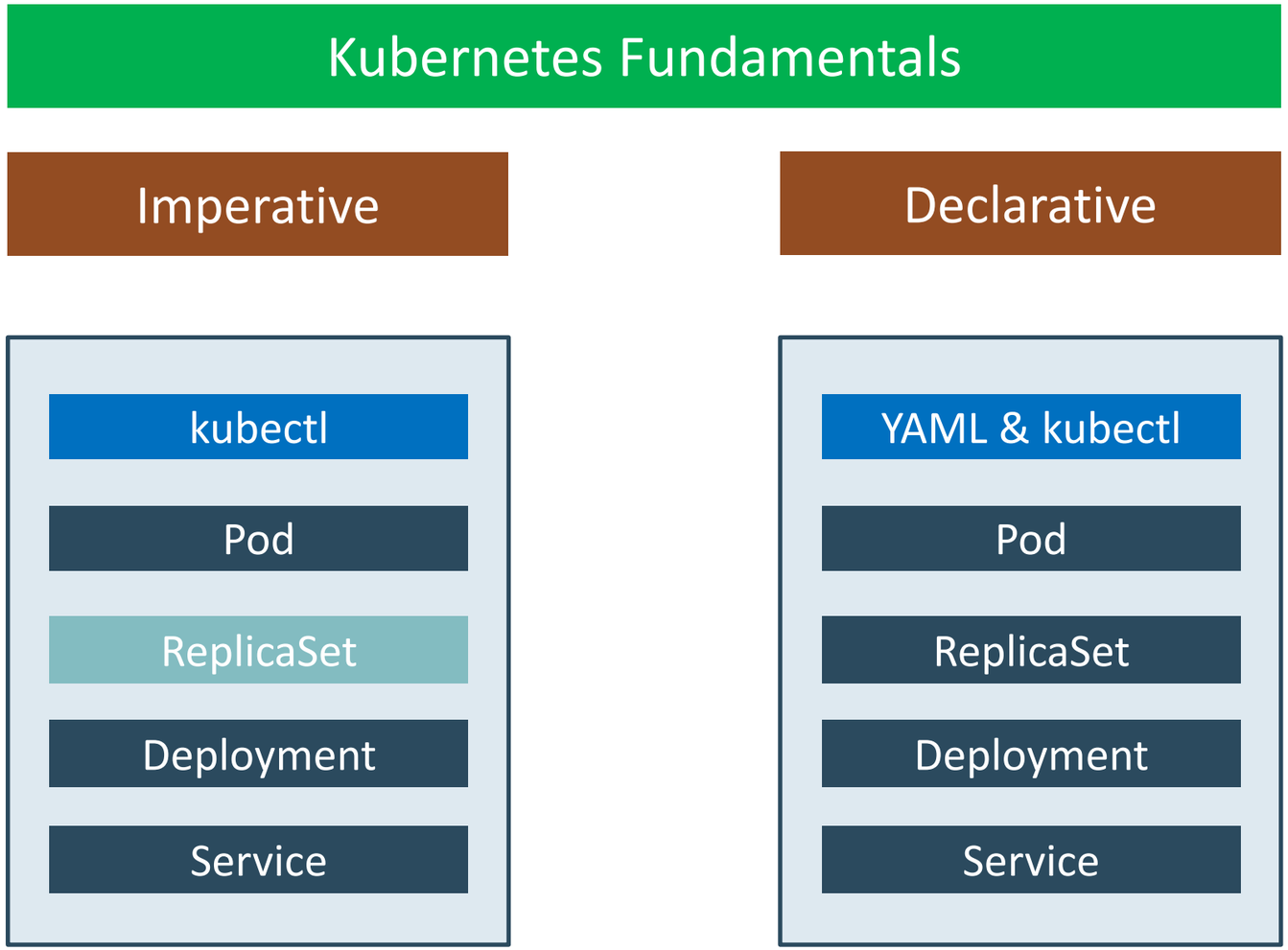

A POD is a single instance of an Application.
A POD is the smallest object, that you can create in Kubernetes.

A ReplicaSet will maintain a stable set of replica Pods running at any given time.
In short, it is often used to guarantee the availability of a specified number of identical Pods

A Deployment runs multiple replicas of your application and automatically replaces any instances that fail or become unresponsive. Rollout & rollback changes to applications. Deployments are well-suited for stateless applications.

A service is an abstraction for pods, providing a stable, so called virtual IP (VIP) address.
In simple terms, service sits Infront of a POD and acts as a load balancer.

# Kubernetes - Imperative & Declarative



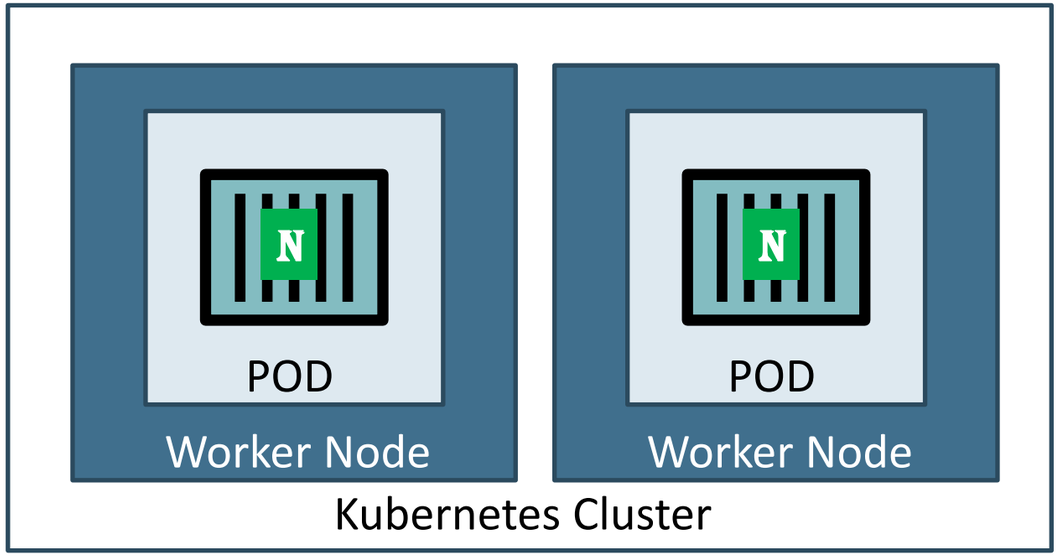
# Kubernetes

# POD

# Kubernetes - POD

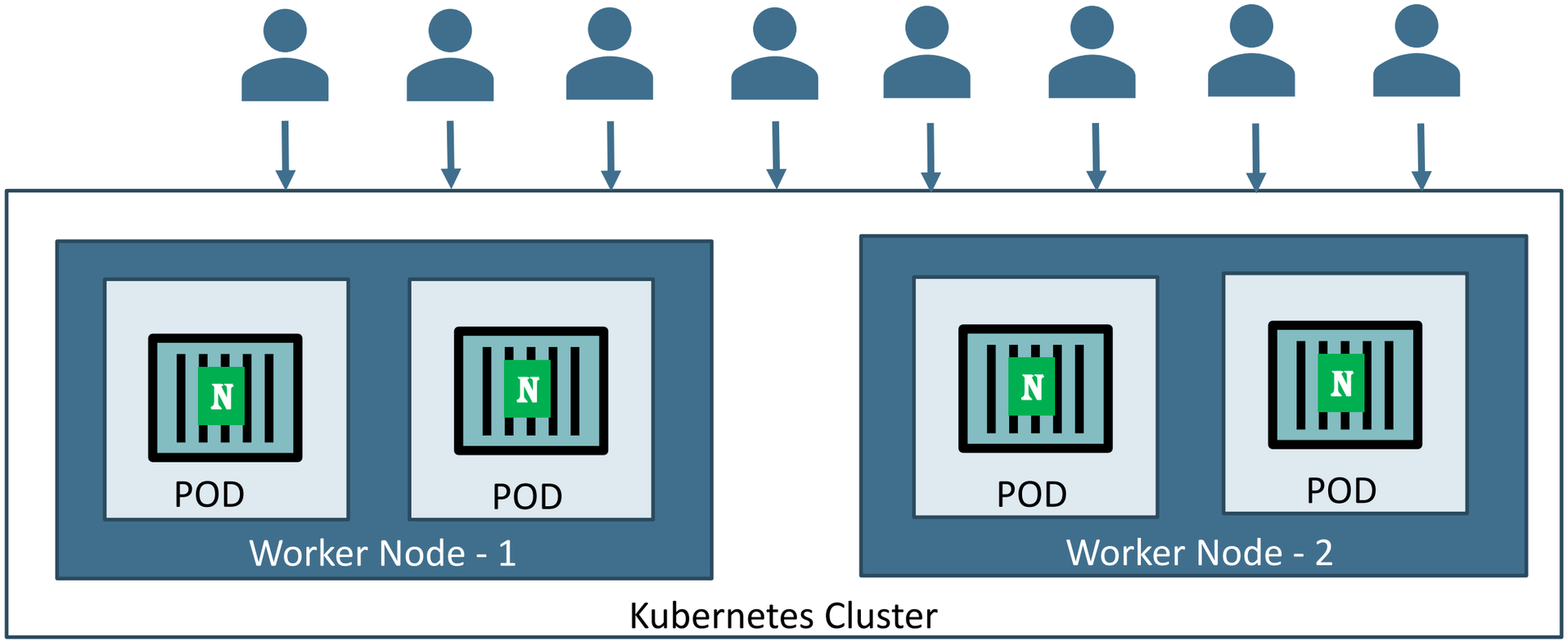
- With Kubernetes our core goal will be to deploy our applications in the form of containers on worker nodes in a k8s cluster.
- Kubernetes does not deploy containers directly on the worker nodes.
- Container is encapsulated in to a Kubernetes Object named POD.
- A POD is a single instance of an application.
- A POD is the smallest object that we can create in Kubernetes.


Nginx Container Image



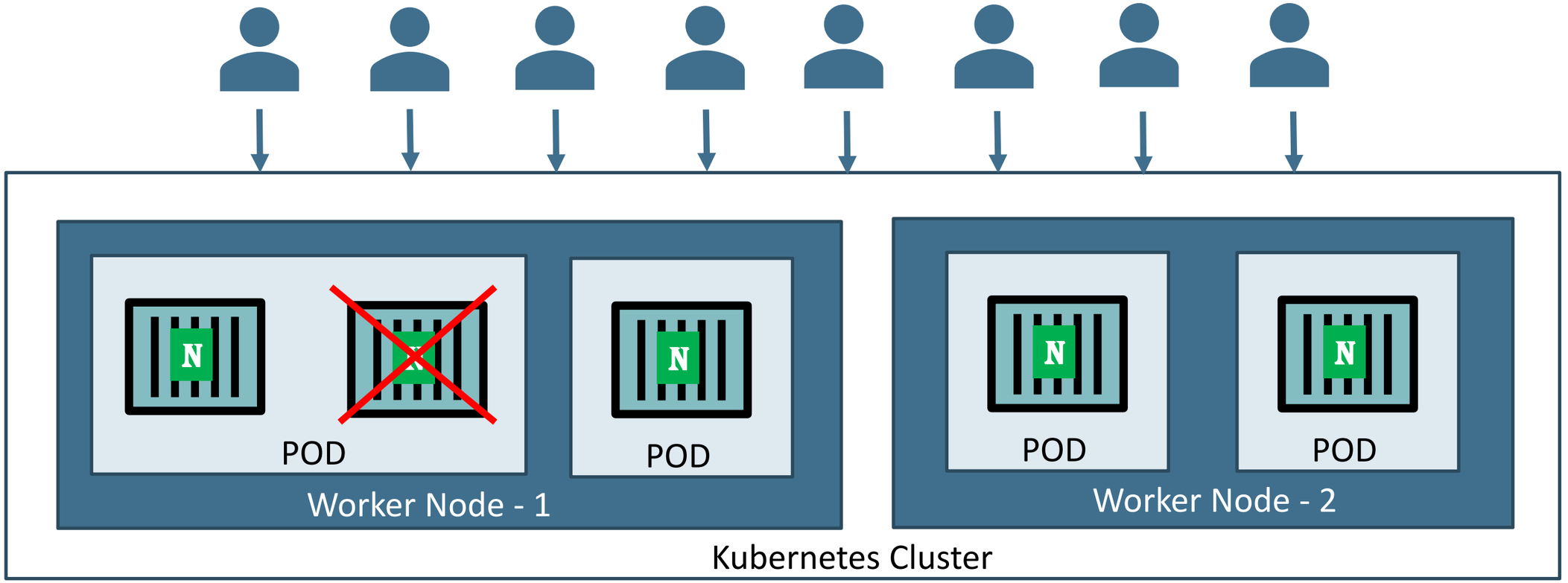
# Kubernetes - POD

- PODs generally have one to one relationship with containers.
- To scale up we create new POD and to scale down we delete the POD.



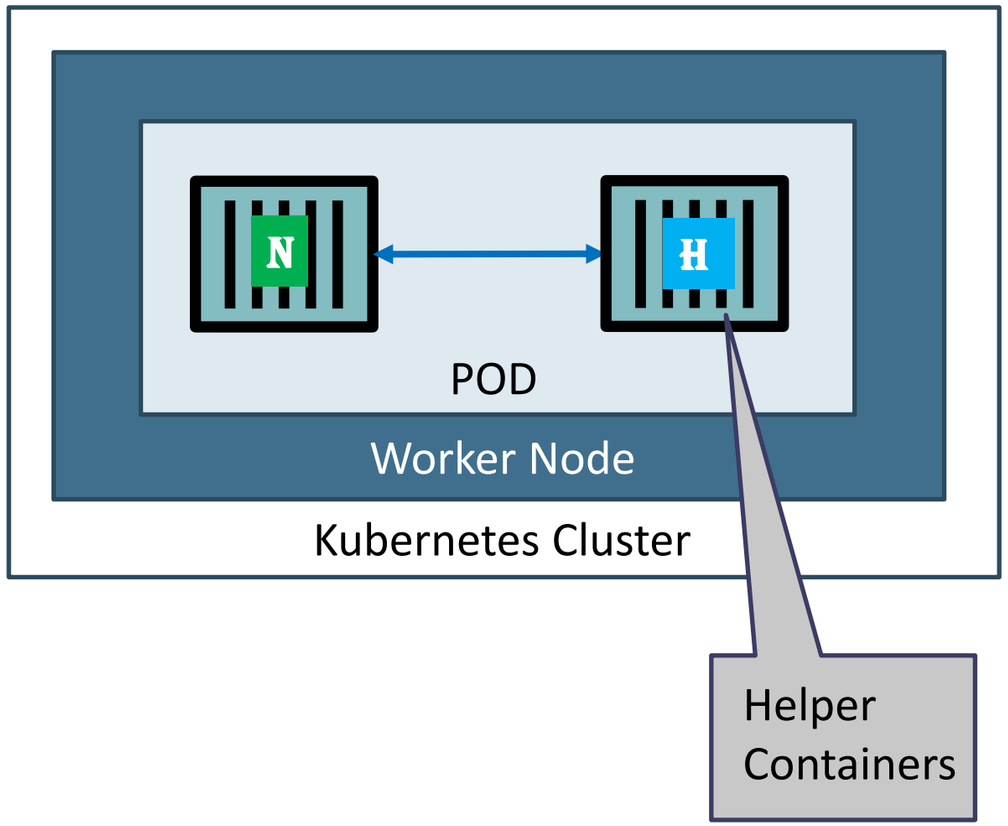
# Kubernetes – PODs

- We cannot have multiple containers of same kind in a single POD.
- Example: Two NGINX containers in single POD serving same purpose is not recommended.



# Kubernetes – Multi-Container Pods

- We can have multiple containers in a single POD, provided they are not of same kind.
- Helper Containers (Side-car)
  - Data Pullers: Pull data required by Main Container
  - Data pushers: Push data by collecting from main container (logs)
  - Proxies: Writes static data to html files using Helper container and Reads using Main Container.
- Communication
  - The two containers can easily communicate with each other easily as they share same network space.
  - They can also easily share same storage space.
- Multi-Container Pods is a rare use-case and we will try to focus on core fundamentals.

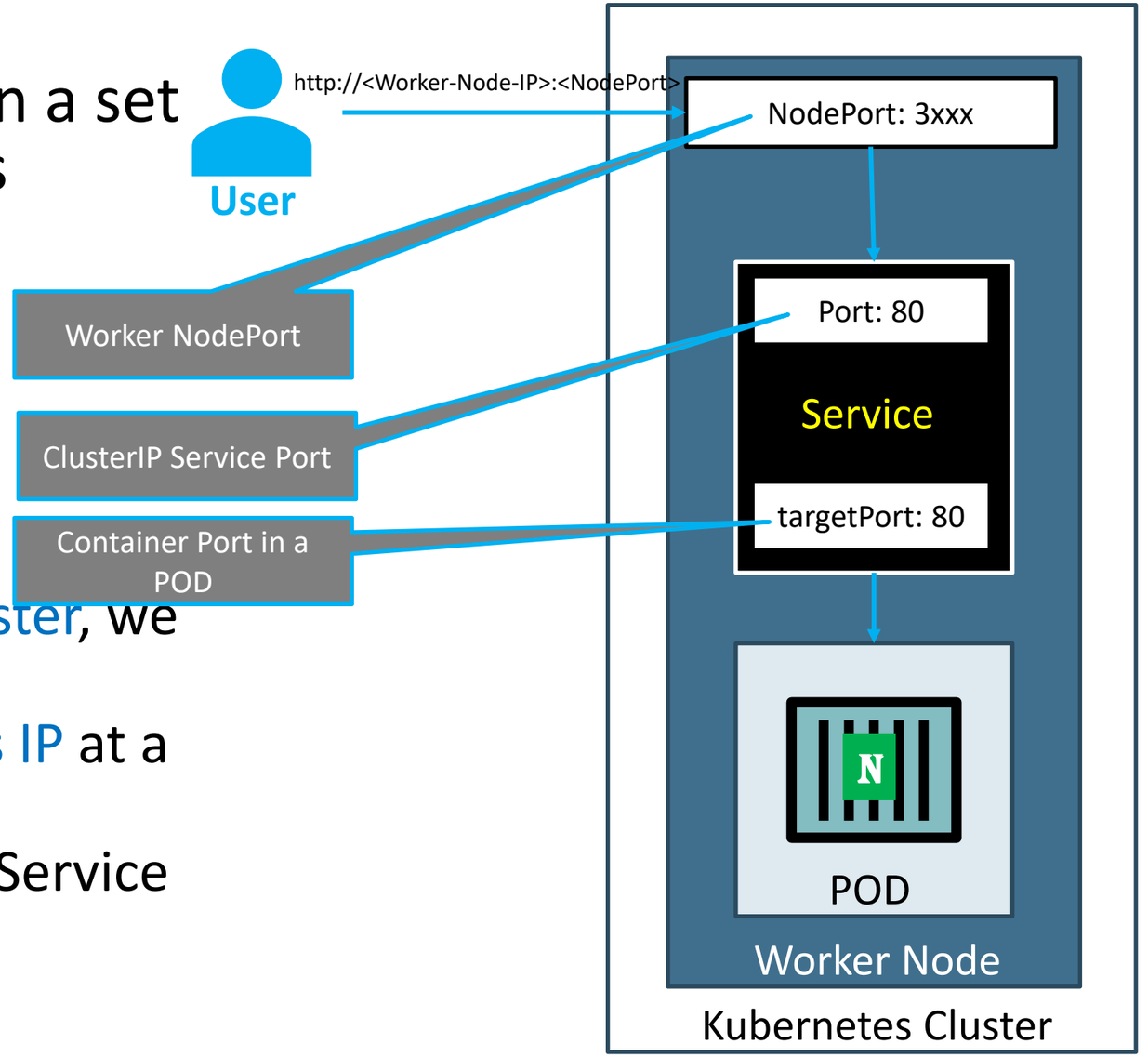


# Kubernetes
# PODs
# Demo

# Kubernetes
# Services - NodePort

# Kubernetes – Service - NodePort

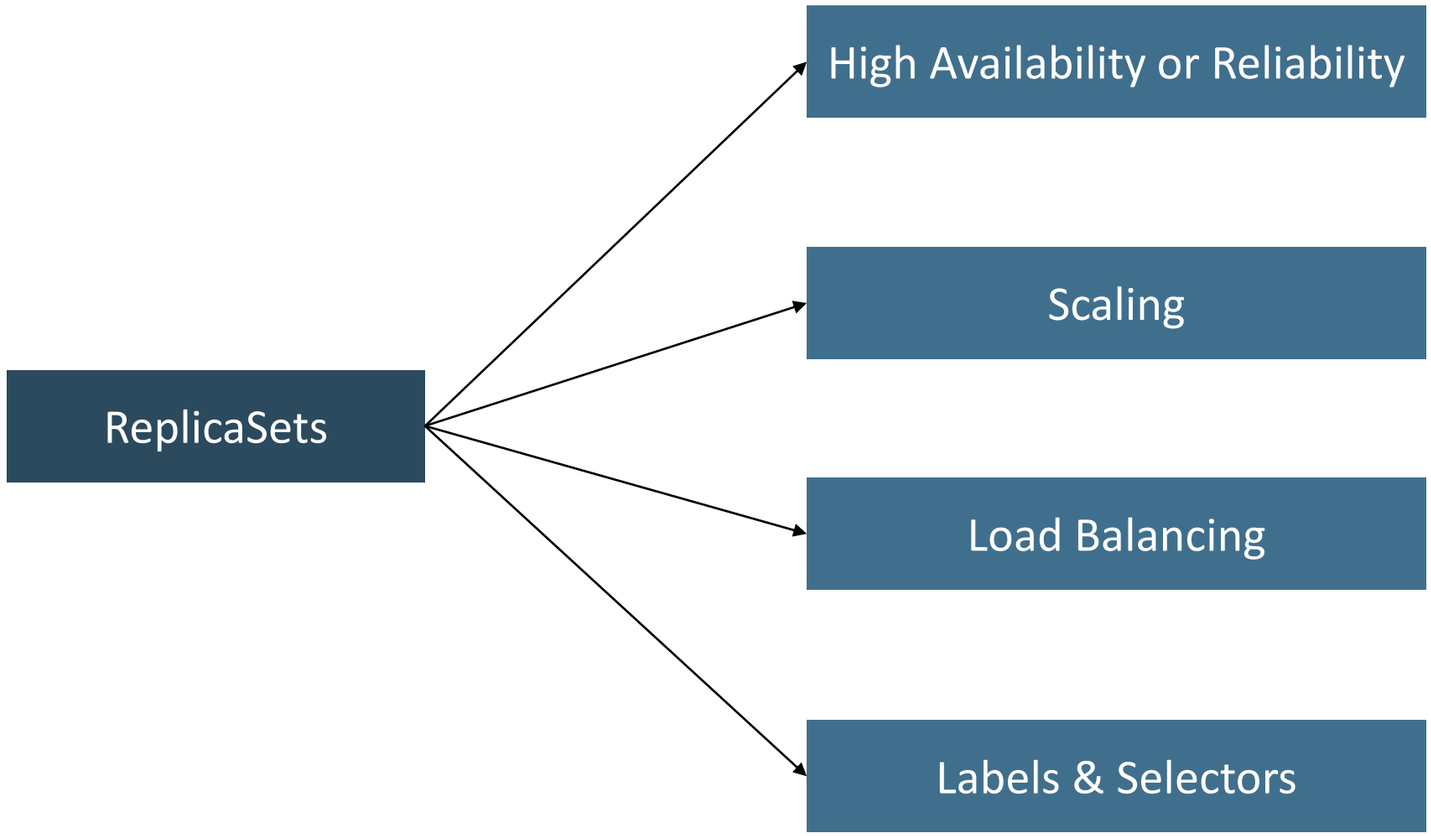
- We can expose an application running on a set of PODs using different types of Services available in k8s.
  - ClusterIP
  - NodePort
  - LoadBalancer
- NodePort Service
  - To access our application outside of k8s cluster, we can use NodePort service.
  - Exposes the Service on each Worker Node's IP at a static port (nothing but NodePort).
  - A ClusterIP Service, to which the NodePort Service routes, is automatically created.
  - Port Range 30000-32767



# Kubernetes
# POD & NodePort Service
# Demo

# Kubernetes
# ReplicaSets

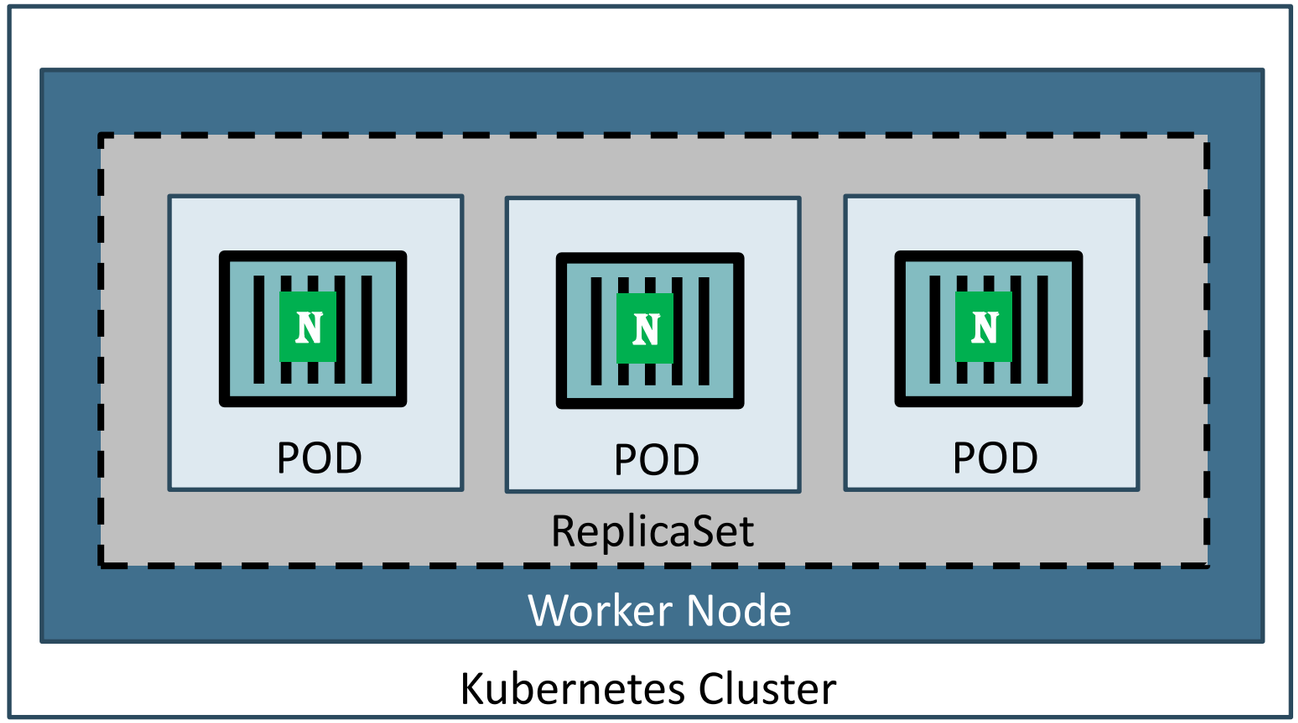
# Kubernetes - ReplicaSets



# Kubernetes – ReplicaSet

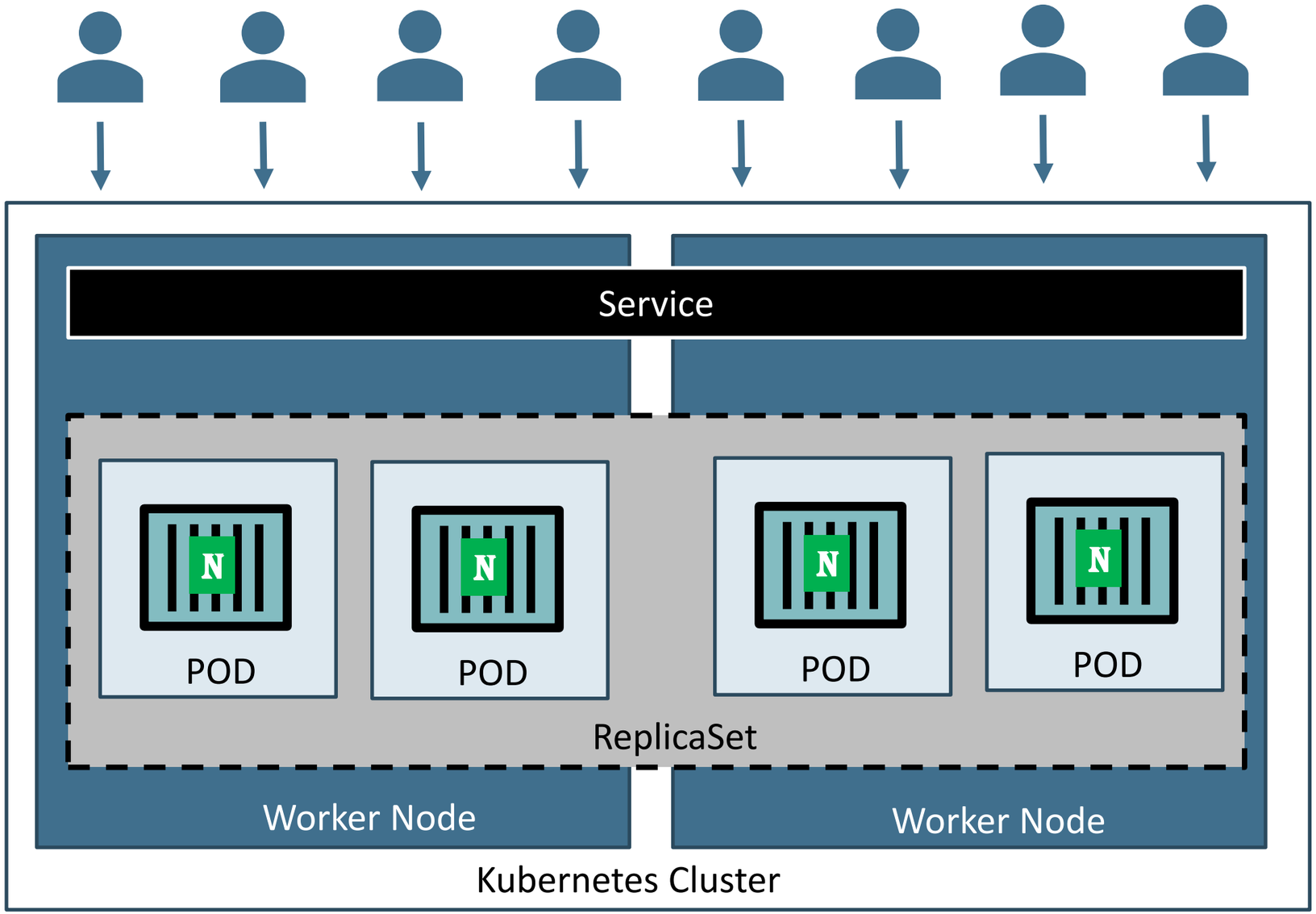
- A ReplicaSet's purpose is to maintain a stable set of replica Pods running at any given time.
- If our application crashes (any pod dies), replicaset will recreate the pod immediately to ensure the configured number of pods running at any given time.

Reliability
Or
High Availability



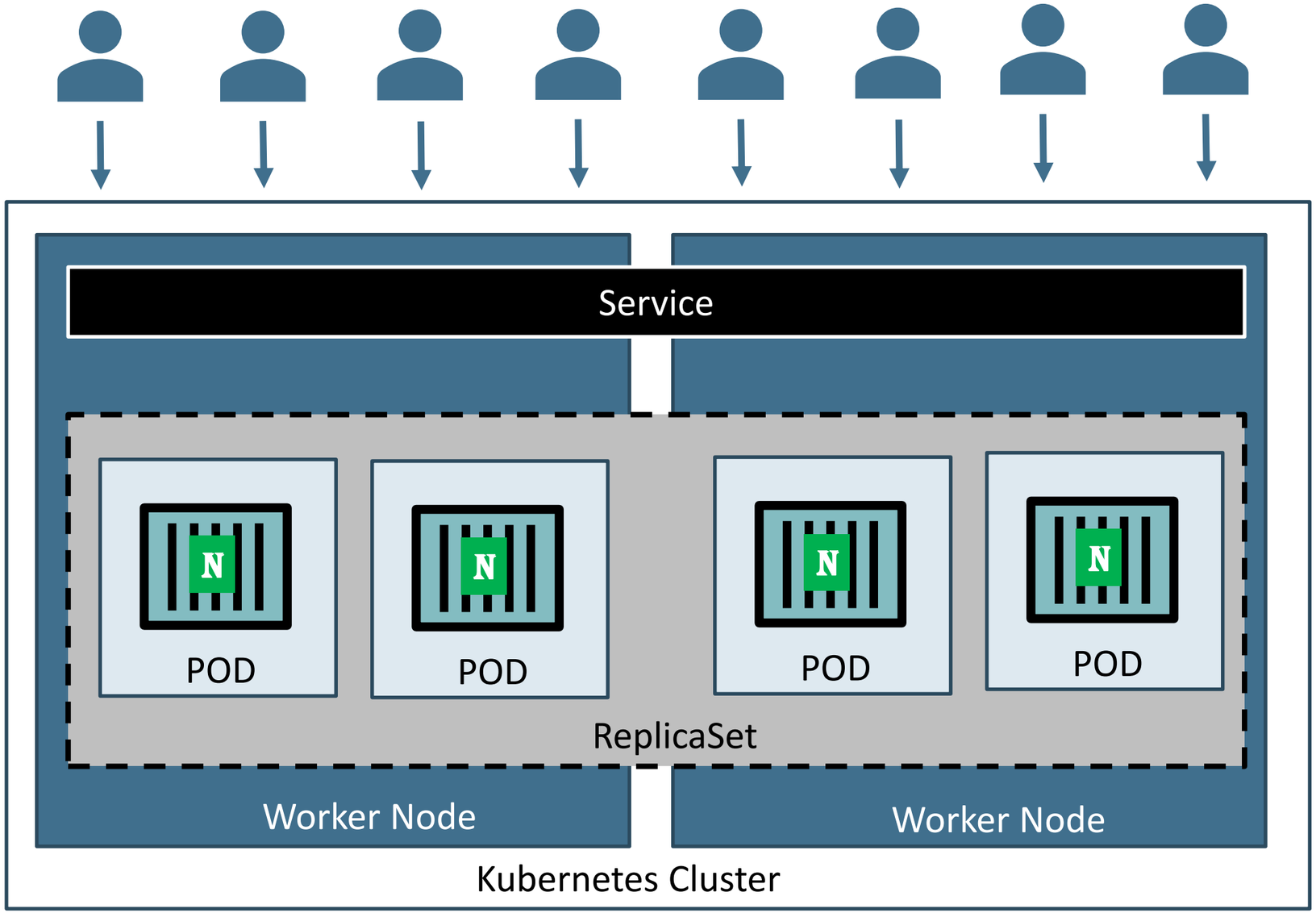
# Kubernetes – ReplicaSet

- Load Balancing
- To avoid overloading of traffic to single pod we can use load balancing.
- Kubernetes provides pod load balancing out of the box using Services for the pods which are part of a ReplicaSet
- Labels & Selectors are the key items which ties all 3 together (Pod, ReplicaSet & Service), we will know in detail when we are writing YAML manifests for these objects



# Kubernetes – ReplicaSet

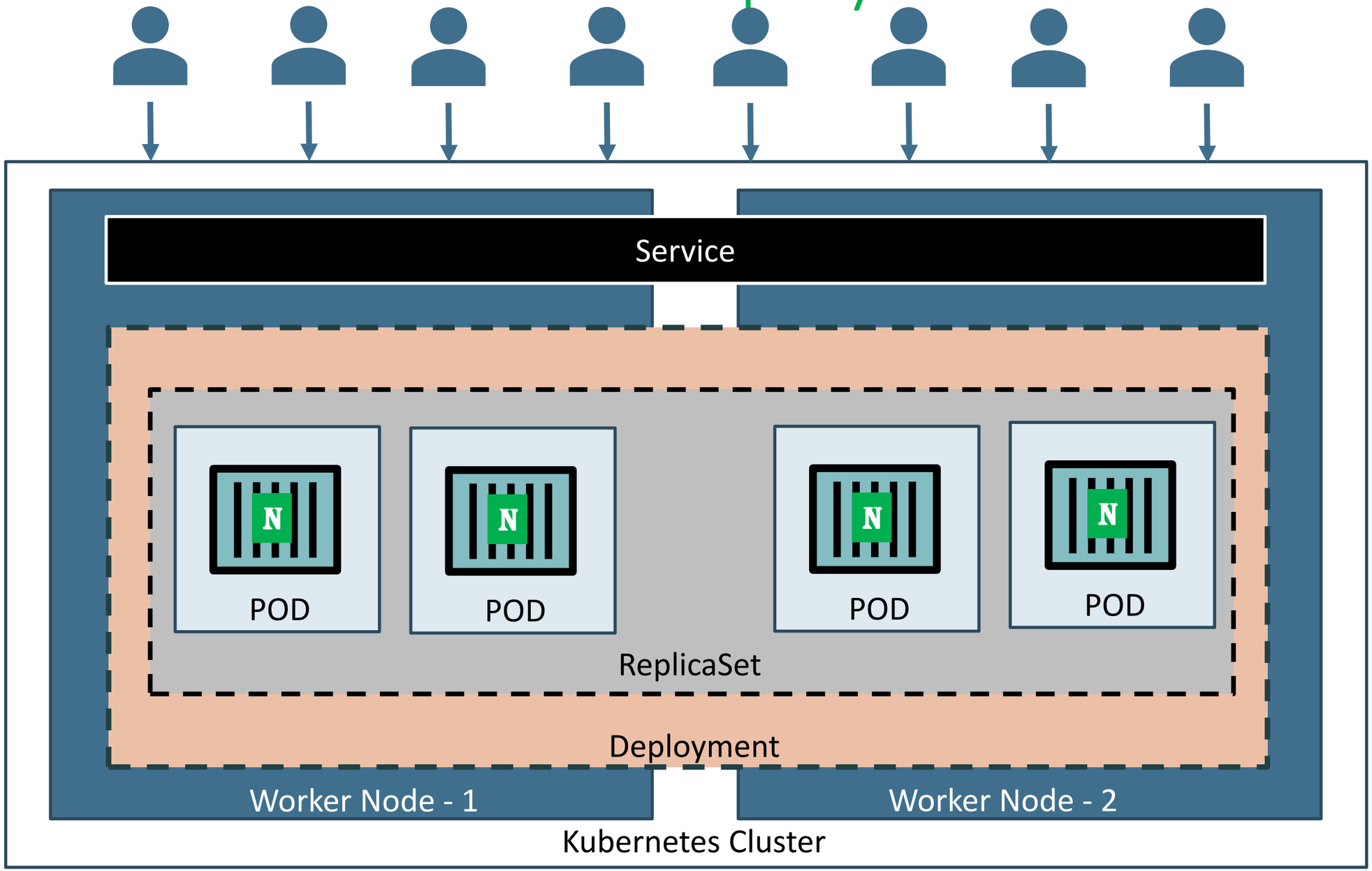
- Scaling
- When load become too much for the number of existing pods, Kubernetes enables us to easily scale up our application, adding additional pods as needed.
- This is going to be seamless and super quick.



