Containers on AWS

Deep Dive into EKS

Bryan Landes, Solutions Architect

May 12th, 2020

Table of contents

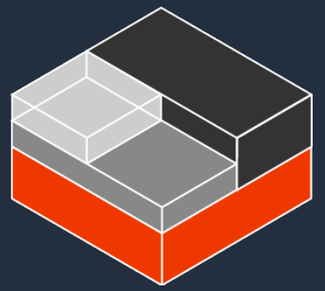
aws

Introduction to Containers

aws

What is a container?

- A lightweight, stand-alone, executable package of software that includes all dependencies: code, runtime, system tools, system libraries, settings.
- Containers isolate software from its surroundings
  - development and staging environments
  - help reduce conflicts between teams running different software on the same infrastructure.
- Long history: chroot, FreeBSD Jails, Solaris Containers, OpenVZ, LXC
- Docker simplified creation/management/operation of containers

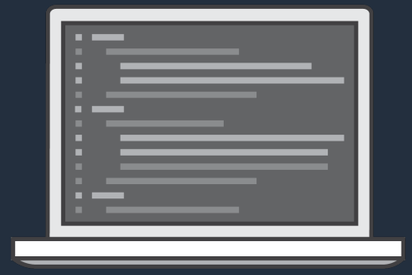


Rocket

containerd

aws

Different environments



Local Laptop

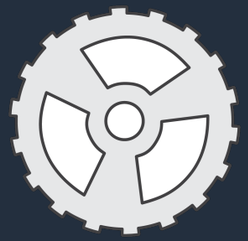
Staging / QA

Production

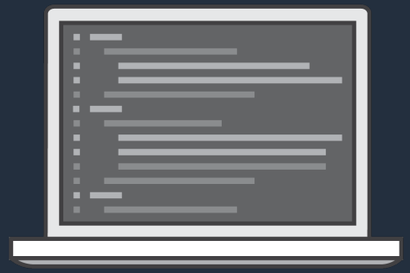
On-Prem

aws

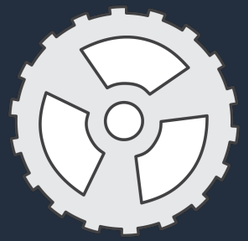
It worked on my machine, why not in prod?



v6.0.0

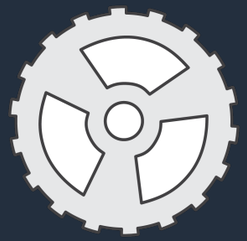


Local Laptop



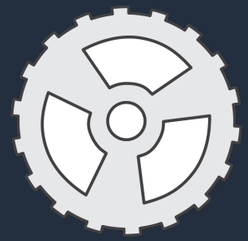
v7.0.0

Staging / QA



v4.0.0

Production



v7.0.0

On-Prem

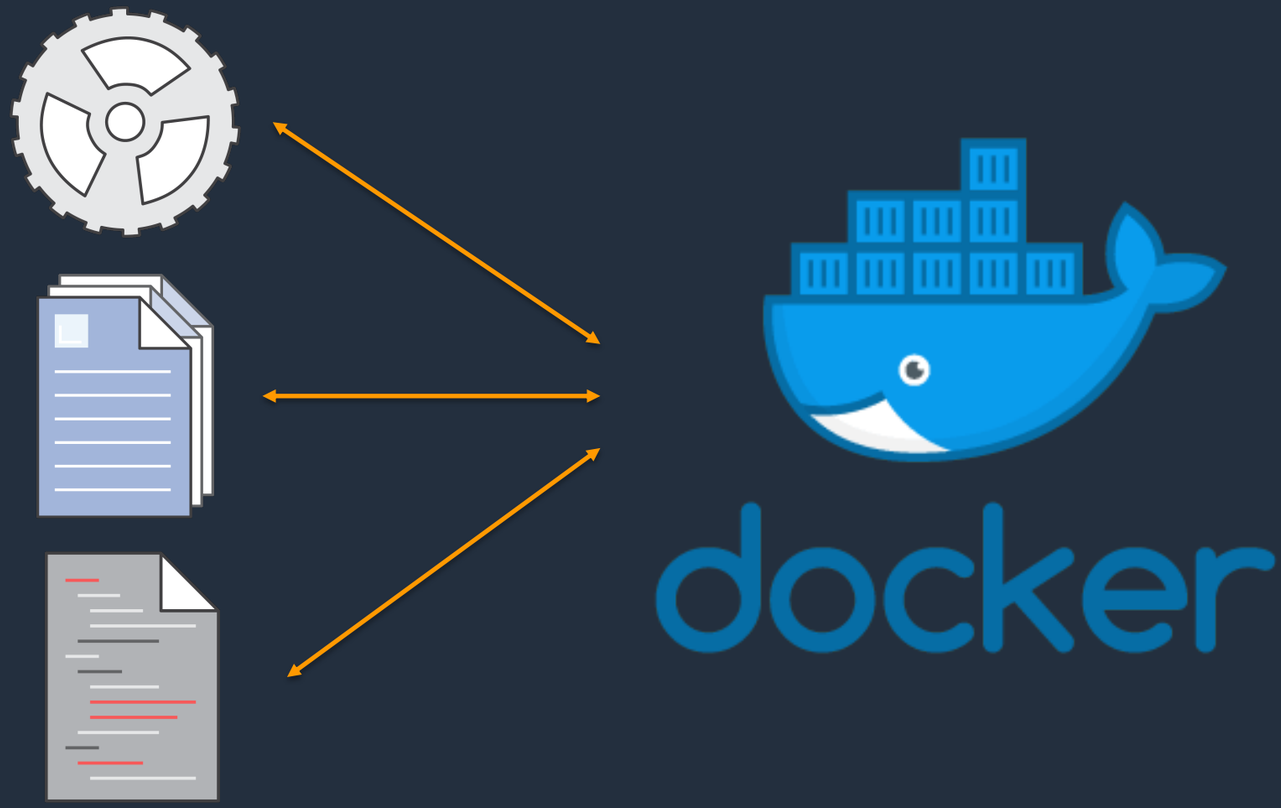
aws

Docker to the rescue

Runtime Engine

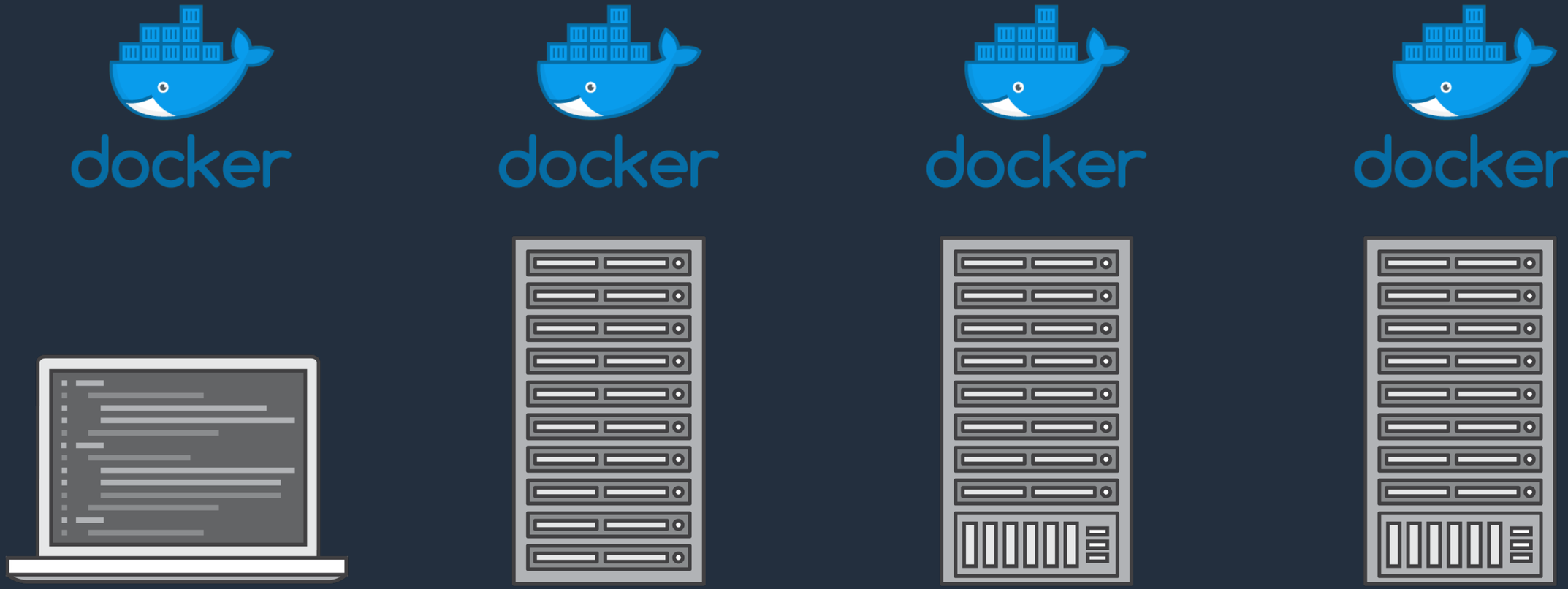
Dependencies

Code



aws

Four environments, same container



Local Laptop

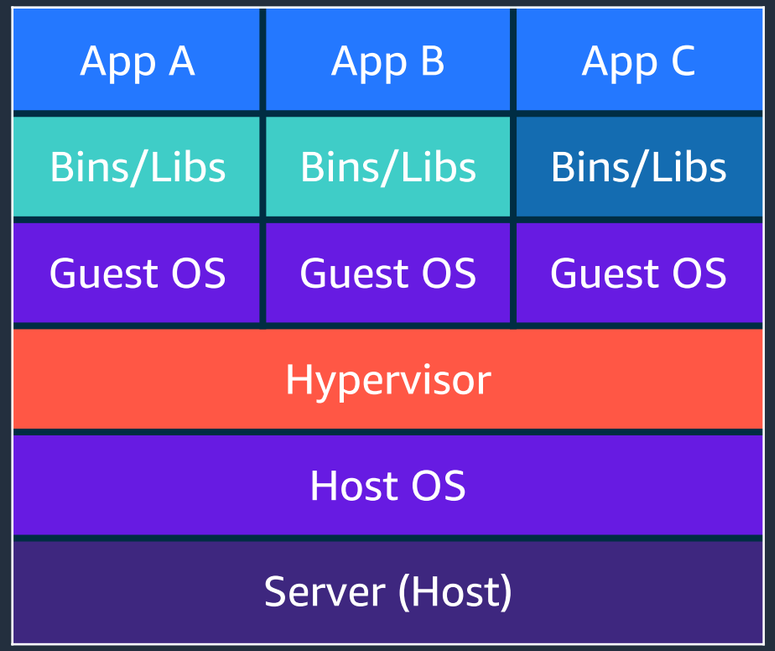
Staging / QA

Production

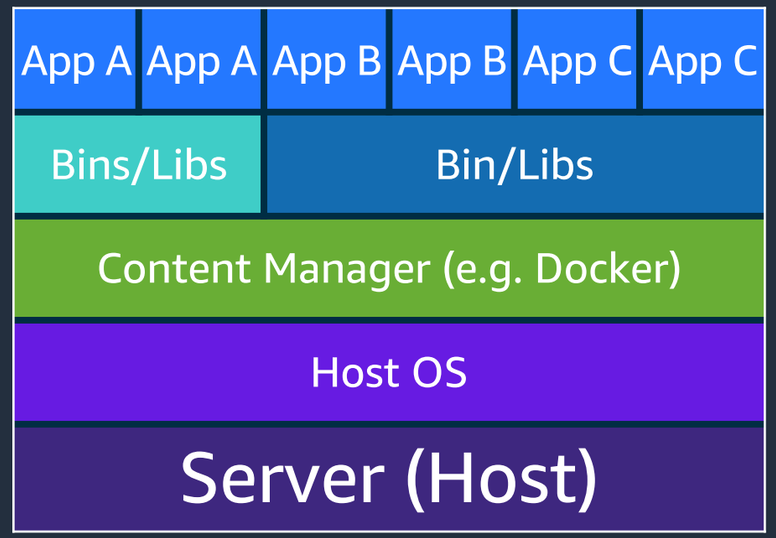
On-Prem

aws

VMs VS. Containers



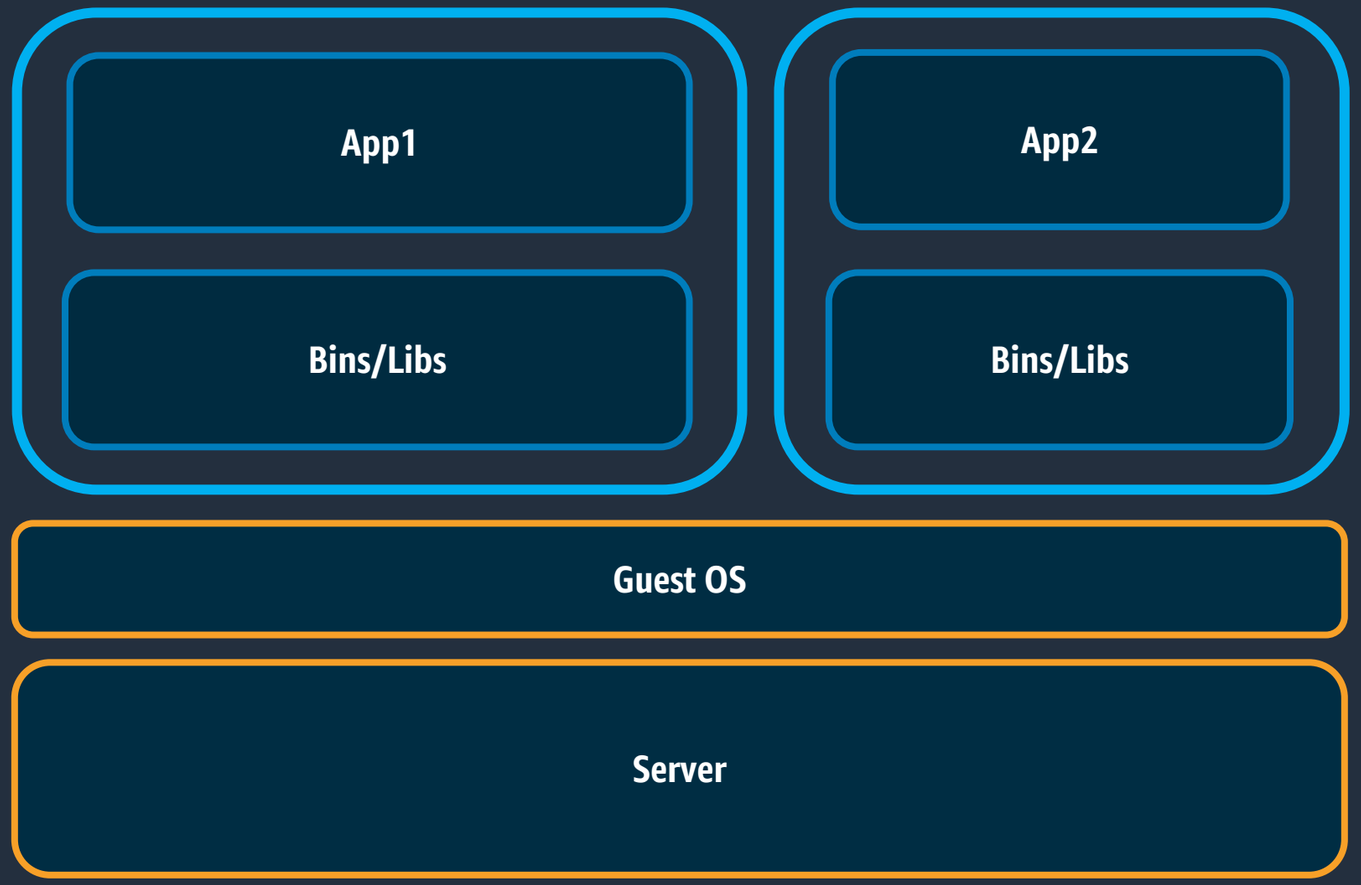
VMs



Containers

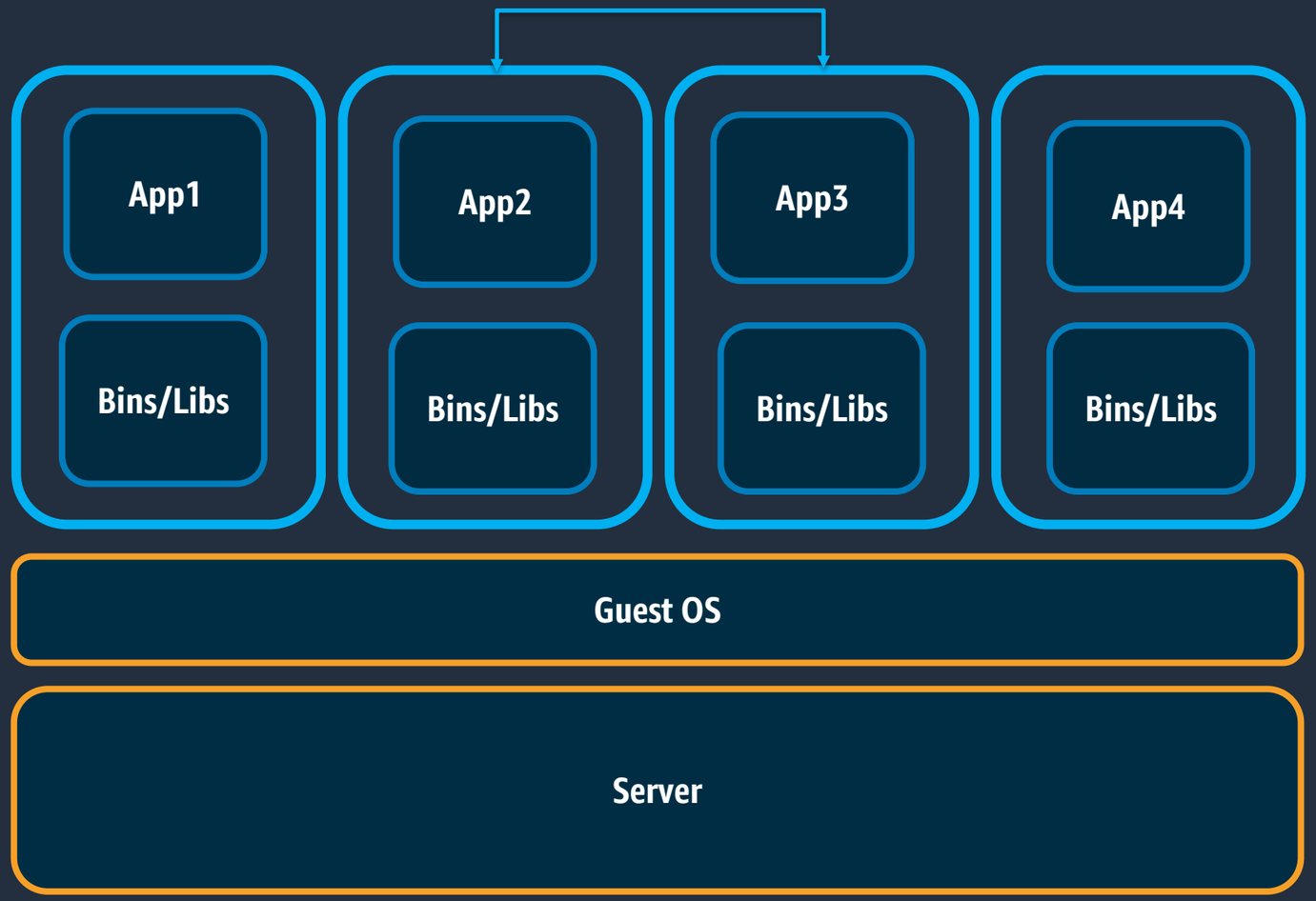
aws

But there are still moving pieces:



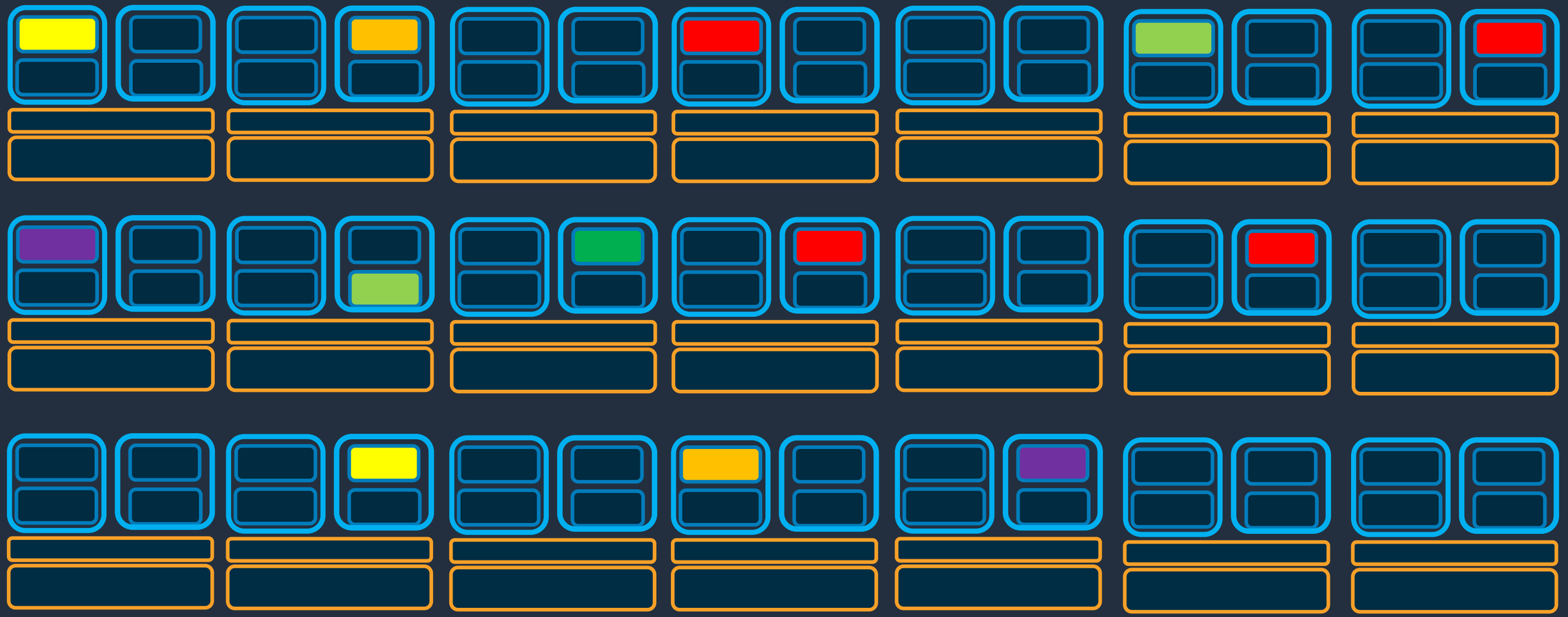
aws

And more moving pieces...



aws

Managing many containers is hard

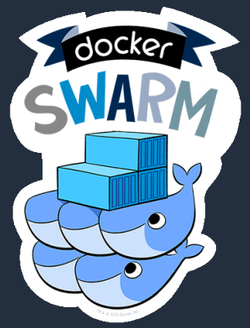


aws

What are container orchestration tools?

Framework for managing, scaling, deploying containers.

kubernetes

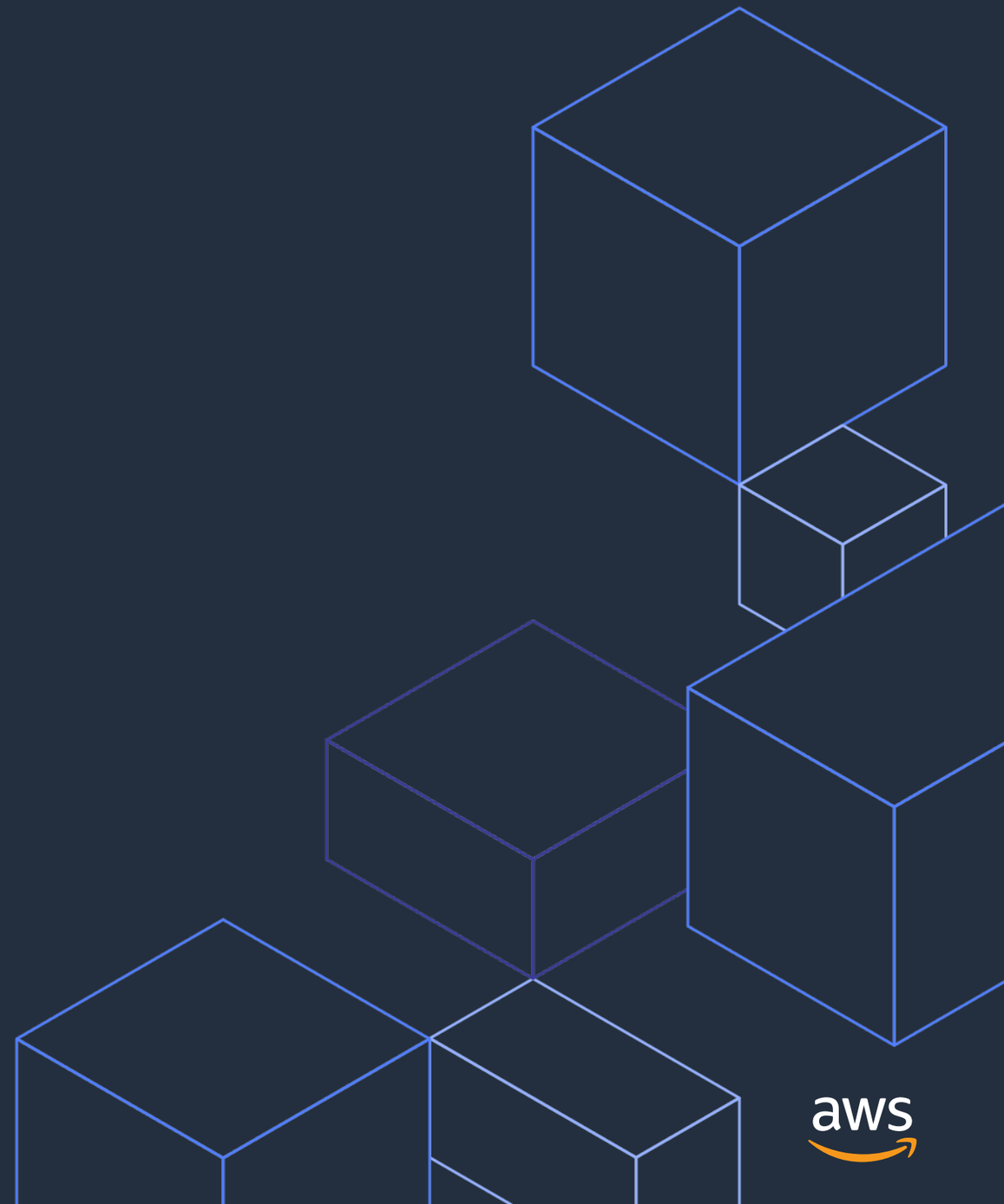


MESOS

RANCHER

aws

Containers on AWS



AWS container services landscape

Management

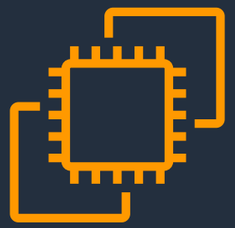
Deployment, Scheduling, Scaling & Management of containerized applications

Amazon Elastic Container Service

Amazon Elastic Container Service for Kubernetes

Hosting

Where the containers run



Amazon EC2

AWS Fargate

Image Registry

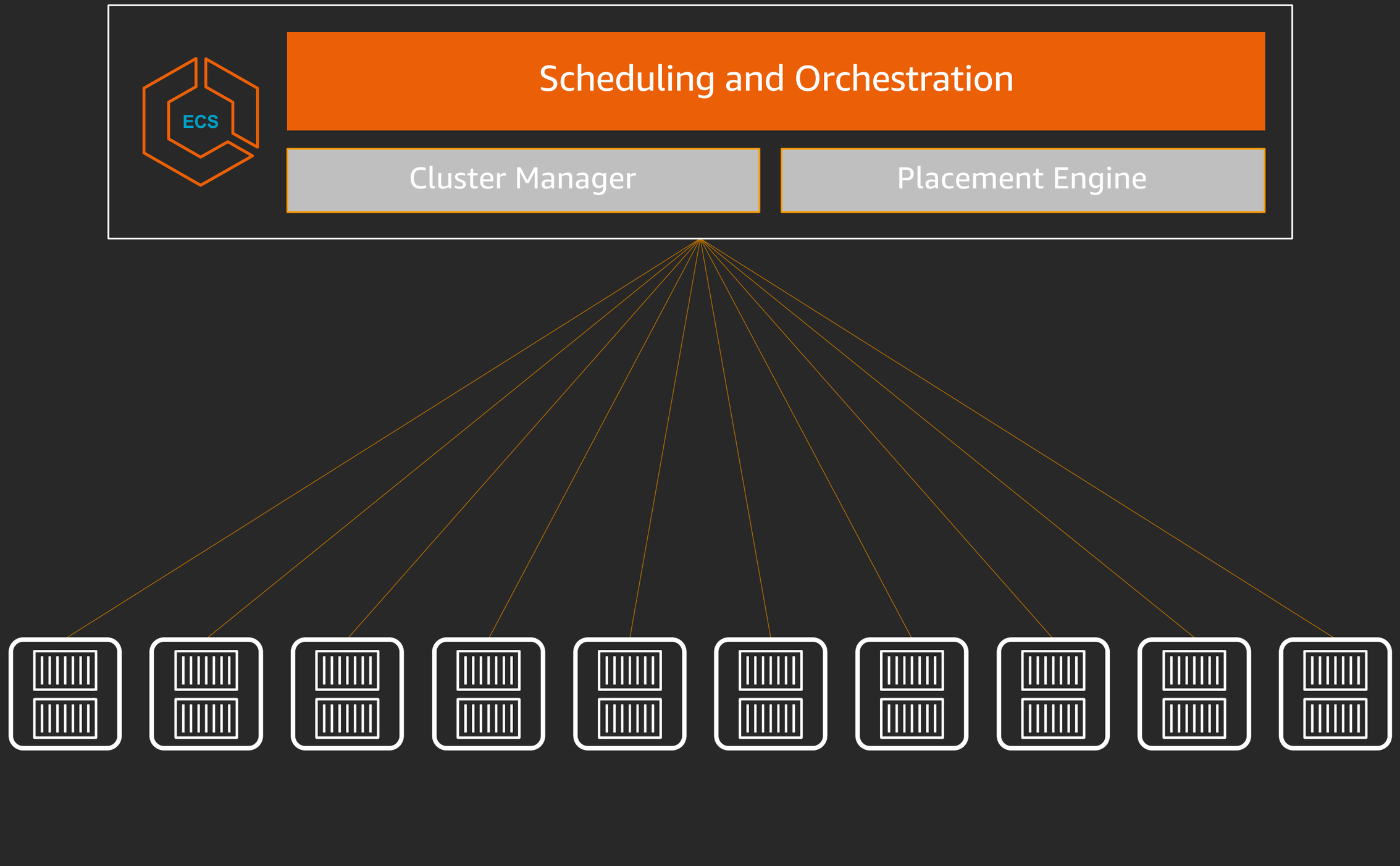
Container Image Repository

Amazon Elastic Container Registry

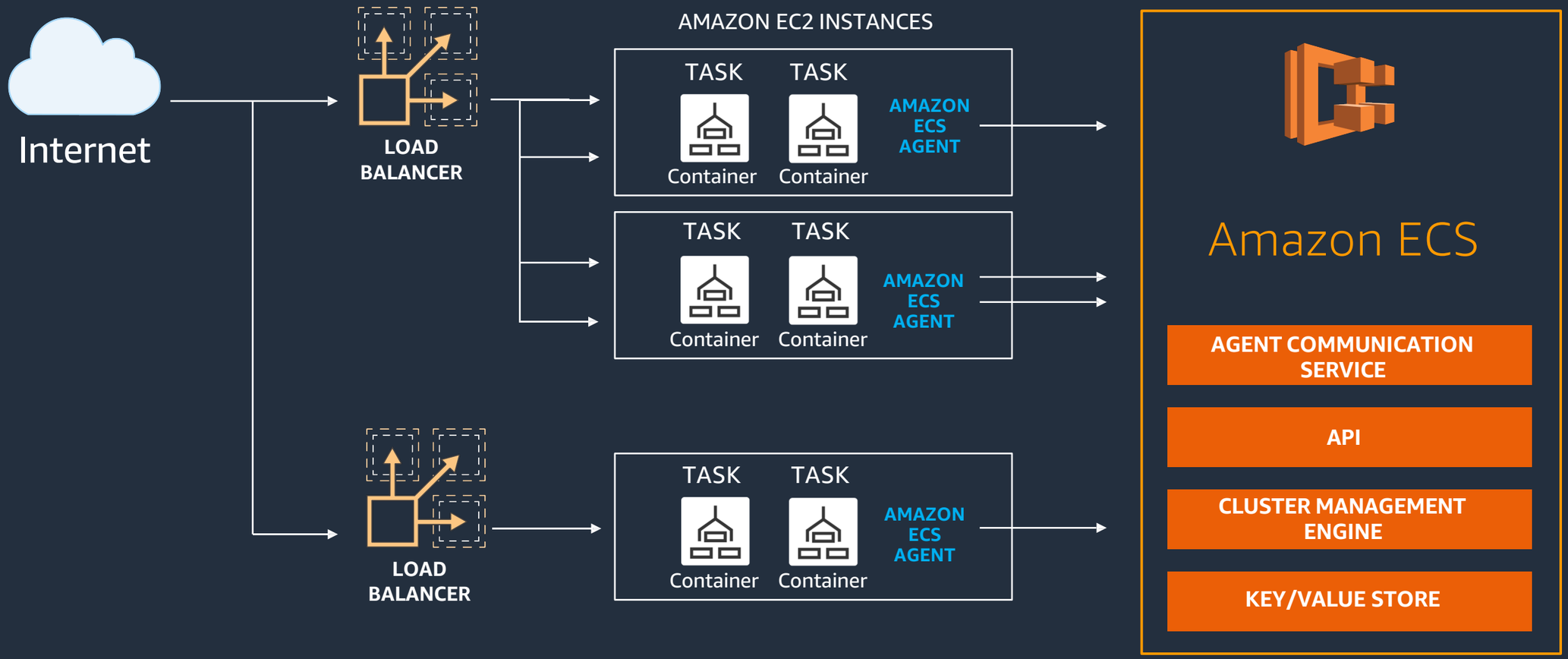
aws

Amazon ECS

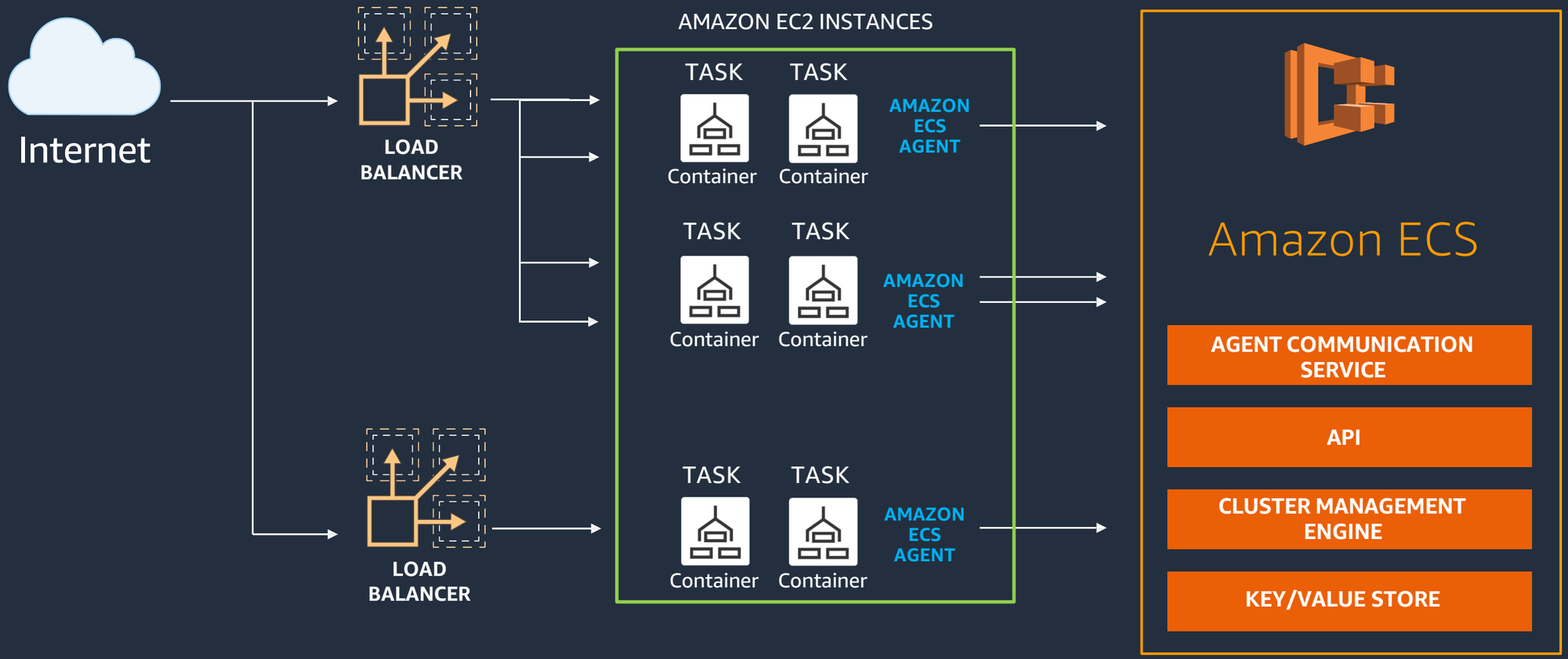
aws


ECS
Scheduling and Orchestration
Cluster Manager
Placement Engine

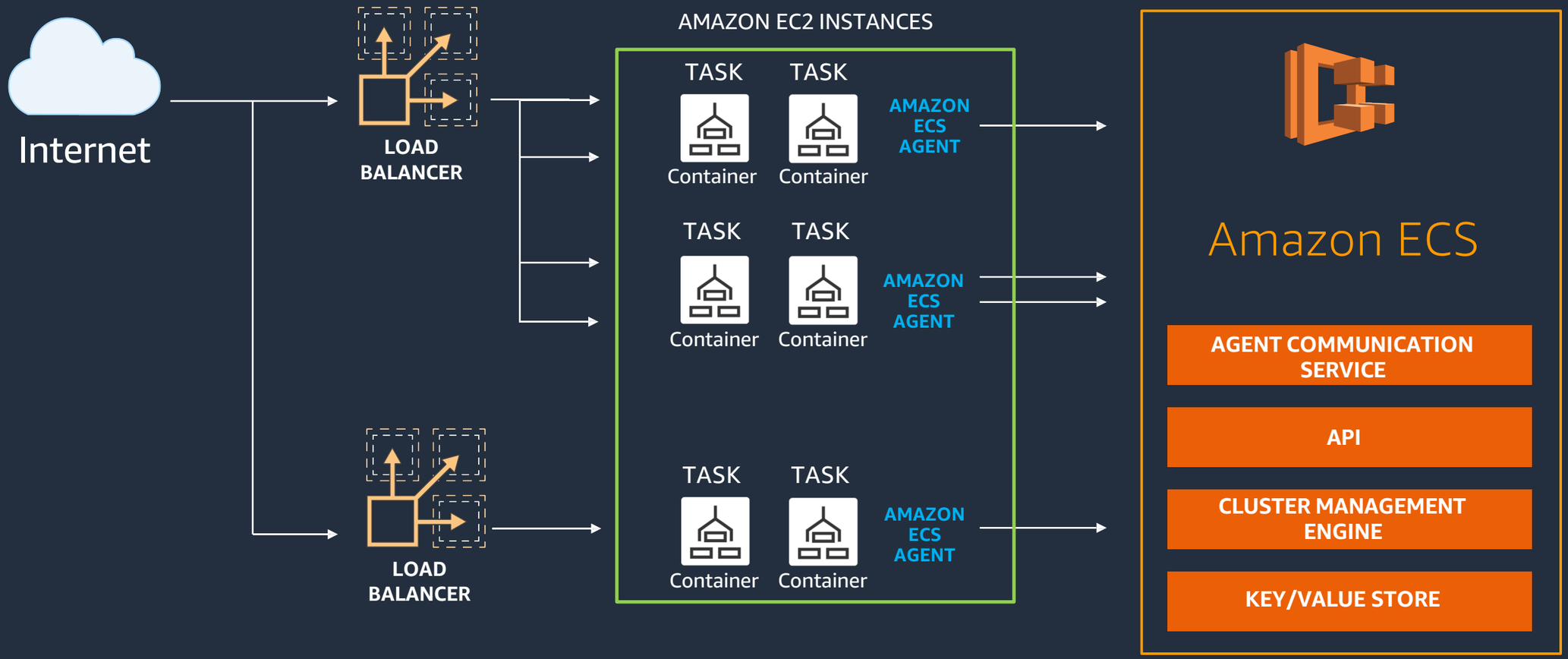
Amazon ECS



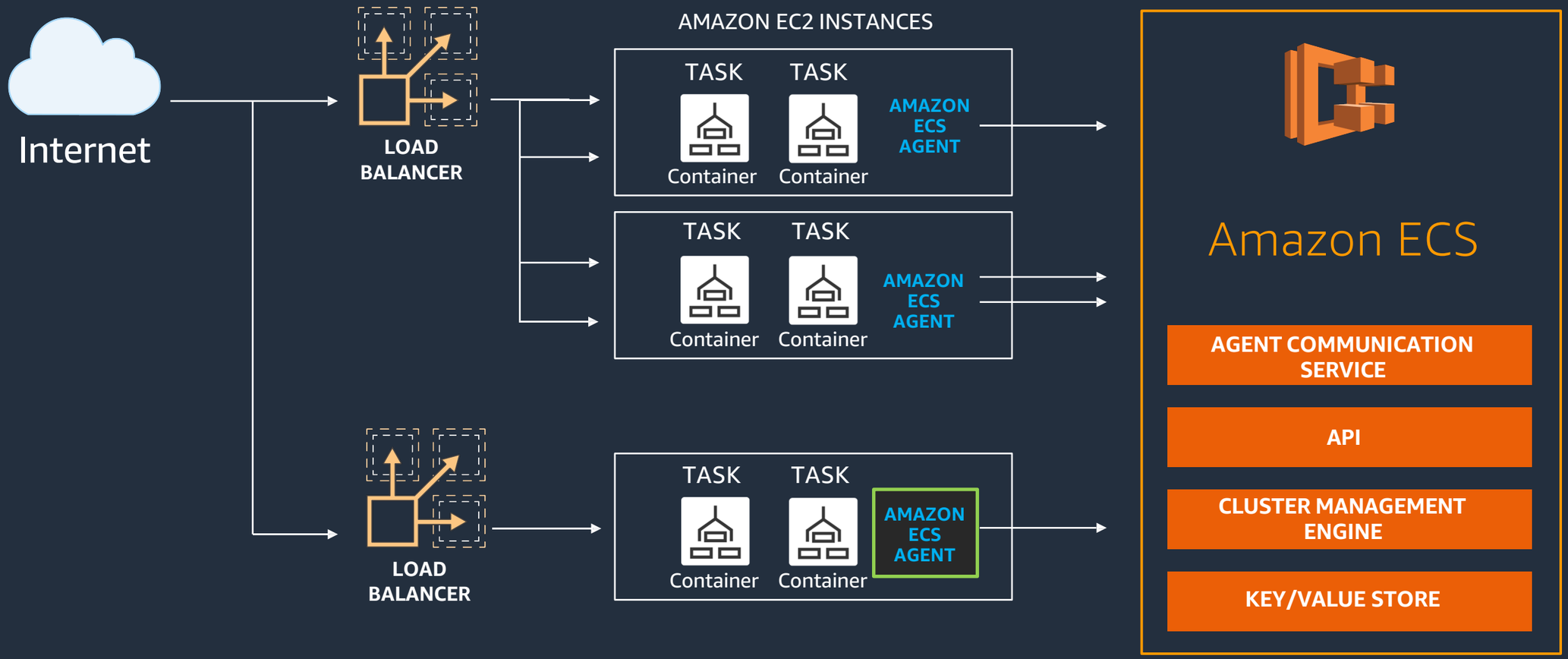
Cluster of hosts on AmazonEC2



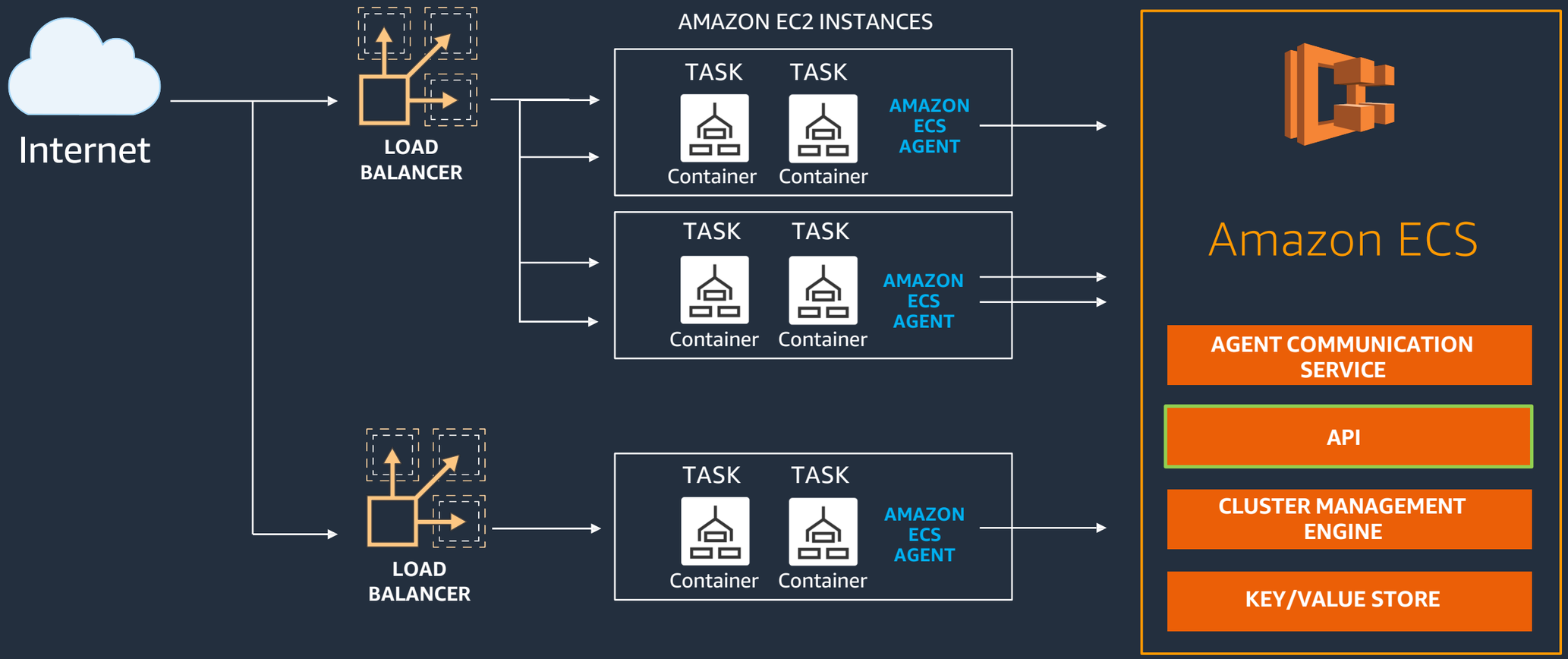
Lightweight agent on each host



Lightweight agent on each host

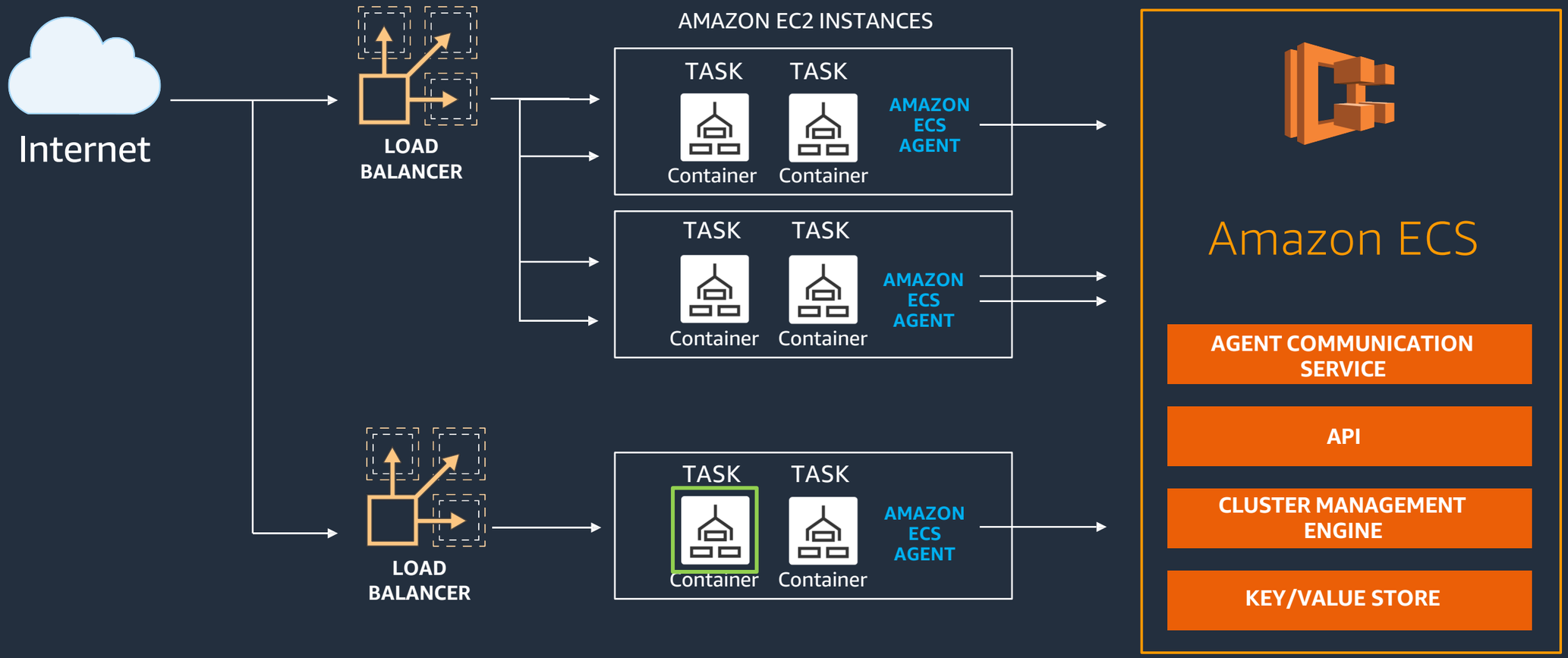


API for launching containers onthe cluster

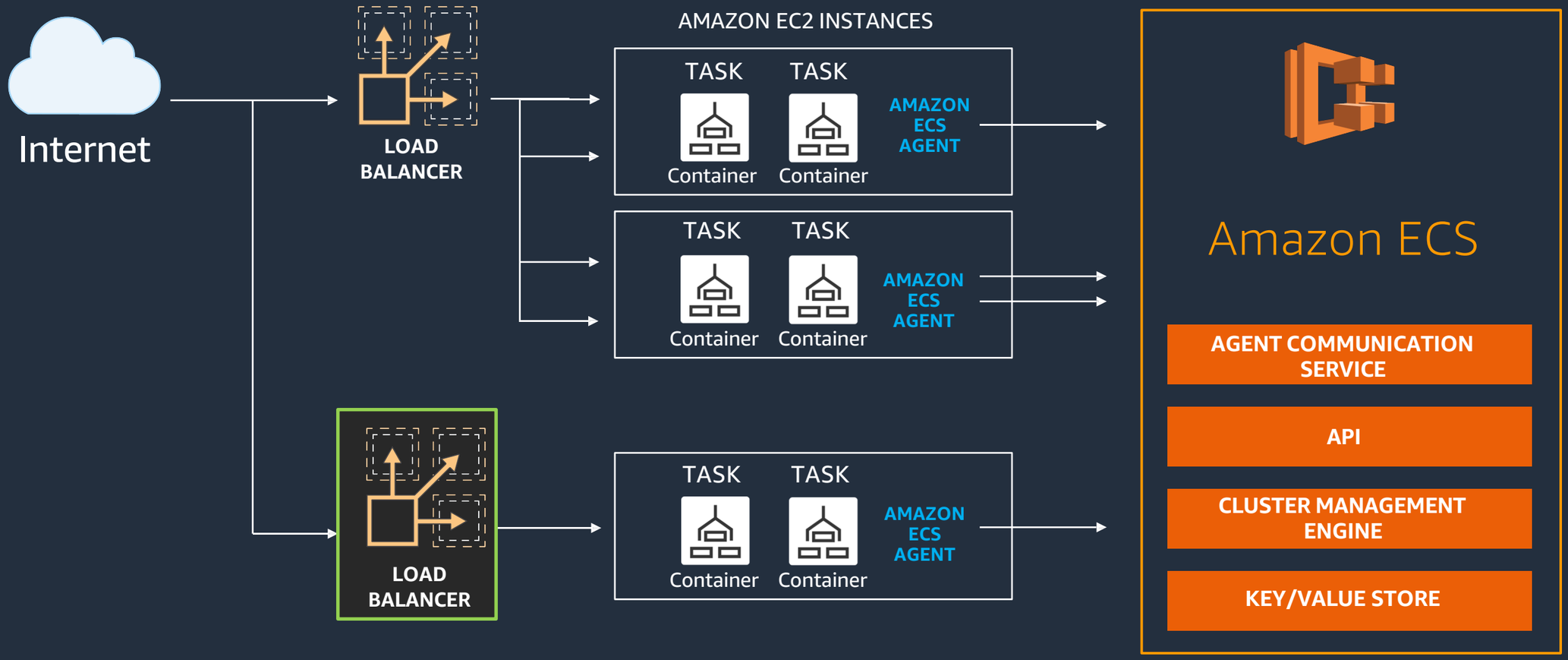


aws

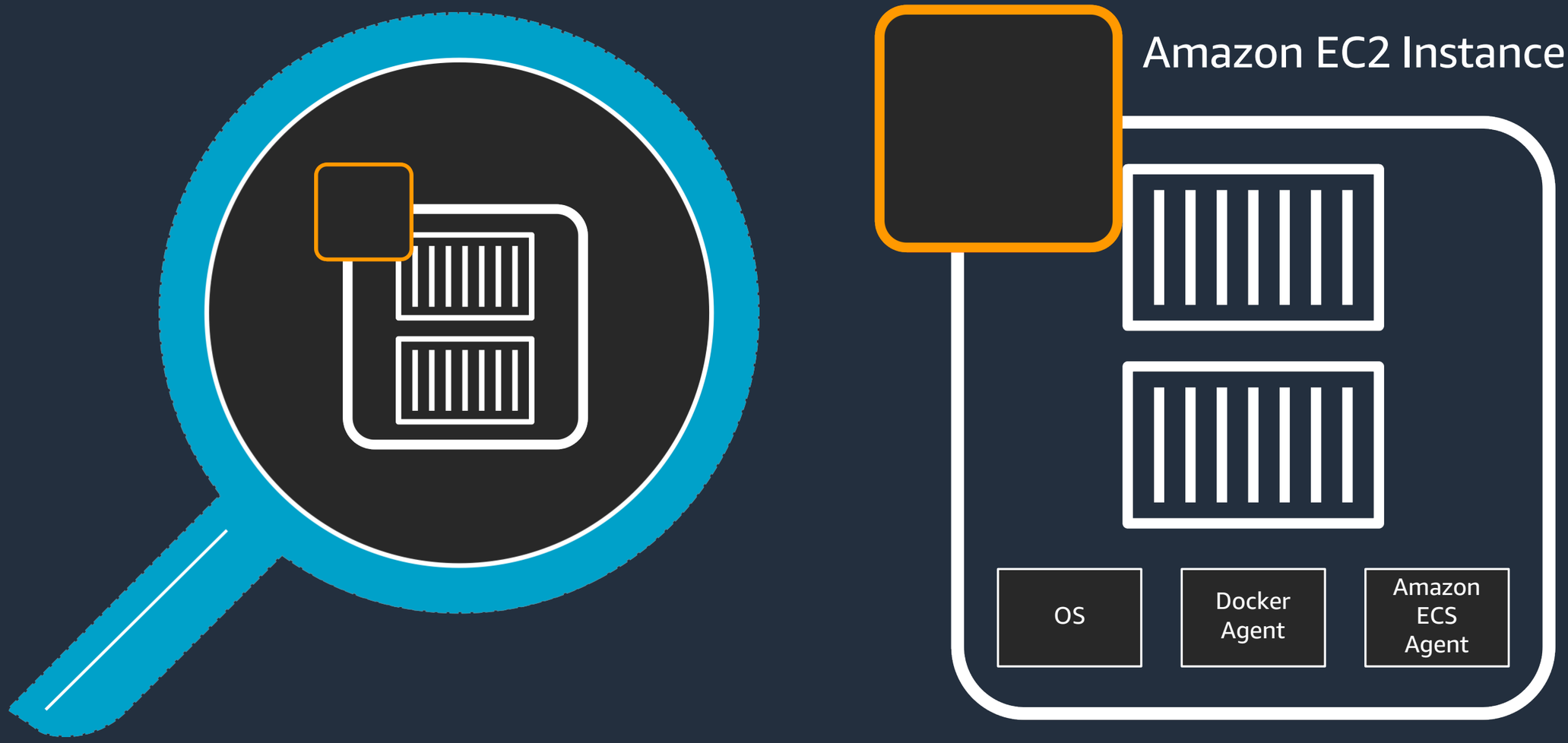
Container task is placed on a host



Traffic is sent to yourhost



You end up managing more than just containers.

