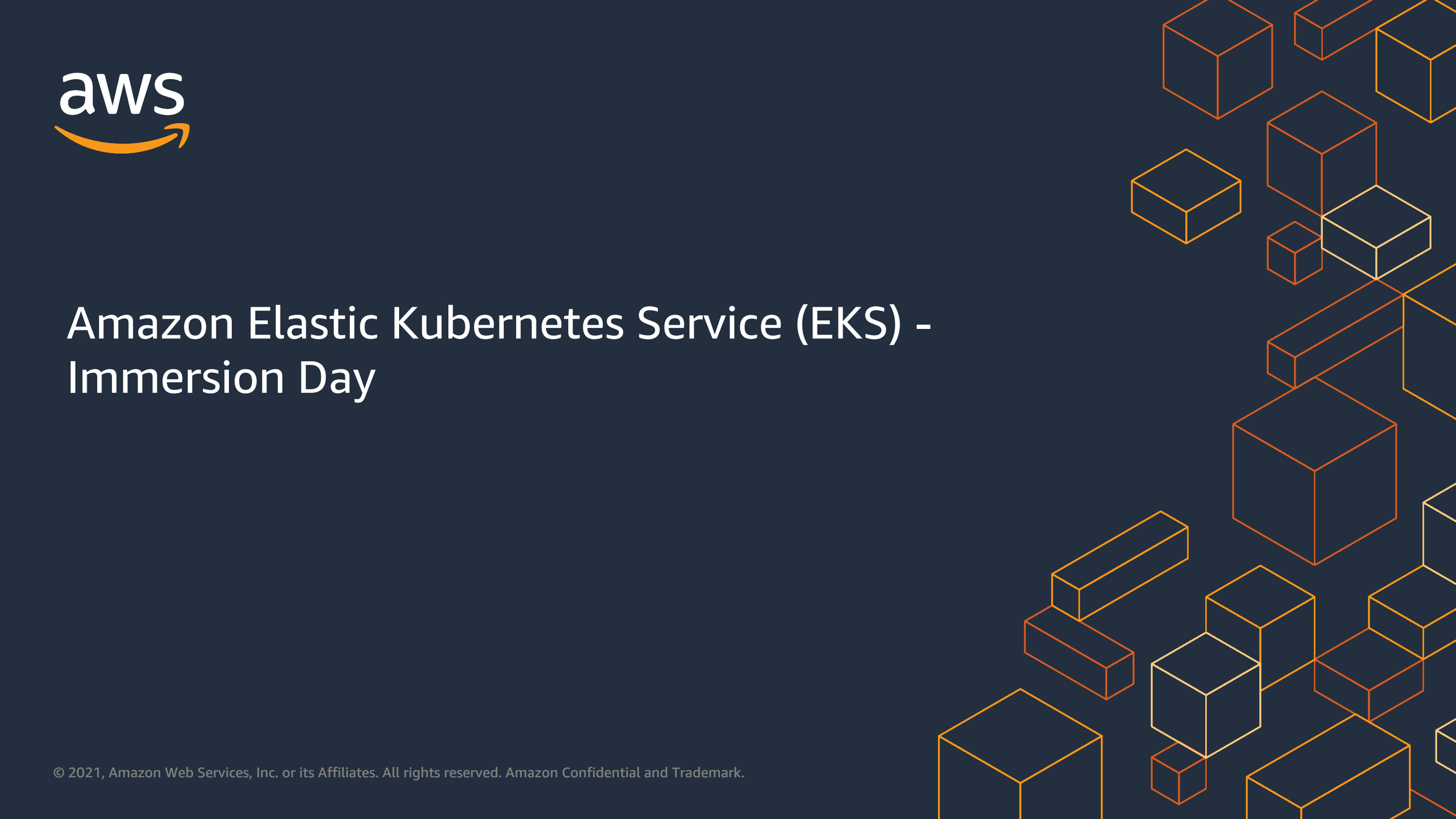


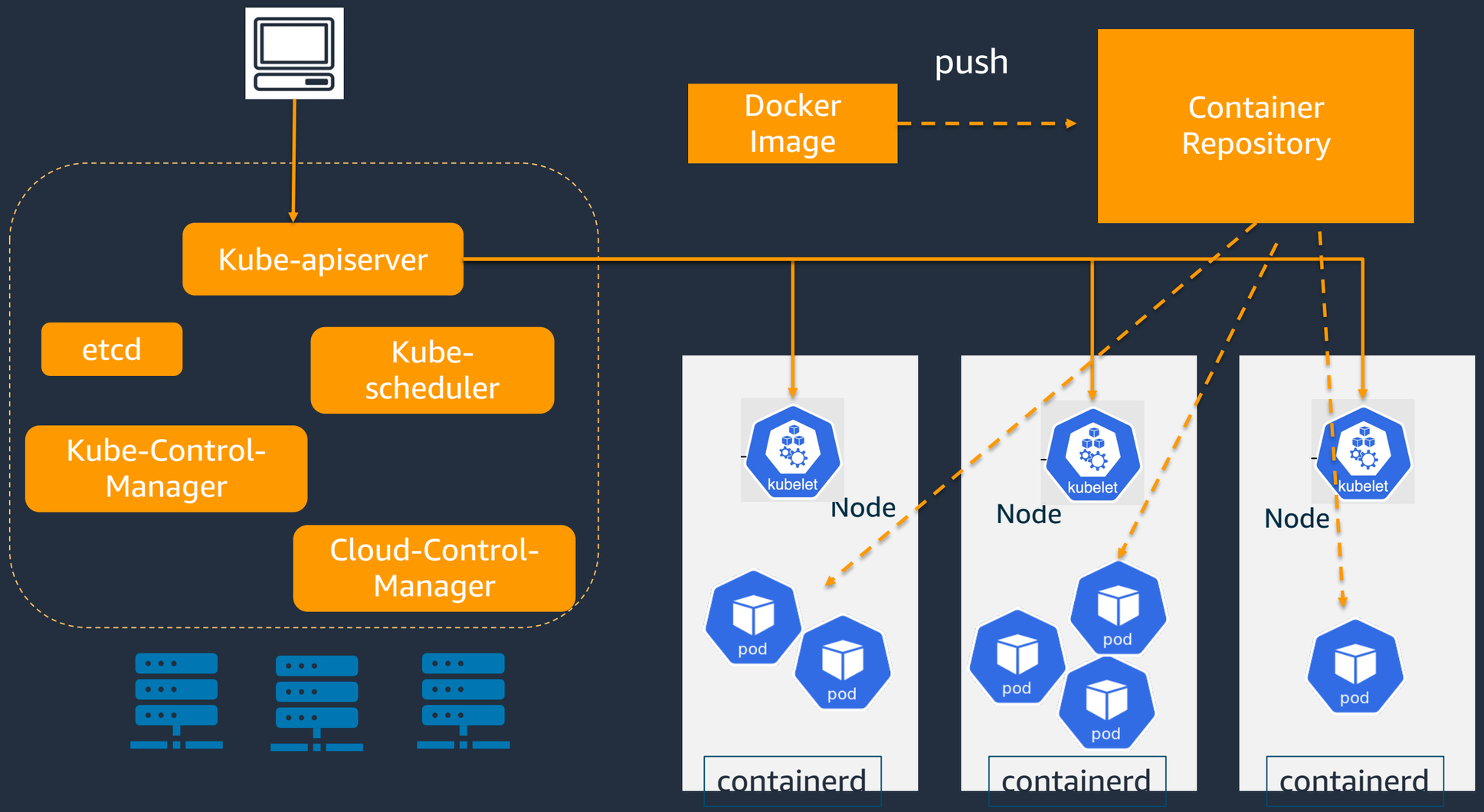
Amazon Elastic Kubernetes Service (EKS) - Immersion Day

Agenda

- Intro to Kubernetes
- Kubernetes Architecture
- EKS Architecture
- Networking
- Service Types and Ingress
- Monitoring
- Logging
- Authentication and Security
- AutoScaling in EKS
- Hybrid your cluster (Spot)
- Best Practices
- Migration between versions
- Backup Strategy
- Container Security
- A day in life of Kubernetes Developer
- EKS Managed Worker Nodes
- EKS with Fargate
- Savings Plan
- Fargate Spot
- Public Roadmap
- Q/A

Kubernetes Overview

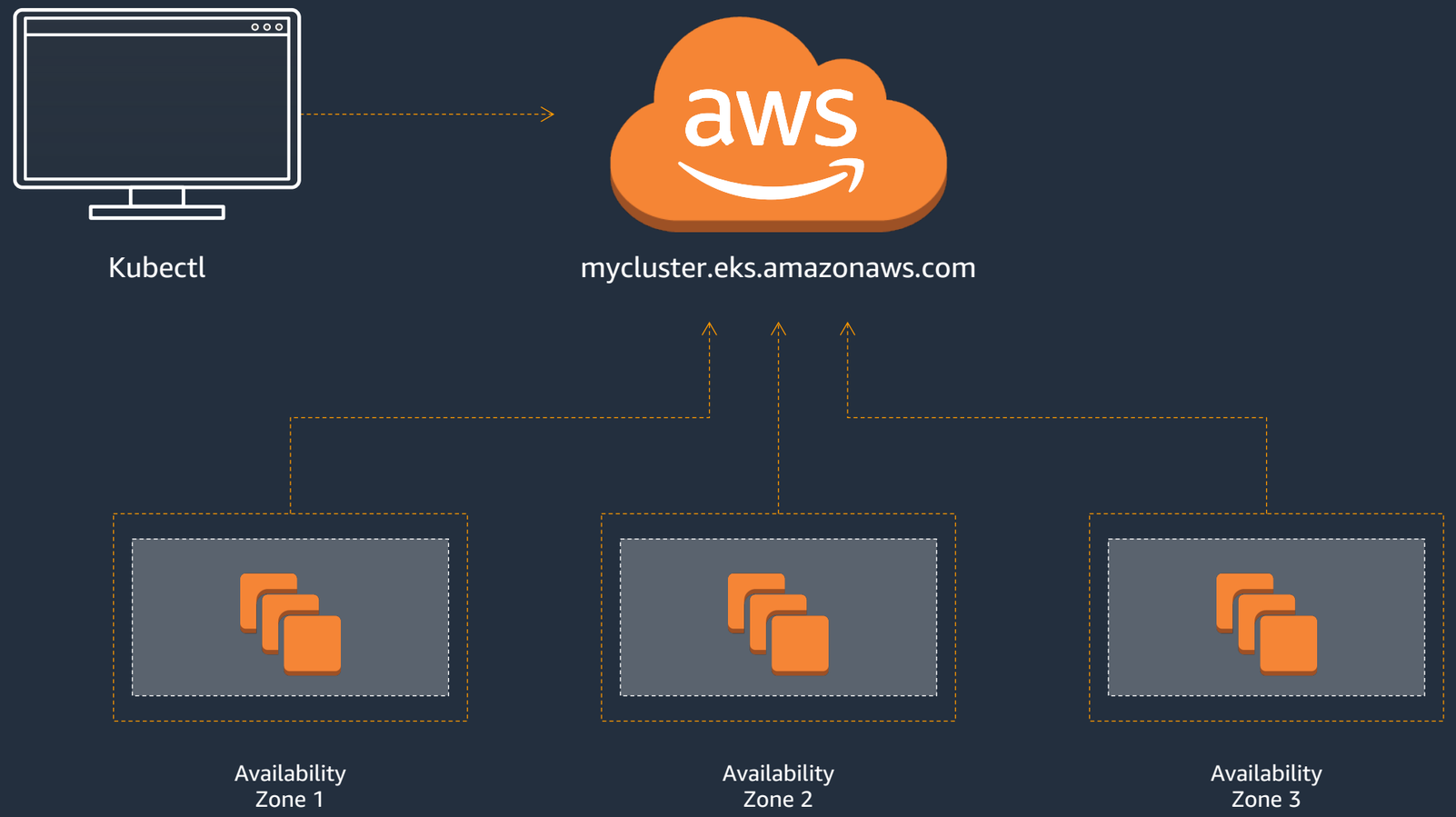
Kubernetes Architecture



- ✓ Automatic bin packing
- ✓ Self-healing
- ✓ High Availability
- ✓ Horizontal Scaling
- ✓ Rollout & Rollback
- ✓ Service Discovery
- ✓ Storage orchestration
- ✓ Batch execution
- ✓ Secret and configuration management

Elastic Kubernetes Service (EKS) Overview

Elastic Container Service for Kubernetes



Amazon EKS enables you to take advantage of other AWS services directly from Kubernetes

Amazon EKS is available anywhere you need to deploy your application

Amazon **EKS** in AWS


Web Applications


Mobile Applications


Gaming


Data Processing

Amazon **EKS** + AWS **Outposts**


Manufacturing


Retail


Gaming


Low Latency


Telecom

Amazon **EKS** + AWS **Local Zones**


Low Latency


Video Rendering


Telecom


Gaming

Amazon **EKS Anywhere**


Data Center


Manufacturing


Telecom


Gaming

Amazon **EKS Distro**


IoT

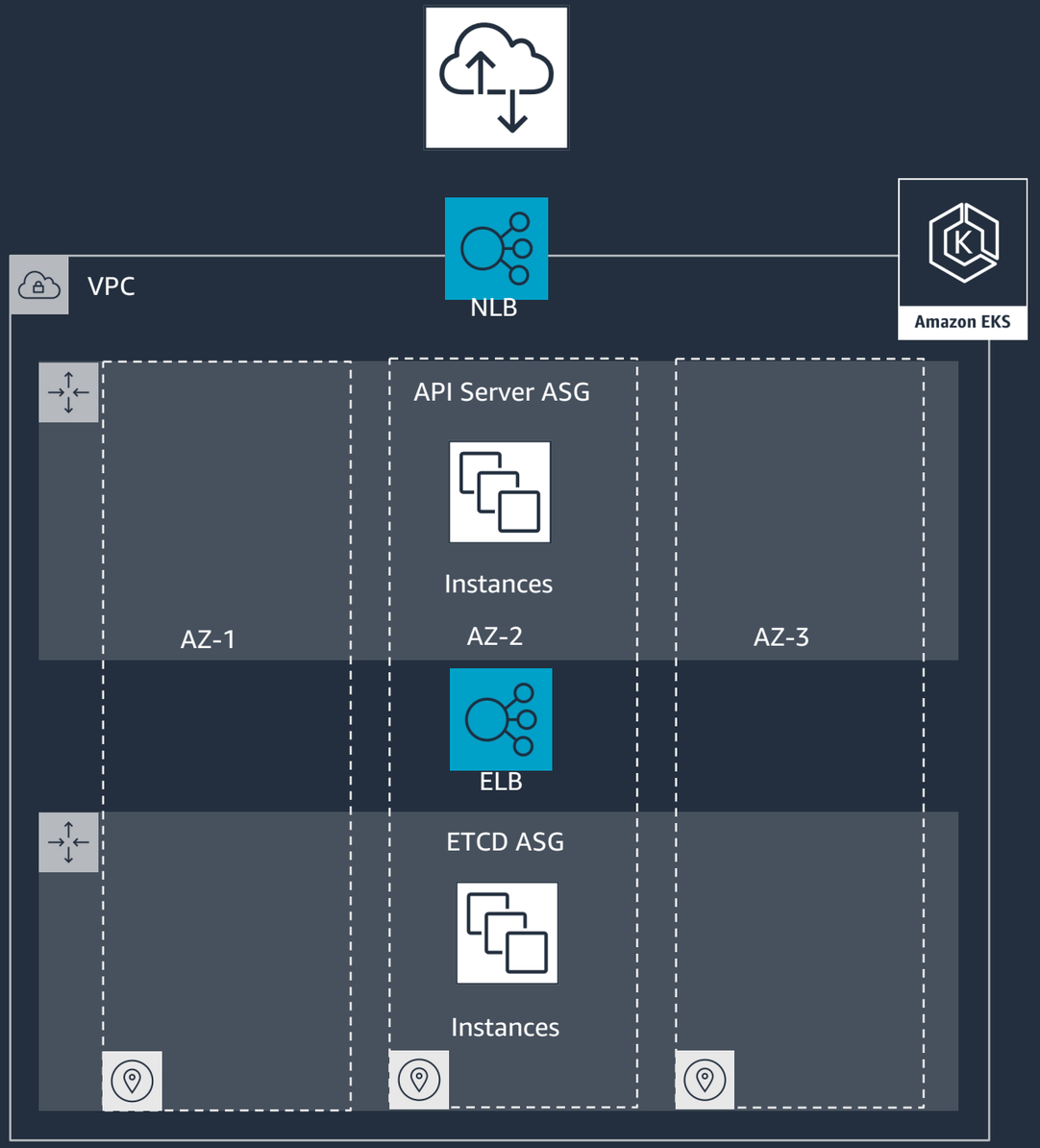

Self-Hosted K8s

Kubernetes Control Plane

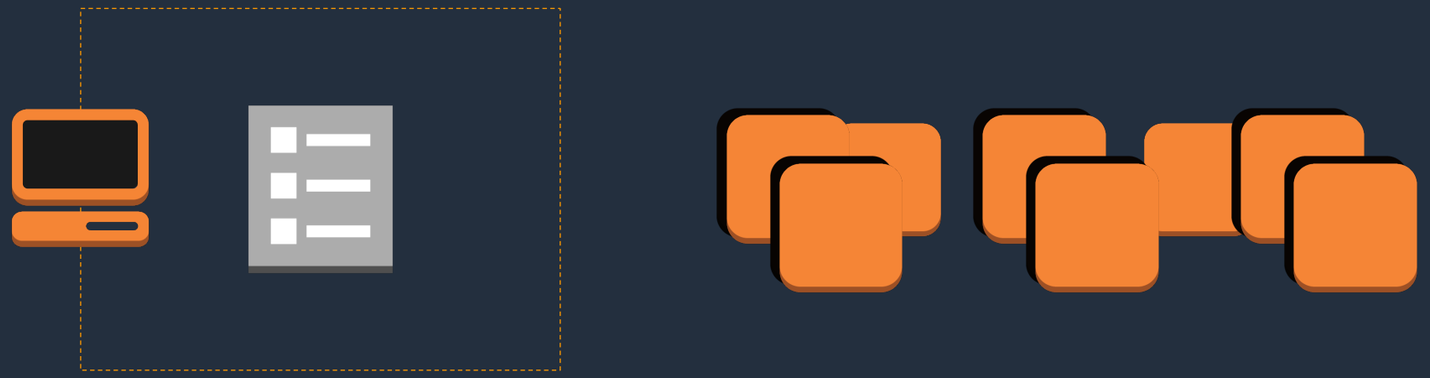
- Highly available and single tenant infrastructure