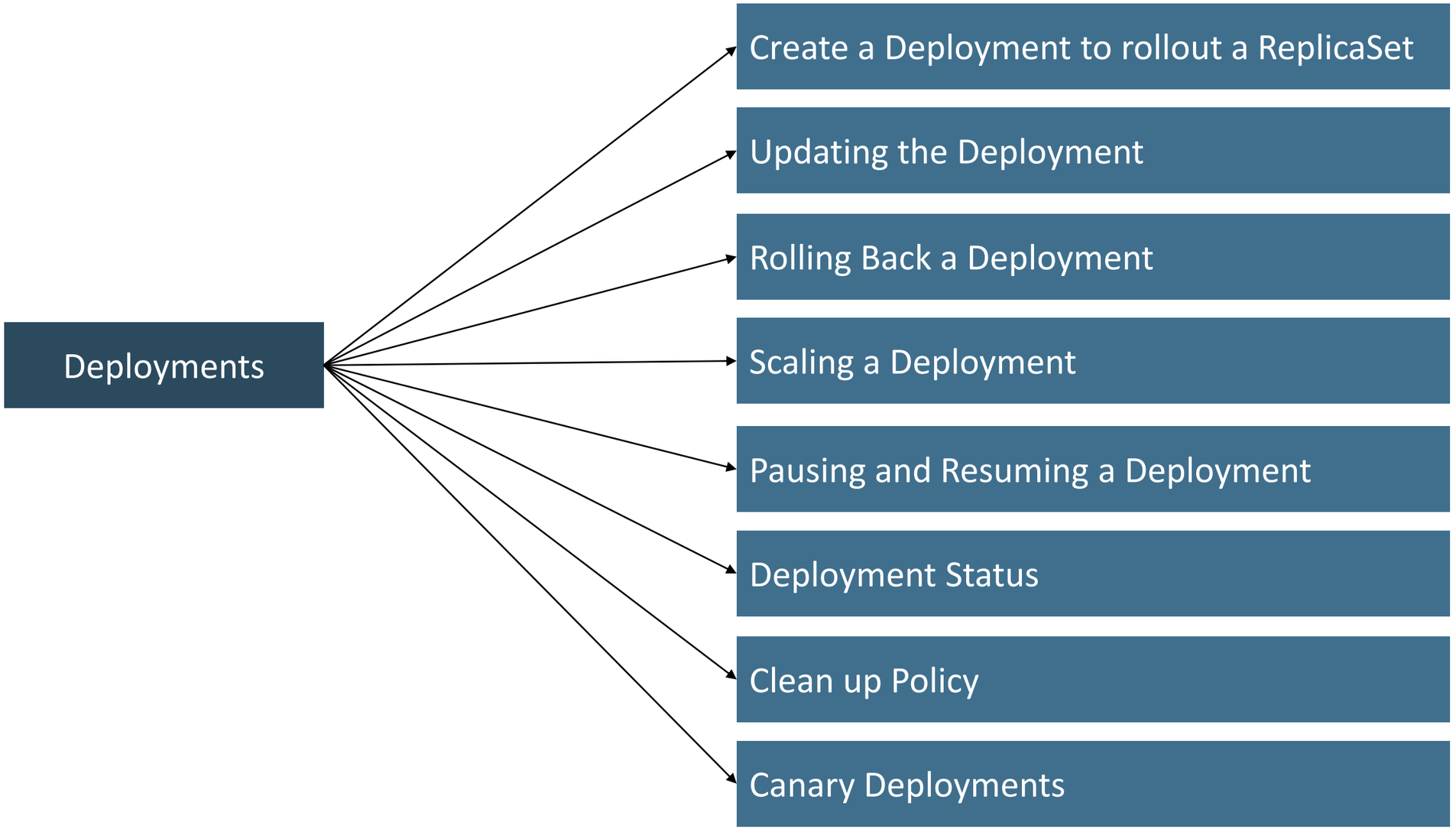
# Kubernetes ReplicaSets Demo

# Kubernetes
# Deployments

# Kubernetes – Deployments



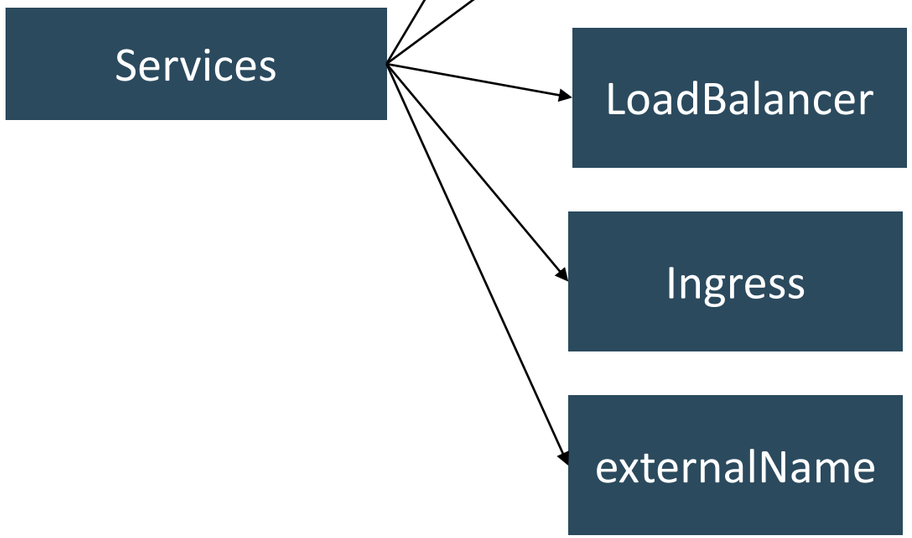
# Kubernetes - Deployment



# Kubernetes Deployments Demo

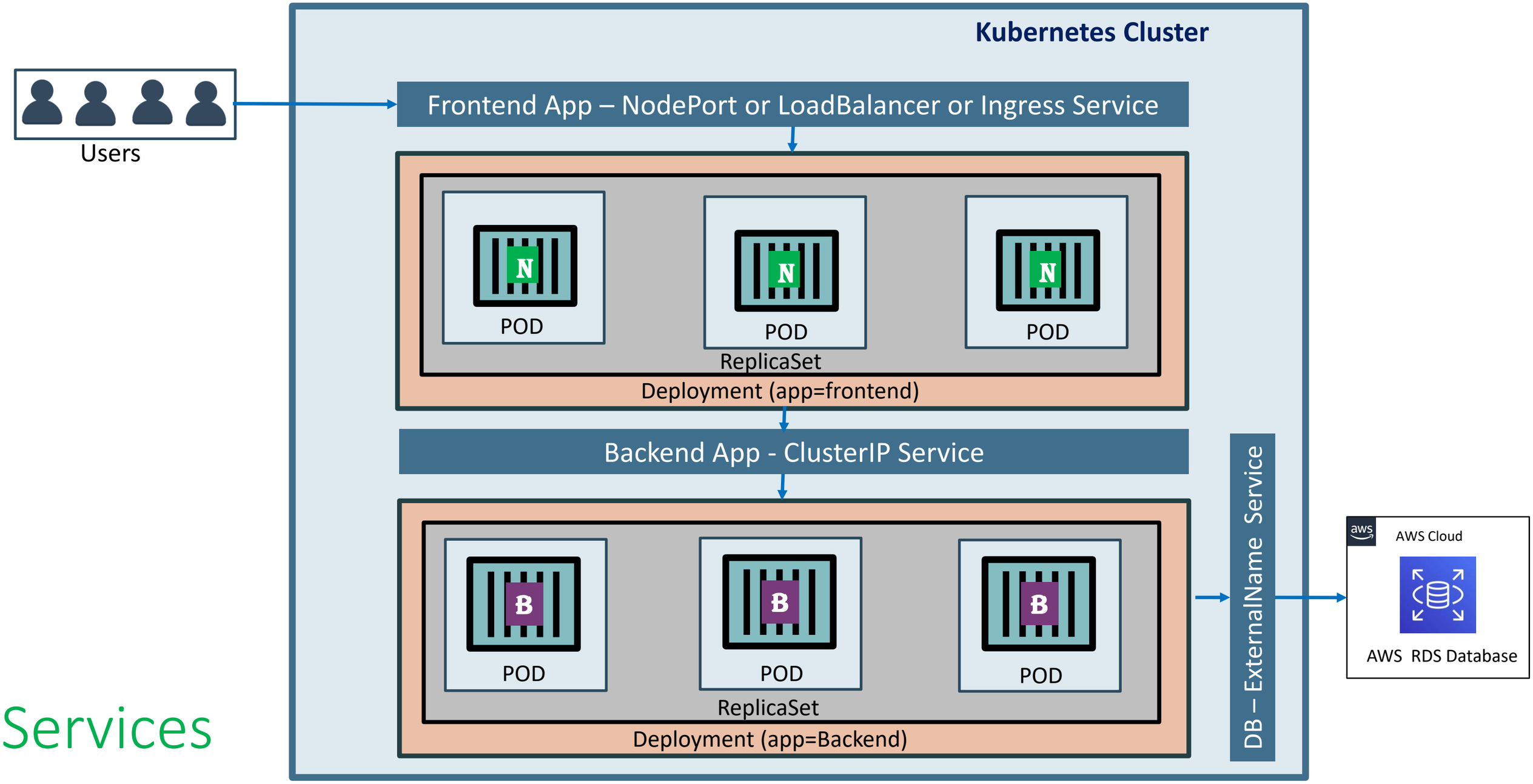
# Kubernetes
# Services

# Kubernetes - Services



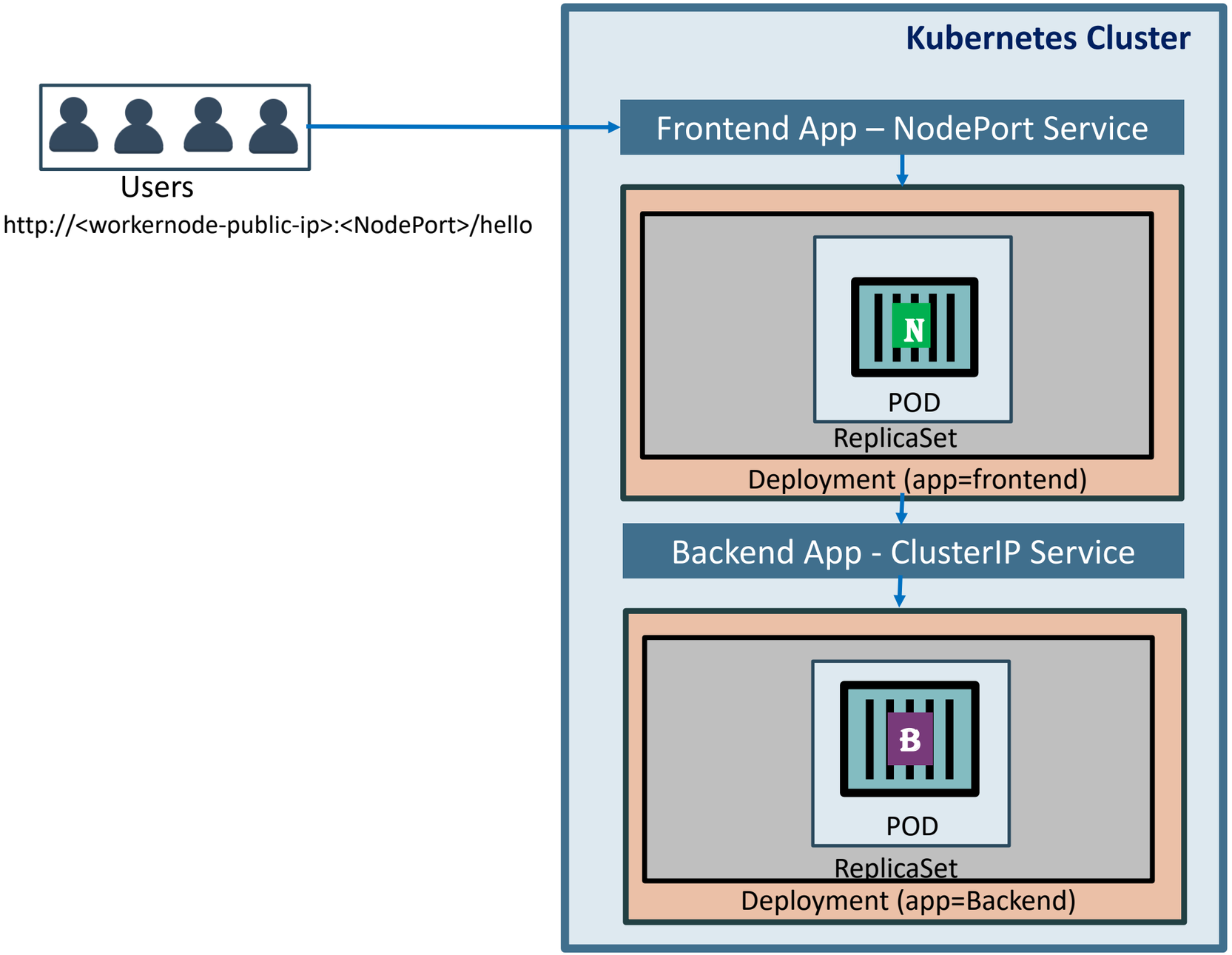

- **ClusterIP** — Used for communication between applications inside k8s cluster (Example: Frontend application accessing backend application)
- **NodePort** — Used for accessing applications outside of of k8s cluster using Worker Node Ports (Example: Accessing Frontend application on browser)
- **LoadBalancer** — Primarily for Cloud Providers to integrate with their Load Balancer services (Example: AWS Elastic Load Balancer)
- **Ingress** — Ingress is an advanced load balancer which provides Context path based routing, SSL, SSL Redirect and many more (Example: AWS ALB)
- **externalName** — To access externally hosted apps in k8s cluster (Example: Access AWS RDS Database endpoint by application present inside k8s cluster)

# Services



# Kubernetes
# Services
# Demo

# Services Demo



# Kubernetes
# YAML Basics

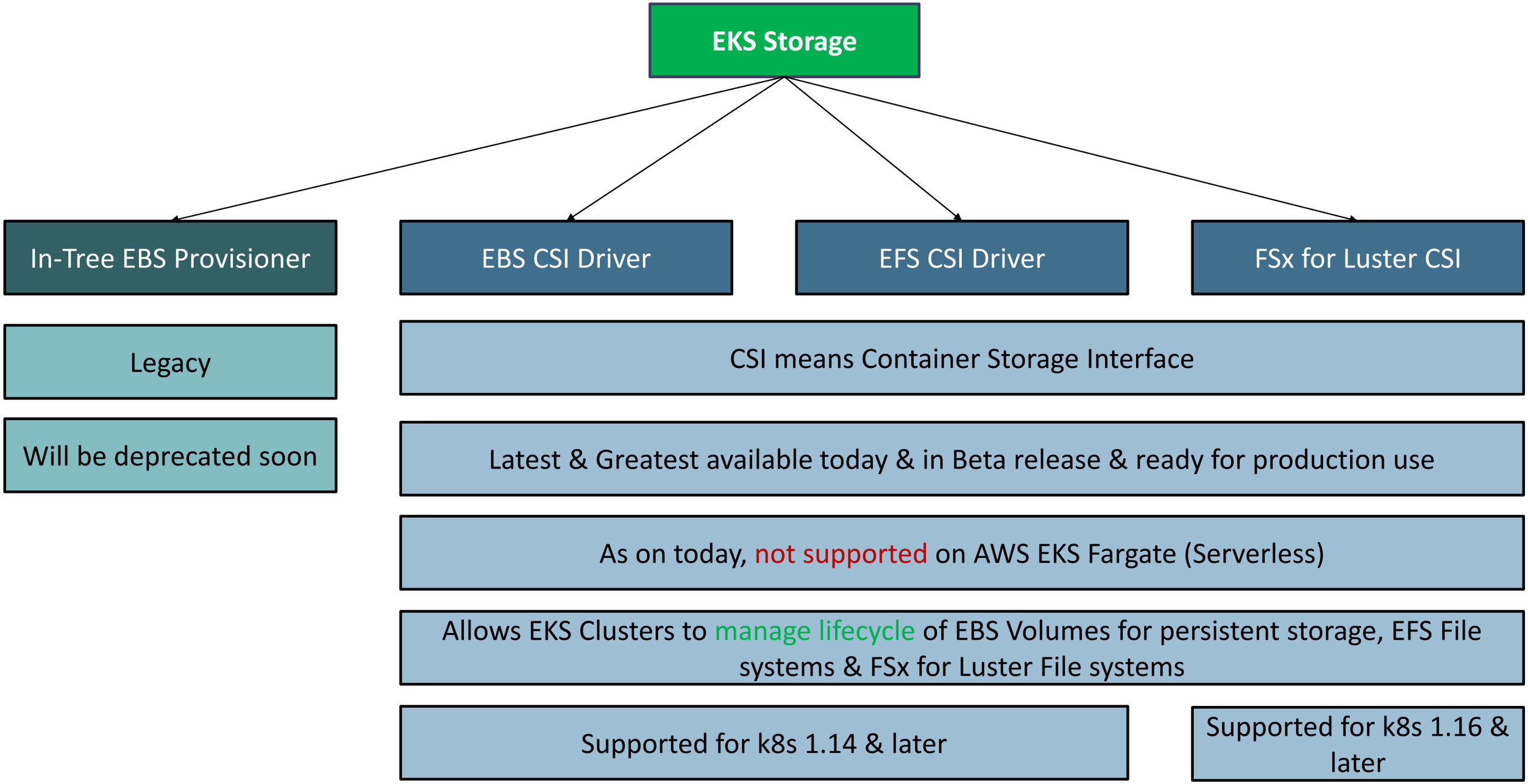
# YAML Basics

- YAML is not a Markup Language
- YAML is used to store information about different things
- We can use YAML  to define key, Value pairs like variables, lists and objects
- YAML is very similar to JSON (Javascript Object Notation)
- YAML primarily focuses on readability and user friendliness
- YAML is designed to be clean and easy to read
- We can define YAML files with two different extensions
  - abc.yml
  - abc.yaml

# YAML Basics

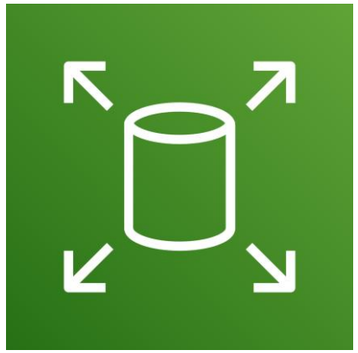
- YAML Comments
- YAML Key Value Pairs
- YAML Dictionary or Map
- YAML Array / Lists
- YAML Spaces
- YAML Document Separator

# AWS EKS
## Storage



# AWS EKS Storage

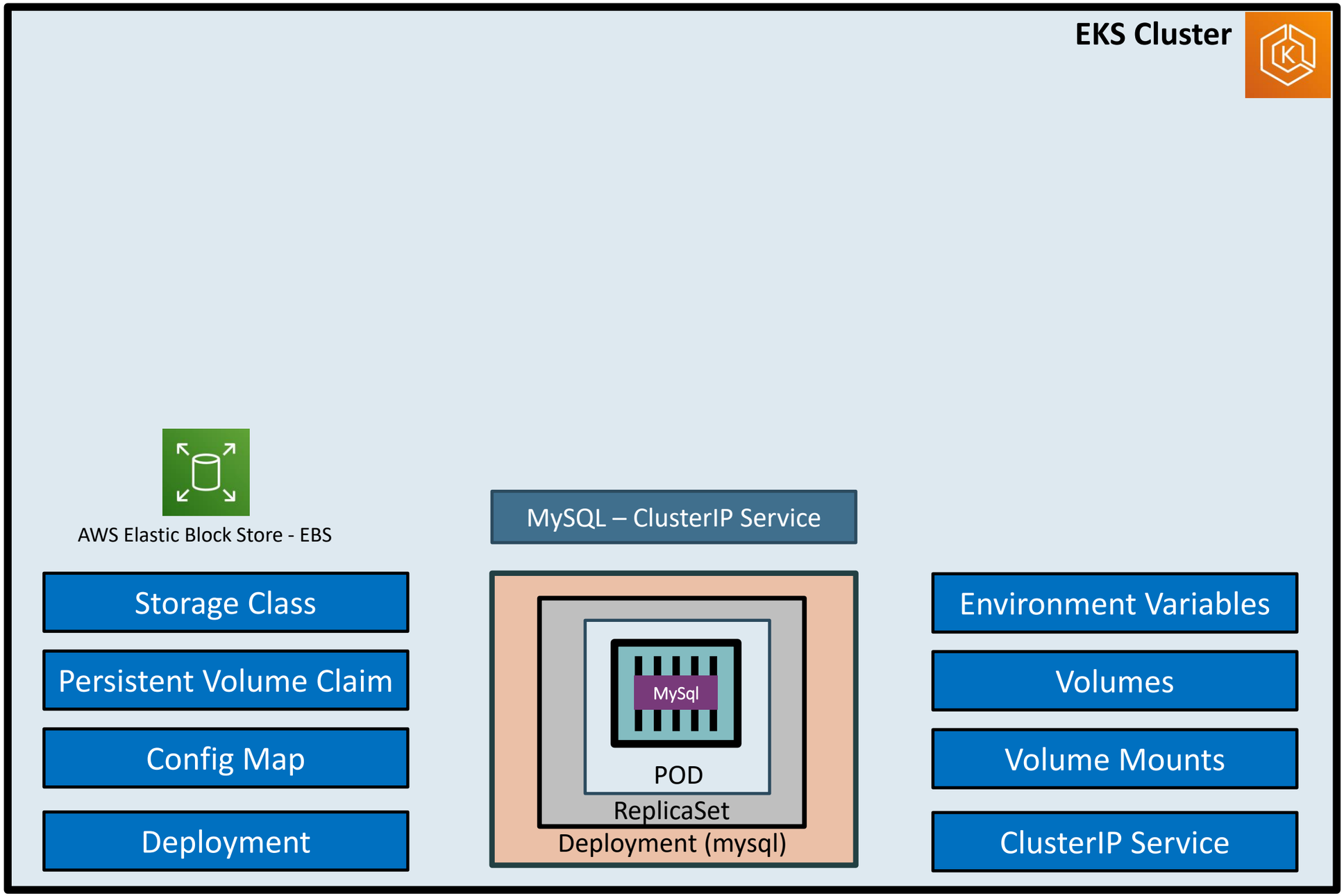
## EBS CSI Driver



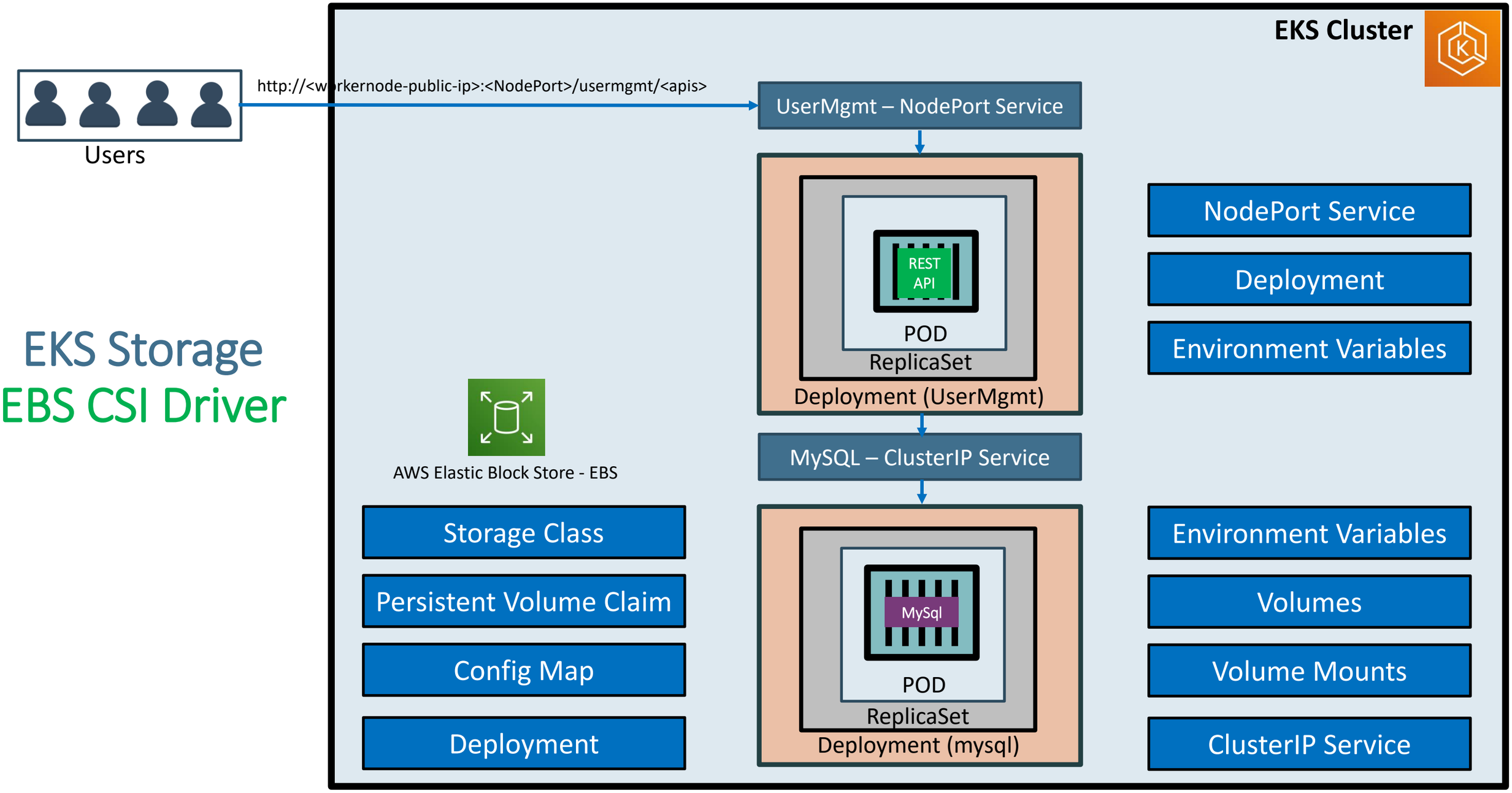
# AWS Elastic Block Store - Introduction

- 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.

# EKS Storage
## EBS CSI Driver



# EKS Storage
# EBS CSI Driver



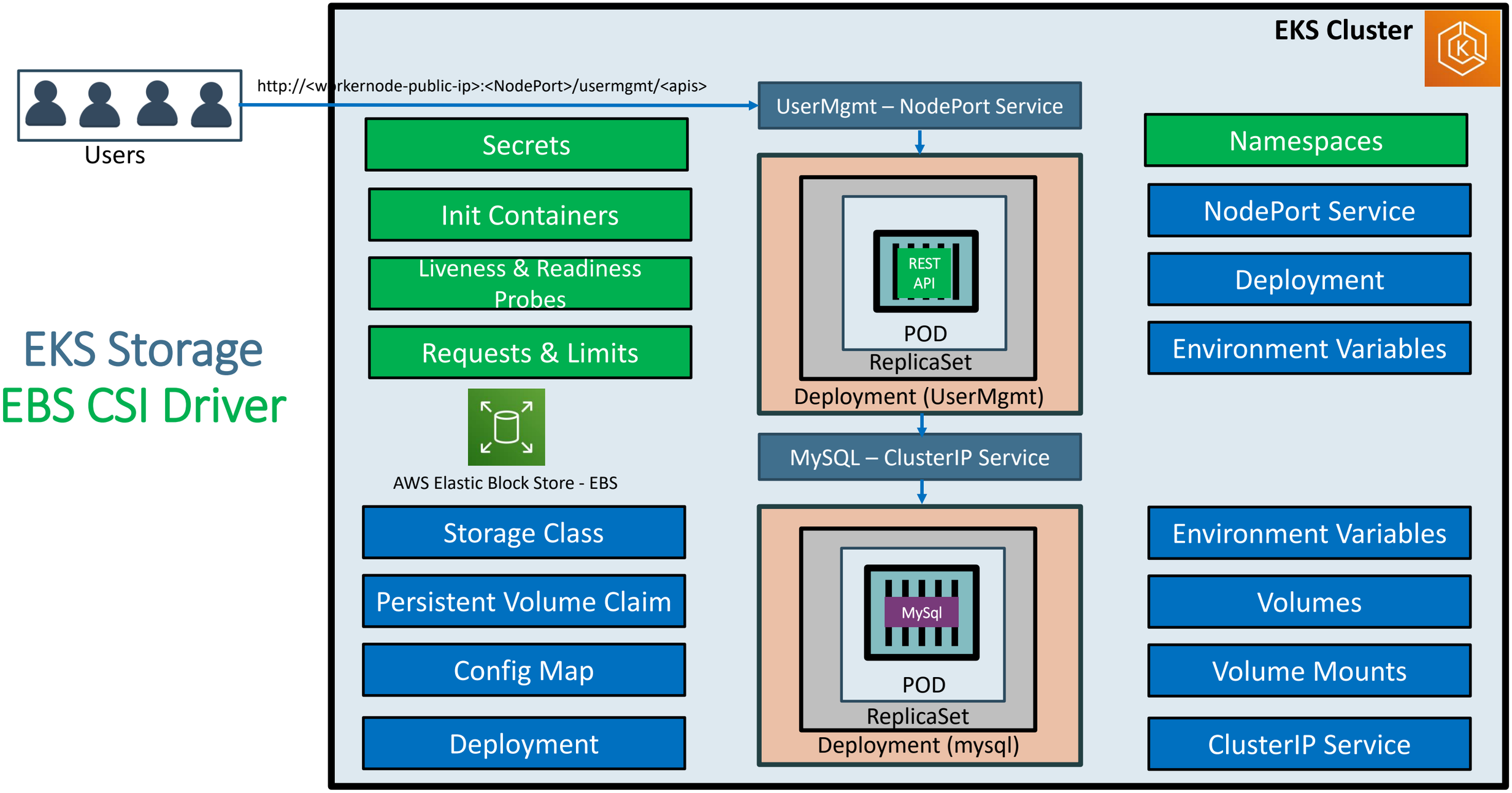
Elastic Block Store

# AWS EKS Storage

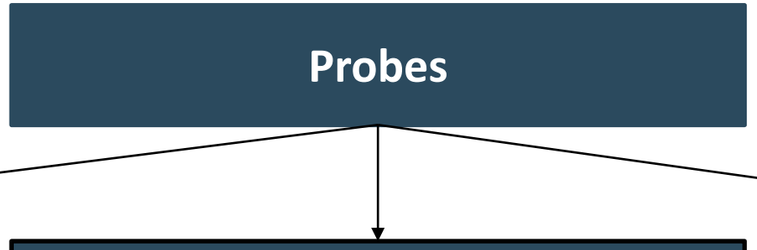
# EBS CSI Driver

## Important k8s Concepts for Application Deployments

# EKS Storage
# EBS CSI Driver



# Probes




## Liveness Probe

Kubelet uses liveness probes to know when to restart a container

Liveness probes could catch a deadlock, where an application is running, but unable to make progress and restarting container helps in such state

## Readiness Probe

Kubelet uses readiness probes to know when a container is ready to accept traffic

When a Pod is not ready, it is removed from Service load balancers based on this readiness probe signal.

## Startup Probe

Kubelet uses startup probes to know when a container application has started

Firstly this proble disables liveness & readiness checks until it succeeds ensuring those pods don't interfere with app startup.

This can be used to adopt liveness checks on slow starting containers, avoiding them getting killed by the kubelet before they are up and running.