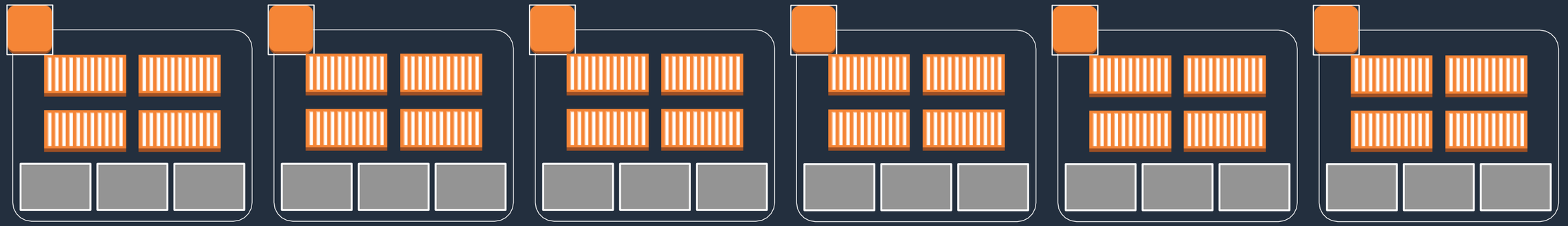


aws

Managing instance fleets is hard work, too

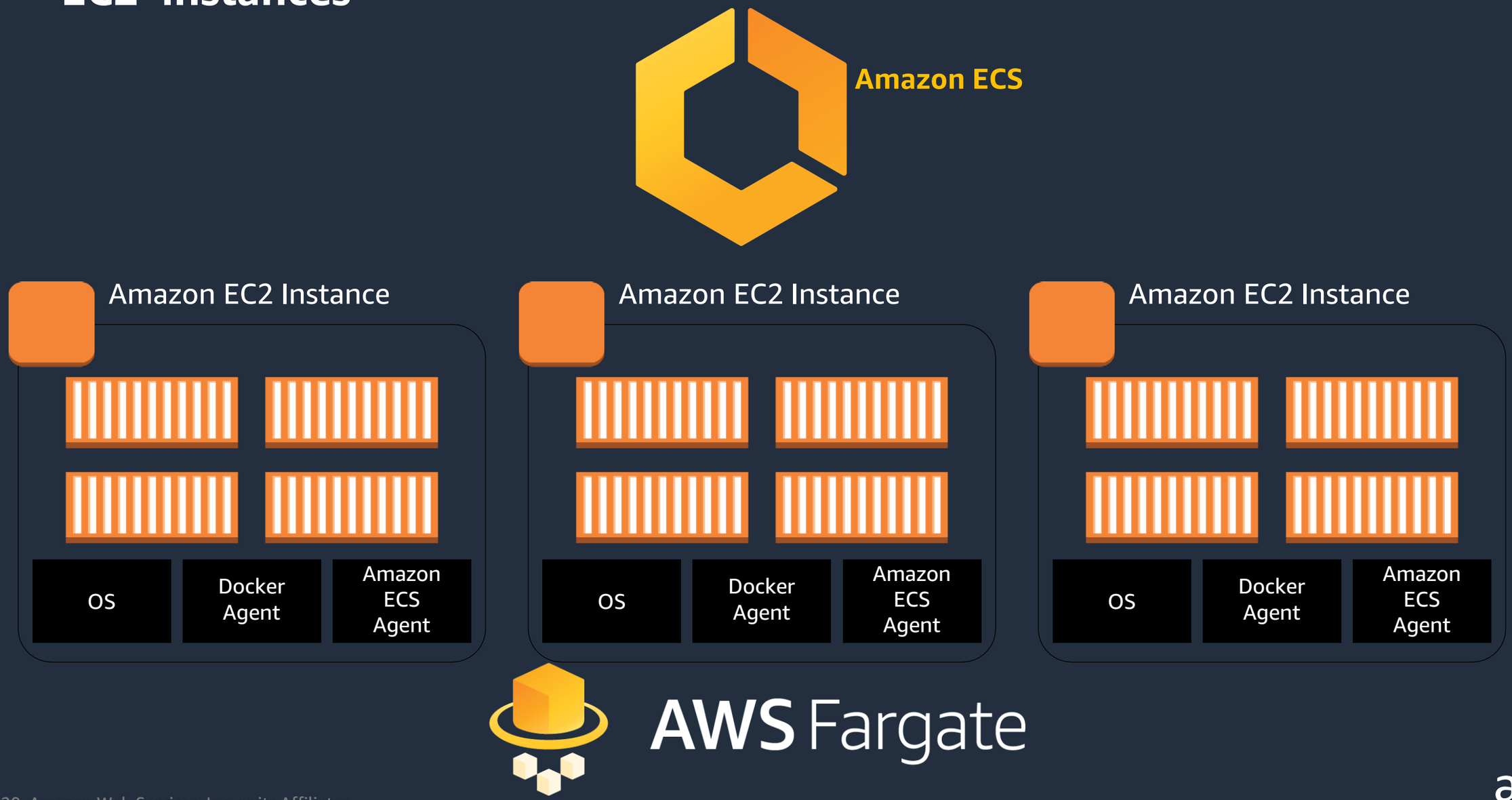
Patching and Upgrading OS, agents, etc.

Scaling the instance fleet for optimal utilization



aws

Customers wanted to run containers without having to manage Amazon EC2 instances



AWS Fargate

Your containerized applications

Serverless

No Amazon EC2 Instances to provision, scale, or manage

Elastic

Scale up and down seamlessly
Pay only for what you use

Integrated

with AWS: Amazon VPC Networking, Elastic Load Balancing, IAM permissions, CloudWatch, and more

aws

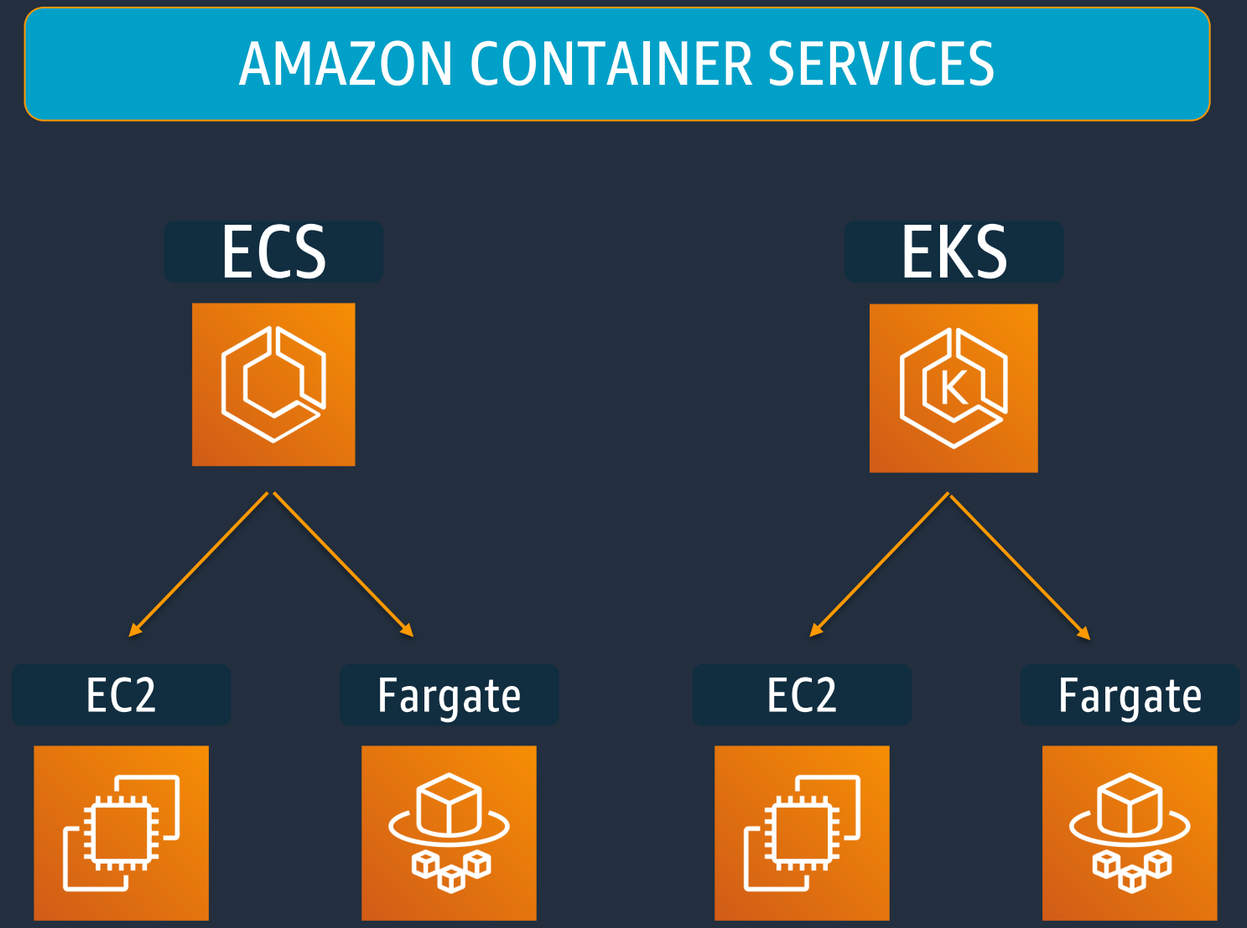
Amazon ECS/Amazon EC2 vs. Amazon ECS/AWS Fargate

	Amazon EC2	AWS Fargate
Managed by	Customer	AWS
Storage	Ephemeral or persistent	Only ephemeral
Sidecar pattern	Yes	Yes
Network mode	Bridge or VPC mode	VPC mode
Daemons	Yes	No
SSH into host	Yes	No
Privileged containers	Yes	No

aws

So you want to run a (managed) container on AWS?

1. Choose your orchestration tool
2. Choose your launch type



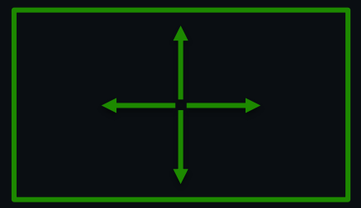
aws

Amazon Elastic Container Service for Kubernetes

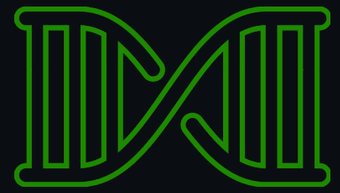
aws

What is Kubernetes?

Open-source container-management platform



Helps you run containers at scale



Gives you primitives for building modern applications

How are customers using Amazon EKS?

Microservices

Platform as a service

Enterprise app migration

Machine learning

aws

Community, contribution, choice

kubernetes

aws

But where you run Kubernetes matters

Quality of the cloud platform

Quality of the applications

Your users

aws

+

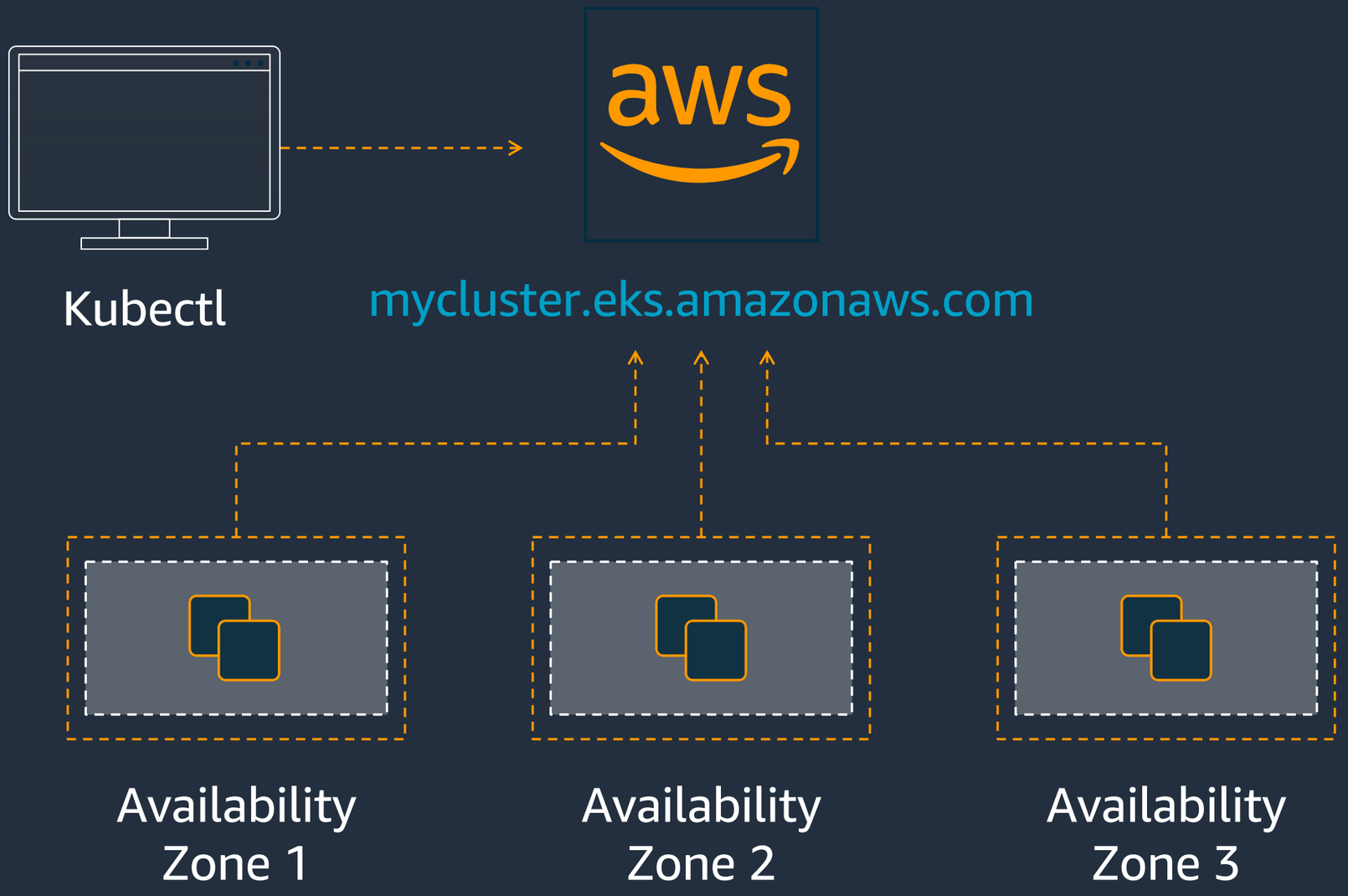
aws

51%

of Kubernetes workloads run on AWS today

—CNCF survey

aws



EKS is Kubernetes Certified

Kubernetes Conformance

- Guaranteed Portability and Interoperability
- Timely Updates
- Confirmability

aws

AWS Cloud Map

AWS
Cloud
Map

Service discovery for all your cloud resources

Constantly monitor the health of every resource

Dynamically update the location of each microservice

Increase developer productivity

Single registry for all app resources

Define resources with user-friendly names

Integration with Amazon container services

AWS Fargate

Amazon ECS

Amazon EKS

aws

AWS App Mesh

Observability & traffic control

Easily export logs, metrics, and traces

Client side traffic policies—circuit breaking, retries

Routes for deployments

Works across clusters and container services

Amazon ECS

Amazon EKS

Kubernetes on EC2

AWS Fargate

AWS built and run

No control plane to manage

Ease of operations

High scale

aws

Customers adopting Kubernetes on AWS

aws

Customer example: Snap

"Undifferentiated heavy lifting is work that we have to do that doesn't directly benefit our customers. It's just work. Amazon EKS frees us up to worry about delivering customer value and allows developers without operational experience to innovate without having to know where their code runs."

More detailed talk: AWS New York Summit 2018 - Run Kubernetes with Amazon EKS (SRV318)

aws

Who is using Amazon EKS?

"We built the next generation of our PaaS using Amazon EKS for large enterprise workloads. We manage thousands of applications and have hundreds of DevOps teams."

aws

Rich partner ecosystem

Foundation

CANONICAL

RANCHER

docker

Red Hat

DevOps

Spinnaker

pulumi

CloudBees
The Enterprise Jenkins Company

GitLab

ATLASSIAN

Monitoring & logging

circleci

DATADOG

sysdig

New Relic.

Security

aqua

Twistlock

NeuVector

Networking

linkerd

TiGERA
CLOUD NETWORKS, SECURED

weaveworks

aws

Open-source and Amazon EKS

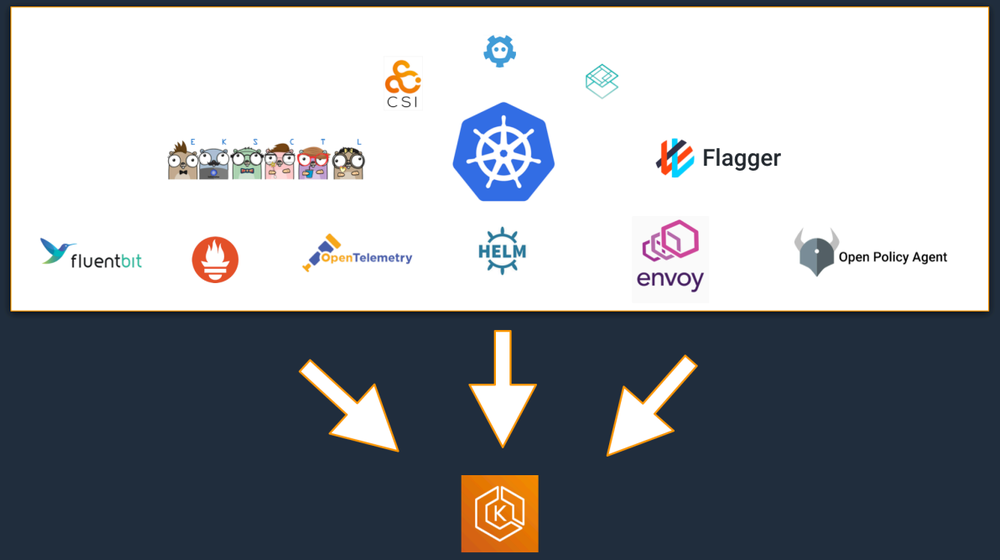
Amazon EKS runs 100% upstream Kubernetes

Key components of Amazon EKS are open source

- Amazon VPC CNI plugin
- AWS Identity and Access Management (IAM) authenticator
- Amazon EKS AMI

Team contributes to or manages 20+ OSS projects

- /kubernetes
- /kubernetes/autoscaler
- /aws-labs/aws-service-operator
- /weaveworks/eksctl
- Amazon EBS, Amazon EFS, Amazon FSx CSI drivers



aws

Kubernetes versions

Latest: 1.15
Amazon EKS will support up to three versions of Kubernetes at once

Deprecation in line with the community stopping support for older versions

aws

eksctl—a CLI for Amazon EKS

- Single command cluster creation

```
eksctl create cluster --nodes=4
```

- Open source and on GitHub
- Built by Weave and AWS
- Official Amazon EKS CLI

aws

Provisioning worker nodes

AWS CloudFormation

eksctl

eksctl

Terraform
Pulumi
Rancher

. . . and more

Partners

aws

Bring your own instances
Instance flexibility

Standard EC2 compute instance types

P2 and P3 accelerated instances

i3 bare metal

Spot Instances

aws

Bring your own OS
Amazon EKS AMI build scripts

https://github.com/awslabs/amazon-eks-ami

Source of truth for Amazon EKS Optimized AMI

Easily build your own Amazon EKS AMI with Packer

Build assets for Amazon EKS AMI for each supported Kubernetes version

aws

Windows containers

Run Windows containers and Windows Server nodes with Amazon EKS

Supports heterogeneous (mixed) clusters

Kubernetes version 1.11+

Available in all Amazon EKS Regions

Developer preview:
https://github.com/aws/containers-roadmap

aws

Amazon EKS-optimized GPU AMI

Includes NVIDIA packages to support Amazon P2 and P3 instances

Easily run TensorFlow on Amazon EKS

Now supporting P3dn.24xlarge instances

CUDA 10 with NVIDIA v410 coming soon!

aws

Our tenets

1. Amazon EKS is a platform to run **production-grade workloads**. Security and reliability are our first priority. After that we focus on doing the heavy lifting for you in the control plane, including lifecycle-related things like version upgrades.

2. Amazon EKS provides a **native and upstream Kubernetes** experience. Amazon EKS provides vanilla, un-forked Kubernetes. In keeping with our first tenet, we ensure the Kubernetes versions we run have security-related patches—even for older, supported versions—as quickly as possible. But there's no special sauce and no lock-in.

3. If you want to use additional AWS services, **integrations** are as **seamless** as possible.

4. The Amazon EKS **team** at AWS **actively contributes** to the **upstream Kubernetes** project and the wider CNCF activities, both on the technical level as well as community, from communicating good practices to participation in SIGs and working groups.

aws

Amazon EKS, a year in review

June – December 2018:

Amazon EKS achieves K8s conformance, HIPAA-eligibility, generally available
Amazon EKS AMI build scripts and AWS CloudFormation templates available in GitHub
Support for GPU-enabled EC2 instances, support for HPA with custom metrics
Amazon EKS launches in Dublin, Ireland
Amazon EKS simplifies cluster setup with `update-kubeconfig` CLI command
Amazon EKS adds support for Dynamic Admission Controllers (Istio), ALB Support with the AWS ALB ingress controller
Amazon EKS launches in Ohio, Frankfurt, Singapore, Sydney, and Tokyo
Amazon EKS adds Managed Cluster Updates and Support for Kubernetes Version 1.11, CSI Driver for Amazon Elastic Block Store (Amazon EBS)

2019:

Amazon EKS launches in Seoul, Mumbai, London, and Paris
Amazon EKS achieves ISO and PCI compliance, announces 99.9% SLA, cluster creation limit raised to 50
API server endpoint access control, AWS App Mesh controller
Windows support (preview), Kubernetes version 1.12
CSI drivers for Amazon Elastic File System (Amazon EFS), Amazon FSx for Lustre, control plane logs, A1 (ARM) instance support (preview)
Deep Learning Benchmark Utility, public IP address support
Simplified cluster authentication, SOC compliance, Kubernetes 1.13, PodSecurityPolicies
Container Insights, CNI 1.5.0, Amazon ECR, AWS PrivateLink support

aws

Pre-re:Invent - 2019

AWS App Mesh controllers for Kubernetes are now available as Helm Charts

Amazon EKS Increases Limits to 100 Clusters per Region

Amazon EKS now available in the Canada (Central) Region

Amazon EKS adds support for provisioning and managing Kubernetes worker nodes

AWS supports Automated Draining for Spot Instance Nodes on Kubernetes

Amazon EKS Generally Available in São Paulo Region

Amazon ECS Service Events Now Available as CloudWatch Events

ECS container monitoring now available in Amazon CloudWatch Container Insights

AWS launches FireLens, a log router for Amazon ECS and AWS Fargate

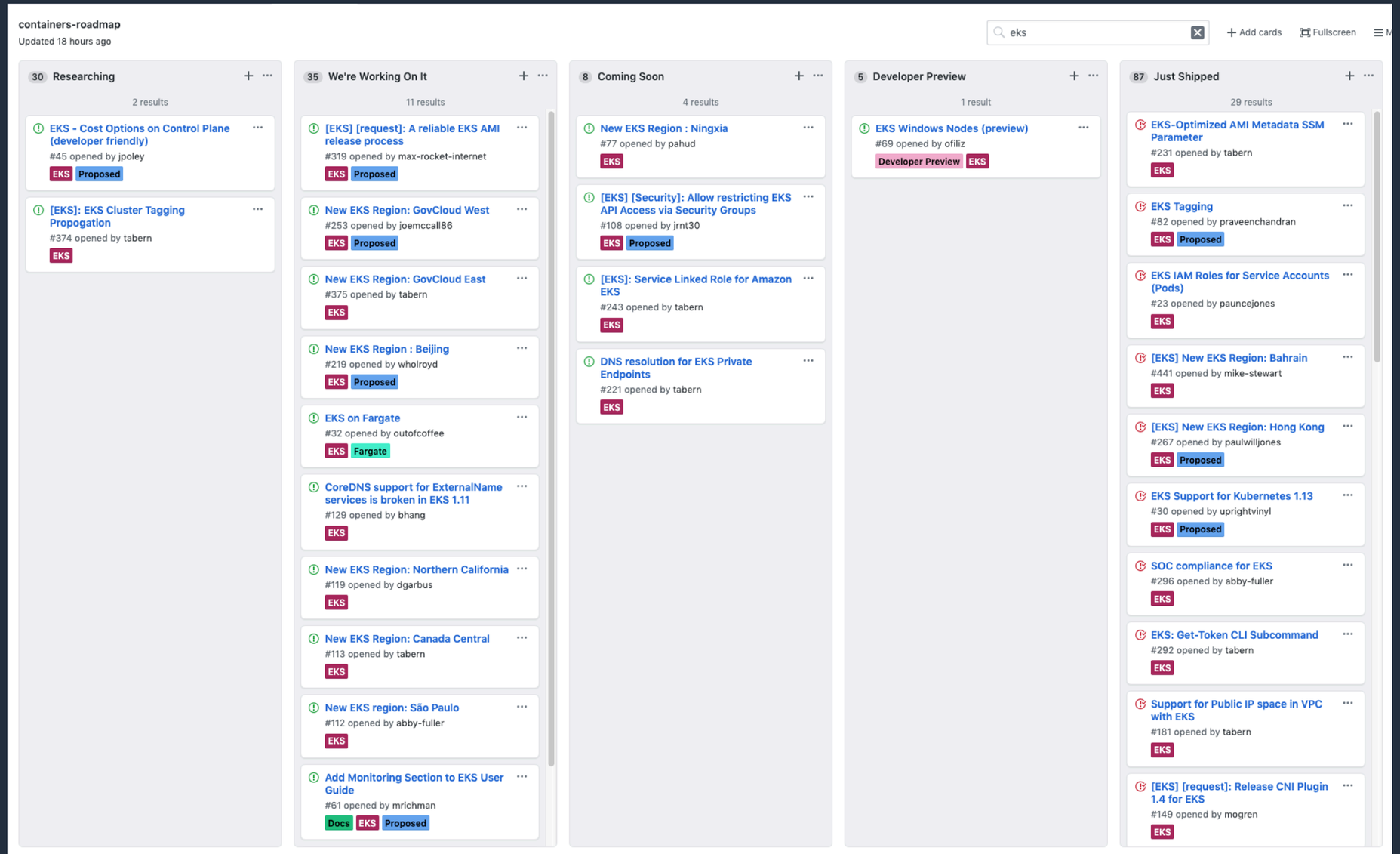
Amazon Elastic Container Service publishes multiple GitHub Actions

ECR events now published to EventBridge

aws

Open-source roadmap

https://github.com/aws/containers-roadmap/



Amazon EKS services roadmap: Highlights

Shipped

- Amazon EKS control plane logs
- Support for public IP space in VPC
- SOC compliance
- Amazon EKS: Deep Learning Benchmarking Utility
- New Amazon EKS Regions: Paris, London, Mumbai
- CNI v1.5.0

Shipped

- Amazon EKS support for K8s version 1.15 + ECR AWS PrivateLink
- Amazon EKS and KMS integration
- New Amazon EKS Regions: Beijing, Hong Kong, Sao Paulo, Canada Central
- Managed Worker Nodes / Fargate
- DNS resolution of Amazon EKS private endpoints

Working on it

- Managed add-ons
- New Amazon EKS Regions: Ningxia
- Next-generation CNI plugin
- New UI

aws

Global availability

Americas

Virginia, Ohio, Oregon, Canada

EMEA

Ireland, Frankfurt, London, Paris, Stockholm

Asia Pacific

Bahrain, Hong Kong, Singapore, Tokyo, Sydney, Seoul, Mumbai

aws

Service level agreement

99.95%

Service commitment

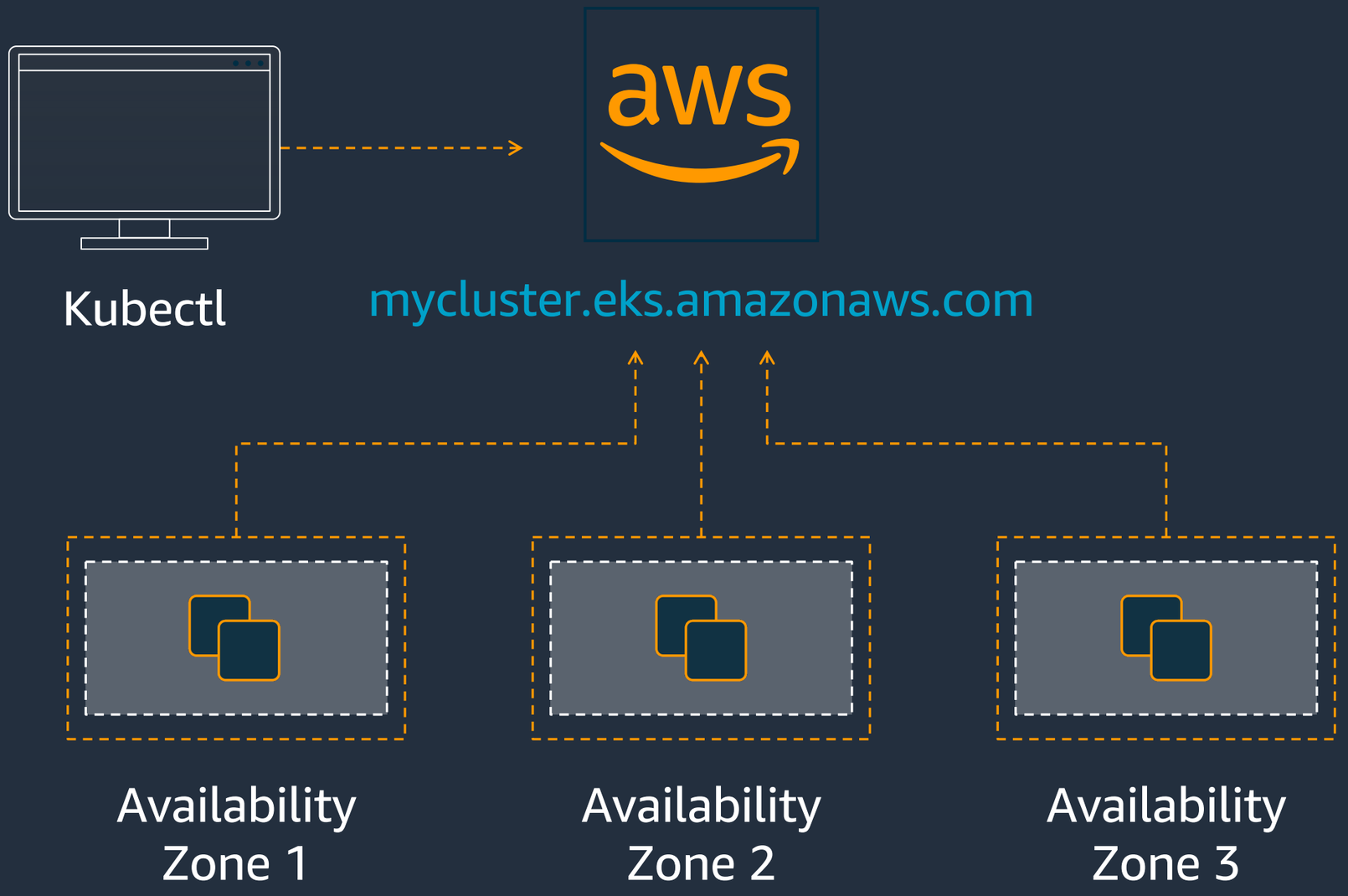
AWS will use commercially reasonable efforts to make the endpoint for an Amazon EKS cluster available with a monthly uptime percentage of at least 99.95% during any monthly billing cycle.

In the event Amazon EKS does not meet the monthly uptime percentage commitment, you will be eligible to receive a Service Credit.

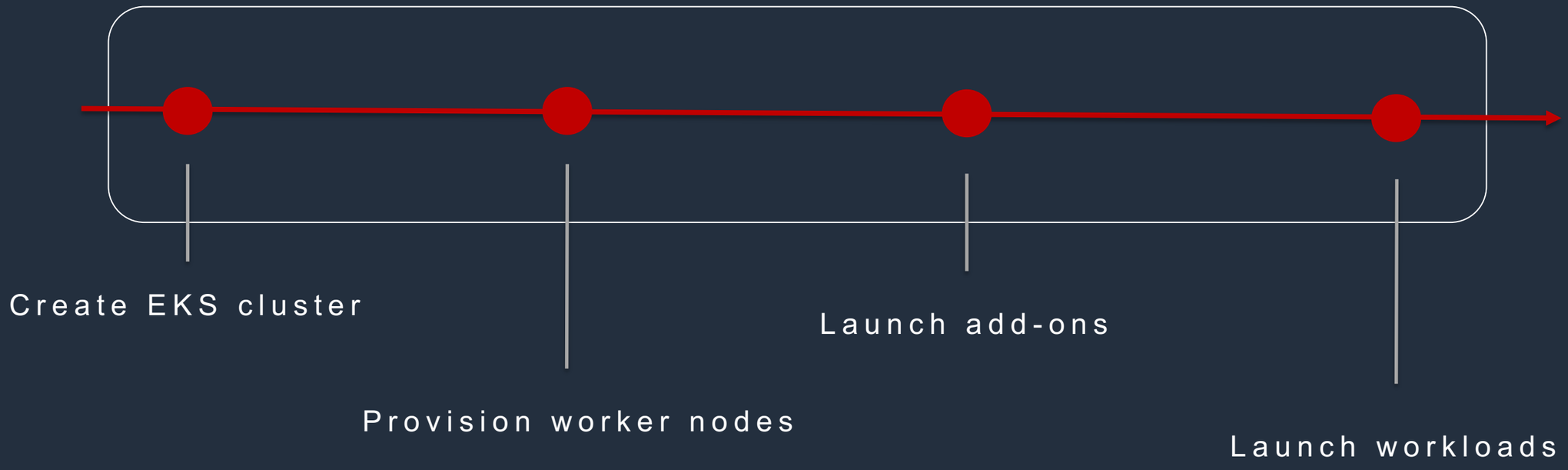
aws

EKS Architecture Overview

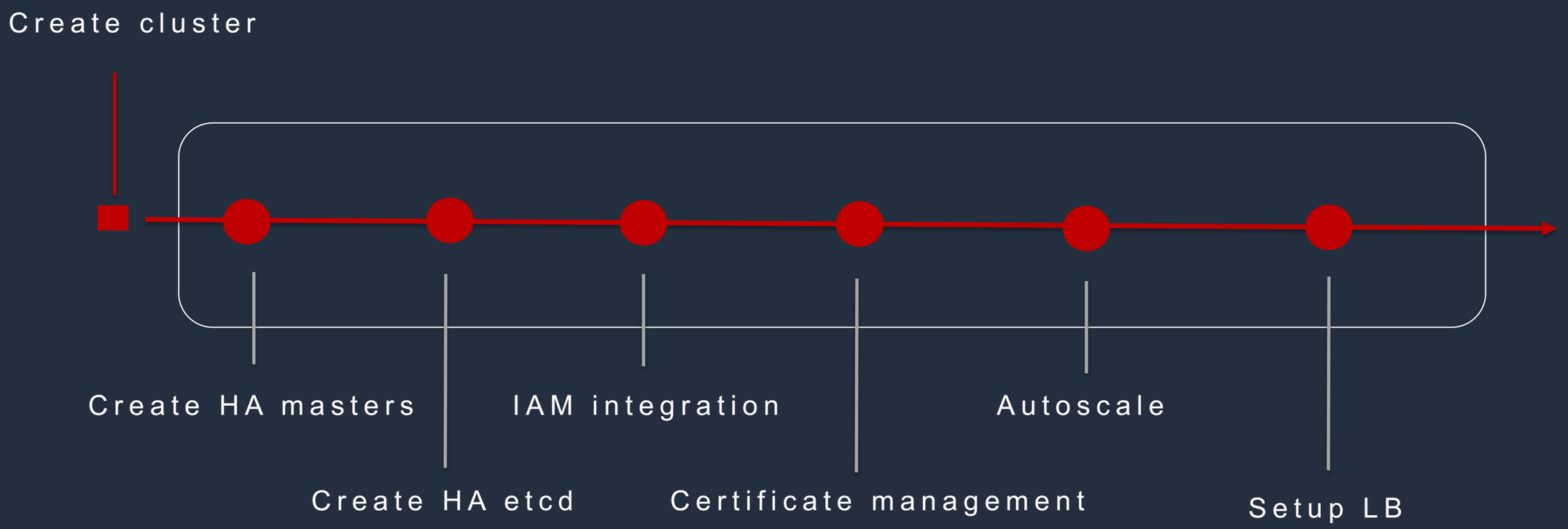
aws



EKS Customers

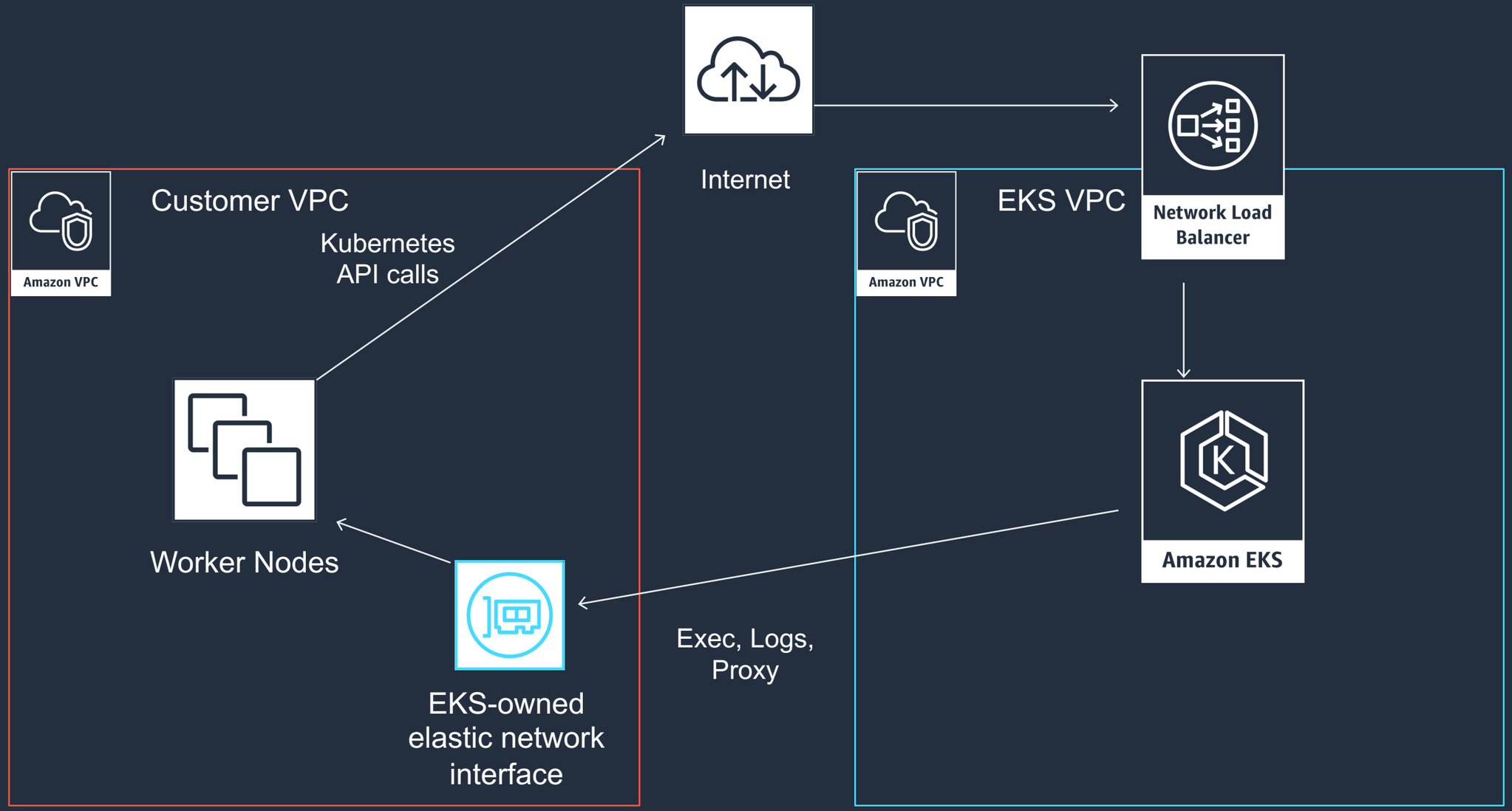


EKS – Kubernetes masters

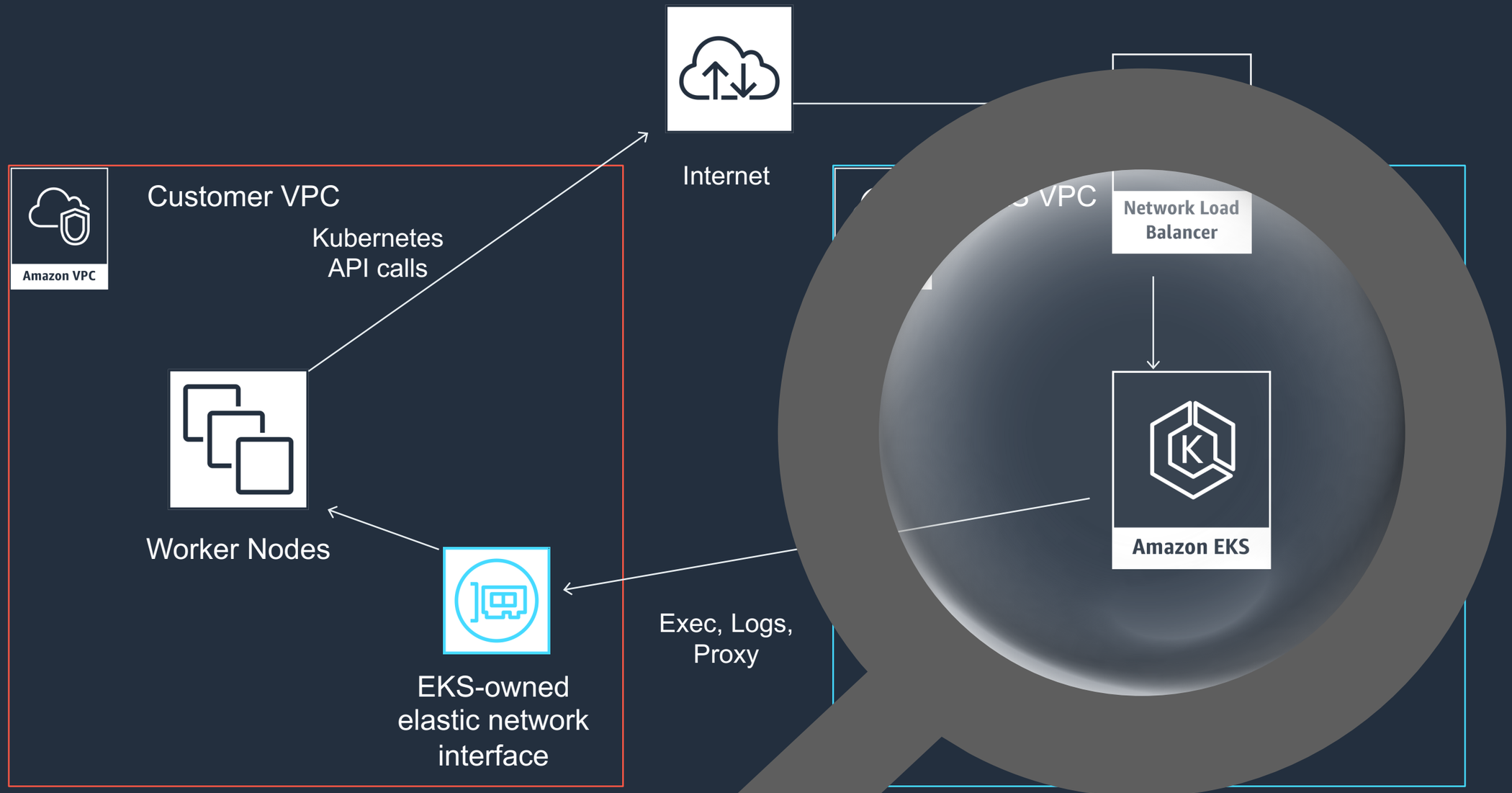


aws

Amazon EKS Architecture



Amazon EKS Architecture

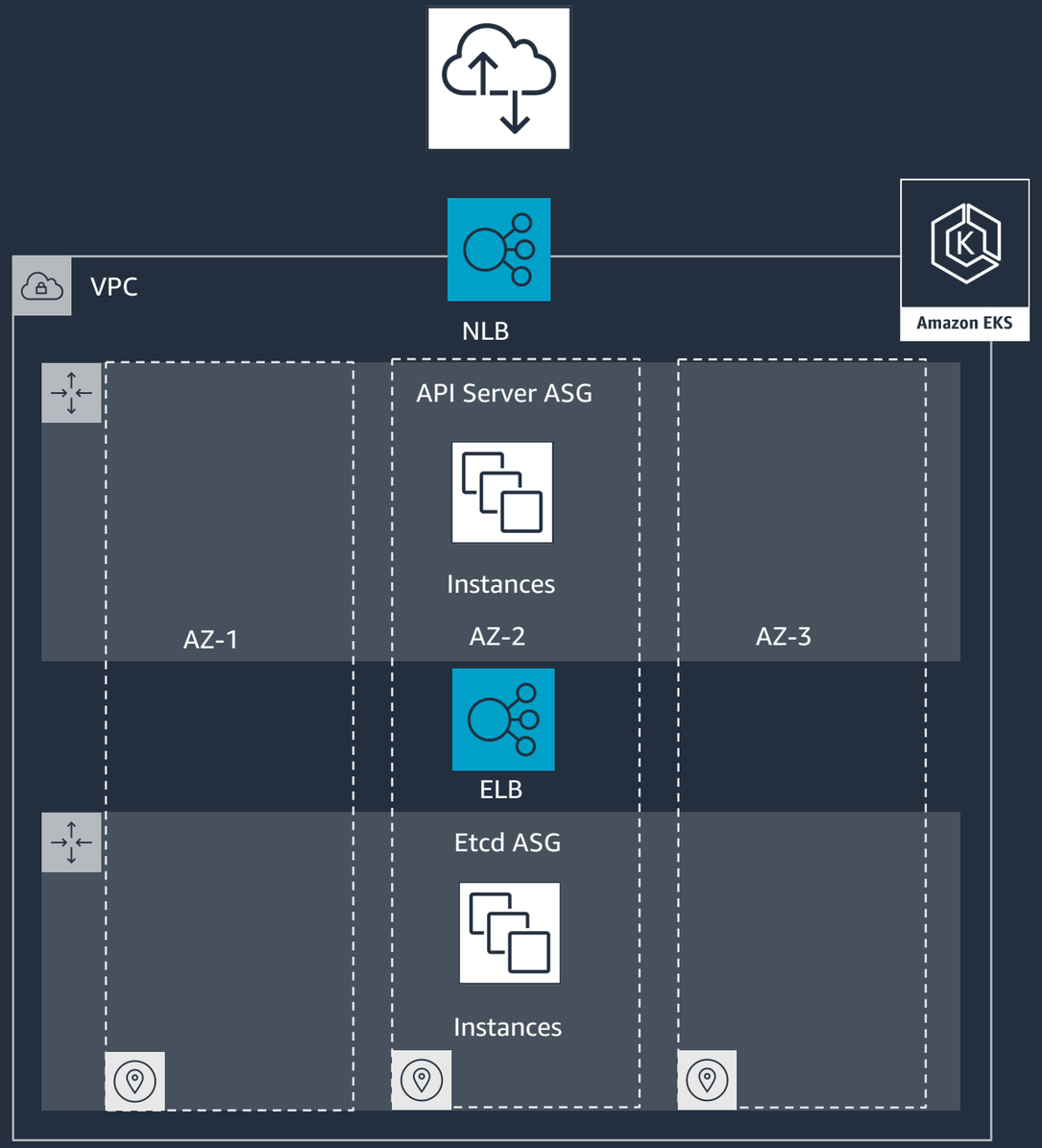


Kubernetes Control Plane

Highly available and single tenant infrastructure

All “native AWS” components

Fronted by an NLB



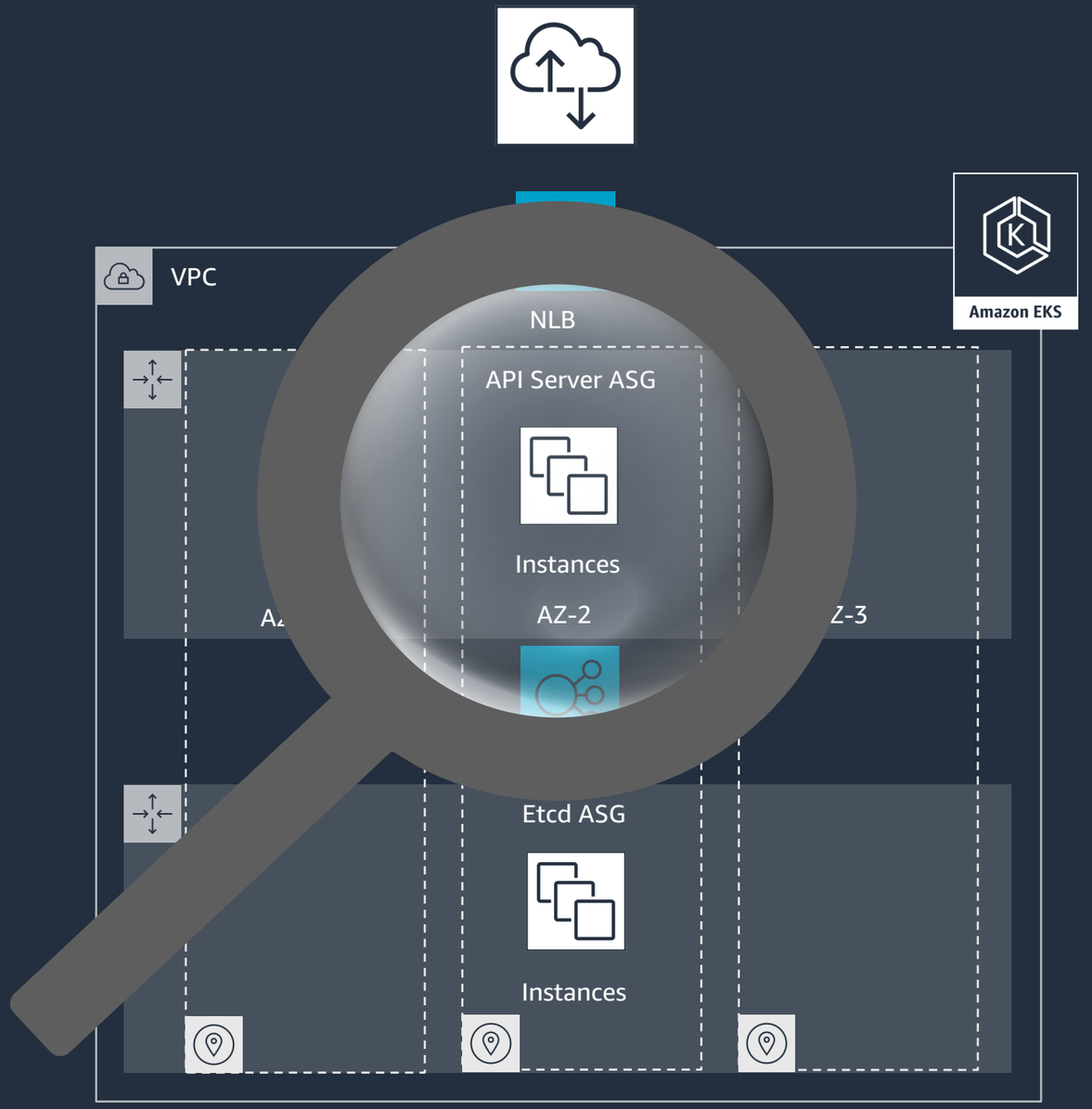
aws

Kubernetes Control Plane

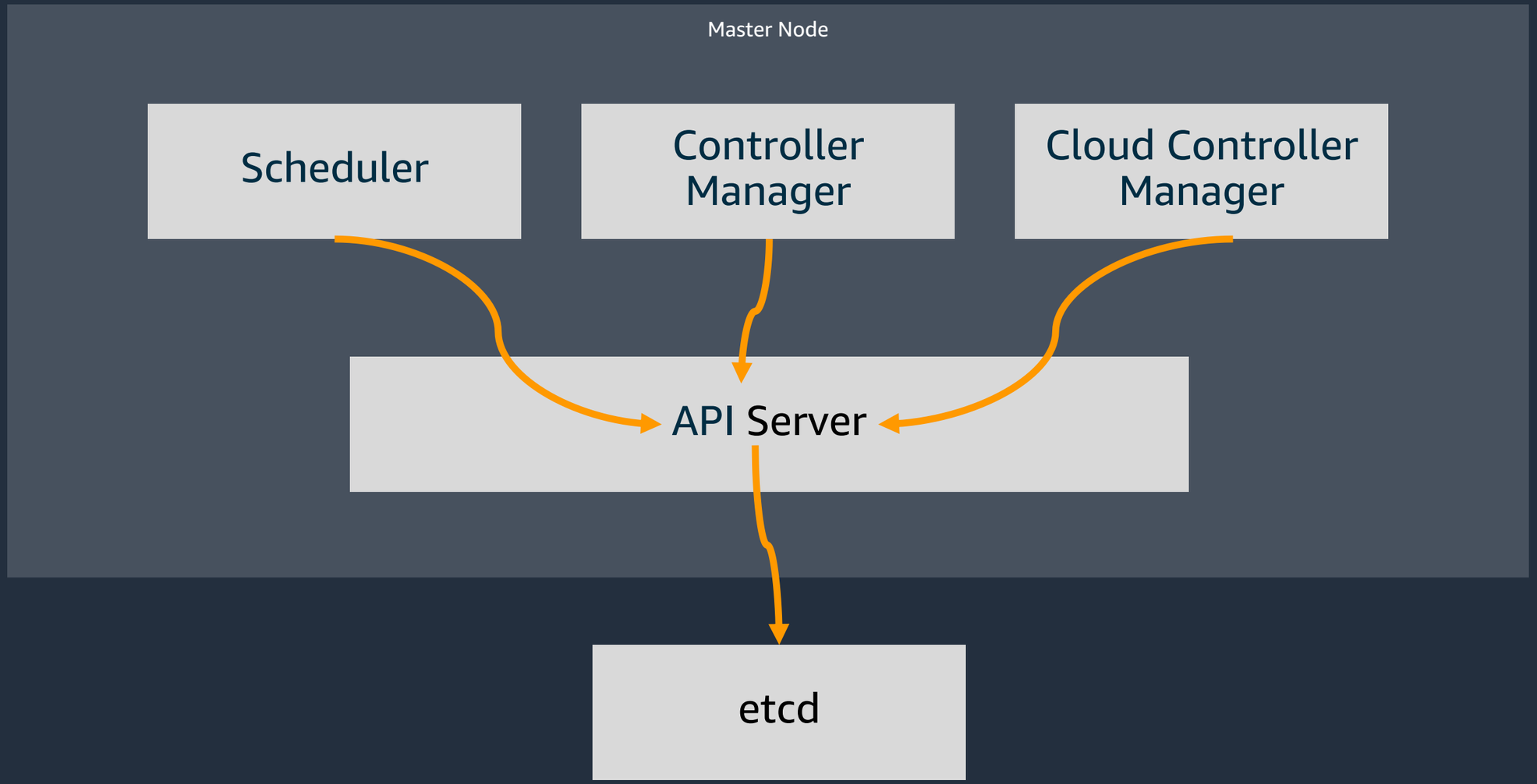
Highly available and single tenant infrastructure