- All “native AWS” components

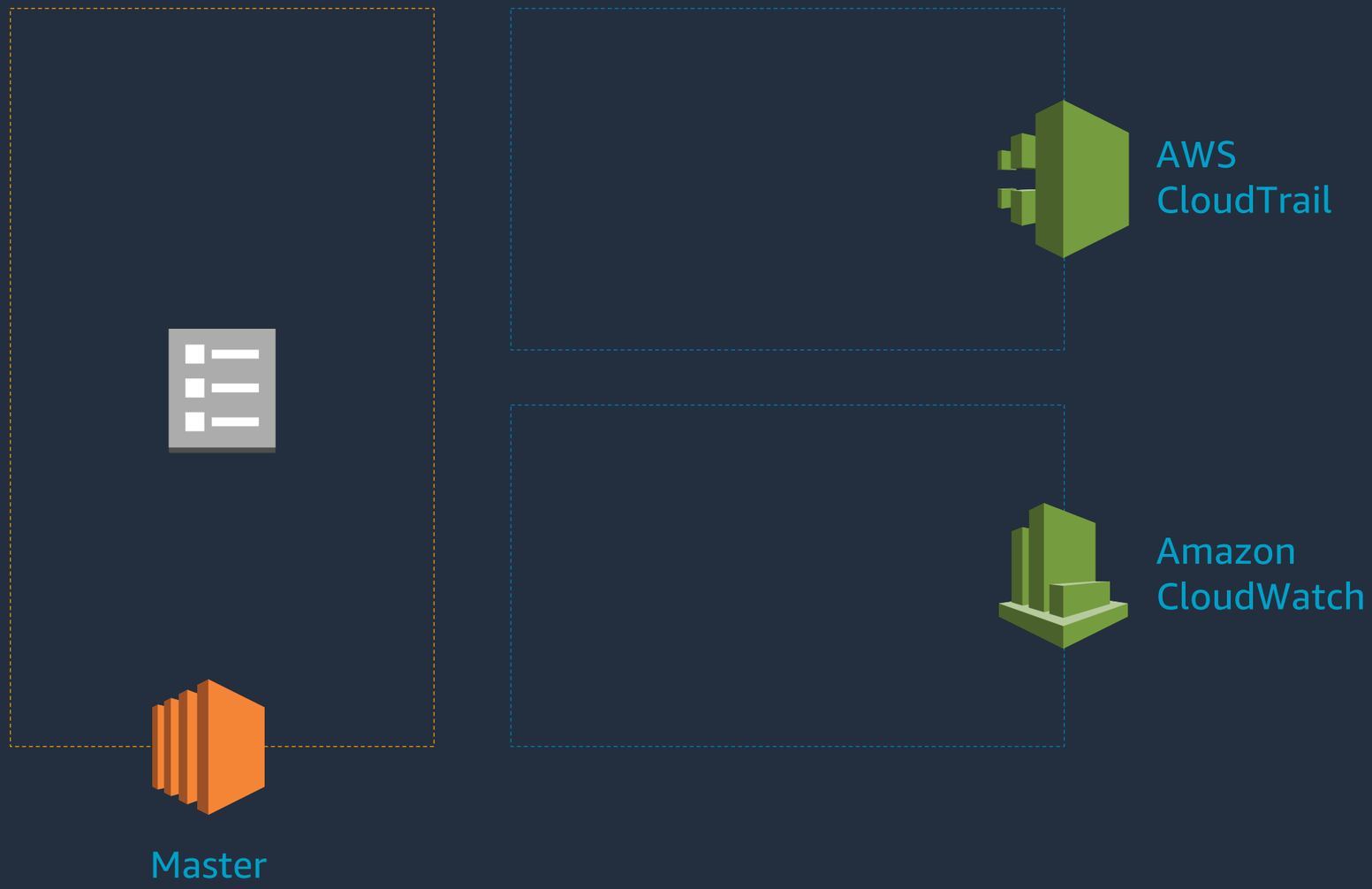
- Fronted by an NLB



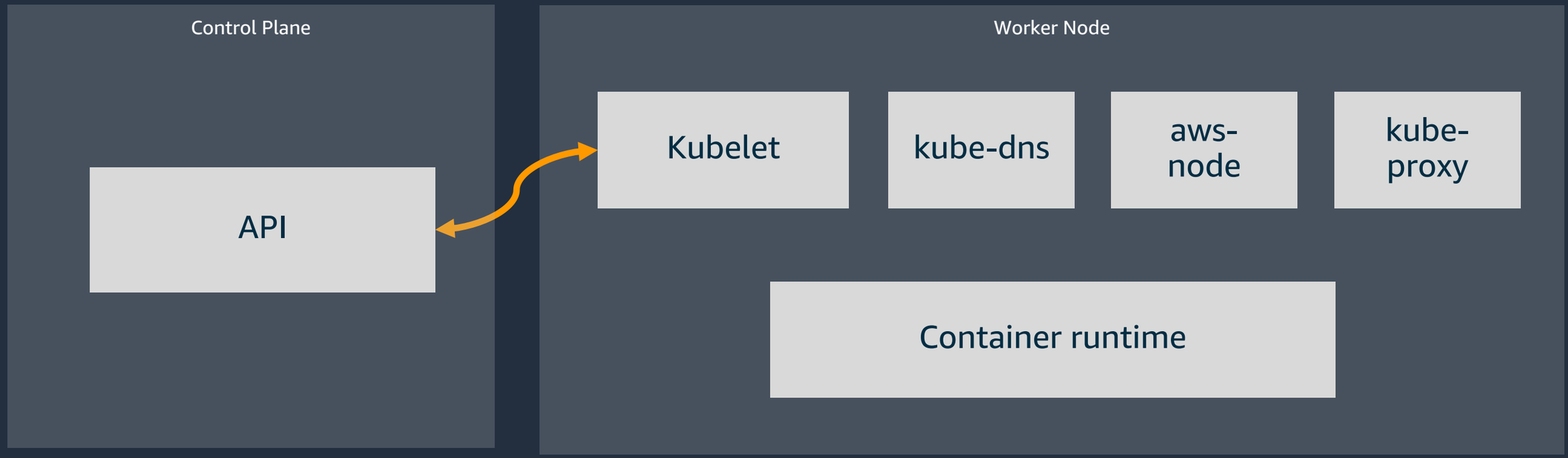
EKS Master Scaling (up)



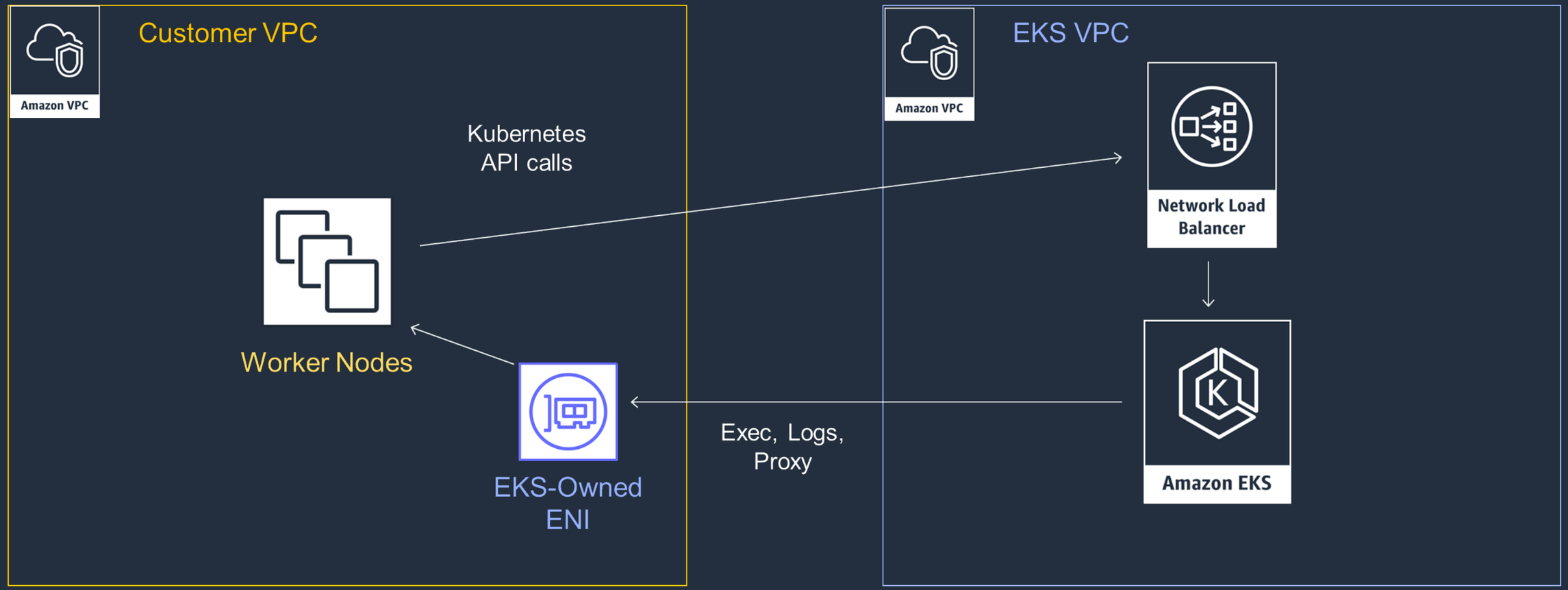
EKS Master Monitor



Kubernetes Data Plane



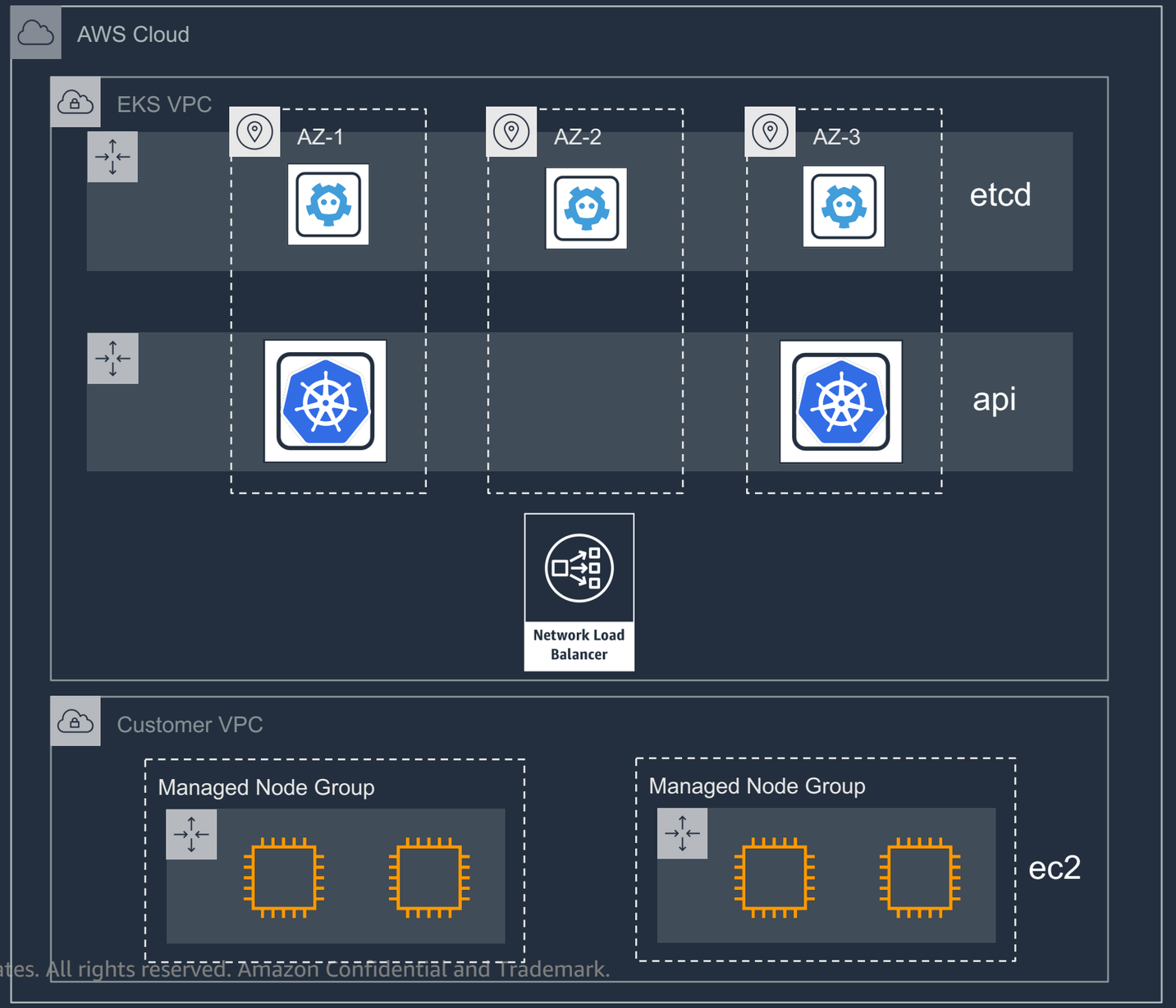
EKS Architecture



EKS Cluster Architecture – Managed Node Group

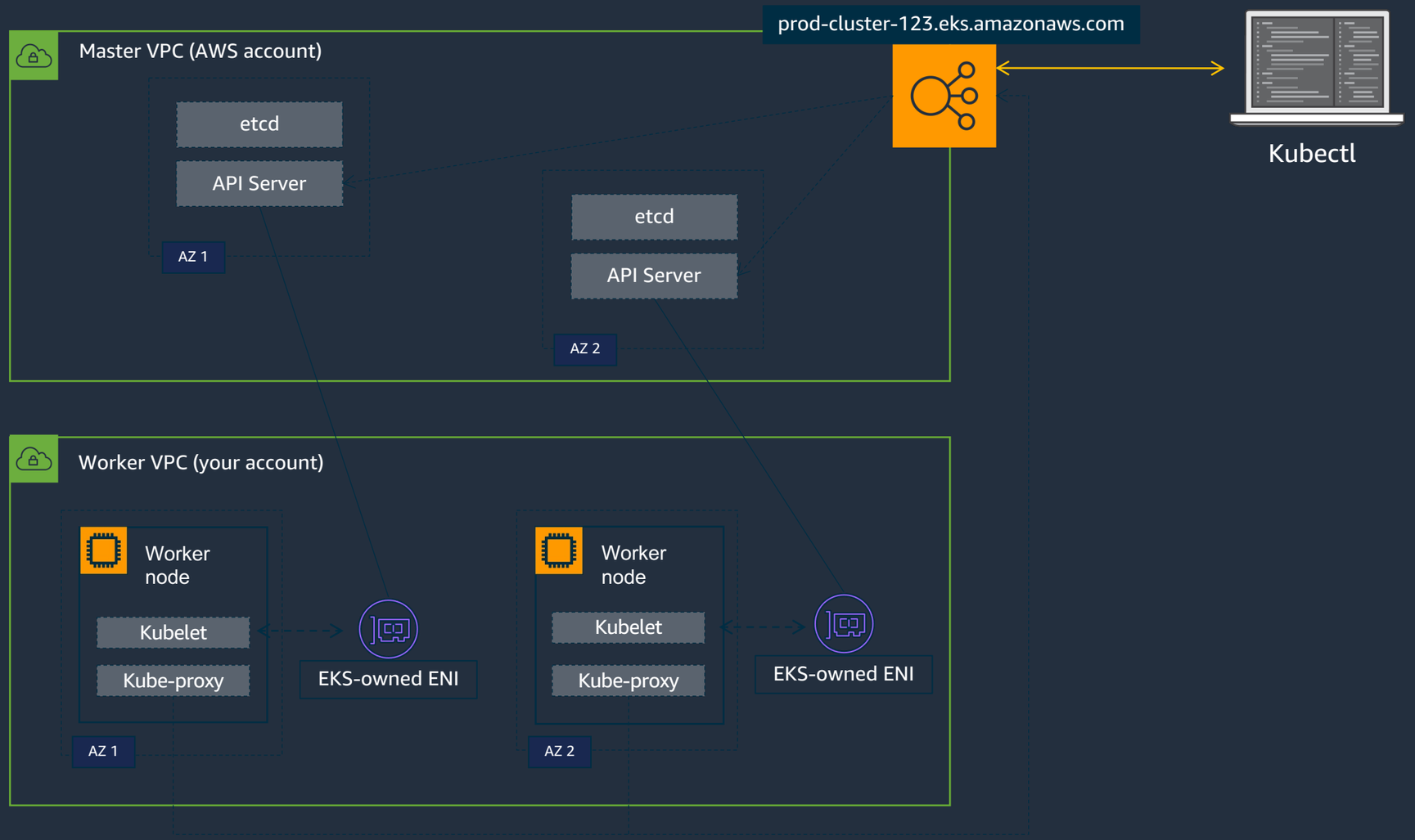
EKS Managed Control Plane

EKS Managed Data Plane



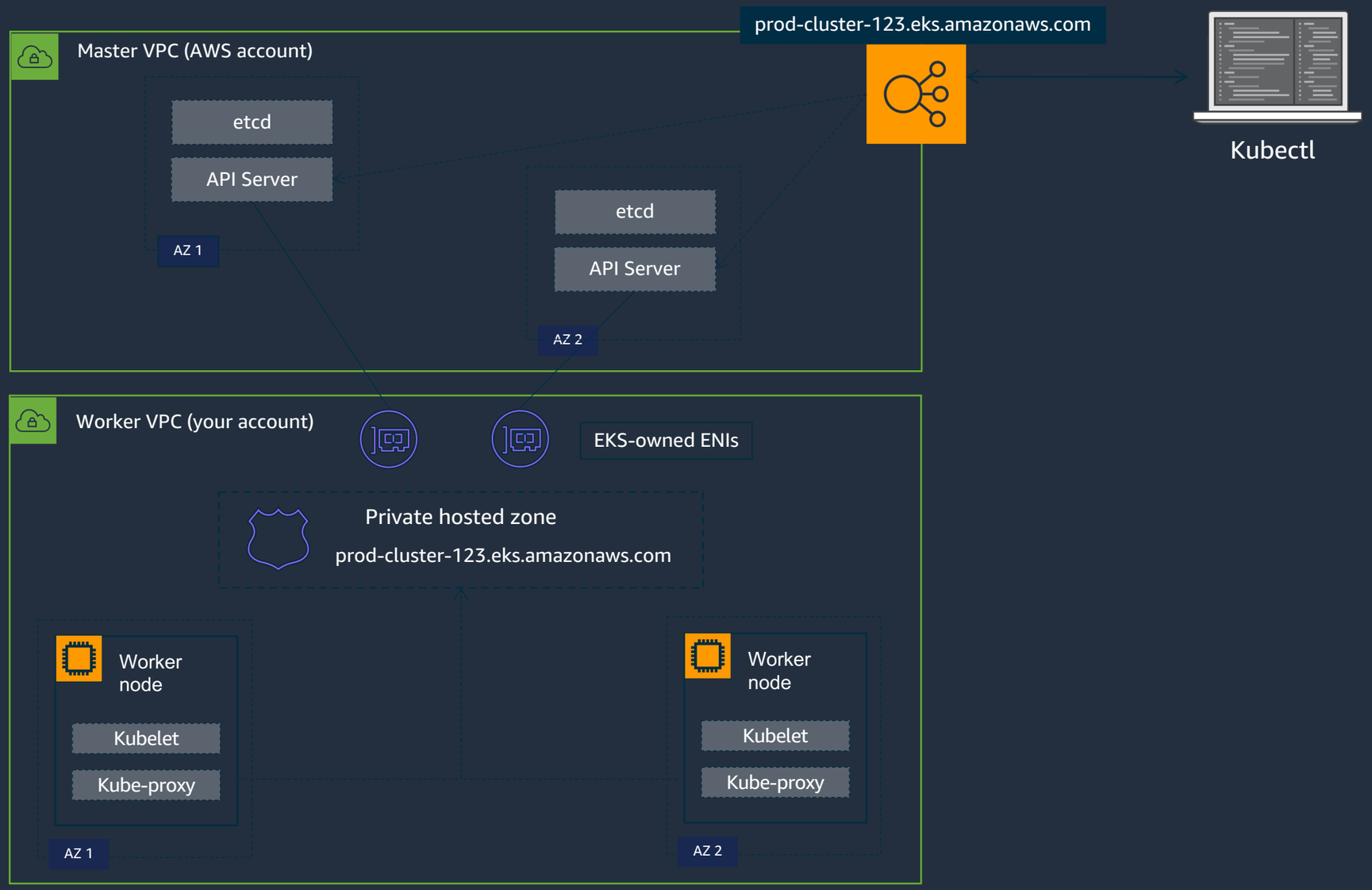
Cloud Native Security: API-server Endpoint Access Control

```
public == true

private == false
```