## Options to define Probes

| | |
|---|---|
| Check using Commands | /bin/sh –c nc -z localhost 8095 |
| Check using HTTP GET Request | httpget path:/health-status |
| Check using TCP | tcpSocket Port: 8095 |

# Kubernetes
# Namespaces

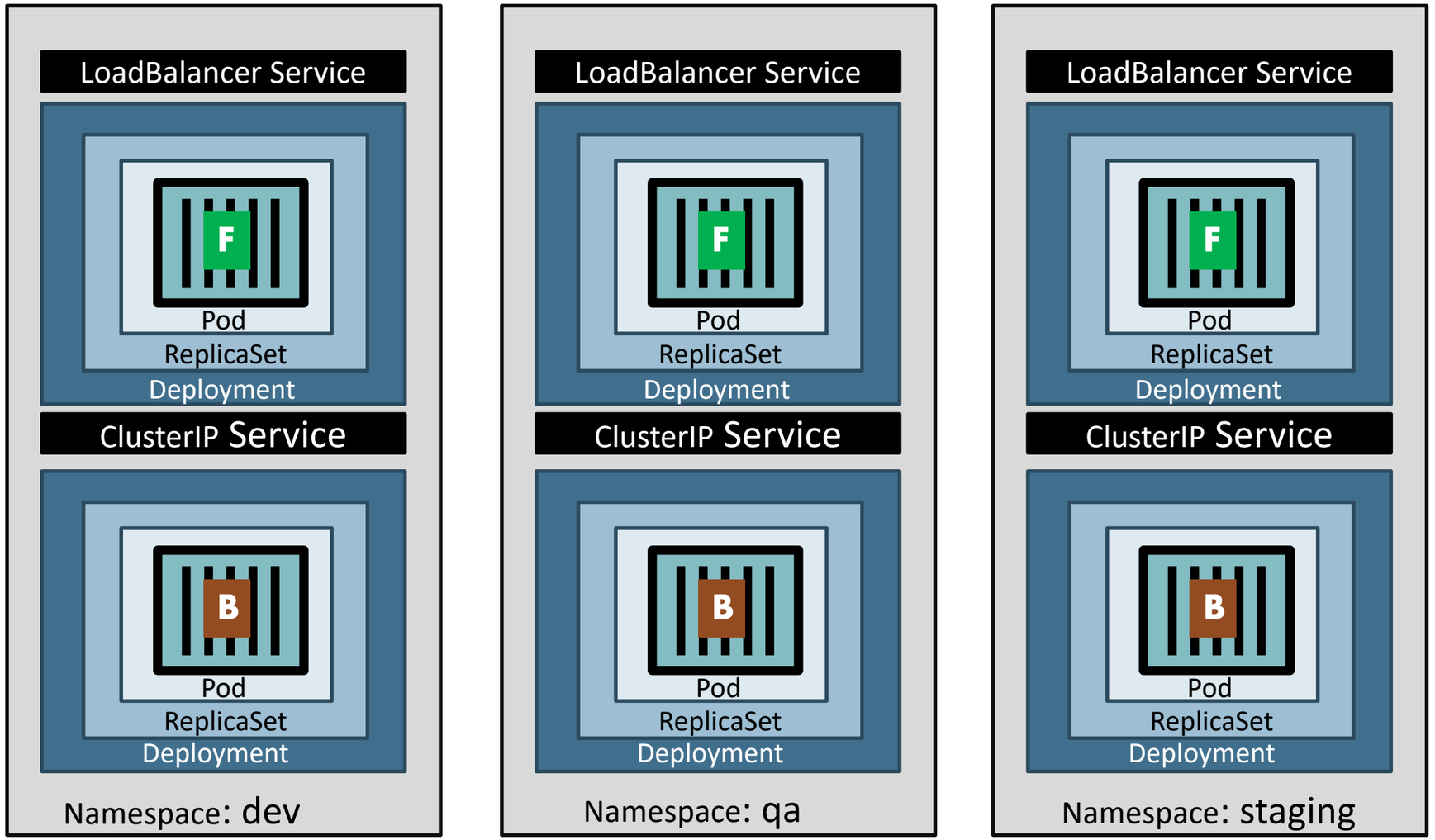
# Namespaces - Introduction

- Namespaces are also called Virtual clusters in our physical k8s cluster
- We use this in environments where we have many users spread across multiple teams or projects
- Clusters with tens of users ideally don't need to use namespaces
- Benefits
  - Creates isolation boundary from other k8s objects
  - We can limit the resources like CPU, Memory on per namespace basis (Resource Quota).

```
kubectl get namespace
```

```
NAME              STATUS   AGE
default           Active   141m
kube-node-lease   Active   141m
kube-public       Active   141m
kube-system       Active   141m
```

# Namespaces



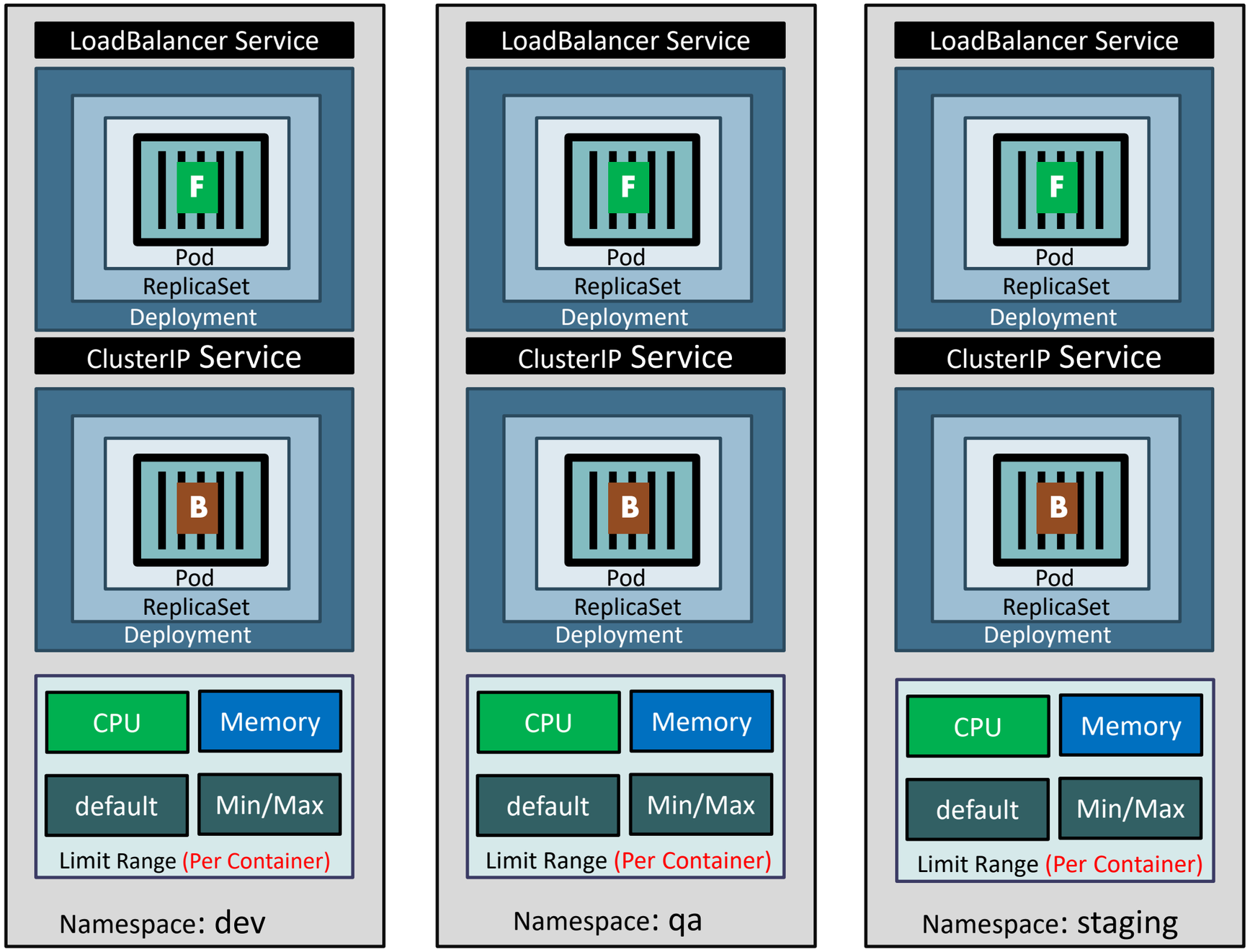

**Namespace Manifest - Declarative**

```
1  apiVersion: v1
2  kind: Namespace
3  metadata:
4    name: dev3
```

**Namespace Manifest - Imperative**

```
kubectl create namespace dev1
```

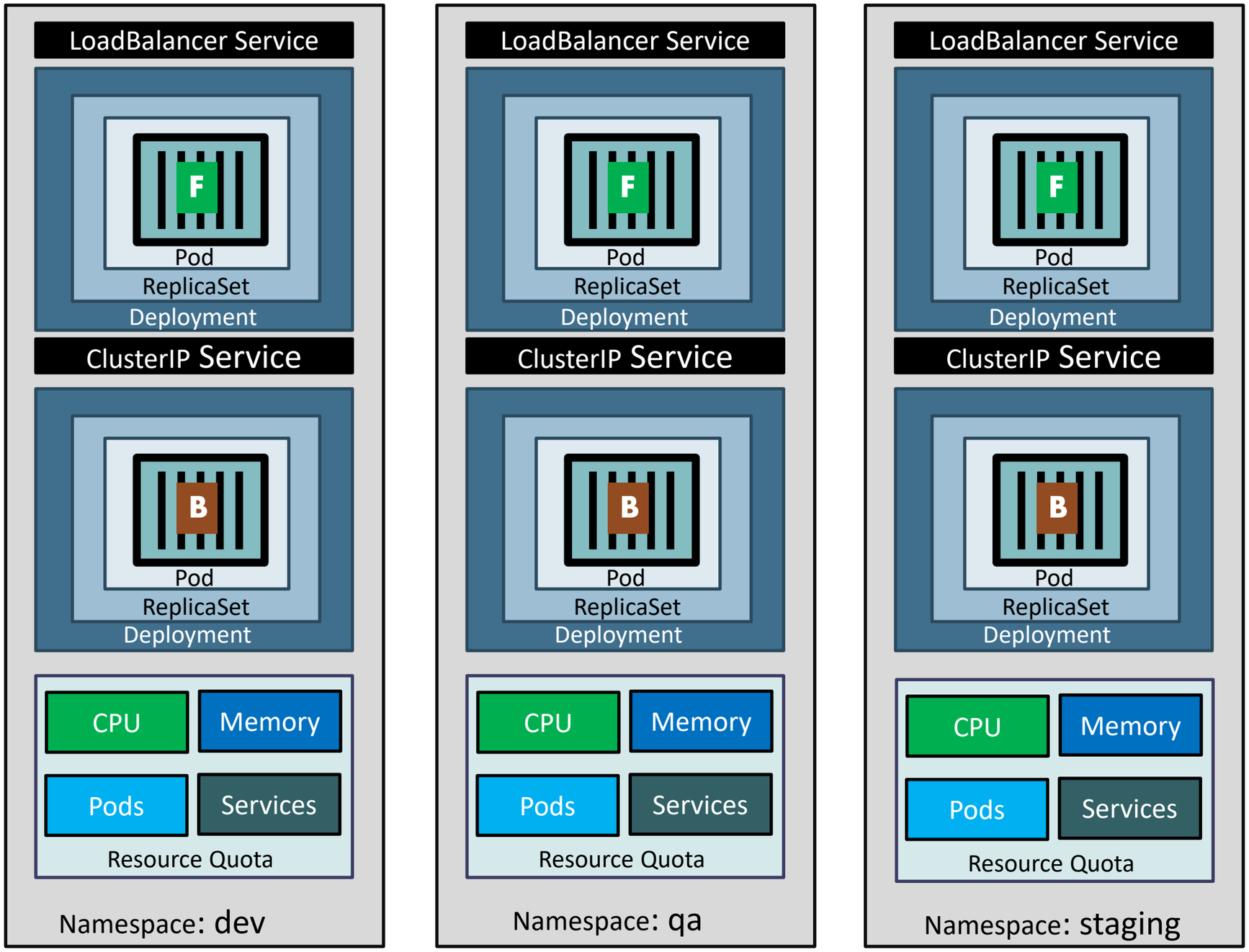
# Limit Range




**Limit Range Manifest**

```yaml
apiVersion: v1
kind: LimitRange
metadata:
  name: default-cpu-mem-limit-range
  namespace: dev3
spec:
  limits:
    - default:
        memory: "512Mi"
        cpu: "500m"
      defaultRequest:
        memory: "256Mi"
        cpu: "300m"
      type: Container
```

# Resource Quota




**Resource Quota Manifest**

```yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: ns-resource-quota
  namespace: dev3
spec:
  hard:
    requests.cpu: "1"
    requests.memory: 1Gi
    limits.cpu: "2"
    limits.memory: 2Gi
    pods: "5"
    configmaps: "5"
    persistentvolumeclaims: "5"
    secrets: "5"
    services: "5"
```

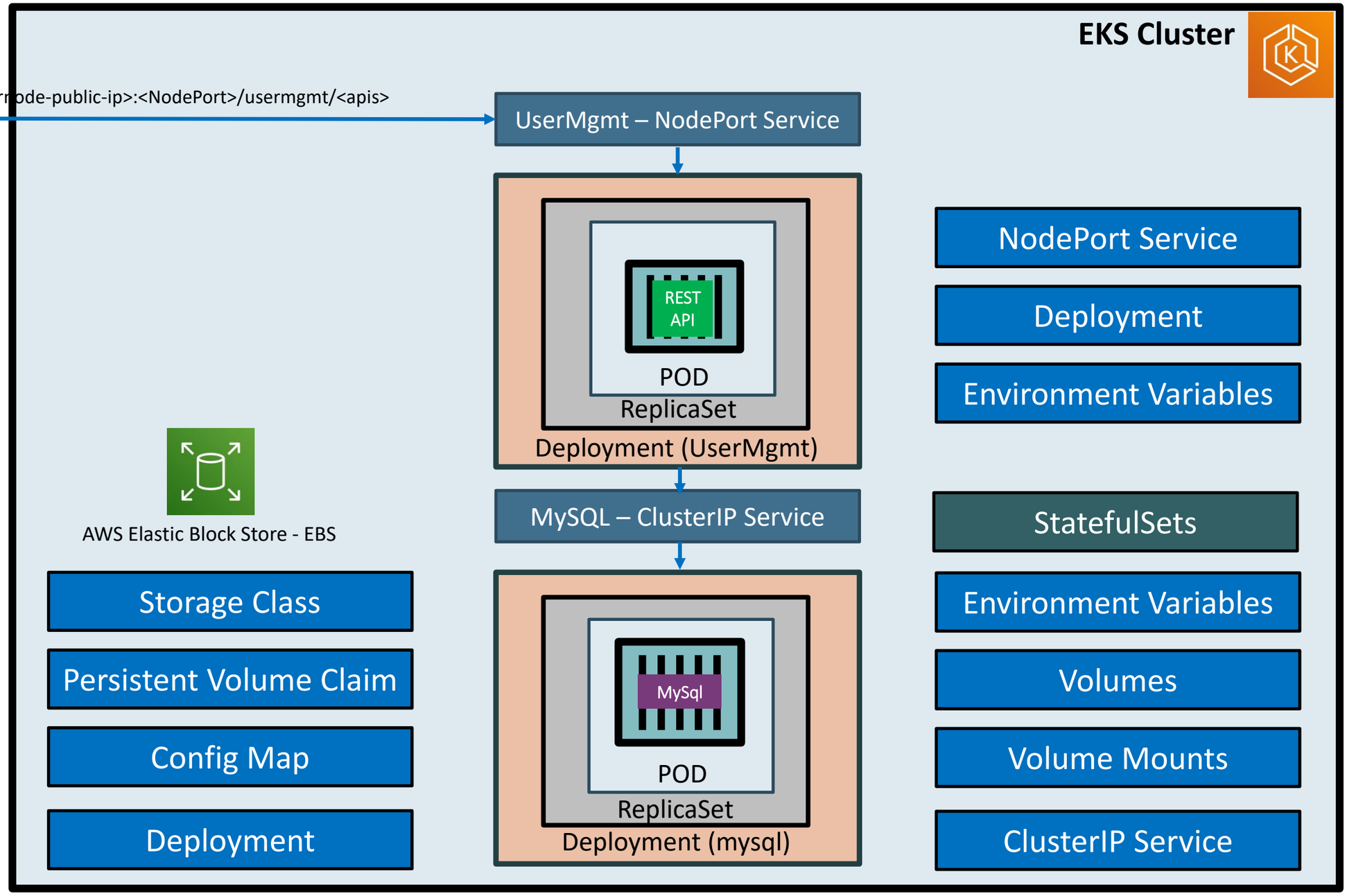
Amazon RDS

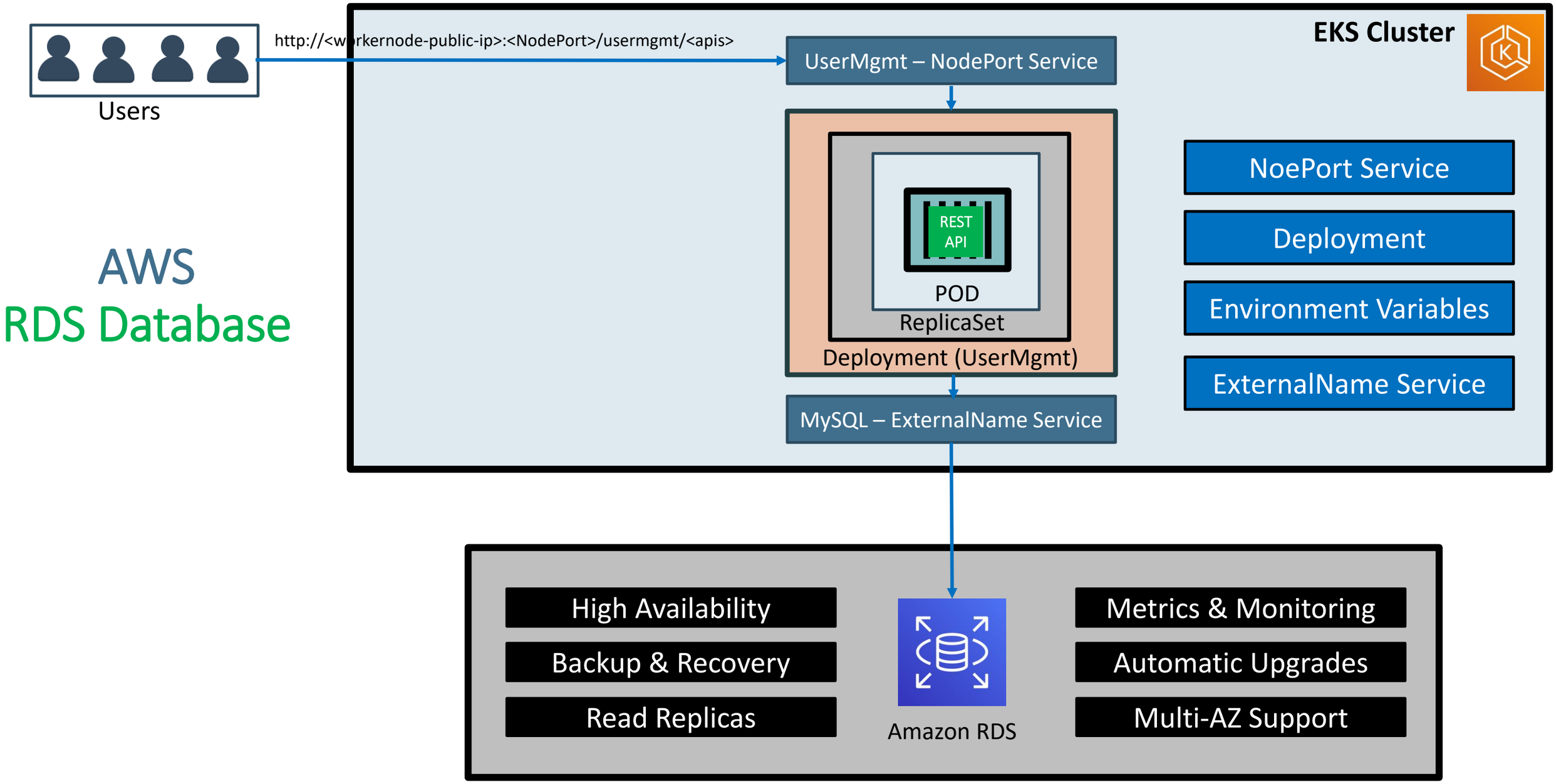
# AWS EKS
# &
# RDS Database

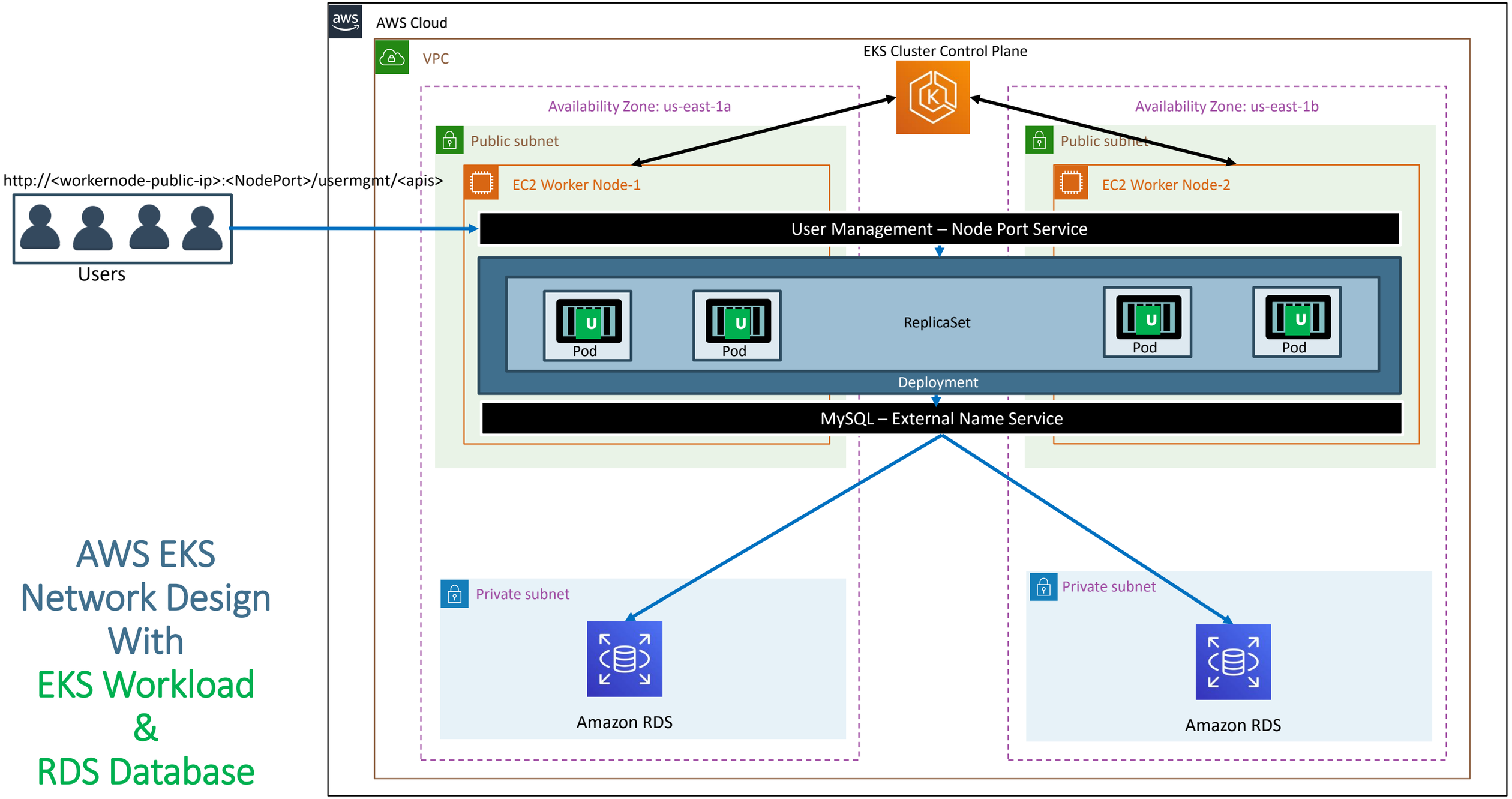
# EKS Storage
# EBS CSI Driver

- Drawbacks of EBS CSI for MySQL DB
- Complex setup to achieve HA
- Complex Multi-Az support for EBS
- Complex Master-Master MySQL setup
- Complex Master-Slave MySQL setup
- No Automatic Backup & Recovery
- No Auto-Upgrade MySQL


http://<workernode-public-ip>:<NodePort>/usermgmt/<apis>

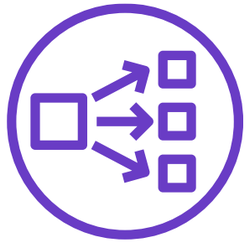







# AWS EKS Network Design With EKS Workload & RDS Database

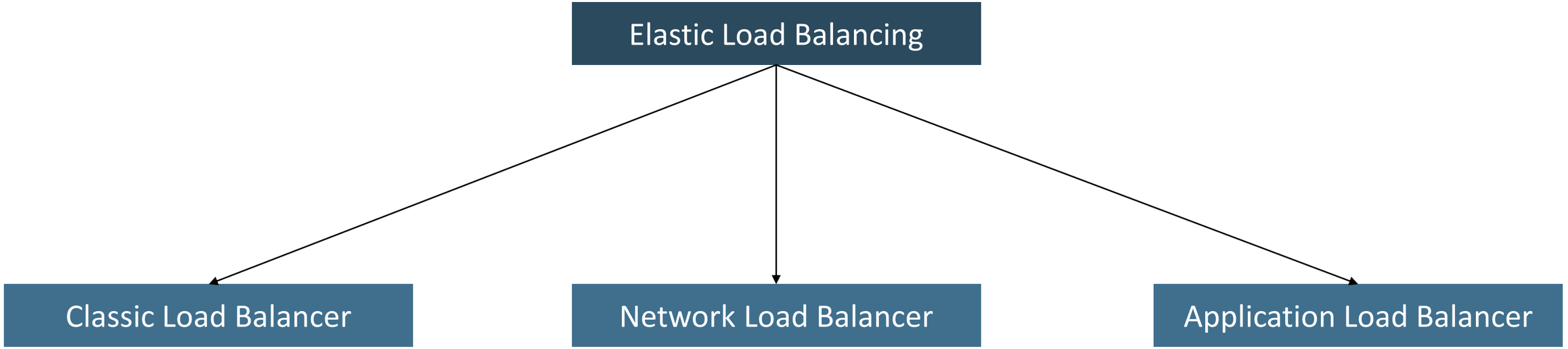
Classic
Load Balancer



Network
Load Balancer

Application
Load Balancer

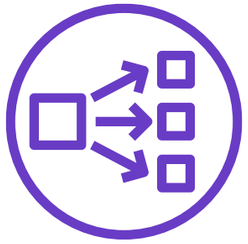
# AWS
# Elastic Load Balancing
## Overview