All “native AWS” components

Fronted by an NLB

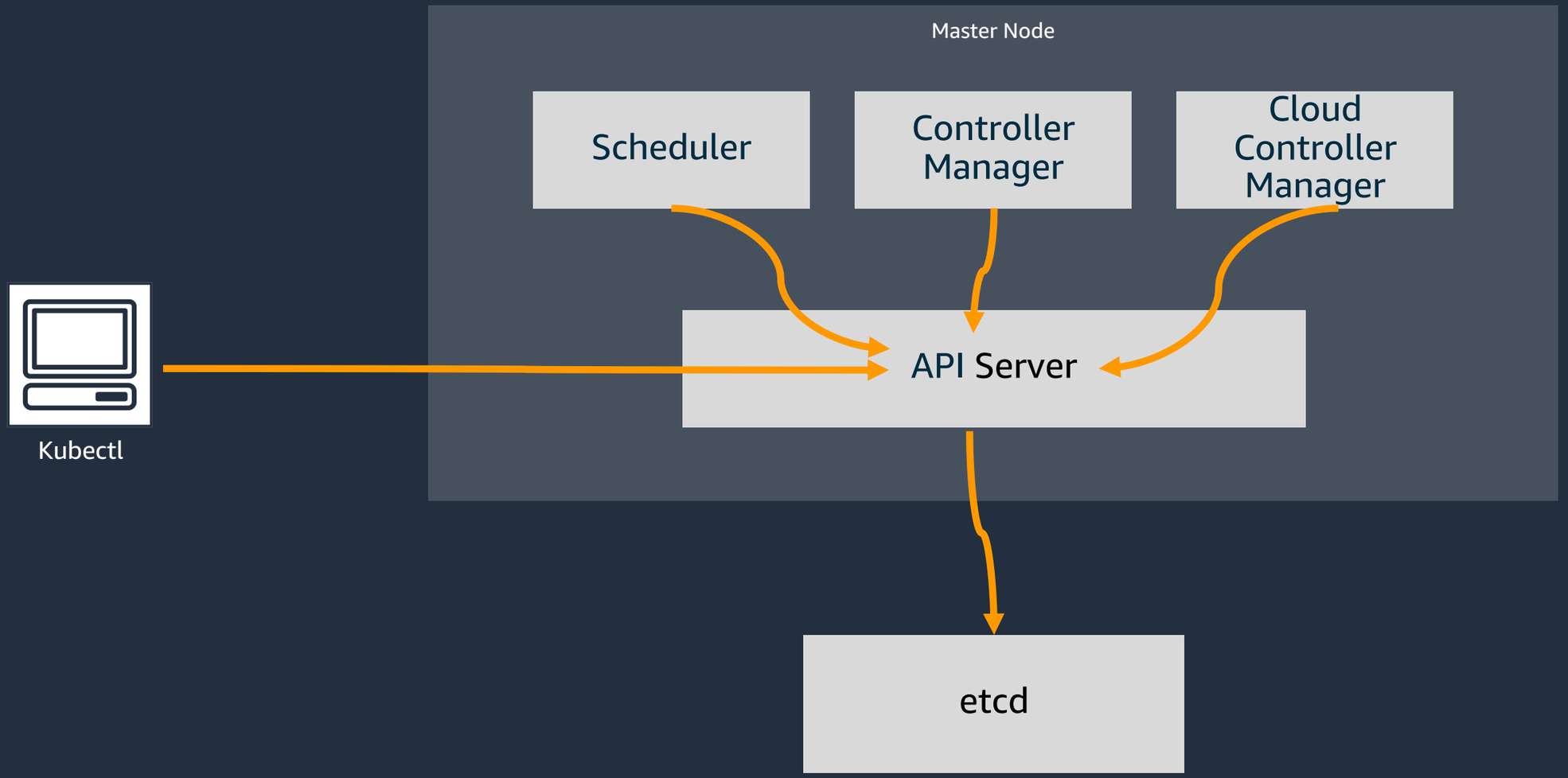


Kubernetes Control Plane



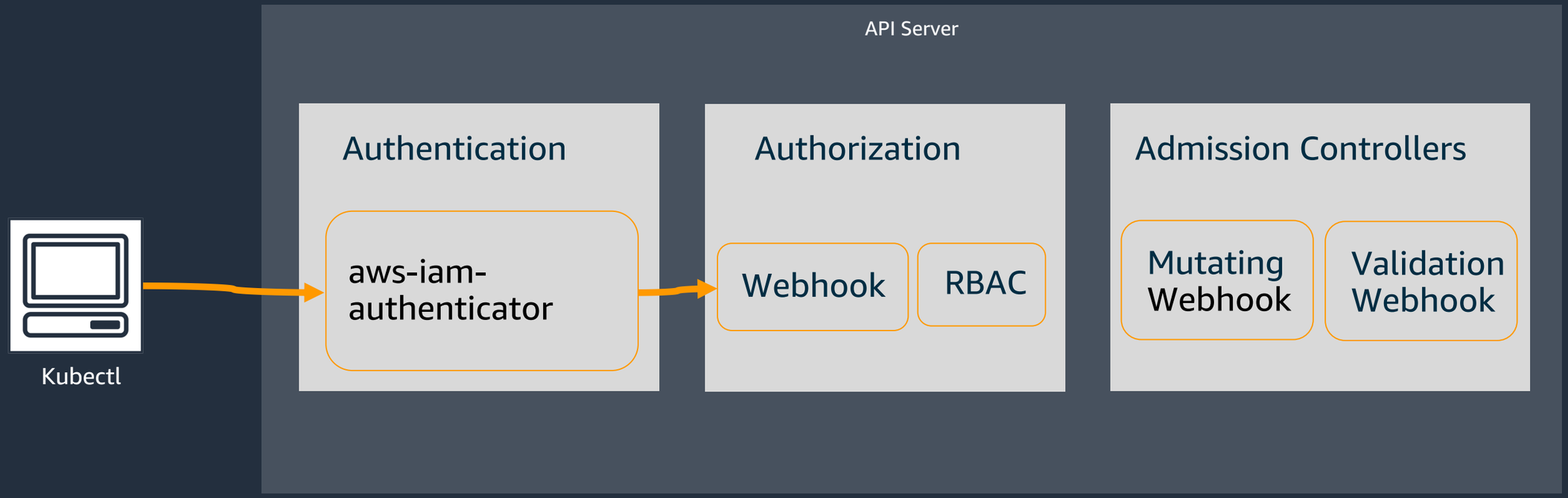
aws

Kubernetes Control Plane



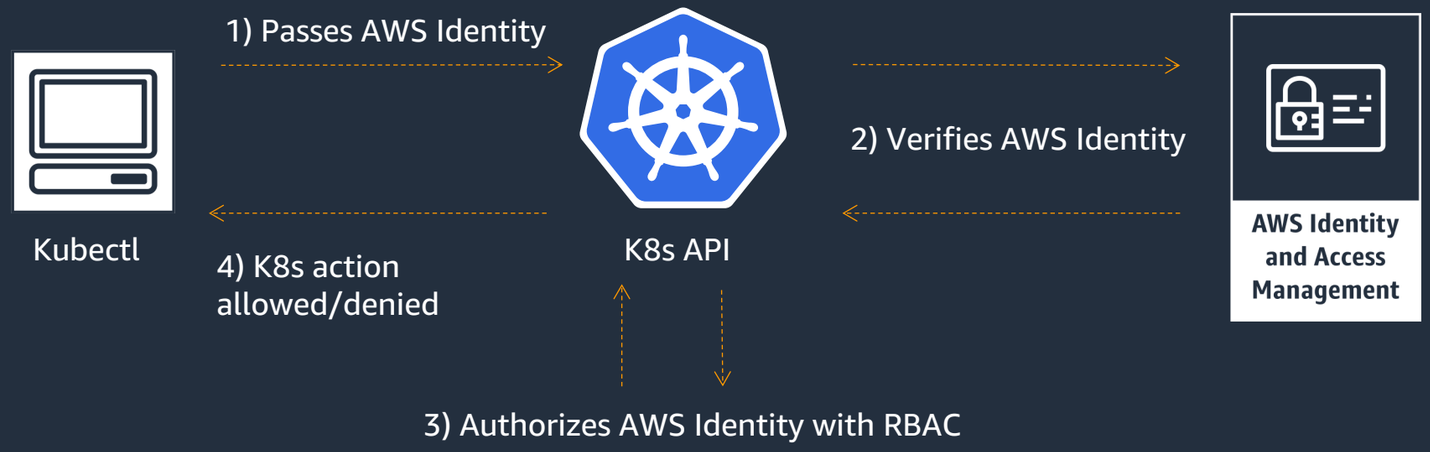
aws

Kubernetes API Server



aws

IAM Authentication



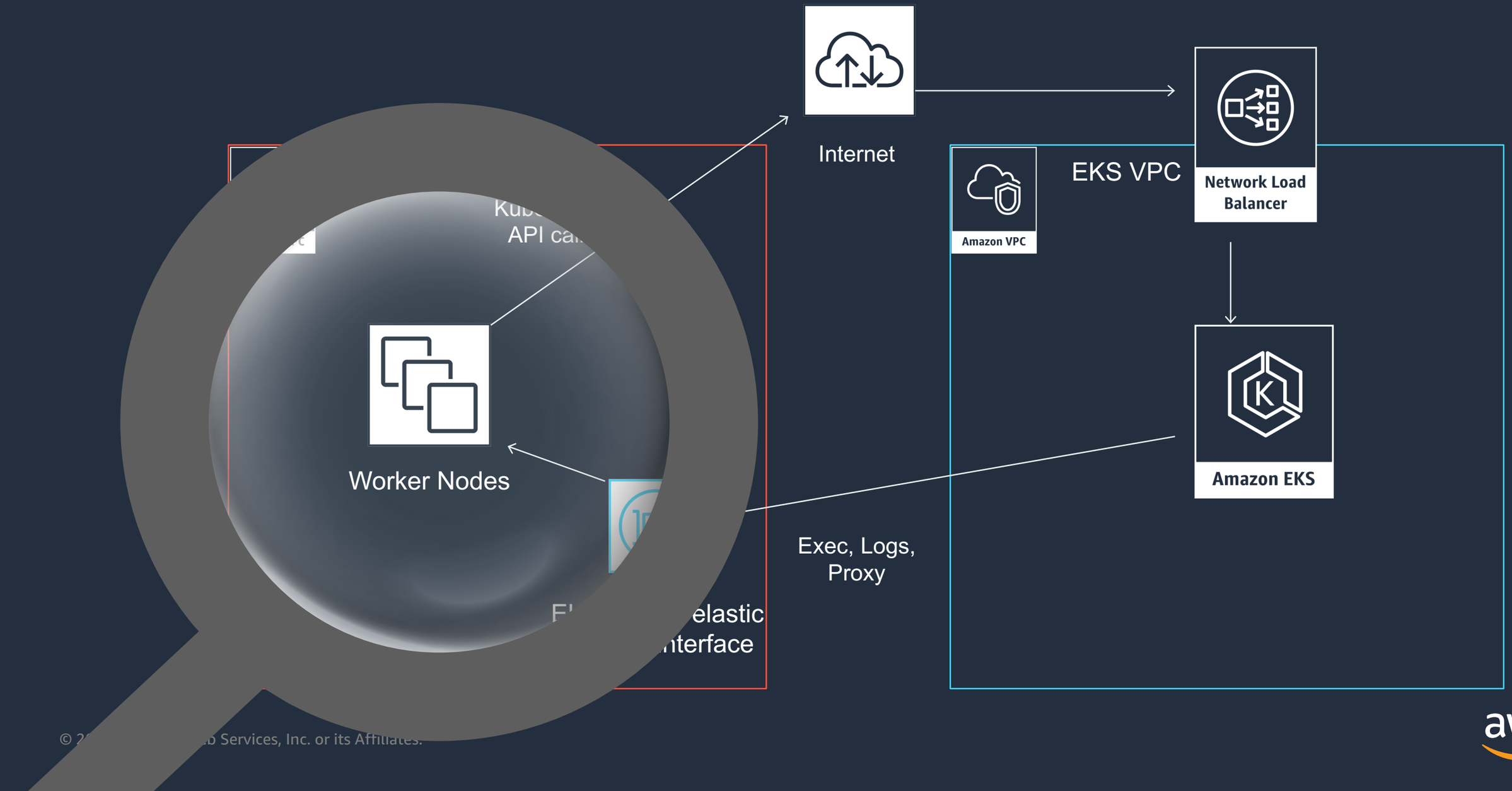
Cluster Authentication and Authorization

- User or IAM role who **creates** Amazon EKS cluster gains Admin privileges
- This {“super”} user/role can then add additional users or IAM roles and configure RBAC permissions
- To add, configure aws-auth Configmap

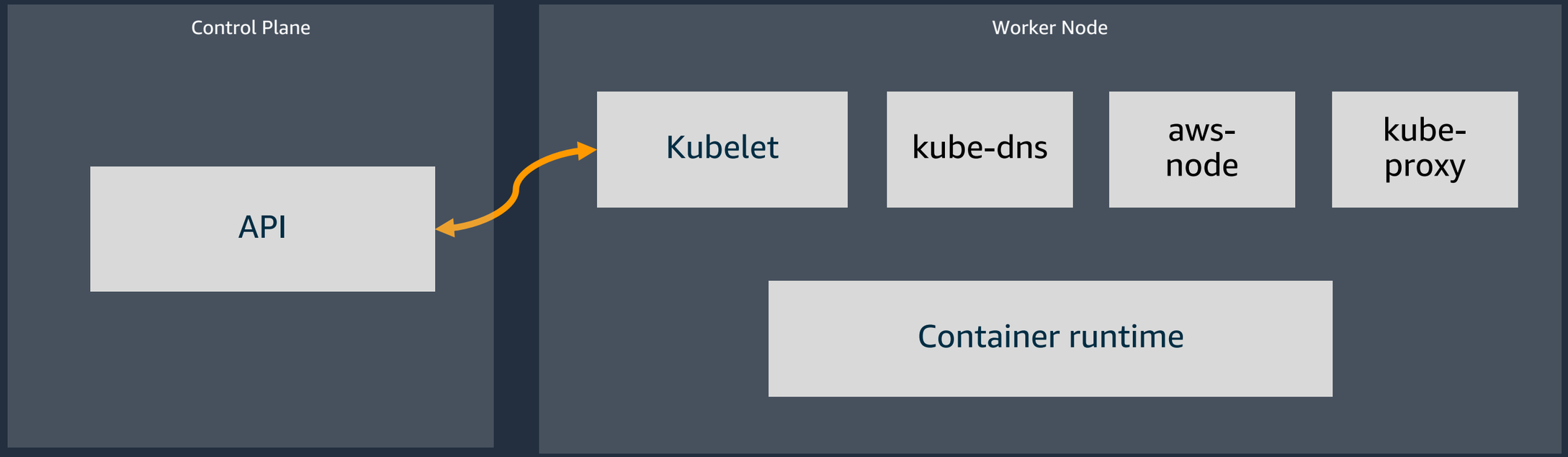
```
kubectl edit -n kube-system configmap/aws-auth
```


aws

Kubernetes Data Plane



Kubernetes Data Plane



aws

EKS and Fargate

aws

Containers

General Availability

Introducing

Amazon EKS for AWS Fargate

The only way to run serverless Kubernetes containers securely, reliably, and at scale

Simplified deployment, management, and scaling of Kubernetes on AWS

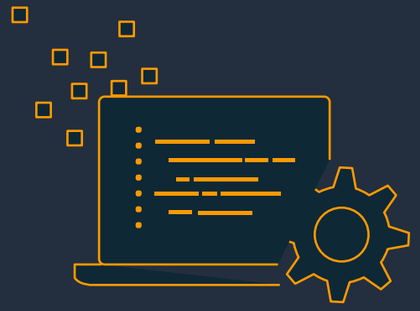
Strong security isolation for every pod by default

Focus on building applications

LEARN MORE	CON-326R - Running Kubernetes Applications on AWS Fargate

aws

Make Kubernetes apps serverless with Amazon EKS + Fargate



Bring existing pods

You don’t need to change your existing pods. Fargate works with existing workflows and services that run on Kubernetes.

Production ready

Launch ten or tens of thousands of pods in seconds. Easily run pods across multiple AZs for high-availability.

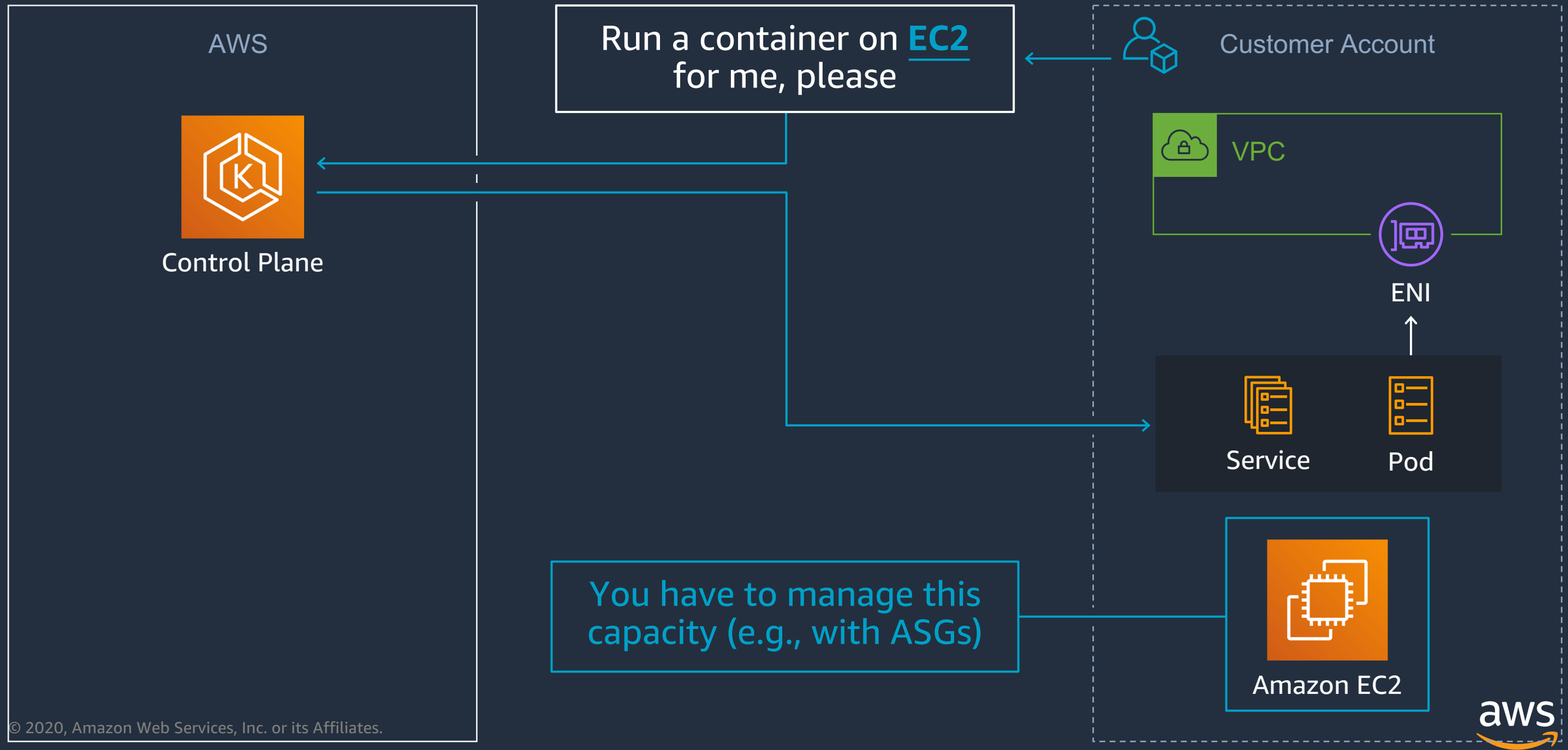
Right-Sized and Integrated

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

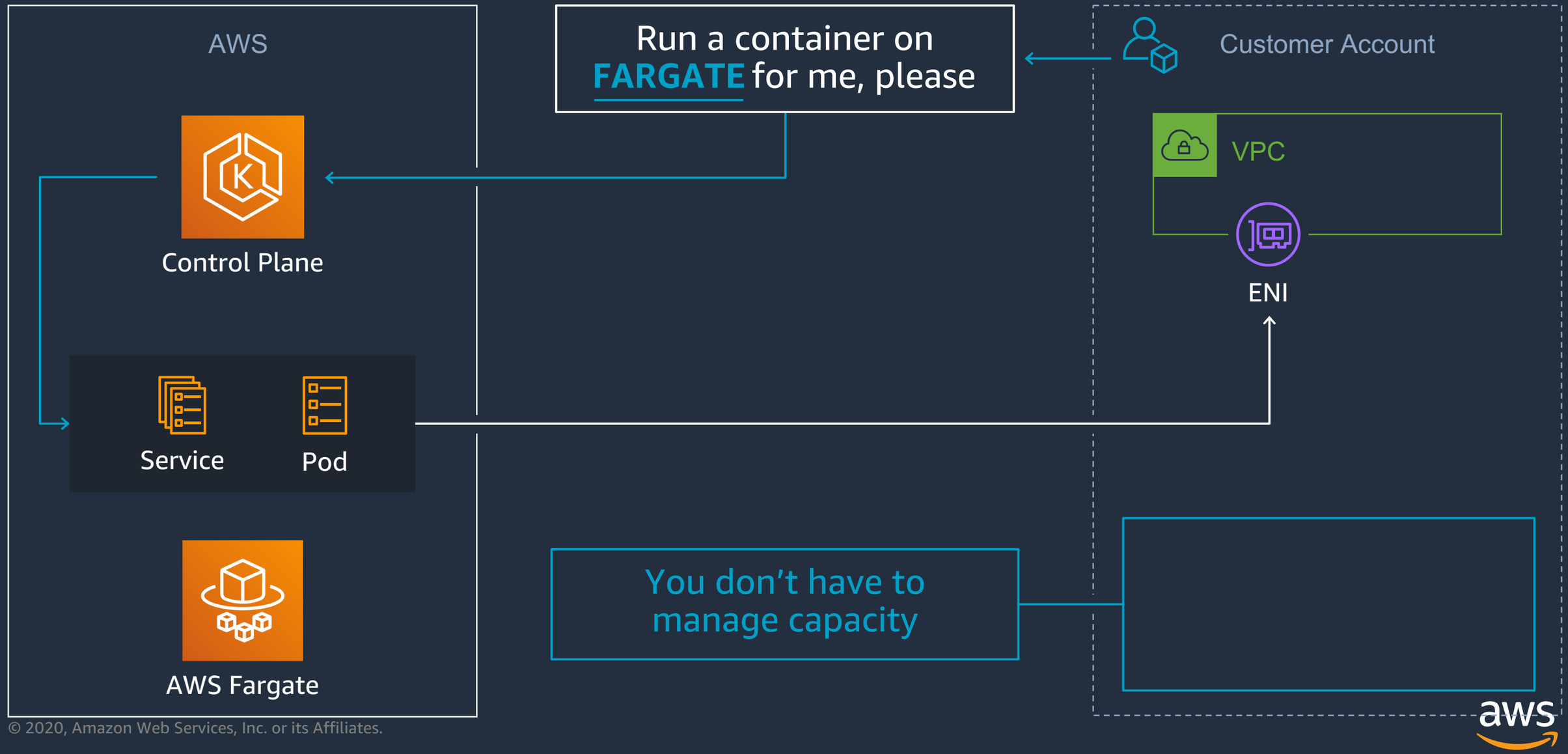
Fargate runs tens of millions of containers for AWS customers every week

aws

The EC2 flow at 33,000 feet

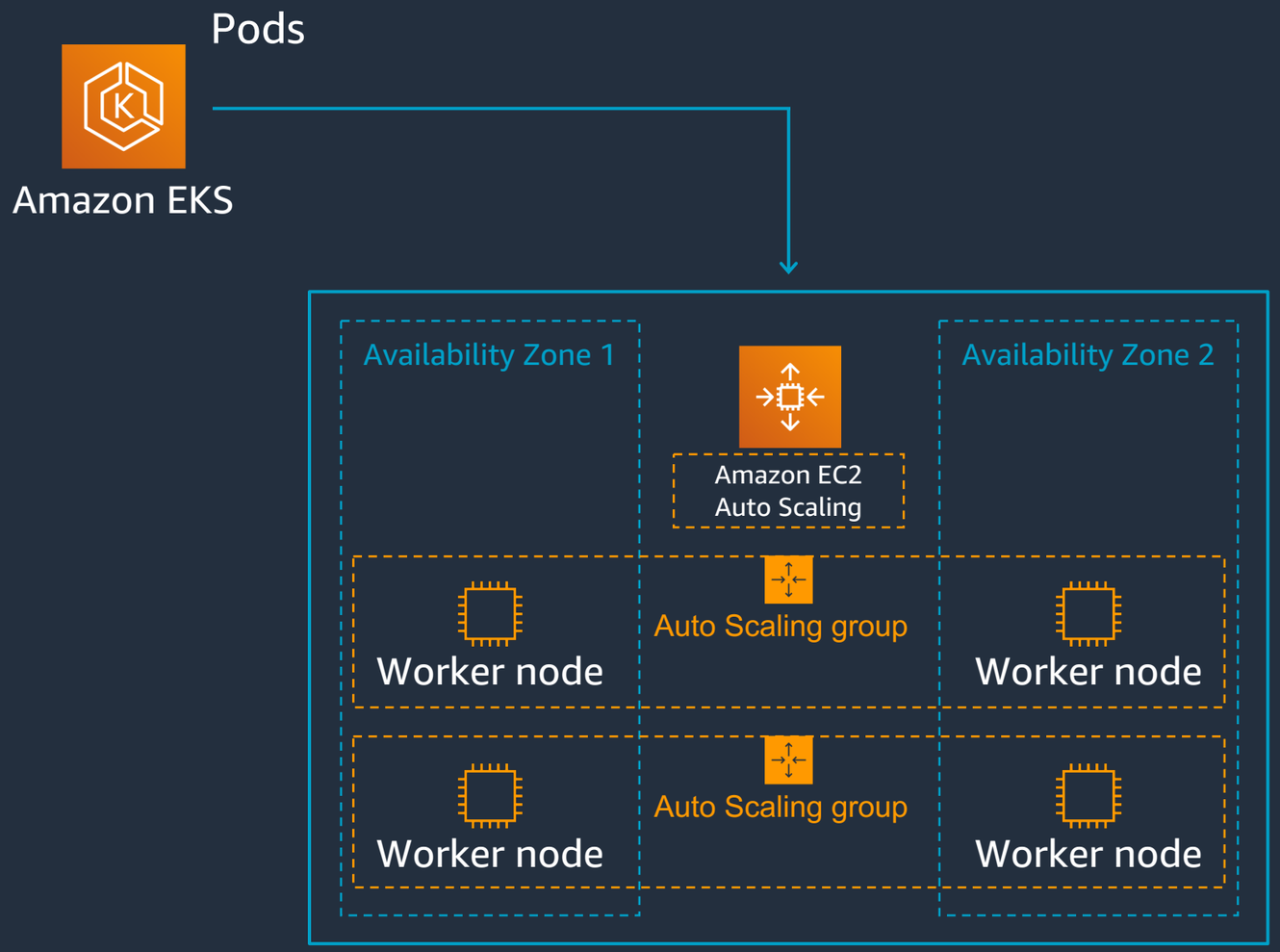


The Fargate flow at 33,000 feet



EKS data plane options

Worker nodes only

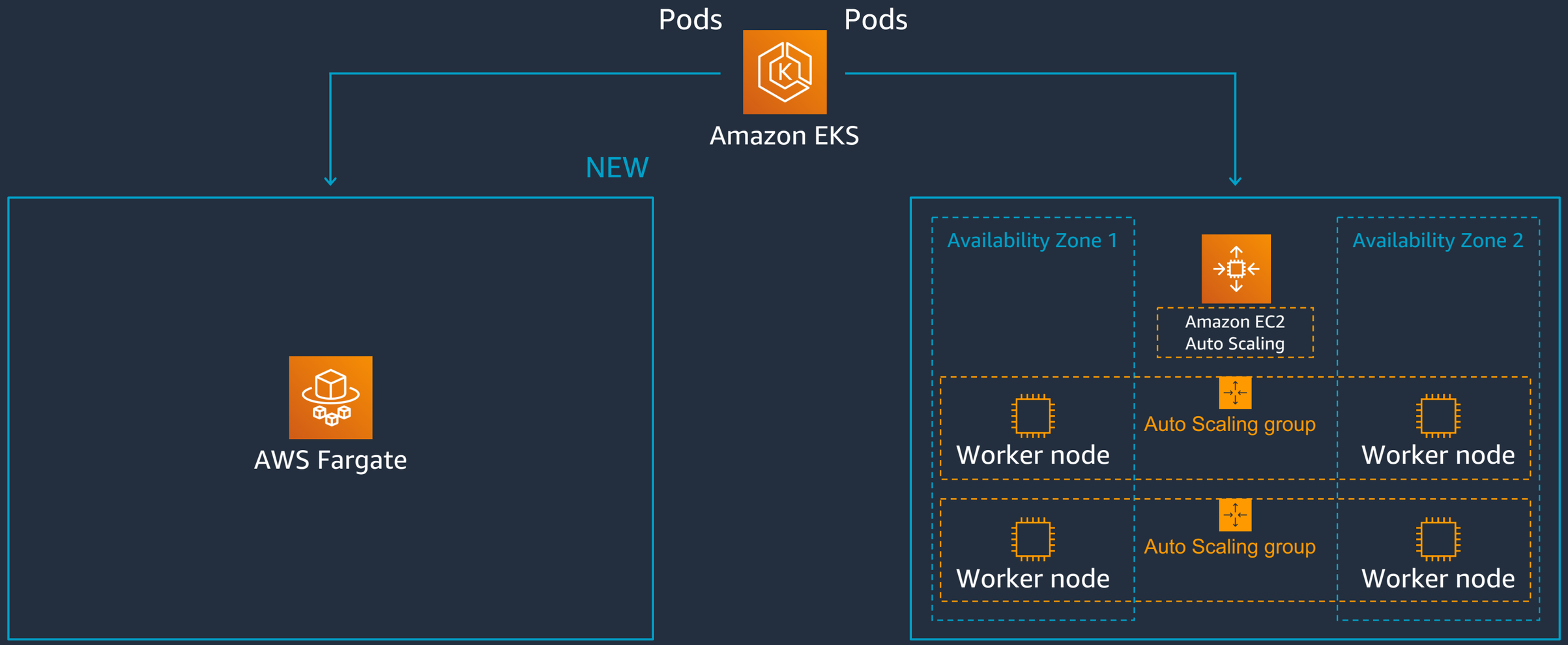


Traditional container data plane

aws

EKS data plane options

Mixed mode



Serverless container data plane

Traditional container data plane

Fargate Vs. (Un)Managed Nodes

	Fargate	Managed nodes	Unmanaged nodes
Units of work	Pod	Pod and EC2	Pod and EC2
Unit of charge	Pod	EC2	EC2

aws

Fargate Vs. (Un)Managed 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

aws

Fargate vs. (Un)Managed 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

aws

Recap: EKS for Fargate introduces UX changes

Things you no longer need to do

- Manage Kubernetes worker nodes
- Pay for unused capacity
- Use K8s Cluster Autoscaler (CA)

Things you get out of the box

- VM isolation at pod level
- Pod level billing
- Easy chargeback in multi-tenant scenarios

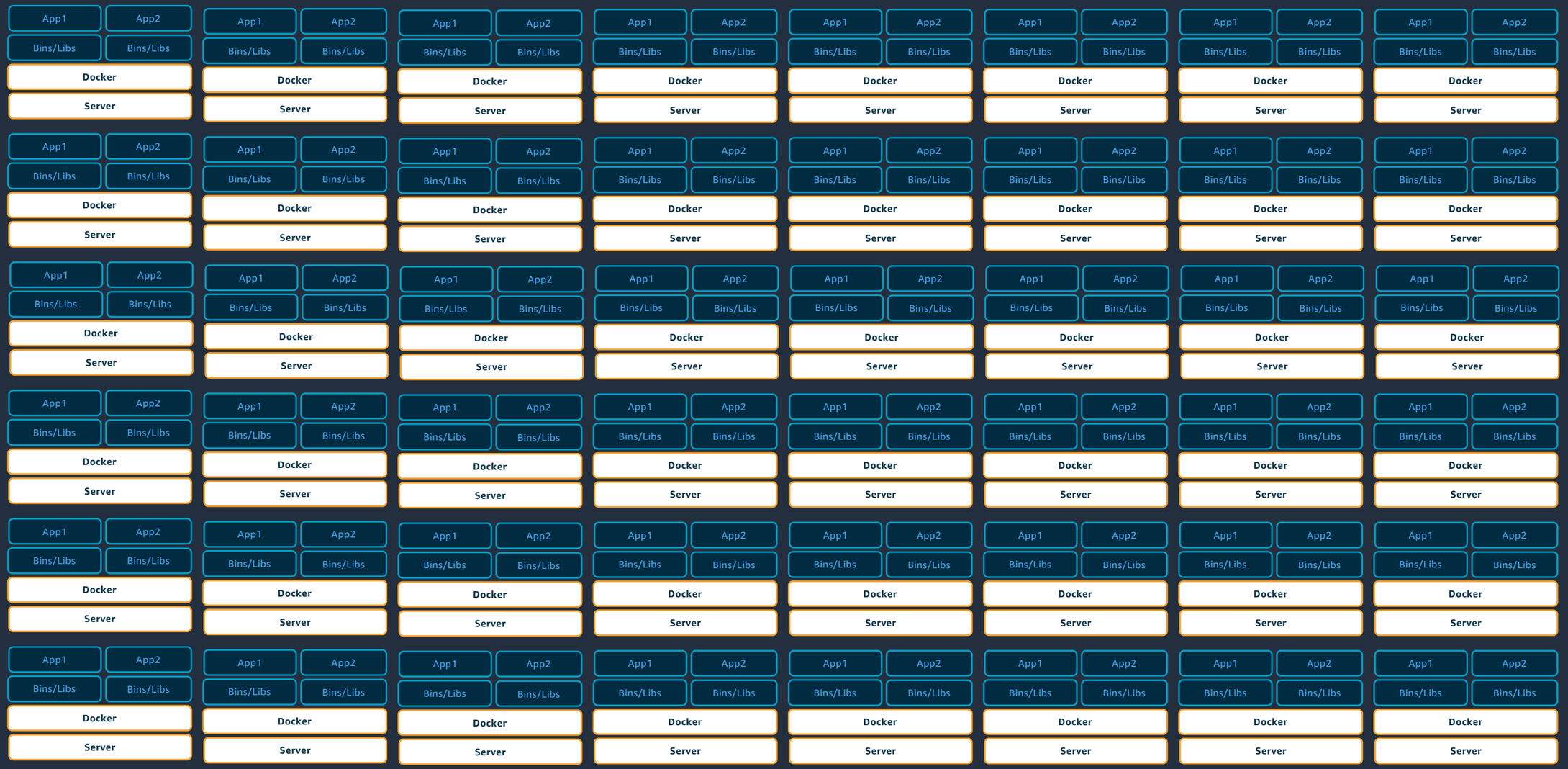
Things you can't do (for now)

- Deploy Daemonsets
- Use service type LoadBalancer (CLB/NLB)
- Running privileged containers
- Run stateful workloads

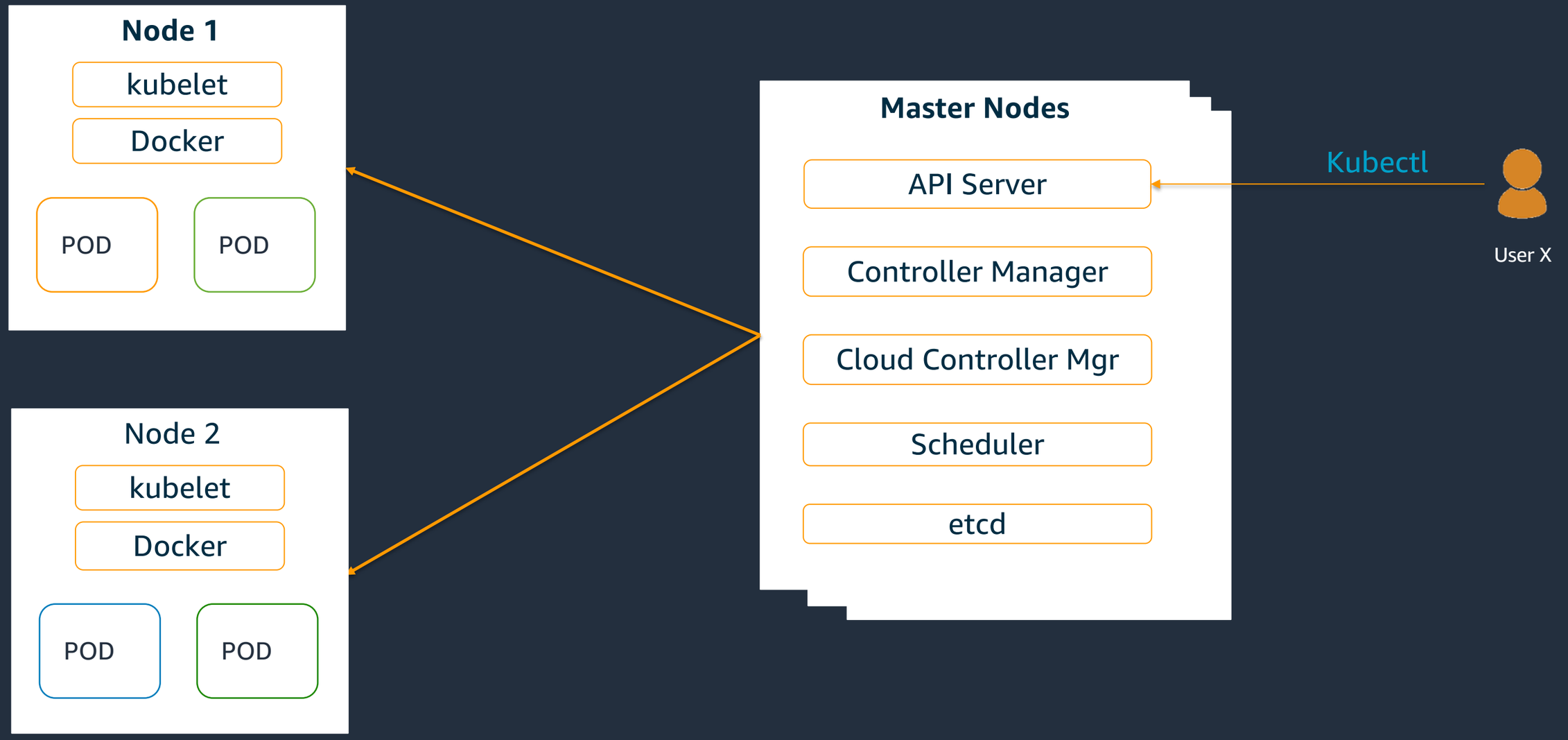
aws

Foundations to Deploy Microservices

Containers



Kubernetes Architecture



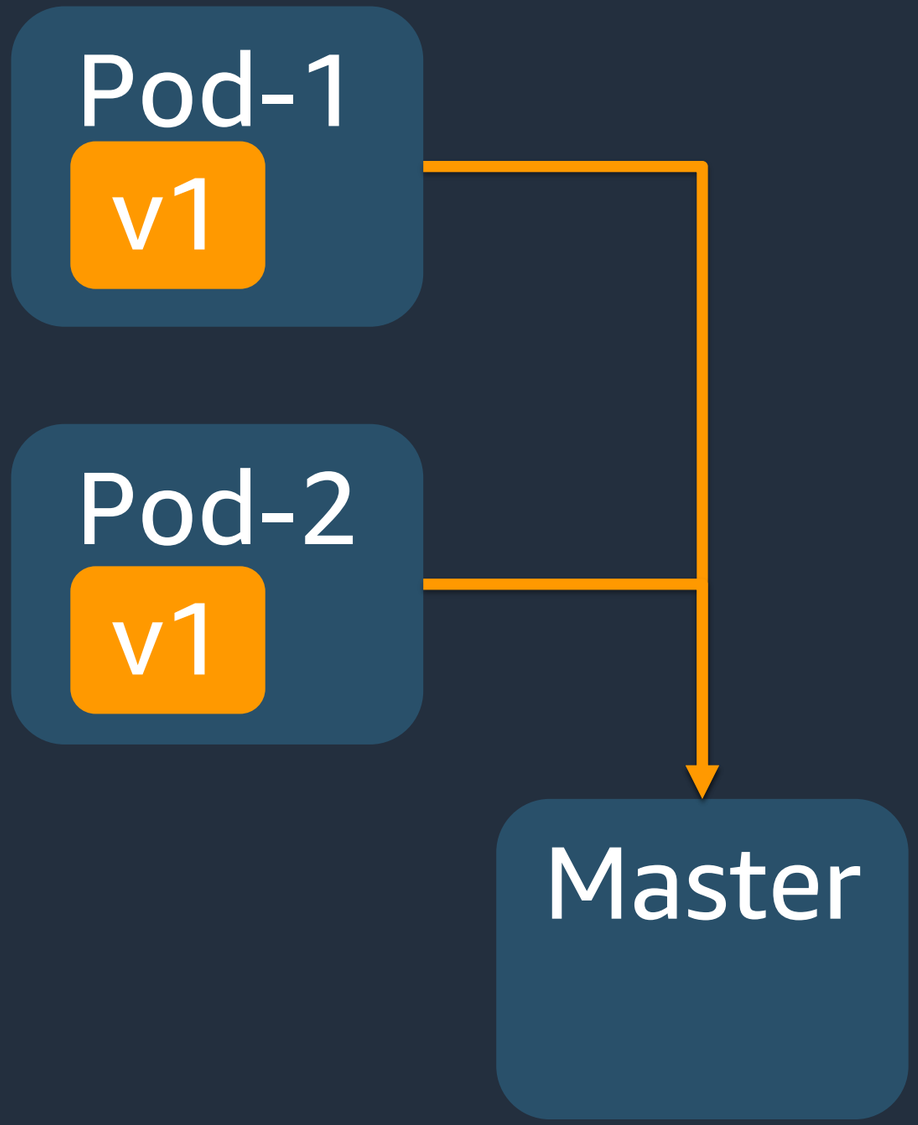
Worker Nodes

Control Plane

aws

What is a Kubernetes Pod?