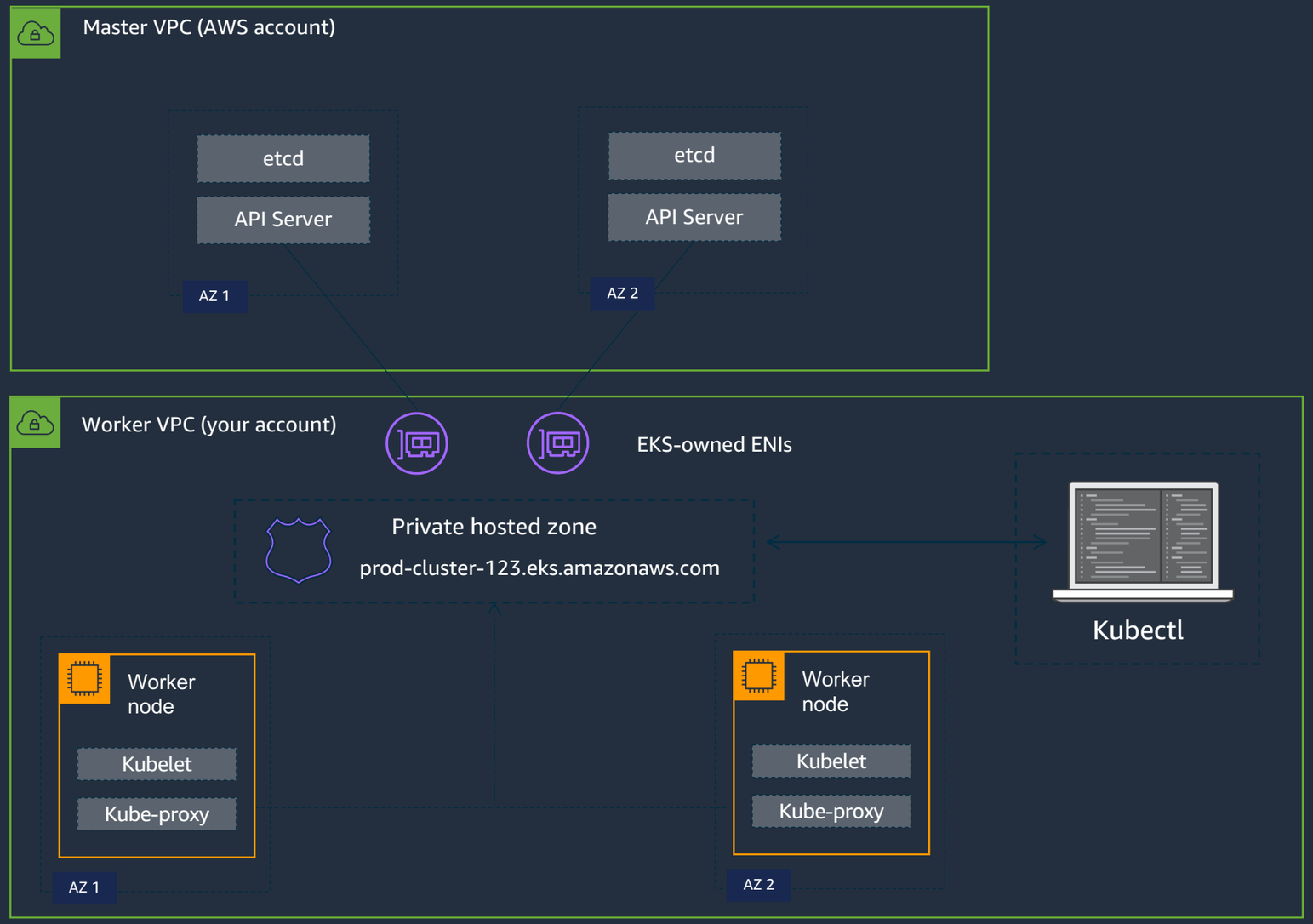
API-server Endpoint Access Control

```
public == true

private == true
```



API-server Endpoint Access Control

```
public == false

private == true
```



Kubernetes Manifests

Kubernetes Manifests

Deployment

Service

DaemonSet

StatefulSets

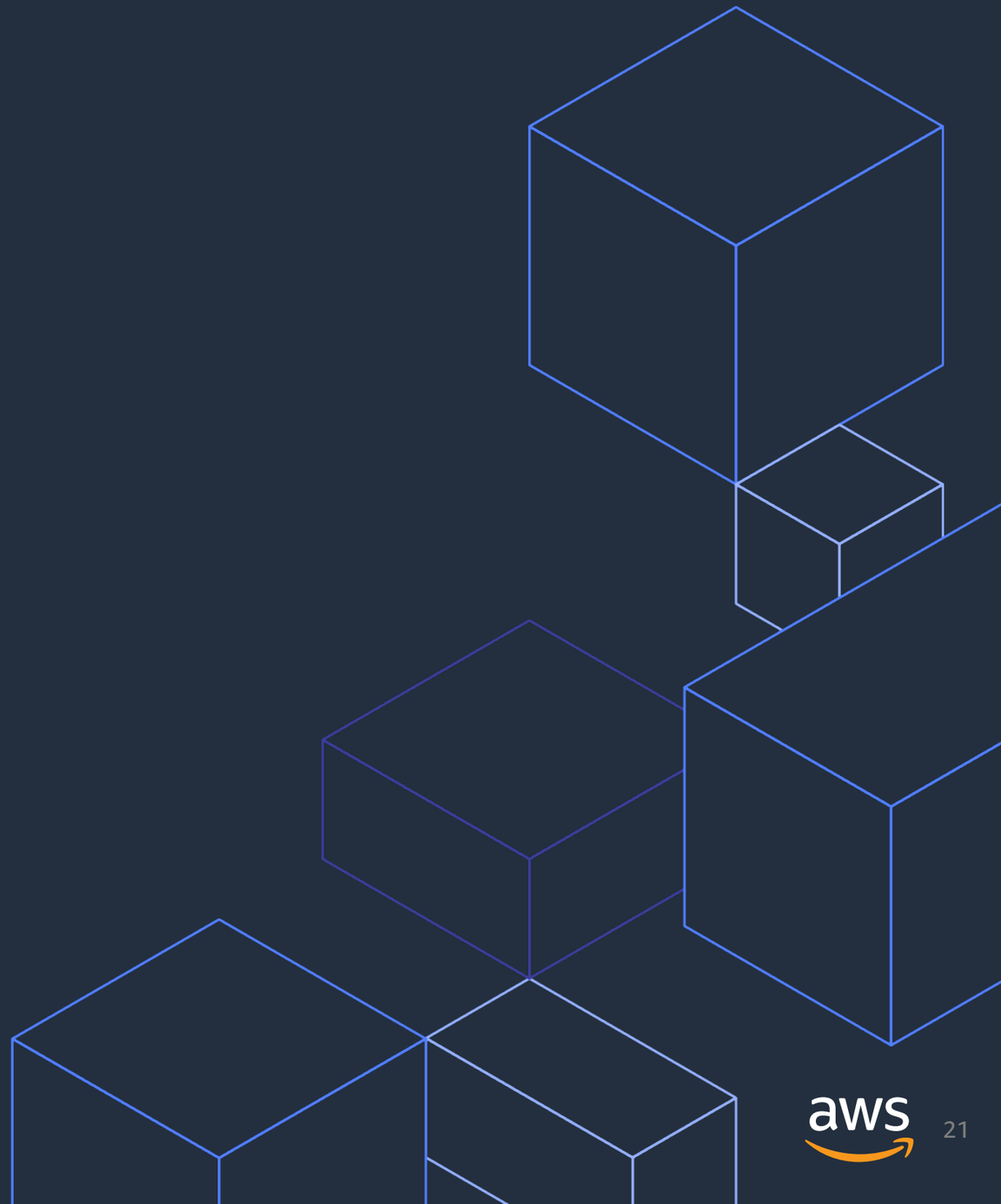
Jobs

PersistentVolume

PersistentVolumeClaim

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: myclaim
spec:
  accessModes:
    - ReadWriteOnce
  volumeMode: Filesystem
  resources:
    requests:
      storage: 8Gi
  storageClassName: slow
  selector:
    matchLabels:
      release: "stable"
    matchExpressions:
      - {key: environment, operator: In, values: [dev]}
```

EKS Networking

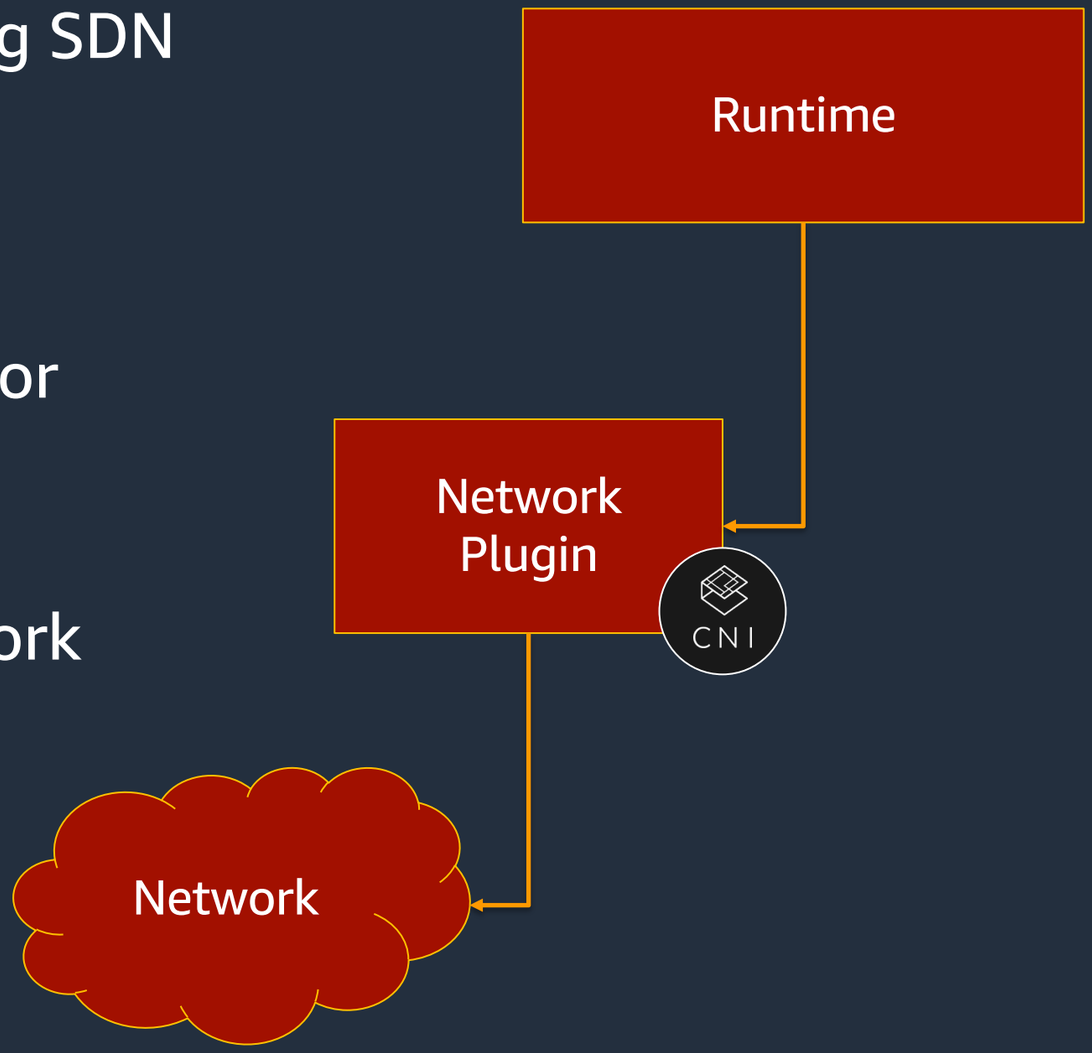


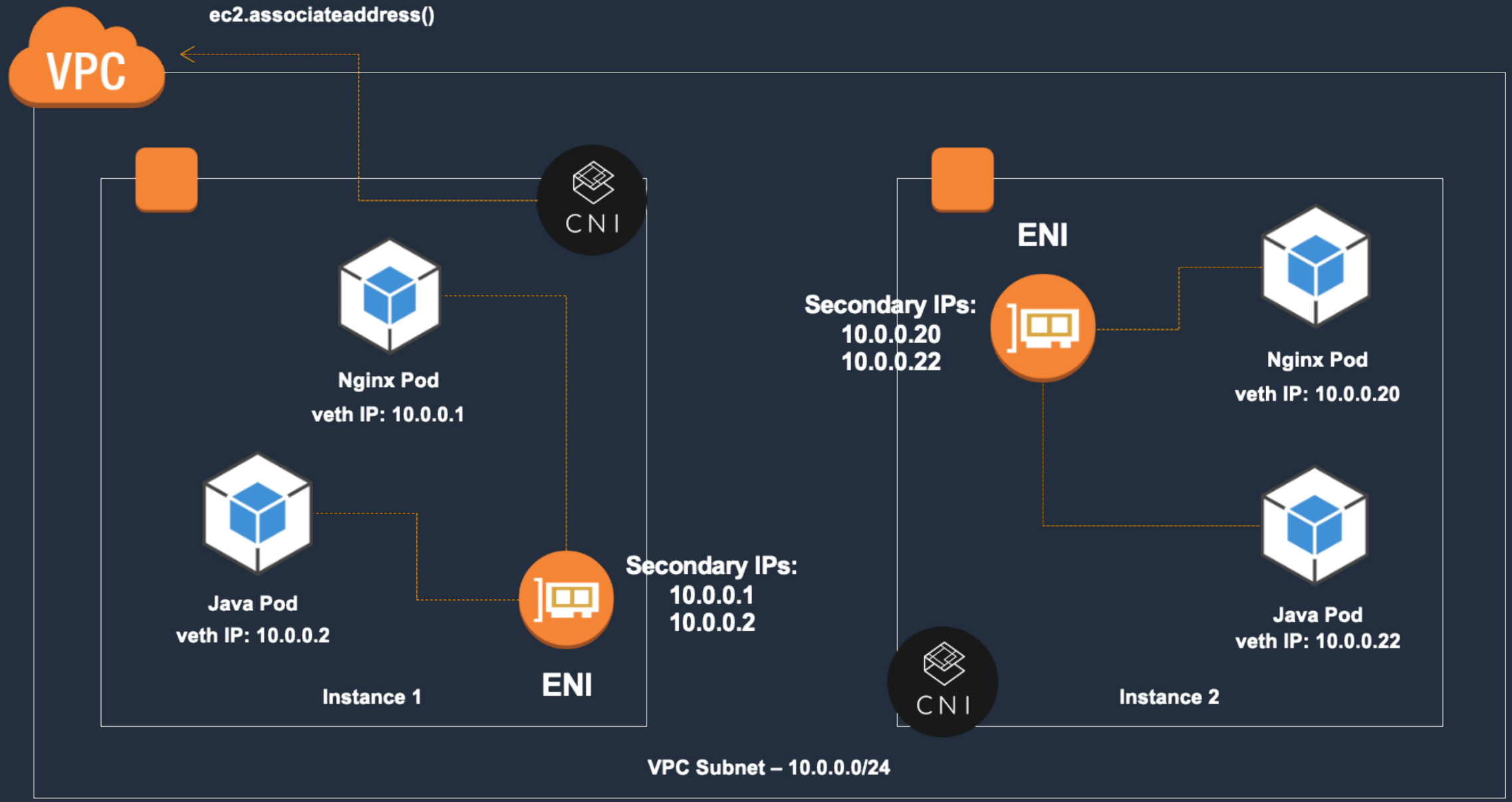
Networking With Kubernetes

"Every pod should have it's own IP address, and all pods should be able to talk to one and other"

What Is CNI

- A way for Kubernetes to tell an underlying SDN that it wants to connect a container to a network.
- Standards based pluggable architecture for container networking.
- API for writing plugins to configure network interfaces for containers.
- CNCF Project

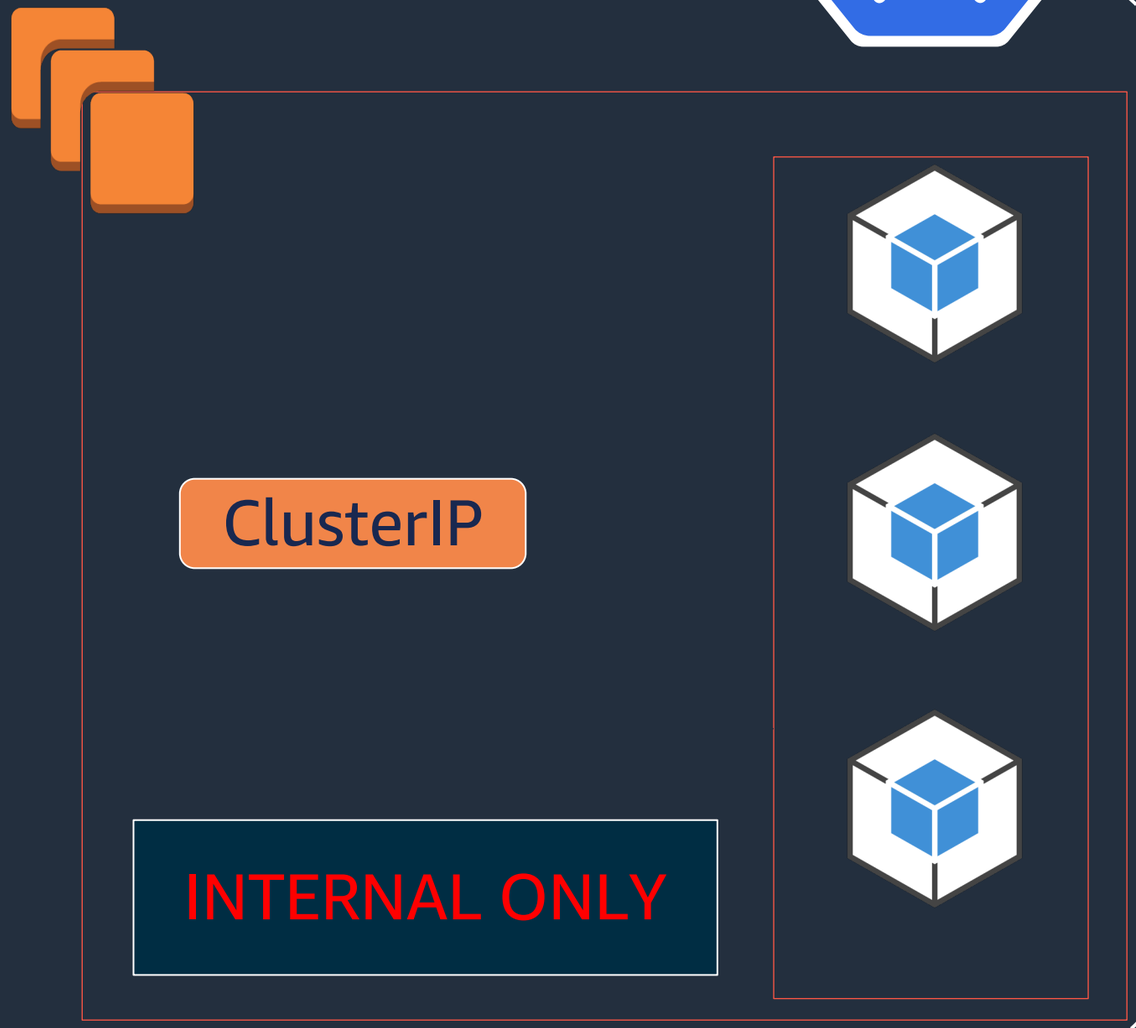




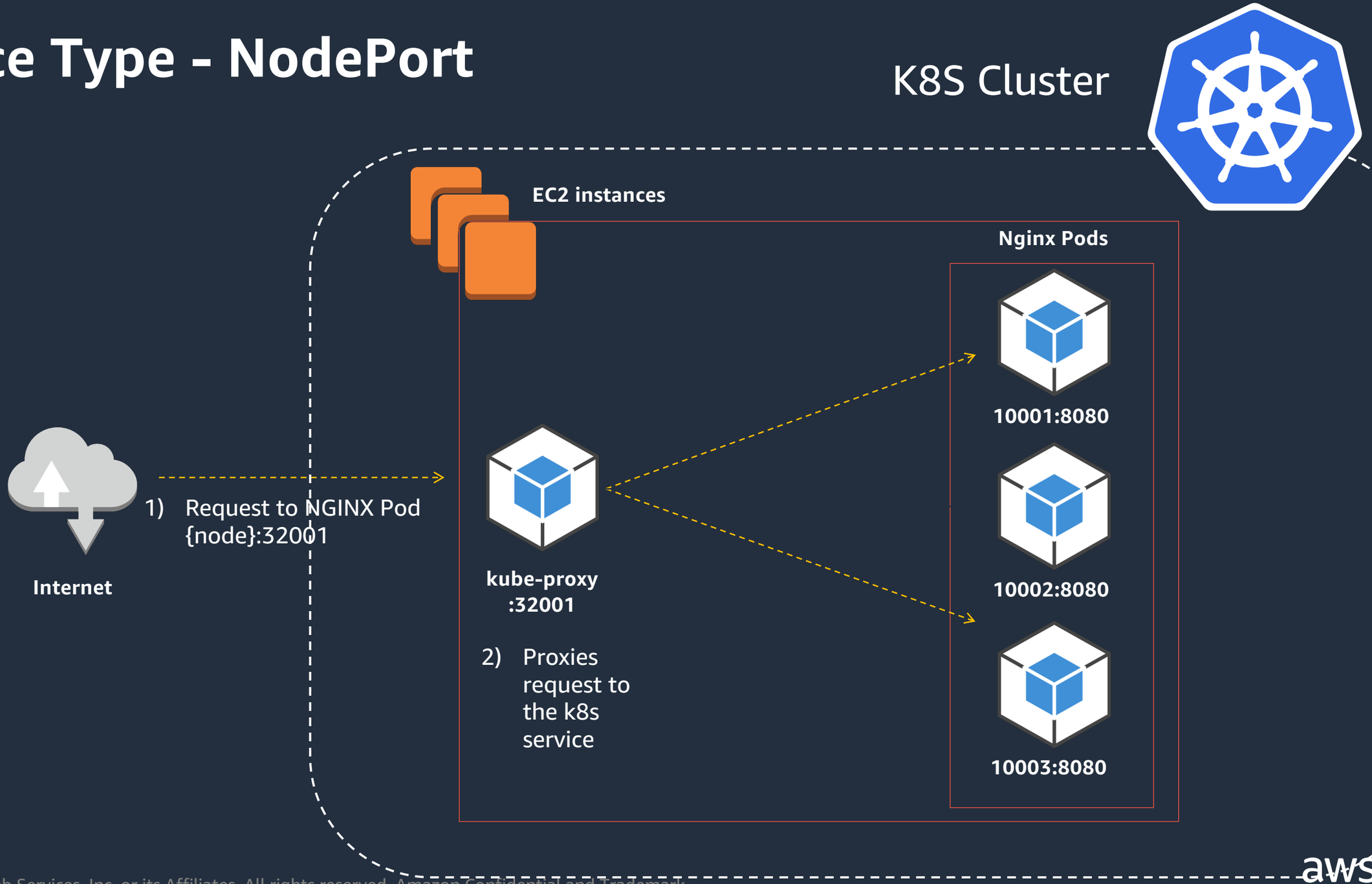
Service Type and Ingress

Service Type – ClusterIP (virtual)

