Amazon EKS Immersion Day

Running Kubernetes Workloads on AWS

Clément Goillot
Solutions Architect

Martin-Zack Mekkaoui
Technical Account Manager

Benjamin Guerin
Solutions Architect

Agenda

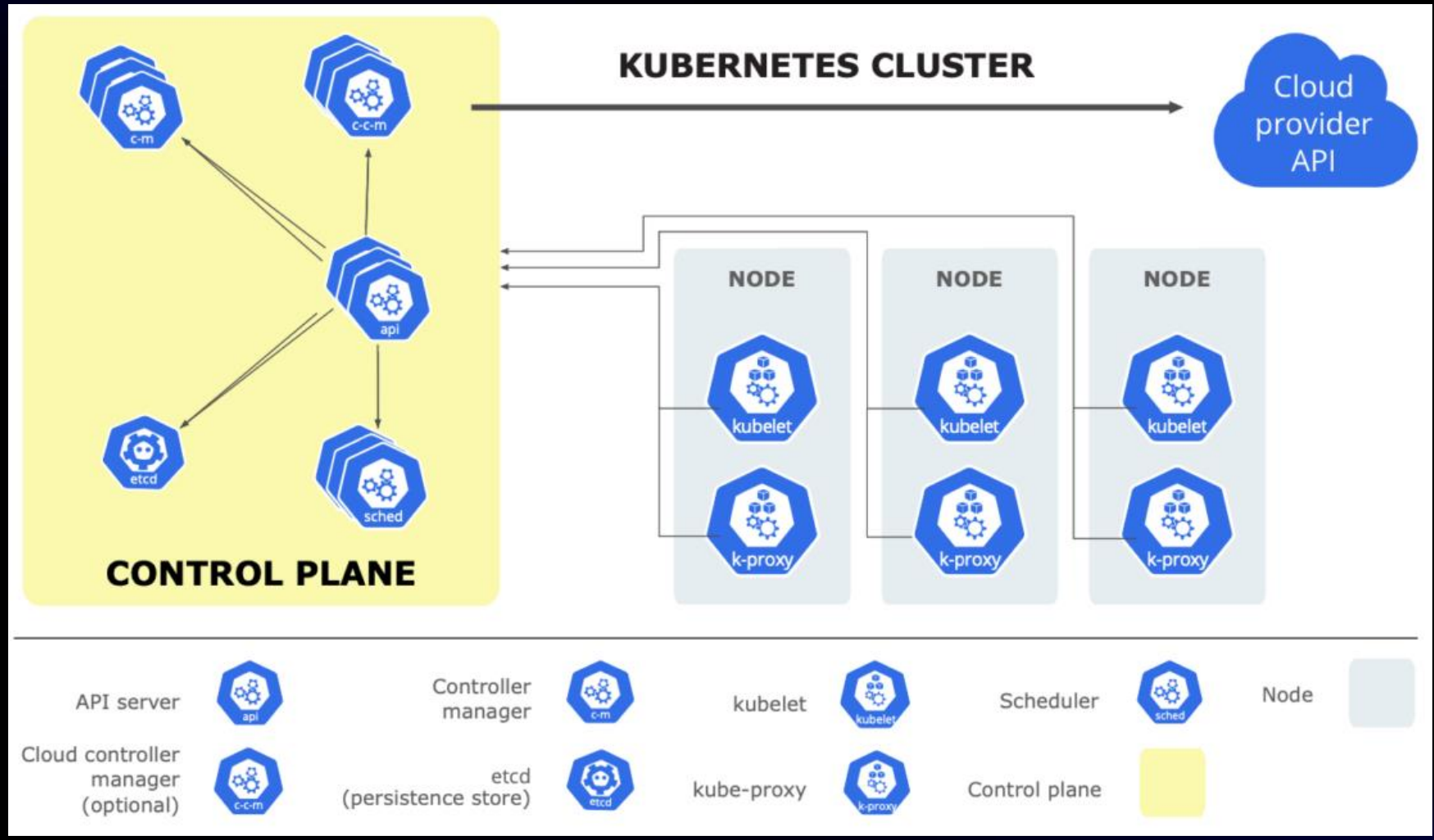
Time	Topic
09H30 - 10H30	Accueil
10H30 - 11H00	Présentation : Introduction à Amazon Elastic Kubernetes Service (EKS)
11H00 - 12H00	Mise en pratique : Environnement de Travail Cloud9 & Découverte du Cluster EKS
12H00 - 12H45	Présentation : Control Plane, Data Plane & Load Balancing
12H45 - 13H45	Déjeuner
13H45 - 14H45	Mise en pratique : Managed Node Groups, Fargate, Load Balancers, Storage
14H45 - 15H15	Présentation : Scalabilité sur EKS
15H15 - 16H00	Mise en pratique : Scalabilité des Pods et du Cluster
16H00 - 16H15	Pause
16H15 - 16H45	Présentation : EKS Security
16H45 - 17H15	Mise en pratique : Gestion des Permissions des Pods avec IAM Roles for Service Accounts (IRSA)
17H15 - 18H00	Open Labs
18h00 - 18h30	Résumé & Conclusion

Introduction

Kubernetes

- Started at Google
- Influenced by Google Borg
- Container Orchestrator
- Contributed to the CNCF

Kubernetes Concepts



https://kubernetes.io/docs/concepts/overview/components/

Kubernetes Concepts

Pods: Co-located group of containers that share an IP, namespace, storage volume

Label: Used to attached metadata to k8s objects. Can be used to select group of objects

Deployment: Manages the lifecycle of pods and ensures specified number are running

Service: Exposes an application running in your cluster behind a single outward-facing endpoint

Ingress: Makes your HTTP(S) network service available using a protocol-aware configuration mechanism, that understands web concepts like URIs, hostnames, paths, and more.