- Smallest unit of deployment
- Grouping of containers
- Share storage and networking
- Unique IP per pod
- Co-located and co-scheduled on nodes



aws

Kubernetes Core Concepts

Manifest File - YAML/JSON used to deploy Kubernetes objects

Deployment – Run specified # of Pods of your application

Service - Maps a fixed IP address to a logical group of pods

Annotation - Key/Value pairs to hold non-identifying information

Label - Key/Value pair used for association and filtering

DaemonSet - Implements a single instance of a pod on a worker node

aws

Deploying Applications

Application Implementation Overview

- Create required container image & image repo / use existing
- Deploy Pods across Worker Nodes
    - Create pod-manifest.yml file
    - Deploy from the kubectl node: **“kubectl apply –f pod-manifest.yaml”**
- Create an Ingress “Service” to route traffic to the Pods
    - Create svc-manifest.yml file
    - Deploy from the kubectl node: **“kubectl apply –f svc-manifest.yaml”**

aws

Example nginx-pods.yaml

```
…
kind: Deployment
replicas: 2
  template:
    metadata:
      labels:
        app: nginx
spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

Create a “ReplicaSet” containing 2 “Pods”

App Name label

Container Image

Listener Port

Implement from kubectl node with one command:

“kubectl apply –f nginx-pods.yaml”

aws

Example nginx-svc.yaml (Classic Load Balancer)

```
…
kind: Service
spec:
 selector:
    app: nginx
  type: LoadBalancer
  ports:
  - name: http
    port: 80
    targetPort: 80
```

Route traffic to Apps named “nginx”

Deploy an AWS Load Balancer

Listener and Target Config

Implement from kubectl node with one command:

“kubectl apply –f. nginx-svc.yaml”

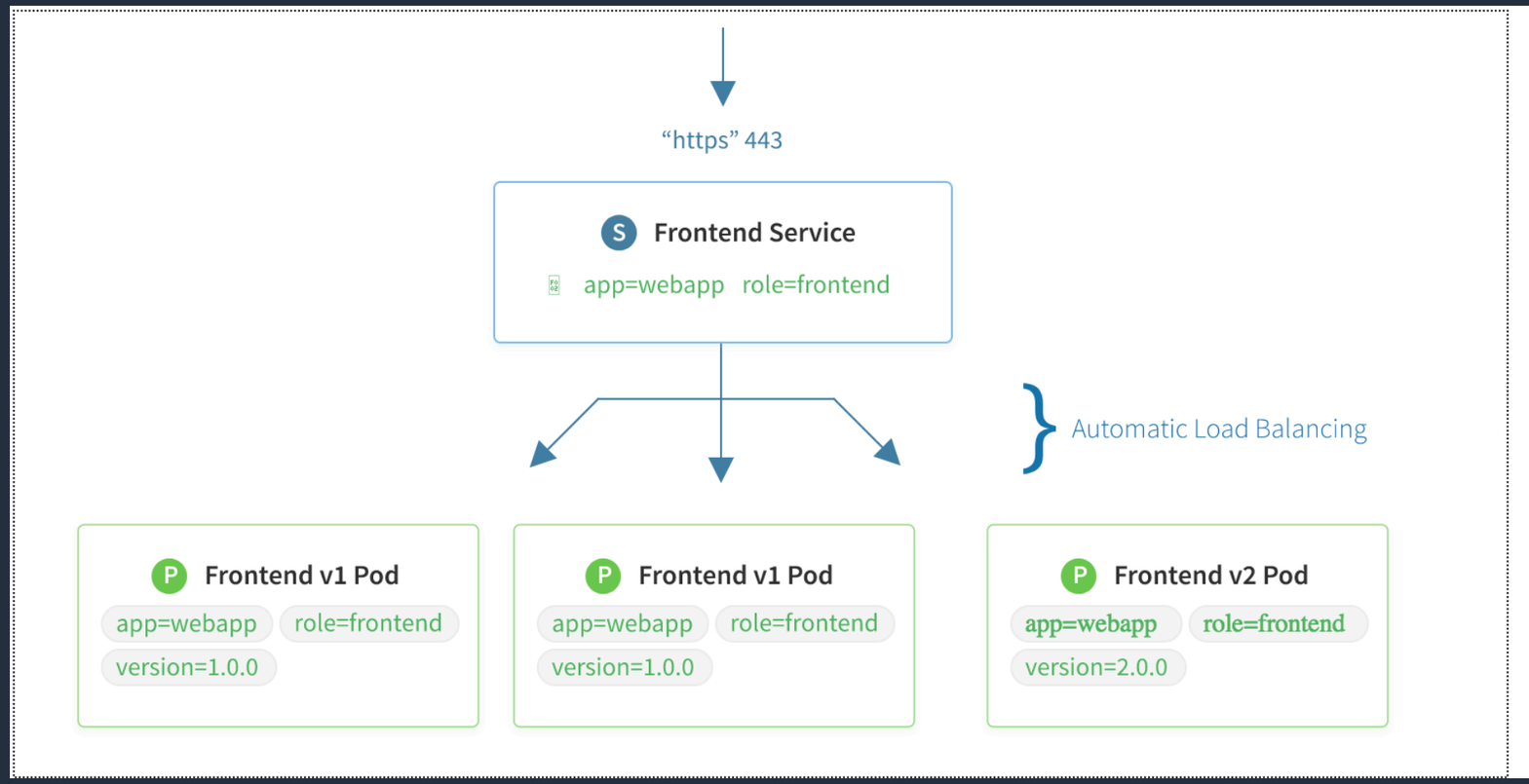
aws

Kubernetes Service Types and Ingress Options

aws

What is a Service?

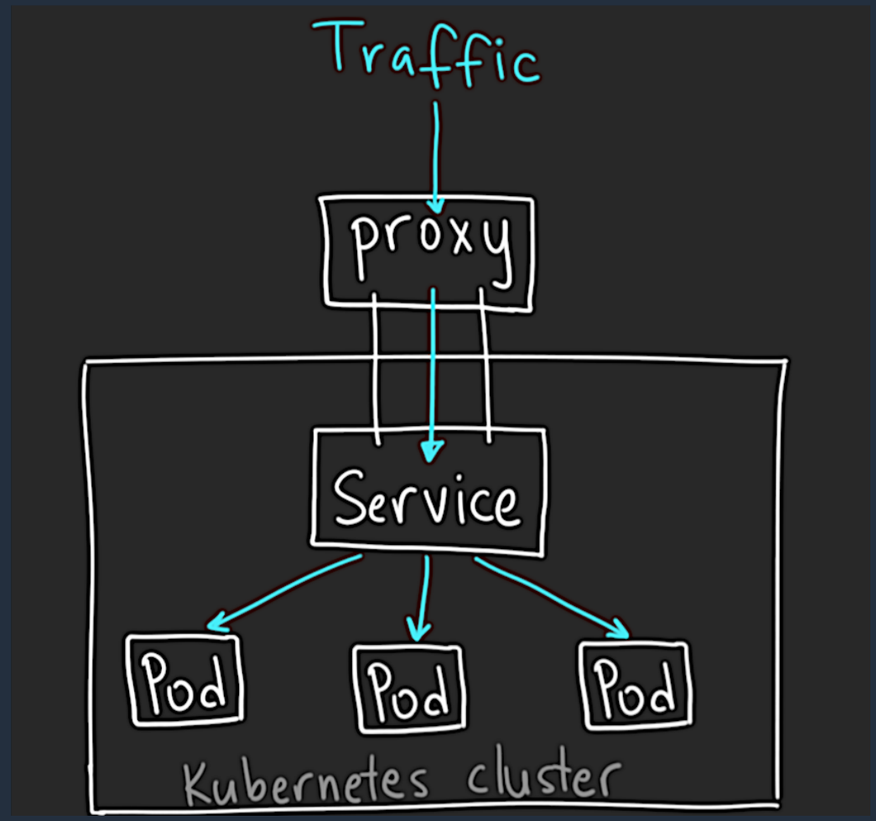
- Grouping of pods.
- Identifies the set of pods using a LabelSelector.
- Services can be exposed in different ways.



aws

ServiceType: ClusterIP

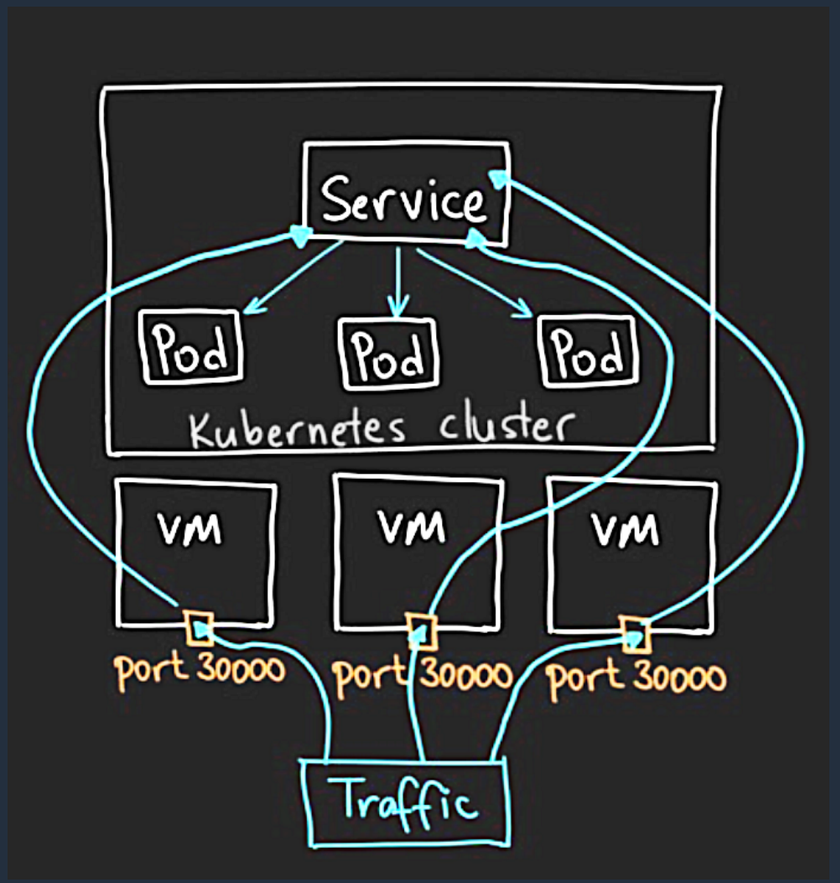
- Exposes the service on a cluster-internal IP
- Only reachable from within the cluster
- Access possible via kube-proxy
- Useful for debugging services, connecting from your laptop or displaying internal dashboards



aws

ServiceType: NodePort

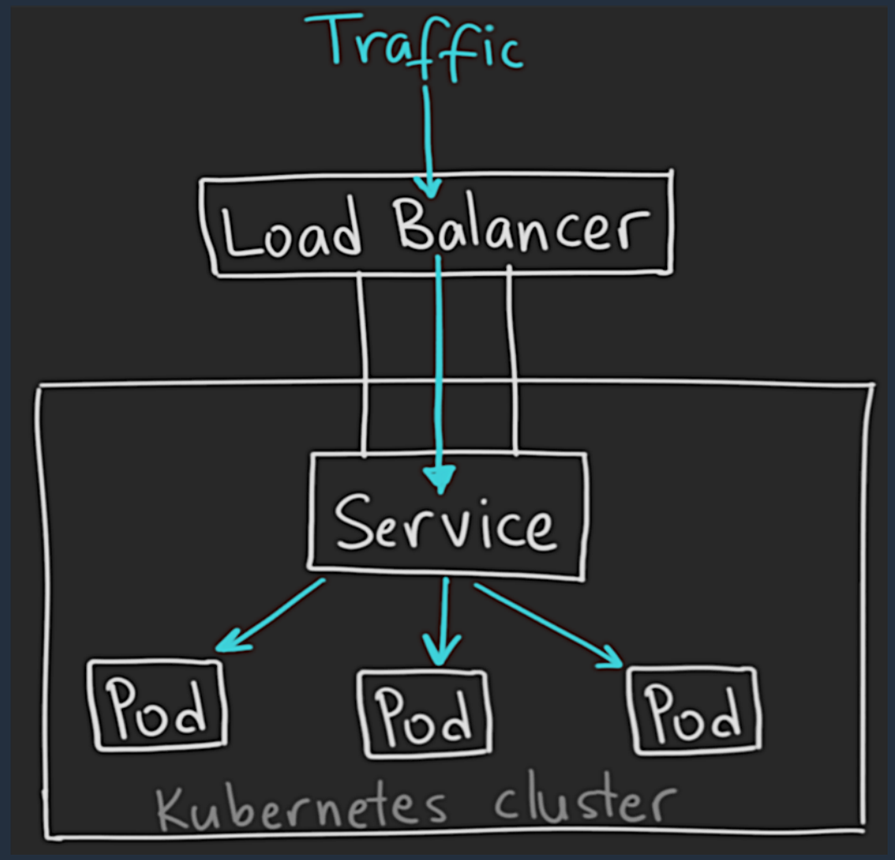
- Exposes the service on each Node's IP at a static port
- Routes to a ClusterIP service, which is automatically created.
- From outside the cluster: <NodeIP>:<NodePort>
- 1 service per port
- Uses ports 30000-32767



aws

ServiceType: LoadBalancer

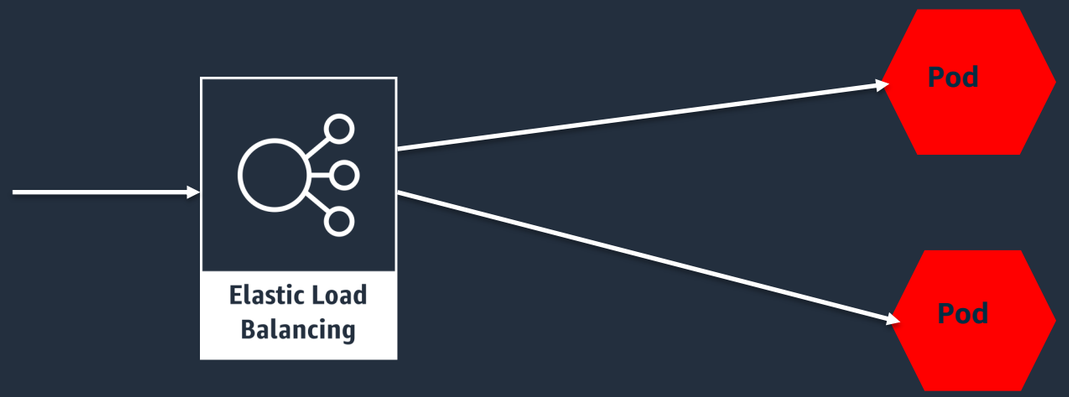
- Exposes the service externally using a cloud provider's load balancer
- NodePort and ClusterIP services (to which LB will route) automatically created
- Each service exposed with a LoadBalancer (ELB or NLB) will get its own IP address
- Exposes L4 (TCP) or L7 (HTTP) services



aws

Implement Kubernetes service by CLB

```
apiVersion: v1
kind: Service
metadata:
 name: nginx
 namespace: default
 annotations: {}
spec:
 ports:
 - name: http
   port: 80
   protocol: TCP
   targetPort: 80
 selector:
   app: nginx
type: LoadBalancer
```



aws

Implement Kubernetes service by NLB

```
apiVersion: v1
kind: Service
metadata:
 name: nginx
 namespace: default
 annotations:
   service.beta.kubernetes.io/aws-load-balancer-type: "nlb"
spec:
 ports:
 - name: http
   port: 80
   protocol: TCP
   targetPort: 80
 selector:
   app: nginx
type: LoadBalancer
```


aws

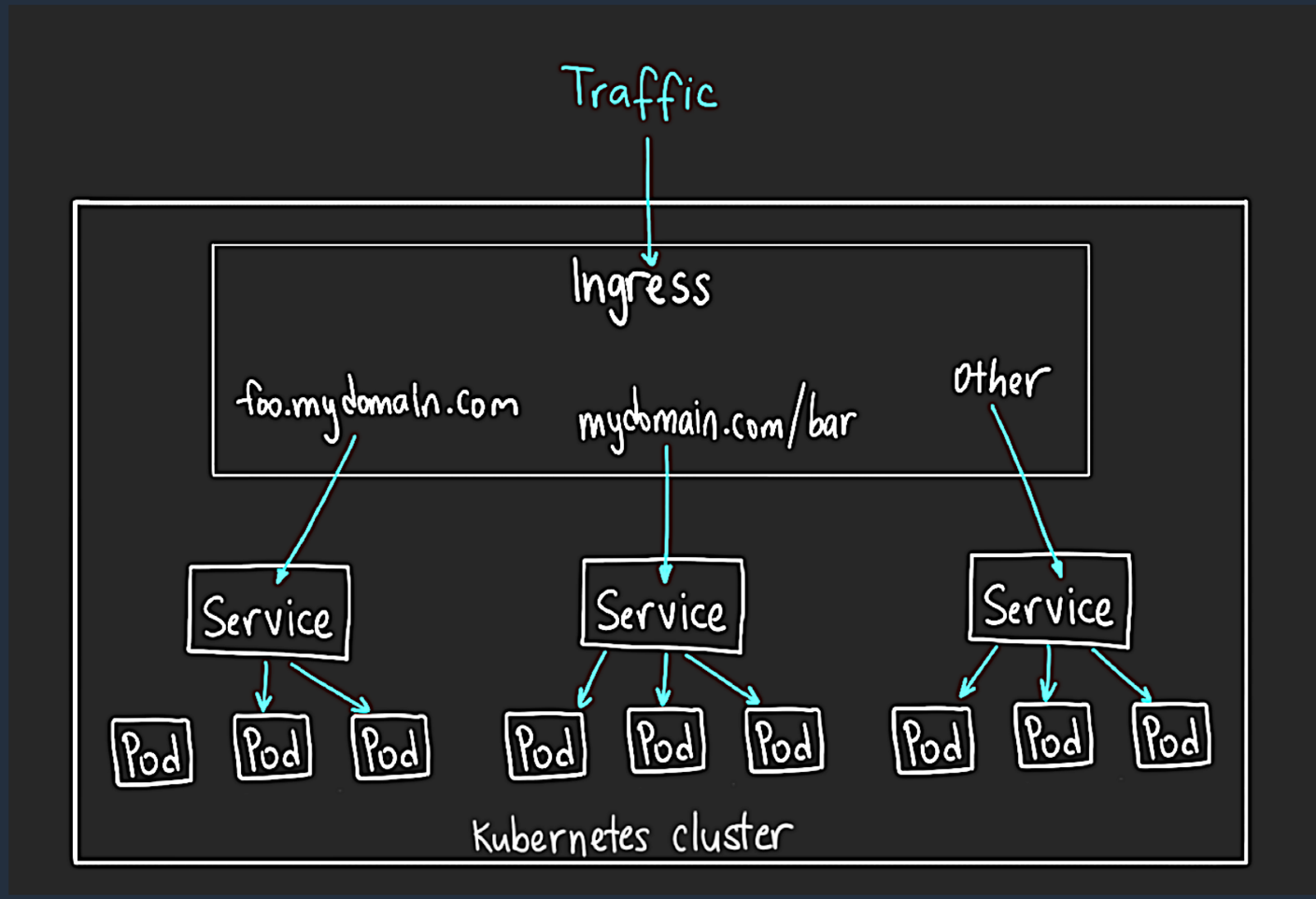
Using annotations to specify additional config (SSL)

```
…
kind: Service
metadata:
 annotations:
    service.beta.kubernetes.io/aws-load-balancer-ssl-cert: "arn:aws:acm:...."
    service.beta.kubernetes.io/aws-load-balancer-ssl-ports: "*"
    service.beta.kubernetes.io/aws-load-balancer-backend-protocol: "http”
spec:
    …
```


aws

Ingress Object

- Exposes HTTP/HTTPS routes to services within the cluster
- Implemented using an Ingress Controller.
- Many implementations: ALB, Nginx, F5, HAProxy etc



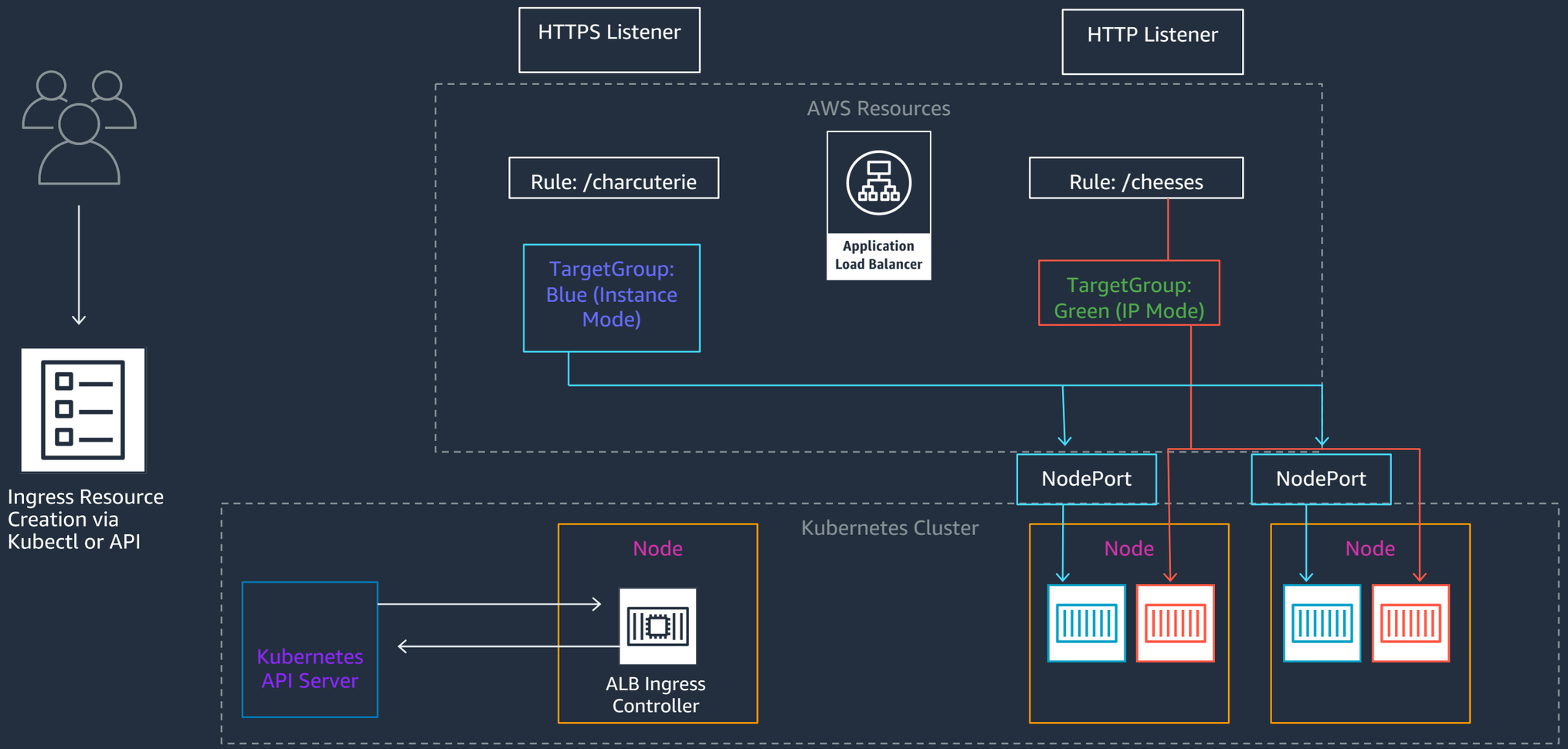
aws

Ingress in Kubernetes

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: echoserver
  namespace: echoserver
  annotations:
    kubernetes.io/ingress.class: alb
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/target-type: ip
    alb.ingress.kubernetes.io/subnets:  subnet-0a611a4986dd598ce ,subnet-024888df2852fa4ae
    alb.ingress.kubernetes.io/tags: Environment=dev,Team=test
spec:
  rules:
    - host: echoserver.example.com
      http:
        paths:
          - path: /
            backend:
              serviceName: echoserver
              servicePort: 80
```


aws

ALB Ingress Controller



https://aws.amazon.com/blogs/opensource/kubernetes-ingress-aws-alb-ingress-controller/

AWS Load Balancer Support

Classic and Network Load Balancer:

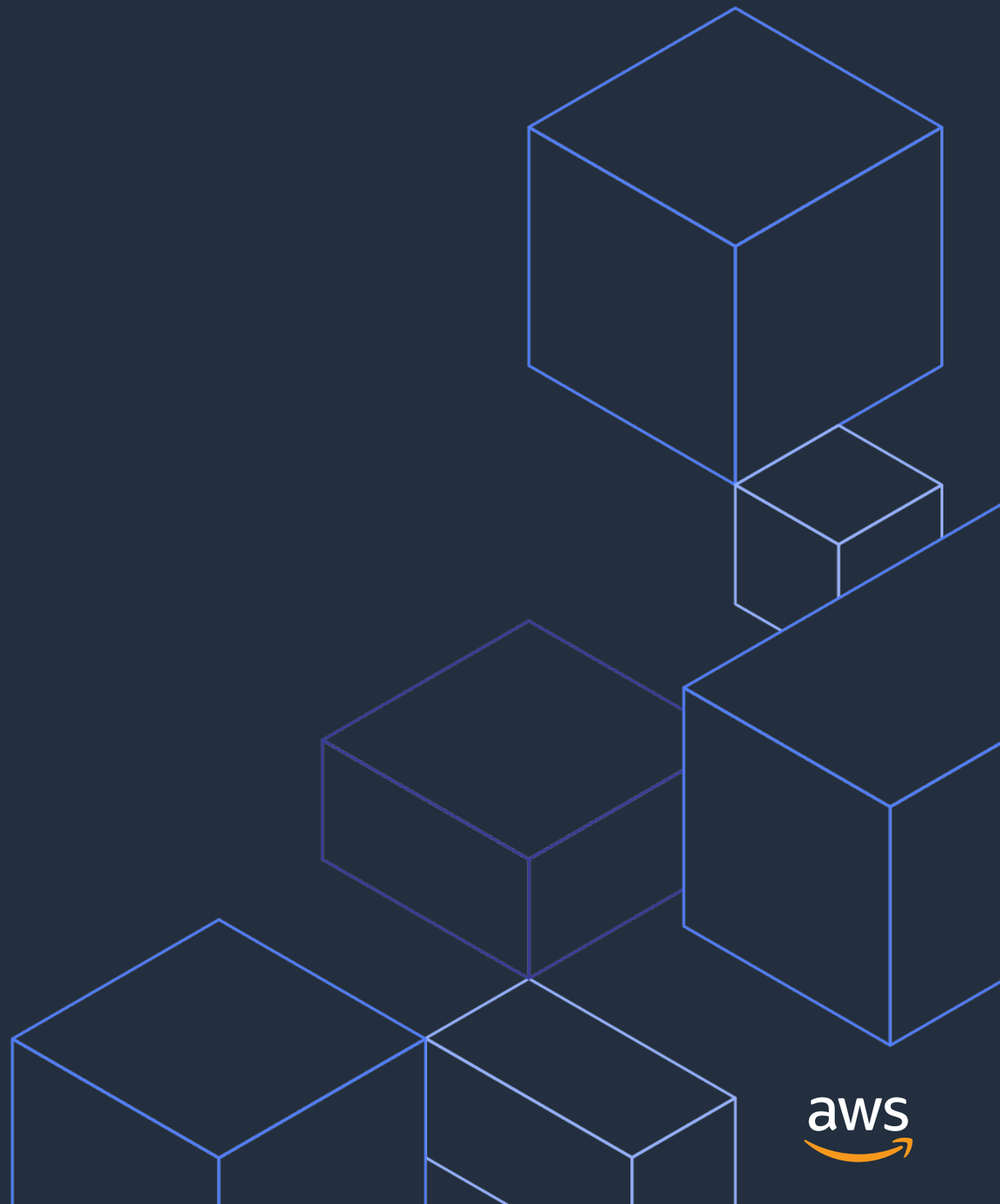
Fully Integrated out of the box.
Deployed using simple Kubernetes manifests

Application Load Balancer:

Integrated using the "ALB Ingress Controller" extension
https://github.com/kubernetes-sigs/aws-alb-ingress-controller

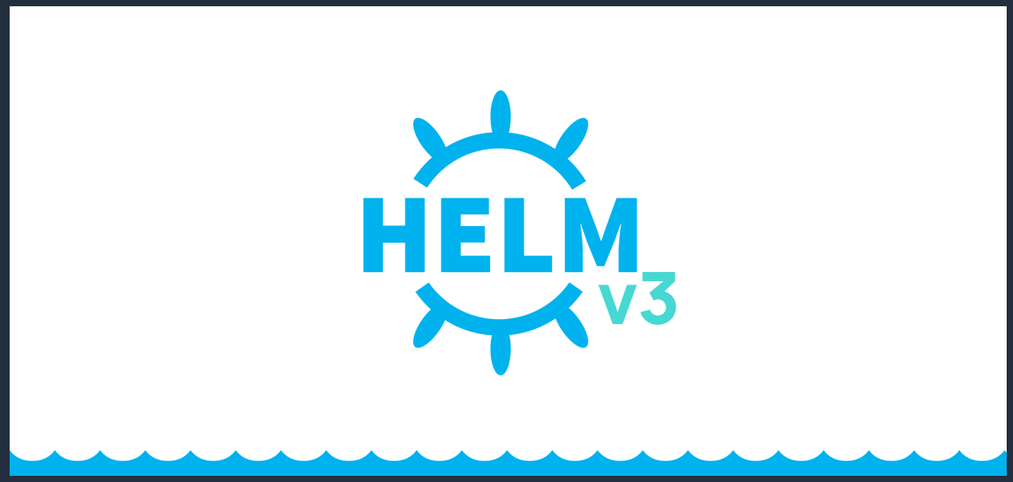
aws

Operations



What is Helm?

- Helm is *the* package manager for Kubernetes
- Inspired by tools like Homebrew, apt, npm
- Maintained by the CNCF
- Consists of two components
    - **Helm** – client component
    - **Tiller** – in-cluster server component



aws

Helm - Why should I care?

- Easier to manage all objects required for applications (Services, Pods, etc.)
- Easier to manage all the version / configurations of applications too
- Reduce the leaning curve associated with deploying production-grade
- Take the pain out of updates with in-place upgrades
- Reduce the complexity by providing sane defaults and exposing parameters in a consistent way

- For example, mysql helm package would create below

  All required Service accounts, secrets, service, configMaps, pvc, deployment, etc required for running mysql pods in the cluster

aws

Basic Helm Terminology

- **Charts** – a Helm package that contains all the resource definitions necessary to run an application, tool or service in a Kubernetes cluster

- **Repository** – the place where charts can be collected and shared

- **Release** – an instance of a chart running in a kubernetes cluster

- **Values** – provide a way to override template defaults with your own information

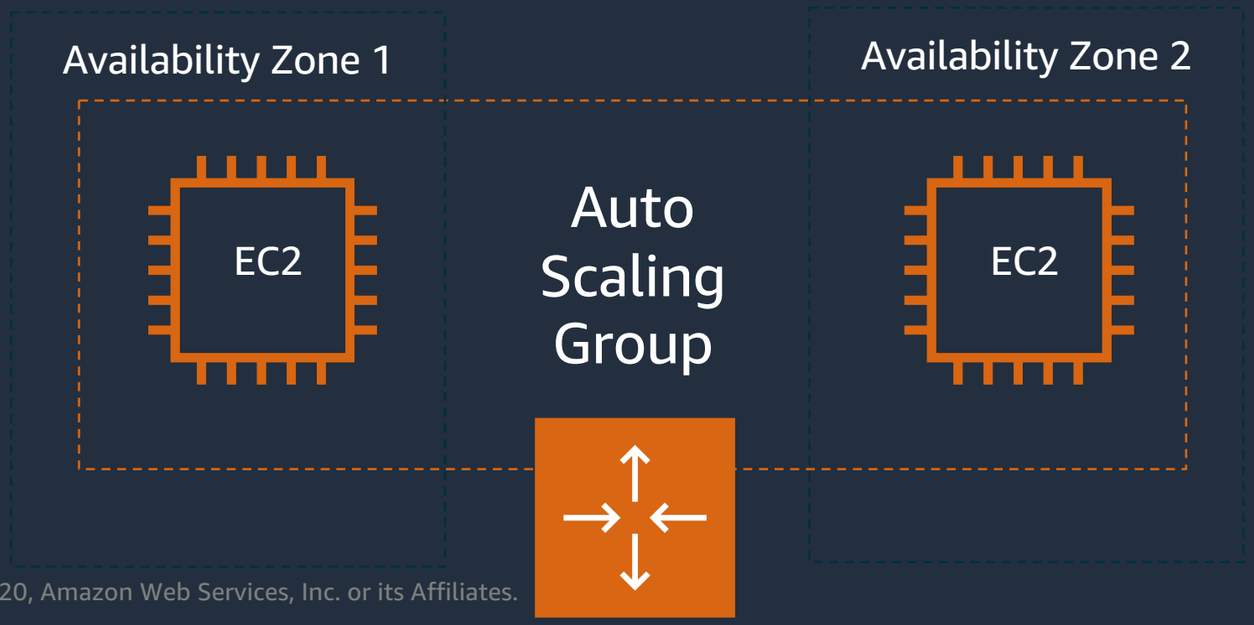
aws

Health Checks – Liveness and Readiness Probe

- Kubernetes restarts container if it crashes
- Liveness probes are used to know when a pod is alive/dead.
- Readiness probes are used to know when pod is ready to serve traffic.
- Pod is considered ready if all of its containers are ready.

Data Plane Scaling

- Integration called Cluster Autoscaler for AWS
- Deployed as daemonset in kube-system namespace
- Scale up/down unschedulable / under-utilized
- Create policy with access to `SetDesiredCapacity`




```
...
resources:
    limits:
        memory: 200Mi
    requests:
        memory: 100Mi
...
```

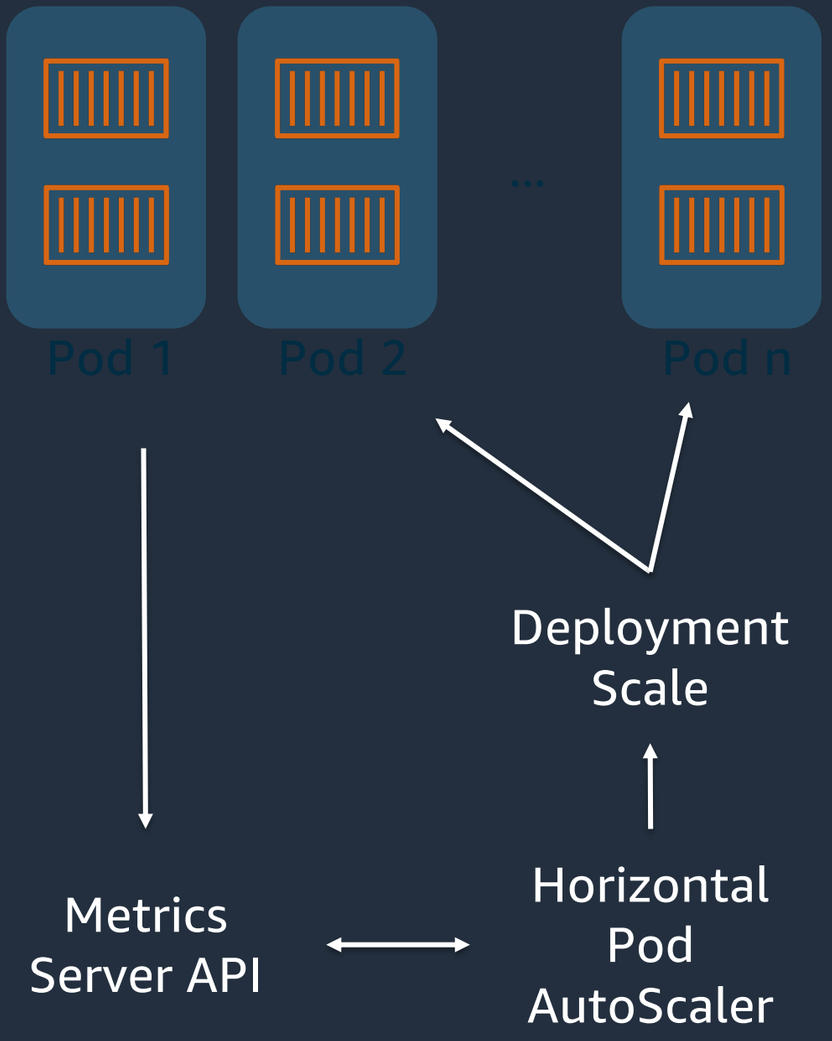

aws

Horizontal Pod AutoScaler (HPA)

- Scaling # pods based on CPU metrics
- Custom metrics supported such as
    - Inbound Number Connections

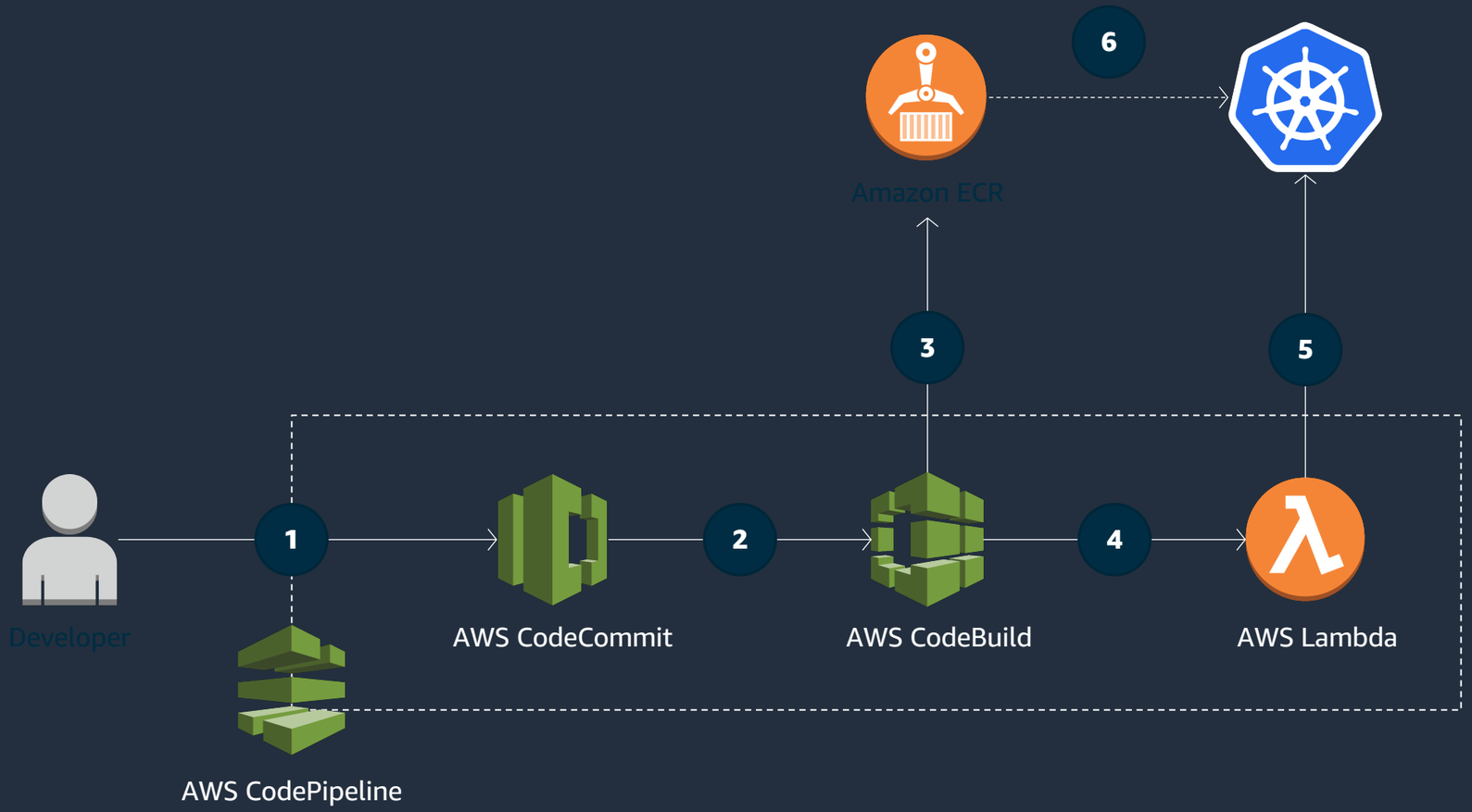
```
kubectl run php-apache --
image=k8s.gcr.io/hpa-example --
requests=cpu=200m --expose --
port=80
```

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



aws

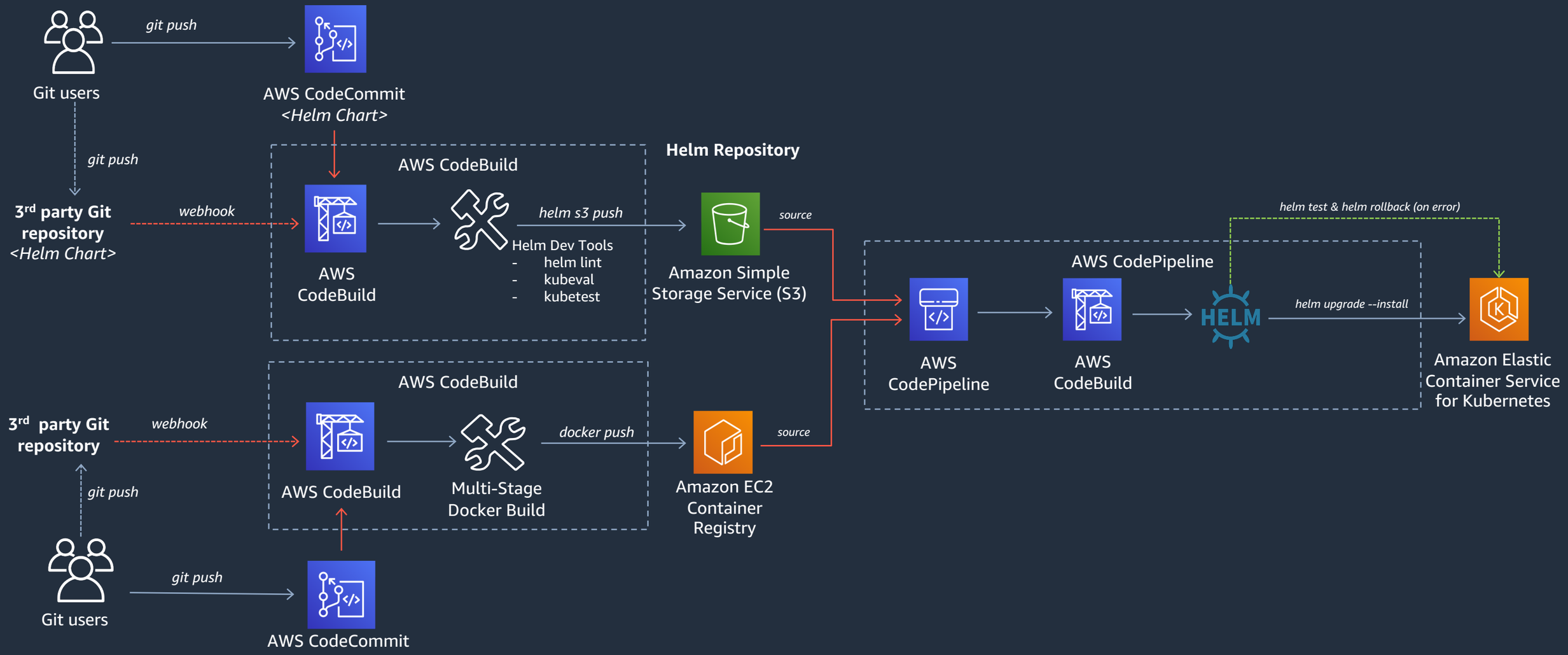
Kubernetes continuous deployment



1. Developers continuously integrate changes into a main branch hosted within a repo
2. Triggers an execution of the pipeline when a new version is found, builds a new image with build id
3. Pushes the newly built image tagged with build id to ECR repo
4. Invokes a Lambda function to trigger application deployment
5. Leverages Kubernetes Python SDK to update a deployment
6. Fetches new container image and performs a rolling update of deployment

aws

EKS Continuous Deployment pipeline



GitOps

aws

GitOps

GitOps is a paradigm or a set of practices that empowers developers to perform tasks which typically fall under the purview of IT operations. **GitOps** requires us to describe and observe systems with declarative specifications that eventually form the basis of continuous everything.

aws

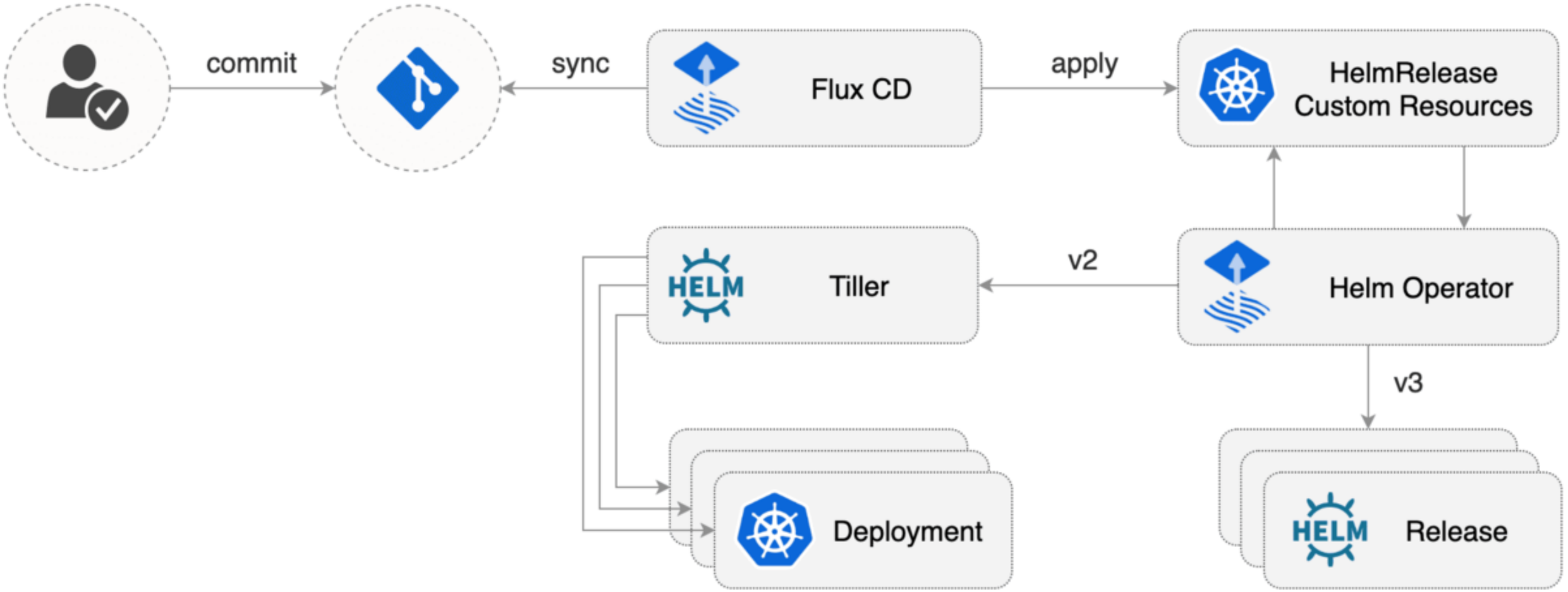
GitOps vs Traditional CI/CD

In a traditional CI/CD pipeline, CD is an implementation extension powered by the continuous integration tooling to promote build artifacts to production.

In the GitOps pipeline model, any change to production must be committed in source control (preferable via a pull request) prior to being applied on the cluster. If the entire production state is under version control and described in a single Git repository, when disaster strikes, the whole infrastructure can be quickly restored without rerunning the CI pipelines.

aws

GitOps



aws

GitOps

- The entire system is declared declaratively
- The canonical desired system state is versioned (with Git)
- Approved changes to the desired state automatically applied to the system.
- Software agents ensures correctness and alert on divergence

Not GitOps

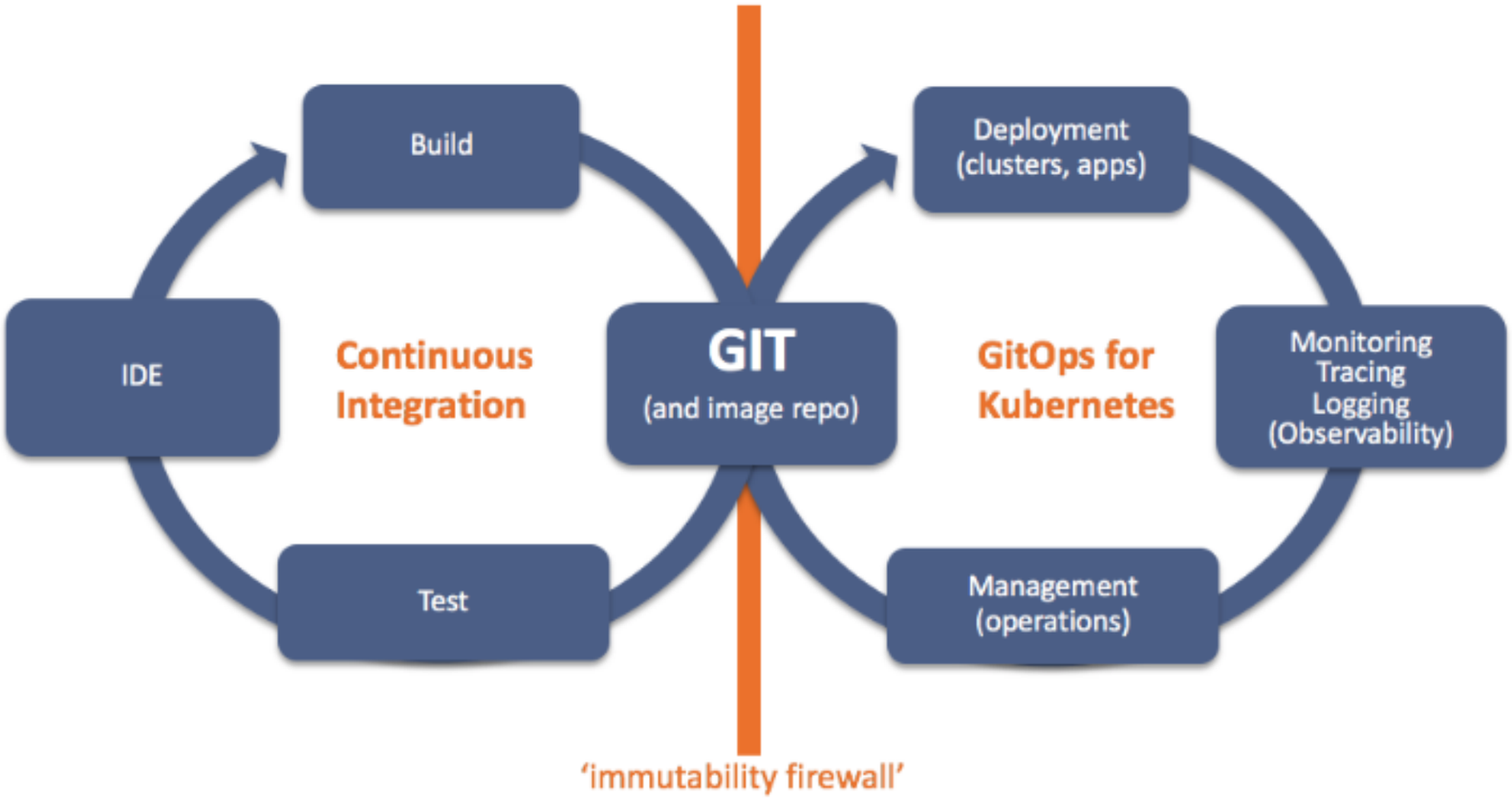
- Application Deployment Should be Different from CI
- Developer Release, Operators Deploy
- This Does Not Separate Concerns

aws

GitOps Benefits

- Stored history of environment changes
- Easy rollback to a previous state
- Secure deployments
- Lightweight approval process
- Modular architecture
- Tool-independent architecture
- Reuse existing knowledge
- Compare different environments
- Backups come out of the box
- Testing your changes like app code
- Highly available deployment infrastructure

aws



Git as the single source of truth of a system's desired state

GitOps Diffs compare desired state with observed state (eg Kubediff, Terradiff, Canary..)