https://aws.amazon.com/elasticloadbalancing/features/#compare

Elastic Load Balancing

Classic Load Balancer



Network Load Balancer

Application Load Balancer

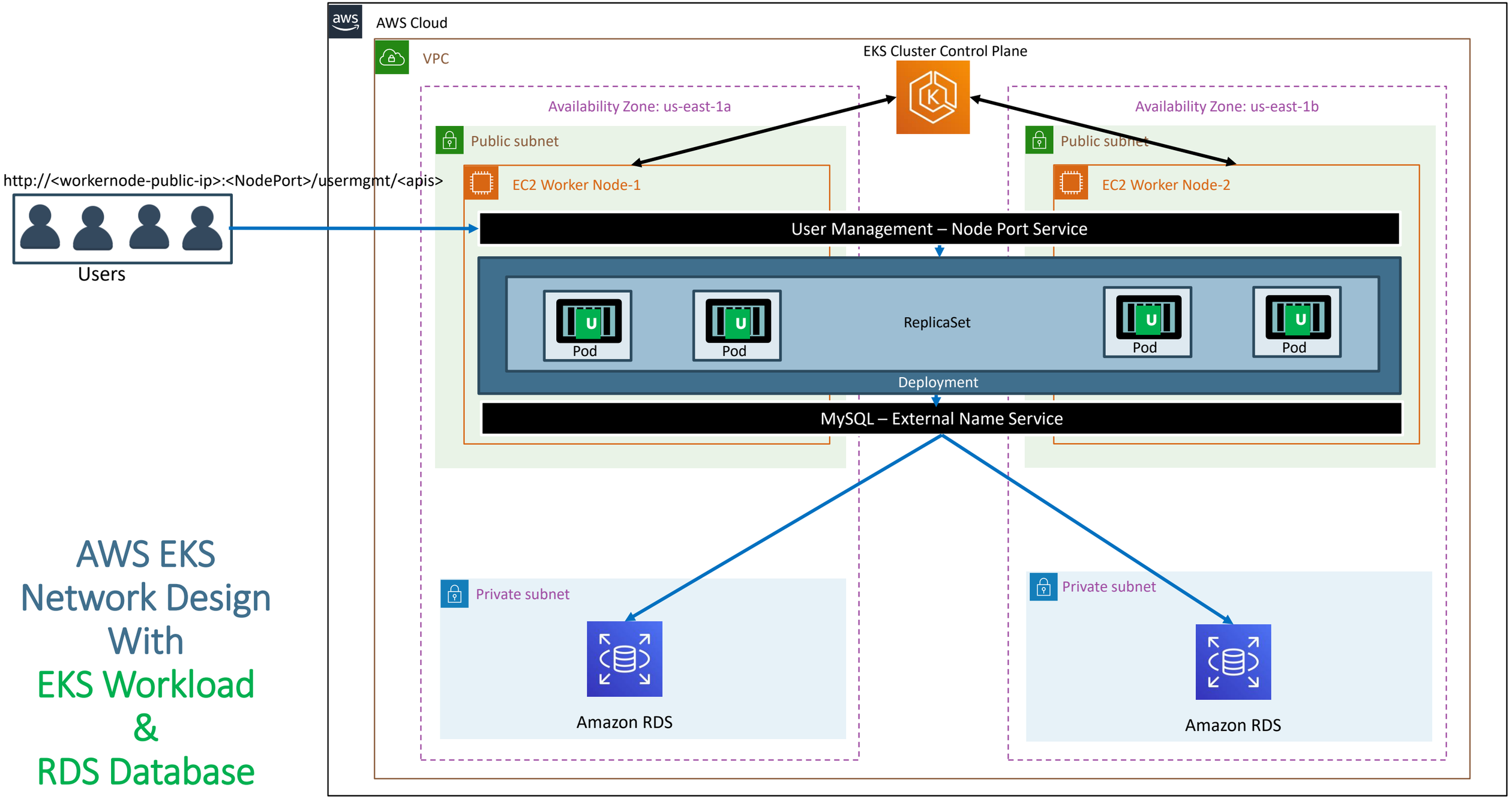
Amazon RDS

# AWS EKS

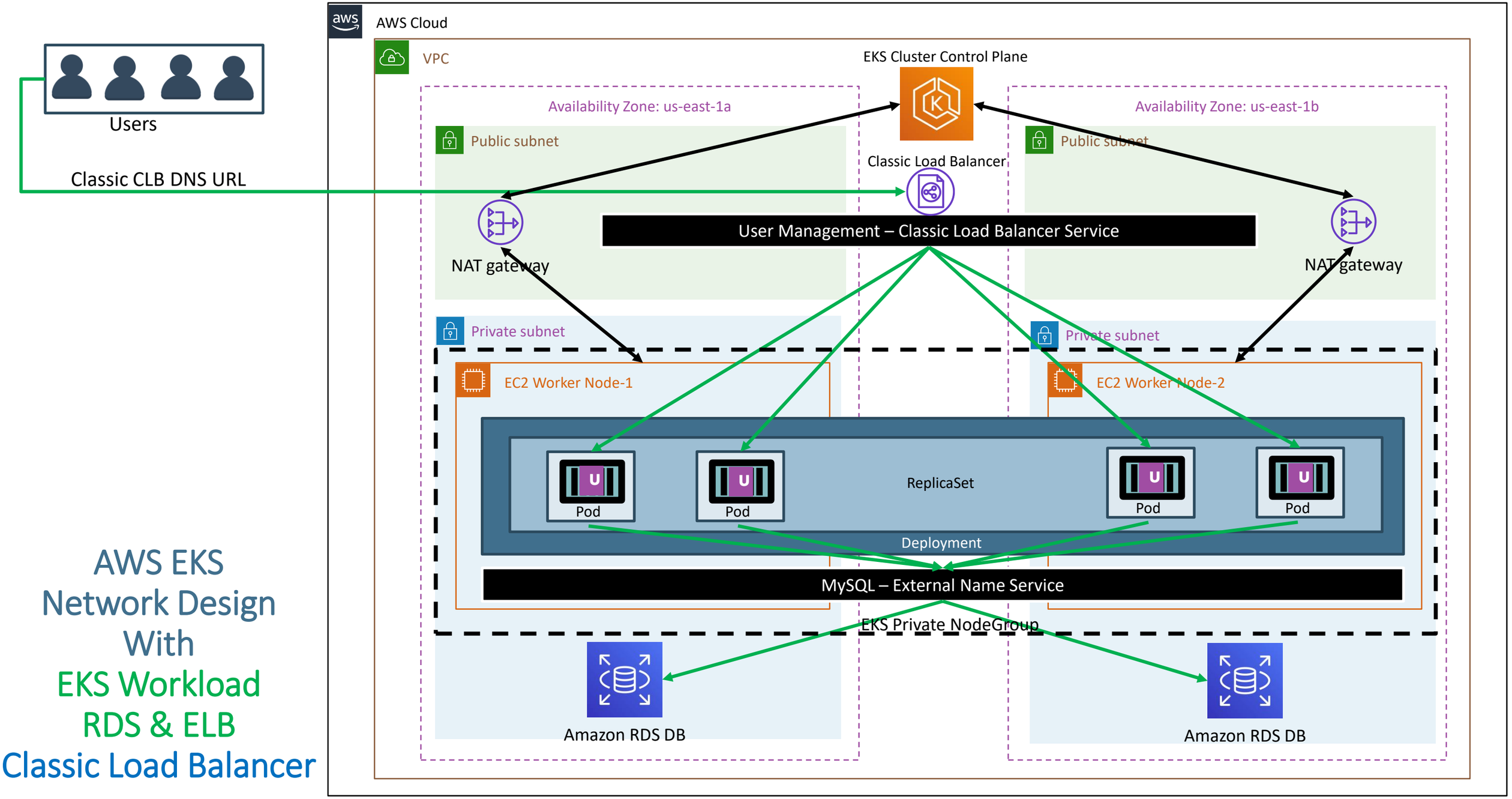
# &

# RDS & ELB

# Classic Load Balancer



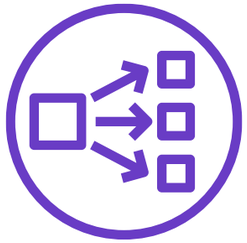

# AWS EKS Network Design With EKS Workload & RDS Database




# AWS EKS Network Design With EKS Workload RDS & ELB Classic Load Balancer

Elastic Load Balancing

Classic Load Balancer



Network Load Balancer

Application Load Balancer

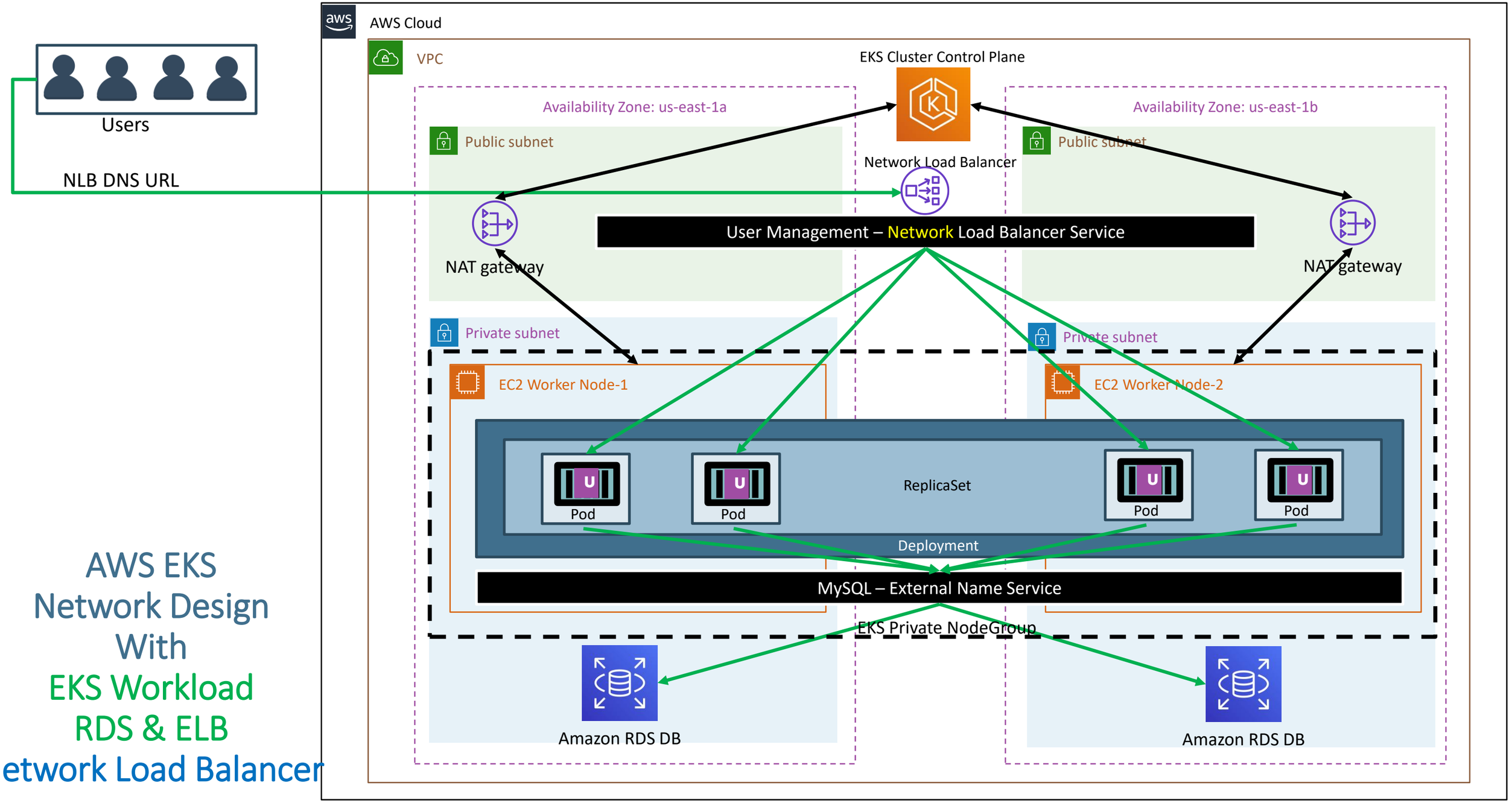
Amazon RDS

# AWS EKS

# &

# RDS & ELB

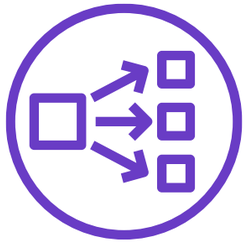
# Network Load Balancer




# AWS EKS Network Design With EKS Workload RDS & ELB Network Load Balancer

Elastic Load Balancing

Classic Load Balancer



Network Load Balancer

Application Load Balancer

Amazon RDS

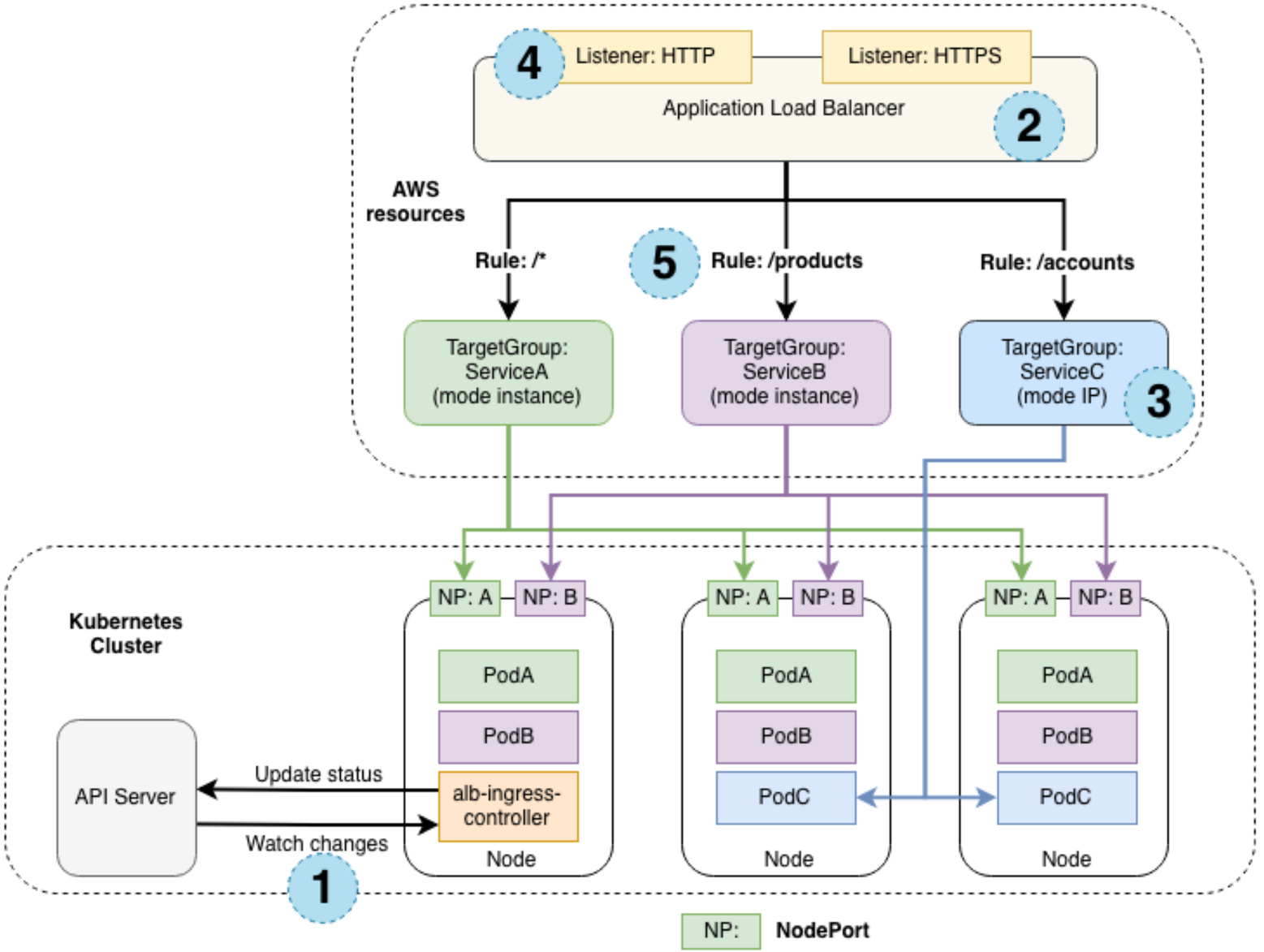
# AWS EKS

# &

# RDS & ELB

# Application Load Balancer

# How Ingress Works?



## k8s ClusterRole

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    app.kubernetes.io/name: alb-ingress-controller
  name: alb-ingress-controller
rules:
  - apiGroups:
      - ""
      - extensions
    resources:
      - configmaps
      - endpoints
      - events
      - ingresses
      - ingresses/status
      - services
      - pods/status
    verbs:
      - create
      - get
      - list
      - update
      - watch
      - patch
  - apiGroups:
      - ""
      - extensions
    resources:
      - nodes
      - pods
      - secrets
      - services
      - namespaces
    verbs:
      - get
      - list
      - watch
```

## k8s ServiceAccount

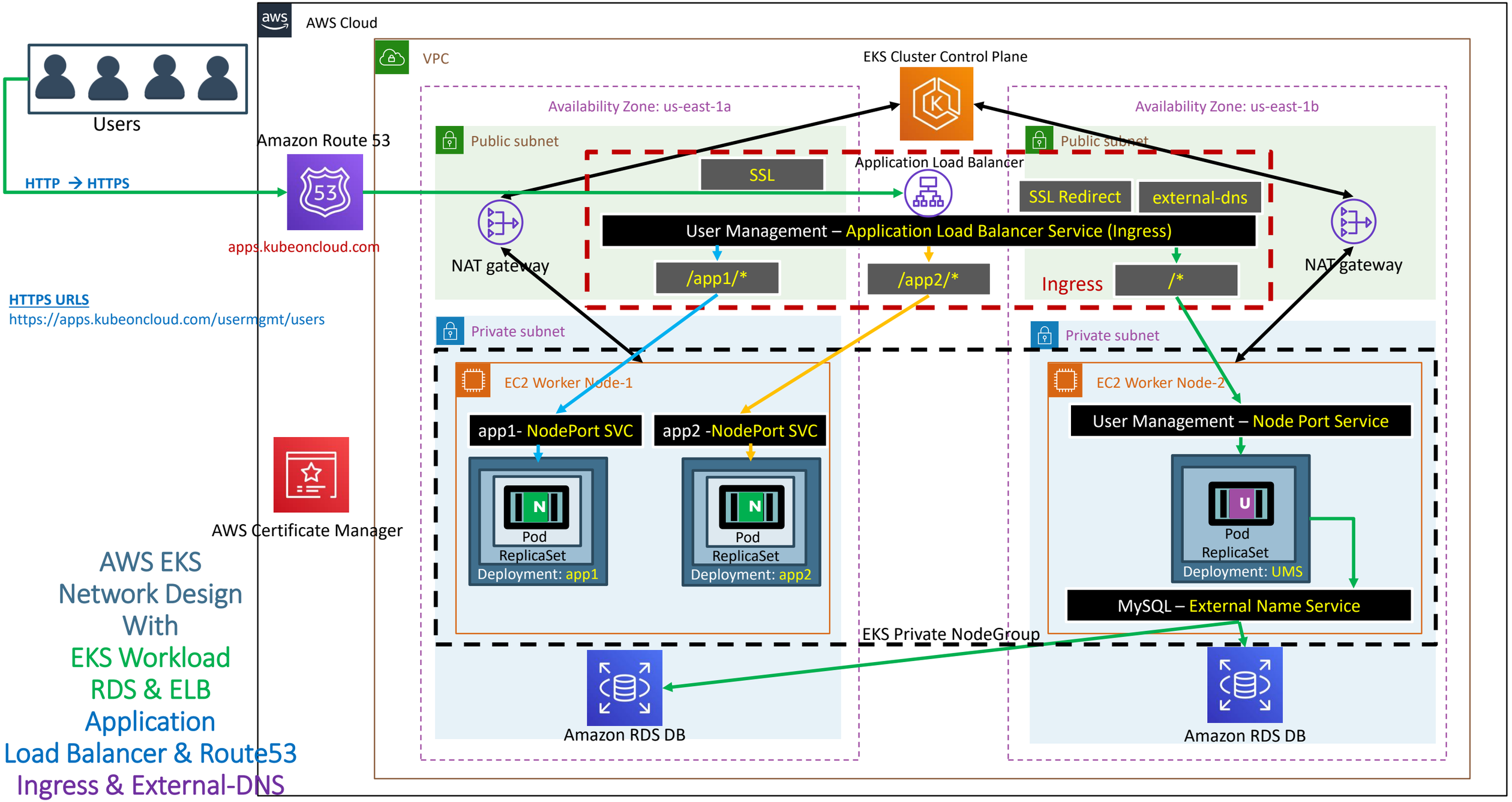
```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    app.kubernetes.io/name: alb-ingress-controller
  name: alb-ingress-controller
  namespace: kube-system
```

## k8s ClusterRoleBinding

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  labels:
    app.kubernetes.io/name: alb-ingress-controller
  name: alb-ingress-controller
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: alb-ingress-controller
subjects:
  - kind: ServiceAccount
    name: alb-ingress-controller
    namespace: kube-system
```

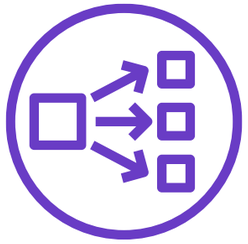
## AWS IAM Policy

https://raw.githubusercontent.com/kubernetes-sigs/aws-alb-ingress-controller/master/docs/examples/iam-policy.json


AWS Cloud
VPC
EKS Cluster Control Plane
Users
Amazon Route 53
HTTP → HTTPS
apps.kubeoncloud.com
Availability Zone: us-east-1a
Availability Zone: us-east-1b
Public subnet
Public subnet
Application Load Balancer
SSL
SSL Redirect
external-dns
User Management – Application Load Balancer Service (Ingress)
NAT gateway
NAT gateway
/app1/*
/app2/*
Ingress
/*
HTTPS URLS
https://apps.kubeoncloud.com/usermgmt/users
Private subnet
Private subnet
EC2 Worker Node-1
EC2 Worker Node-2
app1- NodePort SVC
app2 -NodePort SVC
User Management – Node Port Service
Pod
ReplicaSet
Deployment: app1
Pod
ReplicaSet
Deployment: app2
Pod
ReplicaSet
Deployment: UMS
MySQL – External Name Service
AWS Certificate Manager
EKS Private NodeGroup
Amazon RDS DB
Amazon RDS DB
AWS EKS
Network Design
With
EKS Workload
RDS & ELB
Application
Load Balancer & Route53
Ingress & External-DNS

Elastic Load Balancing

Classic Load Balancer



Network Load Balancer

Application Load Balancer

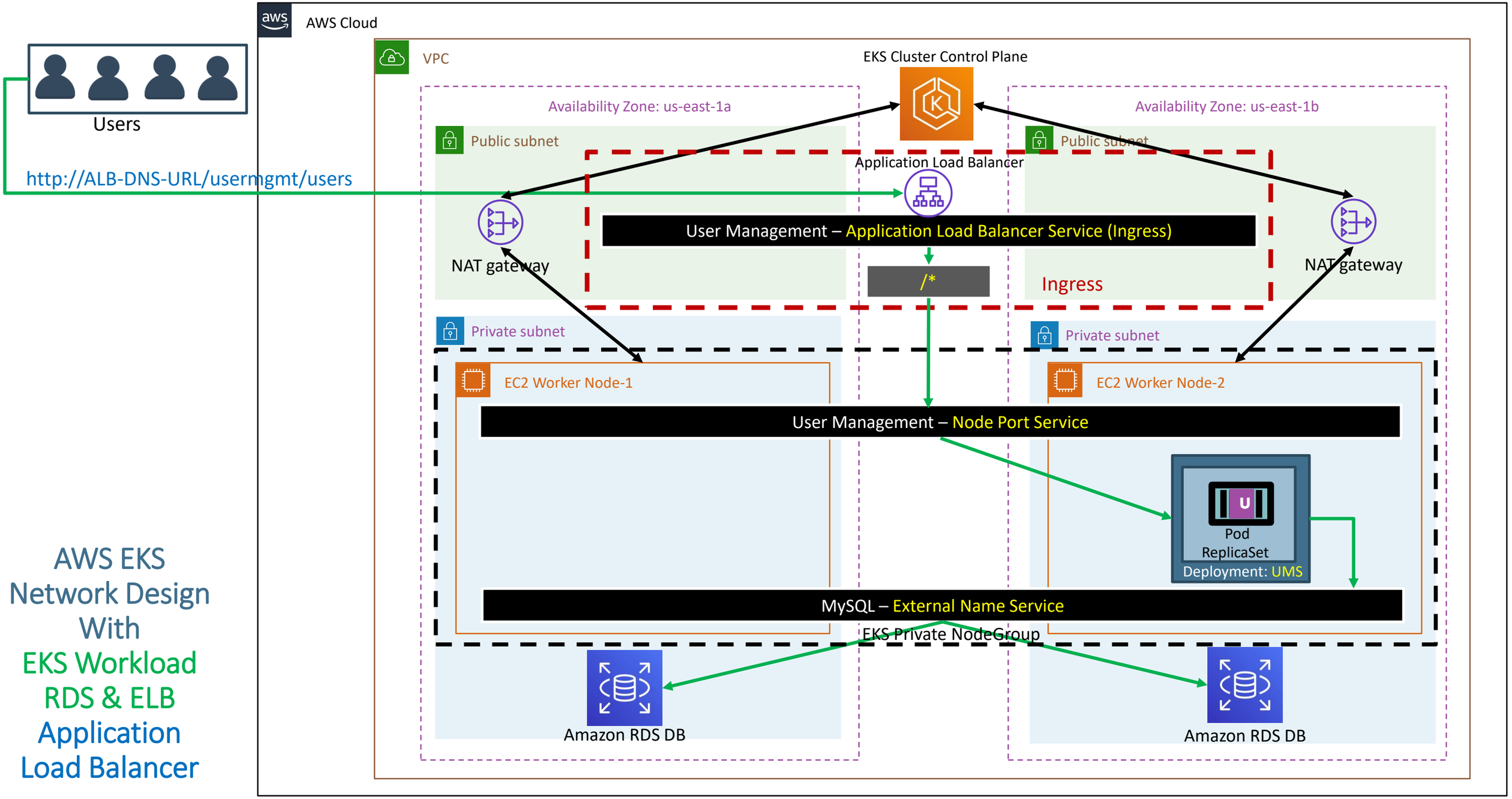
Amazon RDS

# AWS EKS

# RDS & ELB

# Application Load Balancer

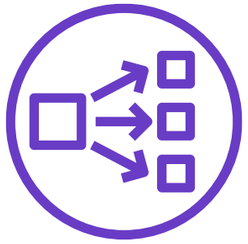
# Ingress Controller Basics




# AWS EKS Network Design With EKS Workload RDS & ELB Application Load Balancer

Elastic Load Balancing

Classic Load Balancer



Network Load Balancer

Application Load Balancer

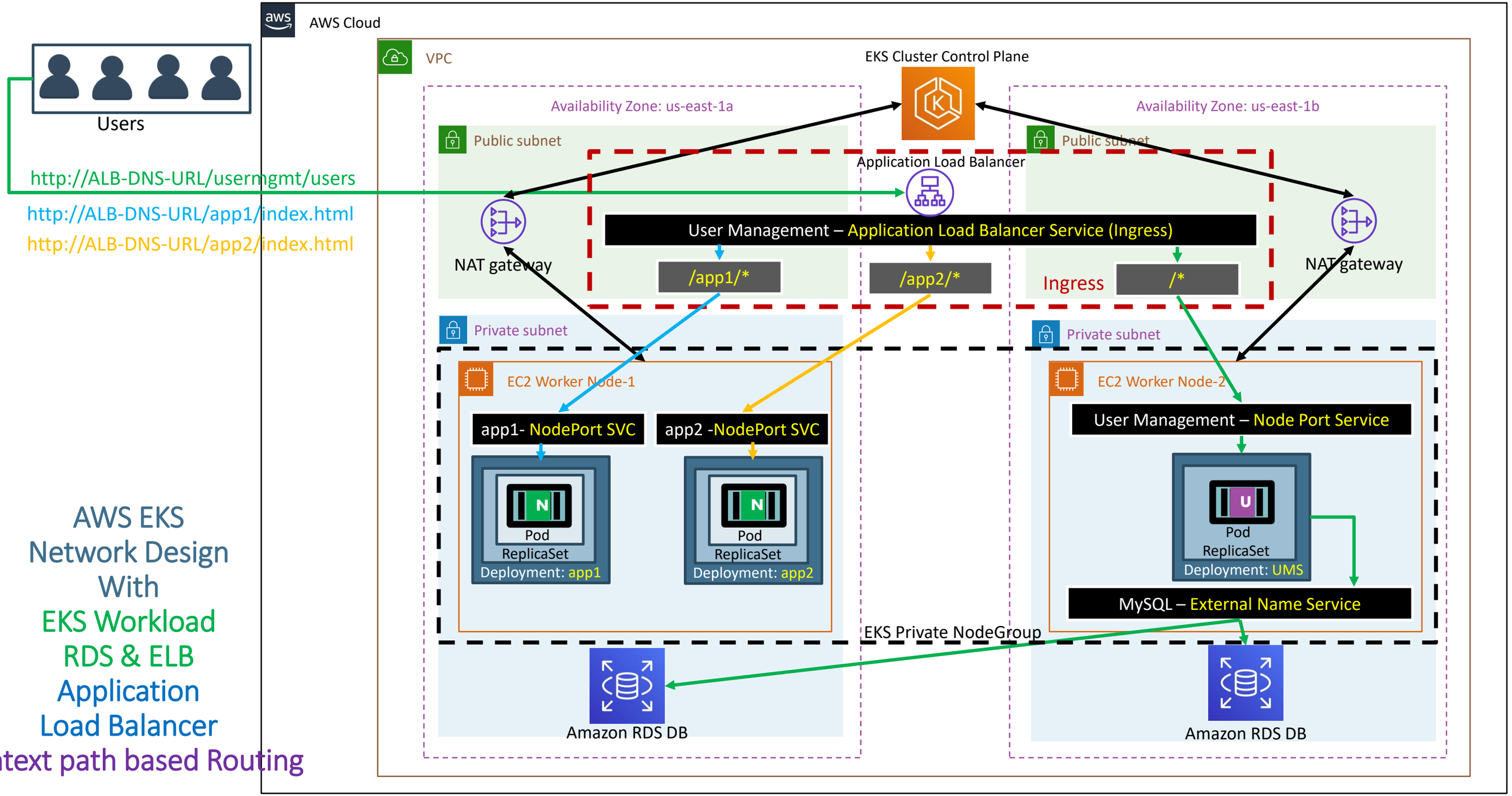
Amazon RDS

# AWS EKS

# RDS & ELB

# Application Load Balancer

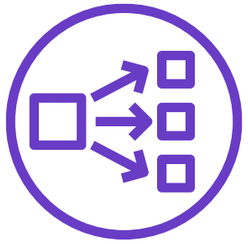
# Ingress Context Path based Routing




AWS EKS
Network Design
With
EKS Workload
RDS & ELB
Application
Load Balancer
ntext path based Routing

Elastic Load Balancing

Classic Load Balancer



Network Load Balancer

Application Load Balancer

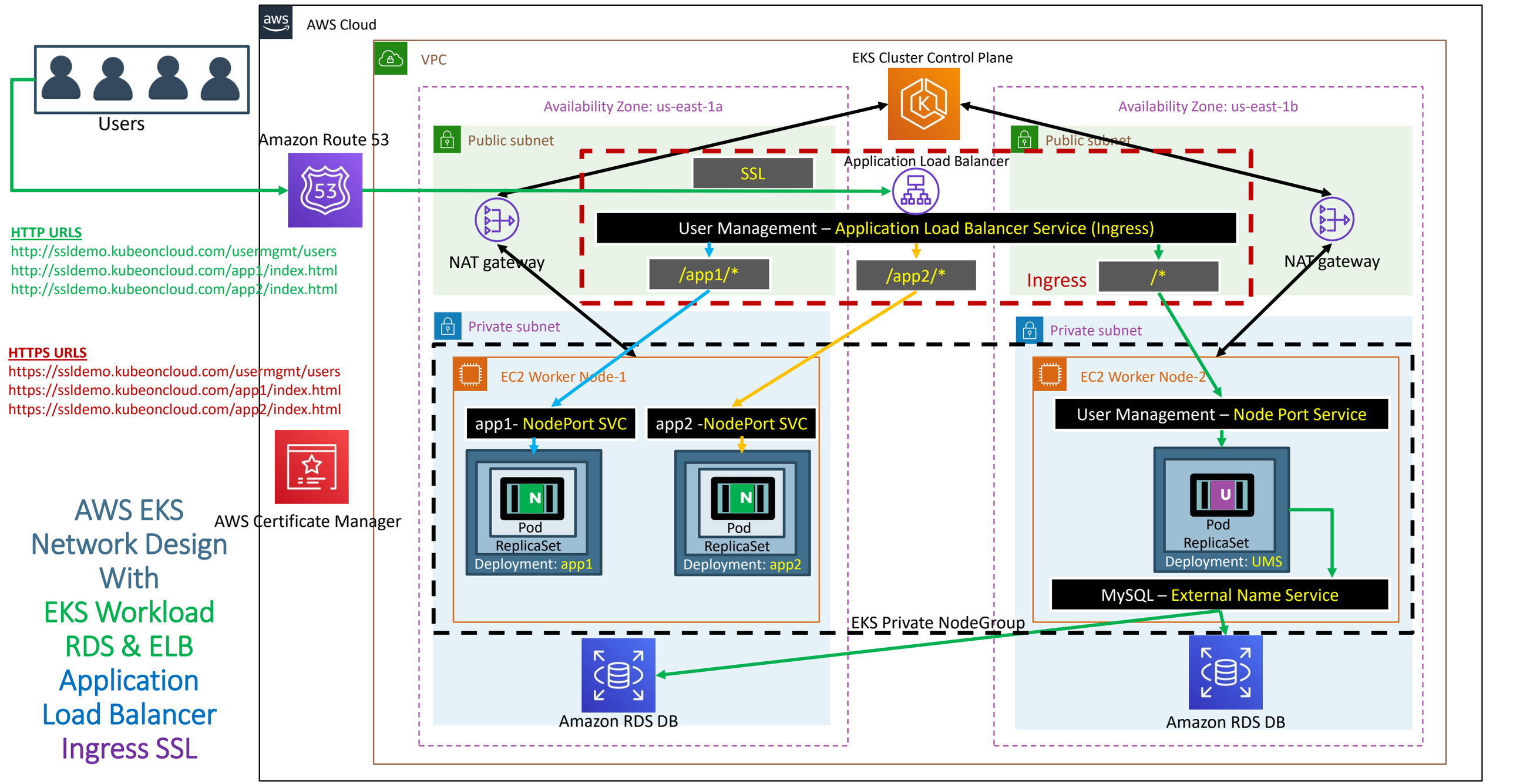
Amazon RDS

# AWS EKS

# RDS & ELB

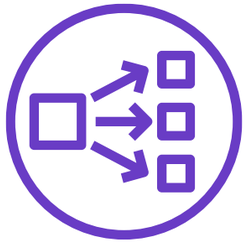
# Application Load Balancer

# Ingress SSL



Elastic Load Balancing

Classic Load Balancer



Network Load Balancer

Application Load Balancer

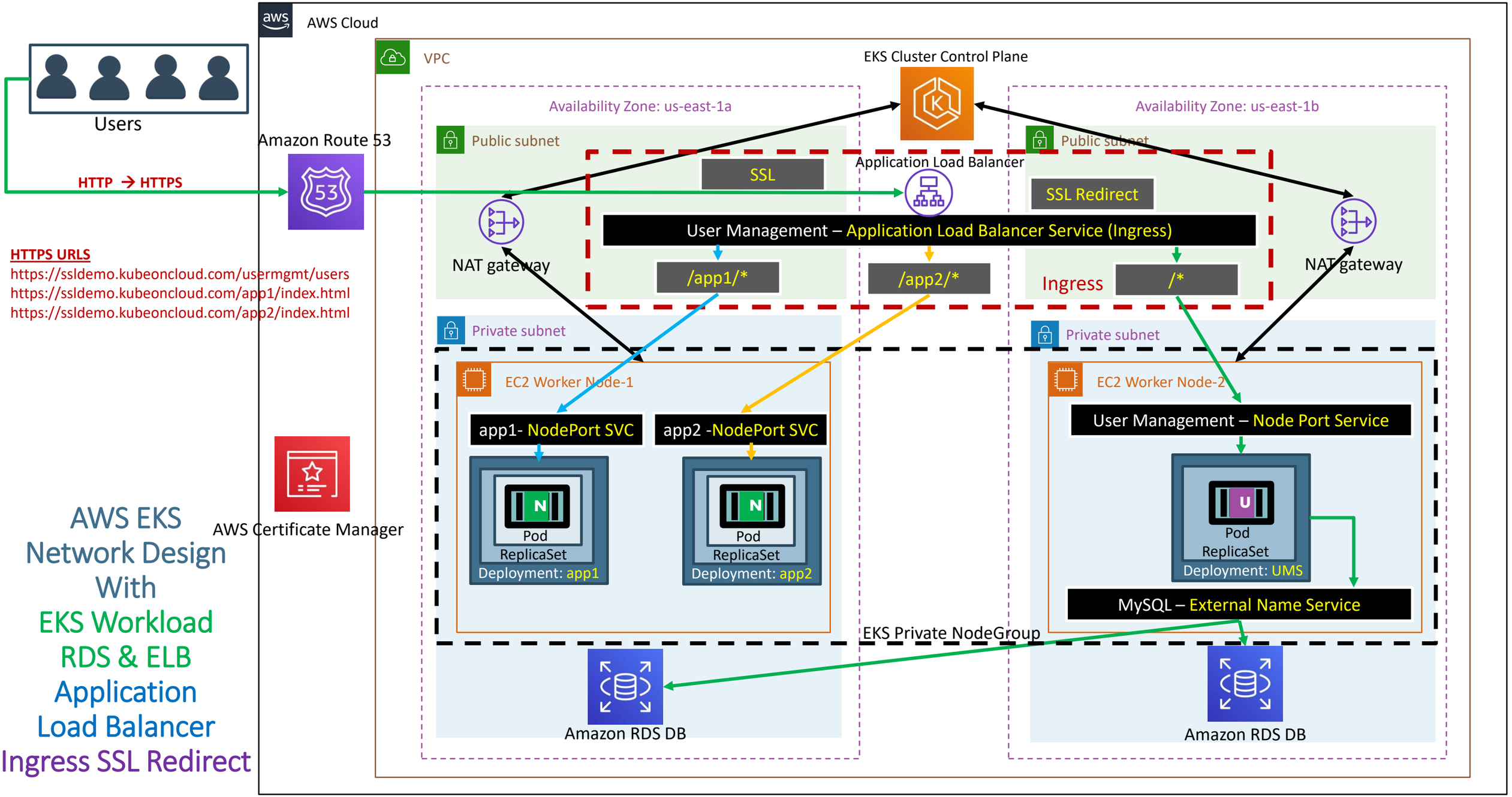
Amazon RDS

# AWS EKS

# RDS & ELB

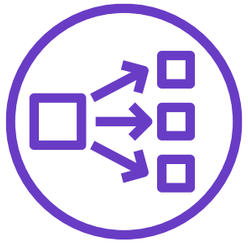
# Application Load Balancer

# Ingress SSL Redirect



Elastic Load Balancing

Classic Load Balancer



Network Load Balancer

Application Load Balancer

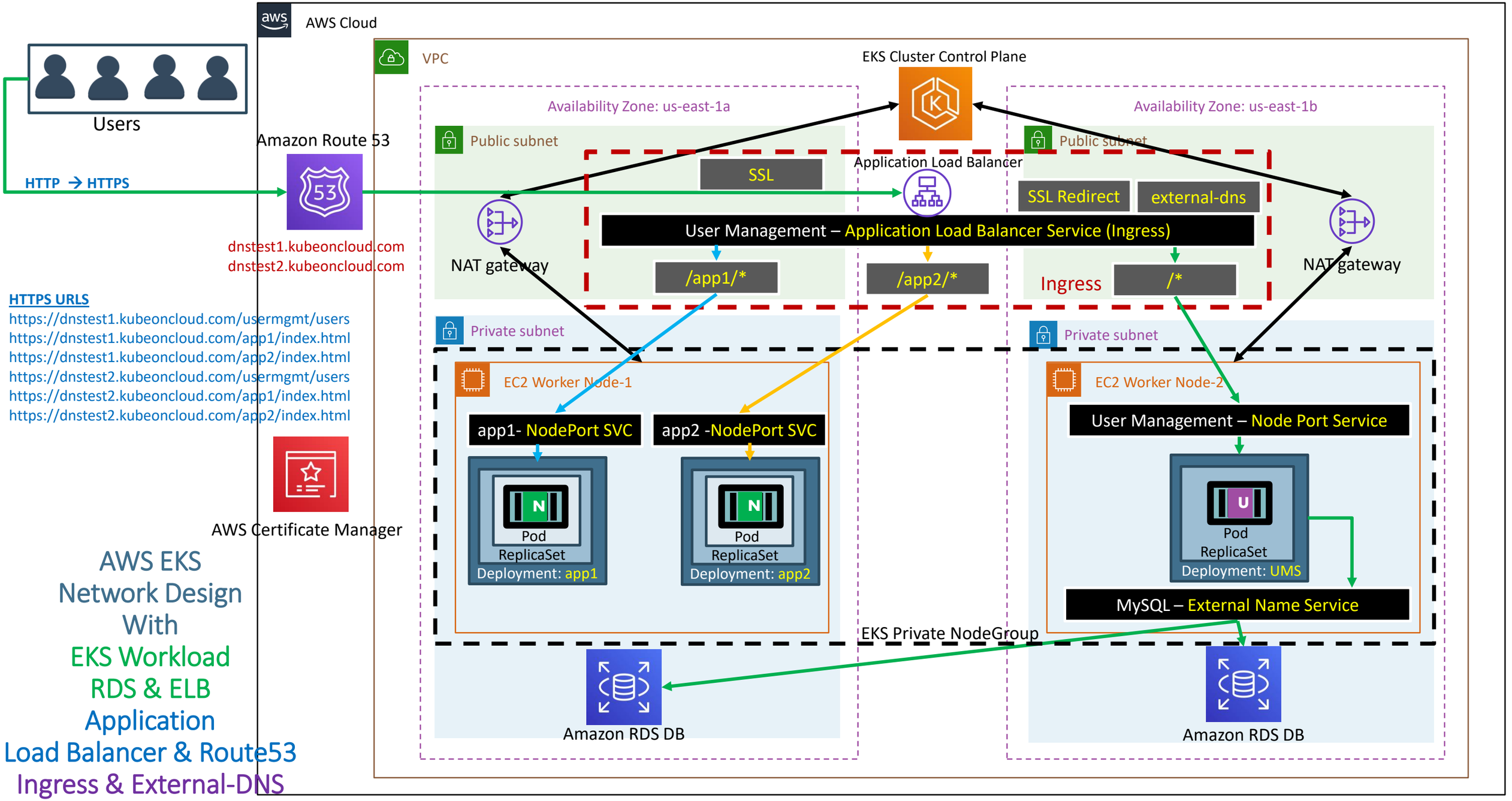
Amazon RDS

# AWS EKS

# RDS & ELB

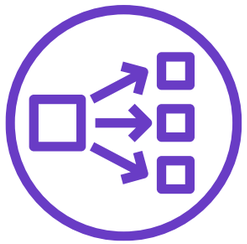
# Application Load Balancer & Route53

# Ingress & External-DNS



Elastic Load Balancing

Classic Load Balancer



Network Load Balancer

Application Load Balancer

Certificate Manager

Route53

Elastic Block Store

Amazon RDS

# AWS EKS

## Fargate Profiles

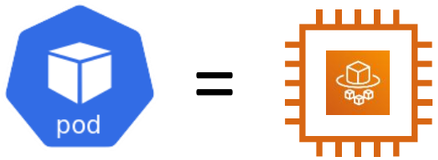
## Serverless

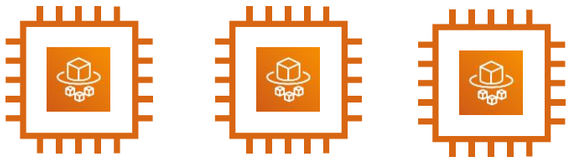
Fargate Profiles

# 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 using the upstream, extensible model provided by Kubernetes.
- 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 in addition to several mutating and validating admission controllers.
- When we start a pod that meets the criteria for running on Fargate, the Fargate controllers running in the cluster recognize, update, 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

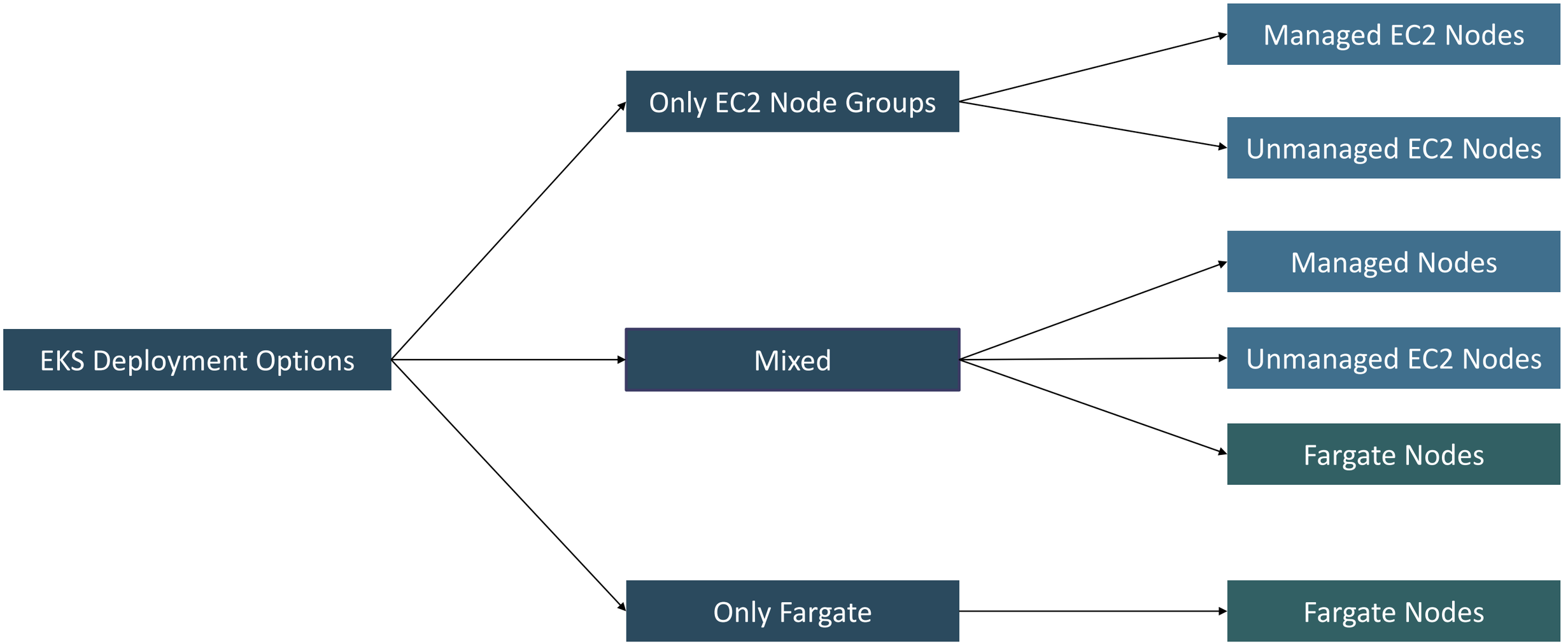
## Production Ready

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

## Rightsized and Integrated

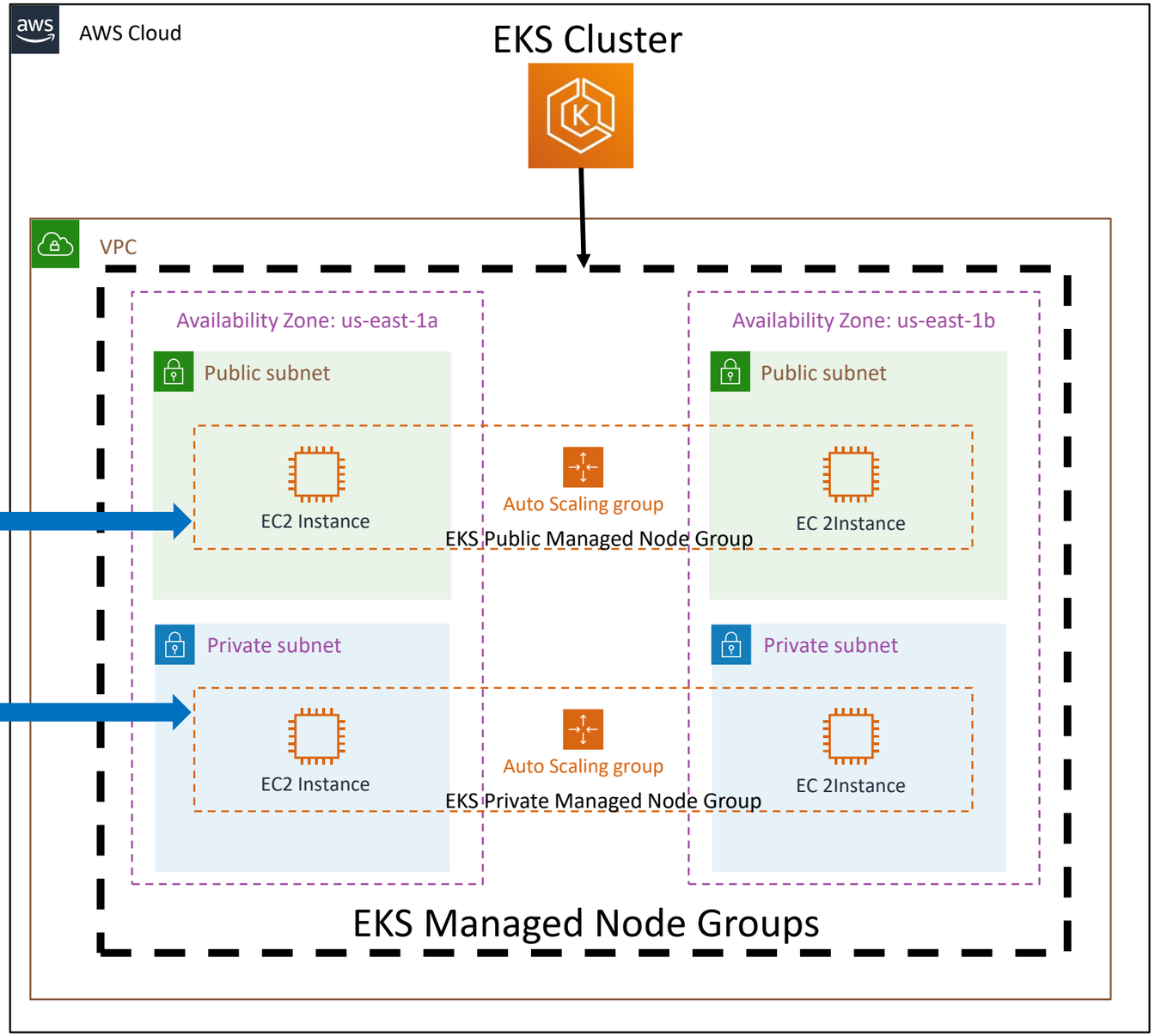
- Only pay for resources you need to run your pods
- Includes native AWS integrations for networking and security