Namespace: In Kubernetes, namespaces provides a mechanism for isolating groups of resources within a single cluster. Names of resources need to be unique within a namespace, but not across namespaces.

DaemonSet: **E**nsures that all (or some) Nodes run a copy of a Pod. As nodes are added to the cluster, Pods are added to them.

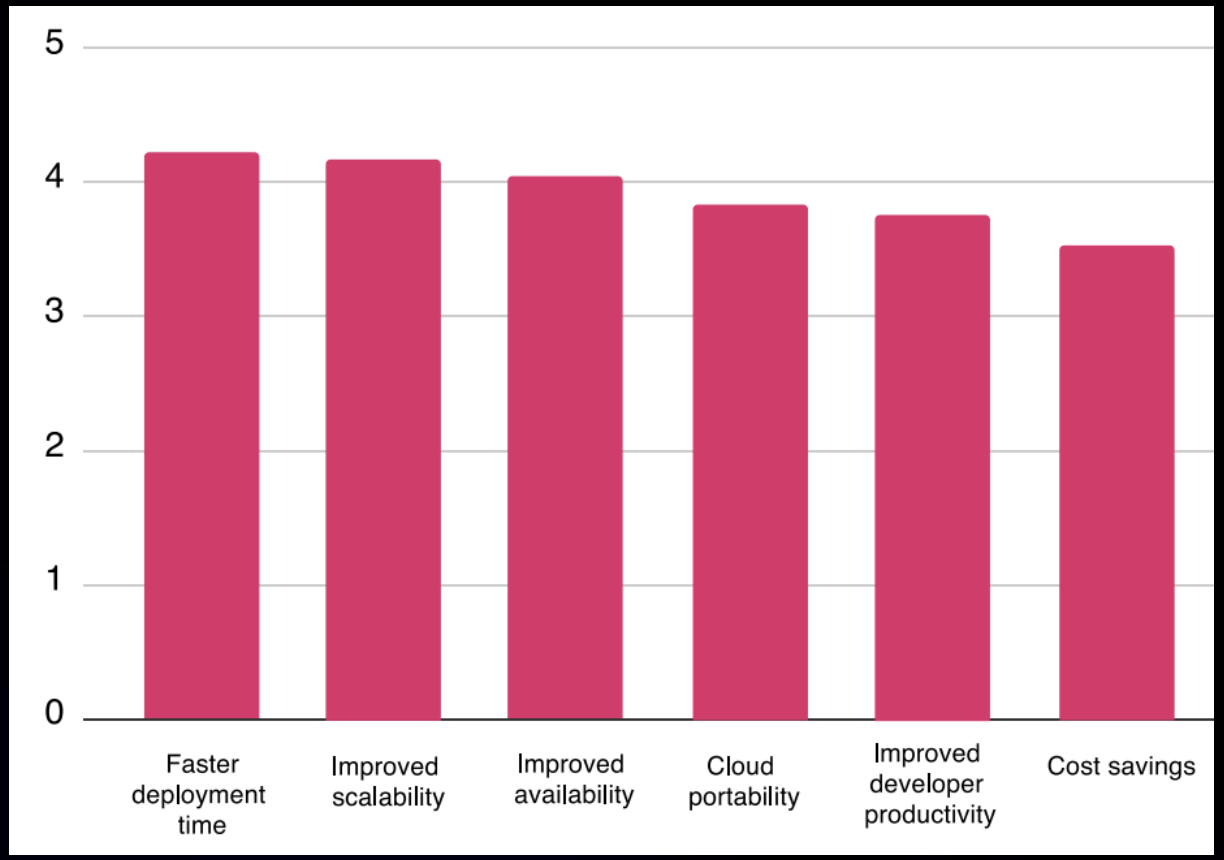
StatefulSet: Manages the deployment and scaling of a set of Pods, and provides guarantees about the ordering and uniqueness of these Pods.

Why Kubernetes?