```
-A KUBE-SEP-JNHR7XFBS7L5NBRR -p tcp -m comment --comment
"default/nginx-service:web" -m tcp -j DNAT --to-destination 10.0.3.33:80
-A KUBE-SEP-MJGBAZNA2WGVIMWN -s 10.0.3.177/32 -m comment --
comment "default/nginx-service:web" -j KUBE-MARK-MASQ
-A KUBE-SEP-MJGBAZNA2WGVIMWN -p tcp -m comment --comment
"default/nginx-service:web" -m tcp -j DNAT --to-destination 10.0.3.177:80
-A KUBE-SEP-XZ3DUOZYSFB3ILKR -s 10.0.1.100/32 -m comment --comment
"default/nginx-service:web" -j KUBE-MARK-MASQ
-A KUBE-SEP-XZ3DUOZYSFB3ILKR -p tcp -m comment --comment
"default/nginx-service:web" -m tcp -j DNAT --to-destination 10.0.1.100:80
-A KUBE-SERVICES -d 172.20.169.0/32 -p tcp -m comment --comment
"default/nginx-service:web cluster IP" -m tcp --dport 80 -j KUBE-SVC-
MCOVNBHDEGIKKKLL
-A KUBE-SVC-MCOVNBHDEGIKKKLL -m comment --comment "default/nginx-
service:web" -m statistic --mode random --probability 0.33332999982 -j
KUBE-SEP-XZ3DUOZYSFB3ILKR
-A KUBE-SVC-MCOVNBHDEGIKKKLL -m coment --comment "default/nginx-
service:web" -m statistic --mode random --probability 0.50000000000 -j
KUBE-SEP-MJGBAZNA2WGVIMWN
-A KUBE-SVC-MCOVNBHDEGIKKKLL -m comment --comment "default/nginx-
service:web" -j KUBE-SEP-JNHR7XFBS7L5NBRR
```



Service Type - NodePort



Load balancing

- All three AWS Elastic Load Balancing products are supported

- NLB and CLB supported by Kubernetes Service type=LoadBalancer

- Internal and External Load Balancer support

Load balancing

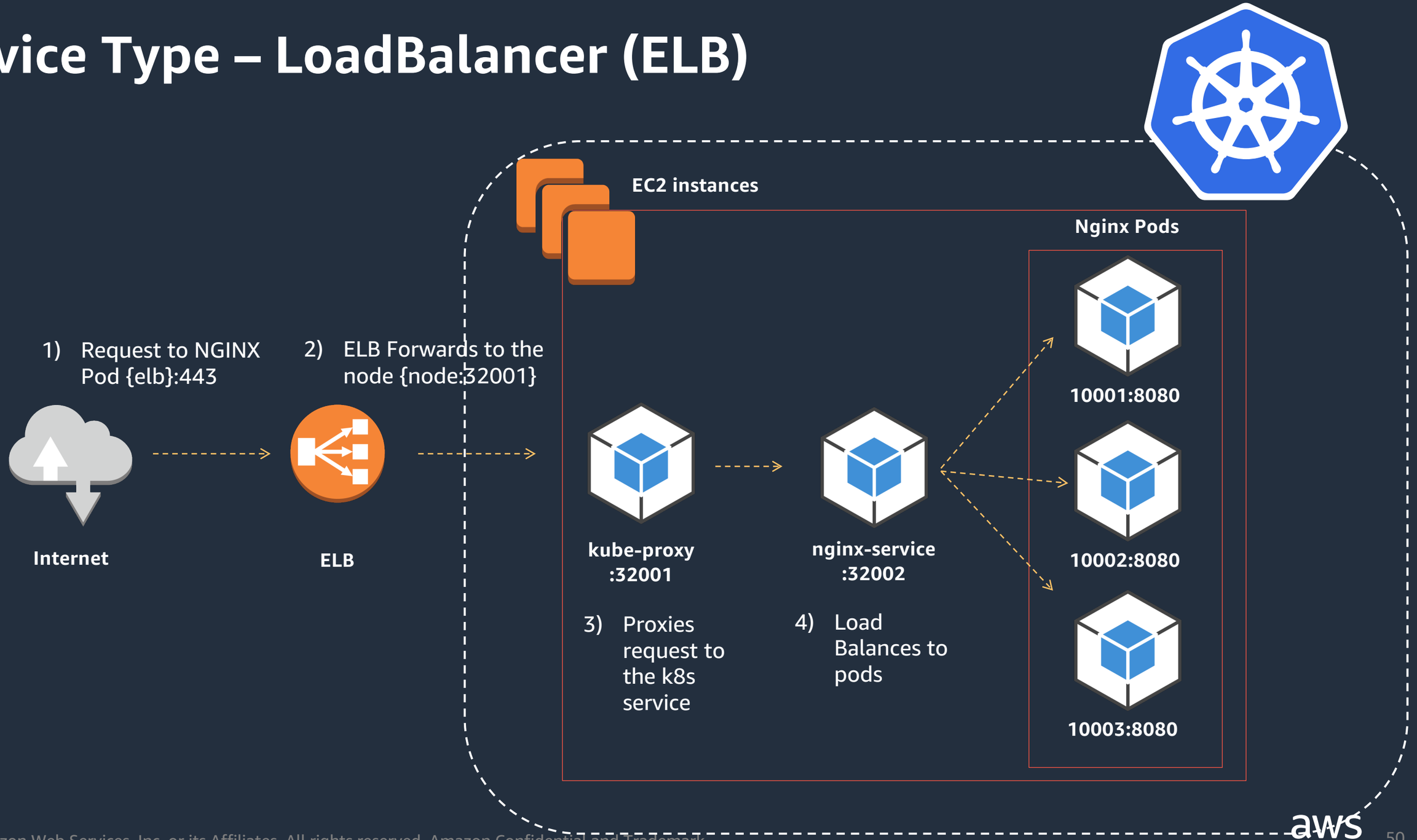
Want to use an Internal Load Balancer? Use annotation:

```
service.beta.kubernetes.io/aws-load-balancer-internal:
0.0.0.0/0
```

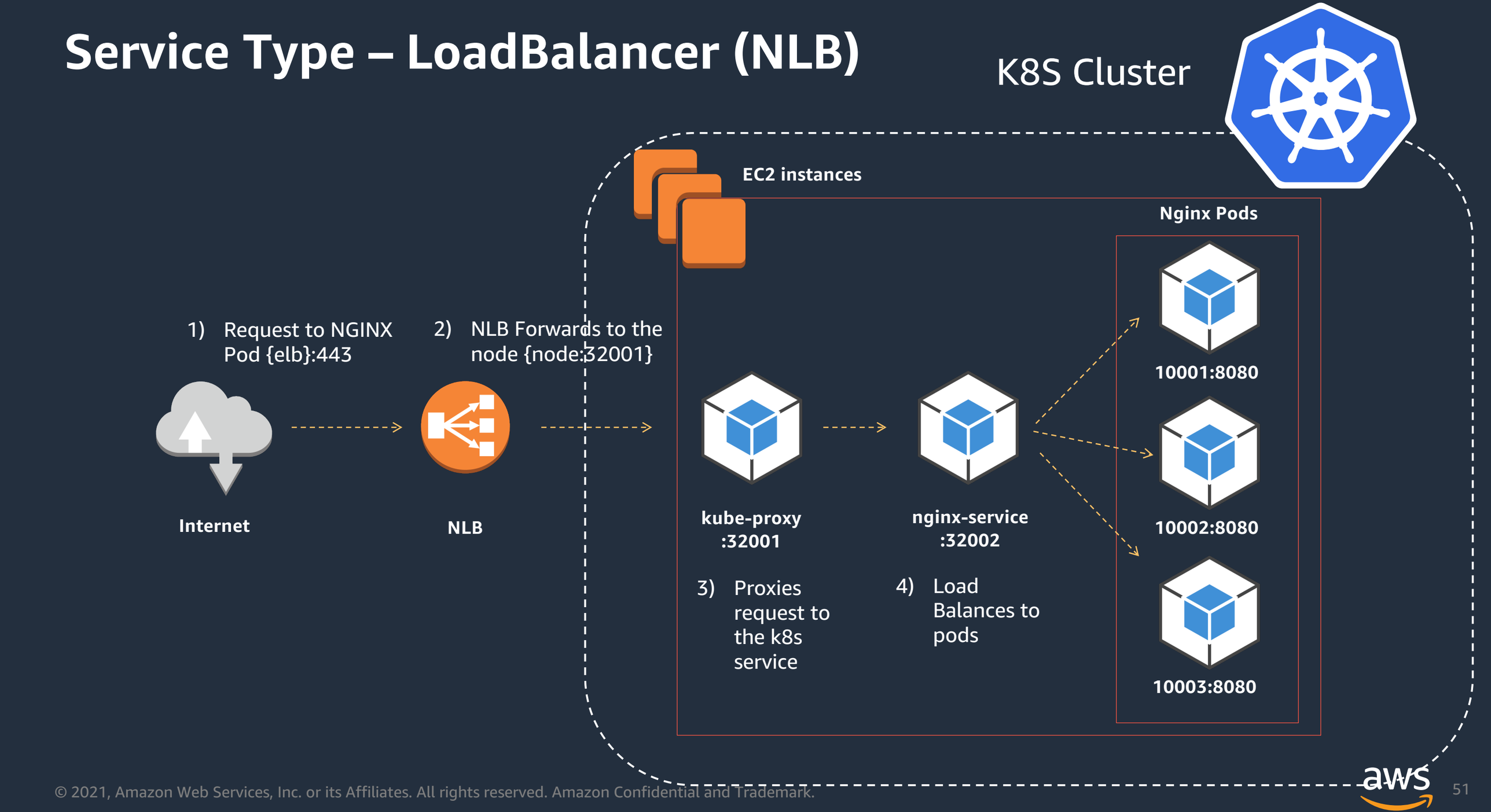
Want to use an NLB? Use annotation:

```
service.beta.kubernetes.io/aws-load-balancer-type: nlb
```

Service Type – LoadBalancer (ELB)



Service Type – LoadBalancer (NLB)



ALB Ingress controller

- Production-ready 1.0 release

- Supported by Amazon EKS team

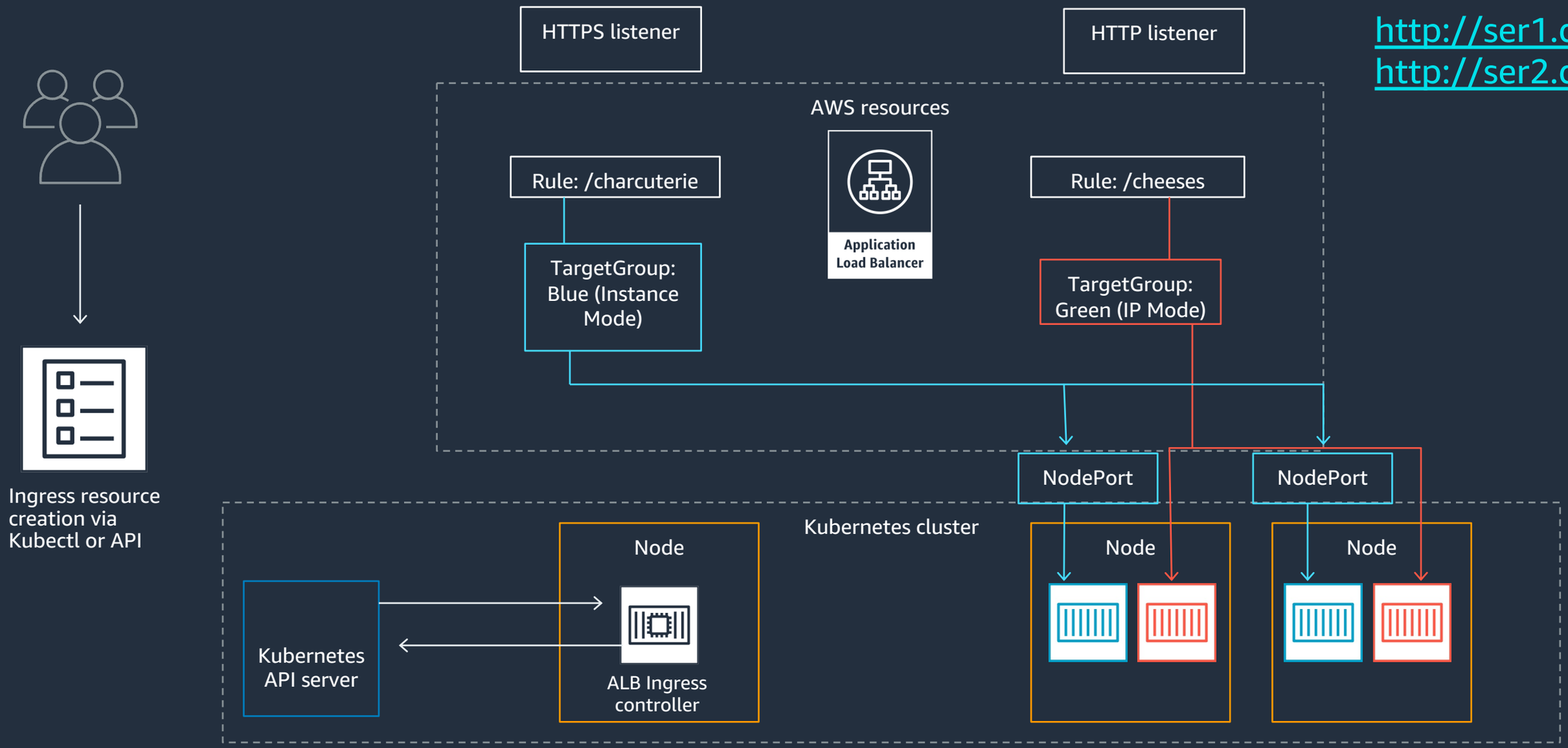
- Open source development: https://github.com/kubernetes-sigs/aws-alb-ingress-controller

- Customers are using it in production today!

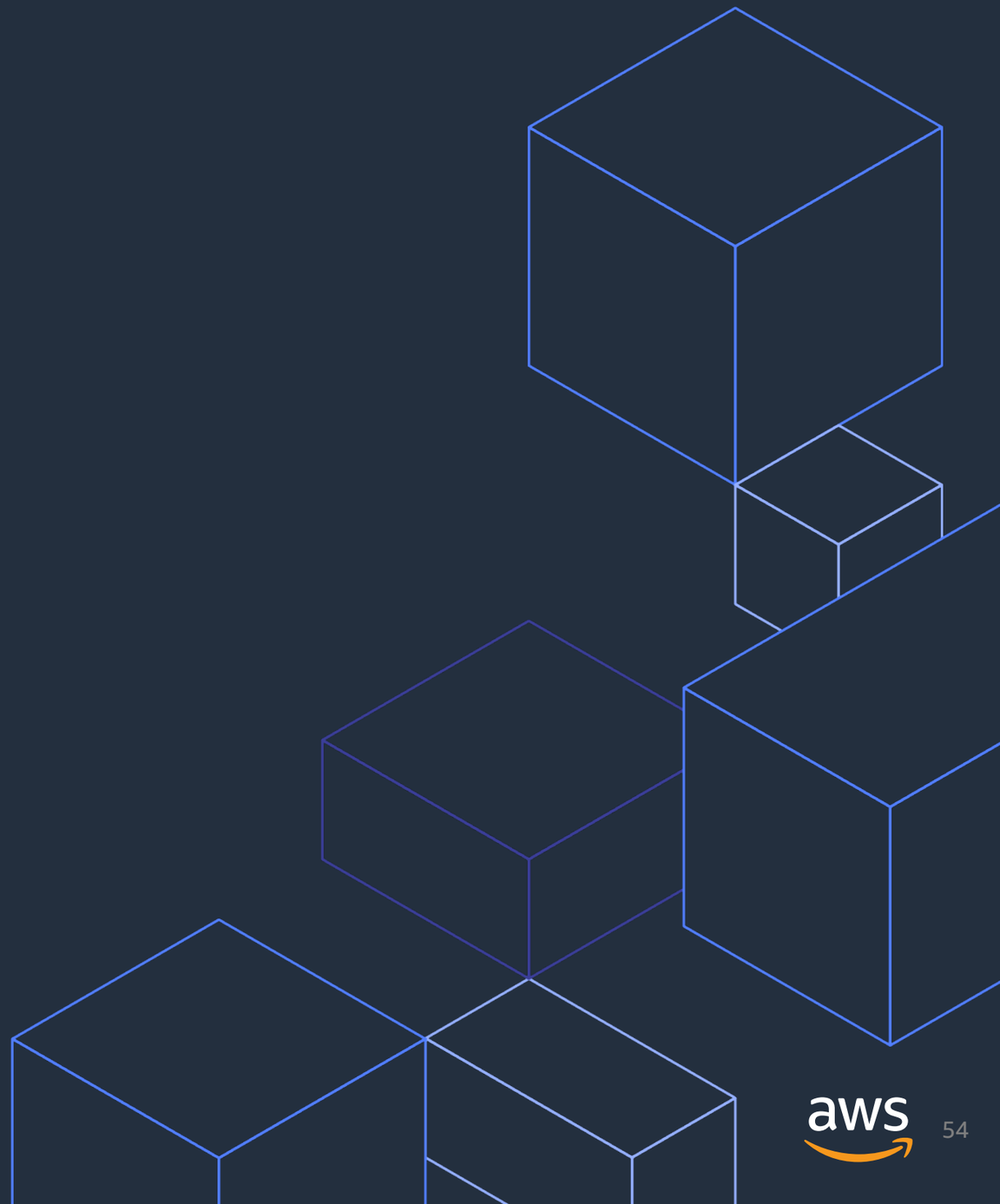
ALB Ingress controller

Nginx-10.10.0.5

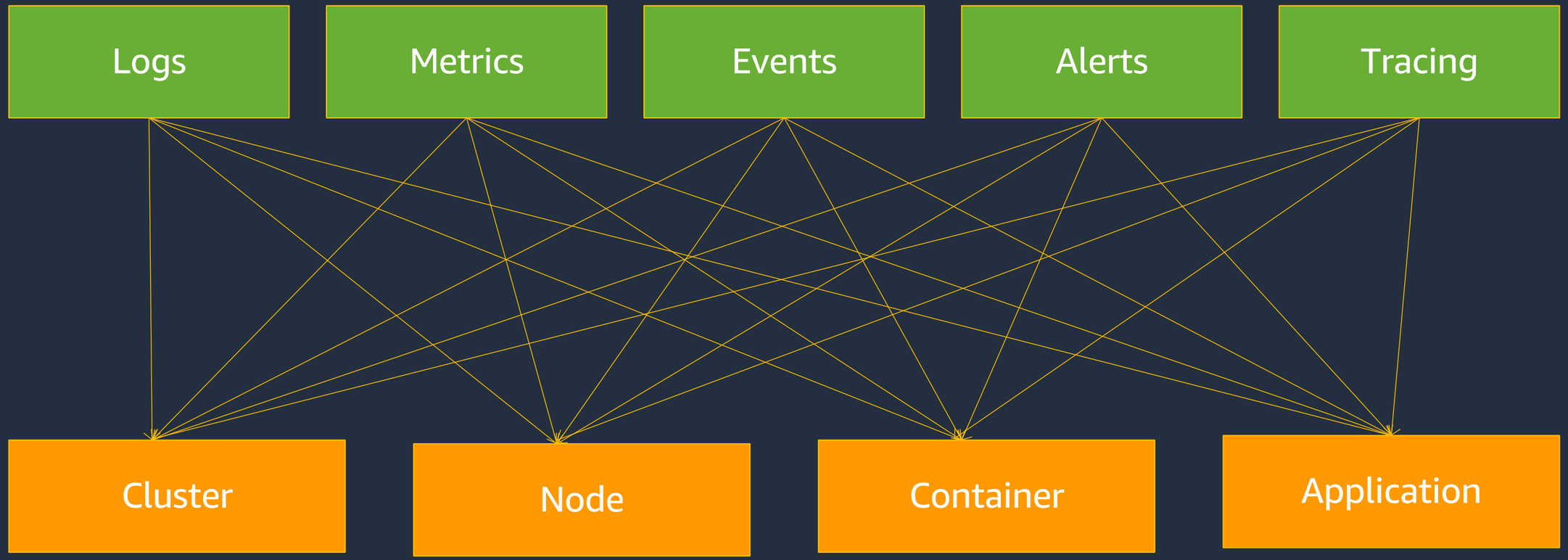
http://ser1.default.svc
http://ser2.defaulrt.svc



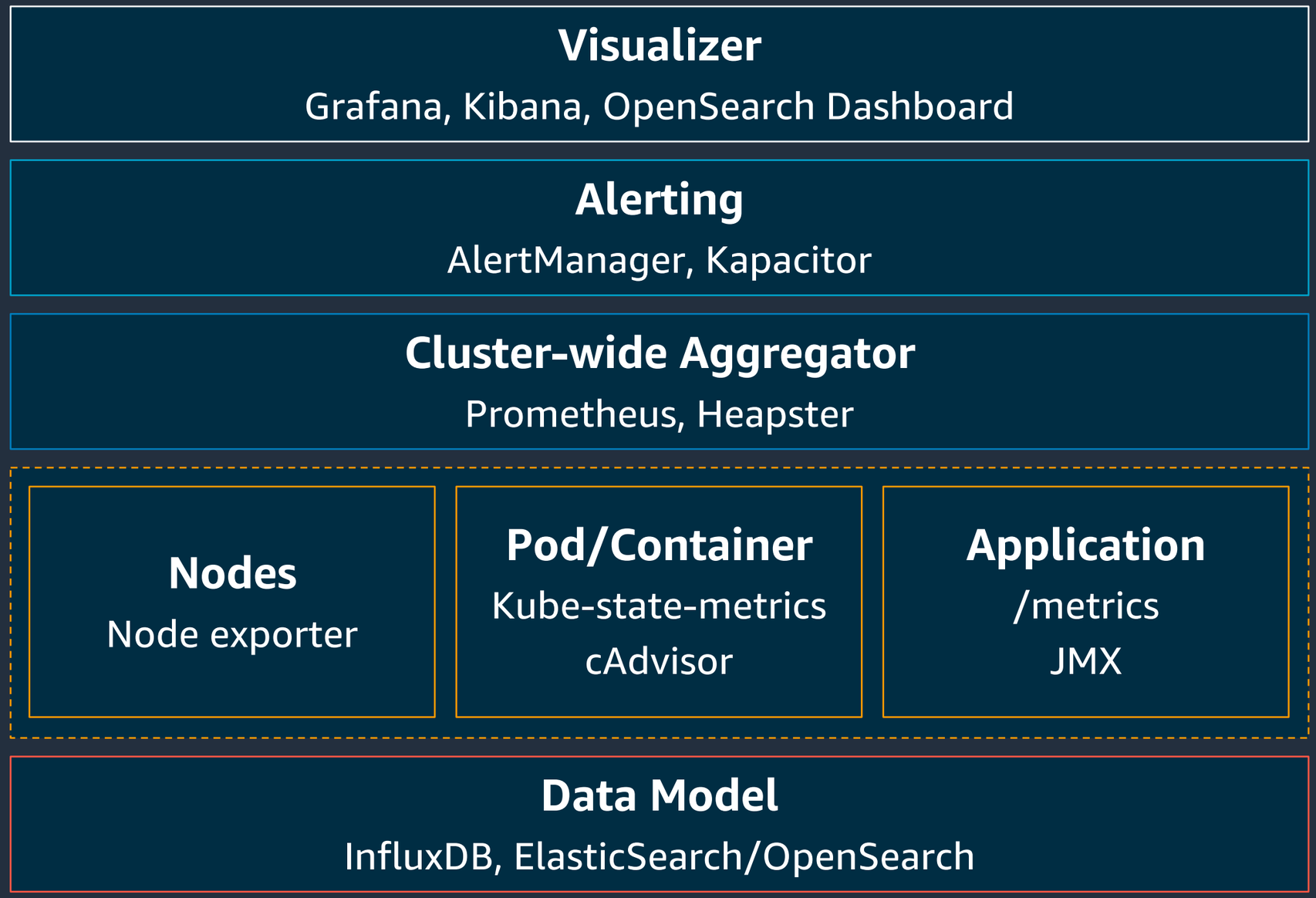
Monitoring



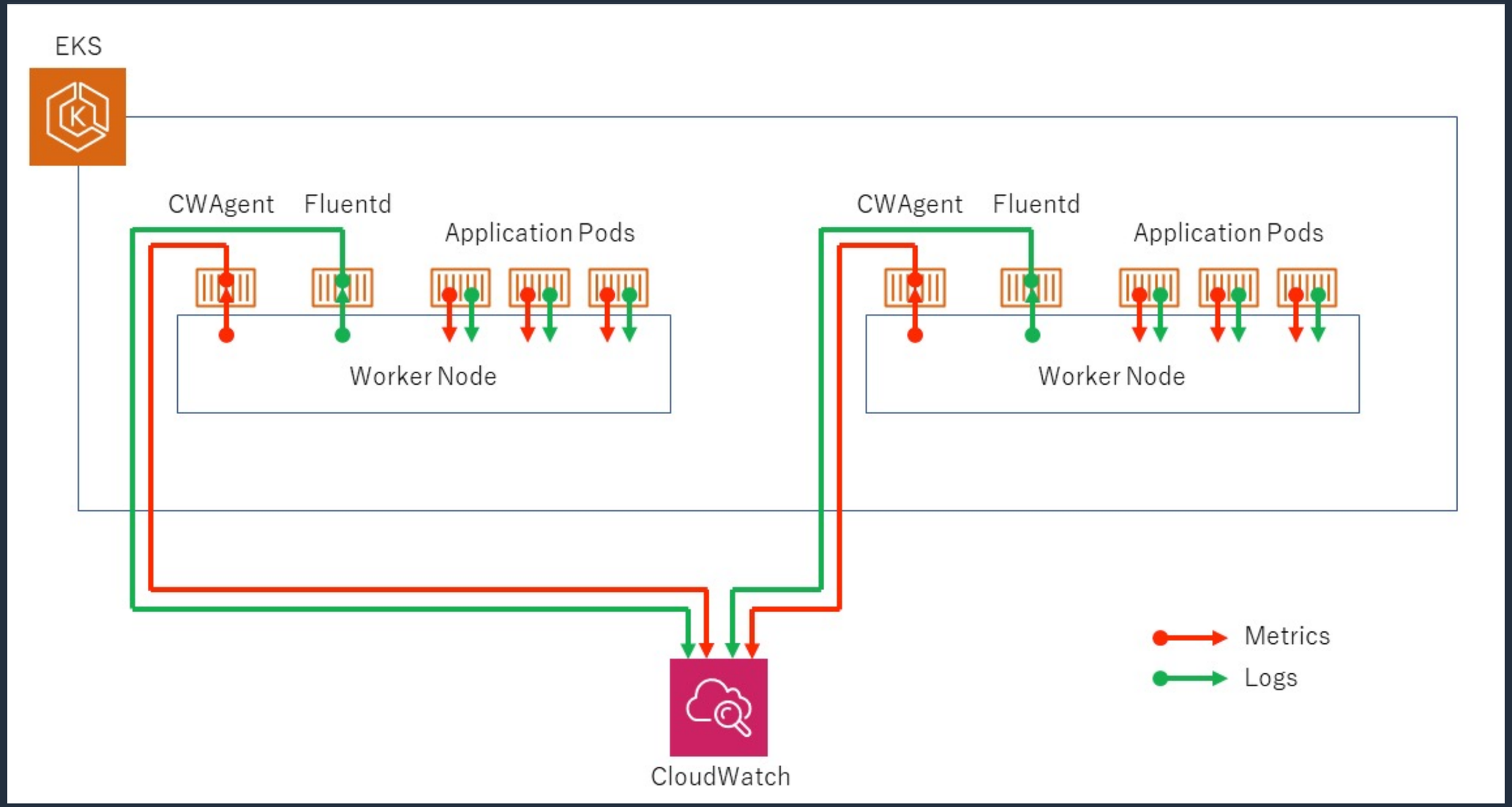
Visibility About Your Kubernetes Cluster



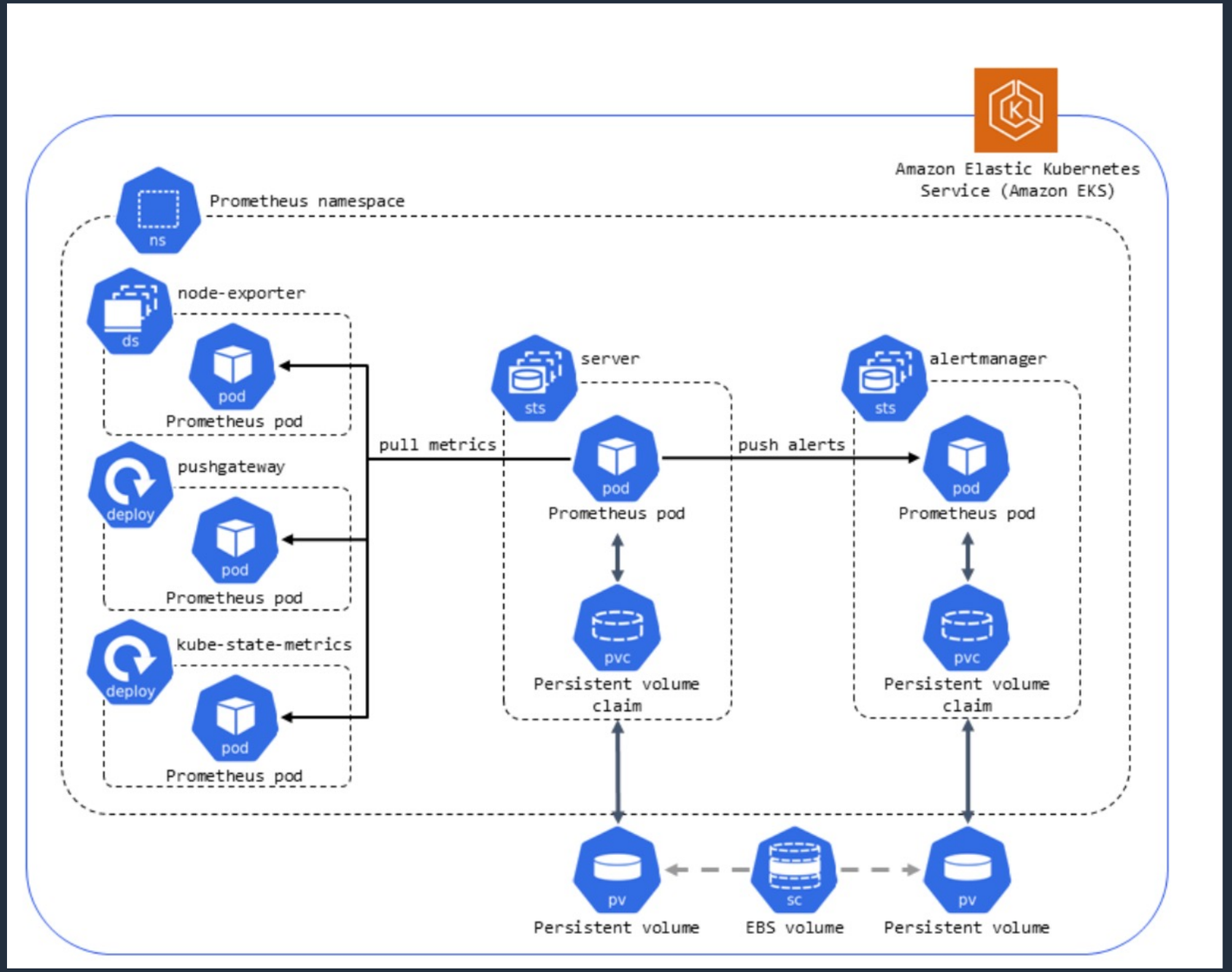
Metrics



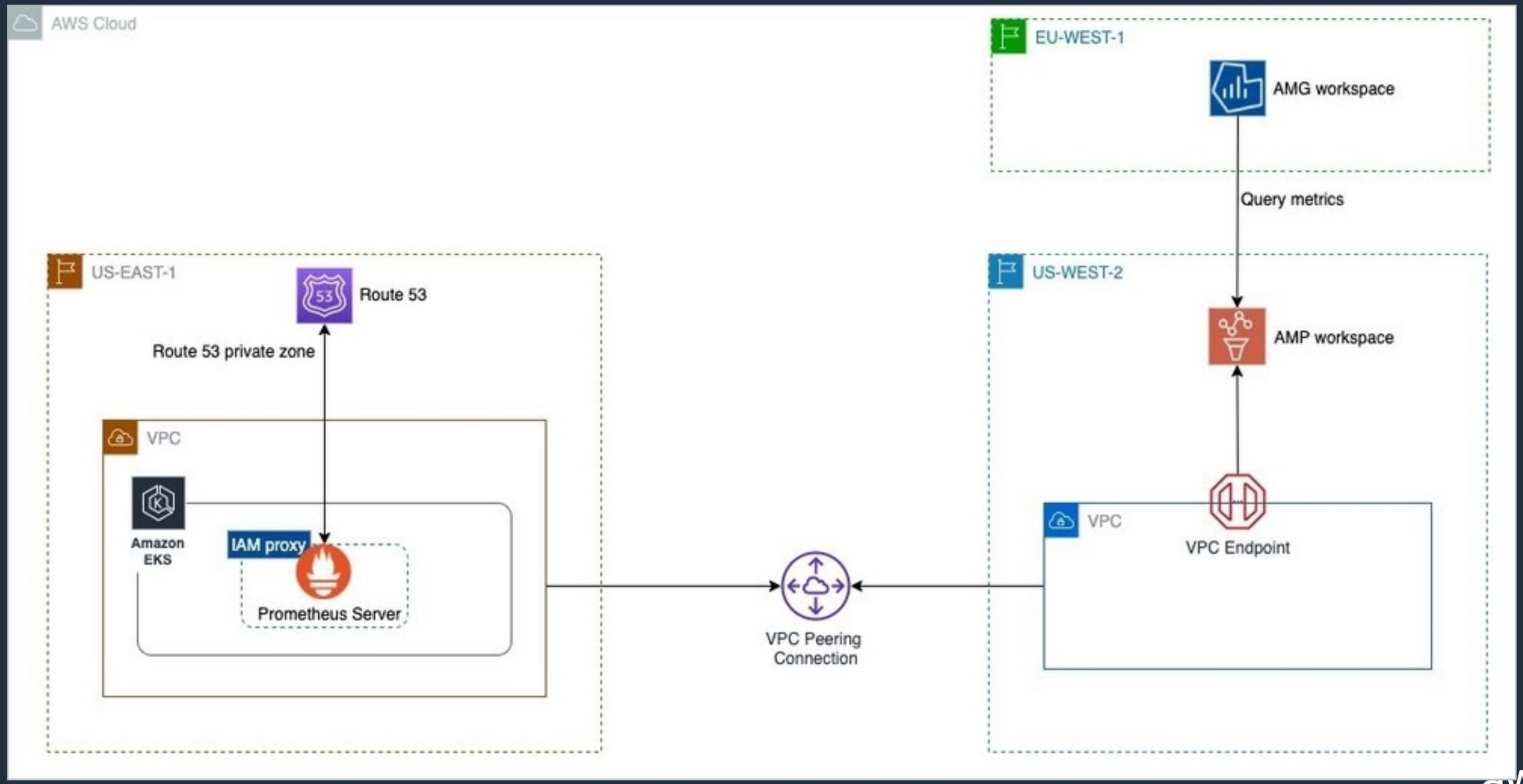
EKS Monitoring with Container Insight



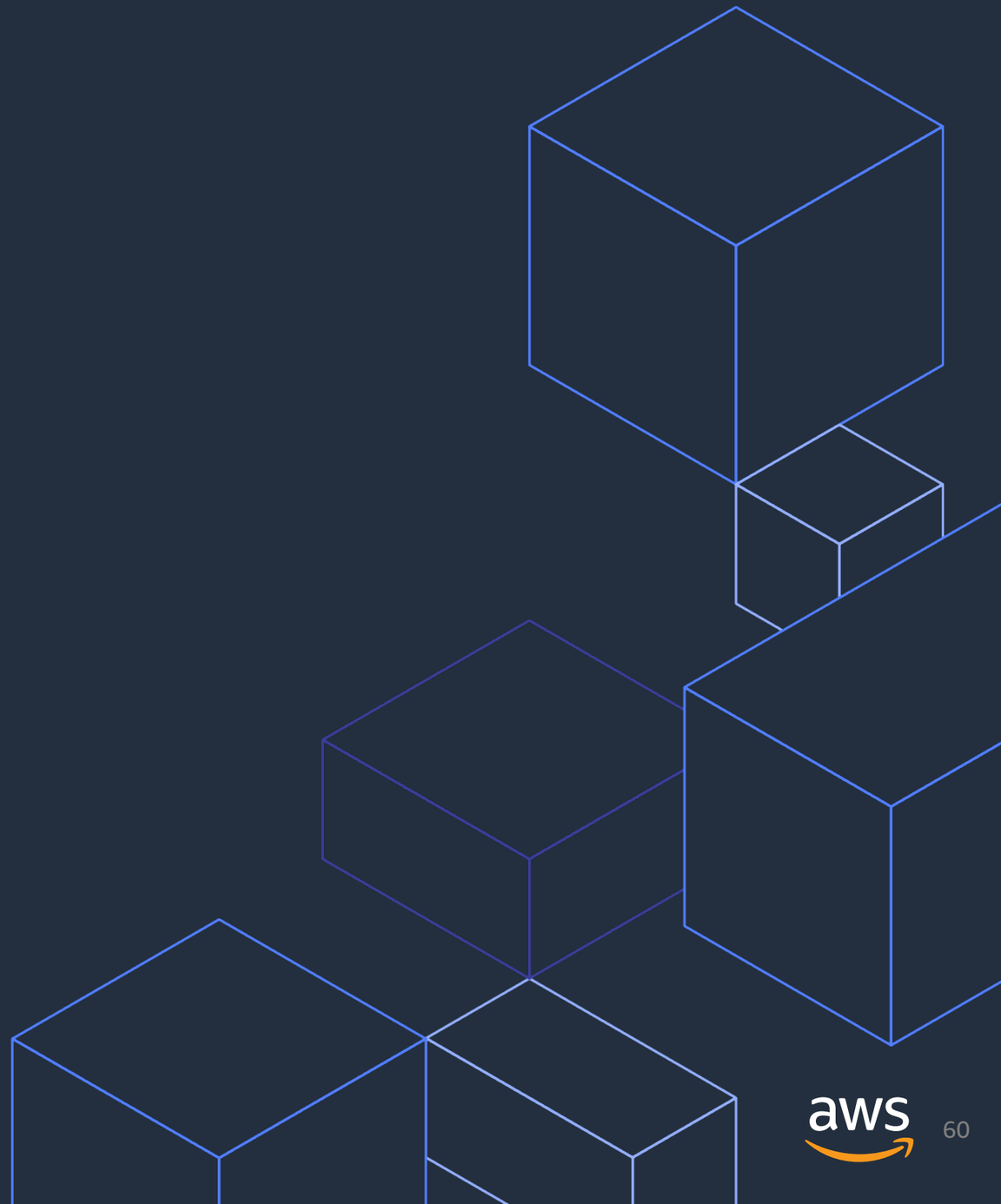
EKS Monitoring with Prometheus



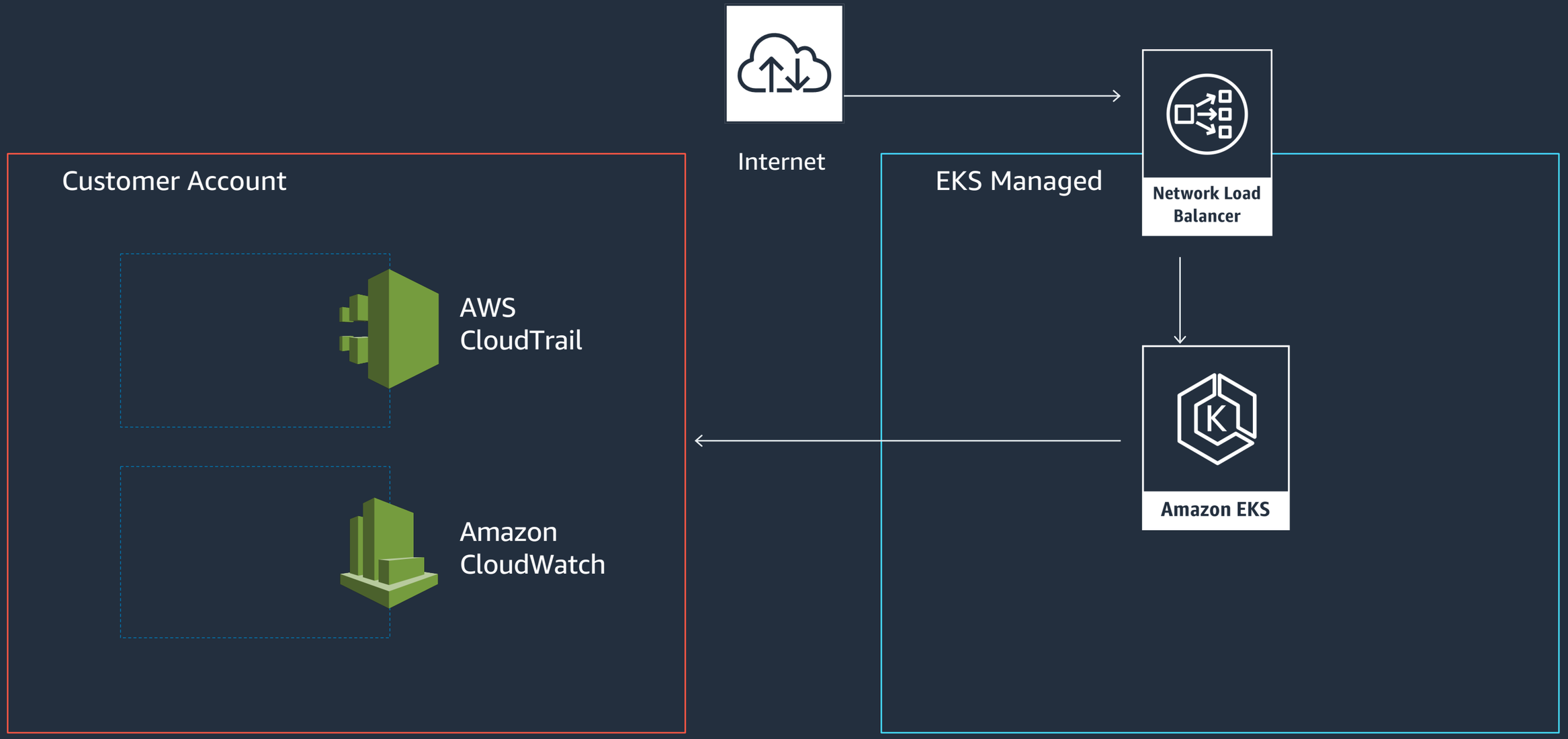
EKS Monitoring with Managed Prometheus & Grafana



Logging



EKS Logging



Cloudwatch and EKS integration: Logging

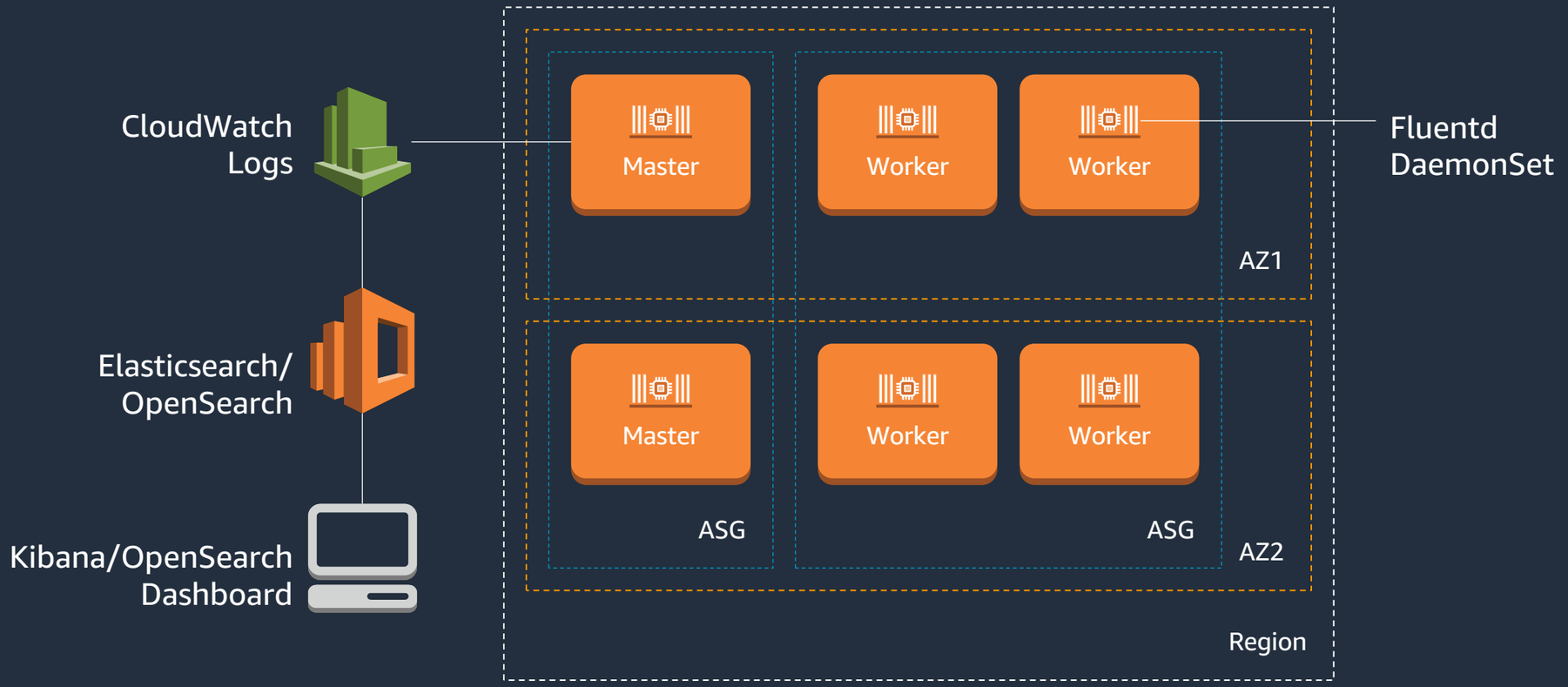
- EKS Control plane logging
  - Kubernetes API server component logs (api)
  - Audit
  - Authenticator
  - Controller manager
  - Scheduler
- Logging to CloudWatch

https://eksworkshop.com/logging/

EKS Logging

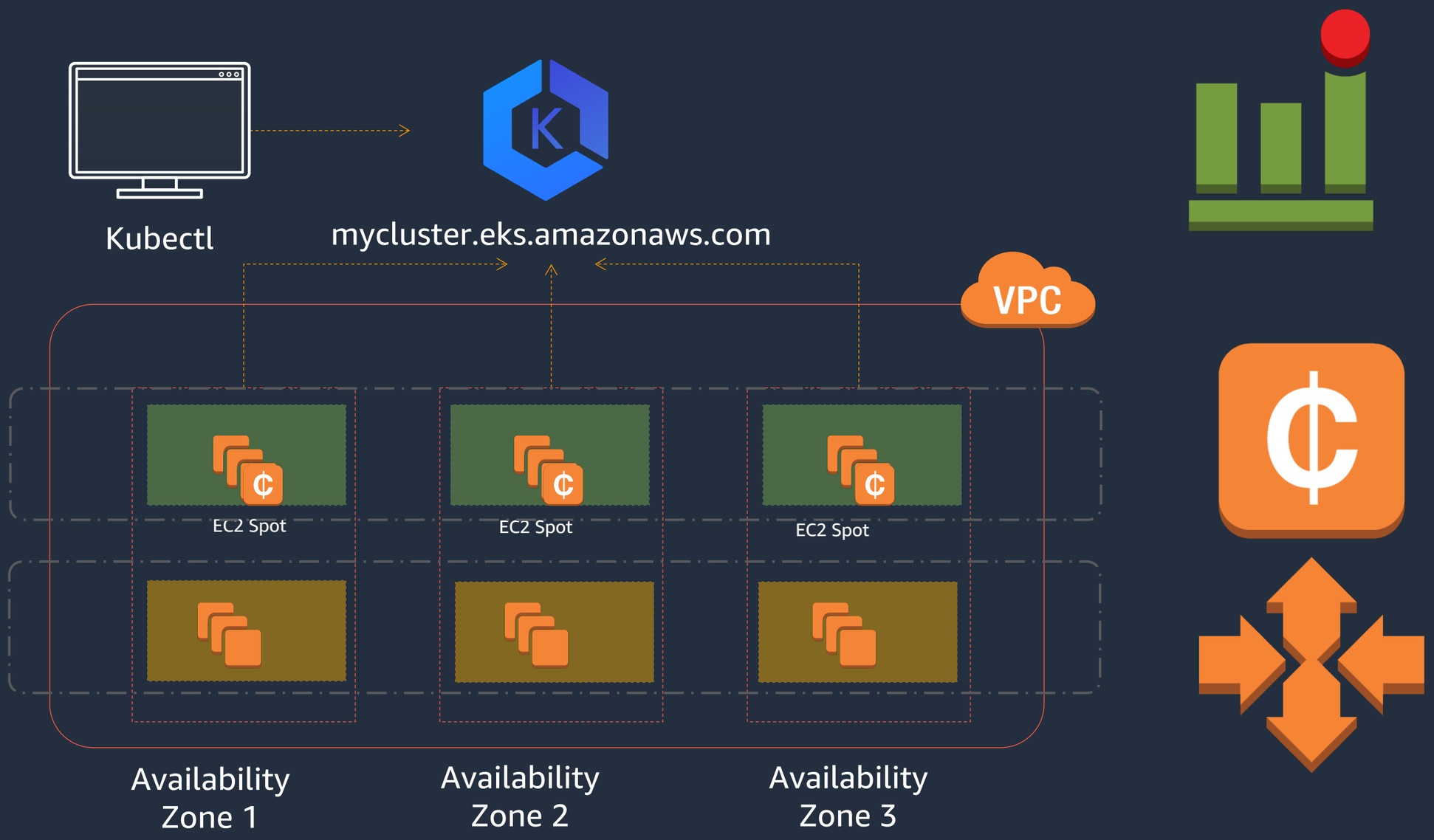
Kubectl logs

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

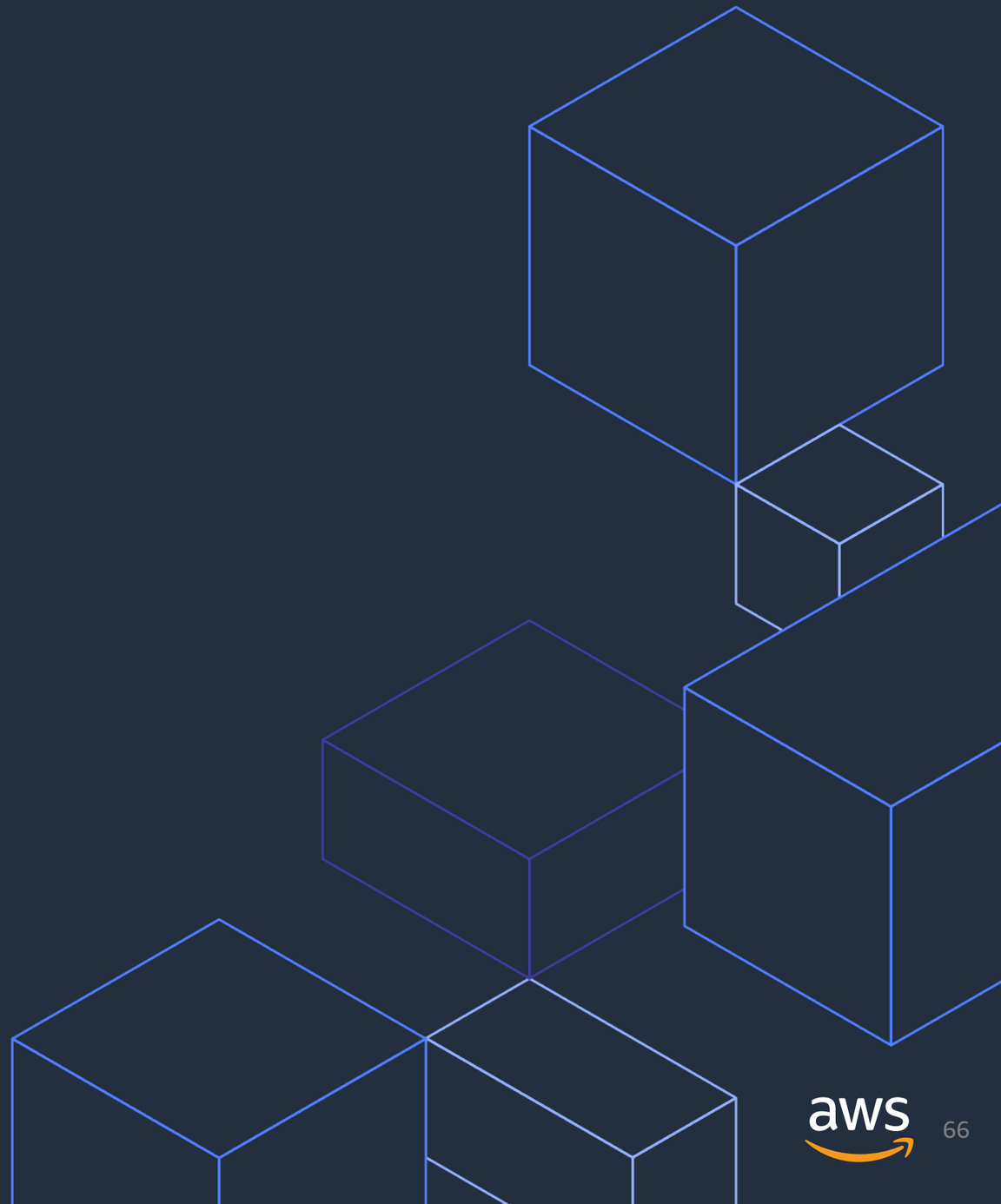


AutoScaling in EKS

Cluster Autoscaling



Storage



Container Storage Interface (CSI)

A flexible standard for orchestration
and storage provider connections

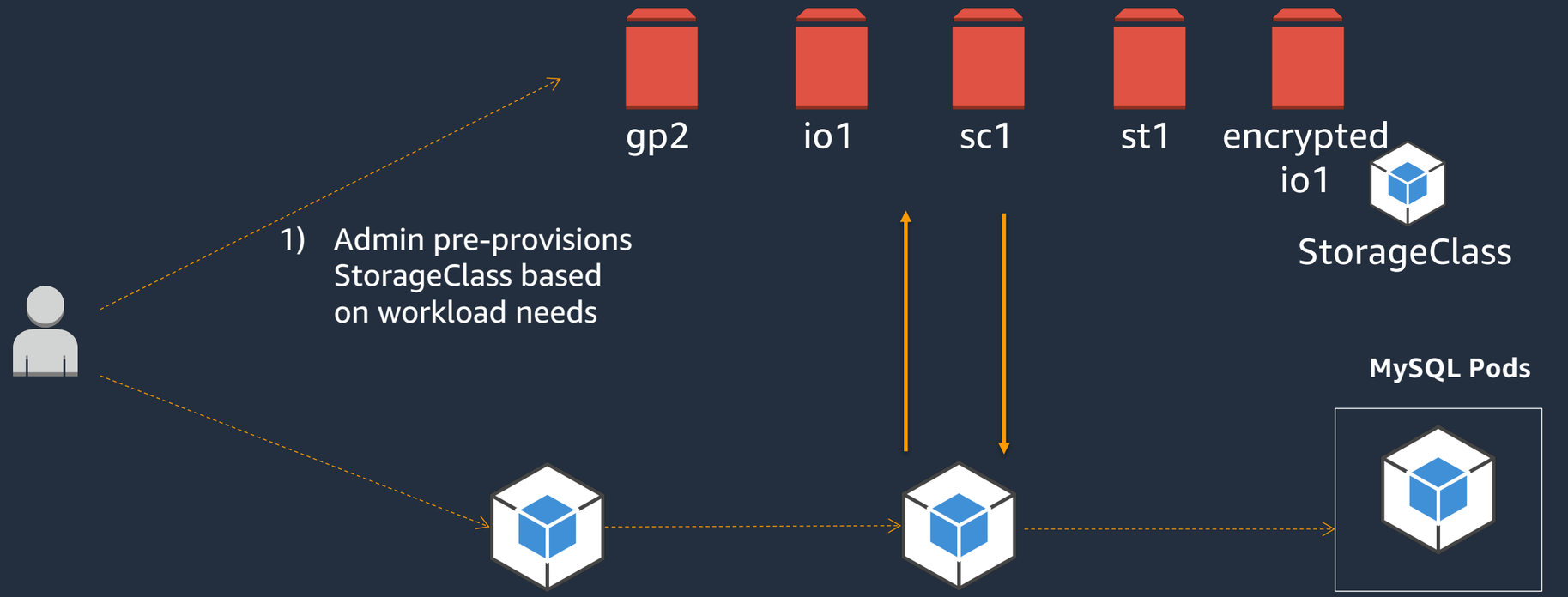
We support the CSI standard through following drivers:

Amazon Elastic Block Store: AWS EBS CSI Driver

Amazon Elastic File System: AWS EFS CSI Driver

Amazon FSx for Lustre: AWS FSx CSI Driver

What if I need specific volume type?



Example – Persistent Volume

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: efs-sc
provisioner: efs.csi.aws.com
parameters:
  provisioningMode: efs-ap
  fileSystemId: fs-92107410
  directoryPerms: "700"
  gidRangeStart: "1000" # optional
  gidRangeEnd: "2000" # optional
  basePath: "/dynamic_provisioning"
```