ALL intended operations are committed by pull request, for all environments

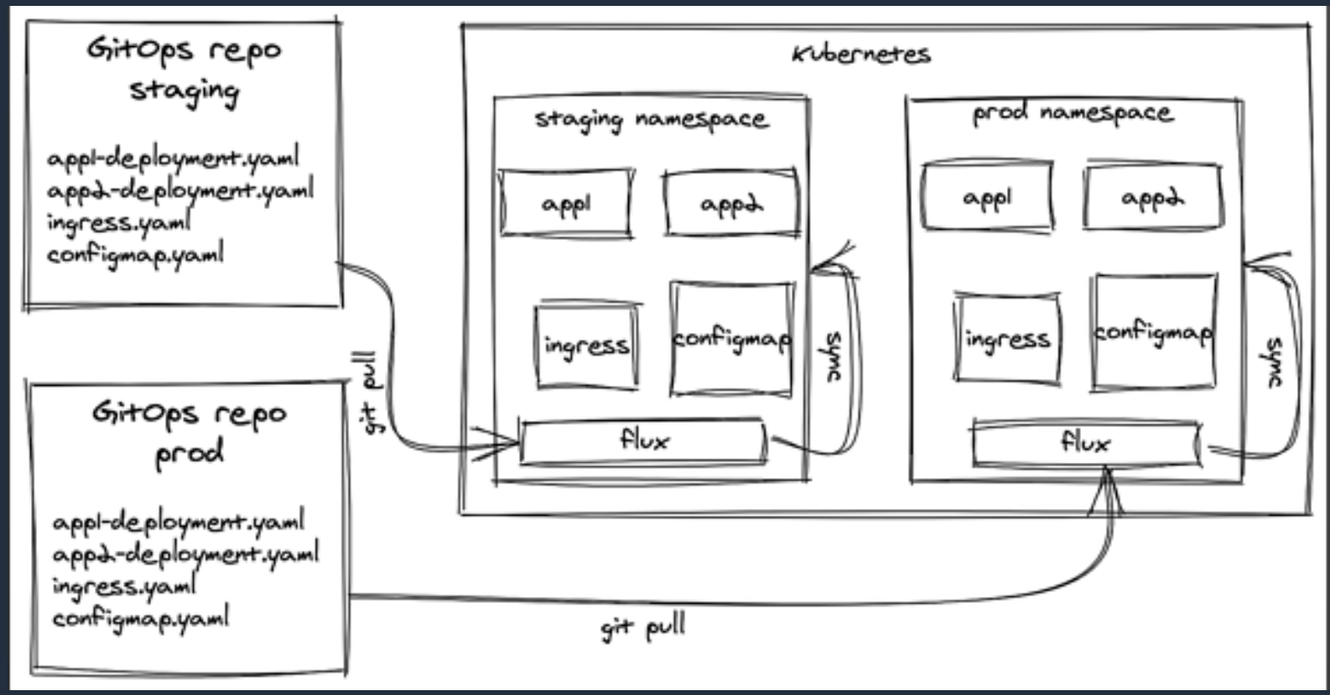
ALL diffs between GIT and observed state lead to (auto) convergence using tools like K8s

ALL changes are observable, verifiable and audited indisputably, with rollback & D/R

aws

FluxCD

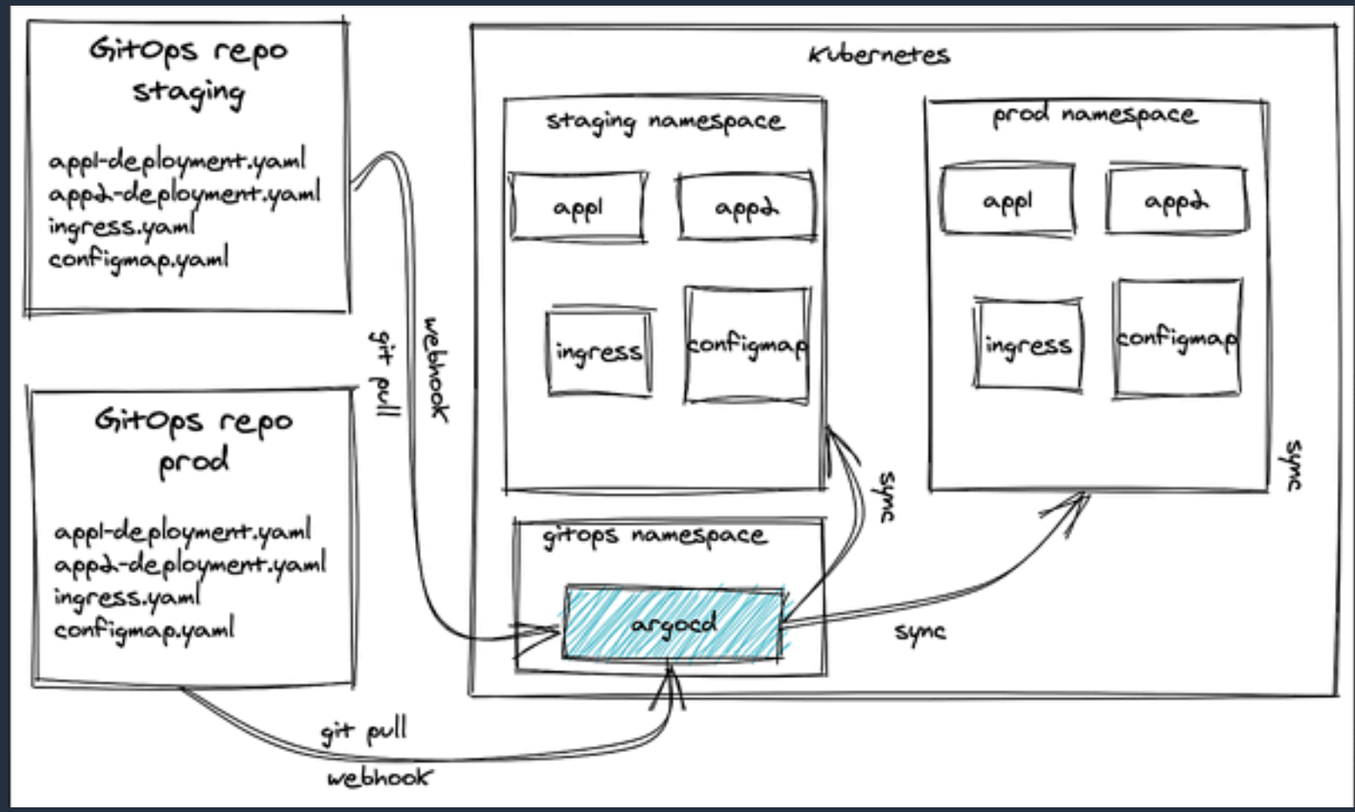
- **Single Repository**
- It periodically pulls a remote Git repository and applies its manifest files to the cluster if there are new changes, in a true GitOps fashion.
- Considering Flux is practically a single binary deployed in a Pod
- Flux doesn't have a multi-tenancy mode.
- For the same reason—simplicity—Flux could be used in a multi-cluster scenario.



aws

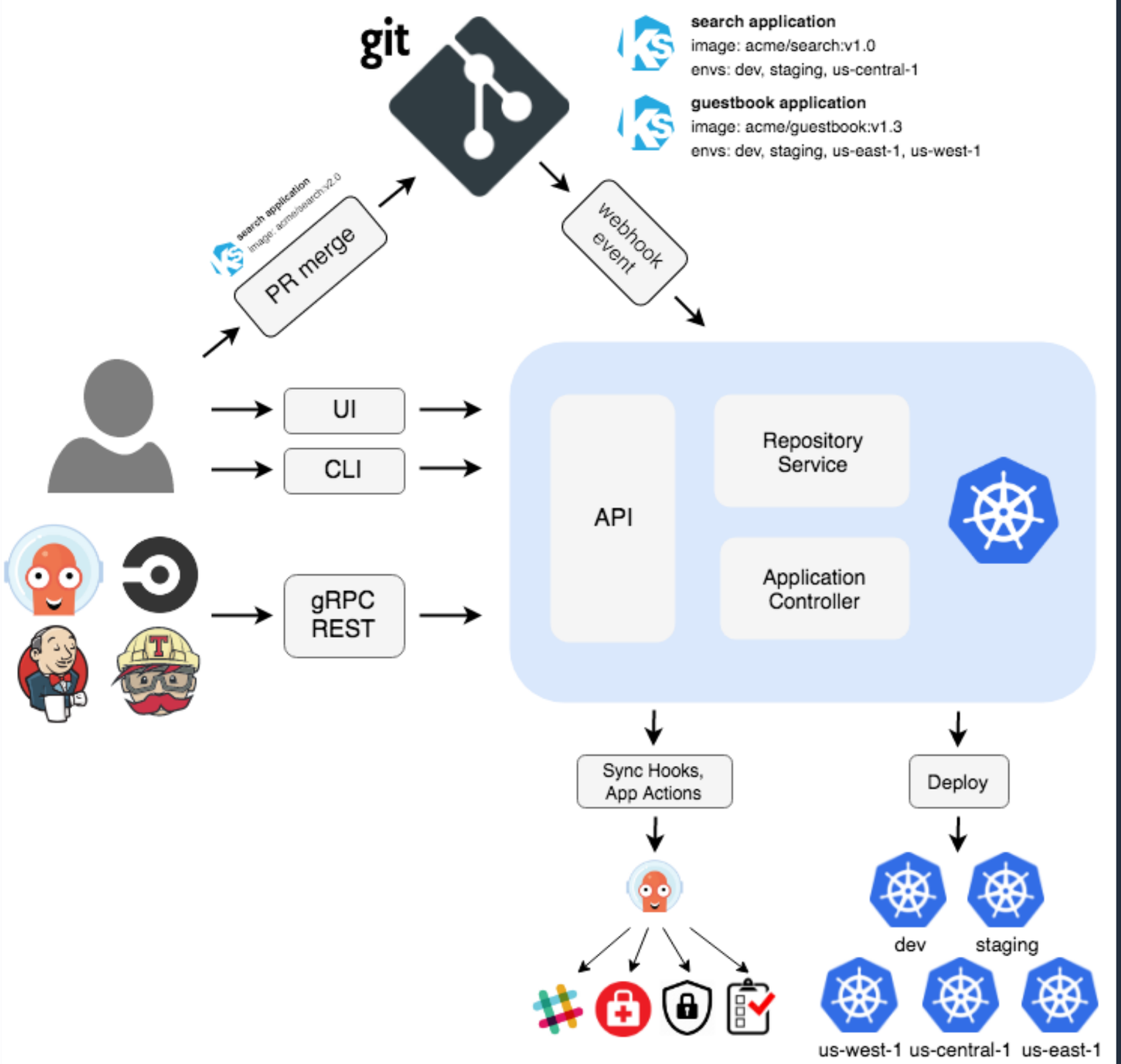
ArgoCD

- ArgoCD watches a Git repositories tracked by similar to how Flux works. However, what makes it shine is the capability to manage **multi-tenant and multi-cluster deployments** with fine controls.
- ArgoCD really shines when it comes to important features like multi-tenancy, but also has a myriad of customization options.
- A single instance of ArgoCD can handle multiple projects
- ArgoCD can sync applications on the Kubernetes cluster it is running on and can also manage external clusters. It can be configured to only have access to a restricted set of namespaces.



aws

ArgoCD



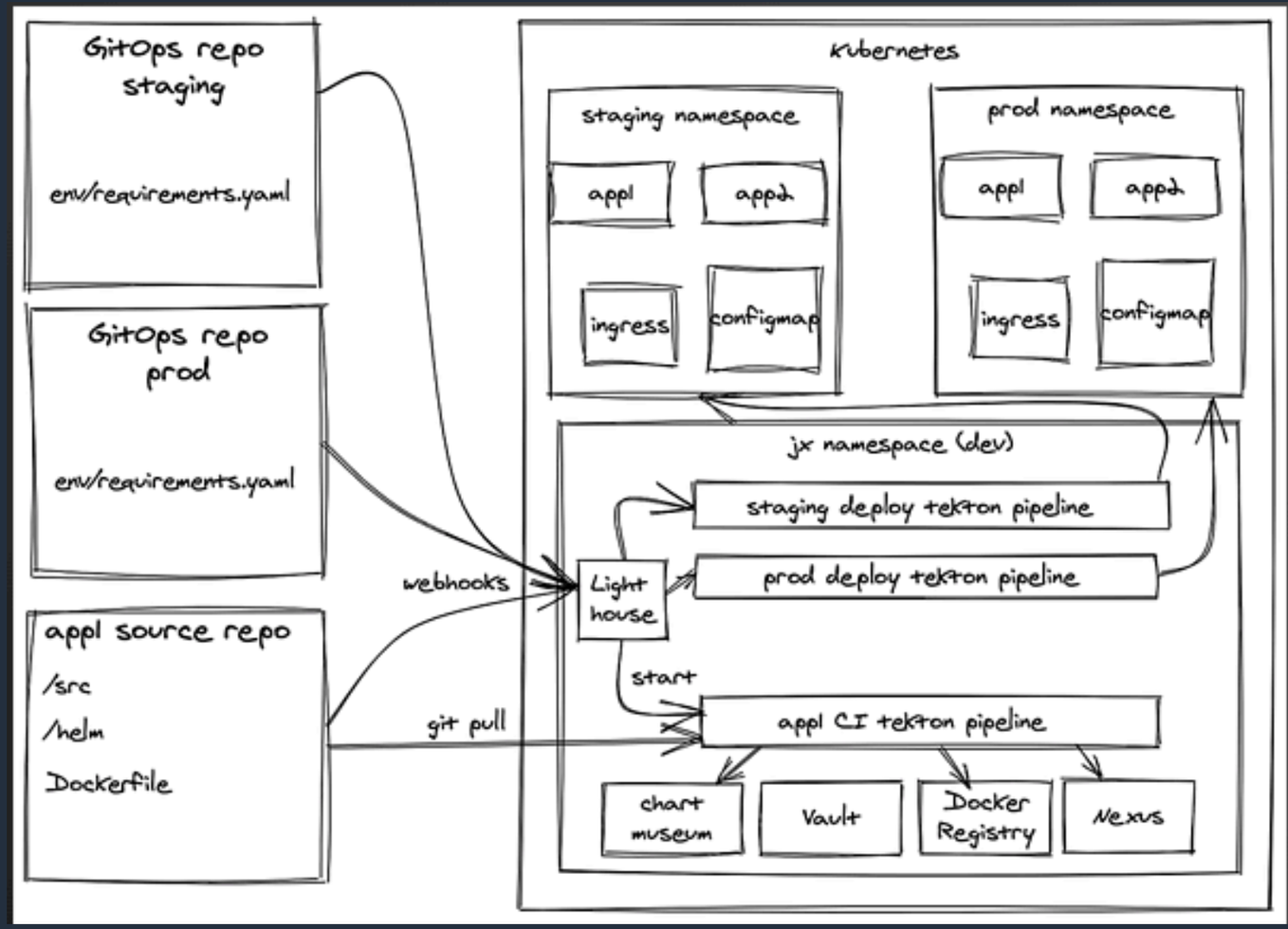
aws

Jenkins-X

- CI/CD solution built around GitOps and using Tekton under the hood. At first, judging by the name, we would expect to see the next version of the Jenkins we all know, with its jobs and plugins.

- Jenkins X has taken a different direction and has very little in common with the classic Jenkins.

- Given a successful application build after a merge to master in the application repository, a new release version of the application Helm chart is created.



AWS, Weaveworks, and Intuit Collaboration

Containers

Help us write a new chapter for Gitops, Kubernetes, and Open Source collaboration

by Jay Pipes | on 14 NOV 2019 | in Amazon Elastic Kubernetes Service, Containers | Permalink | Comments | Share

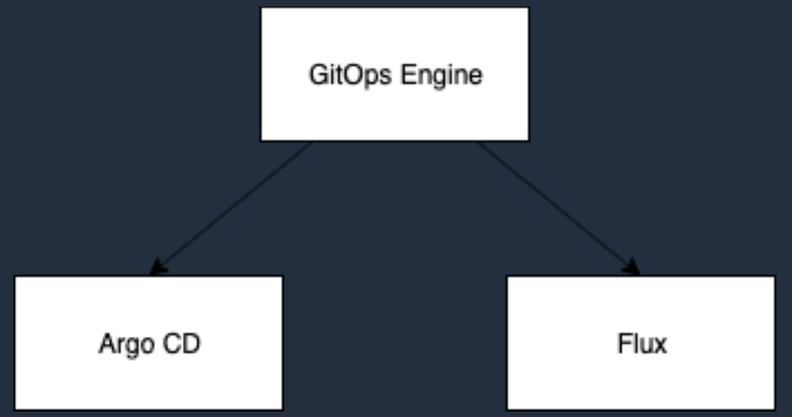
Introduction

The Amazon Elastic Kubernetes Service (EKS) team sees the ecosystem around automated software deployment as technology frontier ripe with potential for groundbreaking innovation.

Over the last twenty years, the way in which developers deploy and manage their applications has changed dramat Technology improvements in packaging, automation, and virtualization as well as shifts in operations culture have profoundly shaped the software deployment landscape.

Most recently, a set of best practices for automated delivery of code has emerged in parallel with the growth in popularity of Kubernetes. These practices, called *GitOps*, underpin the design of two projects in the open source ecosystem, Flux CD and Argo CD, which are joining forces as Argo Flux.

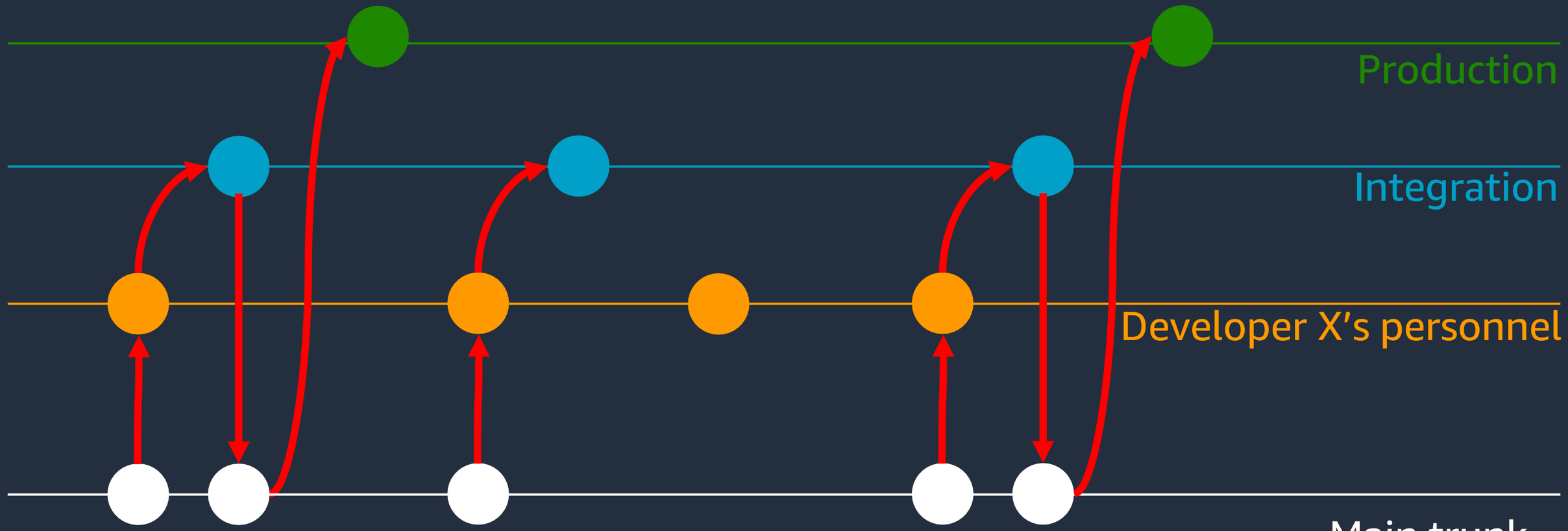
The EKS team is excited to participate in the Argo Flux collaboration and we're asking the community to join us at KubeCon to learn, provide feedback and help us write a new chapter for GitOps and Kubernetes.



aws

What is GitOps?

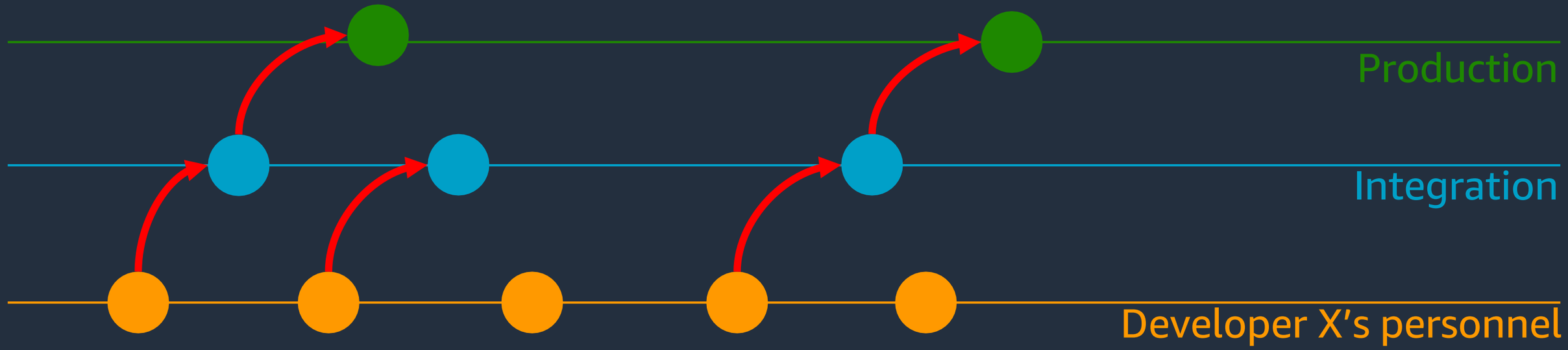
Control via PRs & environment branches (merge to them and it'll deploy there on successful tests)



aws

What is GitOps?

Control via PRs & environment branches (merge to them and it'll deploy there on successful tests)



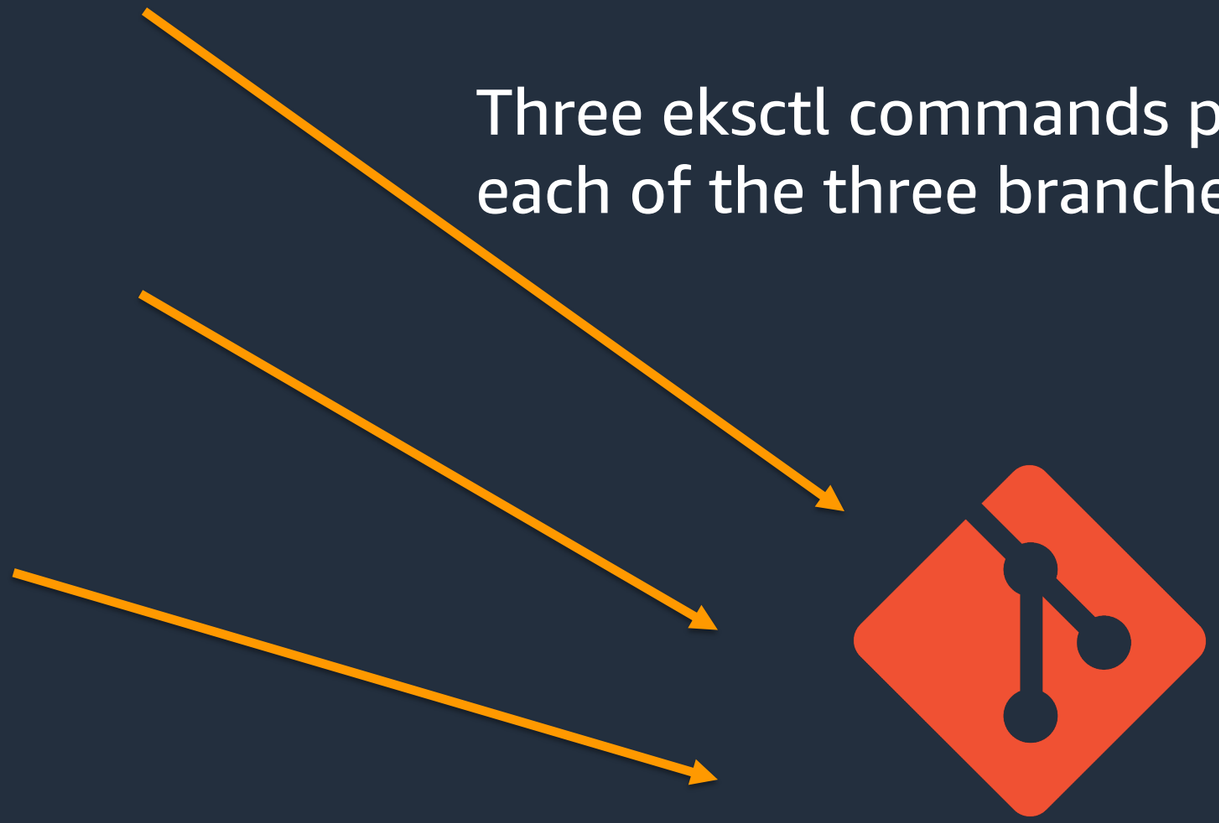
aws

Common Strategies

github.com/bryan/prod

github.com/michele/staging

github.com/cooper/test



Each Development/Deployment stage uses a different branch and the *same* git repository.

Three eksctl commands point to each of the three branches

aws

Common Strategies

github.com/bryan/prod

github.com/michele/staging

github.com/cooper/test

aws

Progressive Delivery

Progressive delivery is an umbrella term for advanced deployment patterns like canaries, feature flags and A/B testing. Progressive delivery techniques are used to reduce the risk of introducing a new software version in production by giving app developers and SRE teams a fine-grained control over the blast radius.

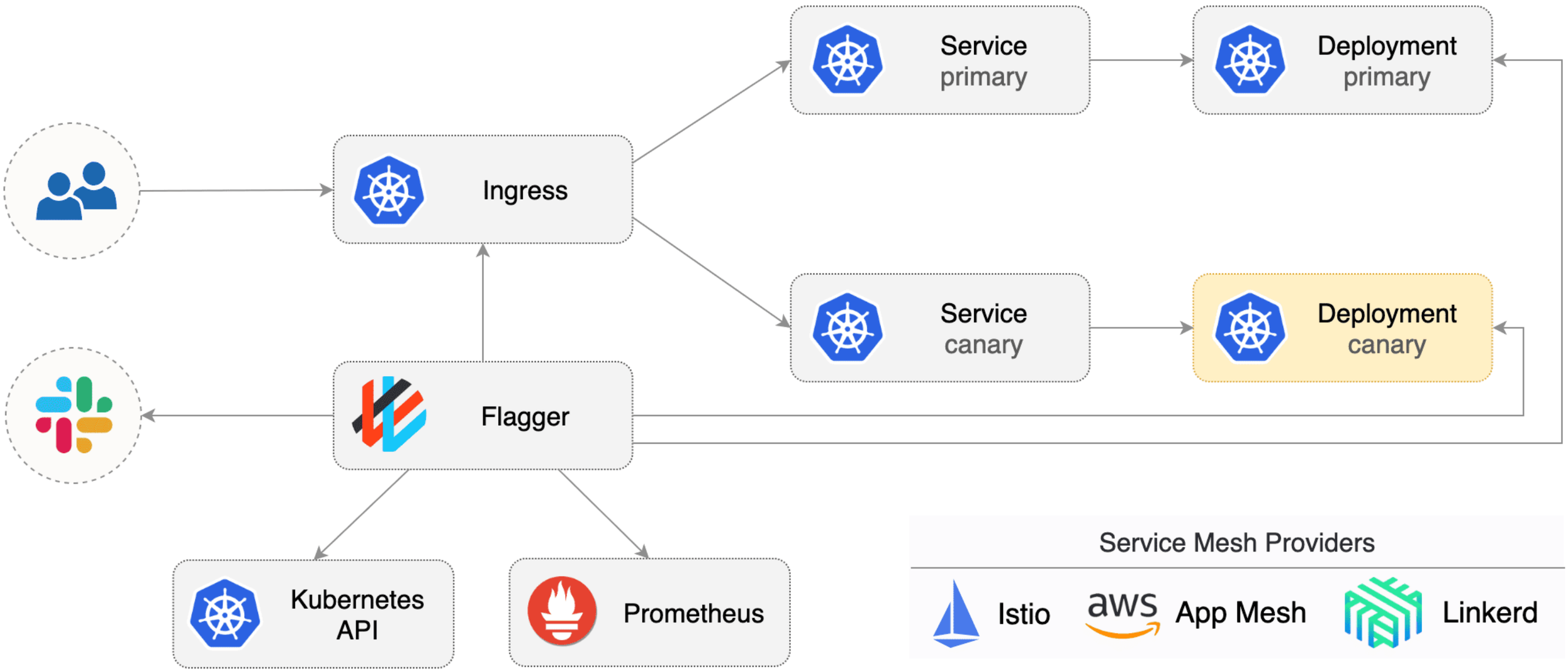
aws

Flagger

Flagger is a progressive delivery tool that automates the release process for applications running on Kubernetes. It reduces the risk of introducing a new software version in production by gradually shifting traffic to the new version while measuring metrics and running conformance tests.

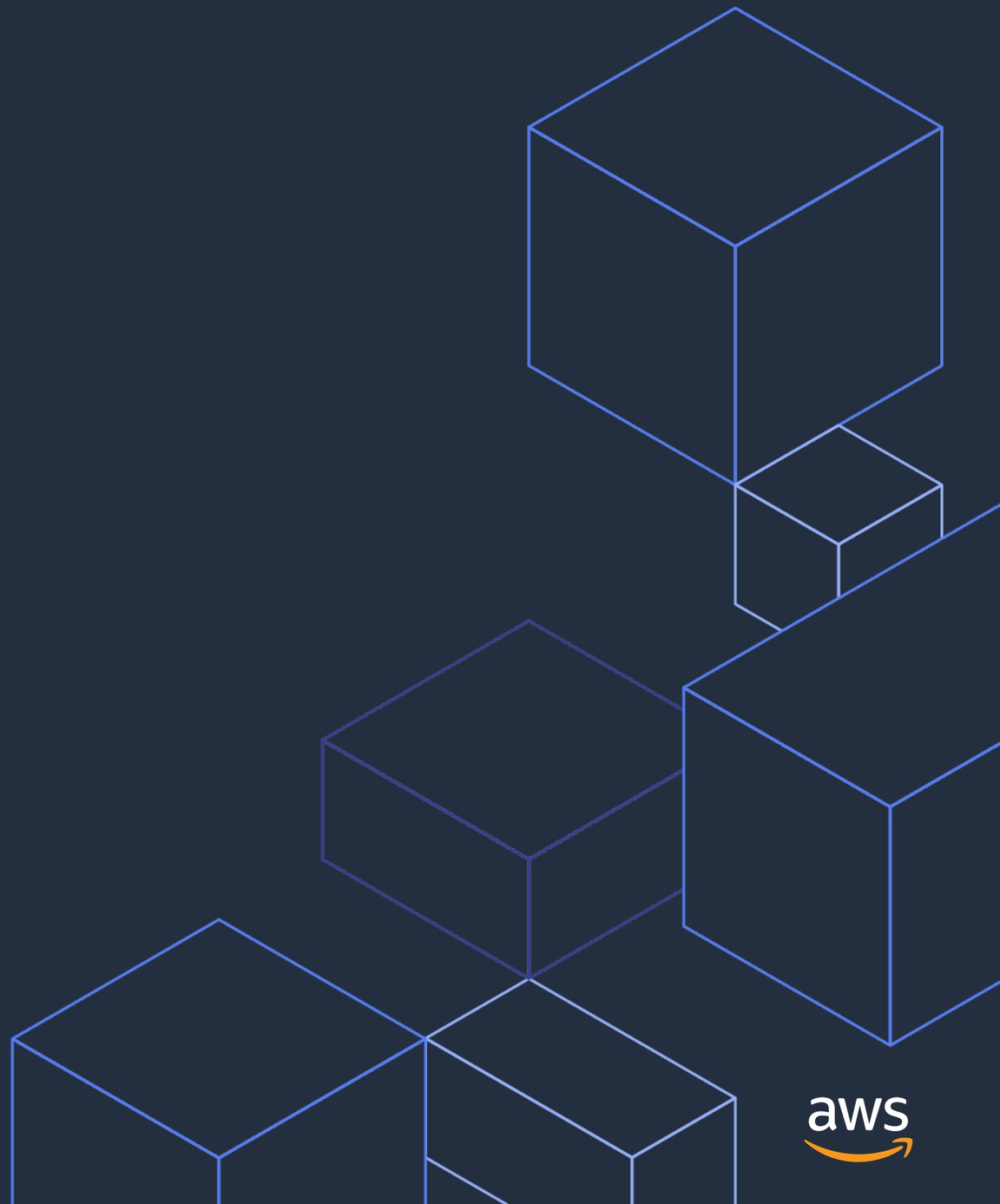
aws

Flagger

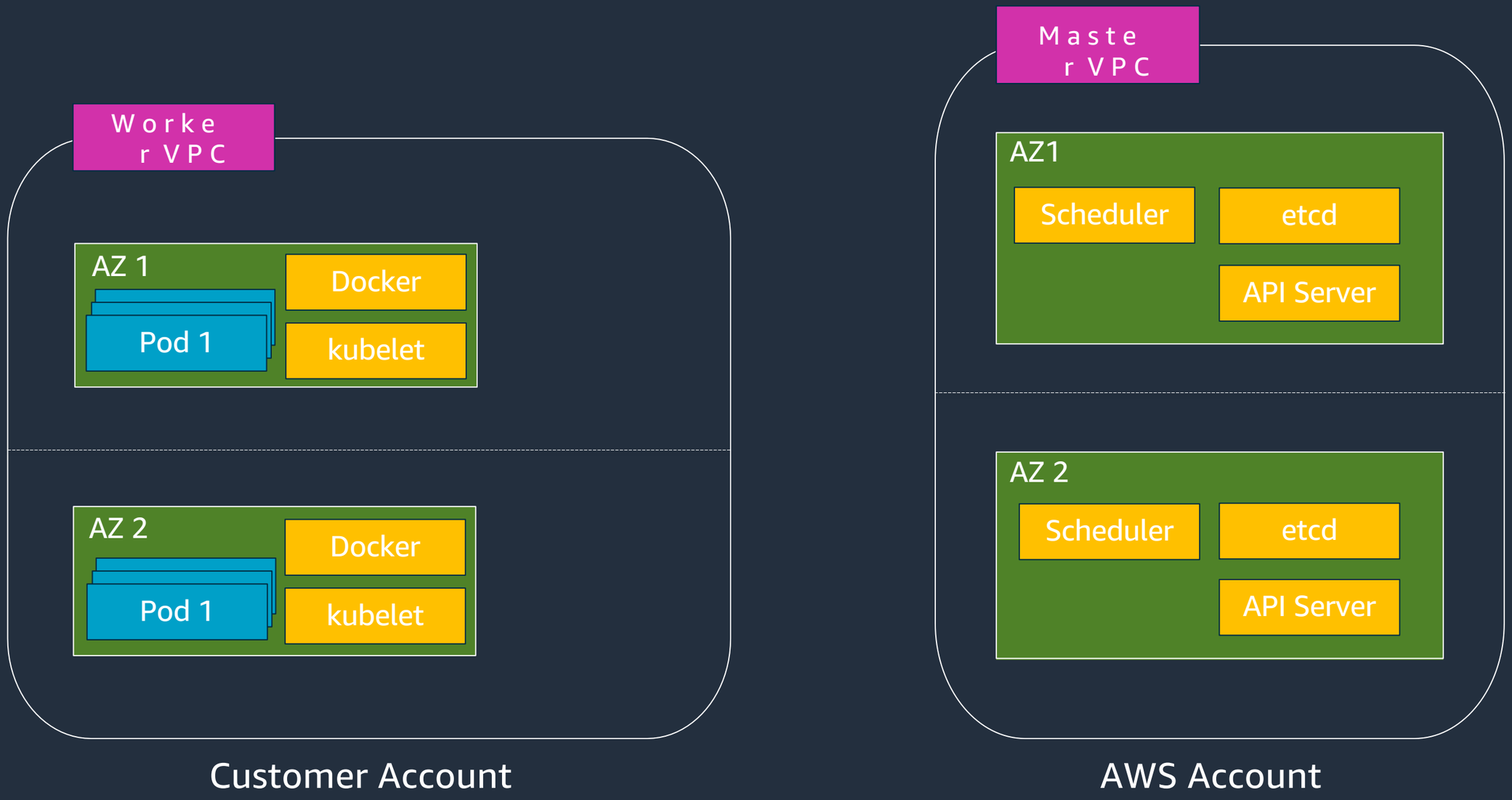


aws

Networking

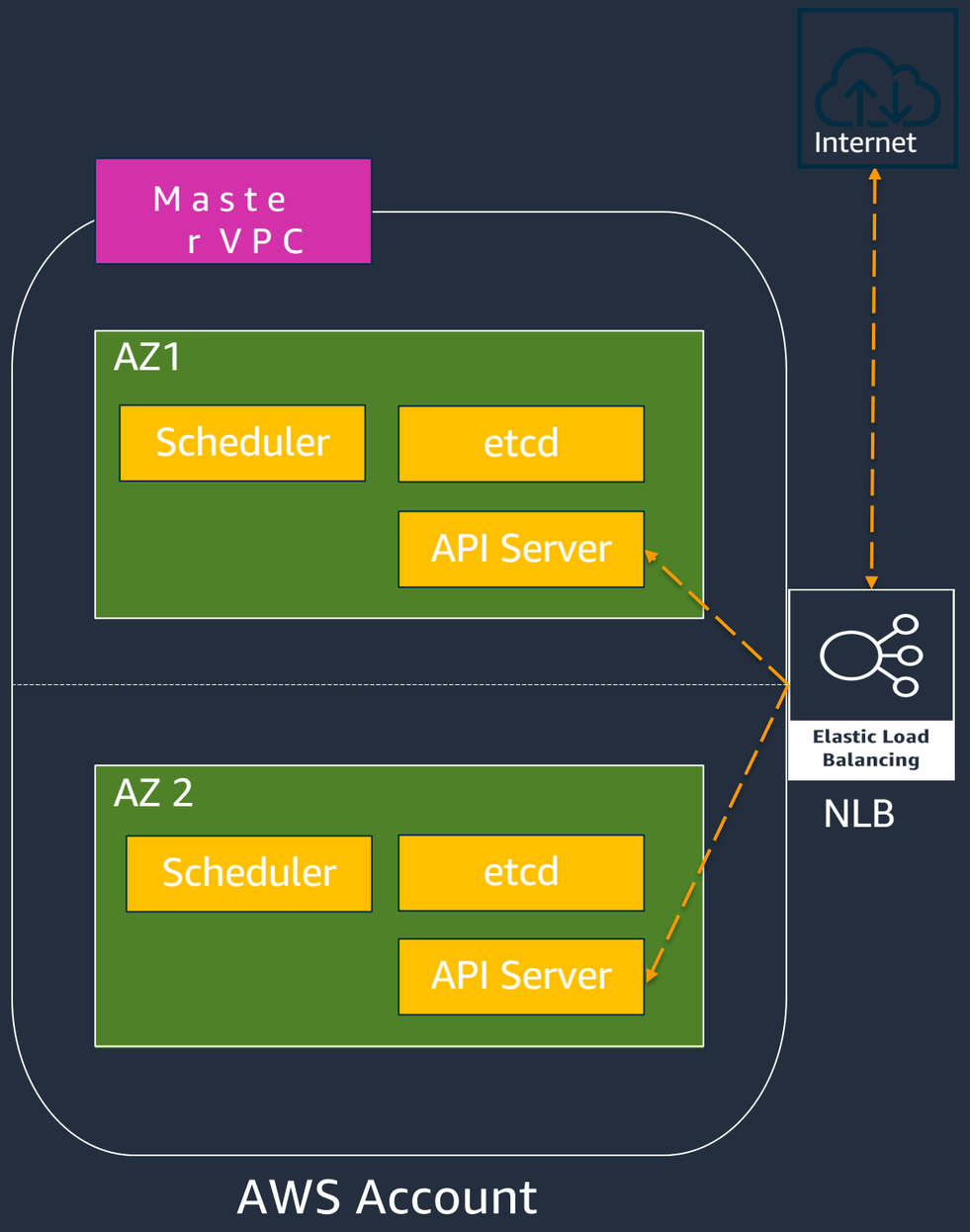


EKS Architecture

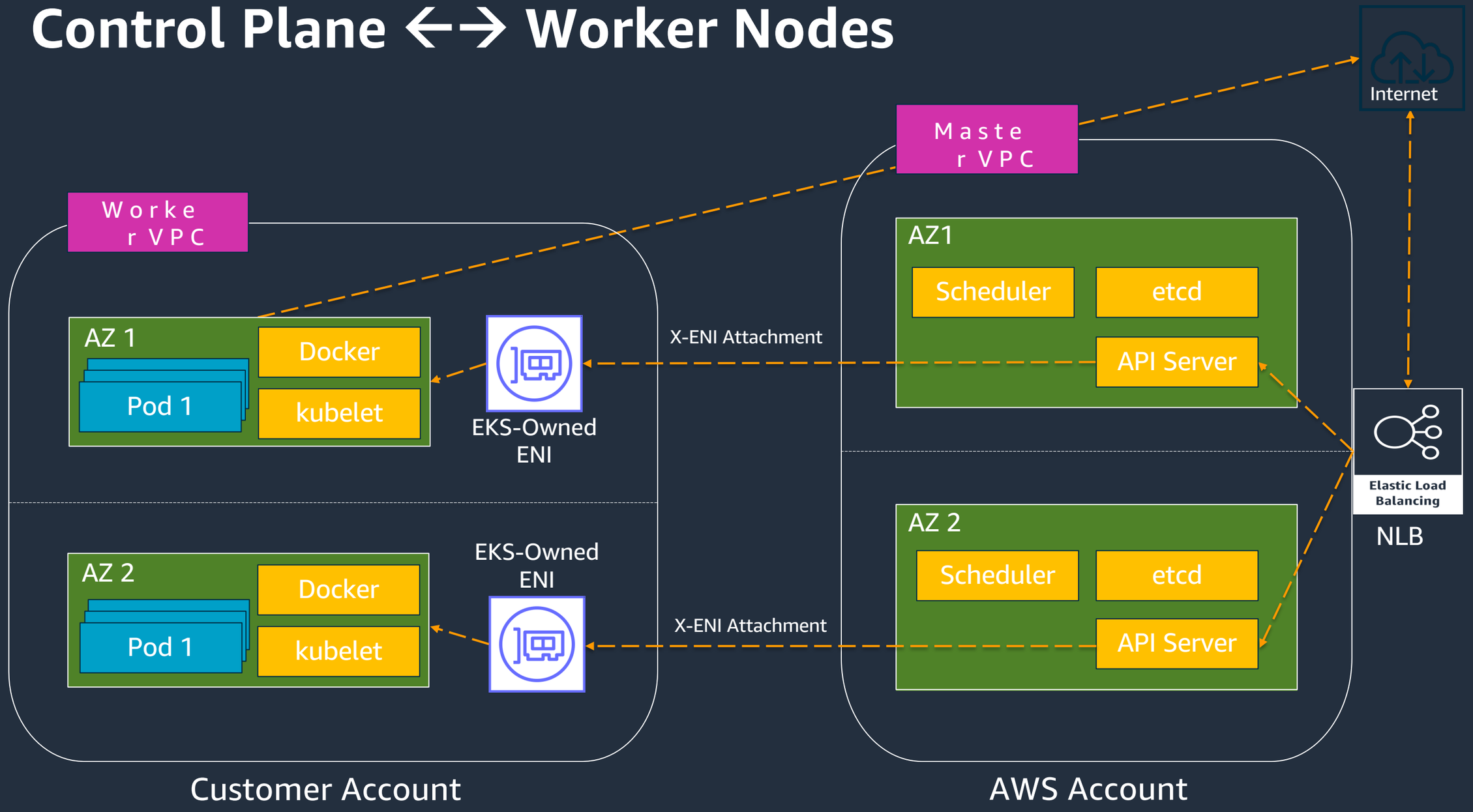


aws

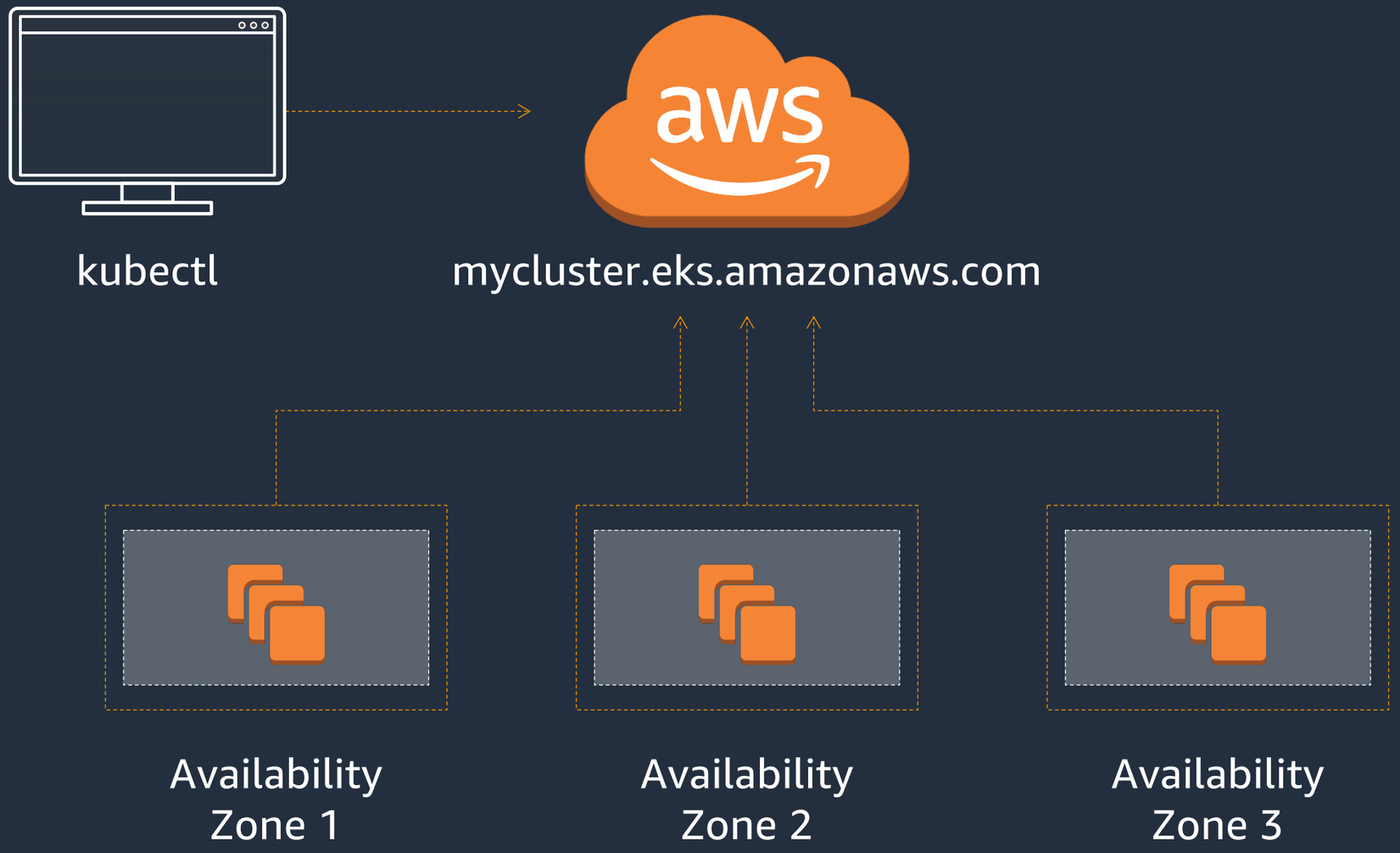
Control Plane Networking



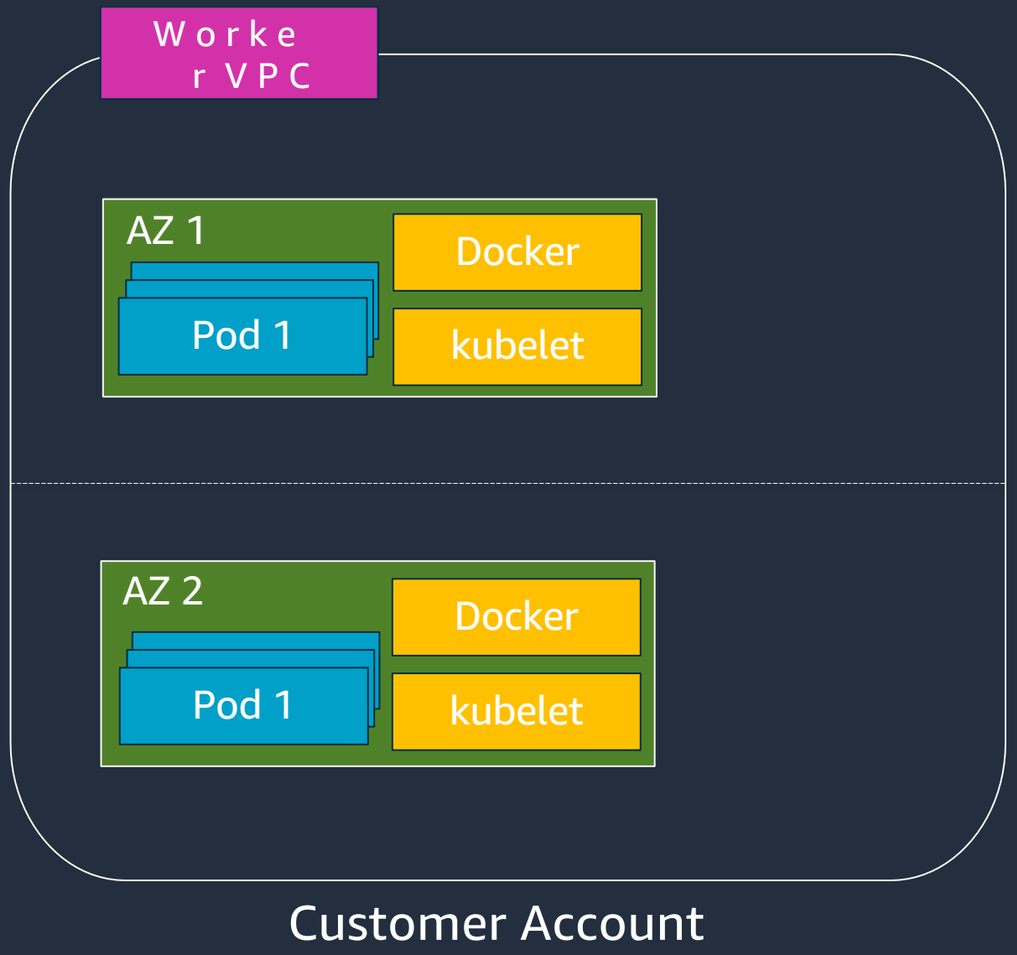
Control Plane ←→ Worker Nodes



Kubernetes Endpoint Private Access



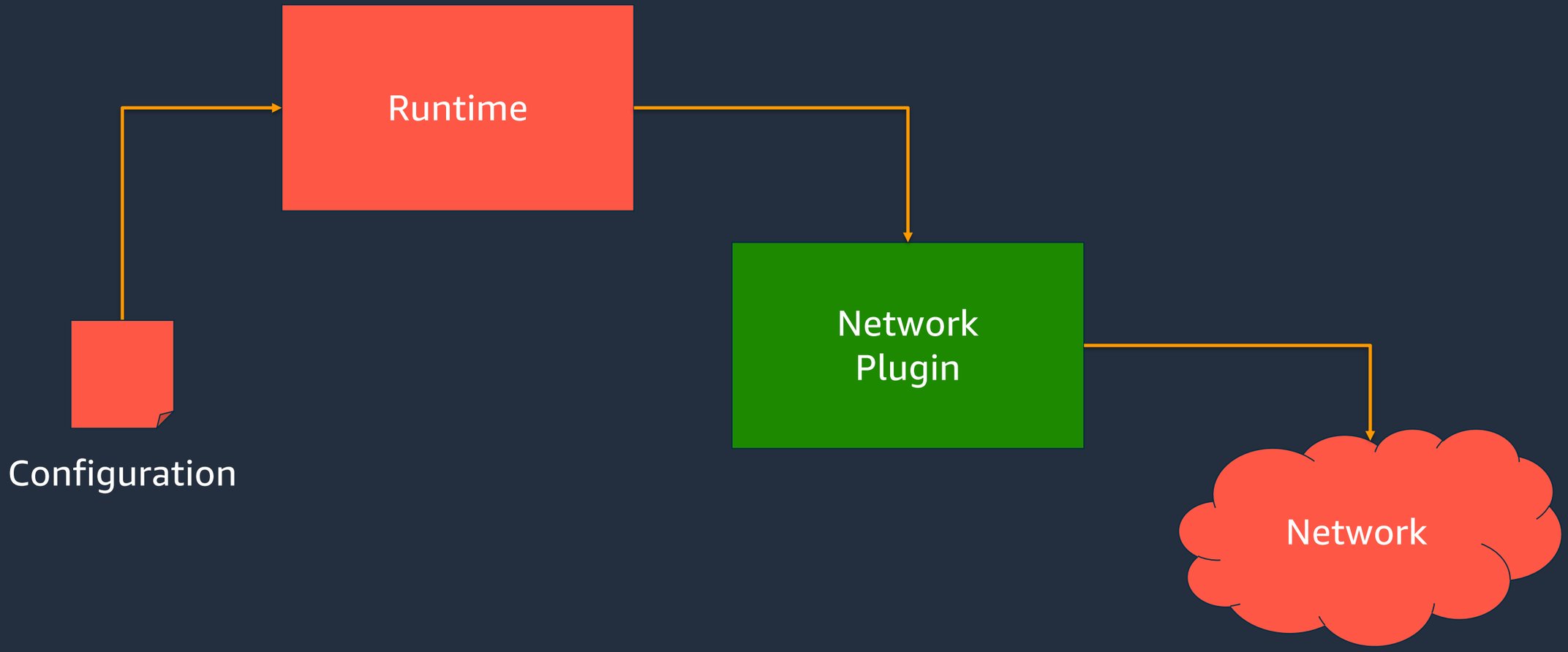
Kubernetes Network Requirements



- All containers can communicate with all other containers without NAT
- All nodes can communicate with all containers (and vice-versa) without NAT
- The IP address that a container sees itself as is the same IP address that others see it as

aws

Container Network Interface (CNI)



aws

Amazon VPC CNI Plugin Goals

1. Simplify networking options for customers

2. Support high throughput, high availability, low latency and minimal jitter

3. Allow customers to reuse AWS VPC networking and security best practices such as use of:
    - VPC flow logs for troubleshooting and compliance auditing
    - VPC routing polices for traffic engineering
    - Security groups for isolation and regulatory requirements