# EKS Deployment Options

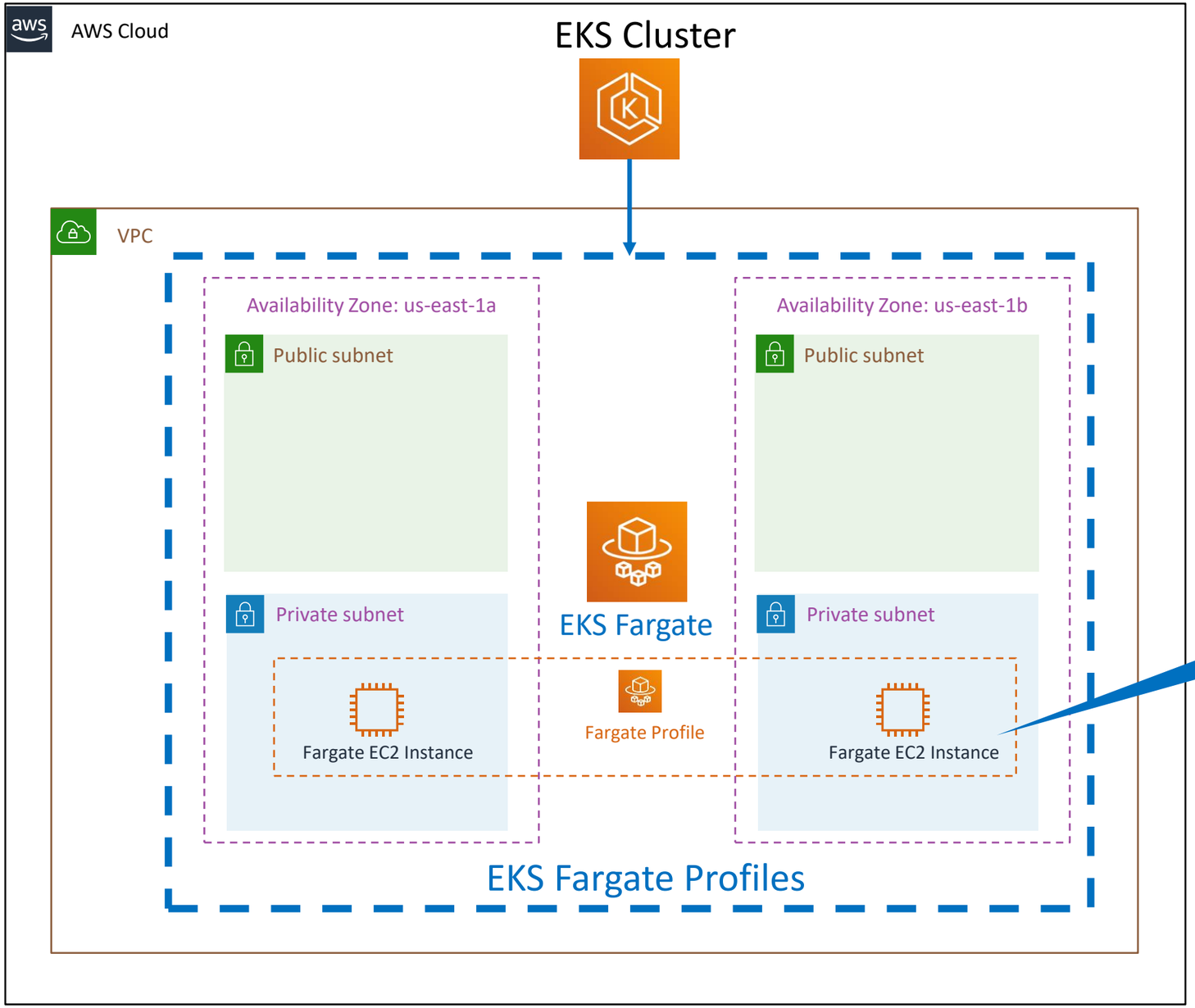


# EKS Deployment Options – EC2 Node Groups

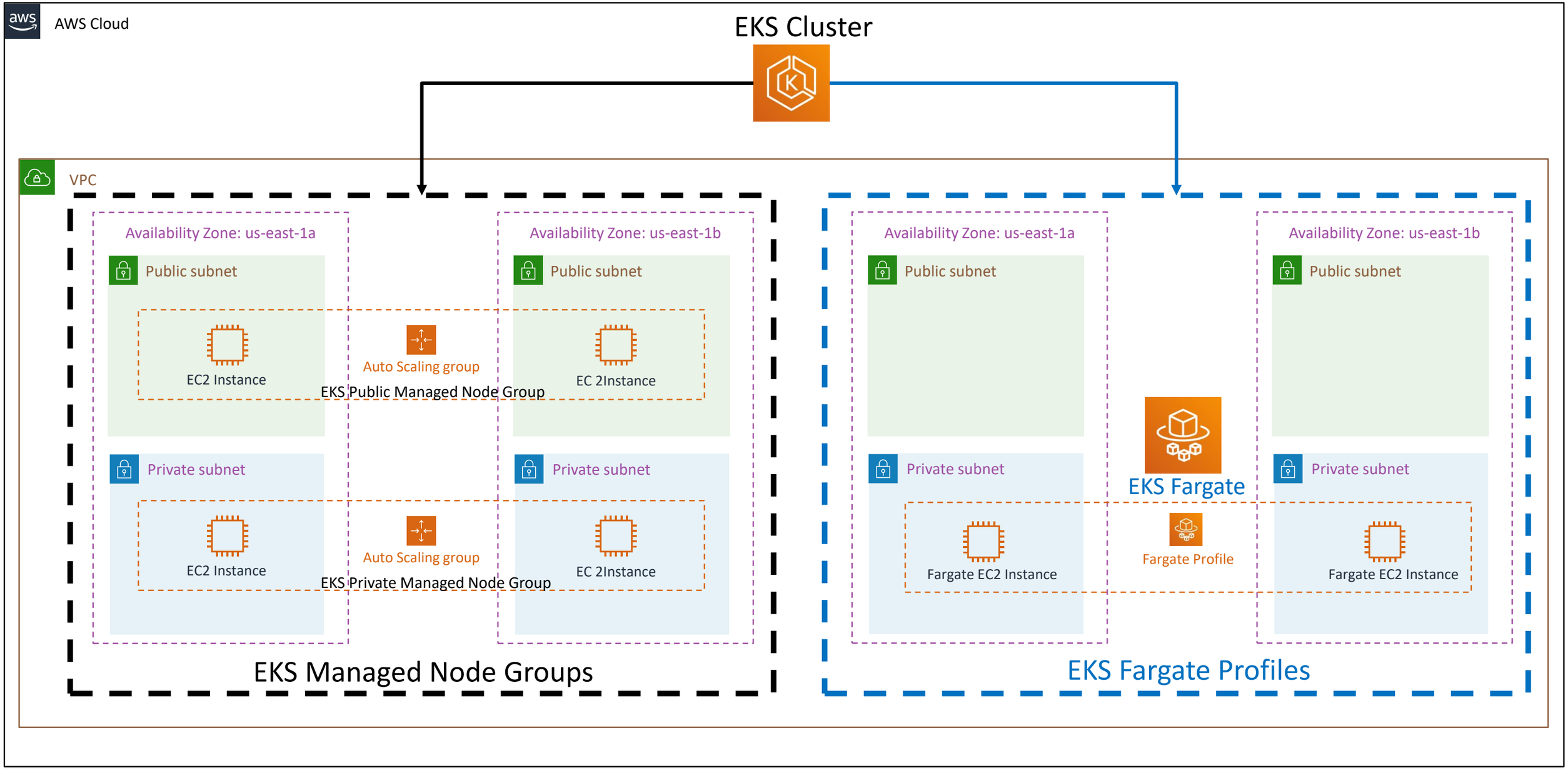
We can deploy EC2 Managed Node Groups in both public and private subnets of a VPC



# EKS Deployment Options – Only Fargate



# EKS Deployment Options - Mixed



# EKS Fargate vs Managed vs Unmanaged Nodes

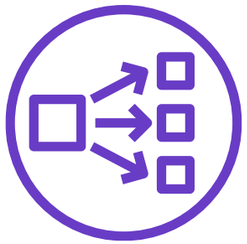
| | Fargate | Managed nodes | Unmanaged nodes |
|---|---|---|---|
| Units of work | Pod | Pod and EC2 | Pod and EC2 |
| Unit of charge | Pod | EC2 | EC2 |
| Host lifecycle | There is no visible host | AWS (SSH is allowed) | Customer |
| Host AMI | There is no visible host | AWS vetted AMIs | Customer 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.
- Documentation Link
- https://docs.aws.amazon.com/eks/latest/userguide/fargate.html

Elastic Load Balancing

Classic Load Balancer



Network Load Balancer

Application Load Balancer

Certificate Manager

Route53

Elastic Block Store

Amazon RDS

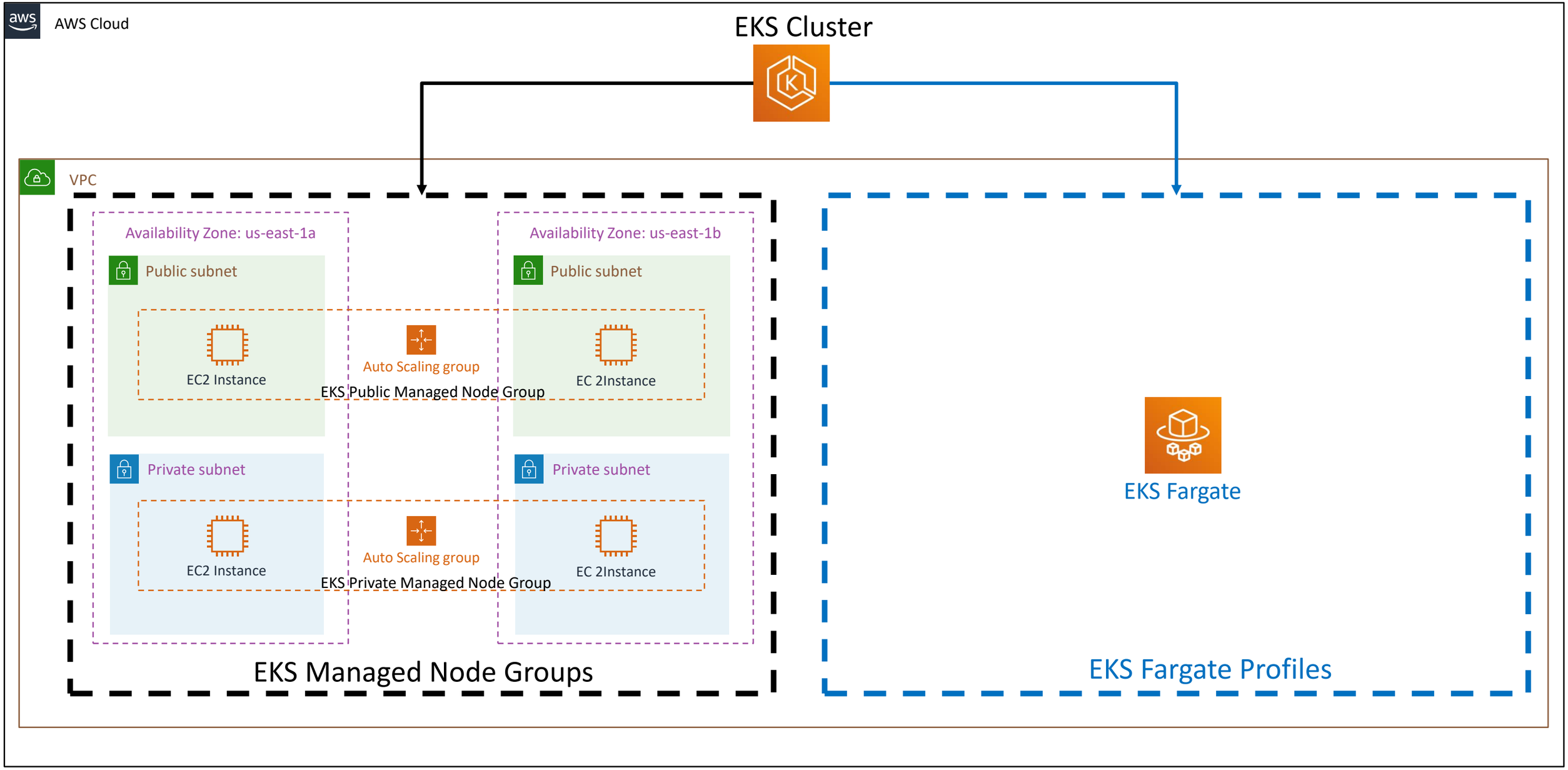
# AWS EKS

## Fargate Profiles

## Basics

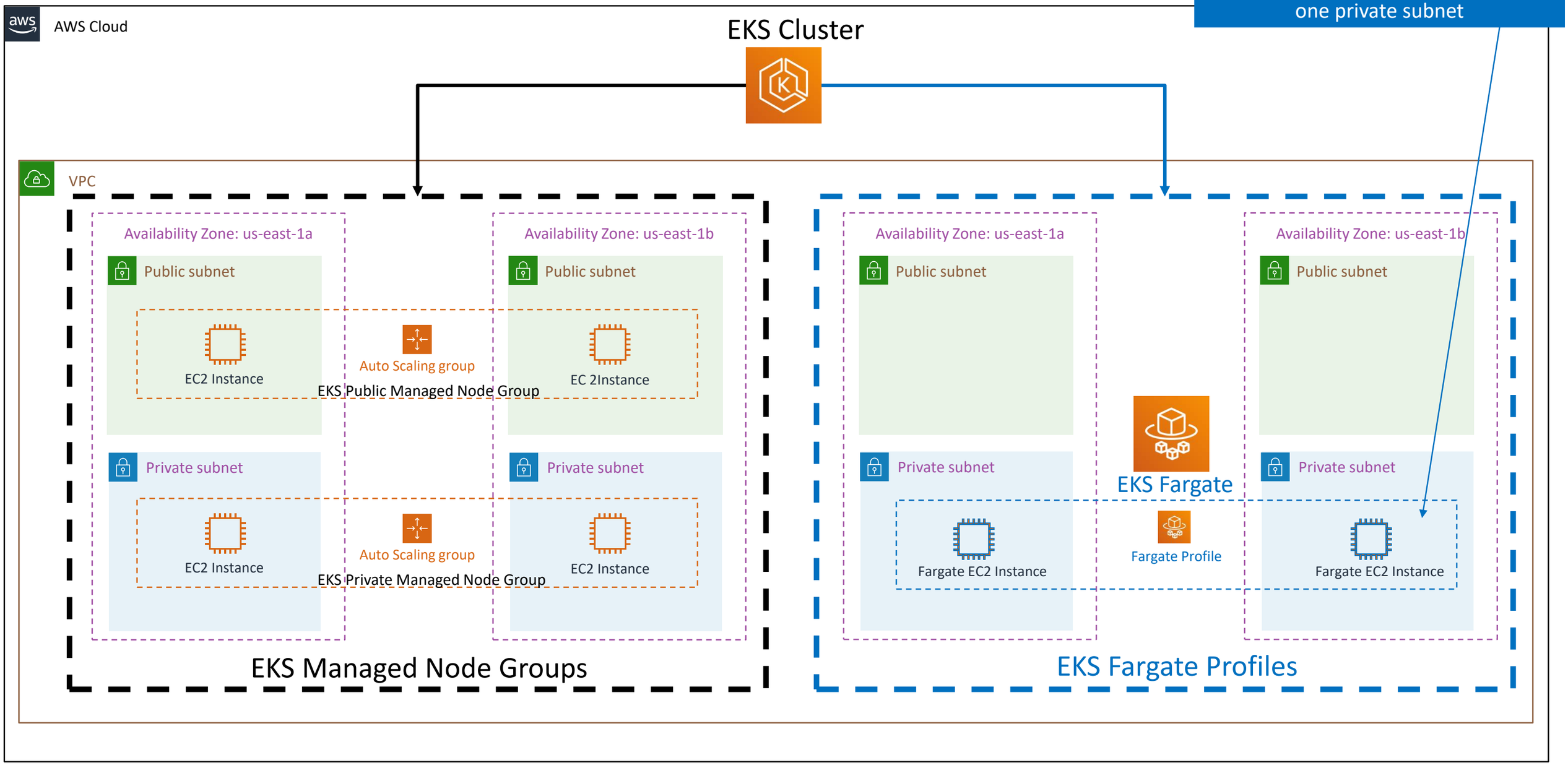
Fargate Profiles

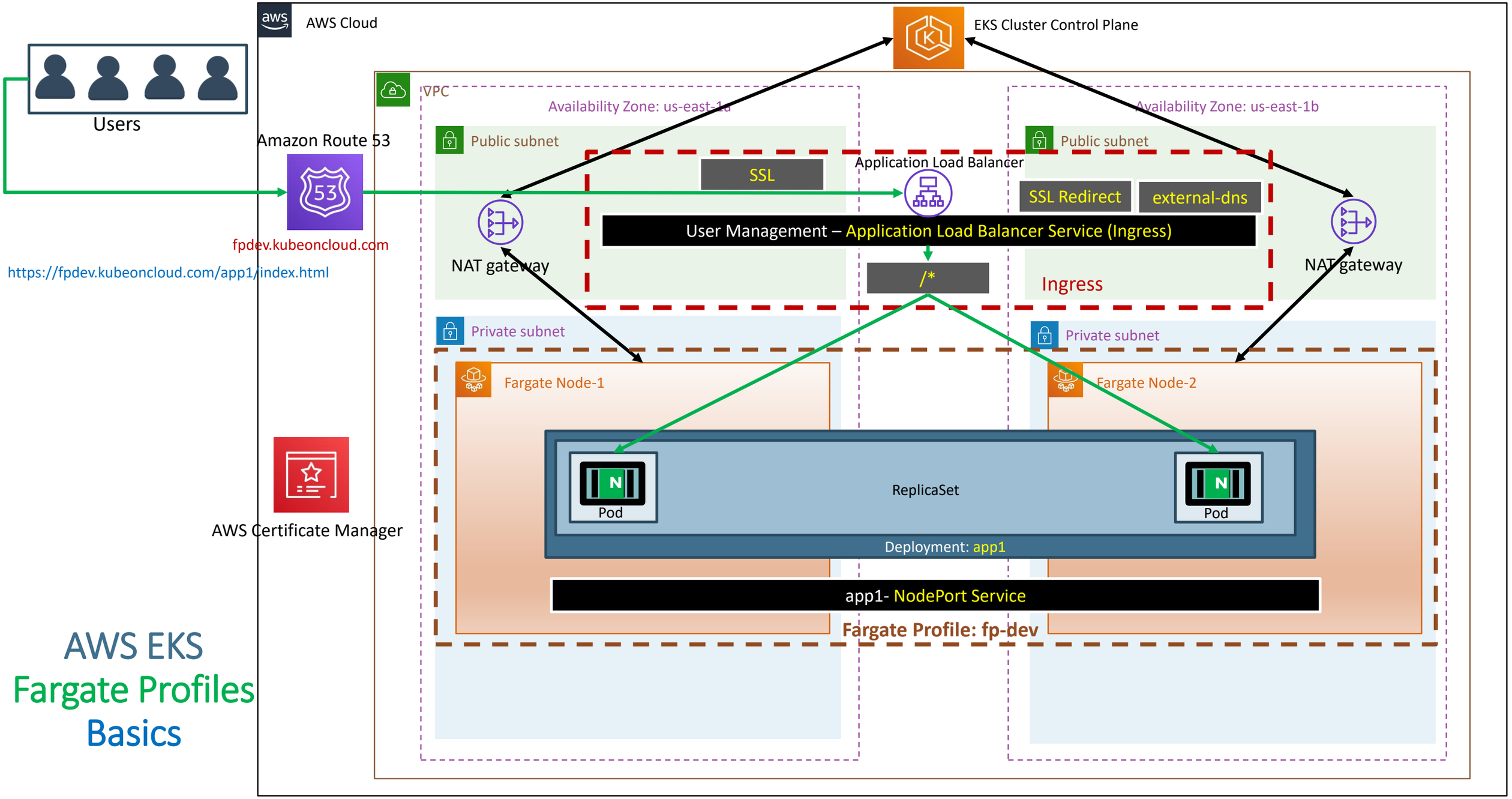
# EKS Deployment Options - Mixed



# EKS Deployment Options - Mixed

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

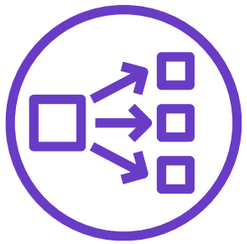





# AWS EKS Fargate Profiles Basics

Elastic Load Balancing

Classic Load Balancer



Network Load Balancer

Application Load Balancer

Certificate Manager

Route53

Elastic Block Store

Amazon RDS

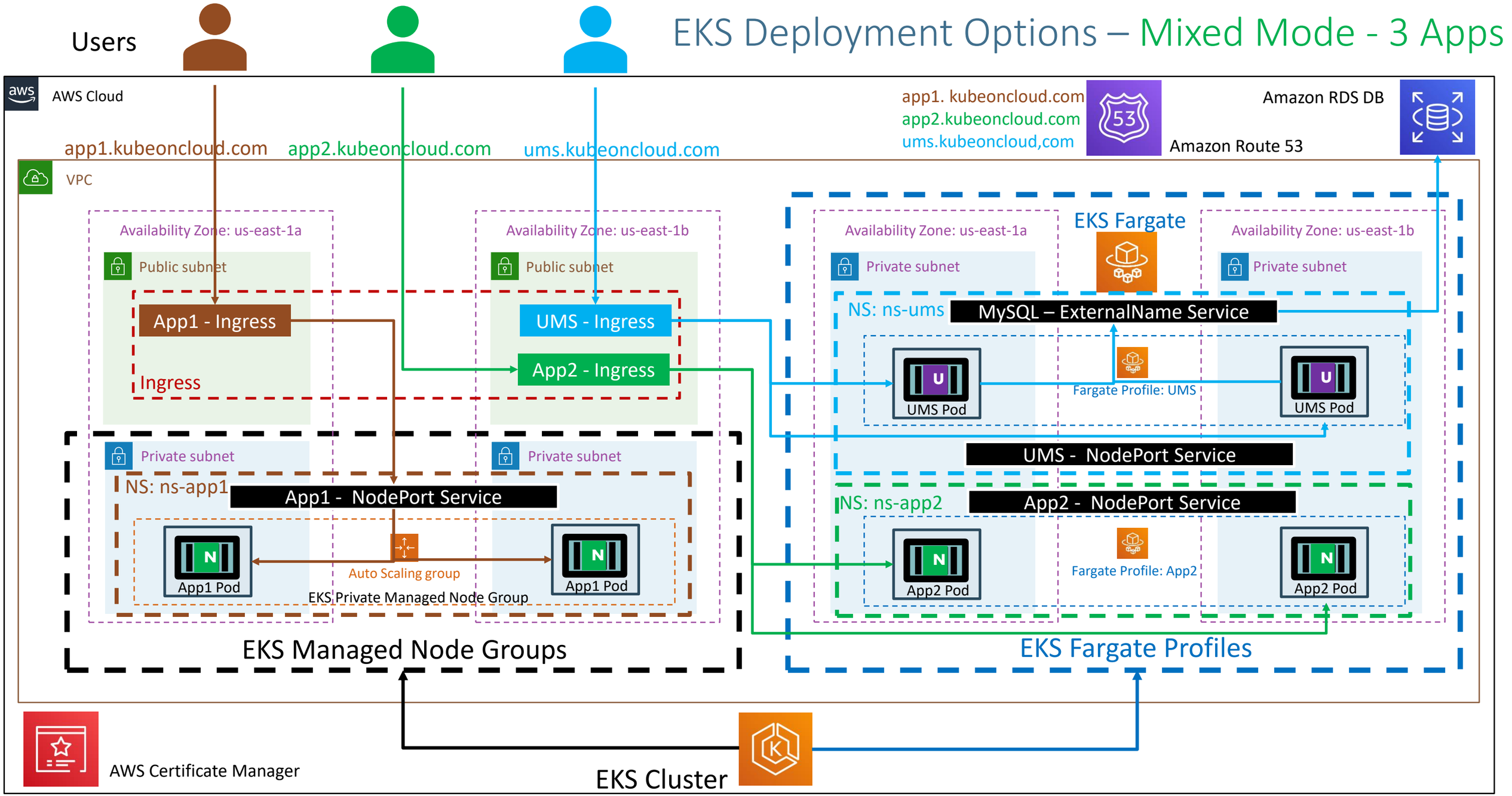
# AWS EKS

## Fargate Profiles

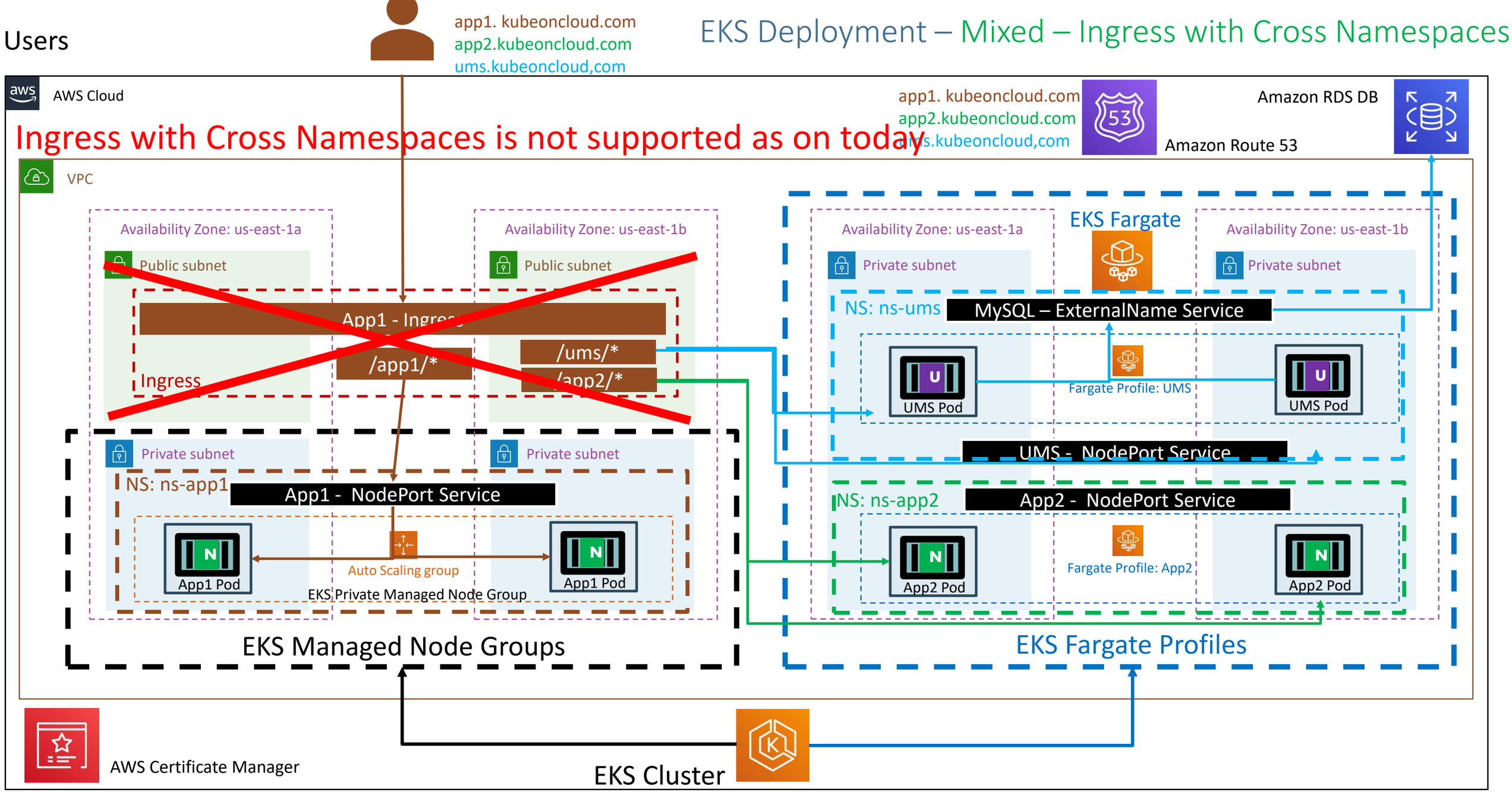
## Advanced with YAML

Fargate Profiles

# EKS Deployment Options – Mixed Mode - 3 Apps

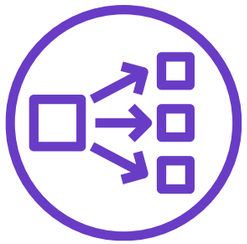


# EKS Deployment – Mixed – Ingress with Cross Namespaces



Elastic Load Balancing

Classic Load Balancer



Network Load Balancer

Application Load Balancer

Certificate Manager

Route53

Elastic Block Store

Amazon RDS

# AWS EKS

## ECR

## Elastic Container Registry

Fargate Profiles

Elastic Container Registry

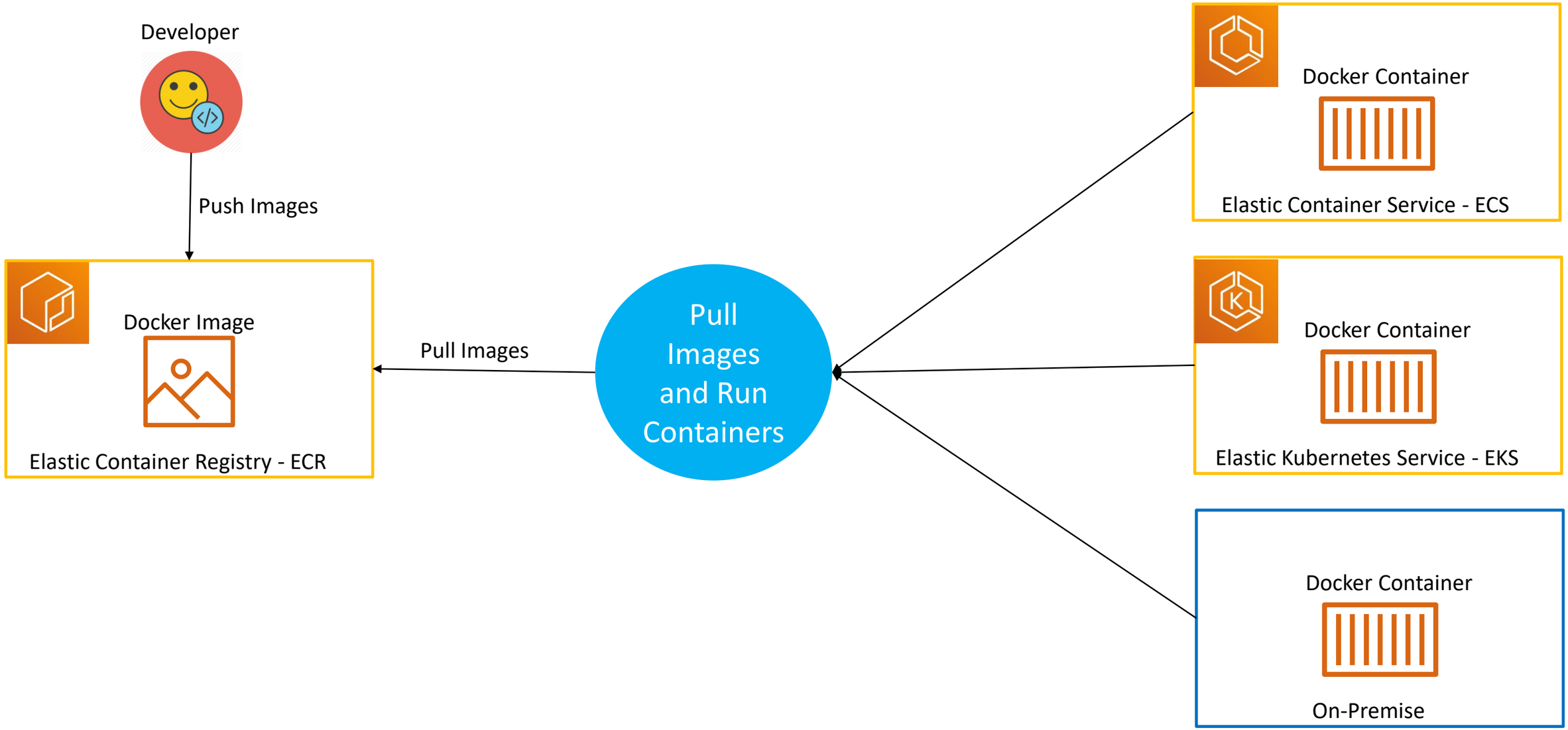
# Elastic Container Registry - ECR

- Elastic Container Registry (ECR) is a fully-managed Docker container registry that makes it easy for developers to store, manage, and deploy Docker container images.
- ECR is integrated with Elastic Kubernetes Service (EKS), simplifying our development to production workflow.
- ECR eliminates the need to operate our own container repositories or worry about scaling the underlying infrastructure.
- ECR hosts our images in a highly available and scalable architecture, allowing us to reliably deploy containers for our applications.
- Integration with AWS Identity and Access Management (IAM) provides resource-level control of each repository.
- With Amazon ECR, there are no upfront fees or commitments. We pay only for the amount of data you store in your repositories and data transferred to the Internet.