Open-source container management platform governed by the community

Run containers at scale, integrating networking, storage, and compute

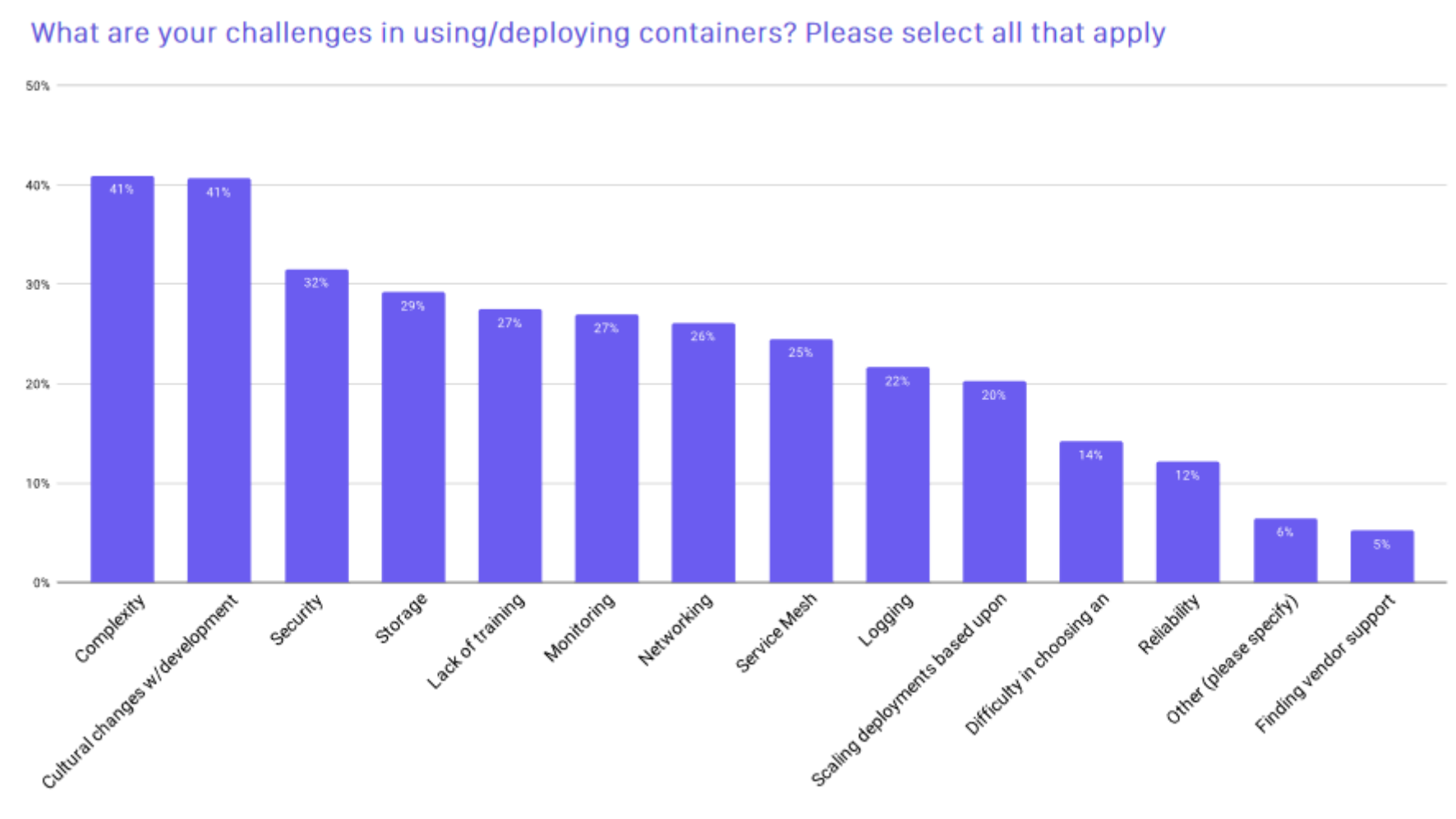
Take advantage of open-source tools that are designed to run on Kubernetes



Source: CNCF Survey – Rate Kubernetes benefits 1 to 5

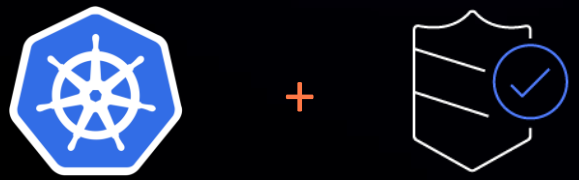
aws

Kubernetes Challenges



https://www.cncf.io/wp-content/uploads/2020/11/CNCF_Survey_Report_2020.pdf

AWS is committed to supporting the Kubernetes community regardless of how customers choose to run Kubernetes



Security is job zero at AWS and that includes Kubernetes. **AWS is a member of the Product Security Committee**, who are responsible for the security the Kubernetes project.

- kubernetes/cloud-provider-aws
- kubernetes/autoscaler
- kubernetes-sigs/aws-ebs-csi-driver
- kubernetes-sigs/aws-efs-csi-driver
- kubernetes-sigs/aws-fsx-csi-driver
- kubernetes-sigs/aws-iam-authenticator
- kubernetes-sigs/aws-load-balancer-controller
- kubernetes-sigs/aws-encryption-provider
- kubernetes-sigs/aws-encryption-provider
- awslabs/karpenter
- aws/eks-distro

CNI, CSI, Kubernetes, and many more…

aws

What is Amazon EKS?

Amazon **EKS**

- EKS runs vanilla Kubernetes; EKS is upstream and a certified conformant version of Kubernetes (with backported security fixes)
- EKS supports 4 versions of Kubernetes, giving customers time to test and roll out upgrades
- EKS provides a managed Kubernetes experience for performant, reliable, and secure Kubernetes
- EKS makes Kubernetes operations, administration, and management simple

Amazon EKS helps you build reliable, stable, and secure applications in virtually any environment

aws

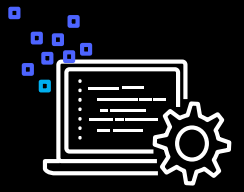
Amazon EKS Core tenets

Security first

Built for production

Native and upstream

Seamless cloud integrations



Committed to open source

Focus on strategic operations

Strategic Operations

Product Releases

Application Security

Organizational Structures

Software Supply Chain

Customer Connectivity

Tactical Operations

Kubernetes Builds

Scaling + Availability

Infrastructure Reliability

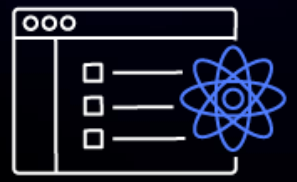
Authentication + Security

Backups and Resiliency

Amazon EKS is used in every industry, around the world

aws

Amazon EKS enables customers across industries and application types



Web Applications

Data Processing

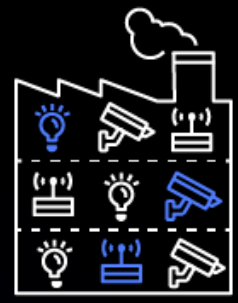
Machine Learning

CI/CD

Mobile Applications

Gaming Platforms

Platform as a Service (PaaS)



Internet of Things

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

AWS ISV Partners for Containers

Foundation

docker
HashiCorp
D2 IQ
VMware Tanzu
CANONICAL
spot
SUSE

Monitoring + Logging

DATADOG
dynatrace
epsagon
splunk>
sumo logic
New Relic.