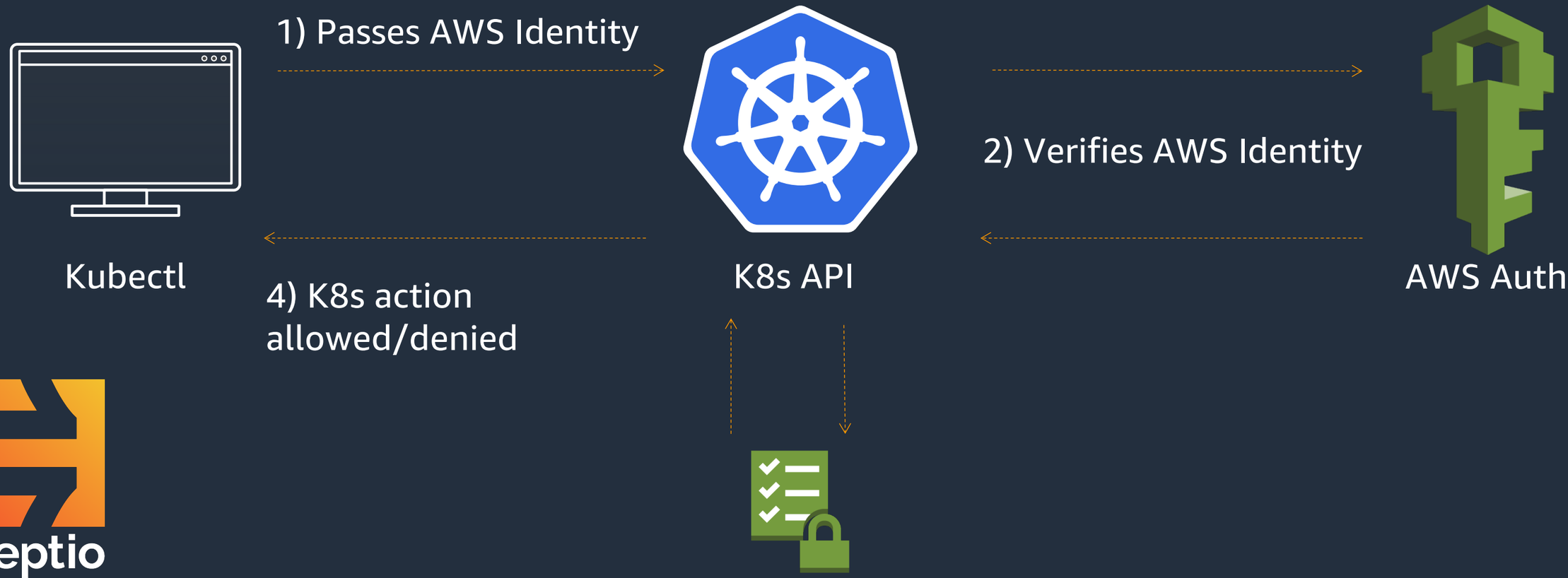
```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: efs-claim
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: efs-sc
  resources:
    requests:
      storage: 5Gi
```

```
apiVersion: v1
kind: Pod
metadata:
  name: efs-app
spec:
  containers:
    - name: app
      image: centos
      command: ["/bin/sh"]
      args: ["-c", "while true; do echo $(date -u)
>> /data/out; sleep 5; done"]
      volumeMounts:
        - name: persistent-storage
          mountPath: /data
  volumes:
    - name: persistent-storage
      persistentVolumeClaim:
        claimName: efs-claim
```