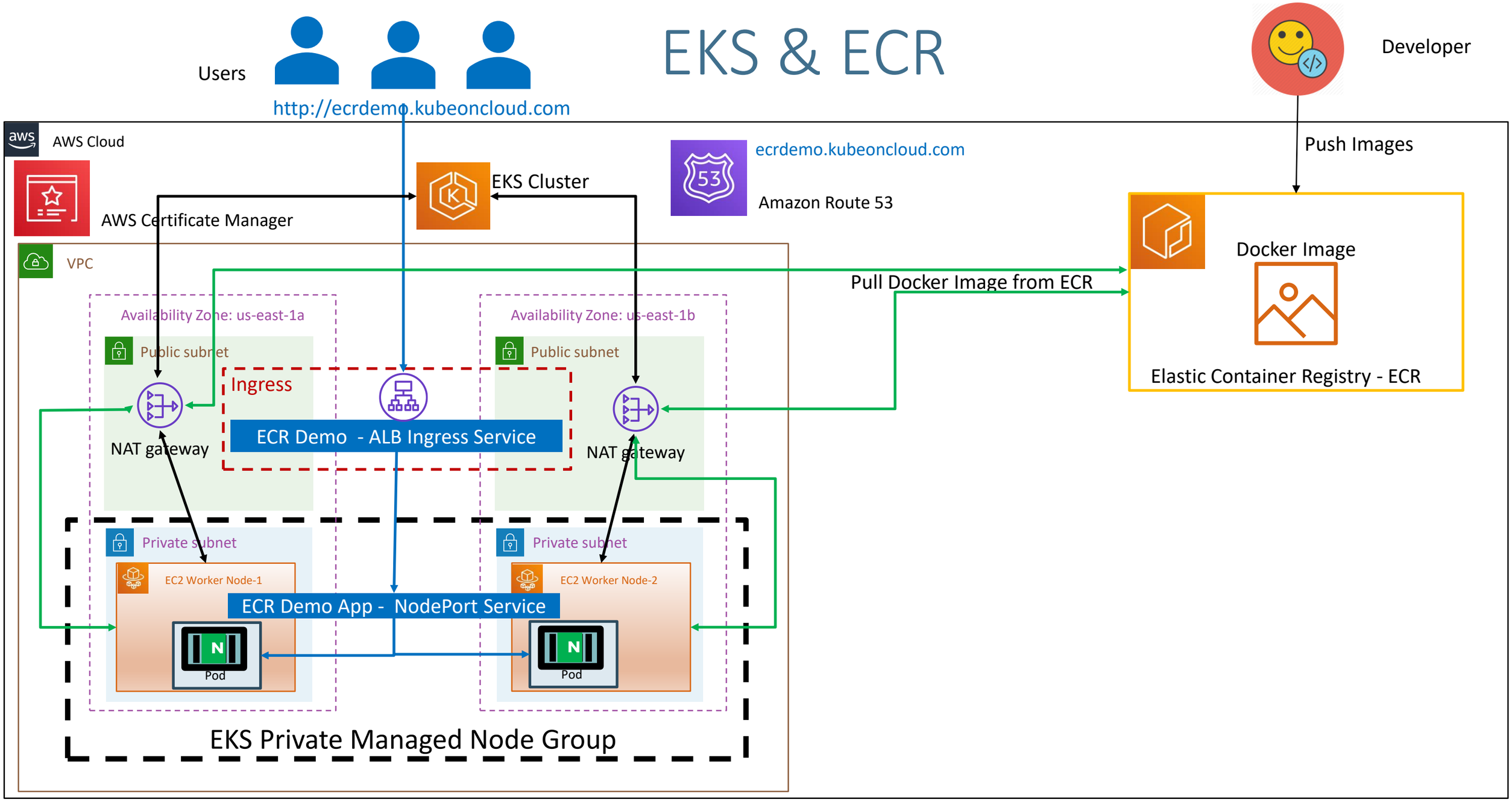
# Elastic Container Registry - ECR

- Benefits
  - Full managed
  - Secure
  - Highly Available
  - Simplified Workflow

# How ECR Works?

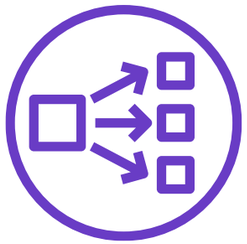


# EKS & ECR



Elastic Load Balancing

Classic Load Balancer



Network Load Balancer

Application Load Balancer

Certificate Manager

Route53

Elastic Block Store

Amazon RDS

# AWS EKS

# &

# AWS Developer Tools

Fargate Profiles

Elastic Container Registry

Code Commit

Code Build

Code Pipeline

# Stages in Release Process

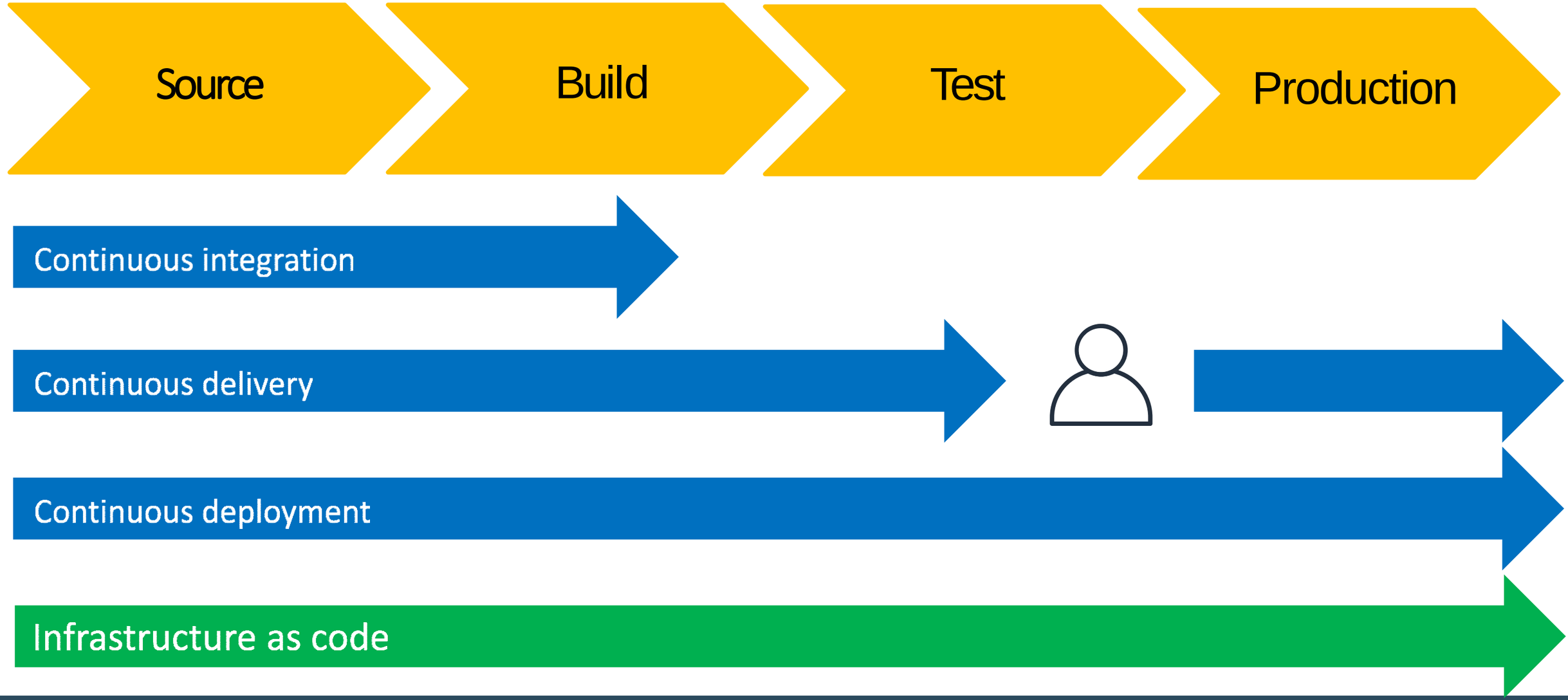

- Check-in source code
- Peer review new code
- Pull Request process

- Compile Code & build artifacts (war ,jar, container images, Kubernetes manifest files)
- Unit Tests

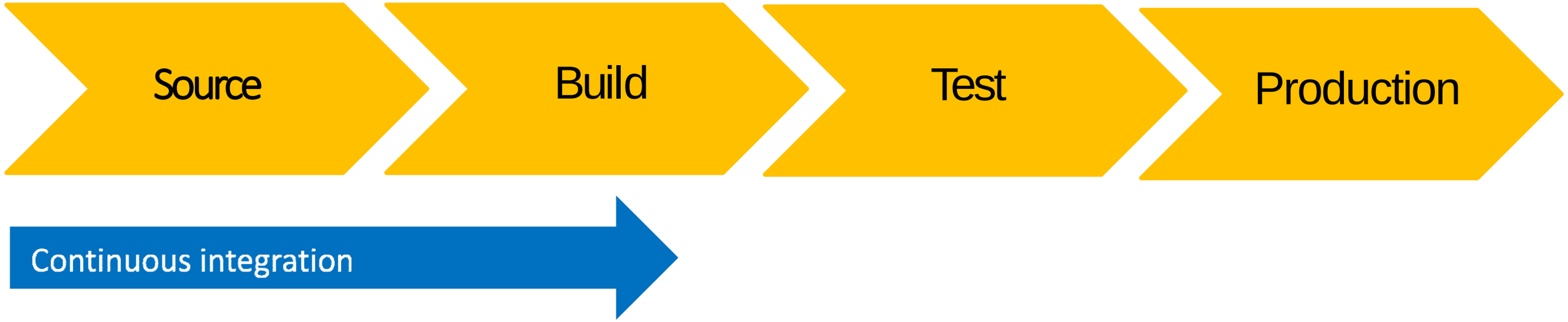
- Integration tests with other systems.
- Load Testing
- UI Tests
- Security Tests
- Test Environments (Dev, QA and Staging)

- Deployment to production environments
- Monitor code in production to quickly detect errors

# Stages in Release Process



# Continuous Integration

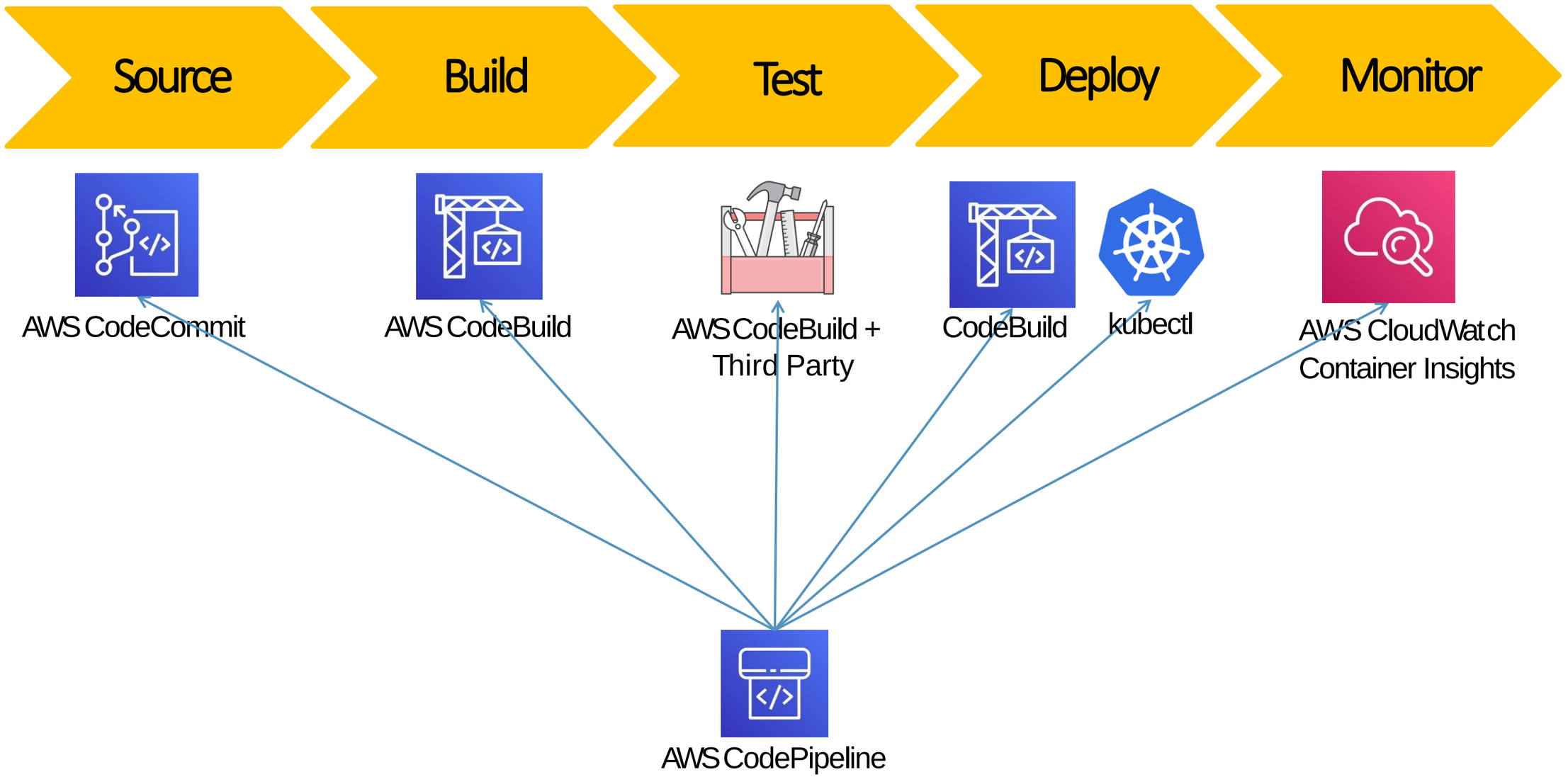



- Automatically kick off a new release when new code is checked-in
- Build and test code in a consistent, repeatable environment
- Continually have an artifact ready for deployment

# Continuous Delivery


- Automatically deploy new changes to staging environments for testing
- Deploy to production safely without affecting customers
- Deliver to customers faster
- Increase deployment frequency, and reduce change lead time and change failure rate

# AWS Developer Tools or Code Services



# AWS Developer Tools or Code Services


AWS CodeCommit

AWS CodeBuild

AWS CodeBuild + Third Party

CodeBuild

kubectl

AWS CloudWatch Container Insights

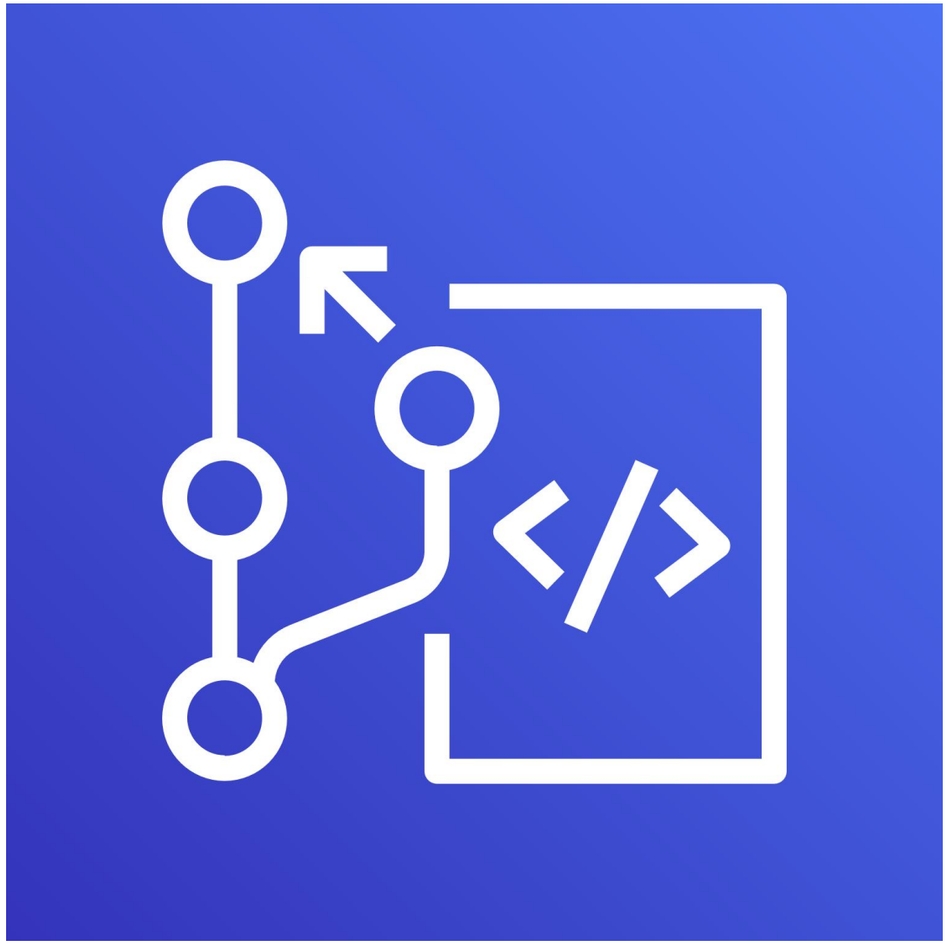
Continuous integration

Continuous delivery

Continuous deployment

AWS CodePipeline

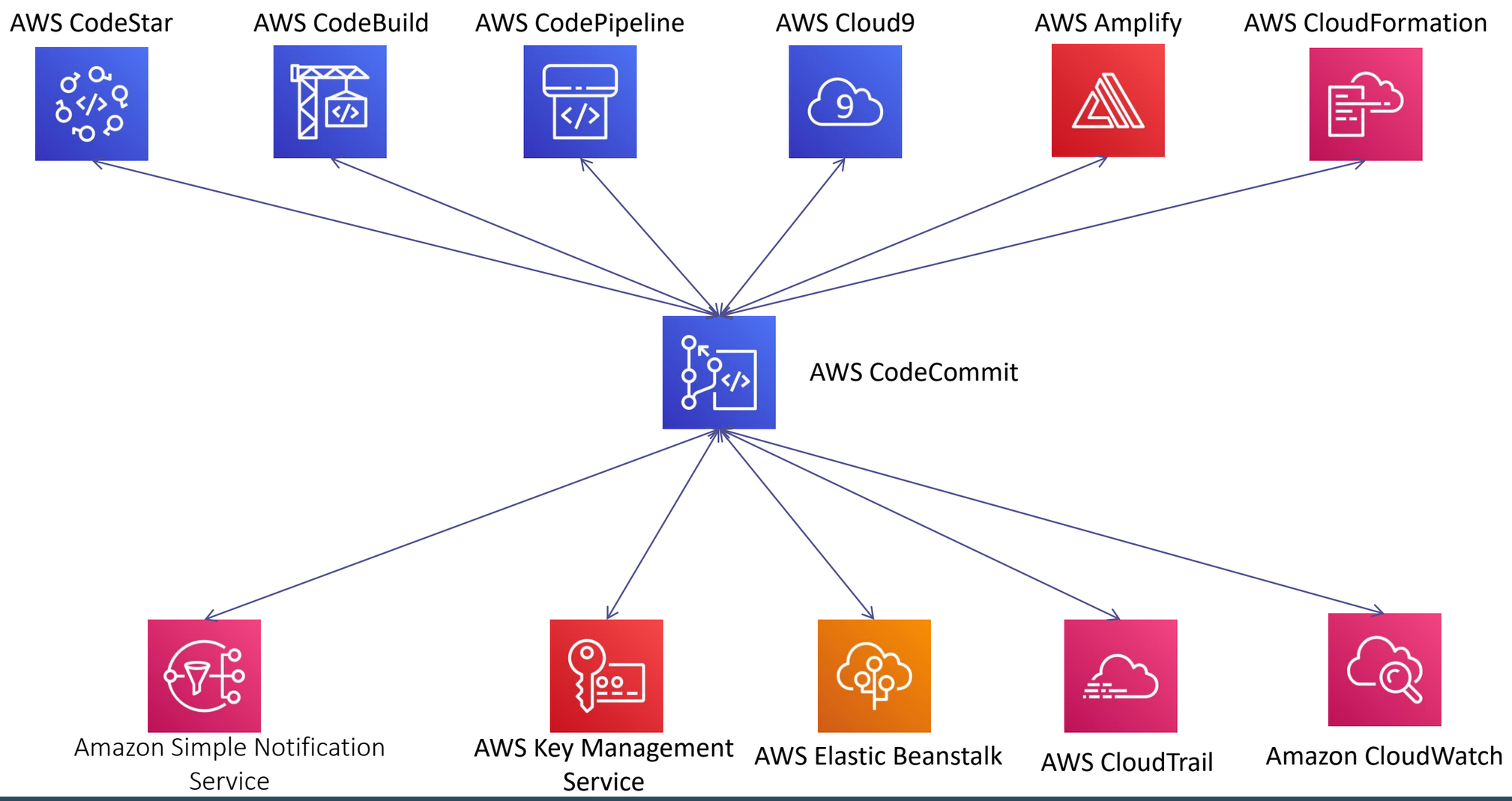
# AWS CodeCommit



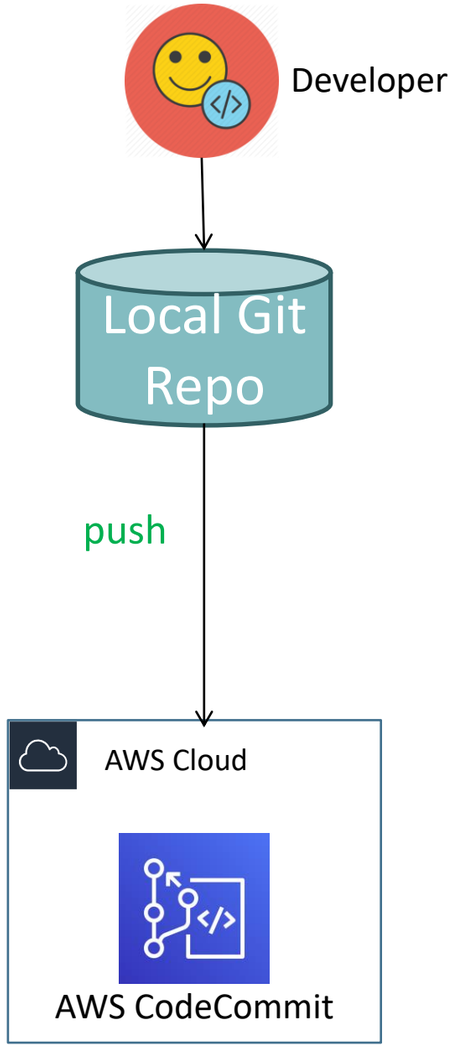
# AWS CodeCommit - Introduction

- Version Control Service hosted by AWS
- We can privately store and manage documents, source code, and binary files
- Secure & highly scalable
- Supports standard functionality of Git (CodeCommit supports Git versions 1.7.9 and later.)
- Uses a static user name and password in addition to standard SSH..

# CodeCommit – Integration with AWS Services



# CodeCommit - Steps

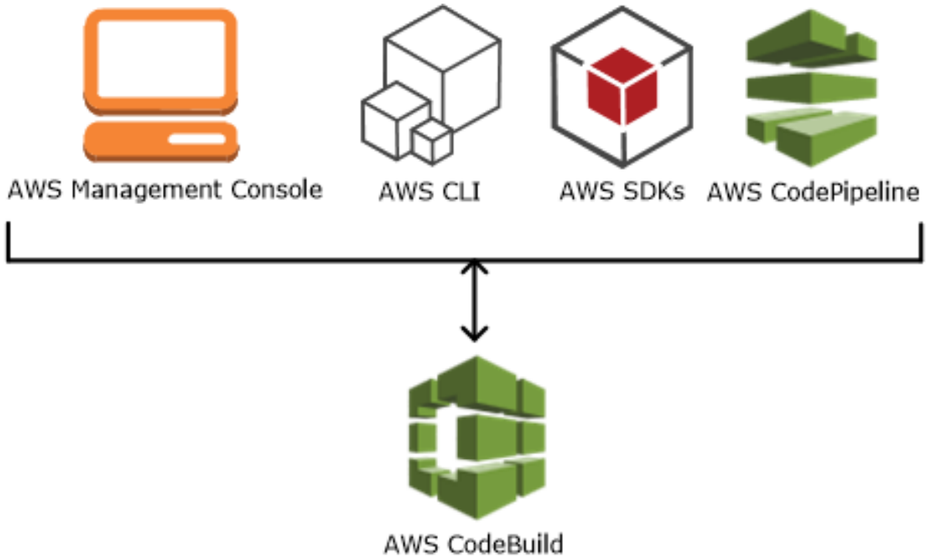


# AWS CodeBuild

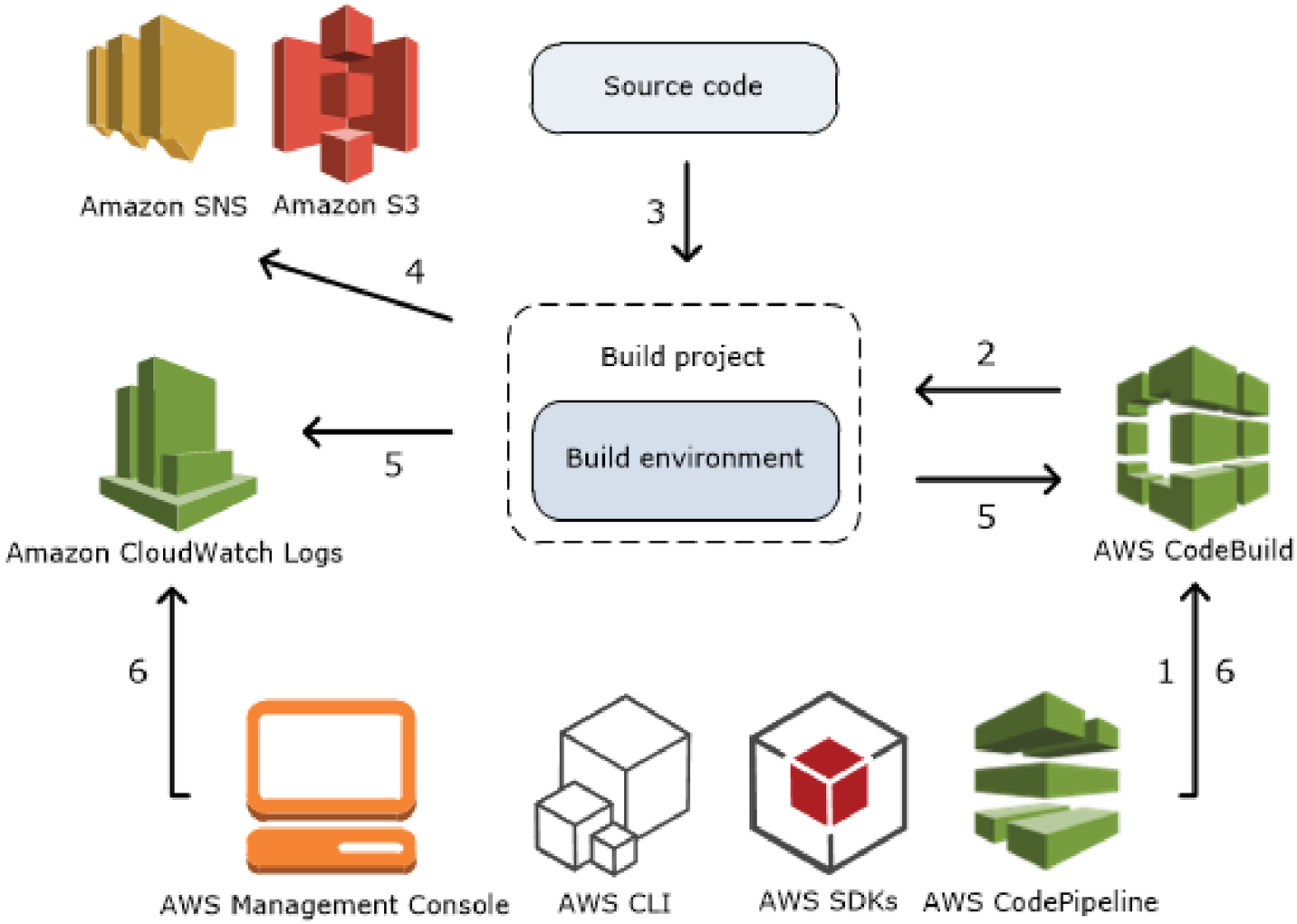
# CodeBuild - Introduction

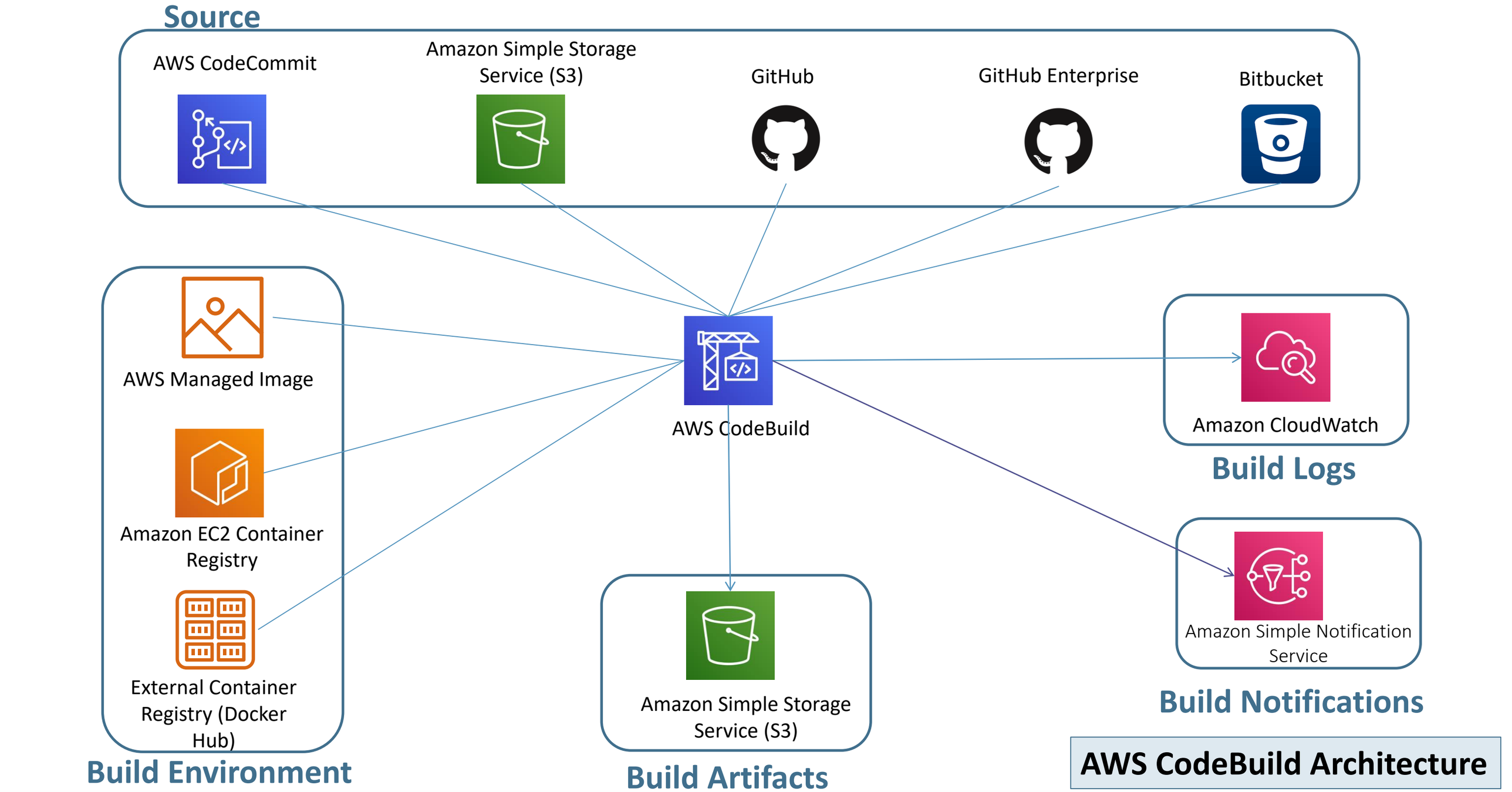
- CodeBuild is a fully managed build service in the cloud.
- Compiles our source code, runs unit tests, and produces artifacts that are ready to deploy.
- Eliminates the need to provision, manage, and scale our own build servers.
- It provides prepackaged build environments for the most popular programming languages and build tools such as Apache Maven, Gradle, and many more.
- We can also customize build environments in CodeBuild to use our own build tools.
- Scales automatically to meet peak build requests.

## How to run CodeBuild?

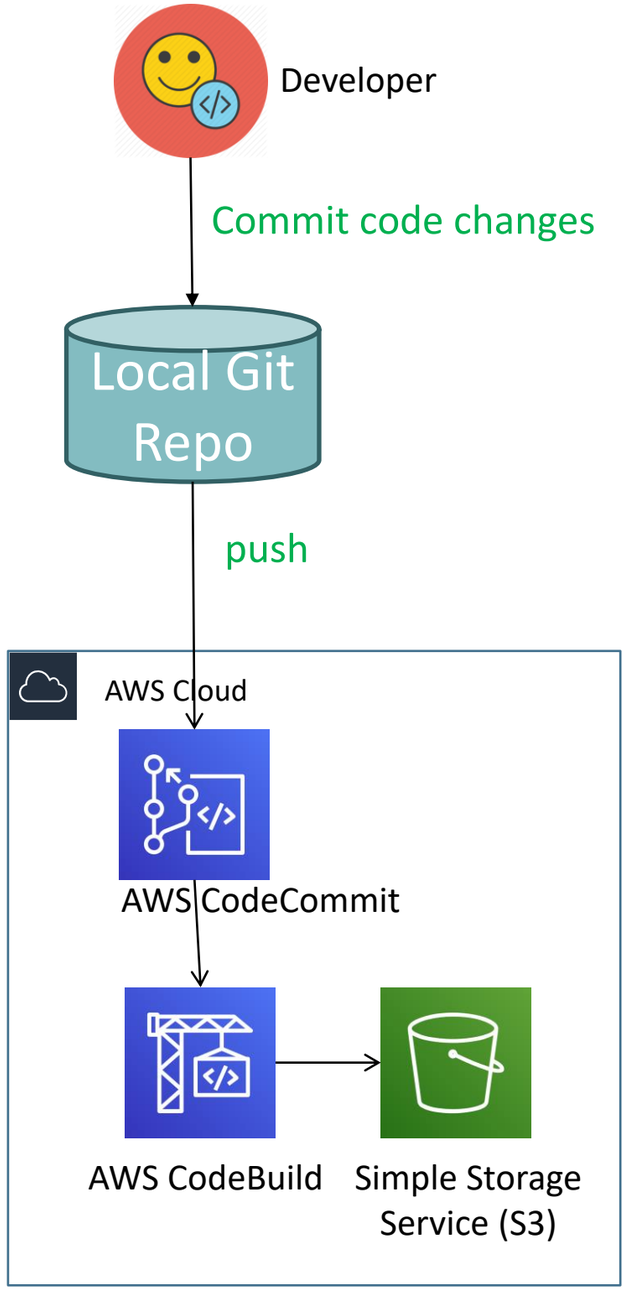



## How CodeBuild works?




Source
AWS CodeCommit
Amazon Simple Storage Service (S3)
GitHub
GitHub Enterprise
Bitbucket
AWS Managed Image
Amazon EC2 Container Registry
External Container Registry (Docker Hub)
Build Environment
AWS CodeBuild
Amazon CloudWatch
Build Logs
Amazon Simple Storage Service (S3)
Build Artifacts
Amazon Simple Notification Service
Build Notifications
AWS CodeBuild Architecture

# CodeBuild - Steps



# AWS CodePipeline

# CodePipeline - Introduction

- AWS CodePipeline is a continuous delivery service to model, visualize, and automate the steps required to release your software.
- Benefits
  - We can automate our release processes.
  - We can establish a consistent release process.
  - We can speed up delivery while improving quality.
  - Supports external tools integration for source, build and deploy.
  - View progress at a glance
  - View pipeline history details.