DevOps + CI/CD

ATLASSIAN
cloudbees
CODESHIP by CloudBees
GitLab
pulumi
Travis CI
XebiaLabs Enterprise DevOps
weaveworks

Security + Networking

alcide
ALERT LOGIC
aqua
paloalto NETWORKS
StackRox
sysdig
threat stack
TIGERA
TREND MICRO

Amazon EKS Compliance

https://aws.amazon.com/compliance/services-in-scope/

Payment Card Industry (PCI) Security Standard

FedRAMP Moderate, High Q1 2021

DOD Cloud Security Req's Guide (SRG) – On the Roadmap

Criminal Justice Information Service Security Policy (CJIS)

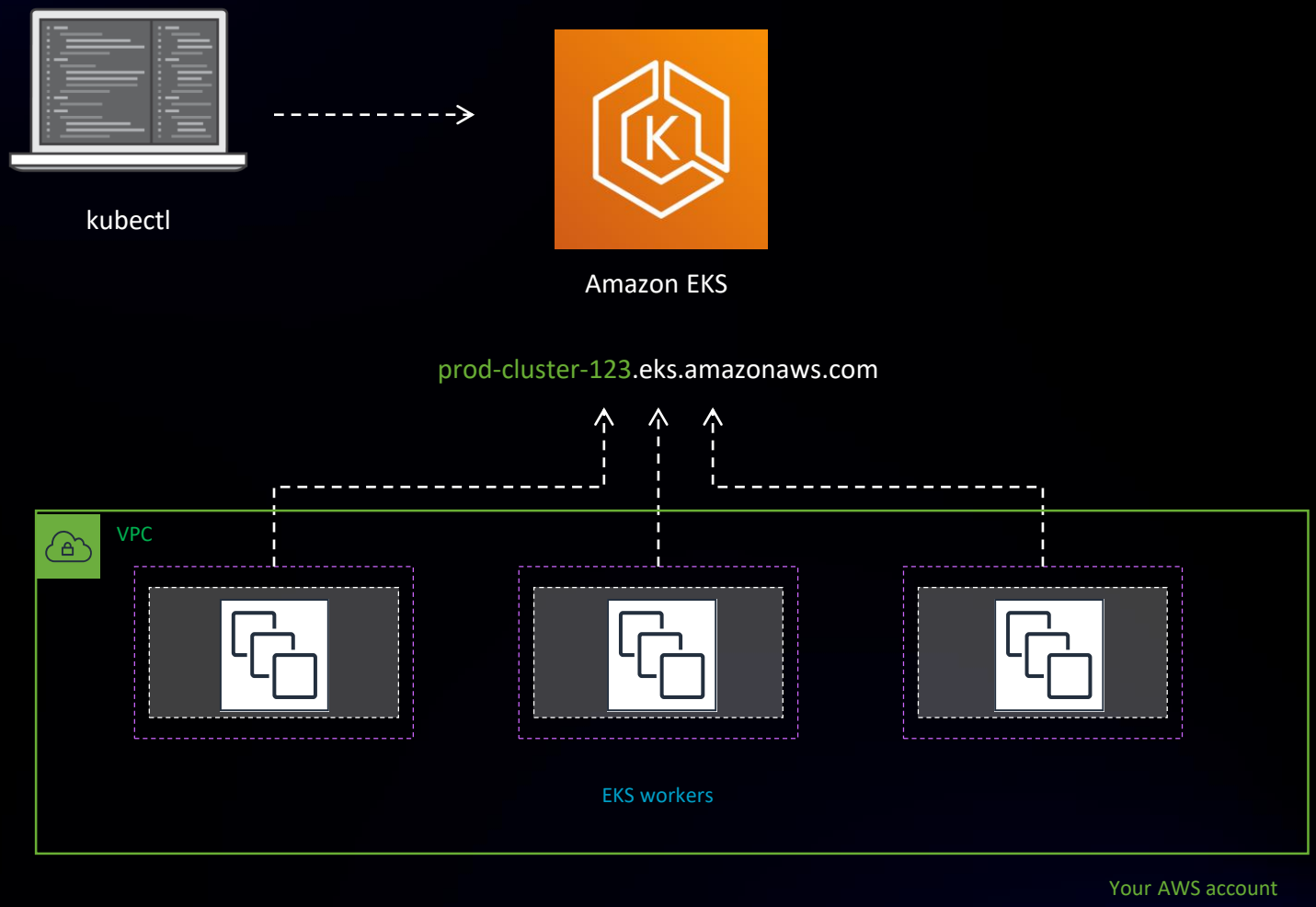
U.S. Health Insurance Portability and Accountability Act (HIPAA)

Federal Information Processing Standard Pub (FIPS) 140-2

SP 800-53 (rev 4)
SP 800-171

Health Information Trust Alliance Common Security Framework (HITRUST CSF)