4. Setup Pod networking within seconds

5. Support cluster scale to a minimum of 5000+

aws

Amazon VPC CNI Plugin

Native VPC networking with CNI plugin

Pods have the same VPC address inside the pod as on the VPC

Simple, secure networking

Open source and on Github

https://github.com/aws/amazon-vpc-cni-k8s

EC2 and VPC Considerations

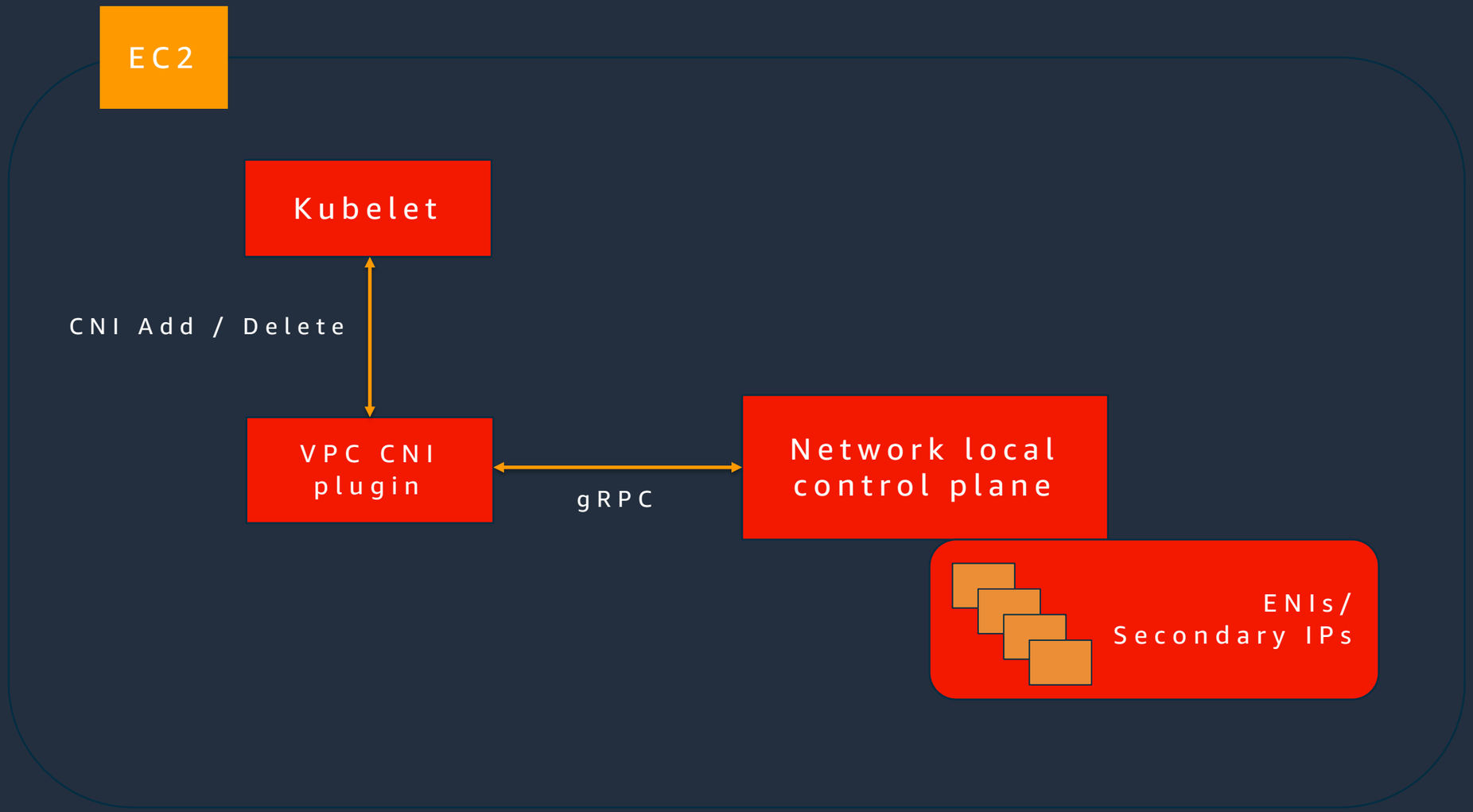
- Use separate VPC for each EKS cluster
- EKS requires subnets in at least two AZ’s
- VPC and Subnet CIDR
- Pod density calculation
  - Max Pods = min((N * M - N), subnet's free IP)
  - N = # of ENIs, M = # of IP per ENI

aws

Amazon VPC CNI Deep Dive

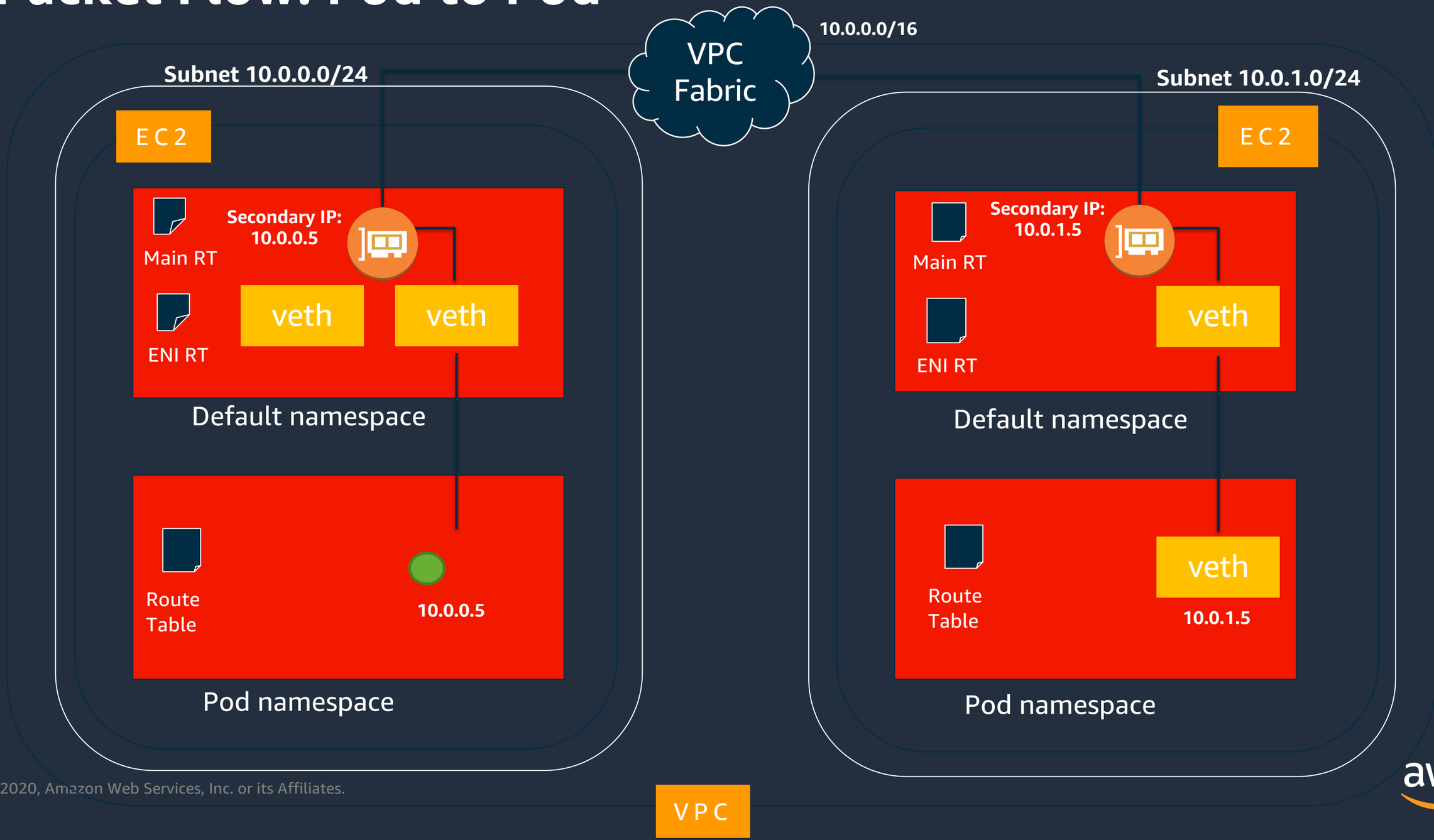
aws

Amazon VPC CNI Plugin Architecture

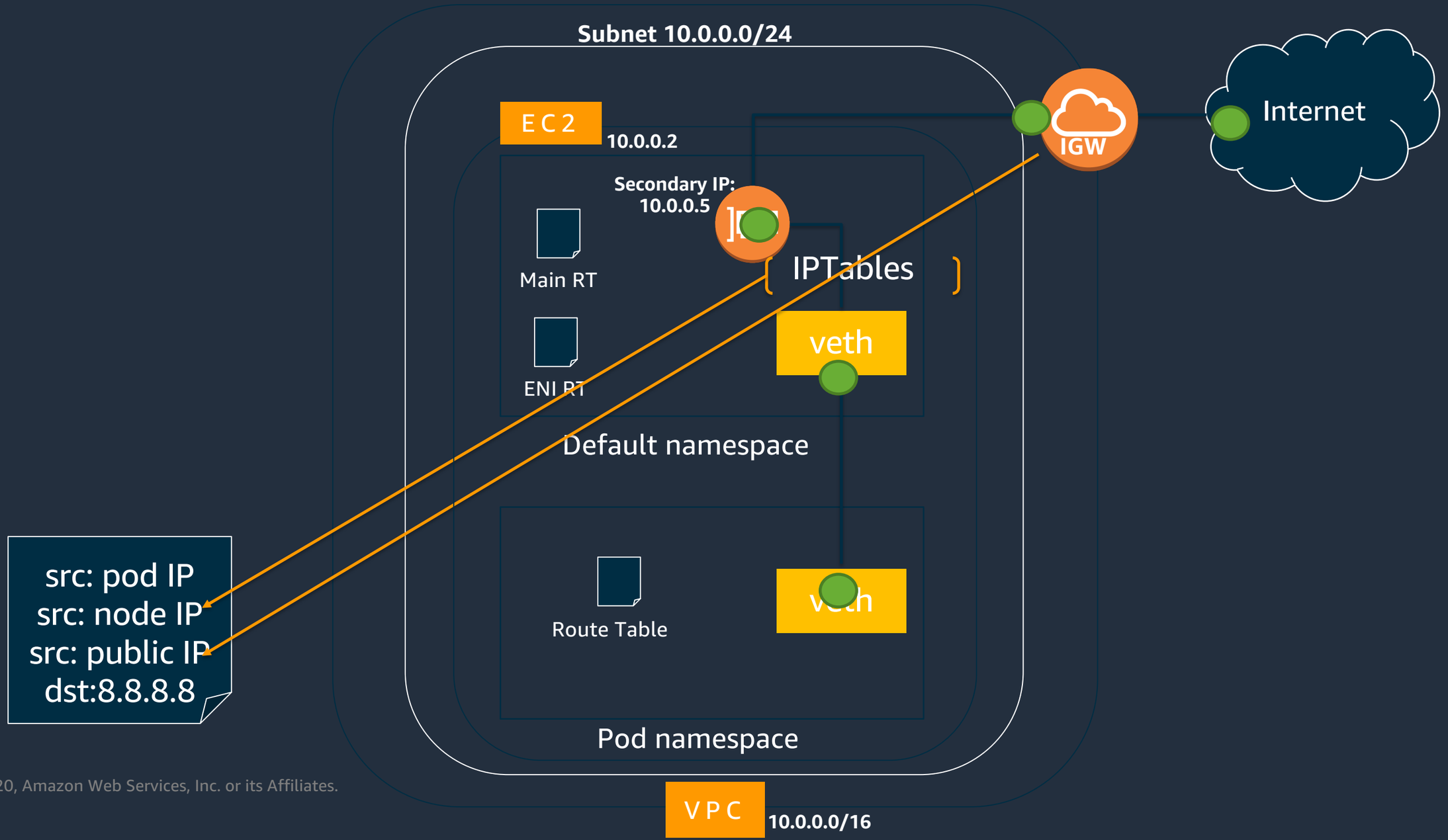


aws

Packet Flow: Pod to Pod



Packet Flow: Pod to Internet



aws

Amazon VPC CNI plugin – Understanding IP Allocation

Primary CIDR range

- RFC 1918 addresses ➔ 10/8, 172.16/12, 192.168/16
- Publicly routable CIDR block (since May 2019)

Used in EKS for:

- Pods
- X-account ENIs for (masters → workers) communication (exec, logs, proxy etc.)
- Internal Kubernetes services network (10.100/16 or 172.20/16)

Secondary CIDR

- non-RFC 1918 address blocks (100.64.0.0/10 and 198.19.0.0/16)

Used in EKS for Pods only

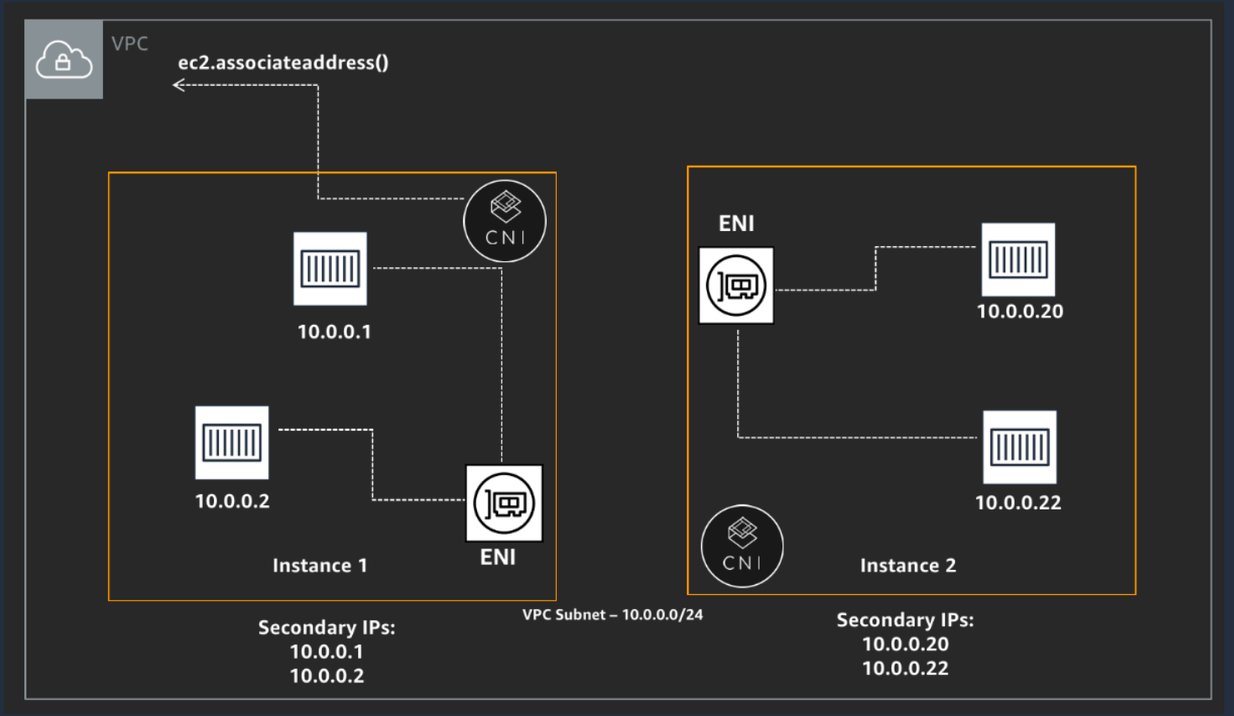
How?

ENIConfig

aws

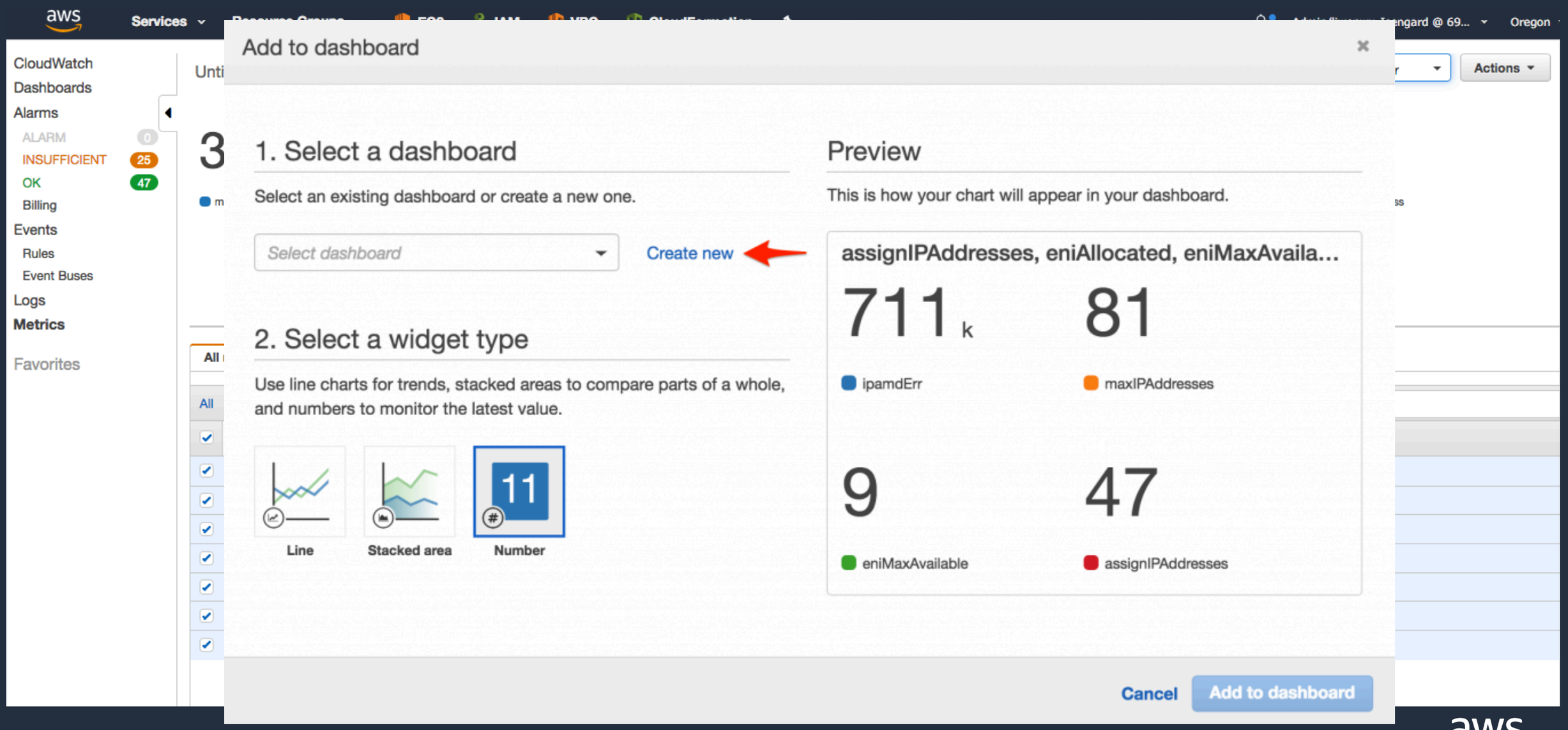
Amazon VPC CNI plugin - Configurability

- Custom Network Configs
- SNAT / External SNAT
- Configurable warm pool



aws

Introspecting the VPC CNI Plugin - CNI Metrics Helper



aws

Calico Network Policy

Kubernetes Network Policies enforce network security rules

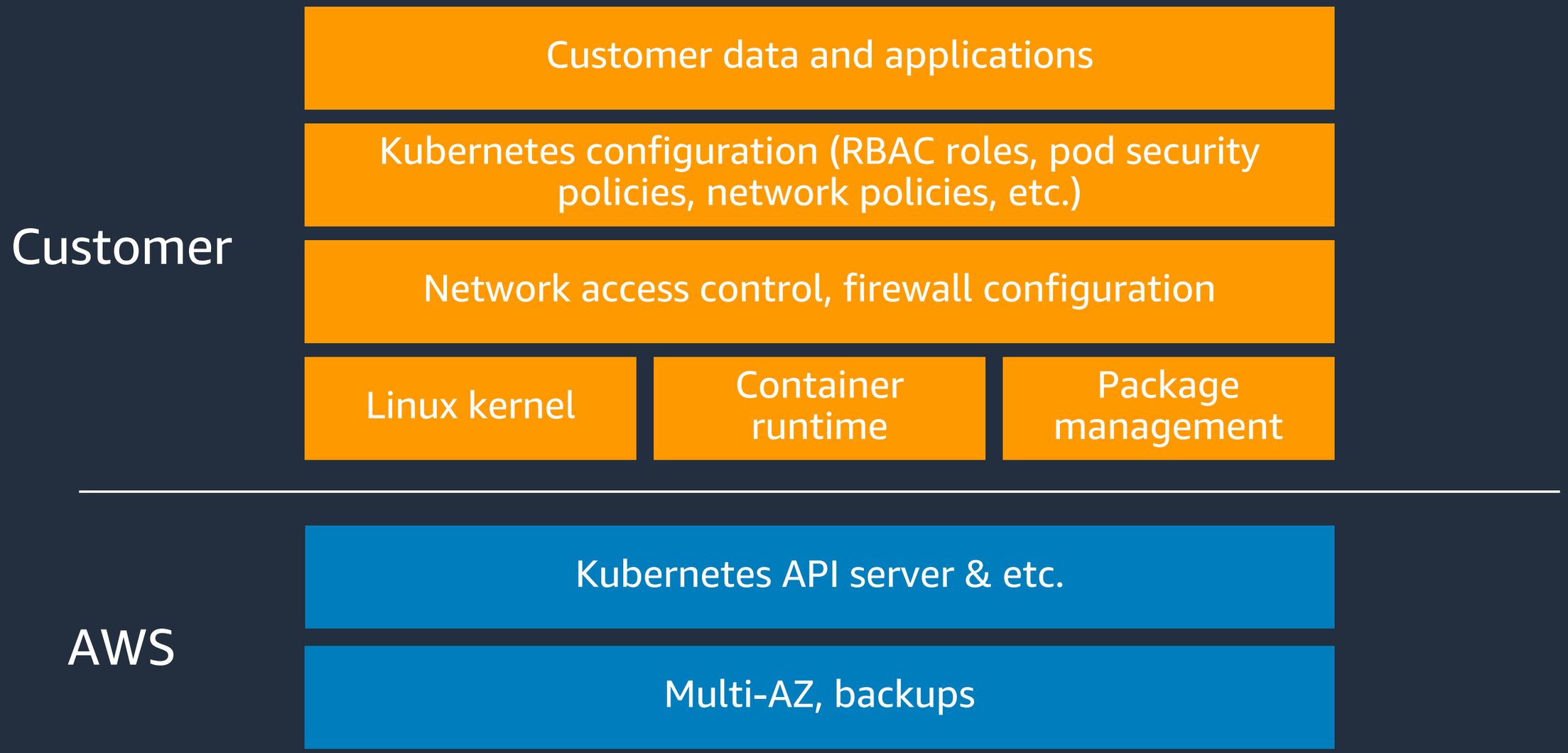
Calico is the leading implementation of the network policy API

Open source, active development (>100 contributors)

Commercial support available from Tigera

Security

The shared responsibility model

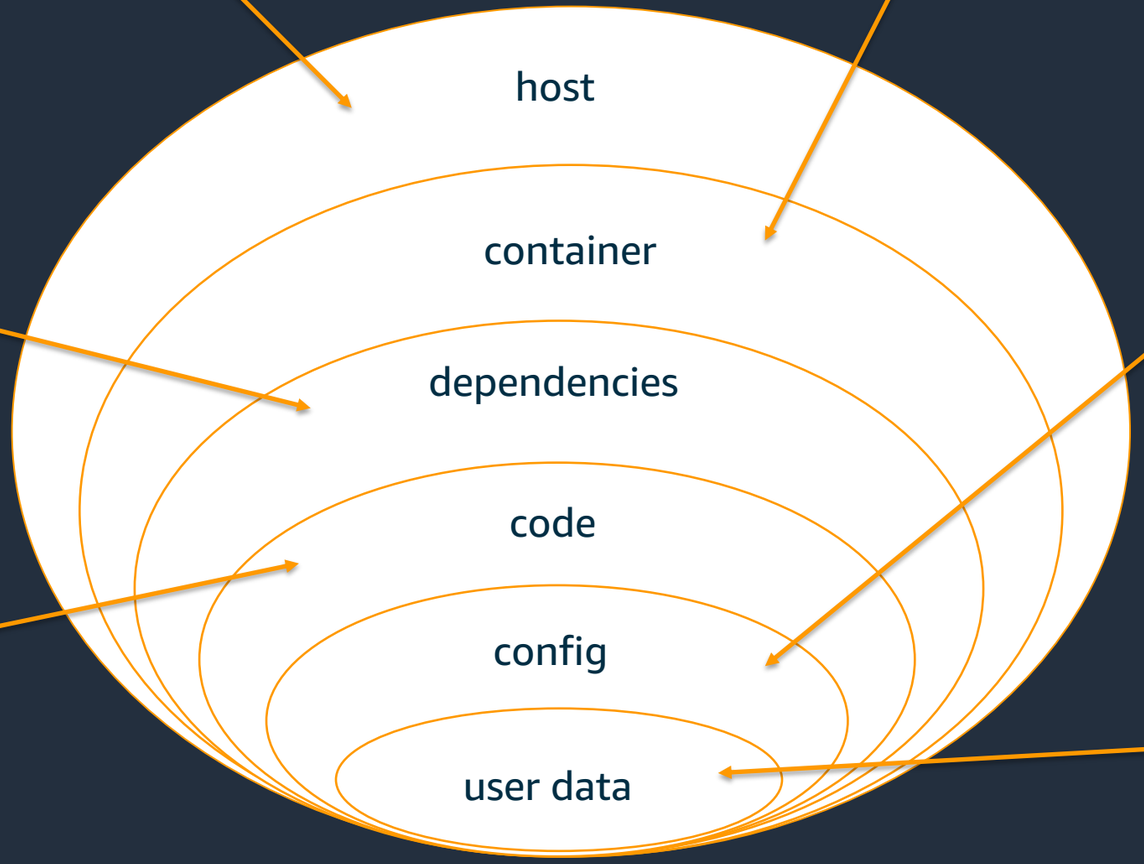


aws

Container security onion model: Defense in depth

- full blown distro (Ubuntu, AL) vs. minimal environment (container-optimized distribution)
- multi-tenancy requirements
- **gotchas**: Linux packages/CVEs, leaks, GDPR (in Europe)

- runtime/standards (OCI)
- immutability of images
- all containers share a kernel (mitigation: Firecracker)
- **gotchas**: unnecessary privileged users, no scans, trust



- code analysis
- source available?
- **gotchas**: big surface, many languages

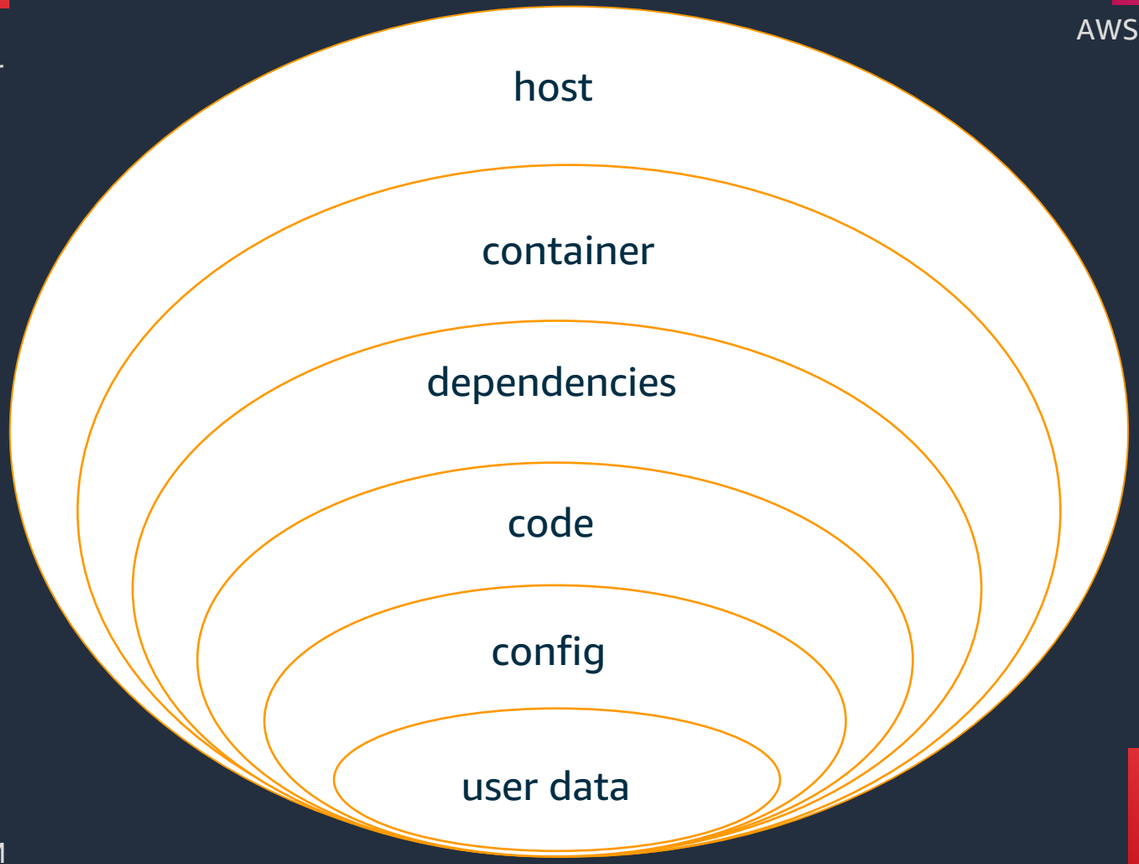
- sensitive config (passwords, API keys, etc.)
- **gotchas**: commits-to-source, non-separated access (dev has cleartext password)

- sanitizing user input
- static code analysis
- **gotchas**: log-leaking

- business core data
- Personal Identifiable Information (PII)
- **gotchas**: leaks, GDPR (in Europe)

aws

Security tooling



AWS Certificate Manager (ACM)

AWS Security Hub

IAM

Amazon Inspector

AWS CloudTrail

Amazon GuardDuty

AWS WAF

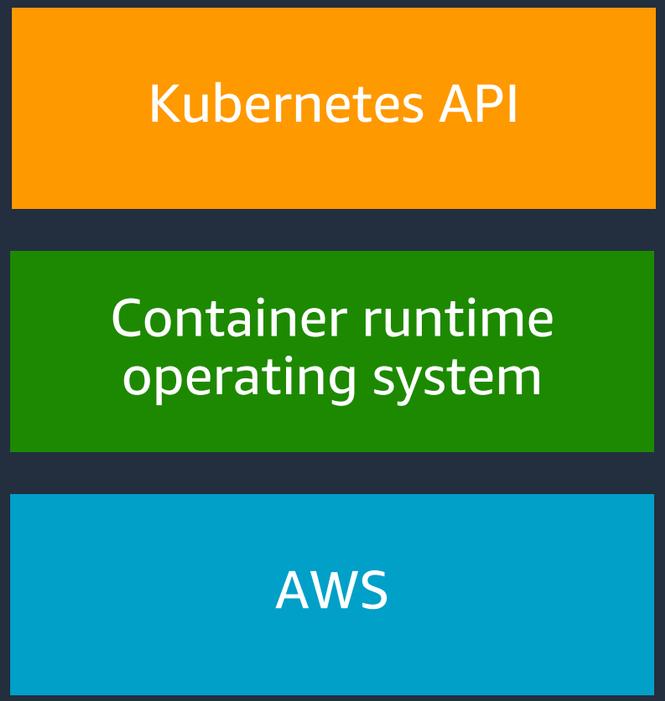
AWS KMS

AWS Secrets Manager

AWS CloudHSM

Amazon Macie

Areas of control



aws

Kubernetes controls

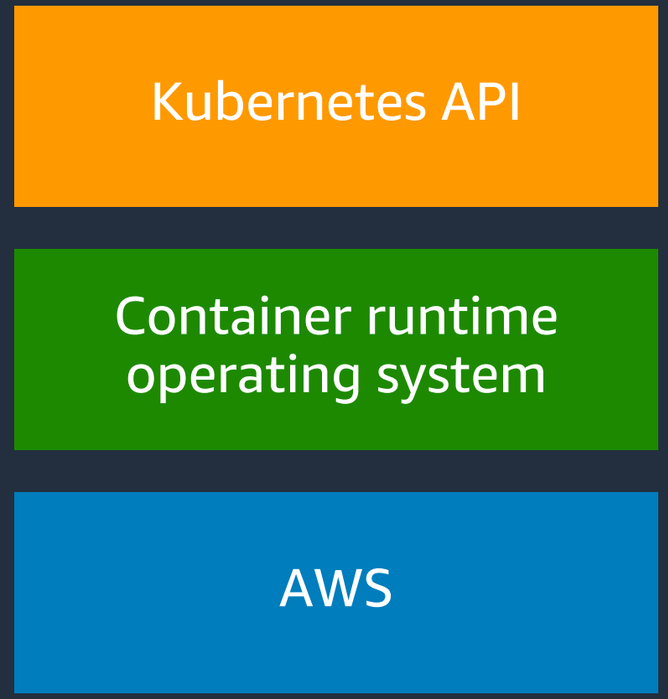
Namespaces
- Logical partition within a cluster

Service accounts
- Kubernetes identities assigned to pods

Resource quotas
- Limit total resources in a namespace

Network policies
- Pod-aware network controls

Limit ranges
- Limit pod, container, or volume resource sizes



aws

Kubernetes controls

Role-based access control (RBAC)

- Authorization policy API

Dynamic admission webhooks

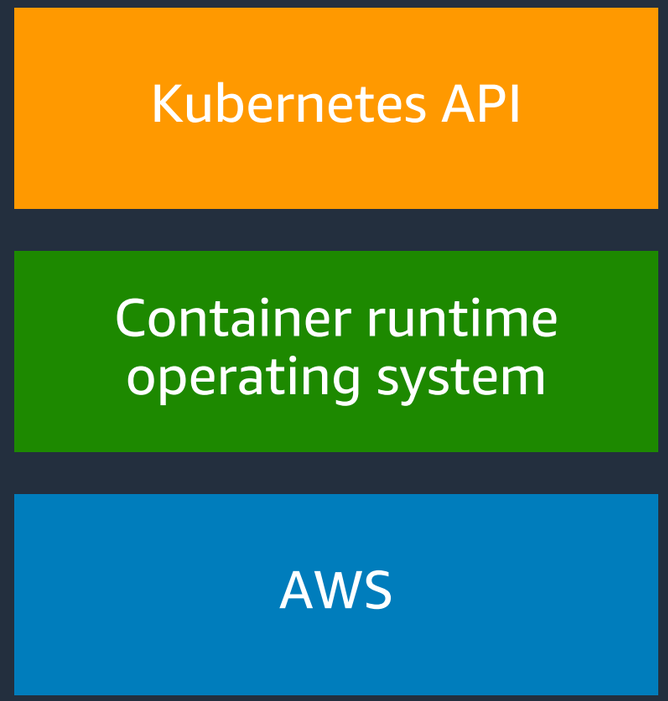
- Can place arbitrary restrictions on requests made to the API

API audit log

- API logging, including request or response bodies

Pod security policies

- Restrictions on pod and container permissions



aws

Pod security policies

Defines the conditions for a pod to run

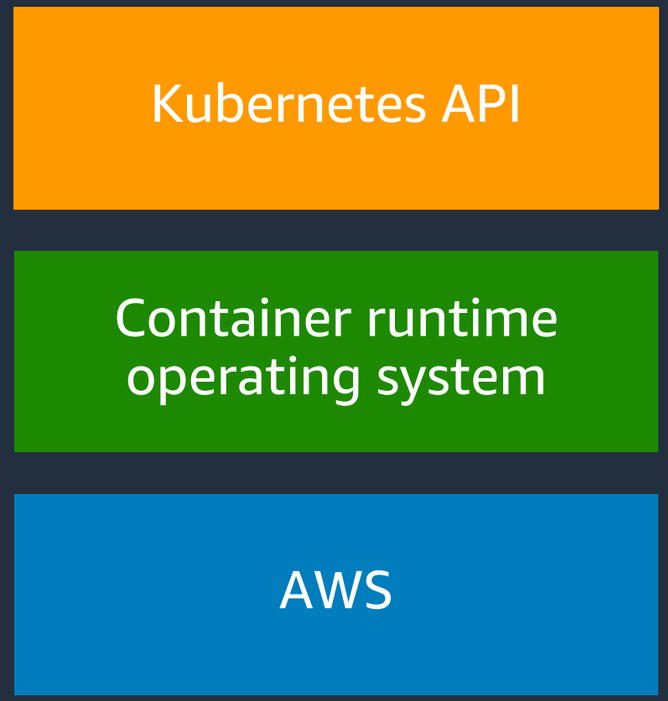
- Disallow running containers as root
- Disallow containers that require root privileges

Enforcement in the Kubernetes API

- A combination of an admission controller and authorization to grant service account access

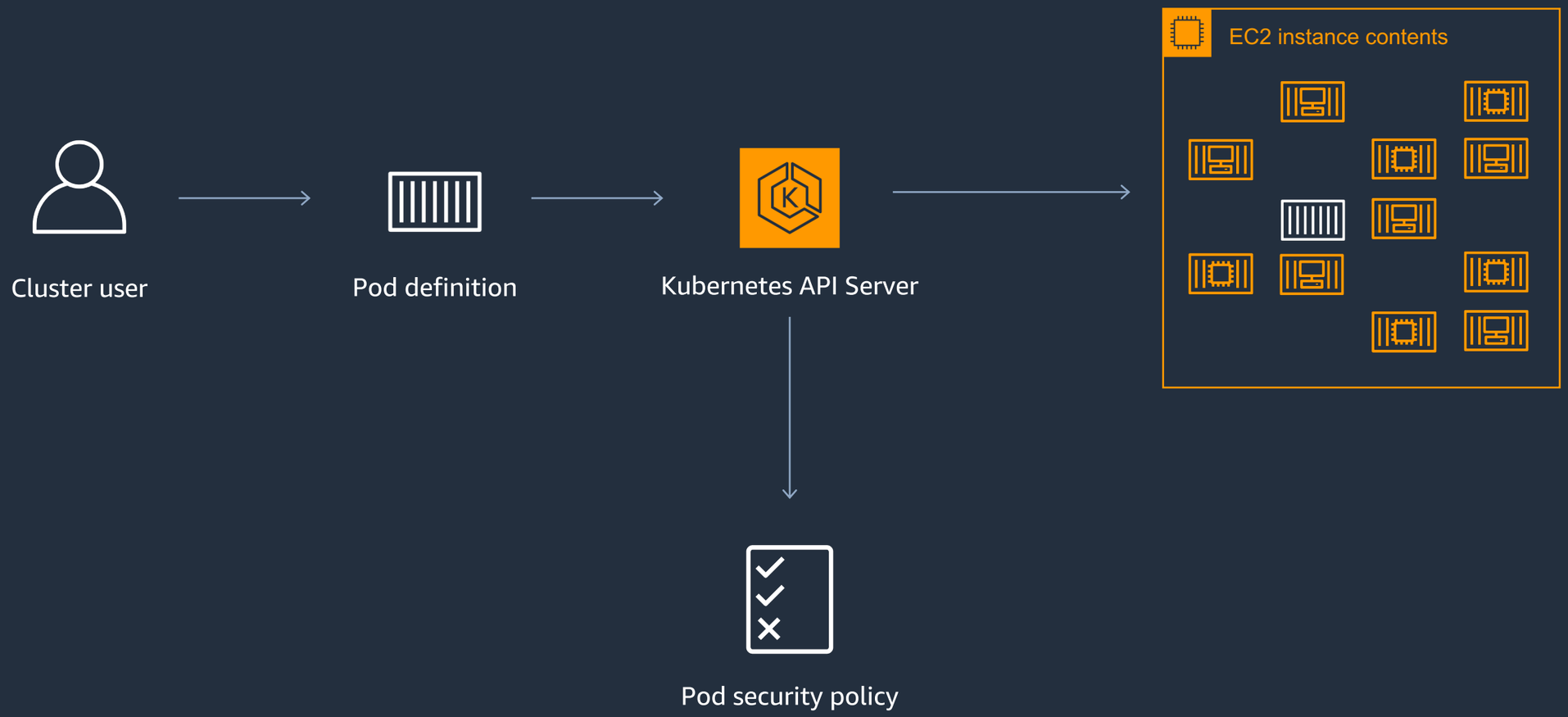
Supports multiple policies

- A policy for privileged containers
- A policy for unprivileged containers



aws

Pod security policies



A restrictive pod security policy example

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: restrictive
spec:
  privileged: false
  allowPrivilegeEscalation: false
  requiredDropCabilities: ['ALL']
  volumes: ['configMap', 'secret', 'projected’]
  hostnetwork: false
  hostIPC: false
  hostPID: false
  readOnlyRootFilesystem: true
```

A restrictive pod security policy example

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: restrictive
spec:
  privileged: false # Prevent escalation to root
  allowPrivilegeEscalation: false
  requiredDropCabilities: ['ALL’]
  volumes: ['configMap', 'secret', 'projected']
  hostnetwork: false
  hostIPC: false
  hostPID: false
  readOnlyRootFilesystem: true
```

A restrictive pod security policy example

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: restrictive
spec:
  privileged: false
  allowPrivilegeEscalation: false
  requiredDropCabilities: ['ALL']
  volumes: ['configMap', 'secret', 'projected'] # Volumes are restricted to a whitelist
  hostnetwork: false
  hostIPC: false
  hostPID: false
  readOnlyRootFilesystem: true
```

A restrictive pod security policy example

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: restrictive
spec:
  privileged: false
  allowPrivilegeEscalation: false
  requiredDropCabilities: ['ALL']
  volumes: ['configMap', 'secret', 'projected']
  hostnetwork: false  # Disallows direct use of host networking namespace and resources
  hostIPC: false
  hostPID: false
  readOnlyRootFilesystem: true
```