Authentication & Security

IAM Authentication + Kubectl

Amazon EKS uses IAM to provide authentication to your Kubernetes cluster (through the aws eks get-token command)



heptio

Cluster Authentication and Authorization

- User or IAM role who **creates** 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
```

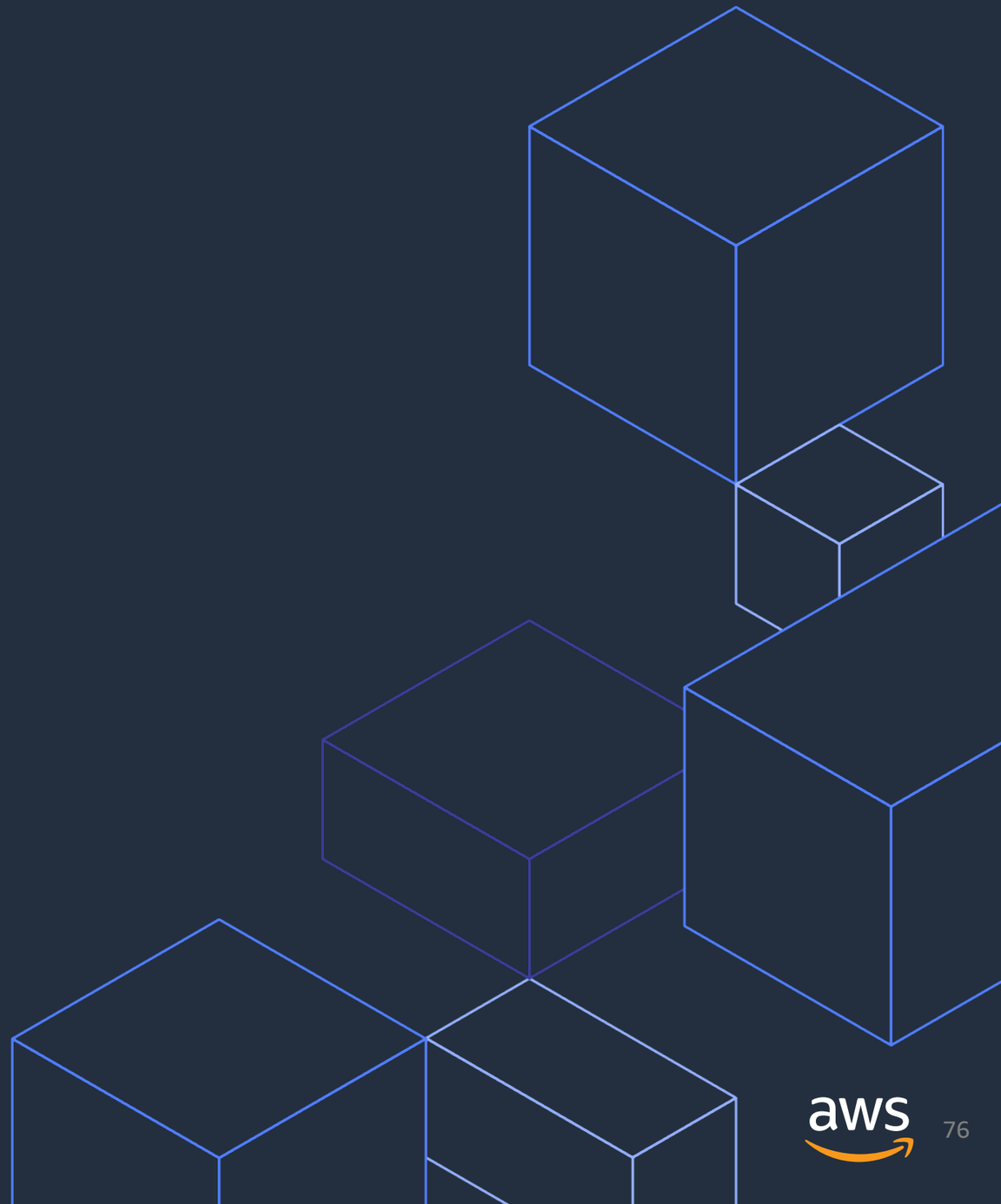
aws-auth configuration

```
apiVersion: v1
data:
  mapRoles: |
    - rolearn: arn:aws:iam::555555555555:role/devel-worker-nodes-NodeInstanceRole-74RF4UBDUKL6
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes
  mapUsers: |
    - userarn: arn:aws:iam::555555555555:user/admin
      username: admin
      groups:
        - system:masters
    - userarn: arn:aws:iam::555555555555:user/john
      username: john
      groups:
        - pod-admin                    # k8s RBAC group
```

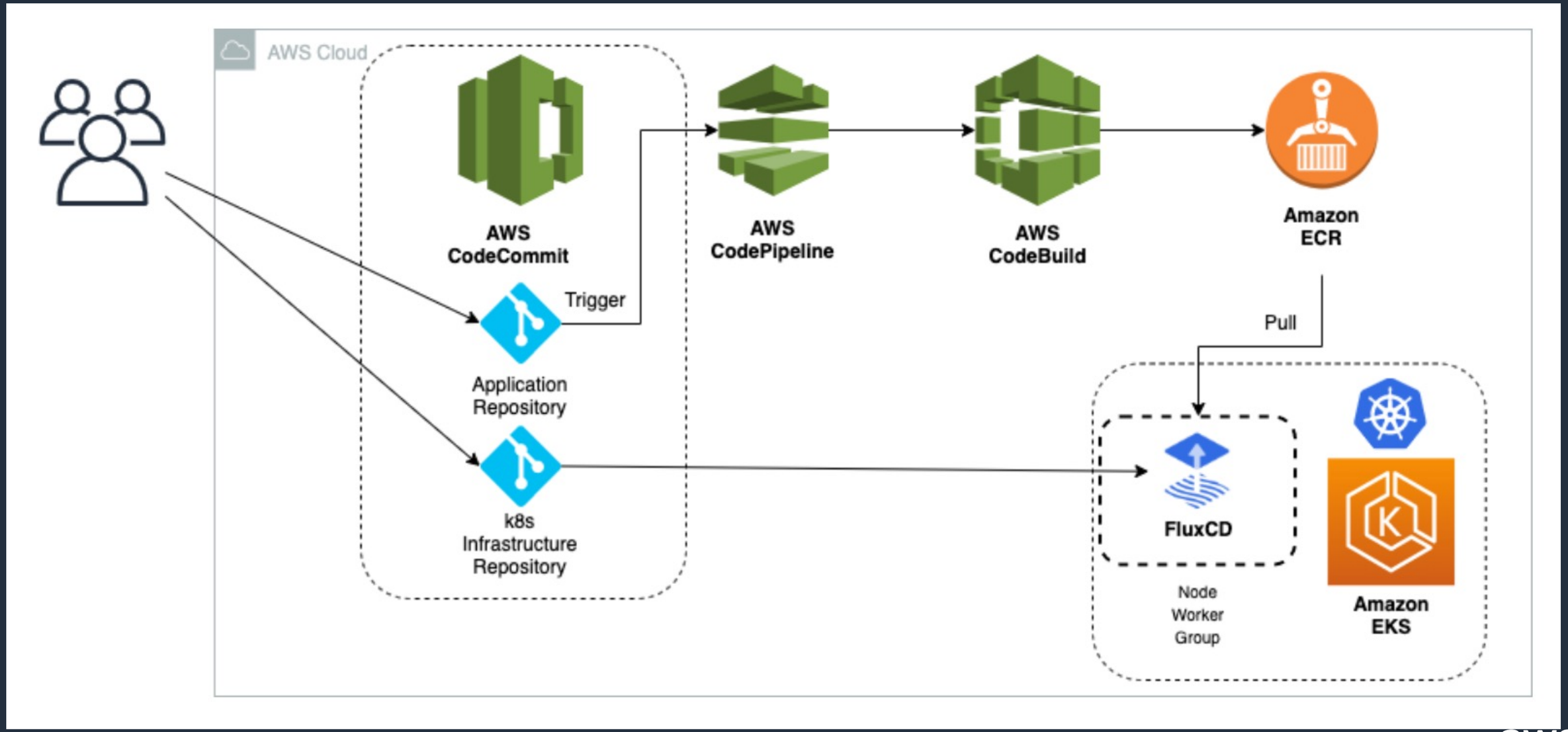
Admin level privileges

Privileges defined by RBAC

Build, Ship, Run...



EKS and GitOps



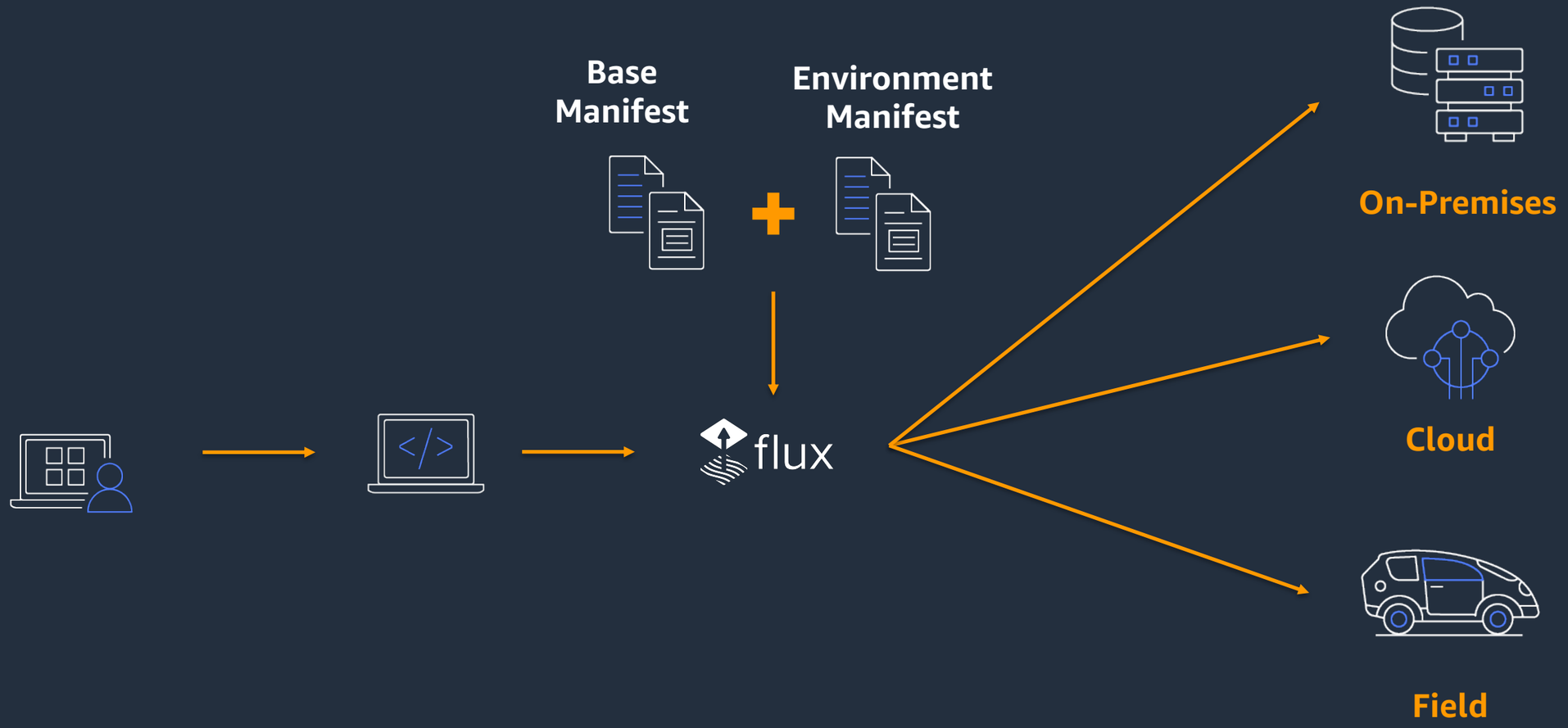
Managing Applications with Helm

Helm is referred to as the package manager for Kubernetes.