Amazon EKS Architecture




NOW GO BUILD
CAPE TOWN

Control Plane

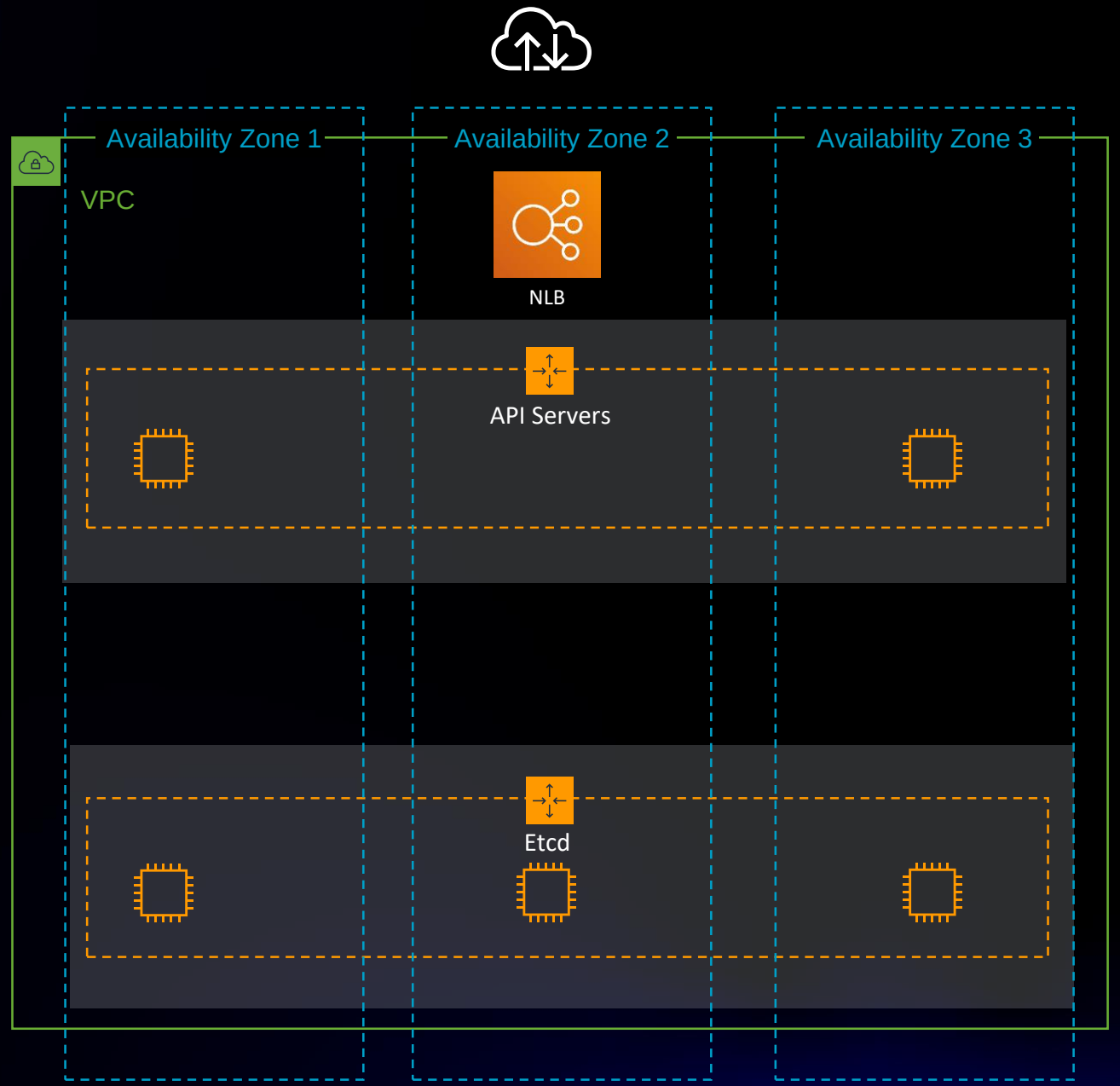
EKS Control Plane

Highly available and single tenant infrastructure

All “native AWS” components

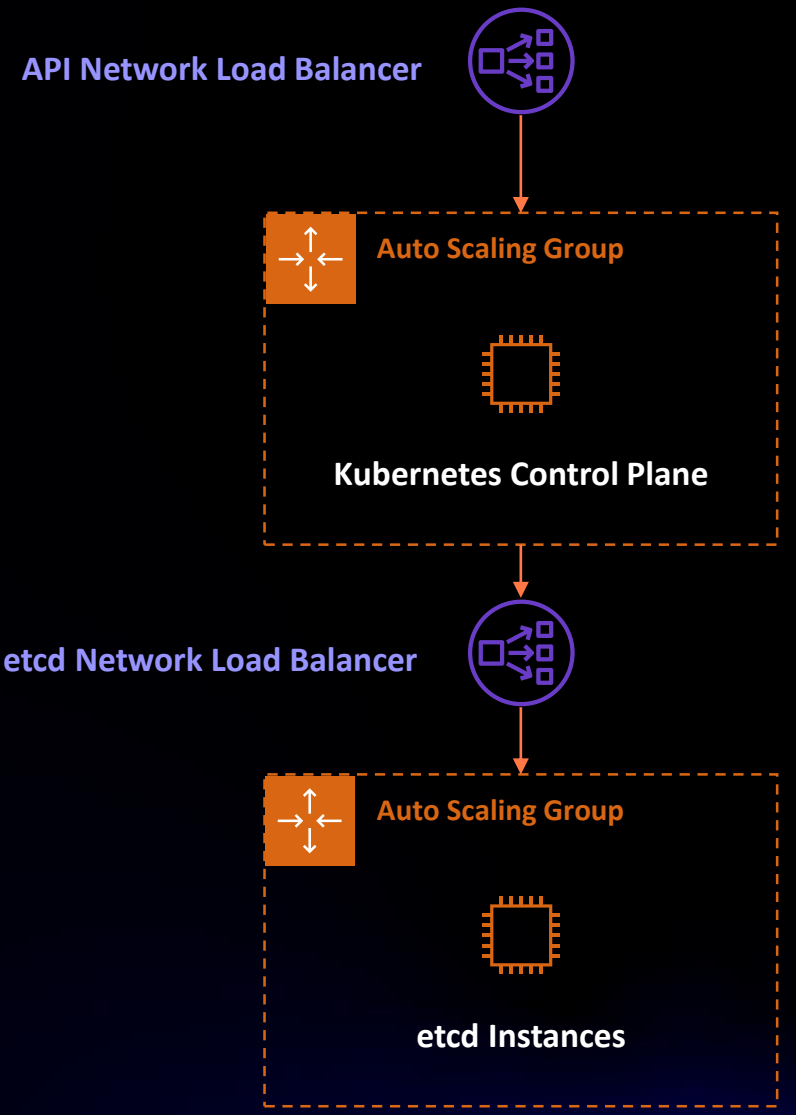
Separate etcd to help with safe and seamless ops/upgrades