A restrictive pod security policy example

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: restrictive
spec:
  privileged: false
  allowPrivilegeEscalation: false
  requiredDropCabilities: ['ALL']
  volumes: ['configMap', 'secret', 'projected']
  hostnetwork: false
  hostIPC: false
  hostPID: false
  readOnlyRootFilesystem: true # Prevents writes to root filesystem
```

Container image management controls

Amazon Elastic Container Registry (Amazon ECR) endpoint policy

- Set condition "PrincipalOrgID"
- Prevents the pushing and pulling of images outside

Open policy agent (OPA)

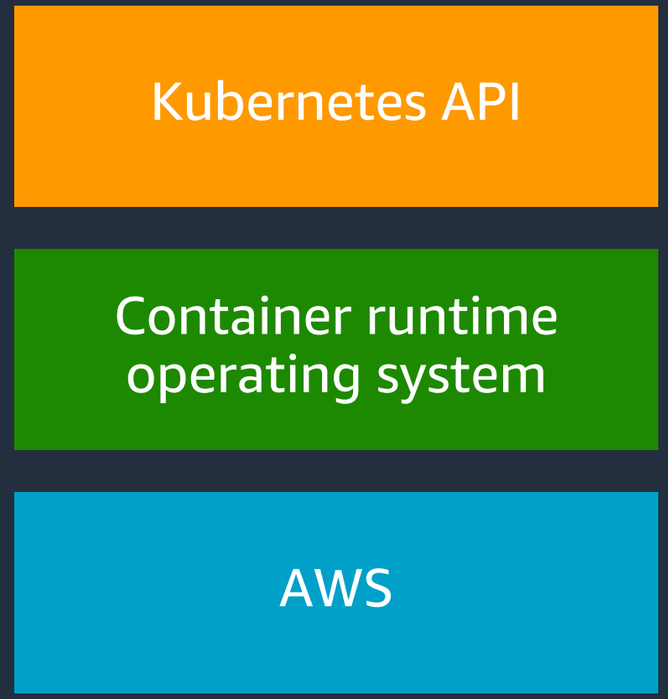
- Restricts the pulling to whitelisted image registries

OPA for MITM attack mitigation to enforce signing

- Restrict the pulling of signed images only

Container runtime API access restriction

- Ensure API is not accessible outside of unix socket



aws

AWS controls

Control plane logging

- Including Kubernetes API audit logs

Endpoint access

- Private endpoint access
- Public CIDR restrictions

Authentication control

- Manage which AWS Identity and Access Management (IAM) identities can authenticate to the cluster

Amazon ECR image scanning

- Scan container images on push

IAM roles for service accounts

- Assign IAM roles to pods via the service account

aws

IAM roles for service accounts

Secure

- IAM policy restrictions can restrict roles to service accounts or namespaces
- Enables isolated AWS permissions per service account
- Credentials are automatically rotated
- The cluster's signing key is automatically rotated

Easy integration

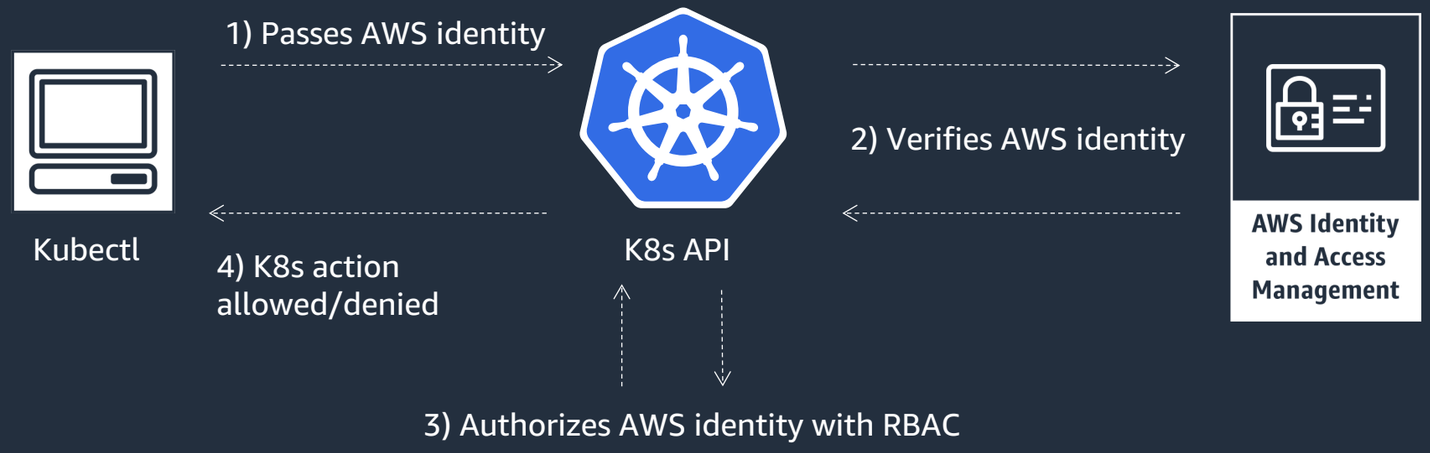
- Annotate the service account
- Built into the default credential chains in the AWS SDKs and AWS CLI

Auditable

- Service account names are logged in AWS CloudTrail

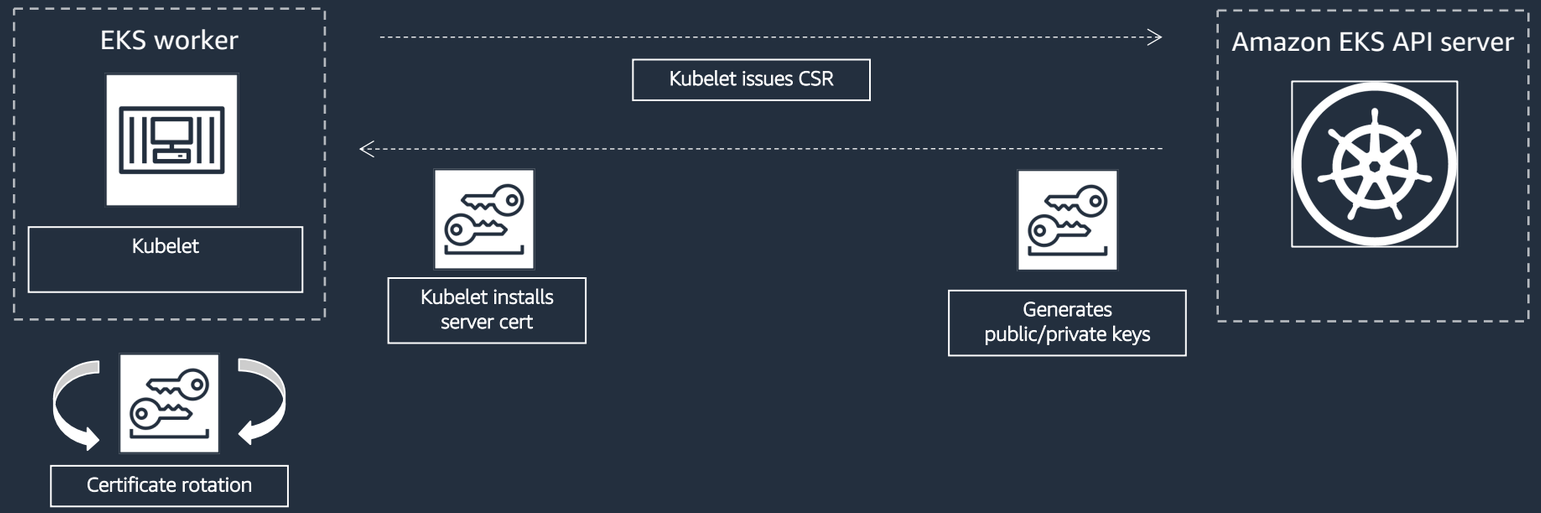
aws

IAM authentication



PKI configuration

Each Amazon EKS cluster is a unique CA



NEW!

IAM for pods

Set IAM access permissions at the pod level

Enables multiple applications with different permission sets to share the same nodes

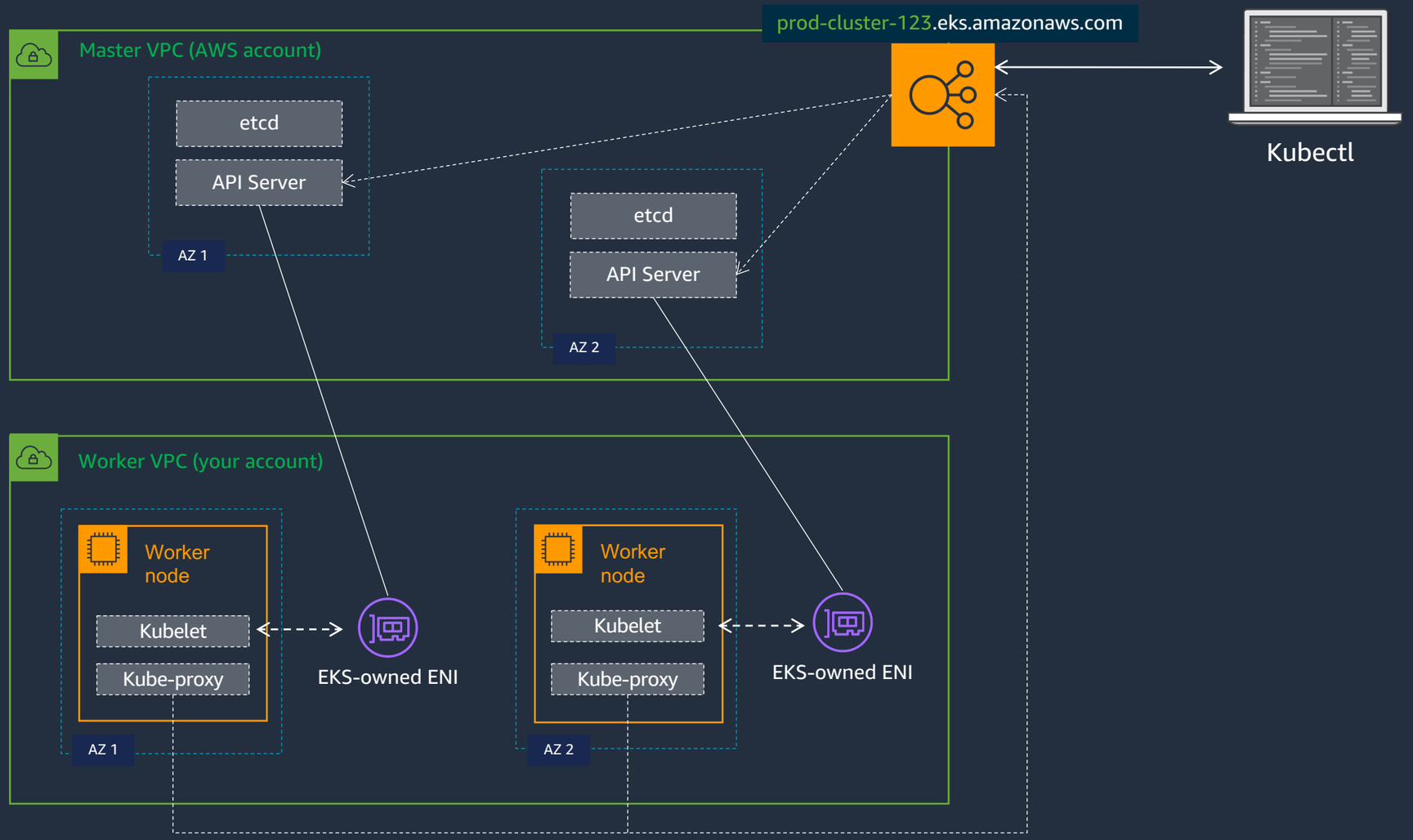
Built using Kubernetes primitives, minimal user configuration

aws

API-server endpoint access control

Public == true

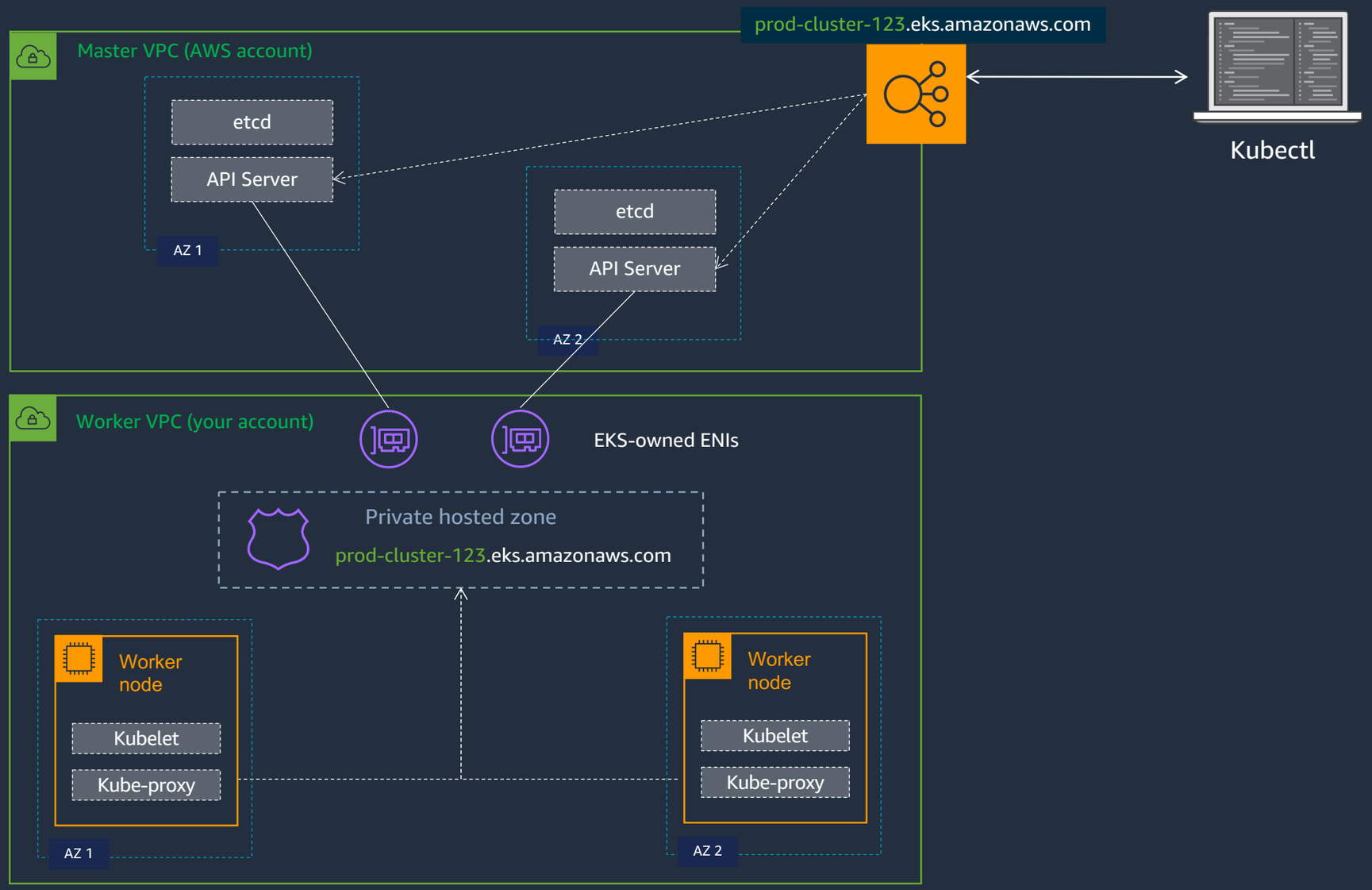
Private == false



API-server endpoint access control

Public == true

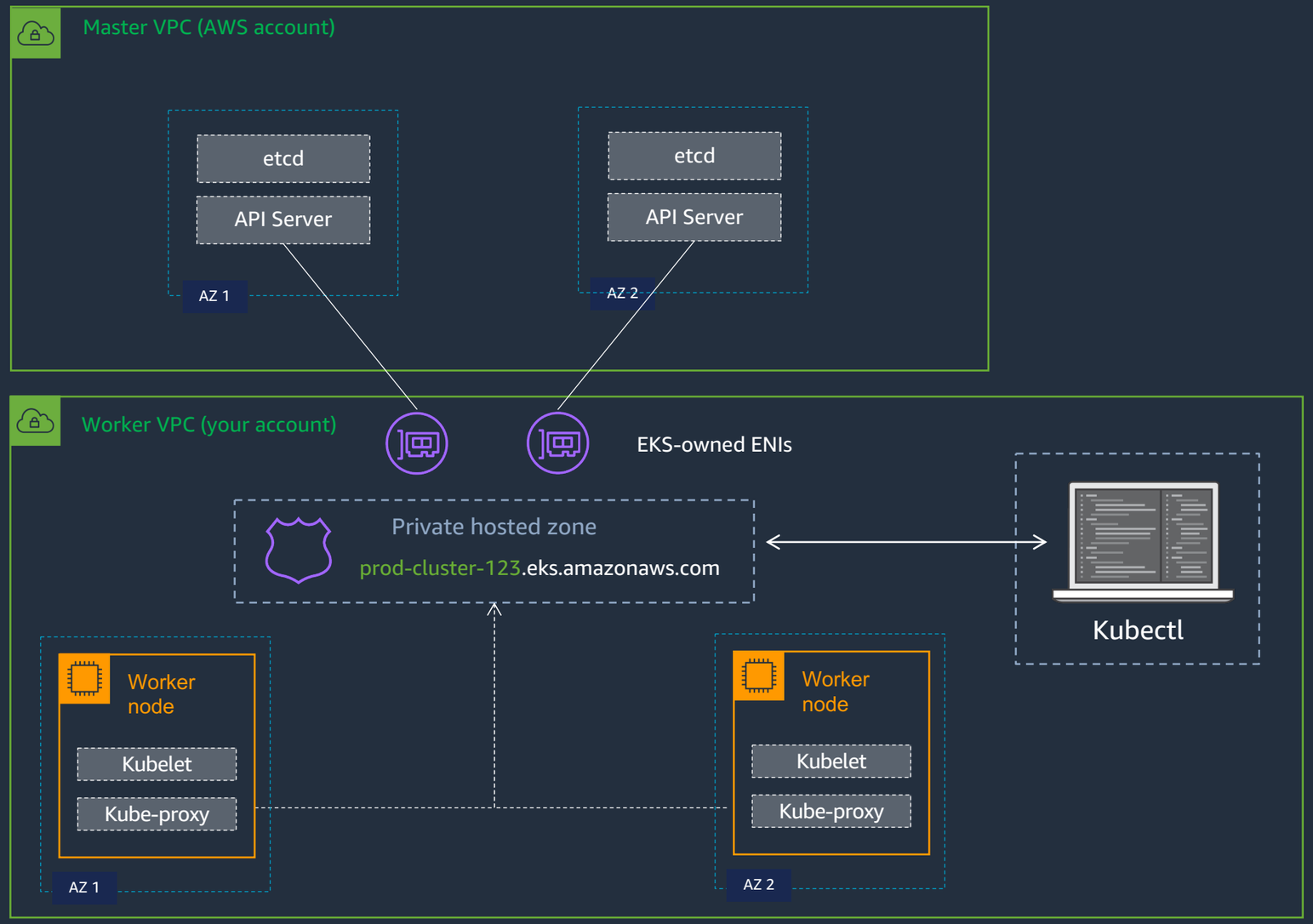
Private == true



API-server endpoint access control

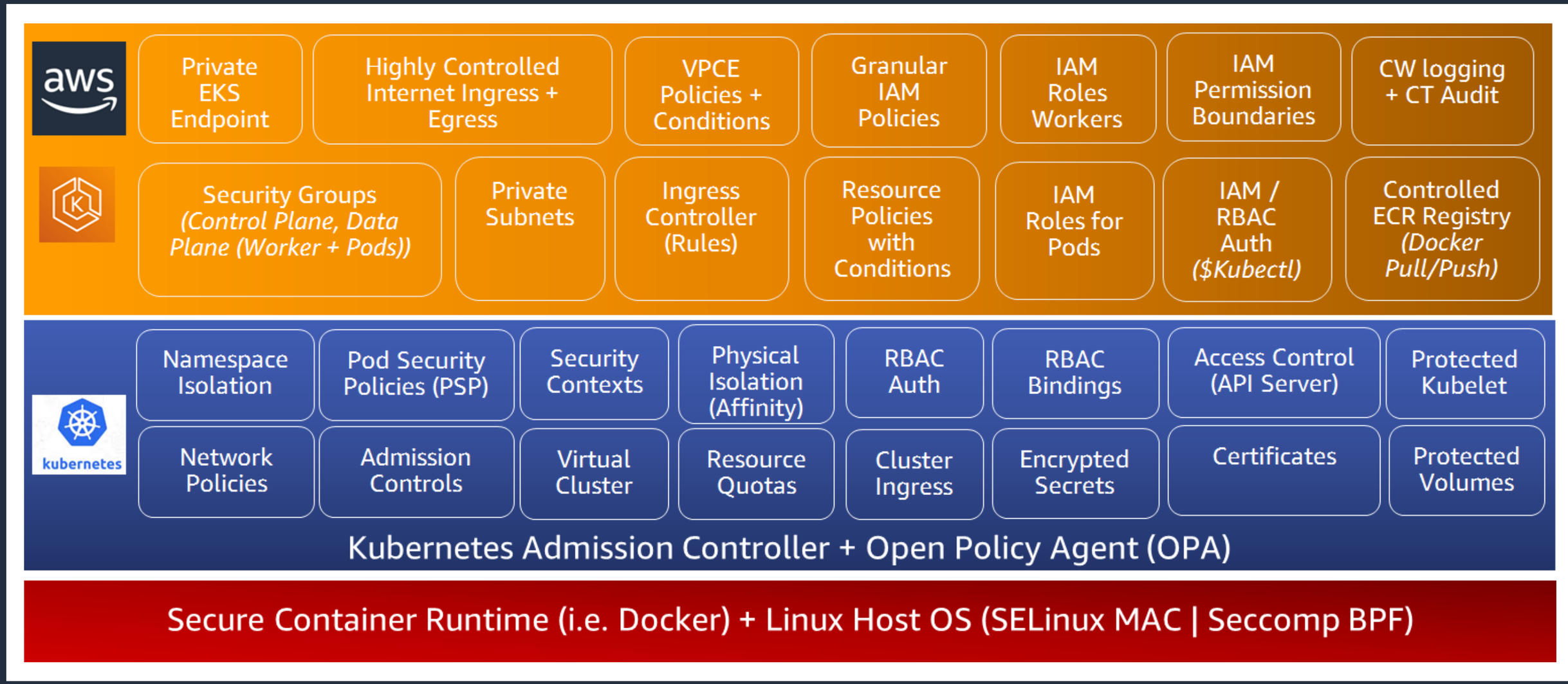
Public == false

Private == true



Example of defense-in-depth strategy

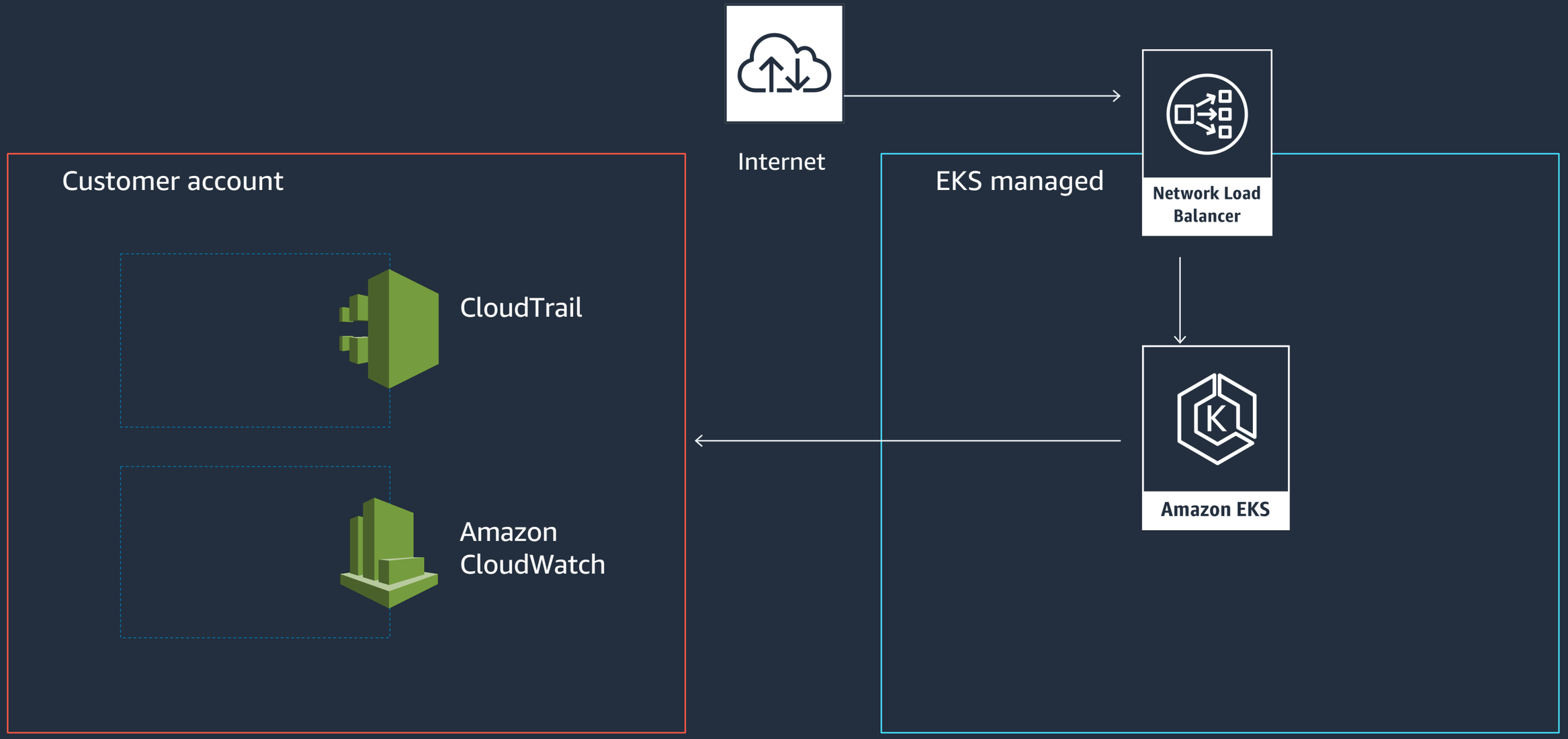
Multi-layered approach - AWS, Kubernetes, Container Runtime and Linux Security Controls



Logging and Monitoring

aws

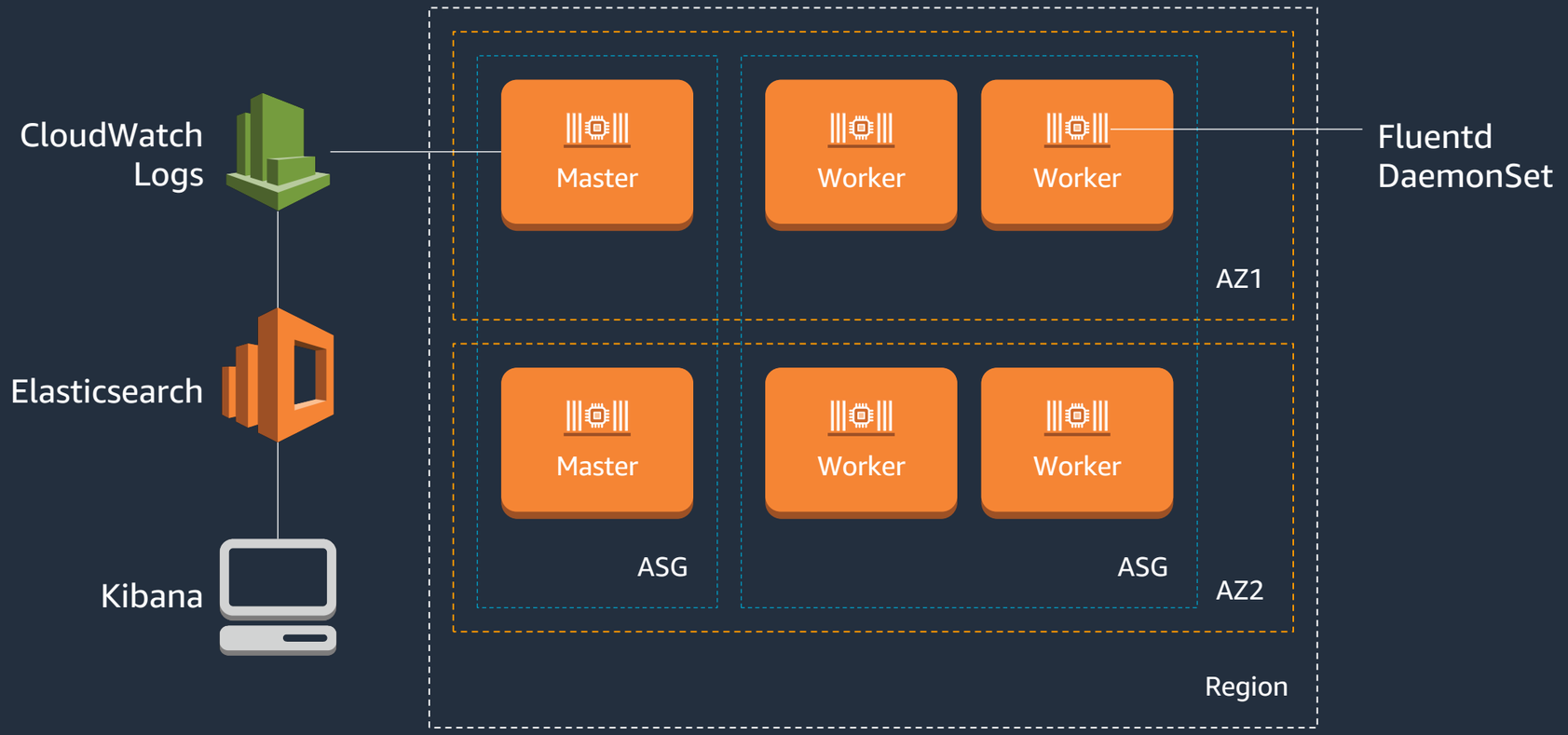
Amazon EKS logging



EKS Logging

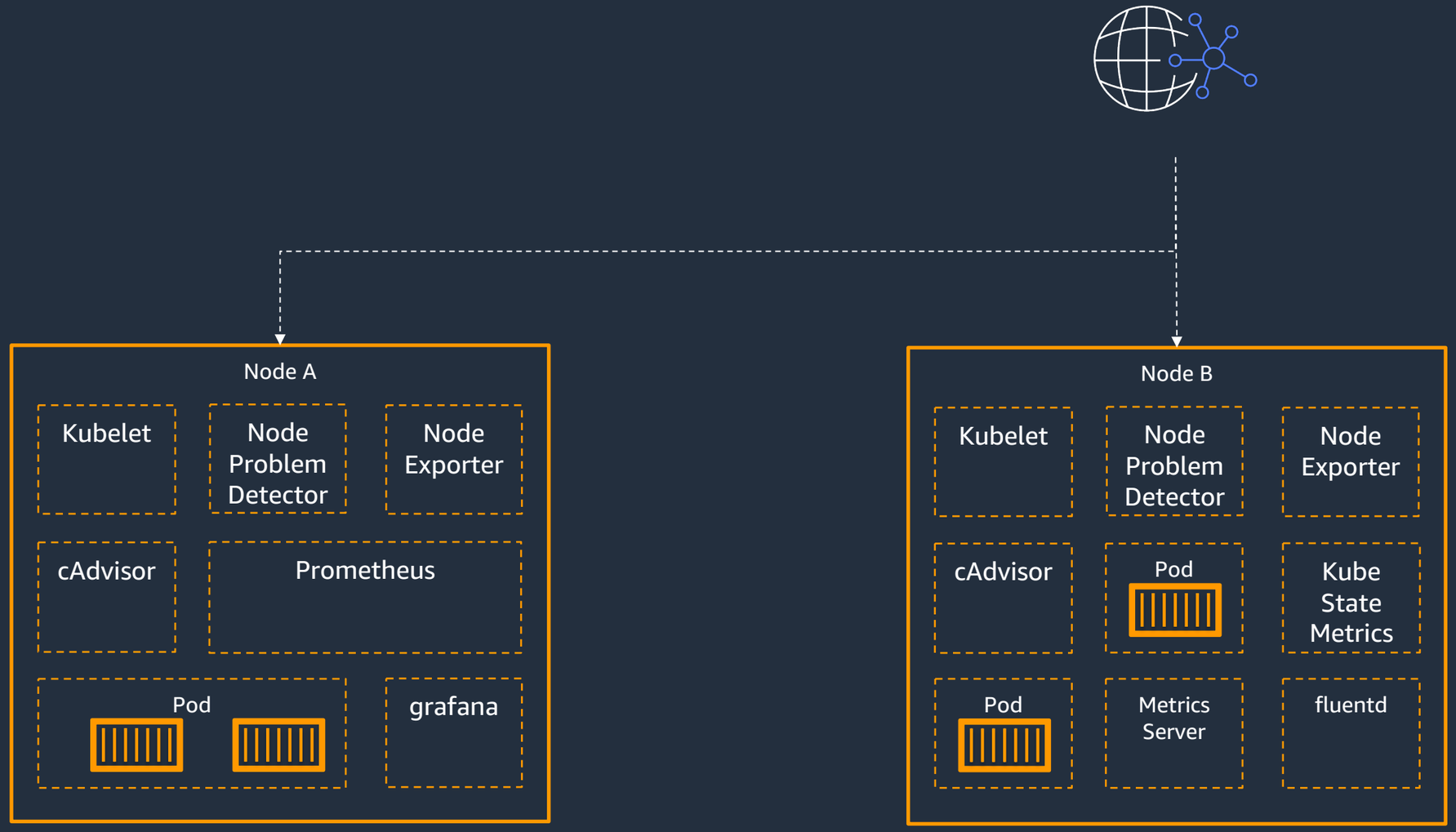
Kubectl logs

Elasticsearch (index),
Fluentd (store), and
Kibana (visualize)



aws

Metrics Sources

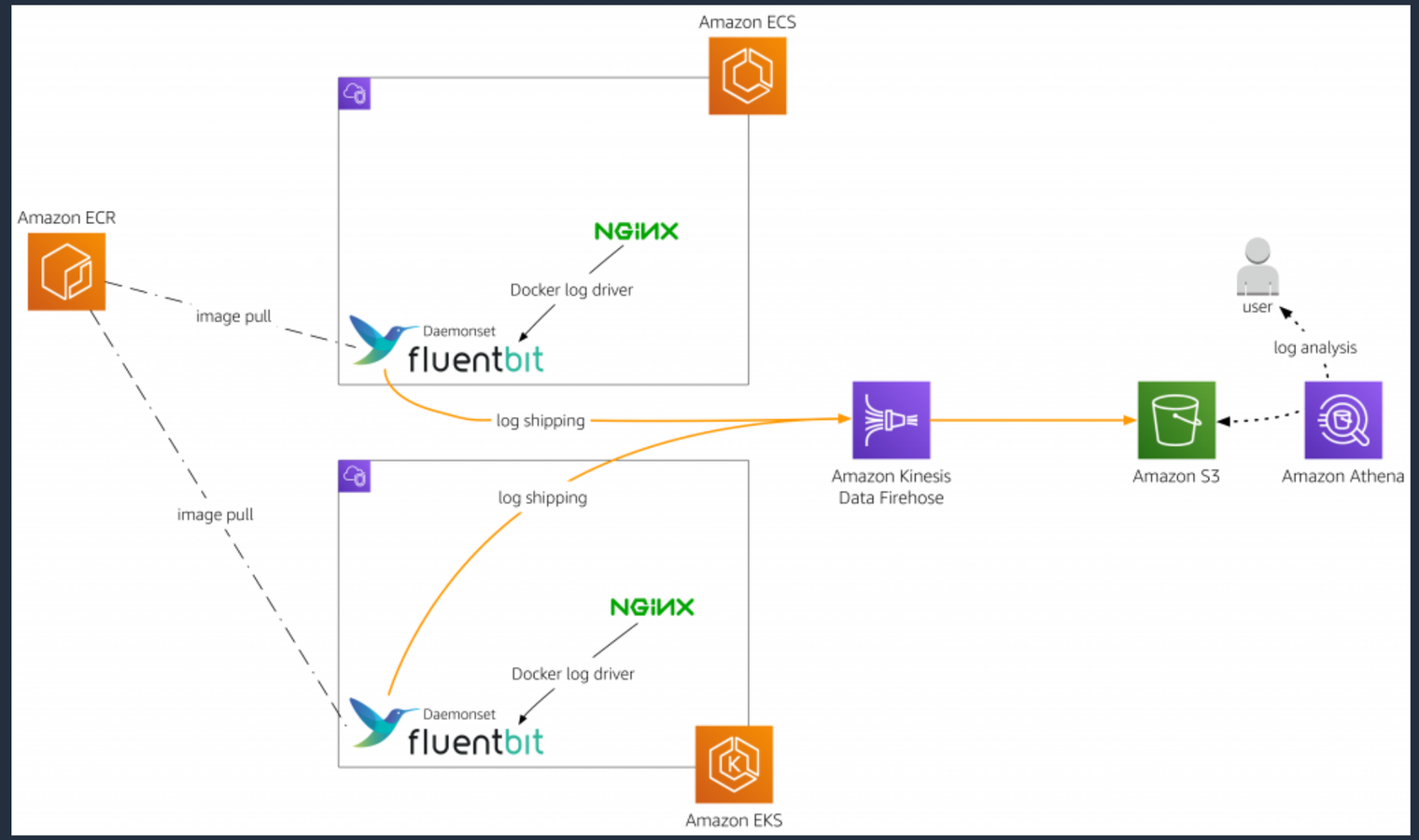


aws

NEW!

Logging with FluentBit

- New AWS Fluent Bit container plugin
- Optimize costs. Route logs from Amazon EKS and Amazon ECS clusters directly to Amazon S3 and query with Amazon Athena
- Open source
- More resource-efficient than Fluentd. Tests show Fluentd uses 4x more CPU and 6x more memory



https://aws.amazon.com/blogs/opensource/centralized-container-logging-fluent-bit/

CloudWatch Container Insights

NEW!

Gives you complete visibility into your cloud resources and applications so you can monitor, troubleshoot, and remediate issues

aws

CloudWatch Container Insights

NEW!

A fully managed observability service for monitoring, troubleshooting, and alarming on your containerized applications and microservices

- ✓ Collects, aggregates, and summarizes
- ✓ Reliable, secure metrics and logs collection
- ✓ Automated dashboards and analysis
- ✓ Observability experience across metrics, logs, traces
- ✓ Ad hoc analytics

aws

Images on DockerHub

NEW!

Performance Metrics—CloudWatch Agent:
https://hub.docker.com/r/amazon/cloudwatch-agent

- Tag: latest

Logs—Fluent Bit:
https://hub.docker.com/r/amazon/aws-for-fluent-bit

- Tag: latest

Logs—Fluentd:
https://hub.docker.com/r/fluent/fluentd-kubernetes-daemonset

- Tag: v1.3.3-debian-cloudwatch-1.4

aws

Container Insights available now

NEW!

1. **Fully managed**, AWS-native observability service providing automated summary and analysis of compute capacity

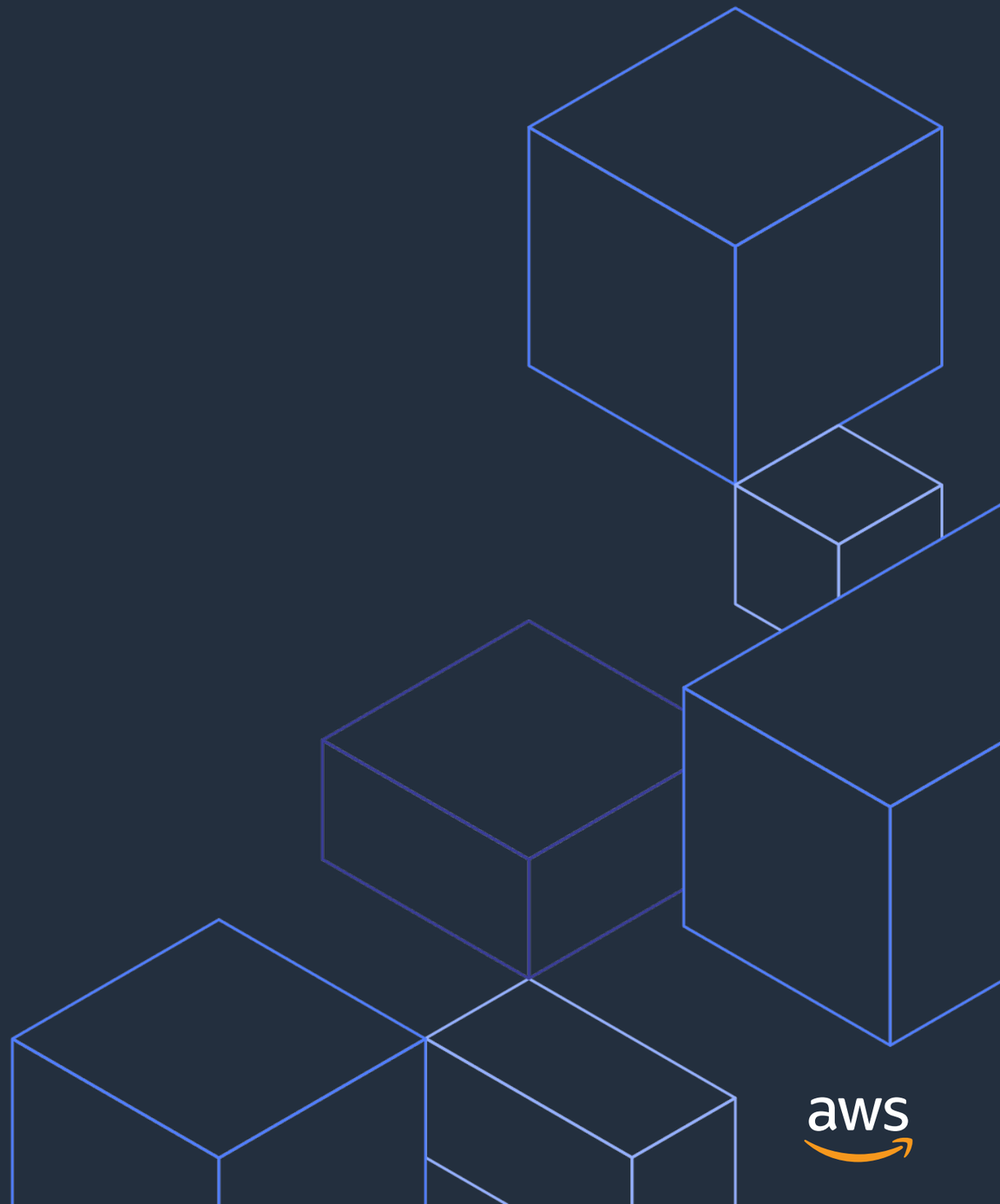
2. **Reliable and secure collection** of application logs with built-in analytics capabilities

3. **Prebuilt visualization** to summarize cluster and node errors

4. **Application & microservice tracing**—troubleshoot and debug application & microservice

aws

Storage



Container storage interface (CSI)

A flexible standard for orchestration
and storage provider connections

We support the CSI standard through the following drivers:

Amazon Elastic Block Store: Amazon EBS CSI Driver

Amazon Elastic File System: Amazon EFS CSI Driver

Amazon FSx for Lustre: Amazon FSx CSI Driver

aws

Storage volume lifecycle

Provisioning

- Static
- Dynamic*

Binding

- Control loop watches for PVC requests and satisfies if PV is available
- For Dynamic, PVC will provision PV
- PVC to PV binding is one-to-one mapping

Using

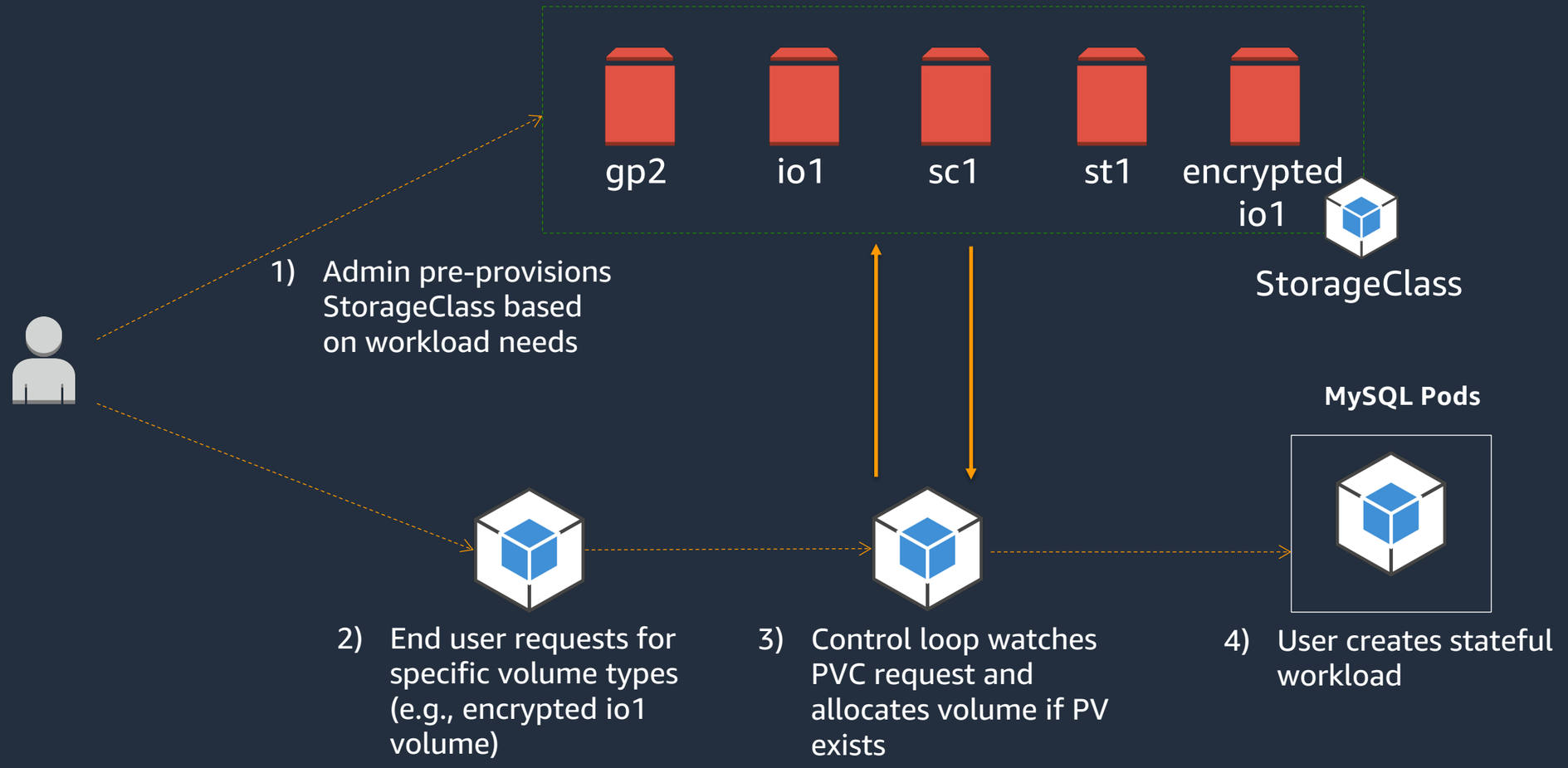
- Cluster mounts volume based on PVC

Reclaiming

- Retain (default)
- Recycle
- Delete

aws

What if I need a specific volume type?



aws

Compare storage systems

	Throughput	Access mode	Availability Zone	Integrate S3
EBS	~1000MB/s	Single node	Single AZ	No
EFS	~250–3,000 MB/s	Multiple nodes	Multiple AZs	No
FSx for Lustre	~720–23,040 MB/s	Multiple nodes	Single AZ	Yes

Access mode versus performance

- Single node versus multiple nodes
- High throughput versus medium throughput

	Throughput	Access mode	Availability Zone	Integrate S3
EBS	~1000MB/s	Single node	Single AZ	No
EFS	~250–3,000 MB/s	Multiple nodes	Multiple AZs	No
FSx for Lustre	~720–23,040 MB/s	Multiple nodes	Single AZ	Yes

Handling AZ failure

- Application layer handles AZ failure (Amazon EBS)
- Storage layer handles AZ failure (Amazon EFS)

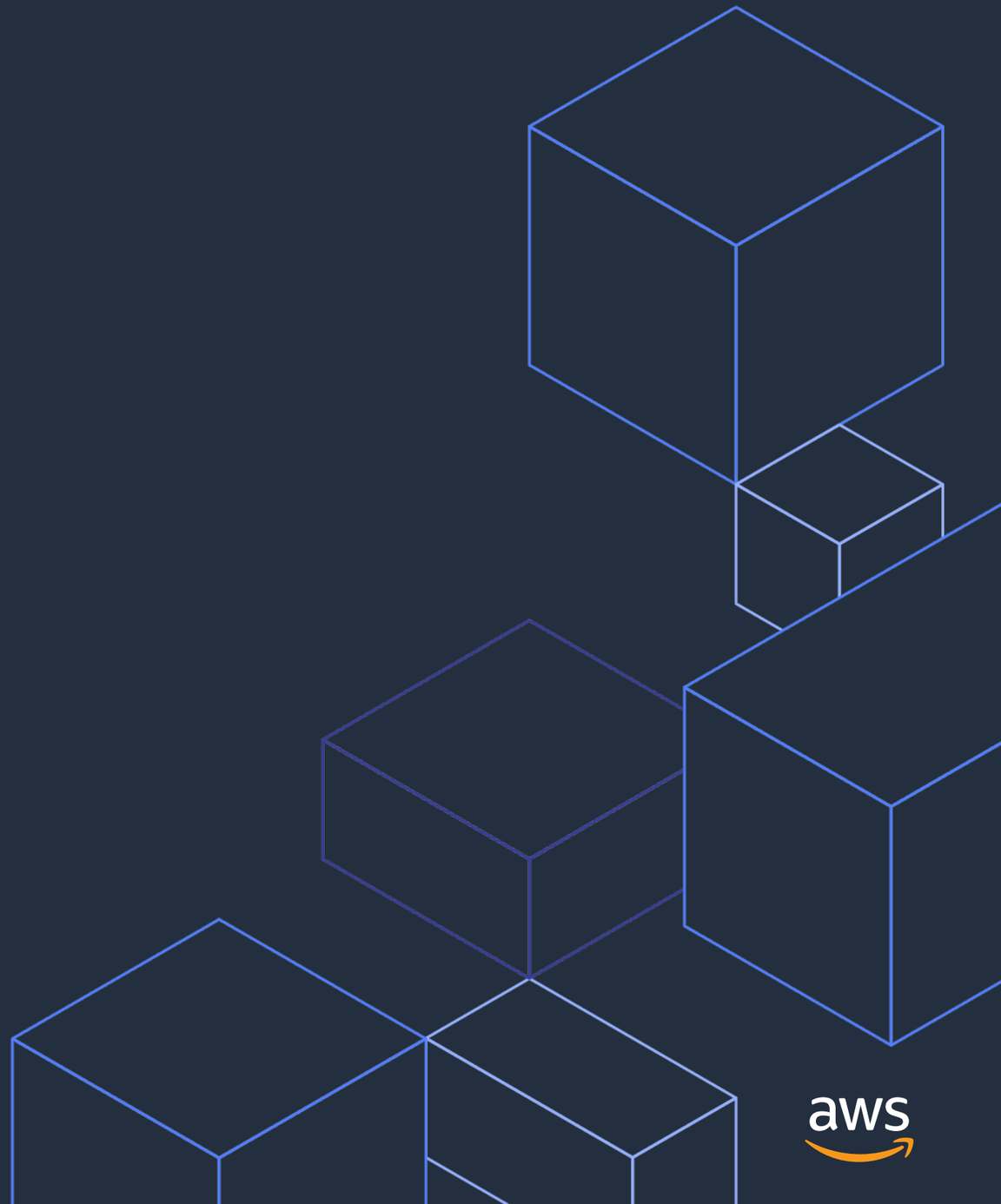
	Throughput	Access mode	Availability Zone	Integrate S3
EBS	~1000MB/s	Single node	Single AZ	No
EFS	~250–3,000 MB/s	Multiple nodes	Multiple AZs	No
FSx for Lustre	~720–23,040 MB/s	Multiple nodes	Single AZ	Yes

Data caching versus data persistence

- Use FSx for Lustre file system as data caching for Amazon S3
- Use Amazon EFS file system as static content store / ML data store

	Throughput	Access mode	Availability Zone	Integrate S3
EBS	~1000MB/s	Single node	Single AZ	No
EFS	~250–3,000 MB/s	Multiple nodes	Multiple AZs	No
FSx for Lustre	~720–23,040 MB/s	Multiple nodes	Single AZ	Yes

Using Spot Instances with EKS



Amazon EC2 purchasing options

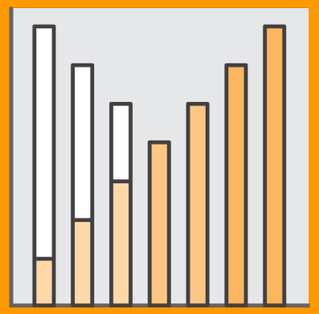
On-Demand Instances

Pay for compute capacity by **the second** with no long-term commitments

Spiky workloads, to define needs

Reserved Instances

Commit for one or three years and receive a **significant discount** on On-Demand prices



Committed, steady-state usage

Spot Instances

Spare Amazon EC2 capacity at **savings of up to 90%** off On-Demand prices

Fault-tolerant, flexible, stateless workloads

aws

Why Spot Instances?

Inexpensive

Significant price savings of up to 90% over On-Demand Instances. Perfect for stateless and fault-tolerant workloads.

Faster results

Easily run multiple projects simultaneously and speed up job flows to generate business results faster and innovate faster.

Easy access

Launch Spot Instances via Auto Scaling groups, Spot Fleet, EC2 Fleet, and RunInstances. Integrated with other AWS and third-party services.

Resource flexibility

Flexibility of ad hoc provisioning for multiple instance types with an option to hibernate, stop, or terminate instances when reclaimed.

aws

Amazon EC2 Spot integrations

Auto Scaling

AWS Batch

Amazon EMR

AWS Data Pipeline

Amazon Elastic Container Service

Amazon Elastic Container Service for Kubernetes

AWS CloudFormation

cloudera

AWS Thinkbox Deadline

docker

kubernetes

Bamboo

Jenkins

Qubole

databricks

HashiCorp Terraform

MESOS

aws

EC2 Spot pools – Flexibility and diversification

C4	1a	1b	1c	On Demand
8XL	$0.50	$0.27	$0.29	$1.76
4XL	$0.21	$0.30	$0.16	$0.88
2XL	$0.08	$0.07	$0.08	$0.44
XL	$0.04	$0.05	$0.04	$0.22
L	$0.01	$0.01	$0.04	$0.11

Each instance family

Each instance size

Each Availability Zone (69)

In every Region (22)

Is a separate Spot pool

aws

Containers + Spot = A match made in heaven

- ✓ Containers are often stateless, fault-tolerant, and a great fit for Spot Instances
- ✓ Deploy containerized workloads and easily manage clusters at any scale at a fraction of the cost with Spot Instances
- ✓ Spot Instances can be used with Amazon ECS or Kubernetes to run any containerized workload

skyscanner

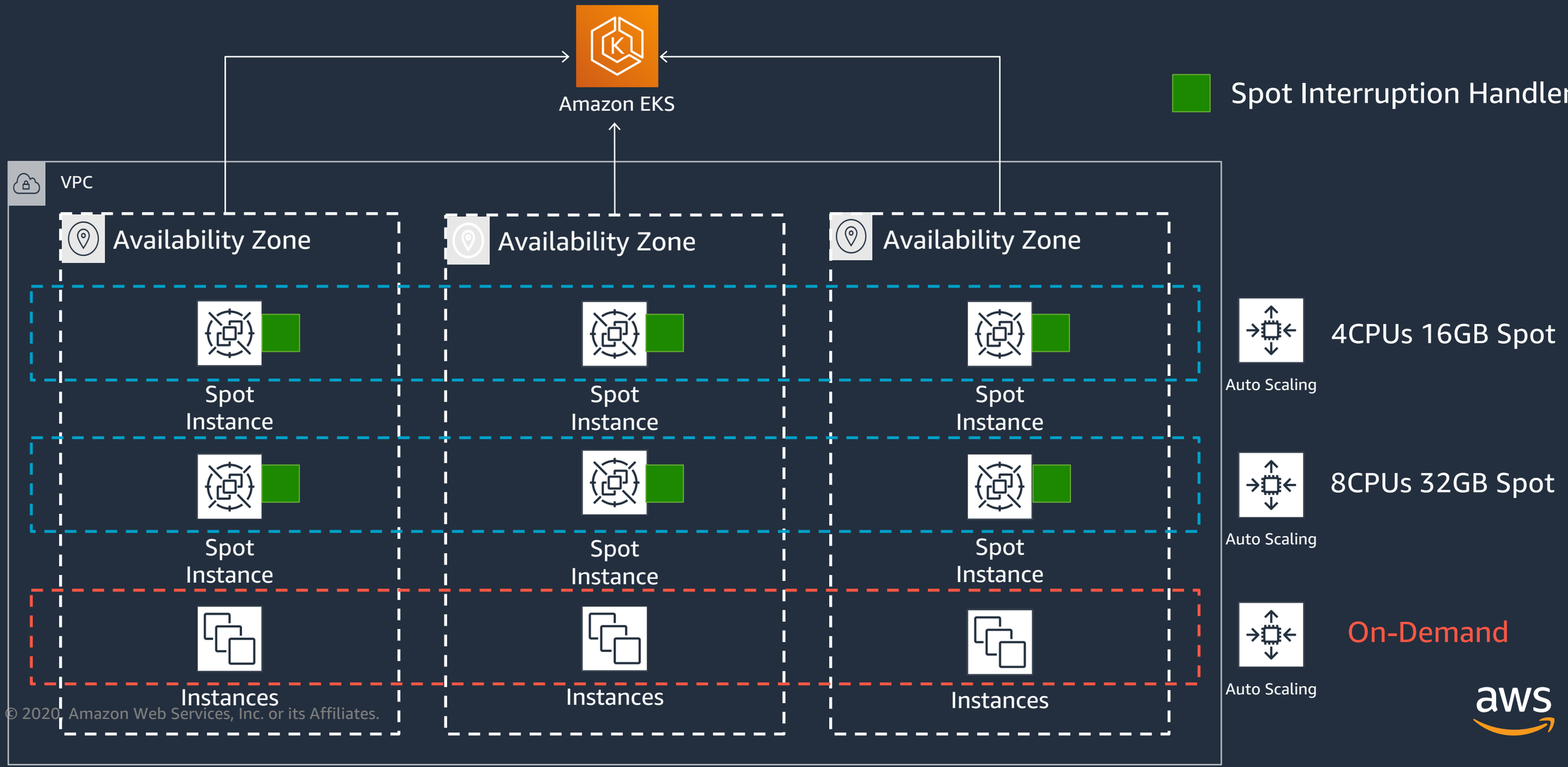
Skyscanner is a travel-fare aggregator website and travel metasearch engine based in Edinburgh, Scotland

"We are currently tracking **74% saving** over all regions."

—Paul Gillespie,
Principal Architect/Tribe Lead

lyft Yelp MapBox Expedia Caltech aws

Multiple Instances Types & Purchase Options in ASG

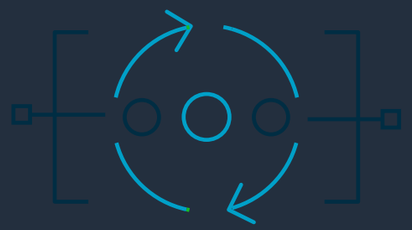


What about interruptions?

Minimal interruptions

Over **95%** of the instances were not interrupted in the last 3 months

The work you are doing to make your applications fault-tolerant also benefits Spot



Spot is optimized for stateless, fault-tolerant, or flexible workloads.

Any application that can can have part or all of the work, paused and resumed or restarted, can use Spot.

Check for 2-minute instance termination notice via instance metadata or CloudWatch Events and ***automate*** *by*:

- ☑ Checkpointing
- ☑ Draining from ELB
- ☑ Using stop-start and hibernate to restart faster

aws

Handling Spot interruptions and DaemonSet

https://github.com/kube-aws/kube-spot-termination-notice-handler

```
NOTICE_URL=${NOTICE_URL:-http://169.254.169.254/latest/meta-data/spot/termination-time}

echo "Polling ${NOTICE_URL} every ${POLL_INTERVAL} second(s)"

# To whom it may concern: http://superuser.com/questions/590099/can-i-make-curl-fail-with-an-exitcode-different-
while http_status=$(curl -o /dev/null -w '%{http_code}' -sL "${NOTICE_URL}"); [ "${http_status}" -ne 200 ]; do
  verbose && echo "$(date): ${http_status}"
  sleep "${POLL_INTERVAL}"
done
```

```
GRACE_PERIOD=${GRACE_PERIOD:-120}
kubectl drain "${NODE_NAME}" --force --ignore-daemonsets --delete-local-data --grace-period="${GRACE_PERIOD}"
```

```
      nodeSelector:
        lifecycle: Ec2Spot
```


aws

Auto scaling the app and cluster

- Horizontal Pod Autoscaler (HPA)
  - Autoscales the number of pods in a deployment/replica set

```
kubectl autoscale deployment hello-k8s --cpu-percent=$HPA_MIN_CPU --min=$HPA_MIN_PODS --max=$HPA_MAX_PODS
```

- Cluster Autoscaler (CA)
  - Autoscales the number of worker nodes in the cluster when:
    - Pods cannot be scheduled due to lack of resources (pending state)
    - Nodes are underutilized and important pods can be rescheduled elsewhere

aws

Taints, toleration, and affinity

```
requiredDuringSchedulingIgnoredDuringExecution:
  nodeSelectorTerms:
  - matchExpressions:
    - key: environment
      operator: In
      values:
      - dev
```

Multi-tenant cluster, group affinity

```
affinity:
  nodeAffinity:
    preferredDuringSchedulingIgnoredDuringExecution:
    - weight: 1
      preference:
        matchExpressions:
        - key: lifecycle
          operator: In
          values:
          - OnDemand
```

Lifecycle affinity and toleration

```
tolerations:
- key: "spotInstance"
  operator: "Equal"
  value: "true"
  effect: "PreferNoSchedule"
```

Creating Spot node groups

Eksctl – upon creation of cluster

- Support for Mixed node groups / Mixed ASG not introduced yet

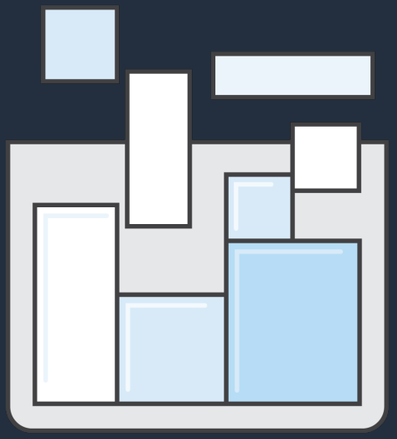
EKS – for existing cluster

- Use the CloudFormation template from eksworkshop.com/spotworkers

aws

Spot and containers – best practices overlap!

- Cluster instances should be redundant and replaceable – cattle not pets
- Treat everything as ephemeral
- Be flexible! Use multiple pools of capacity
- If something is stateful or important (like logs or data), send it somewhere else



aws

Kubernetes/EKS and Spot considerations

- Run Spot Instances as worker nodes whether you're using EKS or Kops or others
- CA (cluster-autoscaler): Use one Auto Scaling group with a diversified set of instance types
- Run a DaemonSet on every worker to catch the Spot interruption and drain the node
- Use labels to identify Spot nodes (for the DaemonSet, and other purposes – schedule non-prod?)

aws

Run the right workloads on Spot worker nodes

- Use the node labels to identify EC2 Spot instances

```
--node-labels  lifecycle=Ec2Spot
```

- Configure Taints and Tolerations to influence scheduling