```
apiVersion: helm.fluxcd.io/v1 kind: HelmRelease
metadata:
        name: users-service
        namespace: application
        annotations:
                fluxcd.io/automated: "true”
                filter.fluxcd.io/chart-image: glob:dev-*
spec:
        releaseName: users-service
        chart:
                git: git@git.acmecorp.com/application/helm-charts
                path: charts/users-service
                ref: master
                values:
                        image:
                                repository: hub.acmecorp.com/users-service
                                tag: master-123456-build10
```

Managing Deployments to Multiple Environments

*Flux supports **Kustomize**, which enables the use of multiple files to **compose the Kubernetes manifest depending on the environment.***



Amazon EKS on Fargate

Amazon EKS on Fargate

Bring existing pods

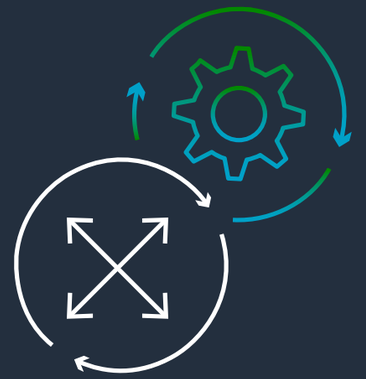
You don’t need to change your existing pods.

Fargate works with existing workflows and services that run on Kubernetes.

Production ready

Launch pods quickly. Easily run pods across multiple AZs for high availability.

Each pod runs in an isolated compute environment.



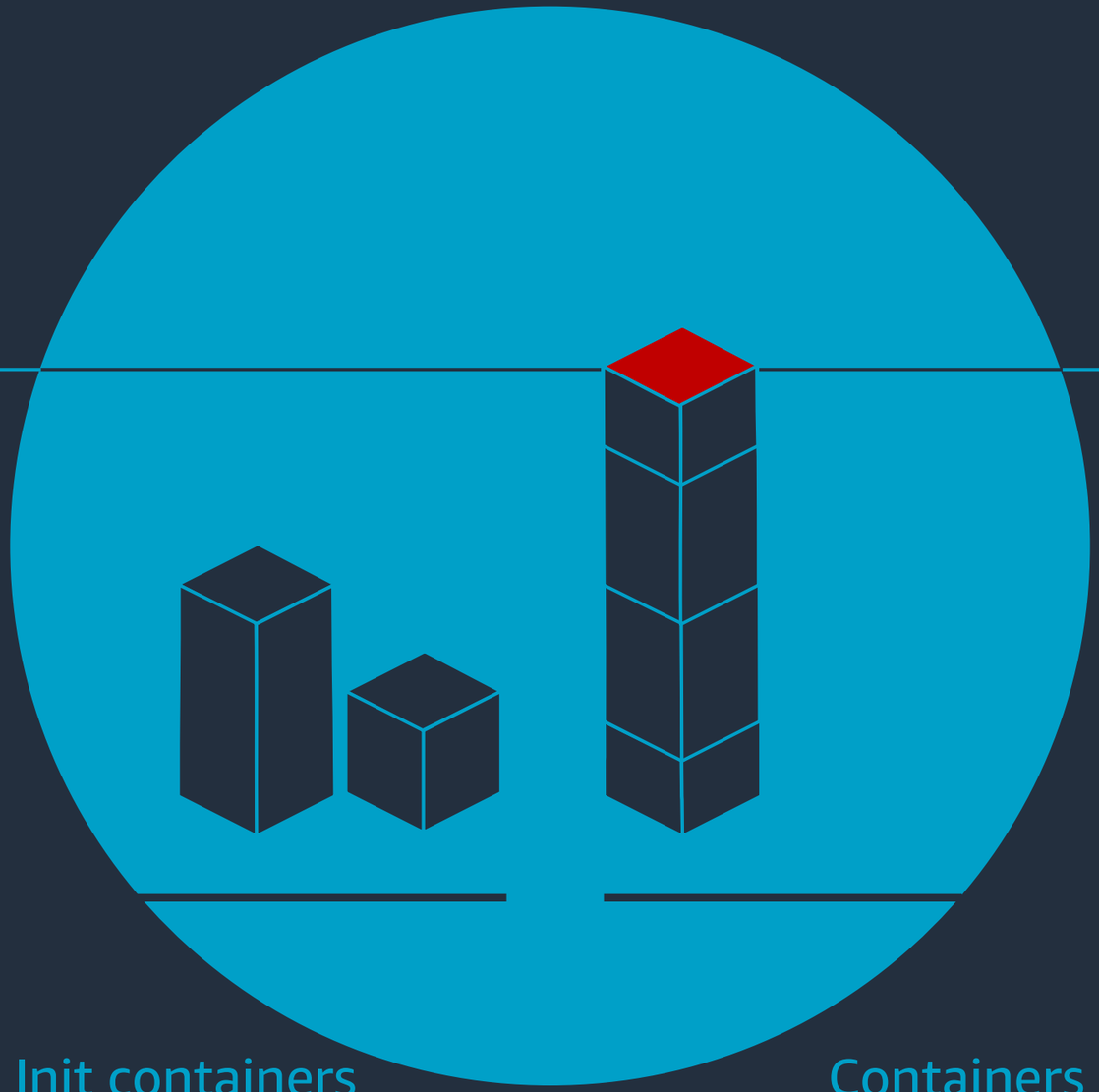
Rightsized and integrated

Only pay for the resources you need to run your pods.

Includes native AWS integrations for networking and security.

How do we pick the size of the pod?

This



Init containers

Start sequentially and then stop

Containers

Long running

Kubernetes components

+256MB

Closest config (rounded up) is picked

CPU

MEM

CPU	Memory
256 (.25 vCPU)	512MB, 1GB, 2GB
512 (.5 vCPU)	1GB, 2GB, 3GB, 4GB
1024 (1 vCPU)	2GB, 3GB, 4GB, 5GB, 6GB, 7GB, 8GB
2048 (2 vCPU)	Between 4GB and 16GB in 1GB increments
4096 (4 vCPU)	Between 8GB and 30GB in 1GB increments

Fargate task size combinations

What is Fargate Spot?

- New deployment option for AWS Fargate
- Offers up to 70% discount compared to ‘regular’ Fargate prices
- Suitable for interruption tolerant workloads
- Simplicity of containers, cost effectiveness of Spot

What are the key benefits of Fargate Spot?

1. Fully managed – no provisioning, patching or diversifying clusters of instances to optimize for pricing option

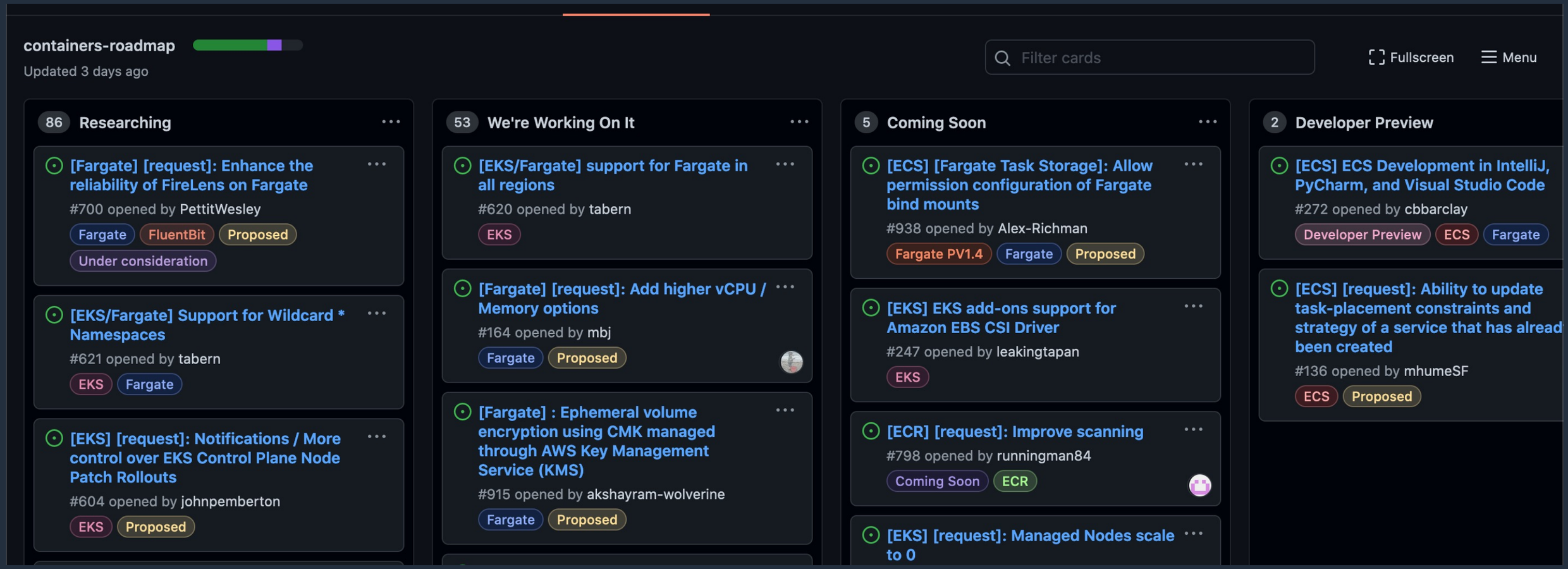
2. Application first controls - service owners can choose the model that suits their service the best

3. Seamless migration – migrate services with little effort

4. Flexible portfolio of pricing options – customers can choose from regular, Savings Plan or Spot pricing options

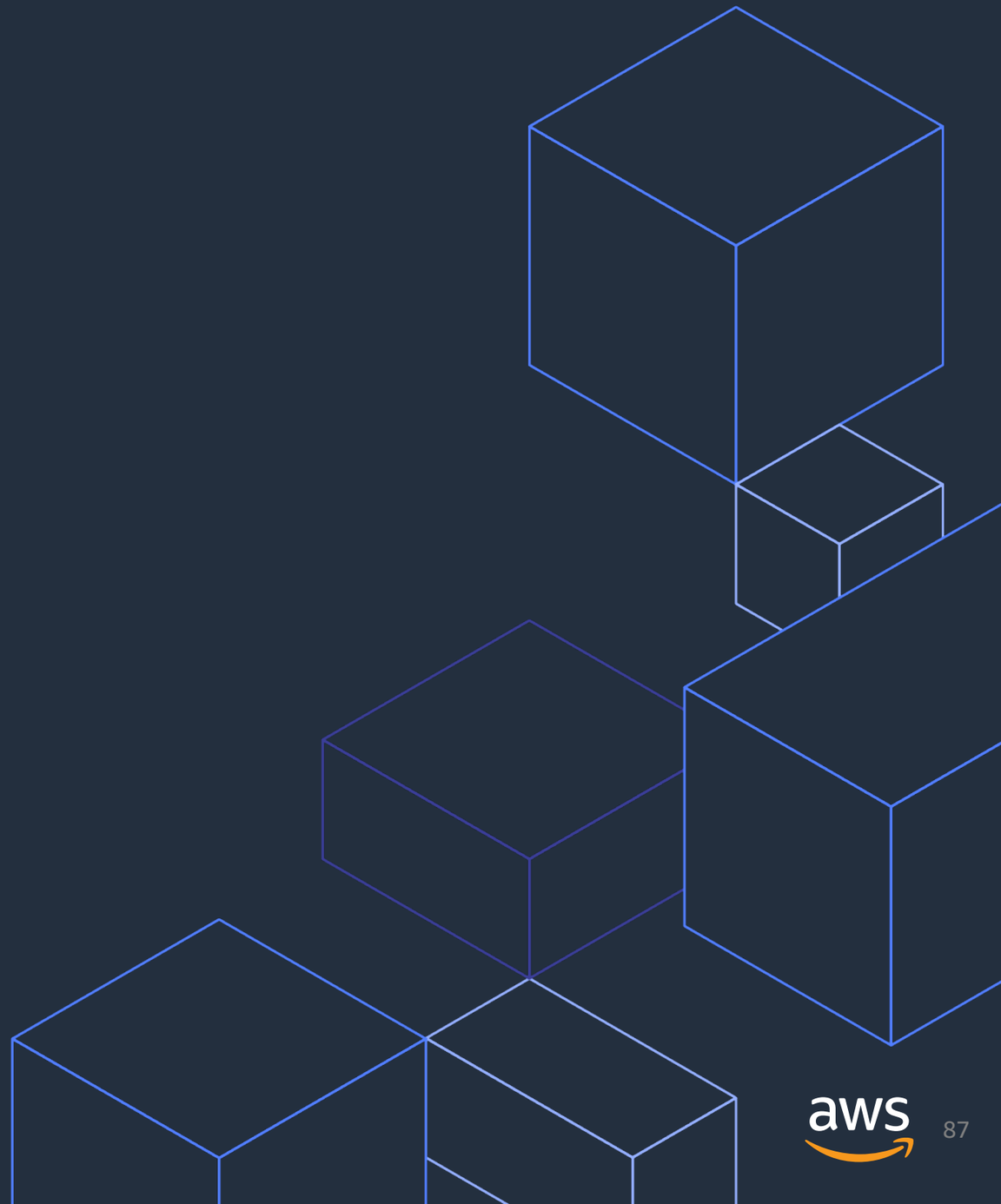
Amazon EKS Public Roadmap

Public Roadmaps



https://github.com/aws/containers-roadmap/projects/1

k8s Best Practices



Namespace Designing

- A good idea to avoid using kube-system namespace
- Kubernetes uses “default” namespace by default and you cannot delete it
- You can create multiple namespaces. But the resources you create lands in the active namespace

  Tip: kubens
- A service in one namespace can speak to service in a different namespace

  <Service Name>.<Namespace Name>.svc.cluster.local

Namespace Design Strategy

- When you have a small team, try to create multiple namespaces for better isolation between team

- If you team is rapidly growing, 10+ micro services, try to run a Dev and Prod Cluster

- When you have a large production system, try to have multiple clusters, multiple namespaces and lockdown namespaces for the teams that does not need access. Helps in management, reducing the blast radius and understanding billing costs

Amazon EKS worker node provisioning with Amazon EC2 Spot

- Recommend using the node labels to identify Amazon EC2 Spot Instances
- Launch Amazon EC2 Spot Instances as part of Auto Scaling group
- Use Amazon EC2 Spot Instances best practice of mixed instance types

Health Checks

Readiness and liveness probes can help maintain the health of applications running inside Kubernetes. By default, Kubernetes only knows whether or not a process is running, not if it's healthy.

Readiness probes are designed to ensure that an application has reached a "ready" state. In many cases there is a period of time between when a webserver process starts and when it is ready to receive traffic. A readiness probe can ensure the traffic is not sent to a pod until it is actually ready to receive traffic.

Liveness probes are designed to ensure that an application stays in a healthy state. When a liveness probe fails, the pod will be restarted.

Optimizing your container

Optimize for smaller size, use a multistage Docker build to reduce the size of the runtime container.

Use a minimalist operating system: Alpine Linux, or similar.

Popular base images have a huge range by size:

node:latest 674MB | node:slim 184MB
java:latest 643MB
ubuntu:latest 85.8MB | alpine:latest 4.41MB | busybox:latest 1.15MB

Configuring resource requests and limits

Setting appropriate resource requests will ensure that all your applications have sufficient compute resources.

Setting appropriate resource limits will ensure that your applications do not consume too many resources.

Admission controllers make it easy to add a lot of sidecars but don’t underestimate the overhead cost.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: app
spec:
  containers:
  - name: app
    image: nathanpeck/app
    resources:
      requests:
        memory: "256Mi"
        cpu: "250m"
      limits:
        memory: "512Mi"
        cpu: "500m"
```

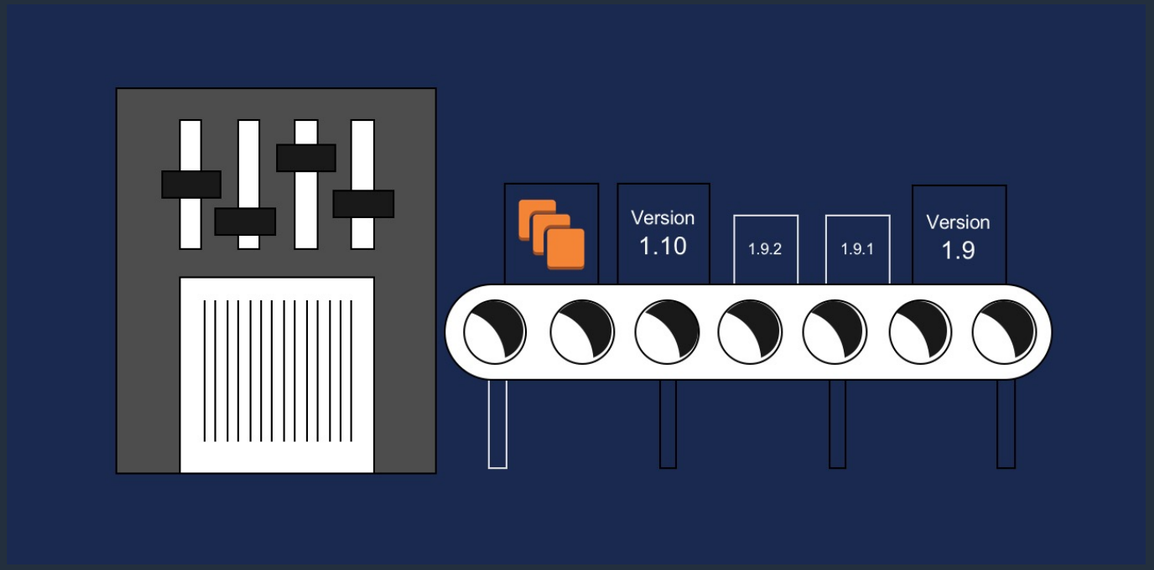
When not configured / not correctly configured

Each container has sufficient access to compute resources. Without resource requests, a pod may be scheduled on a node that is already overutilized.

Without resource limits, a single poorly behaving pod could utilize the majority of resources on a node, significantly impacting the performance of other pods on the same node.

Having these values appropriately configured ensures that Cluster autoscaling can function as intended. New nodes are scheduled once pods are unable to be scheduled on an existing node due to insufficient resources. This cannot happen if the resource requests are not configured.

Kubernetes Upgrades



- We'll make sure customers have the latest security patches on your cluster, while giving customers a choice of minor versions to run.

Migration between versions

Check if the version can be upgraded
https://kubernetes.io/docs/setup/release/version-skew-policy/

Test in a Dev environment if your application is working as expected in the new version

Design a migration strategy

- With minimal downtime
- With no downtime

EKS Lifecycle

https://aws.amazon.com/blogs/compute/updates-to-amazon-eks-version-lifecycle/

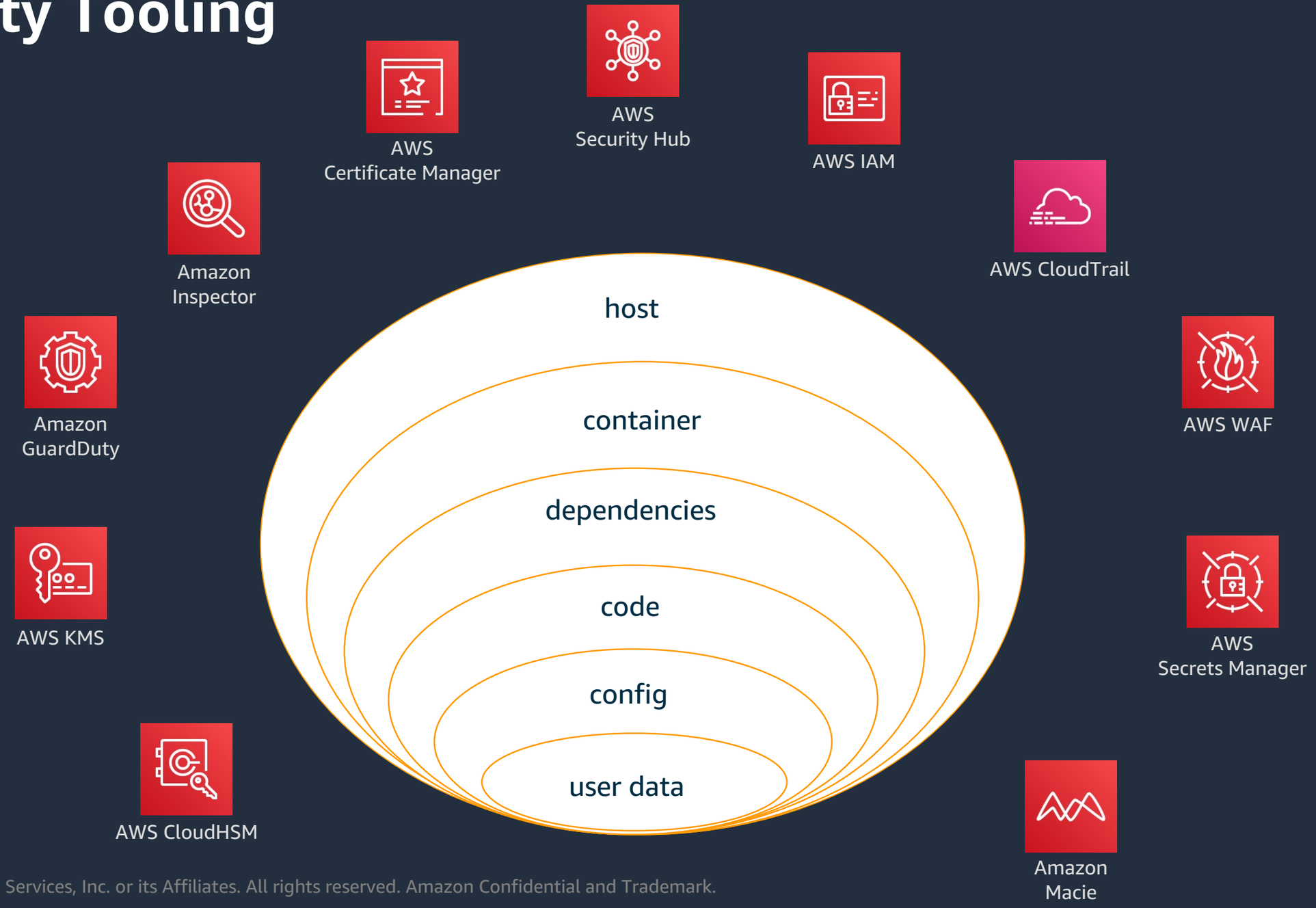
Upgrading EKS cluster using eksctl

http://arun-gupta.github.io/update-eks-cluster/

Upgrading EKS Cluster:

https://docs.aws.amazon.com/eks/latest/userguide/update-cluster.html

Security Tooling



Container and Security Best Practices

Use trusted base images
Use non-root user inside container
Make the file system read only
One process per container
Log to stdout & stdderr
Ensure That Images Are Free of Vulnerabilities
Perform automated CVE scans (on push and/or periodically)
Use private registries (ECR, for example)
Ensure That Only Authorized Images are used in your environment
Limit Direct Access to Kubernetes Nodes
Create Administrative Boundaries between Resources

Container Best Practices

Try to aim for reproducible builds (pin dependencies)

Define Resource Quota

Define liveness and readiness probes, always

Implement Network Segmentation

Apply Security Context to Your Pods and Containers

Integrate Security into your CI/CD pipeline

Regularly Apply Security Updates to Your Environment

Make sure you only push approved images to these registries

https://kubernetes-security.info/
https://kubernetes.io/docs/tasks/configure-pod-container/security-context/