Fronted by an NLB - which enables Private endpoints into your VPC



Amazon EKS Control Plane Architecture

- Amazon EKS is architected across multiple availability zones for high availability.
- The Amazon EKS control plane is built using native services surrounded by operational tooling.
- Each cluster has it’s own control plane infrastructure, meaning each cluster is single tenant.
- Private IPs within the customer VPC enable private communication to the control plane.



Cluster Autotuning

Automatic Resizing

EKS will automatically change the provisioned infrastructure to support larger clusters and higher demand on the API server.

Consistent performance

Go from 1 to 1,000+ nodes without latency

No extra cost

There are no additional EKS costs to run larger clusters.

EKS Supported Kubernetes Versions

- Amazon EKS is commited to supporting at least **4 production-ready versions** of Kubernetes at any given time
- A Kubernetes version is **fully supported for 14 months** after being avalaible on Amazon EKS, even if upstream Kubernetes is no longer supporting a version available on EKS (we **backport security patches** applicable **to our supported versions**)
- Minor versions controlled by customers
    - 1.23, 1.24, 1.25, 1.26 currently available;
    - 1.22 end of support on June 4, 2022.
- Patch versions automatically applied to control plane
- Platform Version defines Kubernetes version and other key control plane capabilities

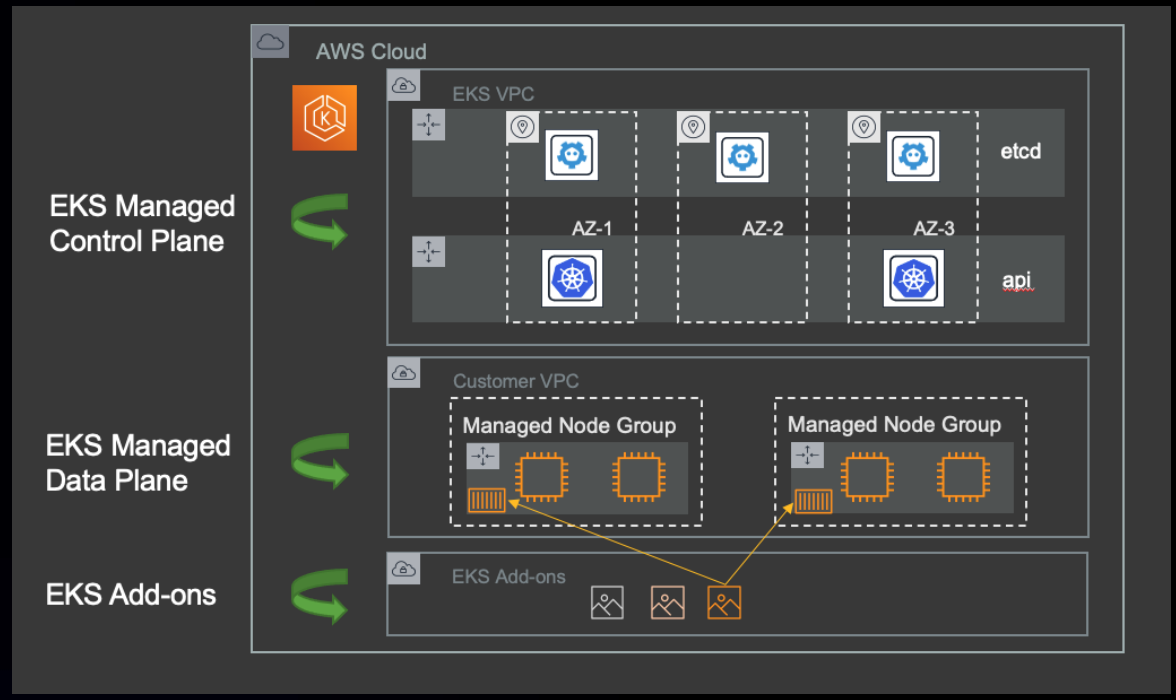
https://docs.aws.amazon.com/eks/latest/userguide/kubernetes-versions.html

aws

How do I upgrade my cluster?