# AWS CodePipeline Architecture

AWS CodePipeline

AWS CodeCommit

Amazon EC2 Container Registry

Simple Storage Service (S3)

GitHub

**Source**

AWS CodeBuild

Jenkins

**Build**

AWS CloudFormation

AWS CodeDeploy

AWS Elastic Beanstalk

AWS Service Catalog

Amazon Elastic Container Service

Amazon Elastic Container Service (Blue/Green)

Simple Storage Service (S3)

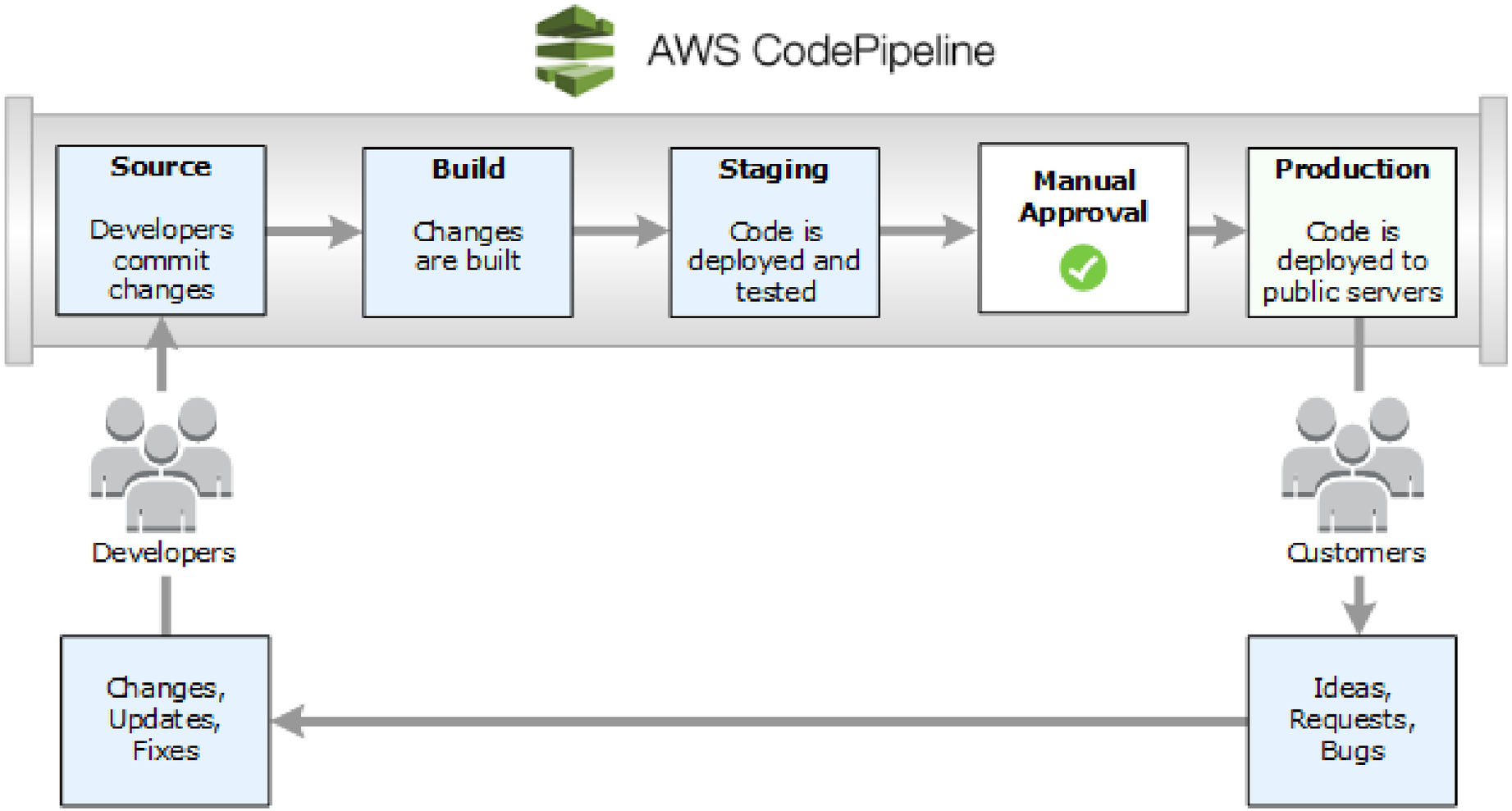
**Deploy**

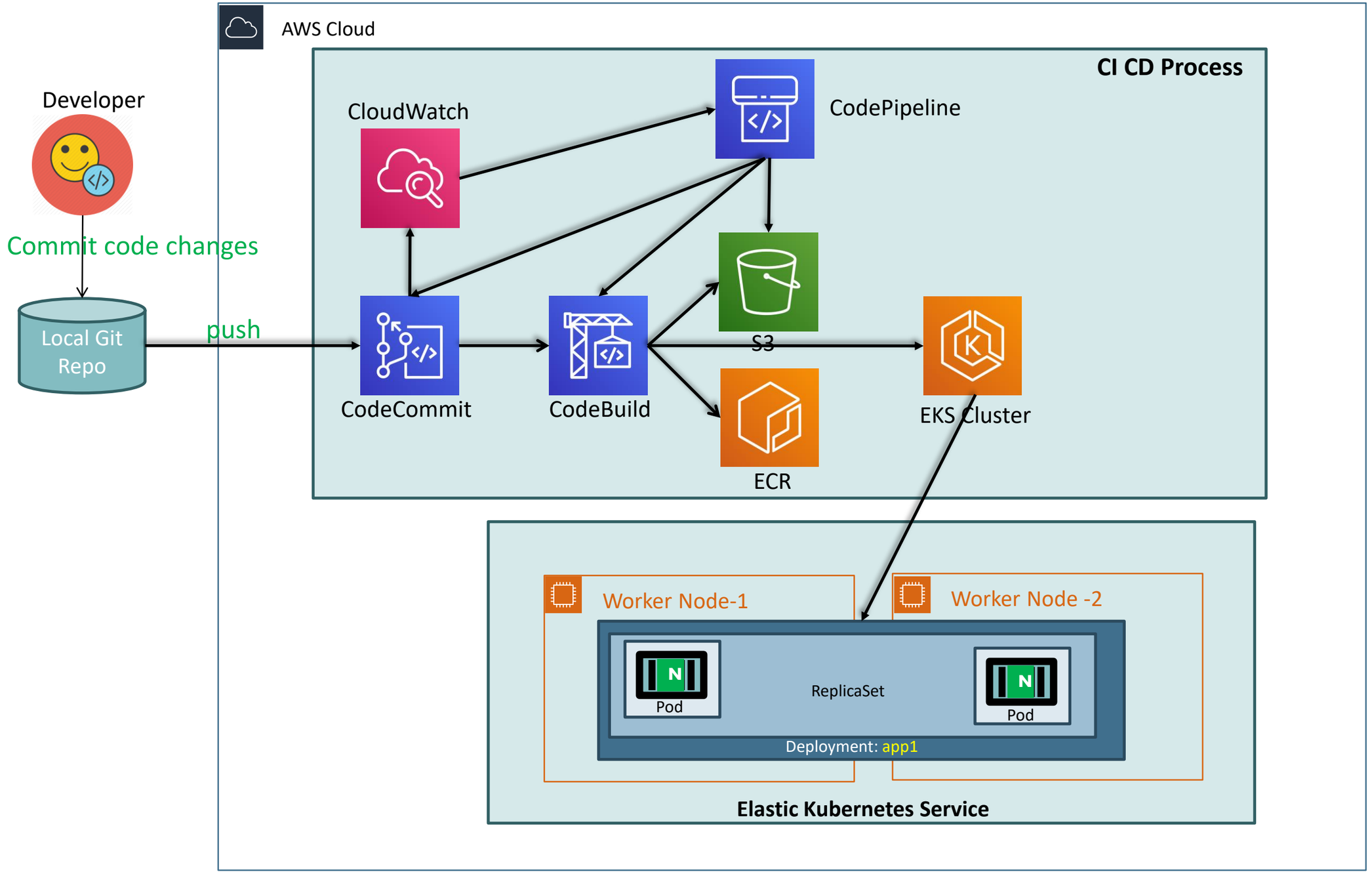
Amazon CloudWatch

GitHub Webhooks

**Monitor Source Changes**

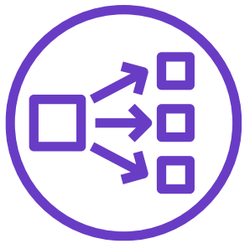
# Continuous Delivery




AWS Cloud
CI CD Process
Developer
Commit code changes
Local Git Repo
push
CloudWatch
CodePipeline
S3
CodeCommit
CodeBuild
ECR
EKS Cluster
Worker Node-1
Worker Node -2
Pod
ReplicaSet
Pod
Deployment: app1
Elastic Kubernetes Service

Elastic Load Balancing

Classic Load Balancer



Network Load Balancer

Application Load Balancer

Certificate Manager


Route53

Elastic Block Store

Amazon RDS

# AWS EKS

## What are Microservices?

Fargate Profiles

Elastic Container Registry

Code Commit

Code Build

Code Pipeline

Simple Email Service

# What are Microservices?

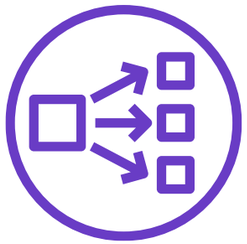
- Microservices - also known as the microservice architecture - is an architectural style that structures an application as a collection of services that are
  - Highly maintainable and testable
  - Loosely coupled
  - Independently deployable
  - Organized around business capabilities
  - Owned by a small team

# Microservices - Benefits

- **Developer independence**: Small teams work in parallel and can iterate faster than large teams.
- **Isolation and resilience**: If a component dies, you spin up another while and the rest of the application continues to function.
- **Scalability**: Smaller components take up fewer resources and can be scaled to meet increasing demand of that component only.
- **Lifecycle automation**: Individual components are easier to fit into continuous delivery pipelines and complex deployment scenarios not possible with monoliths.
- **Relationship to the business**: Microservice architectures are split along business domain boundaries, increasing independence and understanding across the organization.

Elastic Load Balancing

Classic Load Balancer



Network Load Balancer

Application Load Balancer

Certificate Manager


Route53

Elastic Block Store

Amazon RDS

# AWS EKS

## Microservices Deployment

Fargate Profiles

Elastic Container Registry

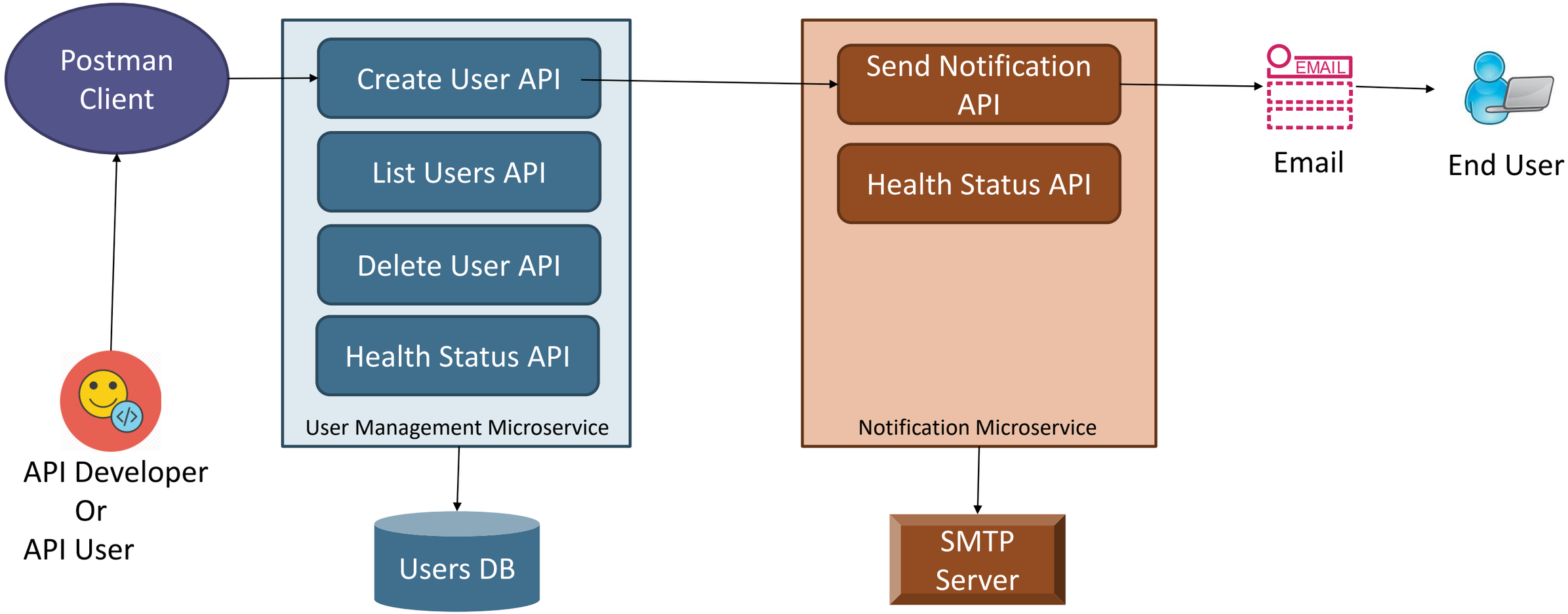
Code Commit

Code Build

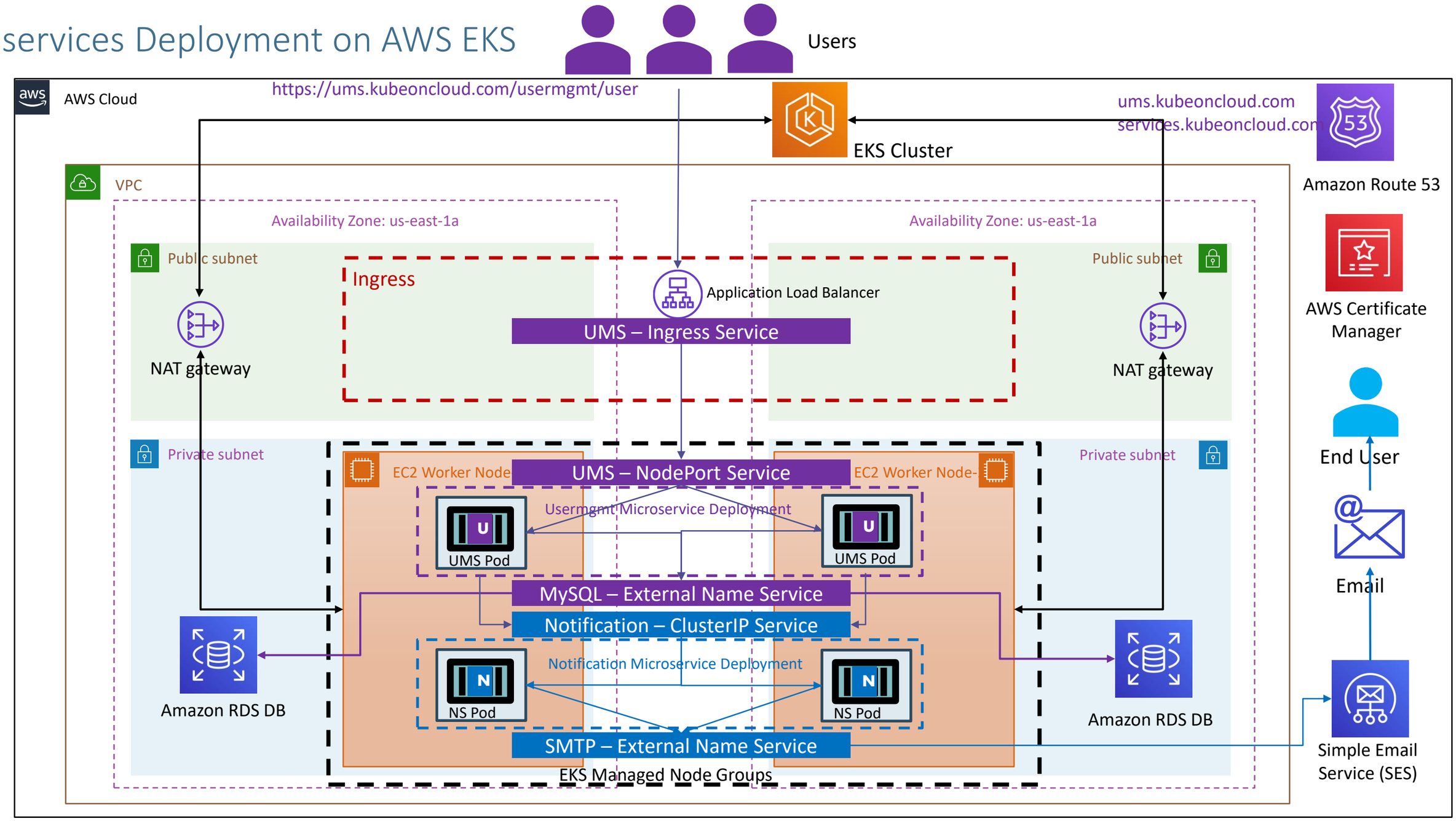
Code Pipeline

Simple Email Service

# Microservices

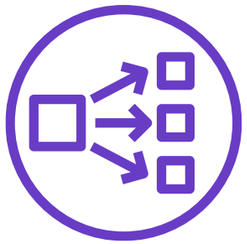


# Microservices Deployment on AWS EKS



Elastic Load Balancing

Classic Load Balancer



Network Load Balancer

Application Load Balancer

Certificate Manager


Route53

Elastic Block Store

Amazon RDS

# AWS EKS

## Microservices Distributed Tracing

## AWS X-Ray

Fargate Profiles

Elastic Container Registry

Code Commit

Code Build

Code Pipeline

Simple Email Service

X-Ray

# AWS X-Ray Introduction

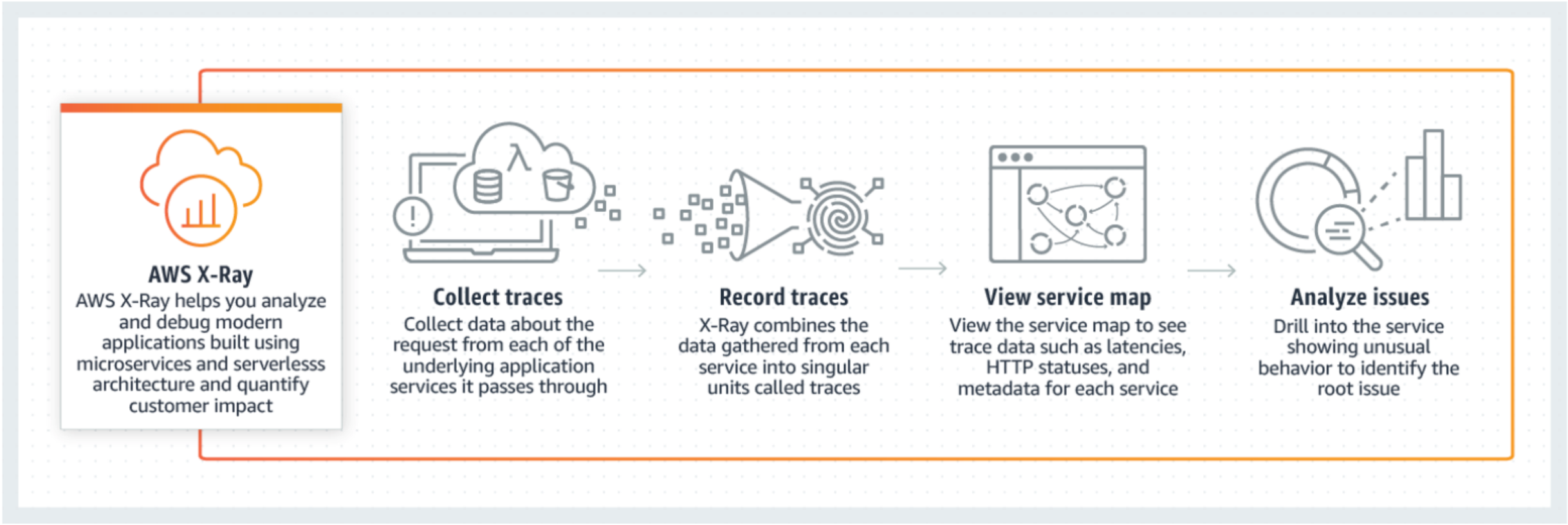
- AWS X-Ray helps analyse and debug distributed applications built using microservices architecture.
- With X-Ray, we can understand how our application and its underlying services are performing to identify and troubleshoot the root cause of performance issues and errors.
- X-Ray provides an end-to-end view of requests as they travel through our application and shows a map of our application's underlying components.
- We can also use X-Ray to analyse applications in development and in production, from simple three-tier applications to complex microservices applications consisting of thousands of services.

# AWS X-Ray - Benefits

- Review request behavior
- Discover application issues
- Improve application performance
- Ready to use with AWS
- Designed for a variety of applications

# AWS X-Ray – How it works?

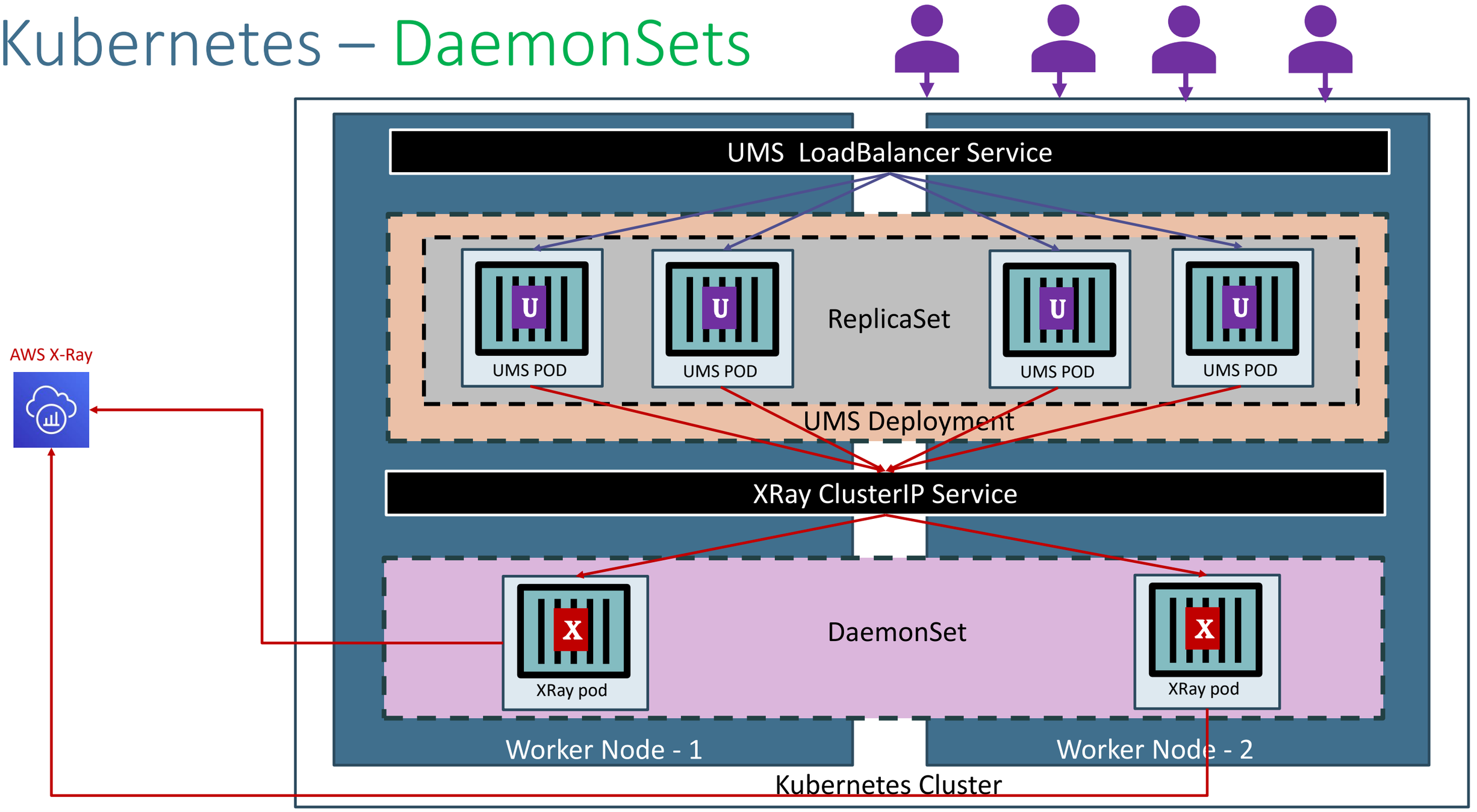
## How it works



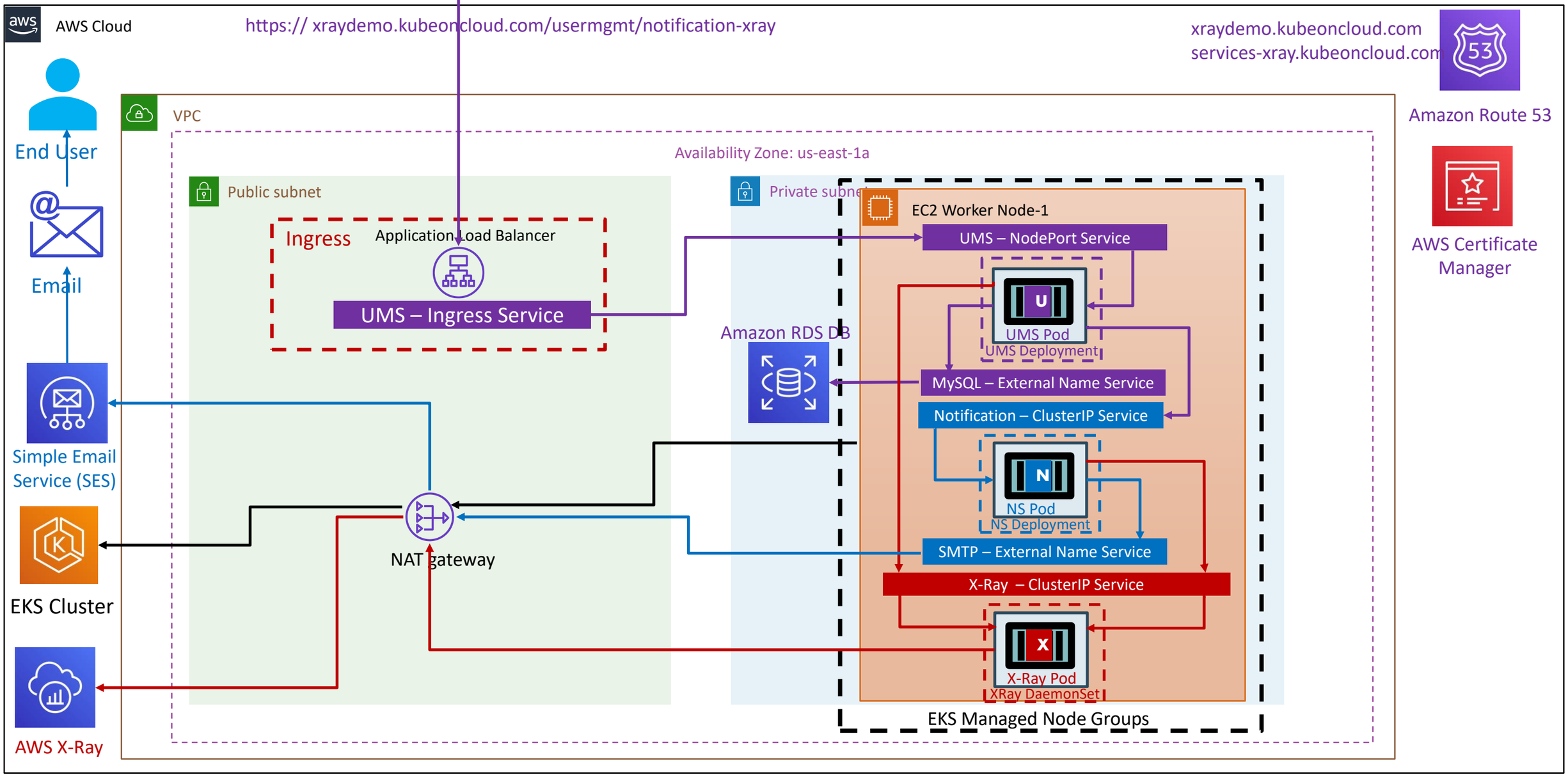
# Kubernetes DaemonSets - Introduction

- A *DaemonSet* ensures that all (or some) Nodes run a copy of a Pod.
  - As nodes are added to the cluster, Pods are added to them.
  - As nodes are removed from the cluster, those Pods are garbage collected.
  - Deleting a DaemonSet will clean up the Pods it created.
- Some typical uses of a DaemonSet are:
  - running a logs collection daemon on every node (Example: fluentd)
  - running a node monitoring daemon on every node (Example: cloudwatchagent)
  - running an application trace collection daemon on every node (Example: AWS X-Ray)
- In a simple case, one DaemonSet, covering all nodes, would be used for each type of daemon.
- A more complex setup might use multiple DaemonSets for a single type of daemon, but with different flags and/or different memory and cpu requests for different hardware types

# Kubernetes – DaemonSets

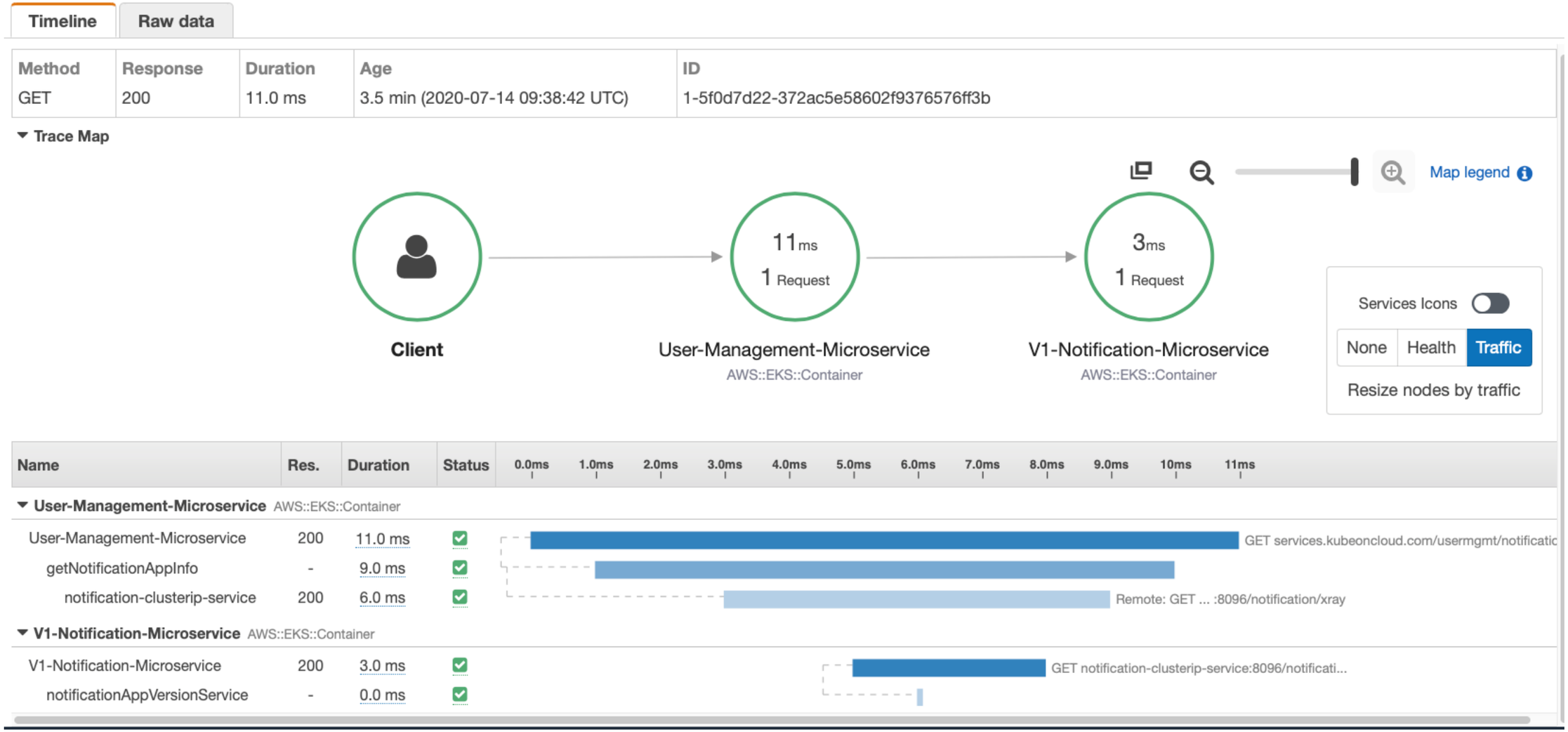


# Microservices Distributed Tracing with AWS X-Ray



# AWS X-Ray – Service Map

# AWS X-Ray - Traces



# AWS X-Ray - Traces

## Segment - User-Management-Microservice

**Overview** | Resources | Annotations | Metadata | Exceptions

| | |
|---|---|
| Segment ID | 722d02f36eb066c4 |
| Parent ID | |
| Name | User-Management-Microservice |
| Origin | AWS::EKS::Container |

**Time**

| | |
|---|---|
| Start time | 2020-07-14 09:38:42.647 (UTC) |
| End time | 2020-07-14 09:38:42.658 (UTC) |
| Duration | 11.0 ms |
| In progress | false |

**Errors & Faults**

| | |
|---|---|
| Error | false |
| Fault | false |

**Request & Response**

| | |
|---|---|
| Request url | http://services.kubeoncloud.com/usermgmt/notification-xray |
| Request method | GET |
| Request user_agent | PostmanRuntime/7.25.0 |
| Request client_ip | 49.206.222.64 |
| Request x_forwarded_for | true |

## Segment - User-Management-Microservice

Overview | **Resources** | Annotations | Metadata | Exceptions

**CloudWatch Logs**

```
[
  {
    "log_group": "/aws/containerinsights//application"
  }
]
```

**EC2**

| | |
|---|---|
| Availability zone | us-east-1b |
| Instance ID | i-0247e0ae1264716b3 |
| Instance size | t3.medium |
| Ami ID | ami-01b33458af4b31b8f |

**X-Ray**

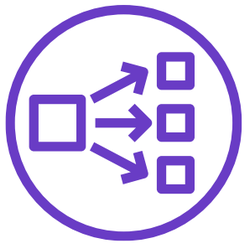
| | |
|---|---|
| SDK version | 2.6.1 |
| SDK | X-Ray for Java |

**Eks**

| | |
|---|---|
| Pod | usermgmt-microservice-7598cb7fcf-txwck |
| Containerid | 1aeabf8732ab8303eb24a138b31f963566d5d113dd571ea965f5b7ad2dd9e34a |