```
--register-with-taints=spotInstance=true:PreferNoSchedule’
tolerations:
        - key: "spotInstance”
        operator: "Equal”
        value: "true"
        effect: "PreferNoSchedule"
```

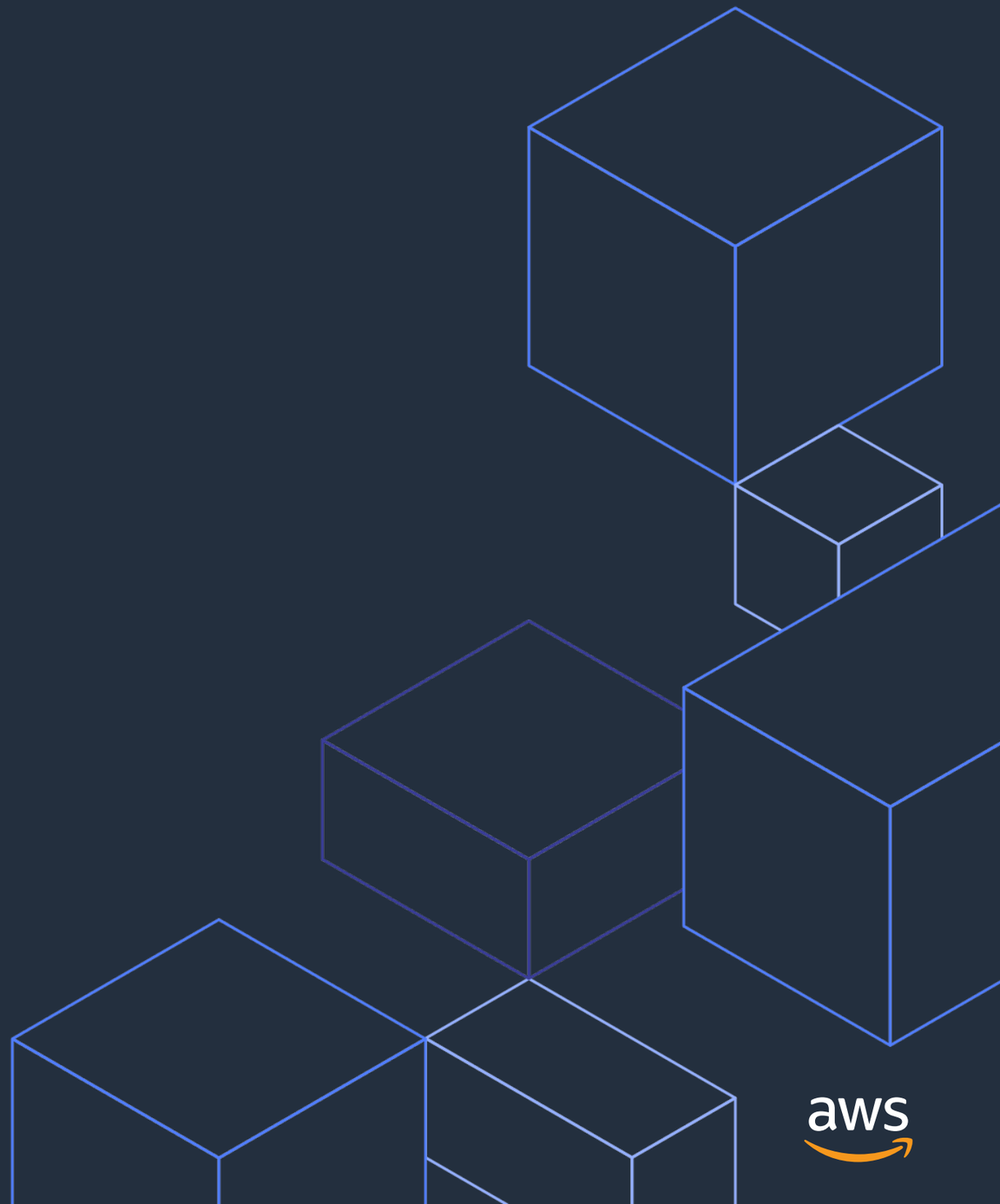

aws

Summary and takeaways

- Self paced workshops for Spot Instances
  http://bit.ly/EC2SpotWorkshops
- Spot Instances module on the EKS workshop website
  http://bit.ly/EKSWorkshopSpot
- The definitive guide to running EC2 Spot Instances as Kubernetes worker nodes
  http://bit.ly/DefintiveSpotK8sGuide

aws

App Mesh

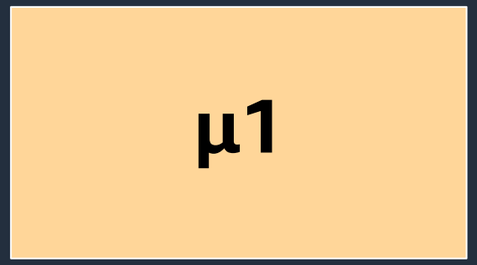


What is a service mesh?

- An infrastructure layer for service-to-service communication
- Makes communication visible, manageable, controlled

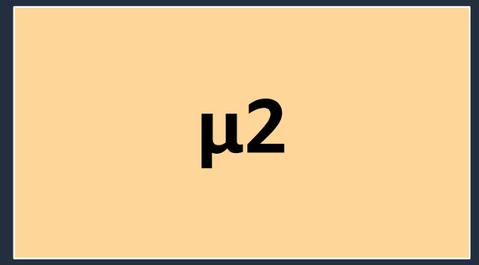
aws

Let’s say µ1 needs to communicate with µ2



Microservice

Host/container/
serverless



Microservice

Host/container/
serverless

aws

They could talk directly by name/IP

Microservice

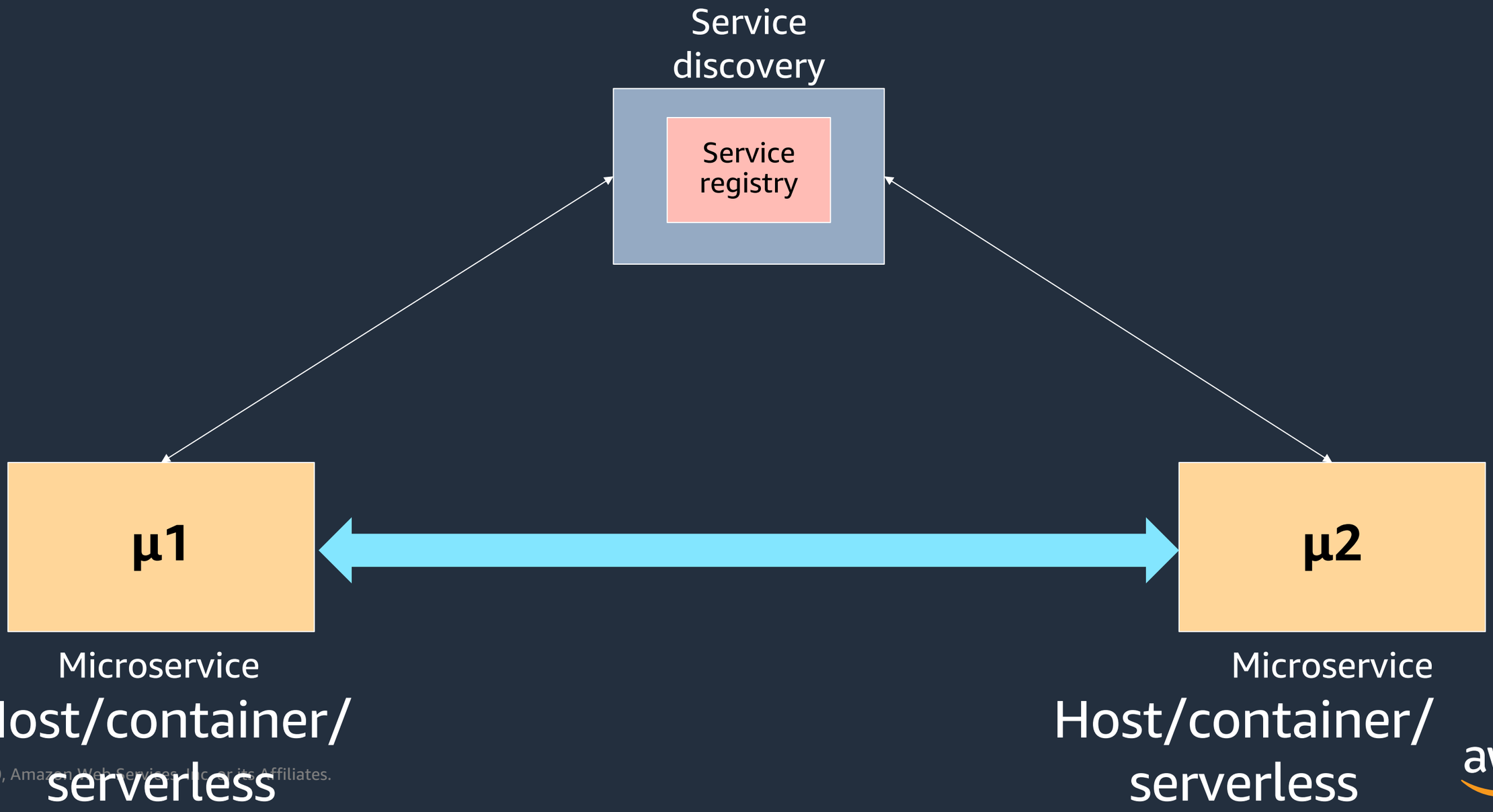
Host/container/
serverless

Microservice

Host/container/
serverless

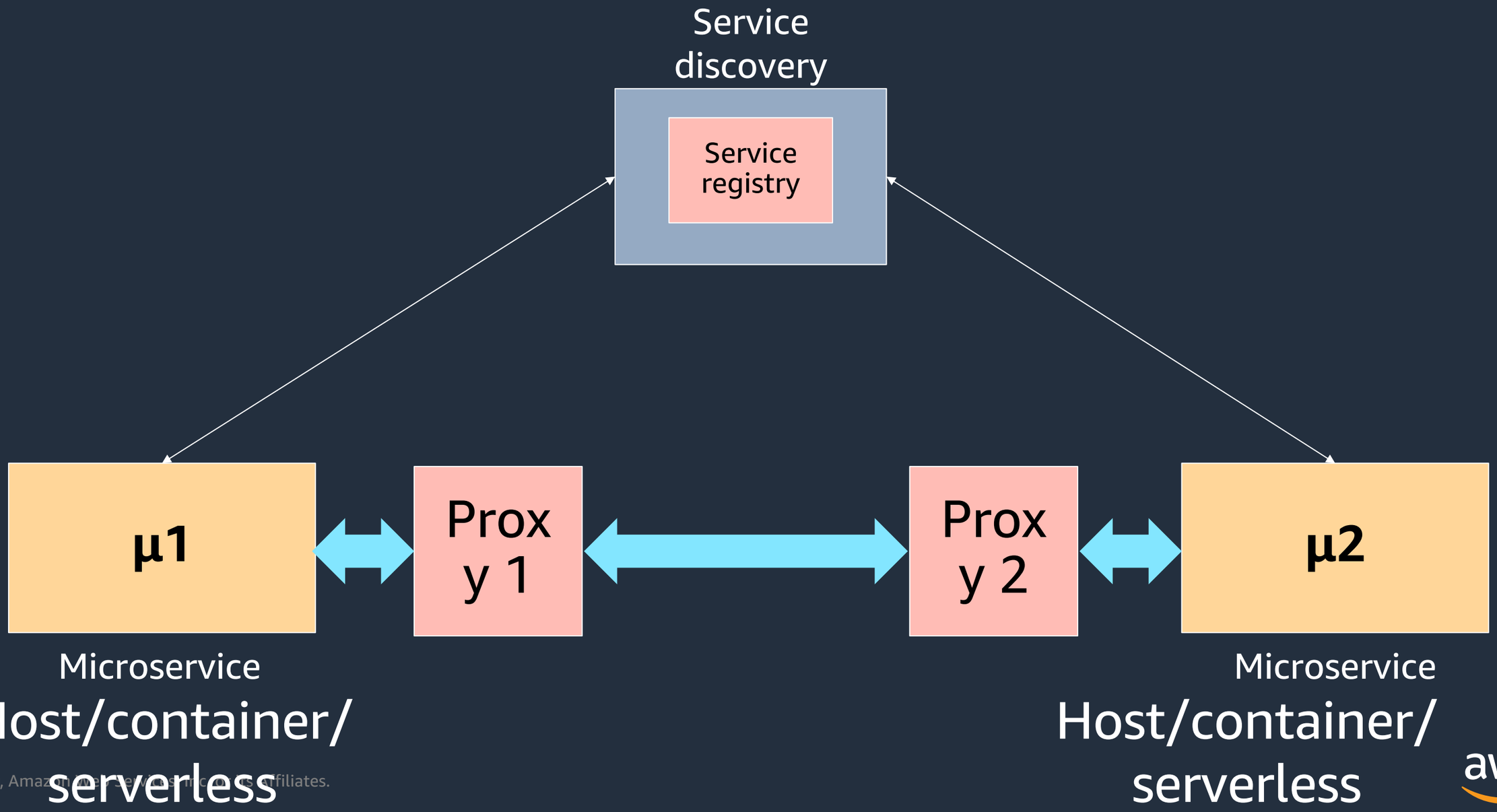
aws

Service discovery typically used to locate services

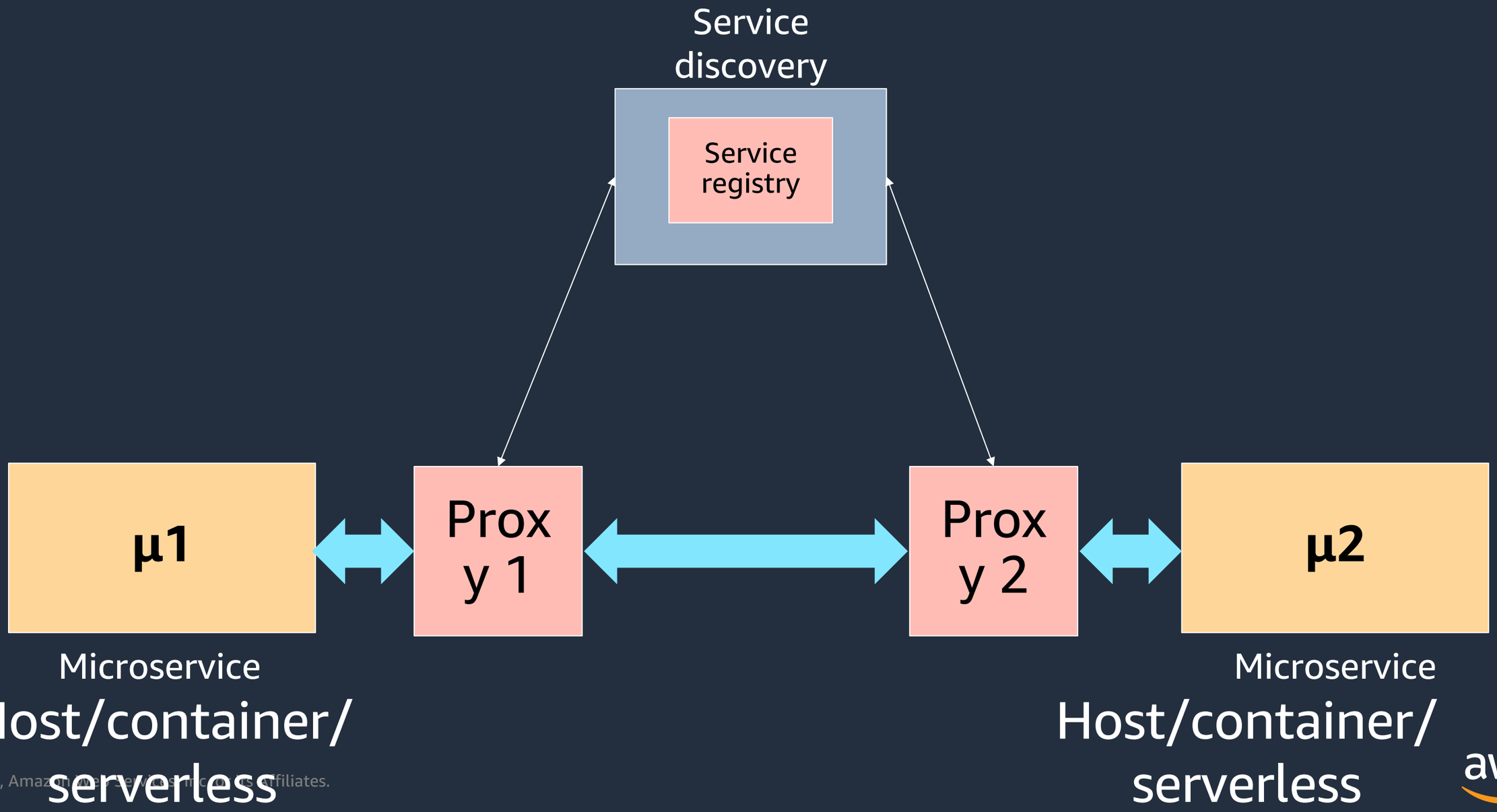


aws

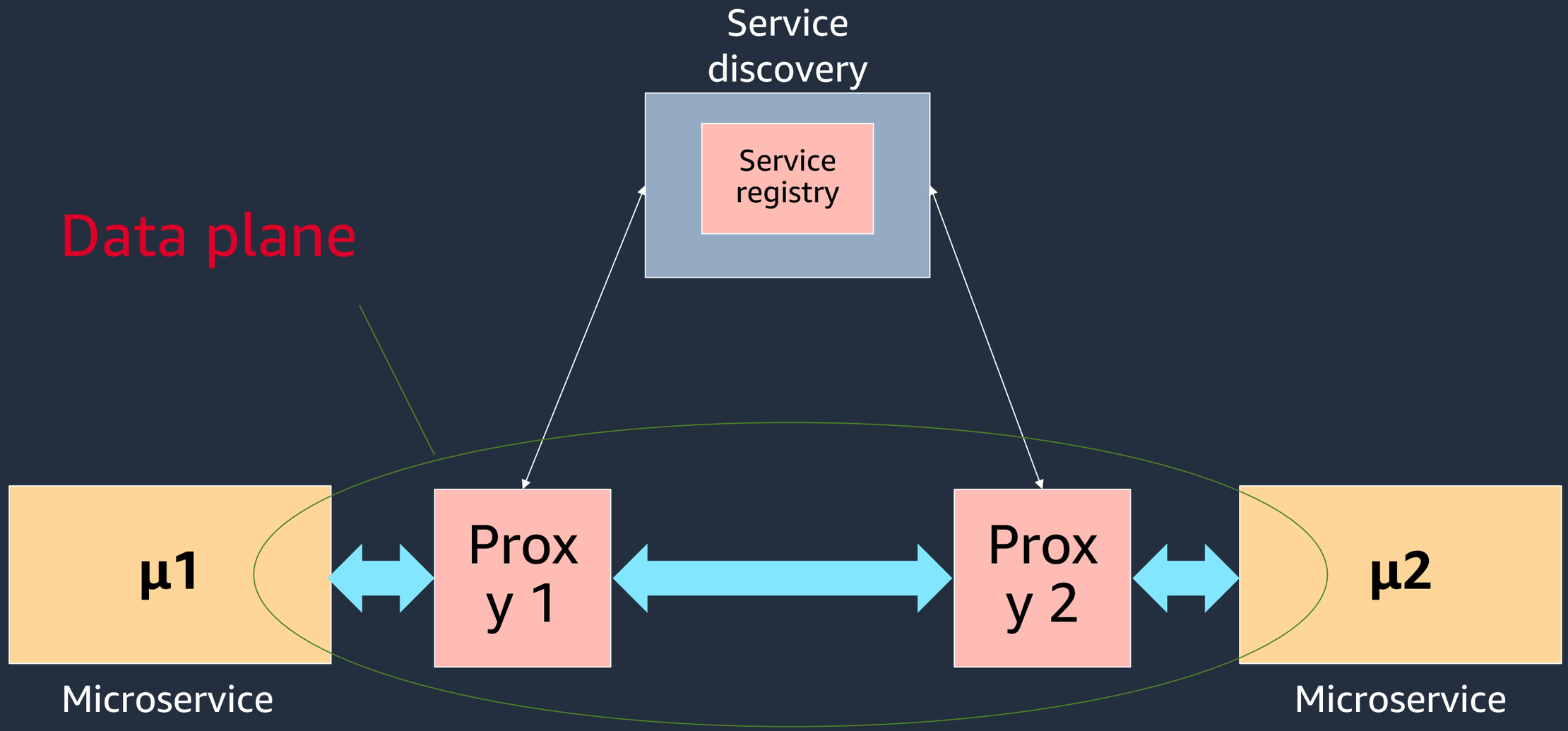
Service mesh adds an L7 proxy with each µ instance



Service discovery handled in the proxy layer

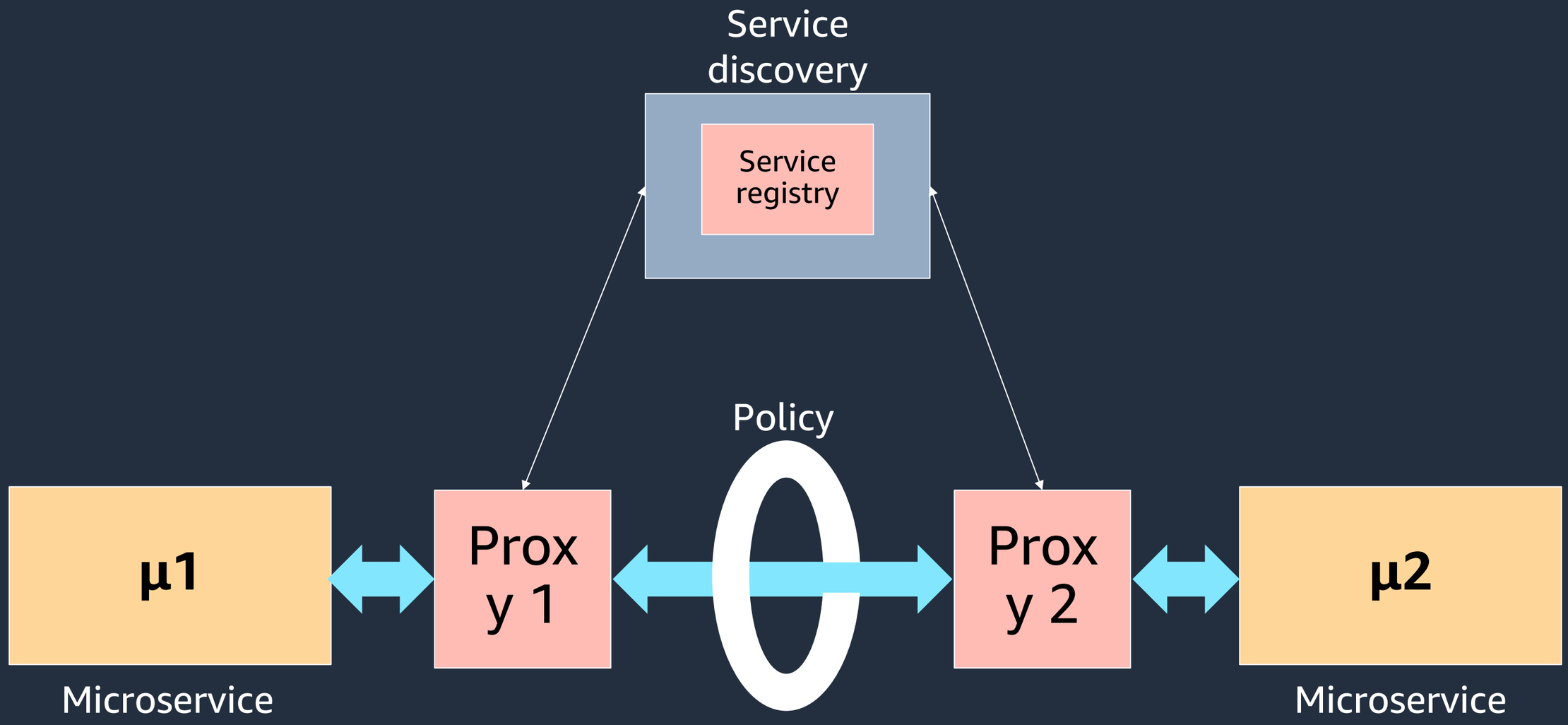


Service mesh data plane



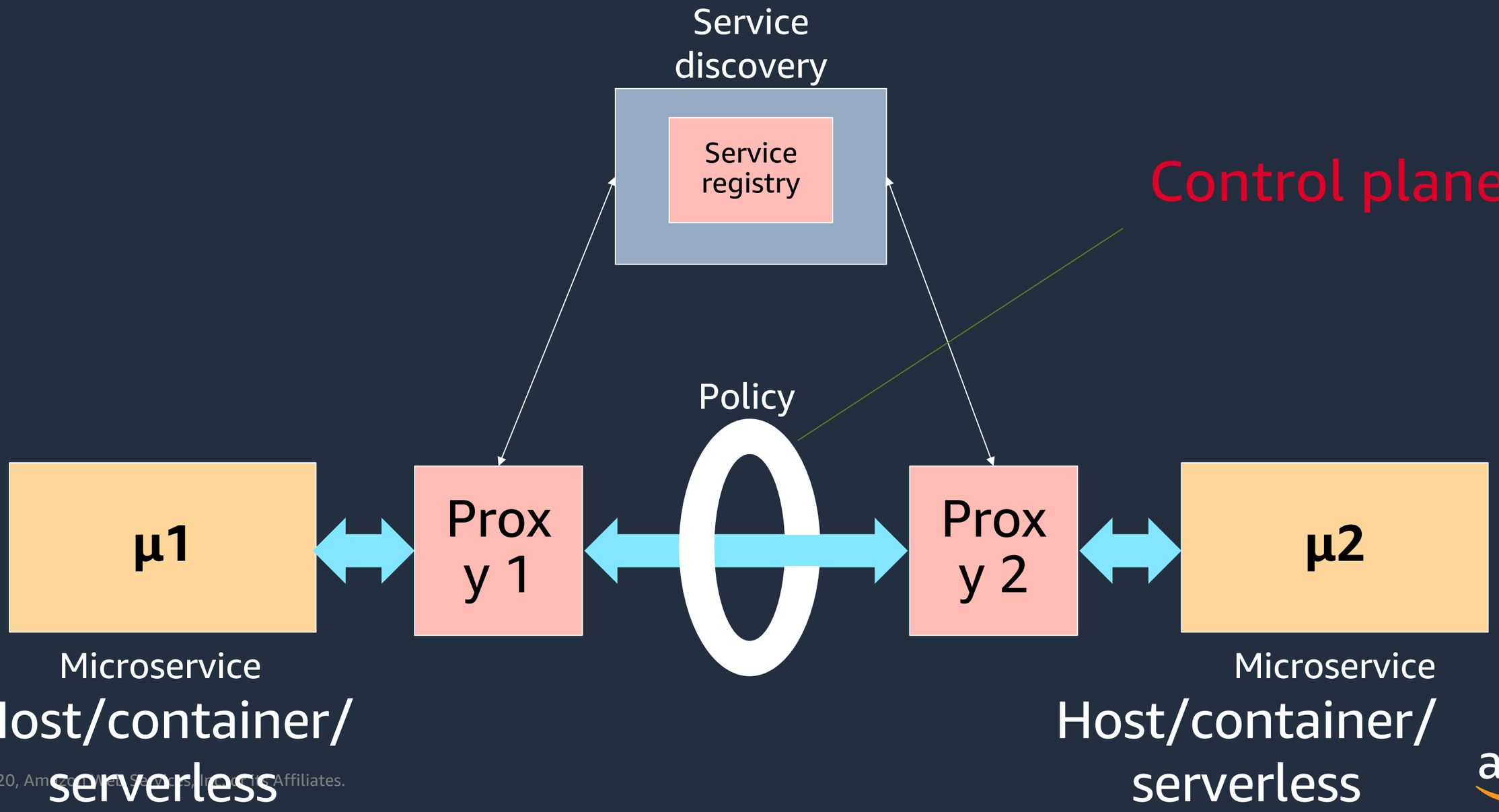
aws

Finally, put a ring on it – Policy layer to control proxies



aws

Service mesh control plane



aws

Why AWS App Mesh?

Common need: Manage interservice traffic

- How to observe (logs, metrics, traces)
- How to load balance E/W traffic
- How to shift traffic between deployments
- How to decouple service teams
- How to minimize impact to app code

aws

Why a side-car proxy?

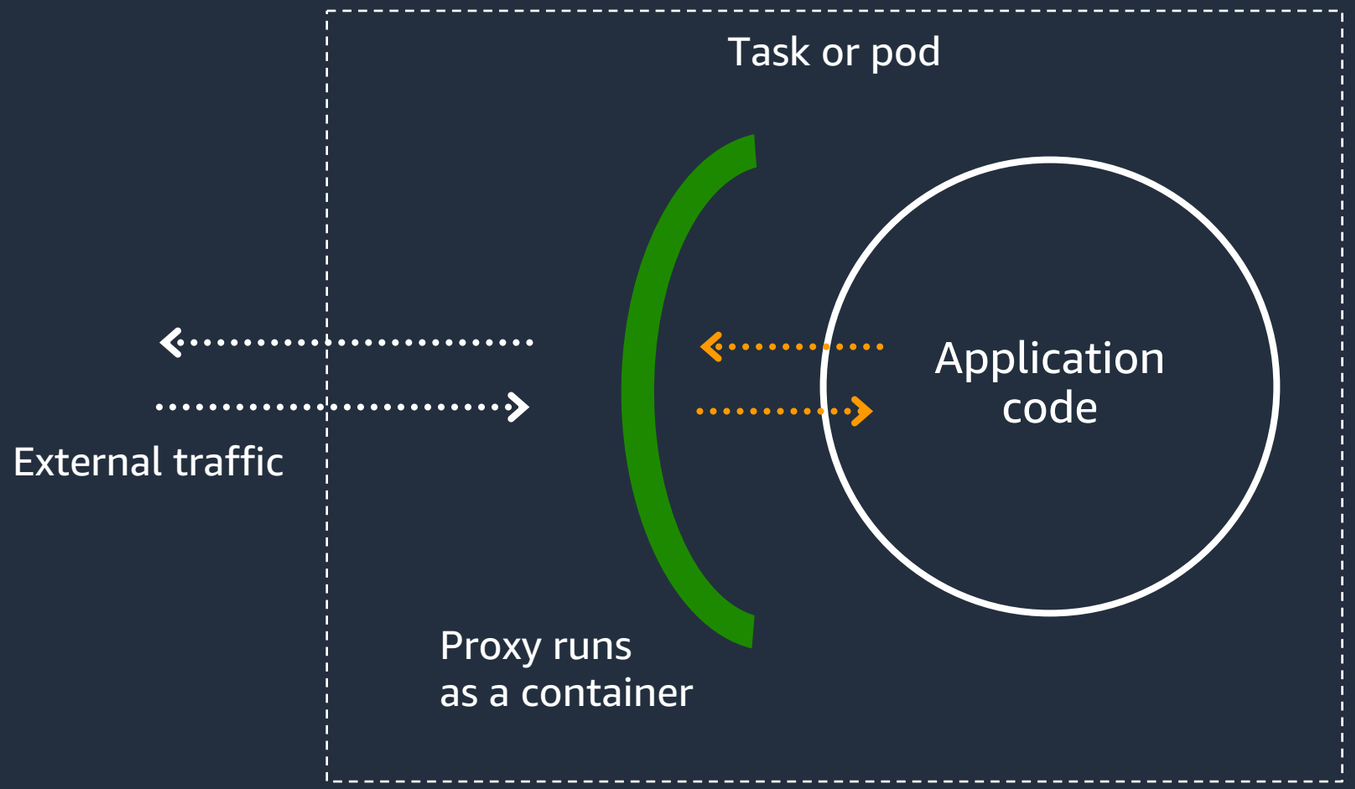
Decouples install/upgrade

Configurable: Separates business logic from operations

Minimizes inconsistencies

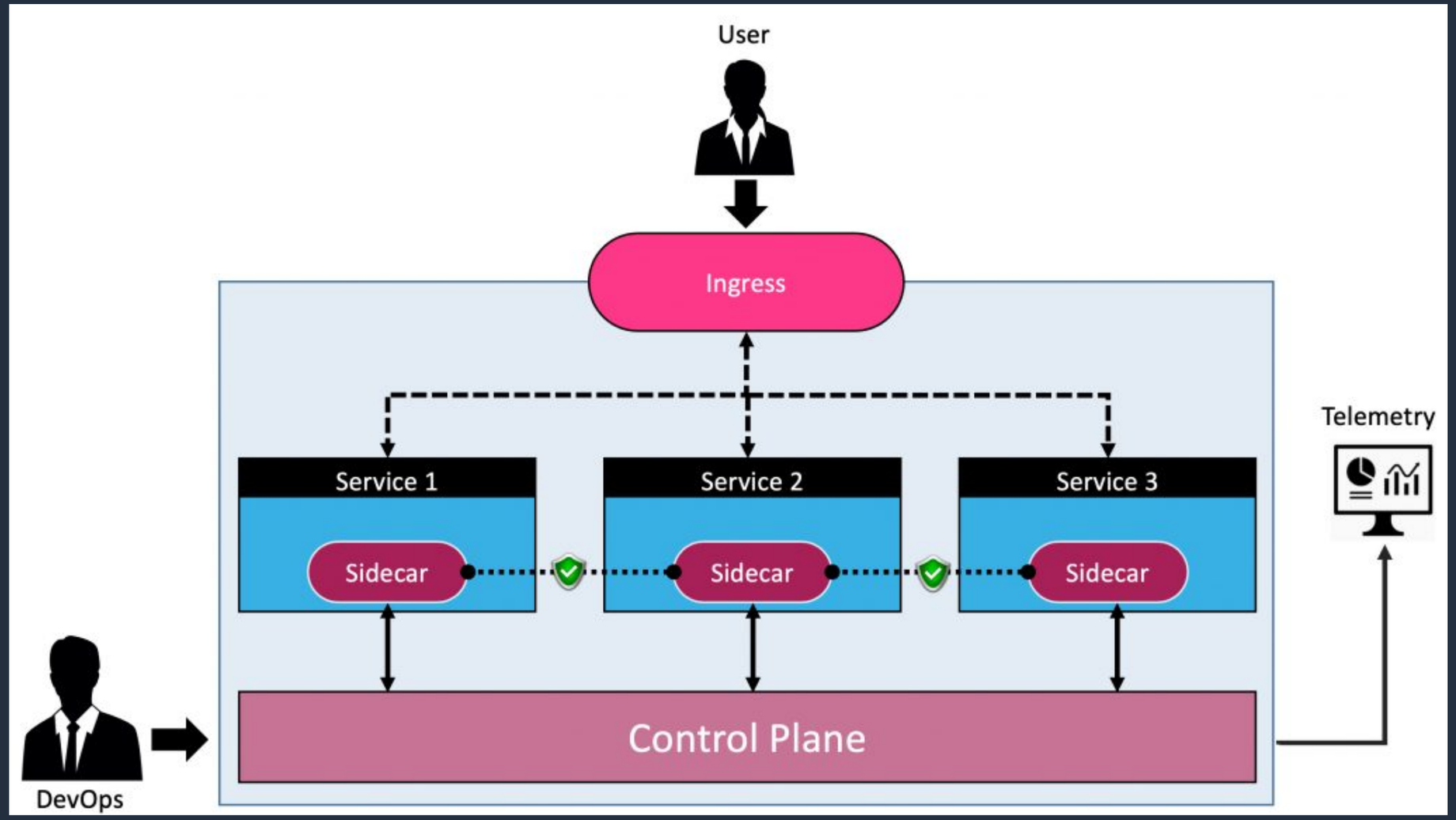
aws

Side-car proxy with containers



aws

Side-car proxy with Containers



App Mesh uses Envoy proxy

OSS community project

Wide community support, numerous integrations

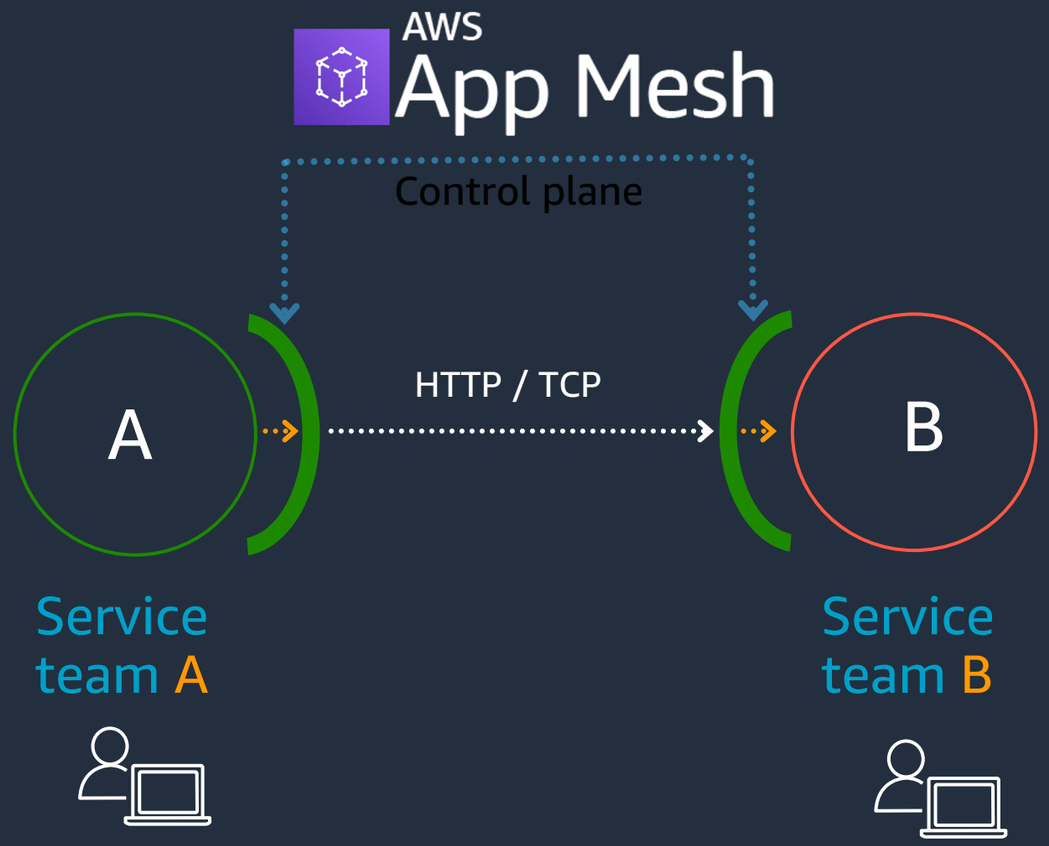
Stable and production-proven

Graduated Project in Cloud Native Computing Foundation

Started at Lyft in 2016

aws

Why App Mesh?



Control plane

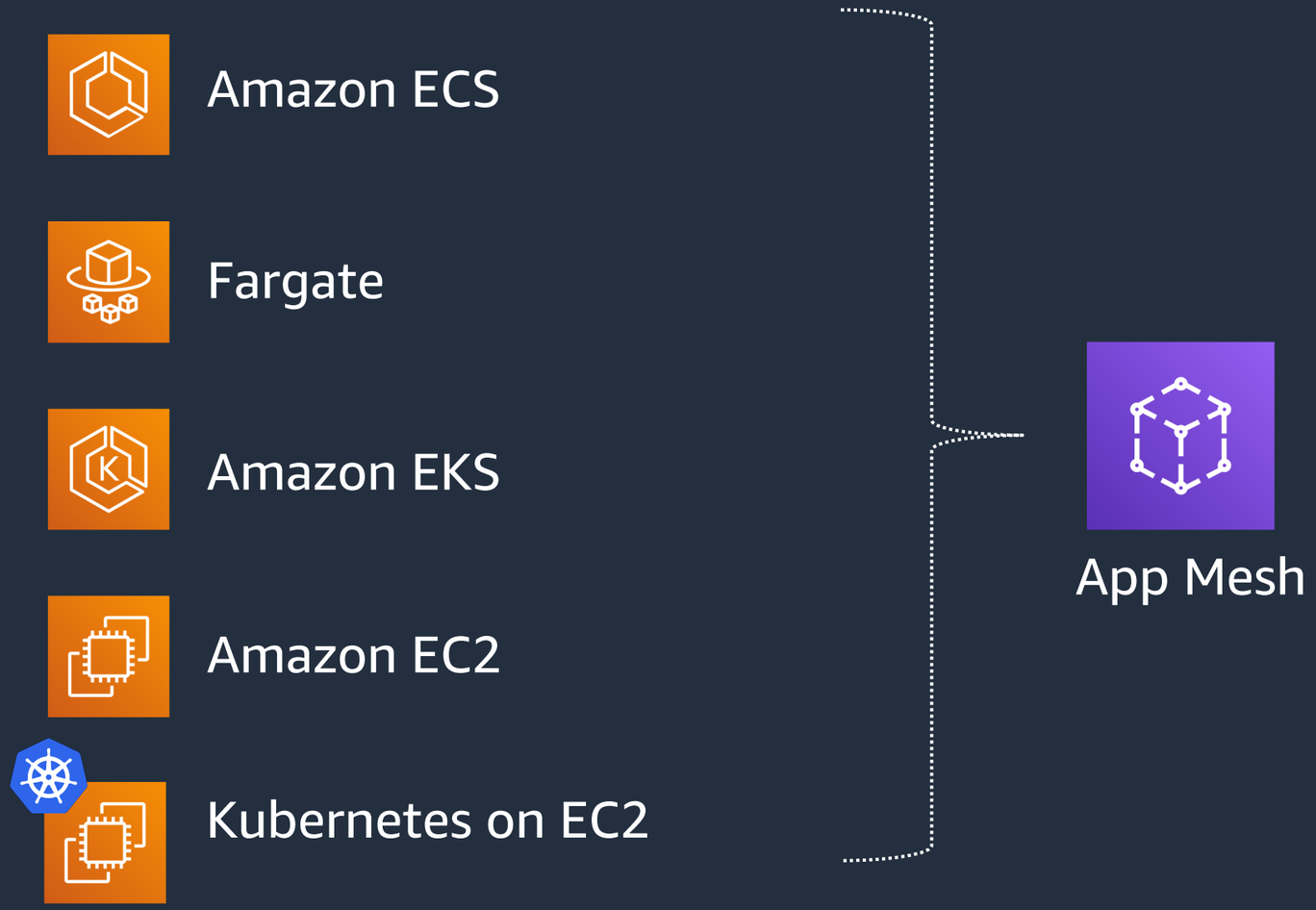
Translates logical intent to proxy config
Distributes proxy config

Proxy

Sits between all services
Manages and observes traffic

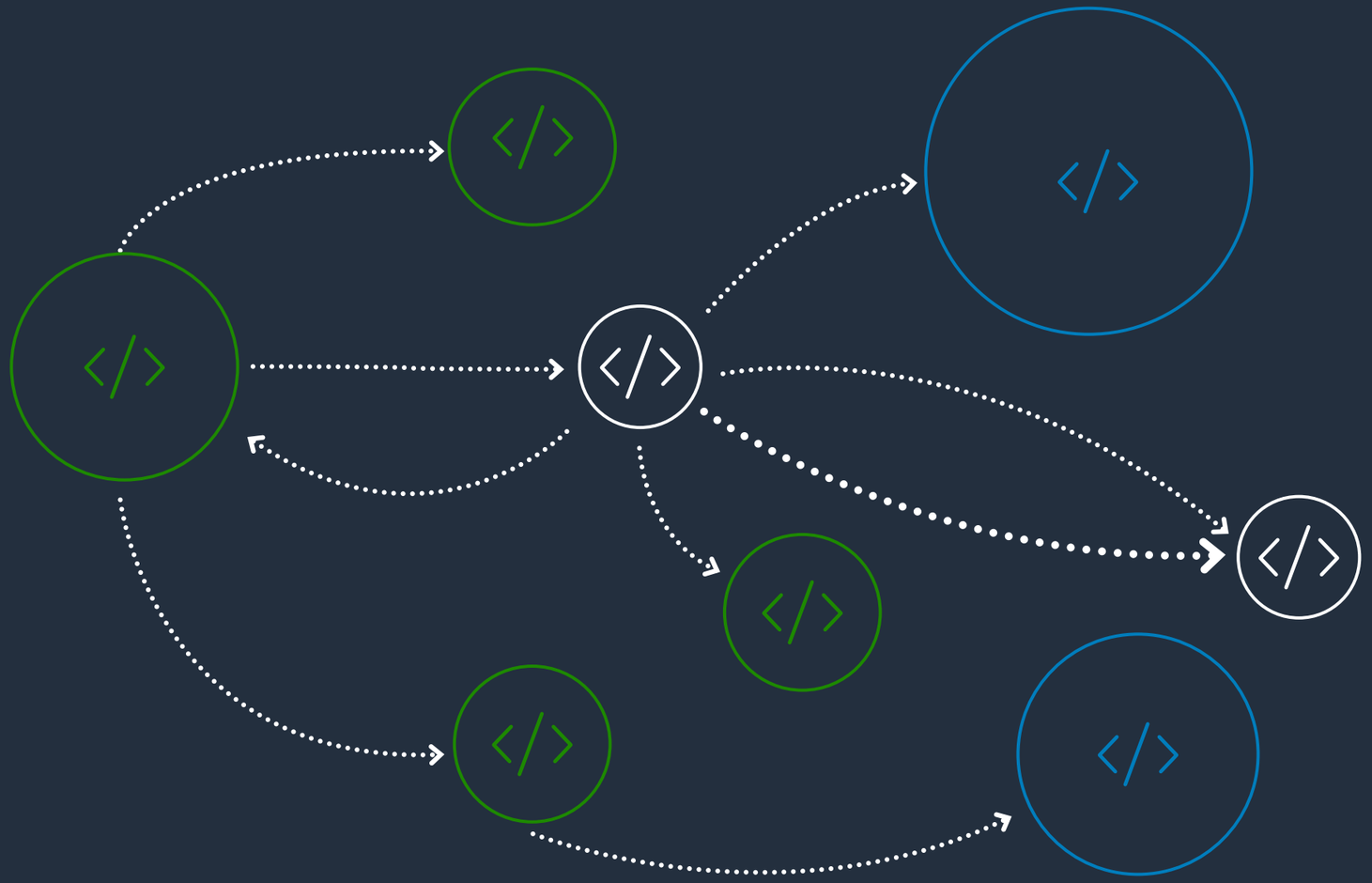
aws

App Mesh: App-level communication across AWS



aws

App Mesh: Application observability



Logging

HTTP access logging

Amazon CloudWatch Logs

Available as container logs on Amazon ECS, Amazon EKS, Fargate

Metrics

CloudWatch metrics

StatsD (with tags)

Prometheus

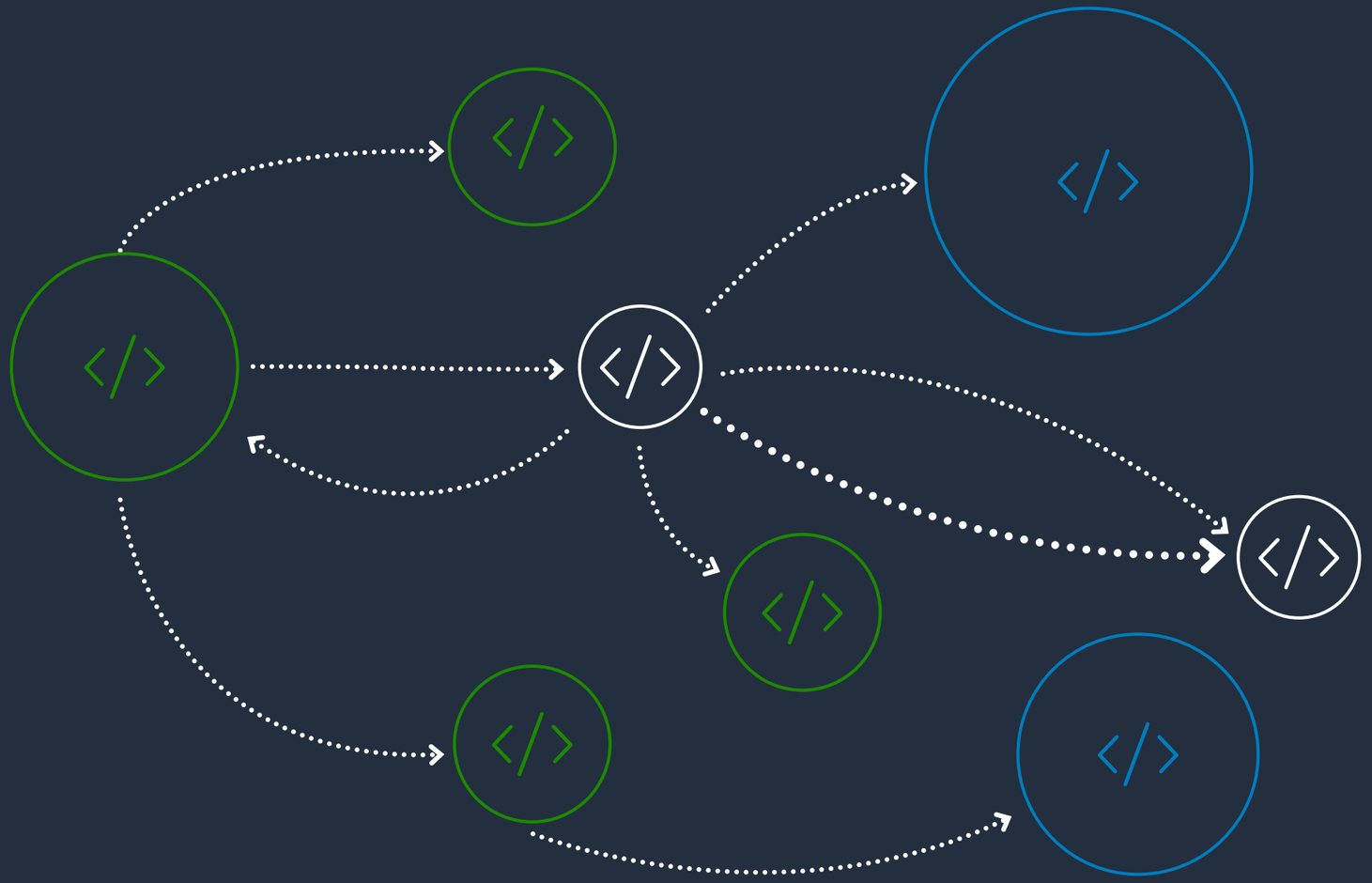
Tracing

AWS X-Ray

Other Envoy tracing drivers

aws

App Mesh: Client-side traffic management



Traffic shaping

Load balancing
Weight targets
Service discovery (DNS + AWS Cloud Map)
Health checks
*Retries**
*Timeouts**
*Circuit breakers**

Routing controls

Protocols support (HTTP, TCP, gRPC*)
Path-based
*Header-based**
*Cookie-based**
*Host-based**

*Coming soon

aws

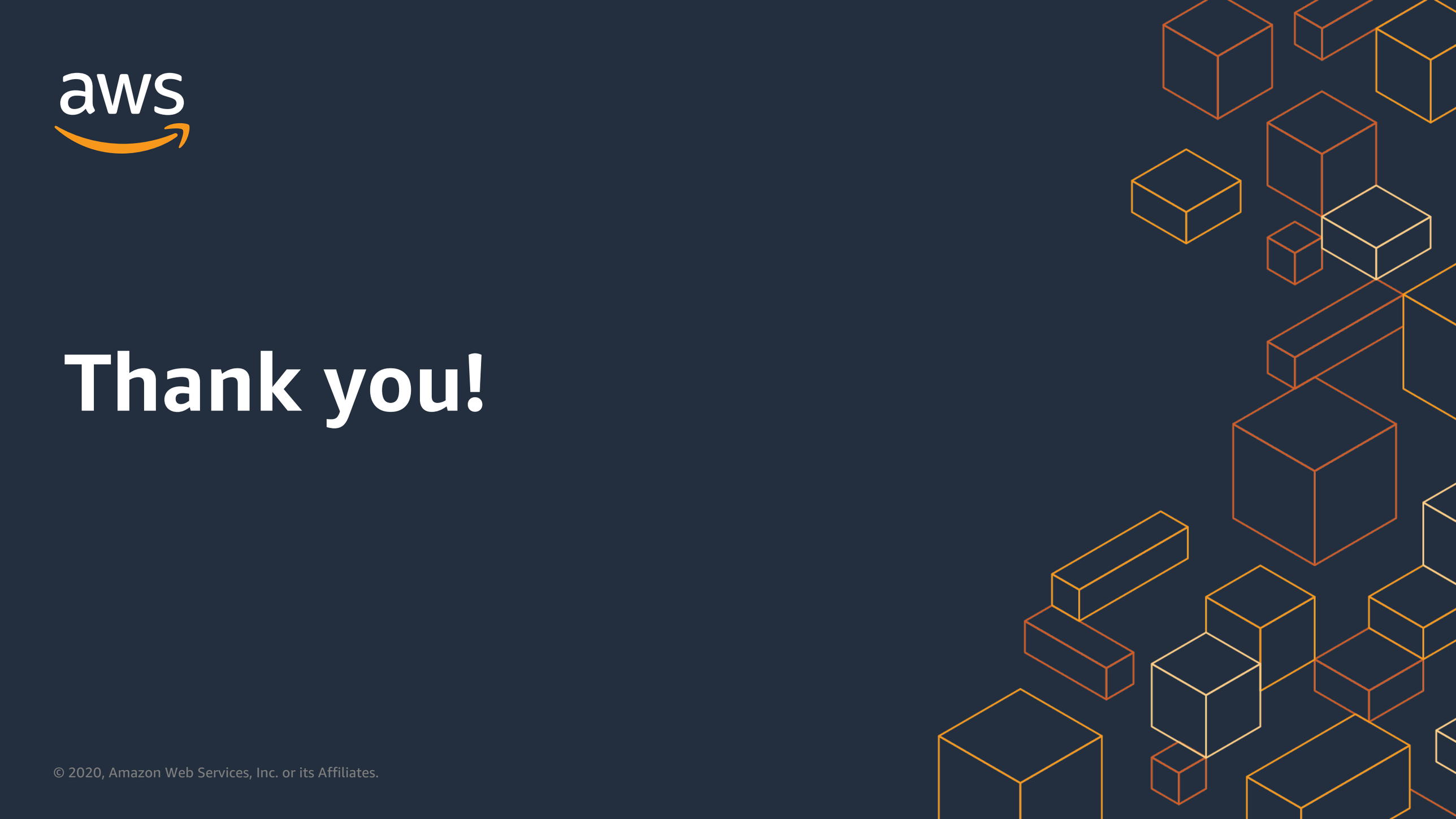
Workshops

https://ecsworkshop.com

https://eksworkshop.com

https://ec2spotworkshop.com

https://appmeshworkshop.com



Thank you!