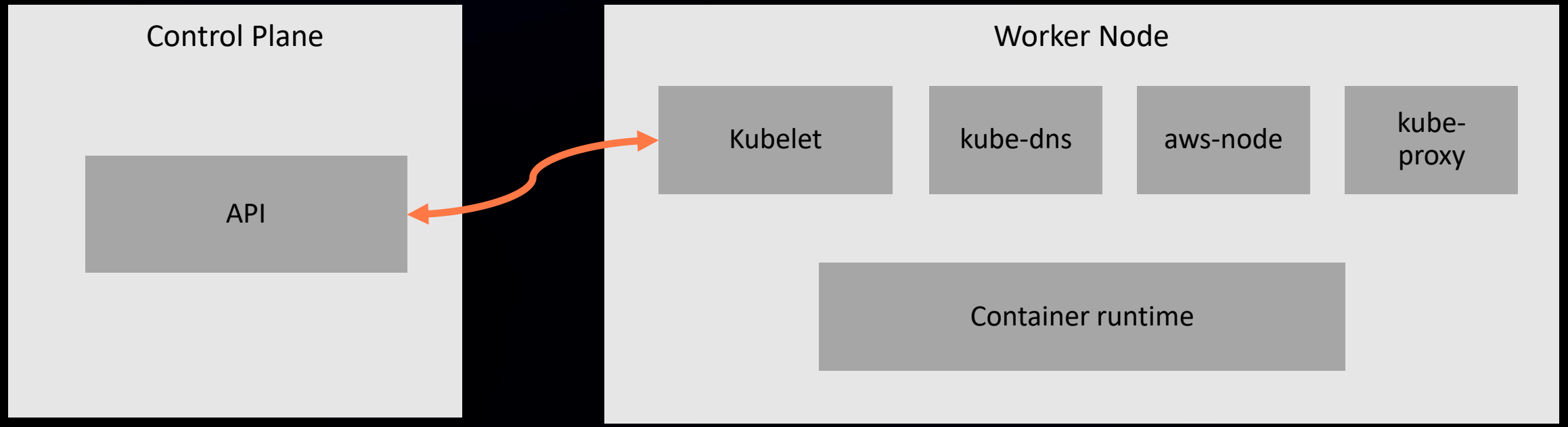
WHICH VERSION OF KUBERNETES DO I USE?

- Amazon EKS aims to be ~100–150 days behind upstream release
- You must plan for version upgrade at least yearly, ideally 2x-3x/yr
- Use EKS managed capabilities to simplify upgrades
- Run upgrade process in test env. first
- Check EKS and Kubernetes release notes
- Test your application manifests against deprecated/removed APIs