Elastic Load Balancing

Classic Load Balancer



Network Load Balancer

Application Load Balancer

Certificate Manager


Route53

Elastic Block Store

Amazon RDS

# AWS EKS

## Microservices

## Canary Deployments

Fargate Profiles

Elastic Container Registry

Code Commit

Code Build

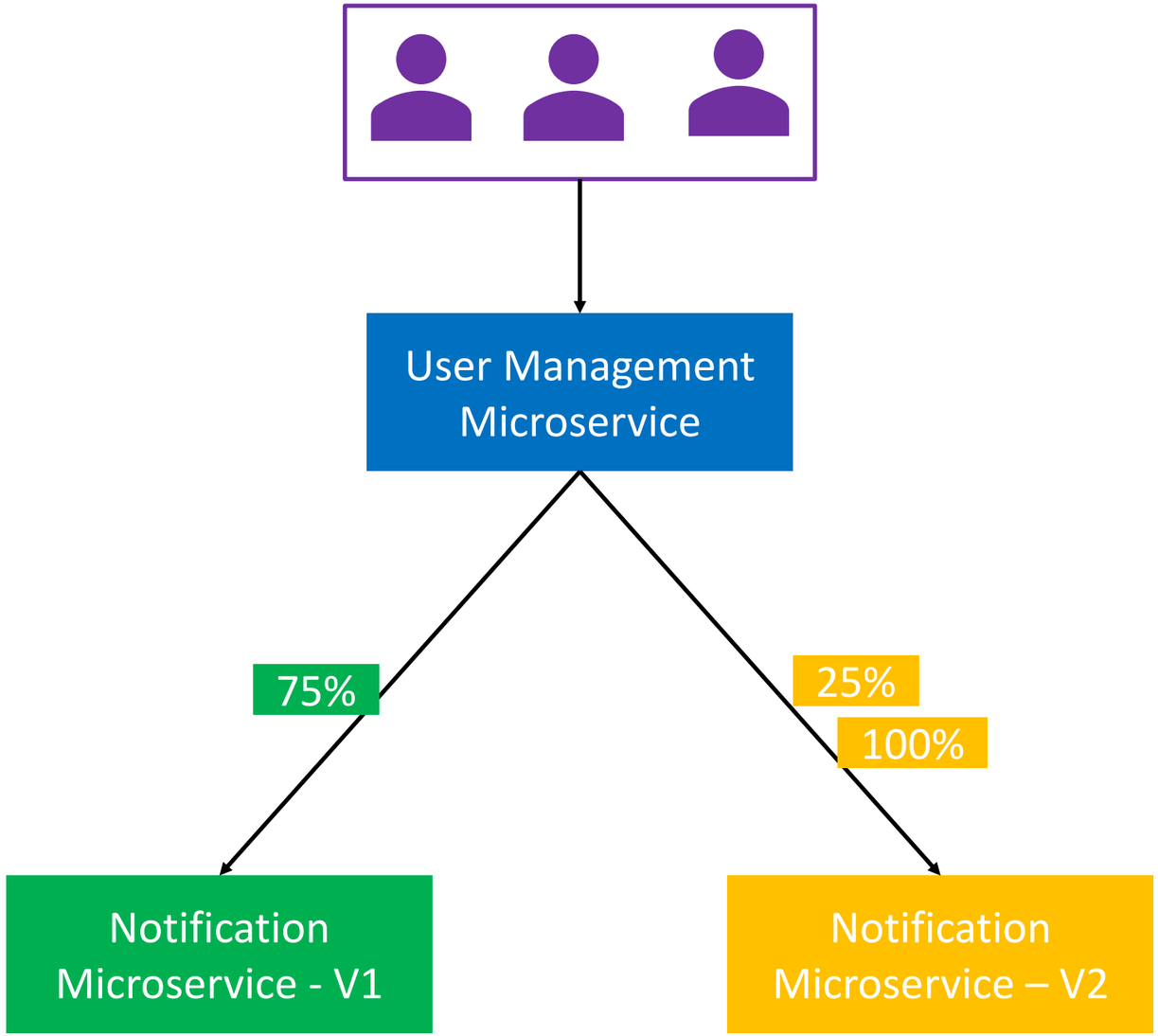
Code Pipeline

Simple Email Service

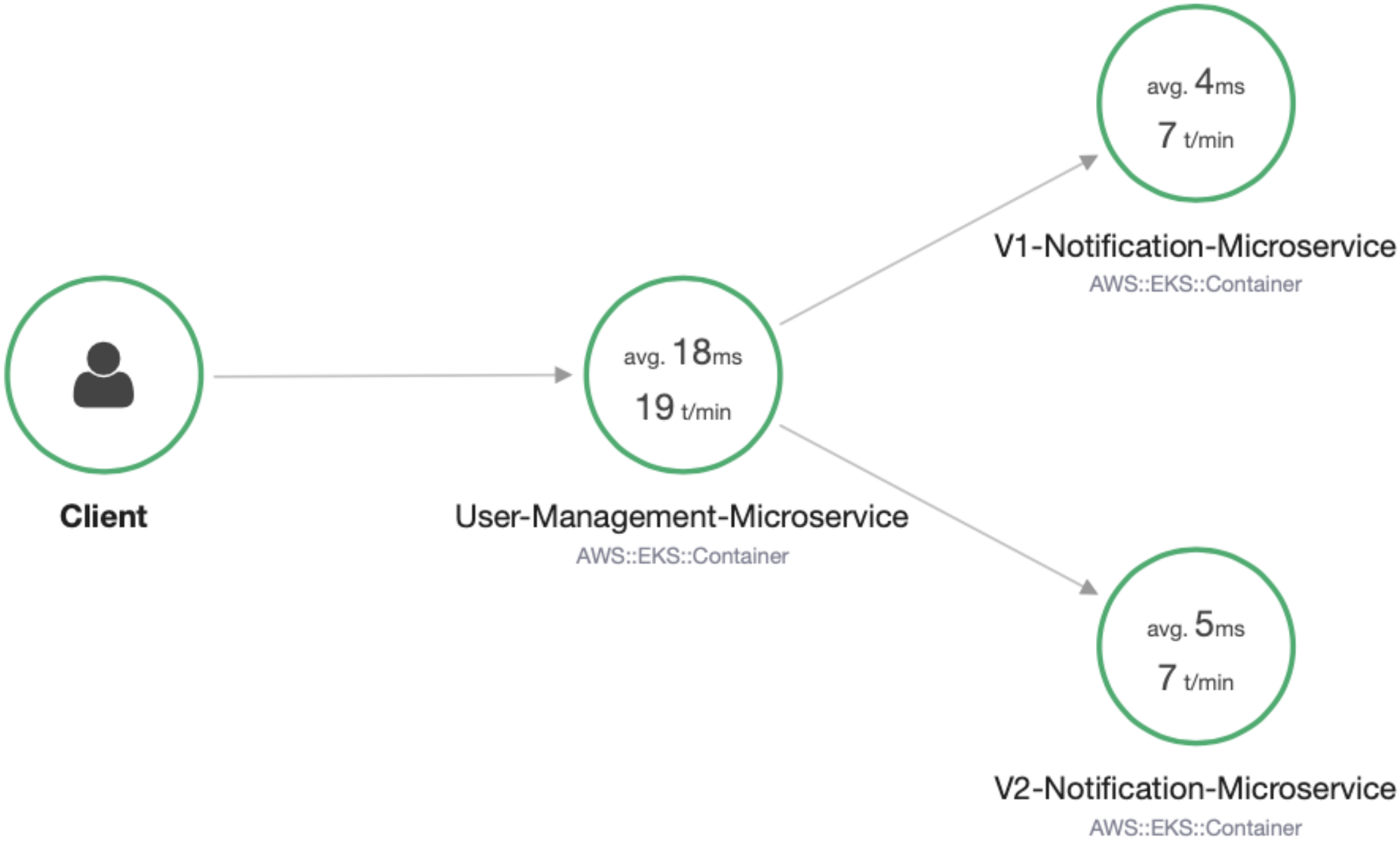
X-Ray

# What are Canary Deployments?

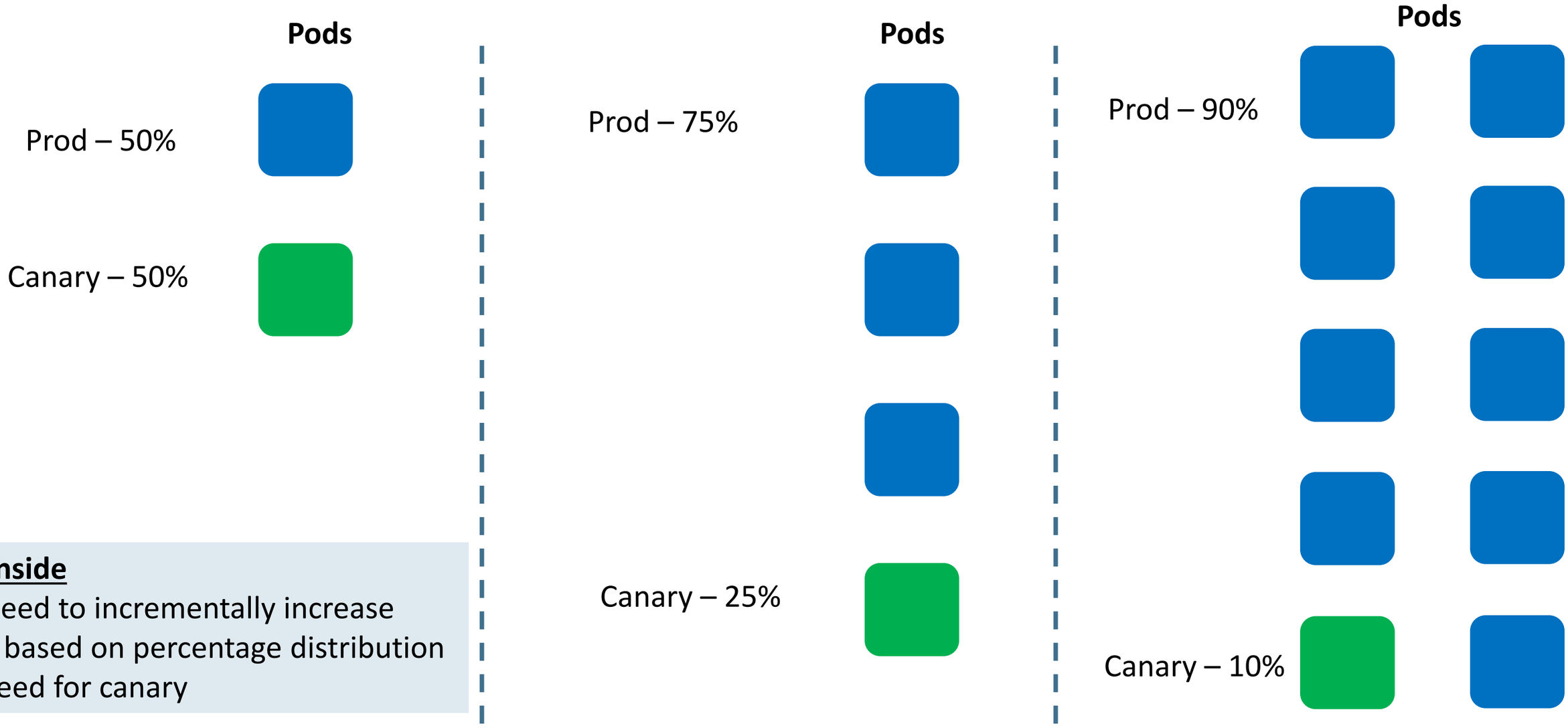
- Canaries means incremental rollouts
- With canaries, the new version of the application is slowly deployed to the Kubernetes cluster while getting a very small amount of live traffic
- In short, a subset of live users are connecting to the new version while the rest are still using the previous version
- Using canaries, we can detect deployment issues very early while they effect only a small subset of users
- If we encounter any issues with a canary, the production version is still present, and all traffic can simply be reverted to it.



# Microservices – Canary Deployments



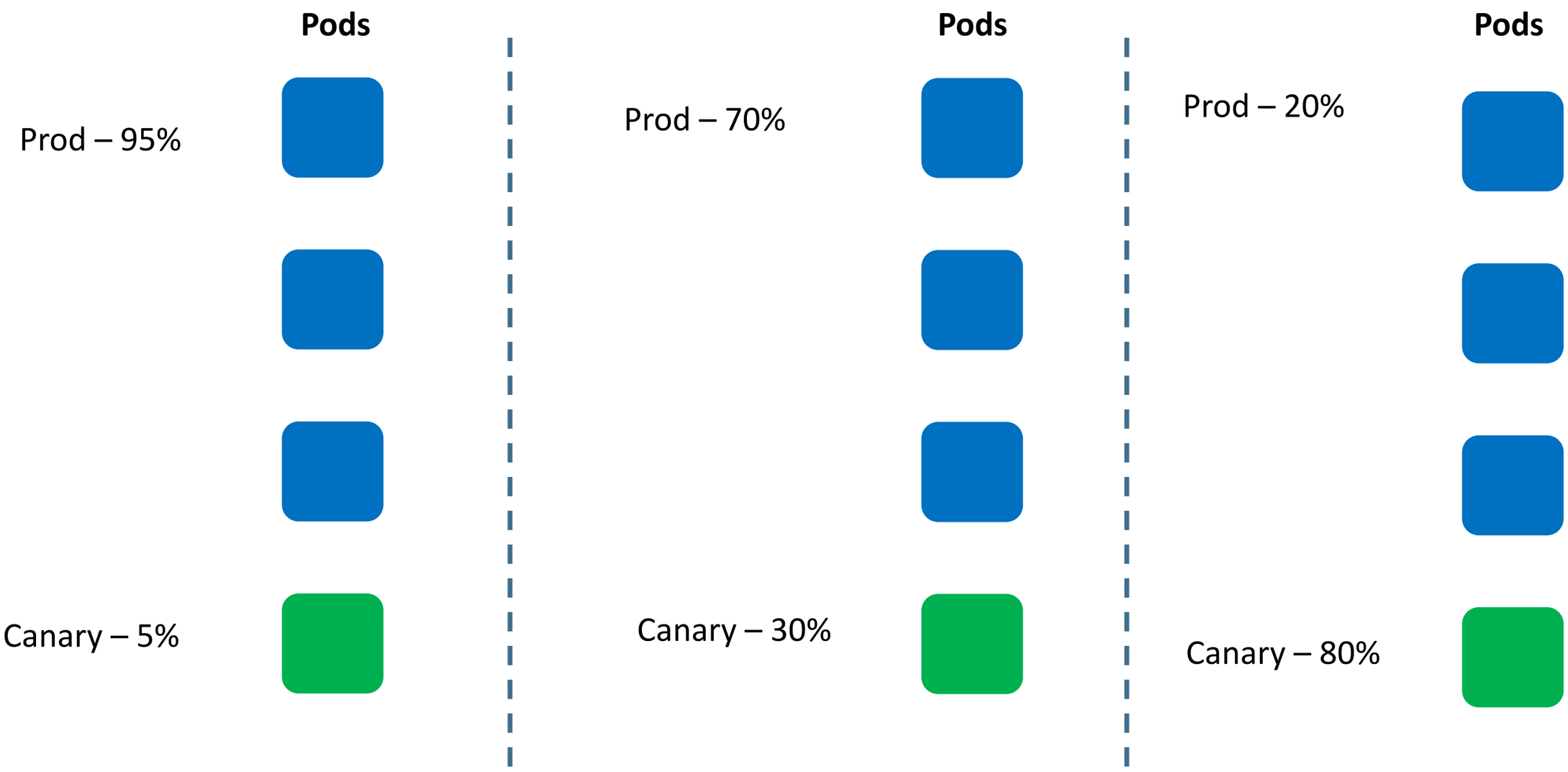
# Canary Deployments out of box on Kubernetes




**Downside**

We need to incrementally increase pods based on percentage distribution we need for canary

# Canary Deployments on Kubernetes with AWS App Mesh

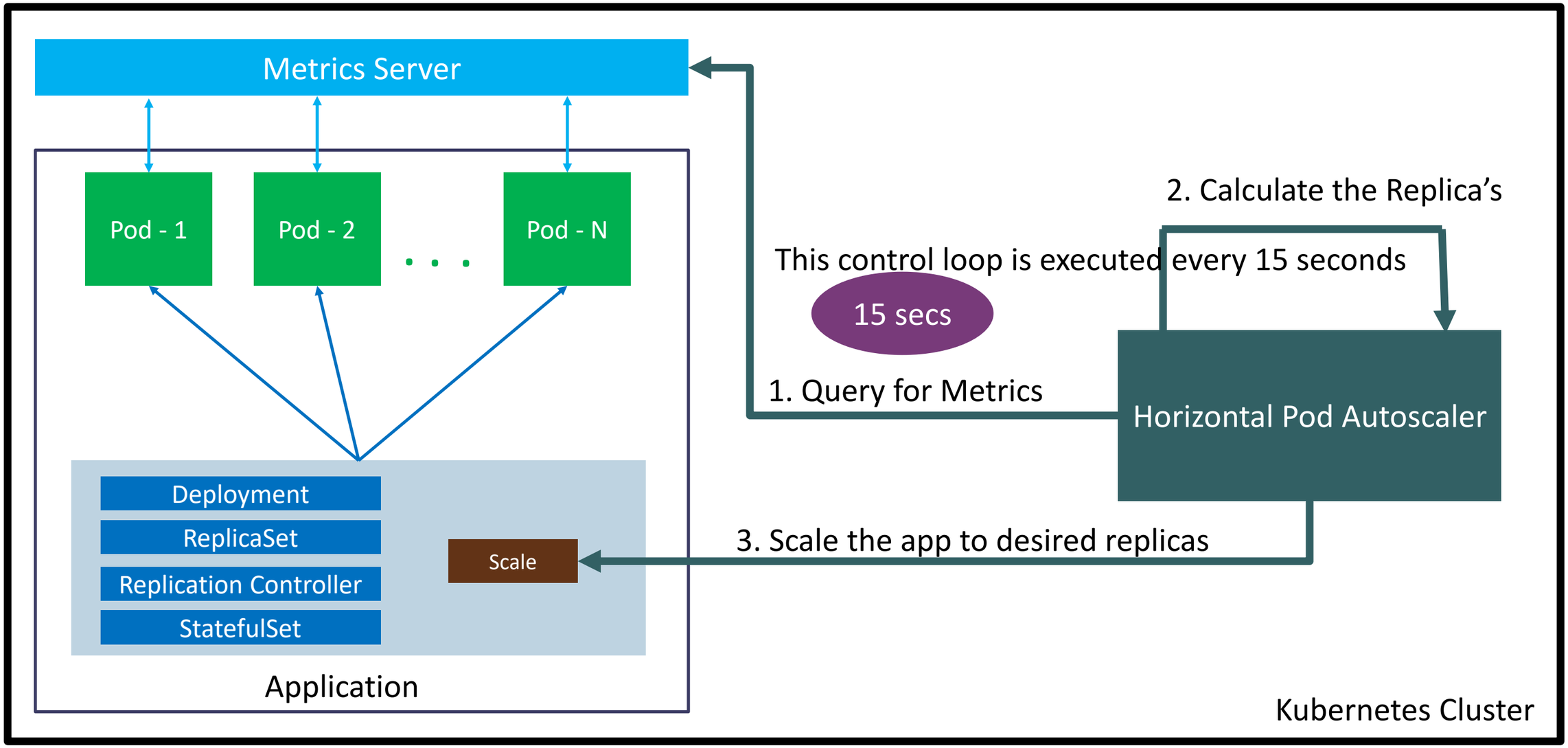


# AWS EKS
# Autoscaling
# Horizontal Pod Autoscaler

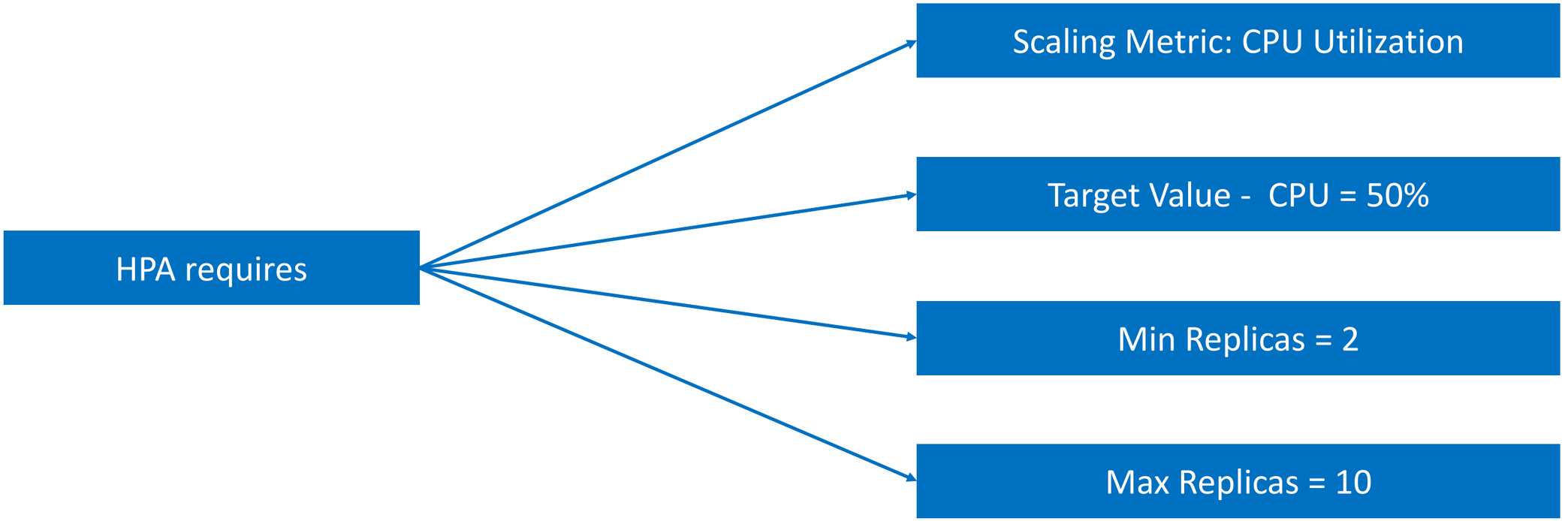
# Horizontal Pod Autoscaler – HPA - Intoduction

- In a very simple note 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.
- 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.

# How HPA works?



# How is HPA configured?

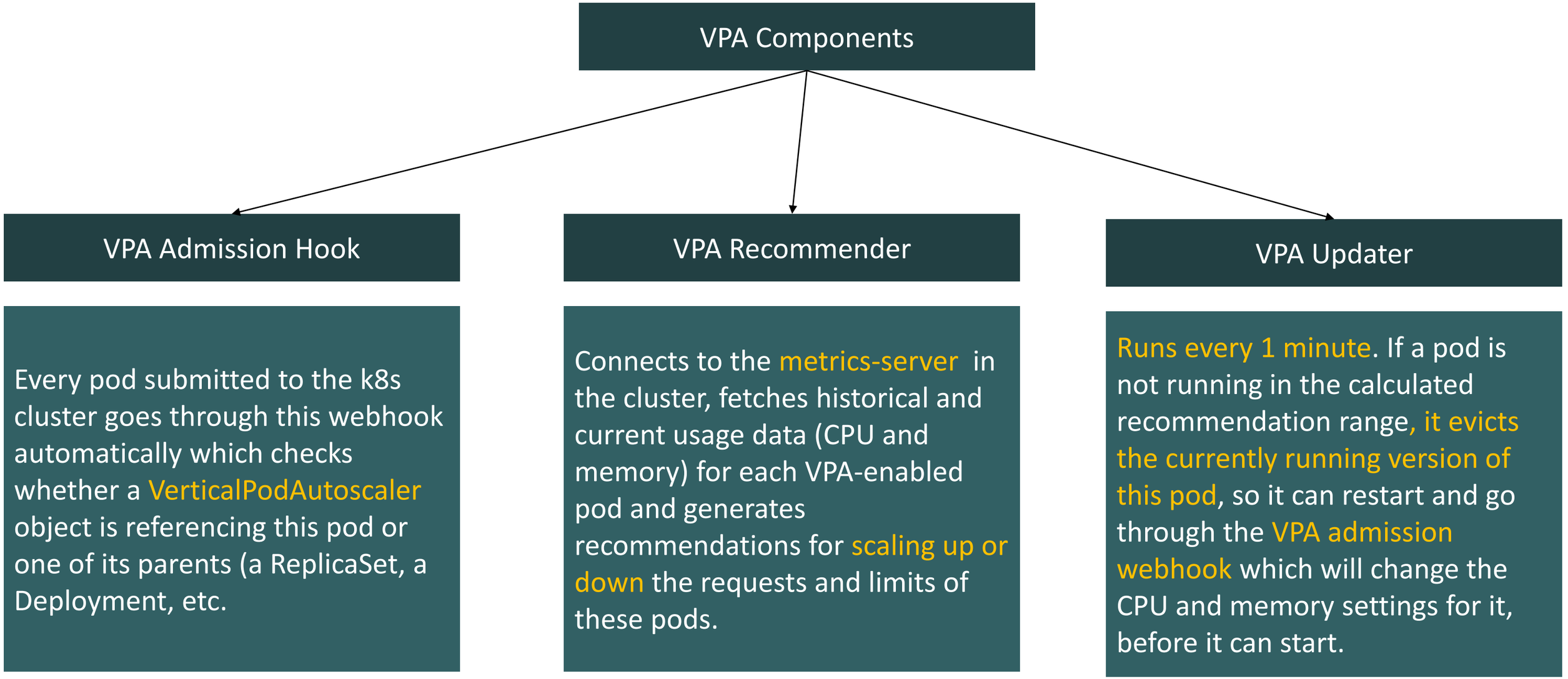



```
kubectl autoscale deployment demo-deployment  --cpu-percent=50 --min=1 --max=10
```

# AWS EKS
# Autoscaling
# Vertical Pod Autoscaler

# Vertical Pod Autoscaler – VPA - Introduction

- 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 autoscaler can adjust CPU and memory requests over time without any action on your part.



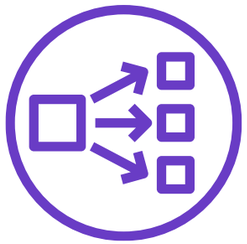
# AWS EKS
# Autoscaling
# Cluster Autoscaler

# Cluster Autoscaler - Introduction

- Cluster Autoscaler is a tool that automatically adjusts the size of a Kubernetes cluster when one of the following conditions is true:
- There are pods that failed to run in the cluster due to insufficient resources.
- There are nodes in the cluster that have been underutilized for an extended period of time and their pods can be placed on other existing nodes.
- The Cluster Autoscaler modifies our worker node groups so that they scale out when we need more resources and scale in when we have underutilized resources.

Elastic Load Balancing

Classic Load Balancer



Network Load Balancer

Application Load Balancer

Certificate Manager


Route53

Elastic Block Store

Amazon RDS

# AWS EKS

# CloudWatch

# Container Insights

Fargate Profiles

Elastic Container Registry

Code Commit

Code Build

Code Pipeline

Simple Email Service

X-Ray

CloudWatch

Simple Notification Service

# Container Insights

- A fully managed observability service for monitoring, troubleshooting and alarming on our containerized applications.
- Container Insights to collect, aggregate, and summarize metrics and logs from our containerized applications and microservices.
- The metrics include utilization for resources such as CPU, memory, disk, and network.
- It also provides diagnostic information, such as container restart failures, to help us isolate issues and resolve them quickly.
- We can also set CloudWatch alarms on metrics that Container Insights collects.
- The metrics that Container Insights collects are available in CloudWatch automatic dashboards.
- We can analyze and troubleshoot container performance and logs data with CloudWatch Logs Insights.

Container Map

Container Resources

Performance Dashboards

Log Groups

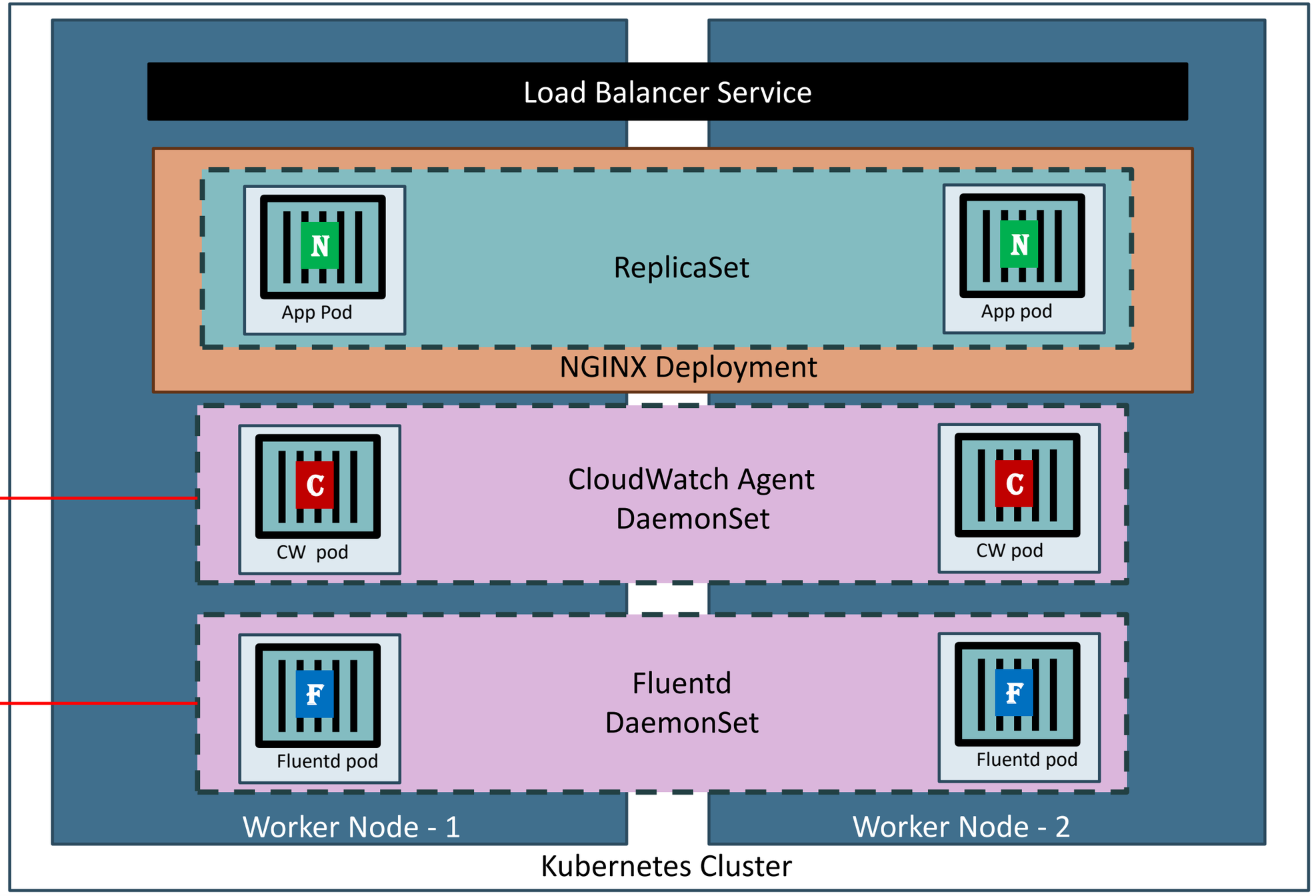
Log Insights

Alarms

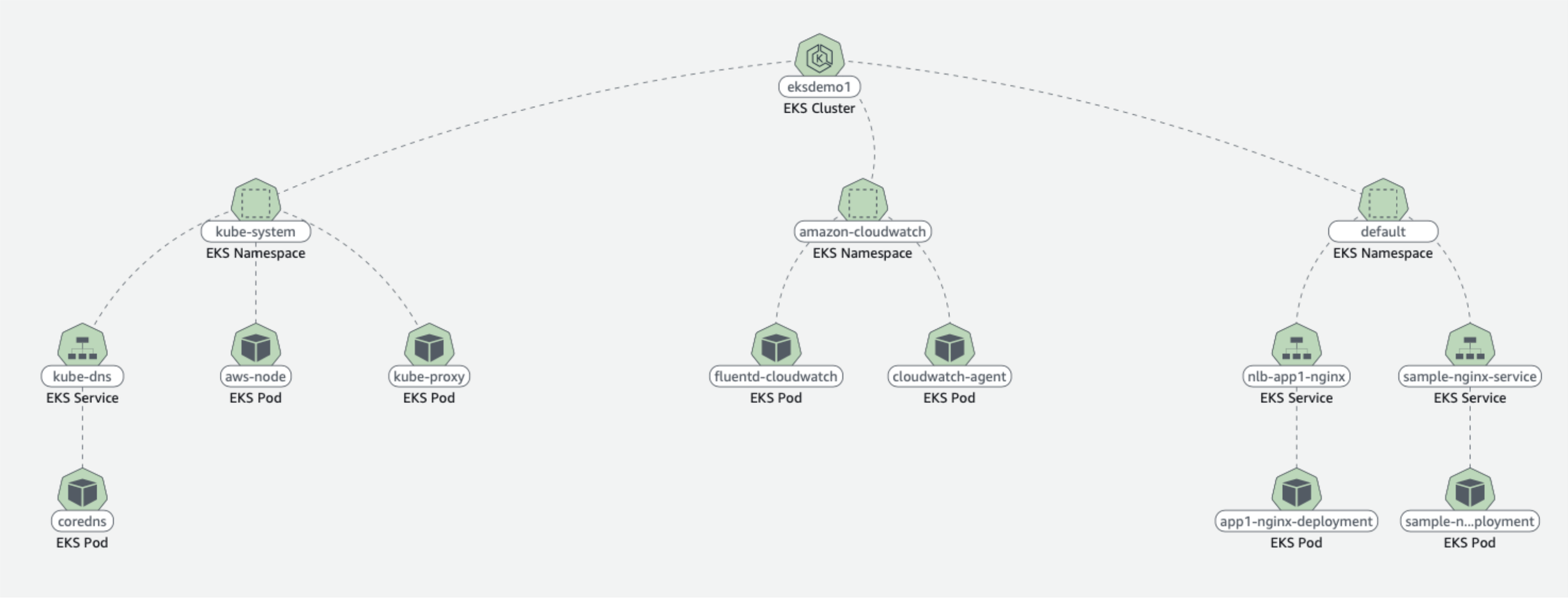
Developer or Operations User

CloudWatch

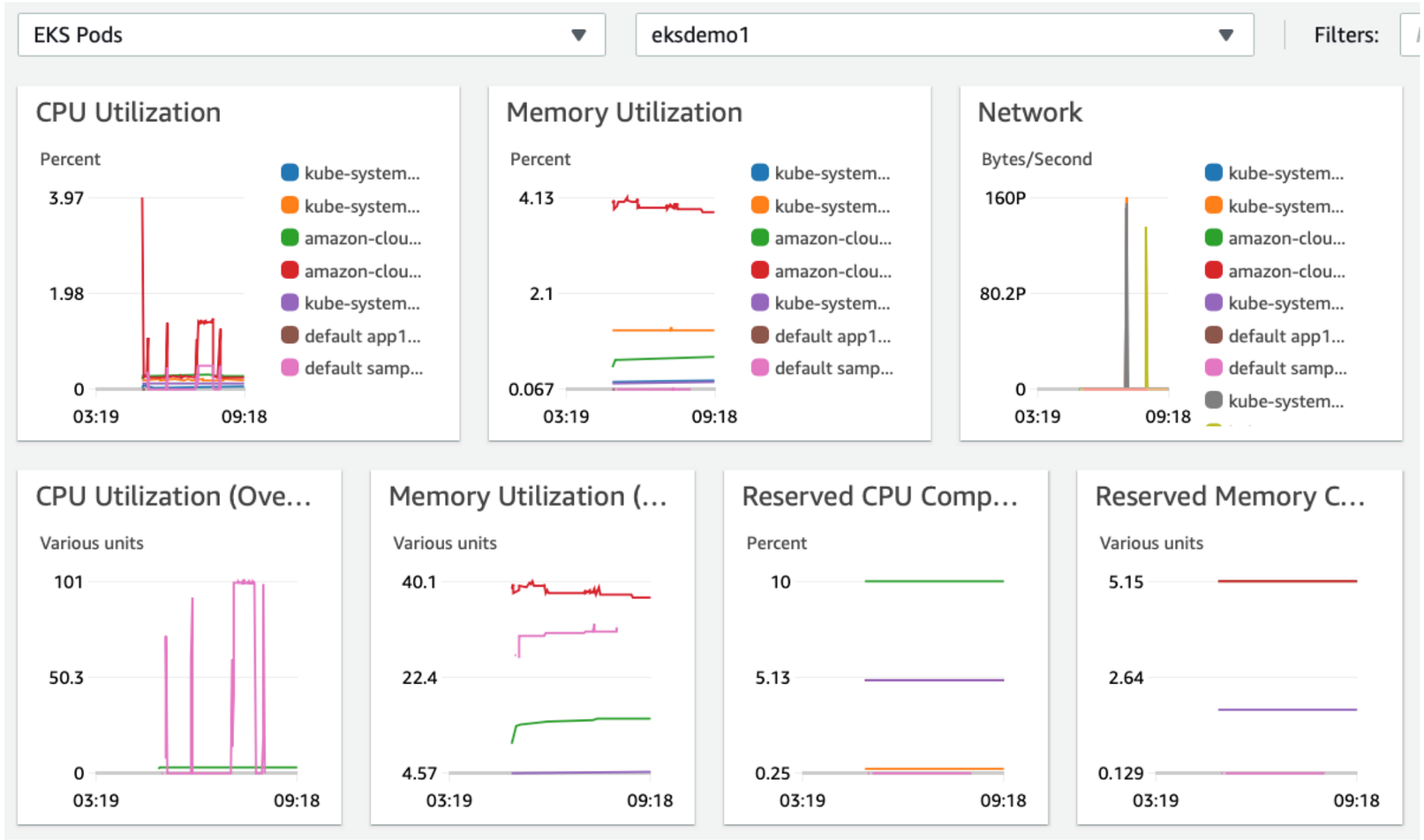
# Container Insights

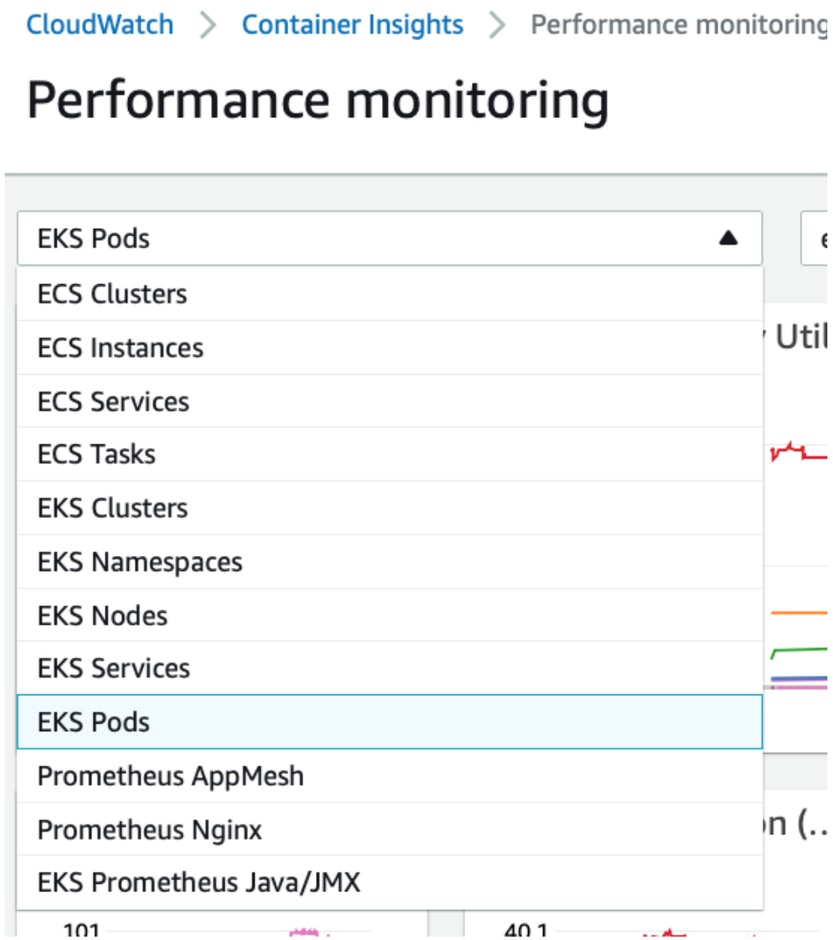


# CloudWatch Container Insights Map



# Automatic Performance Dashboard



# THANK YOU