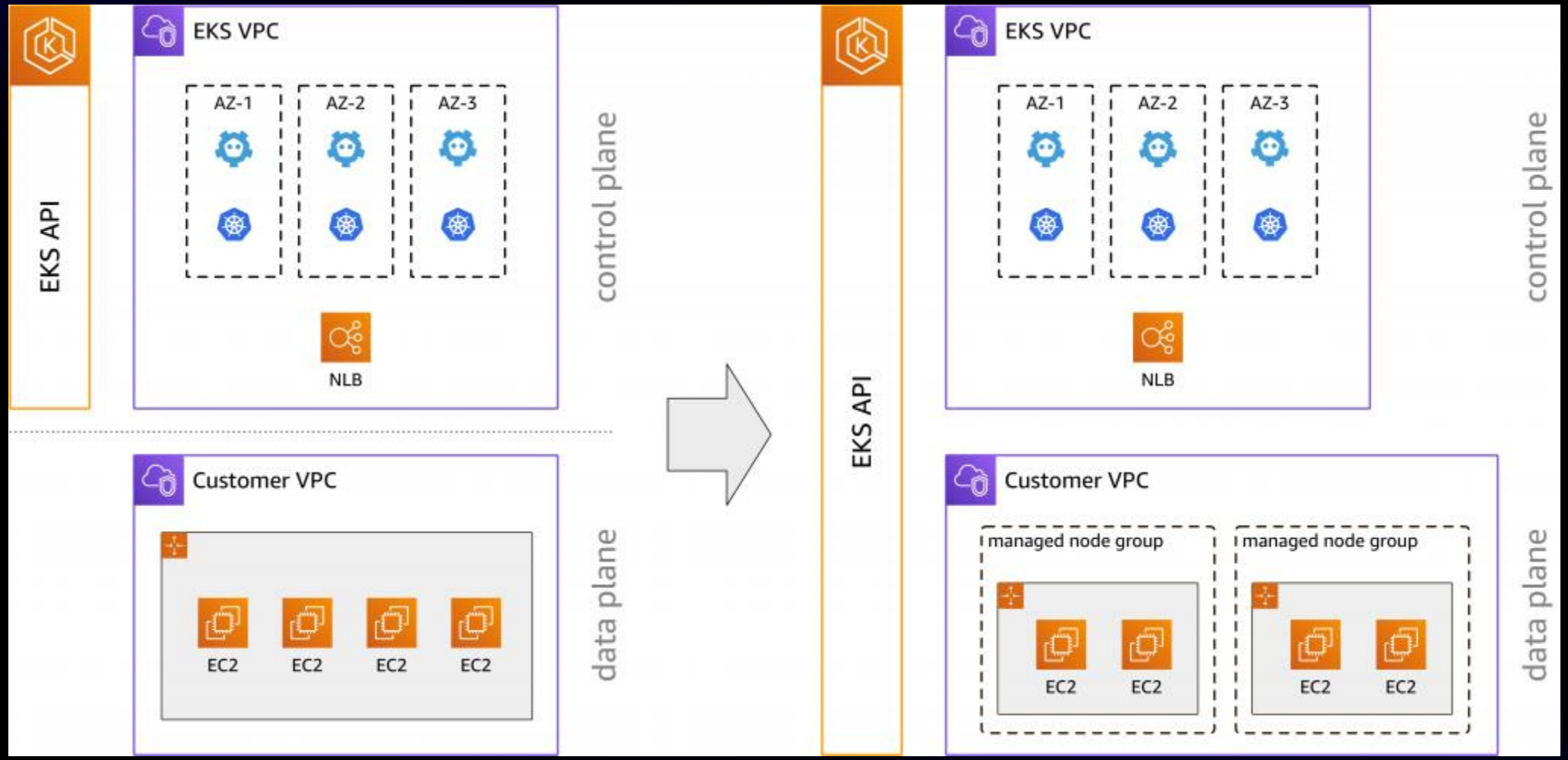


Data Plane

Kubernetes Data Plane

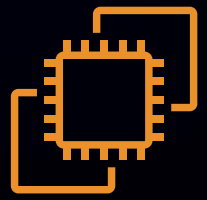


Managed Node Group Introduction



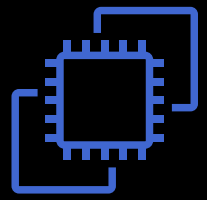
https://aws.amazon.com/blogs/containers/eks-managed-node-groups/

Amazon EC2



Self-Managed Node Groups

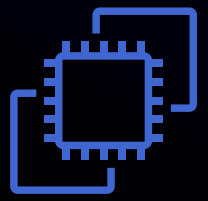
Bring your own Autoscaling Groups running your own custom AMI. You are responsible for patching and the underlying OS.



Managed Node Groups

Provisioned by EKS in your VPC. They run the latest EKS optimized AMI by default. You are responsible to update them once a new AMI is available. Handles automatic draining and rolling out new AMIs.

Managed Node Groups for Amazon EKS



Managed Node Groups

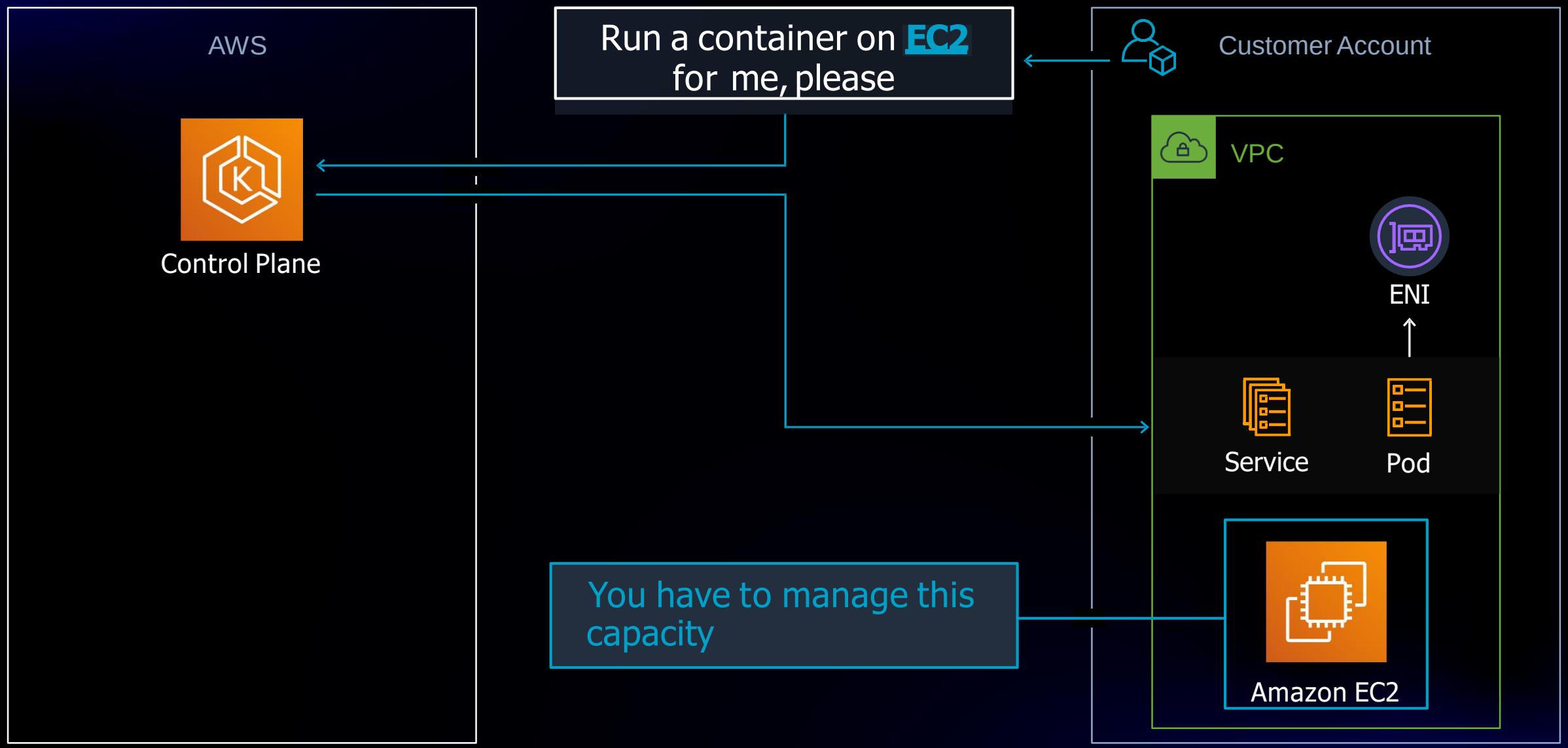
- Creates and associates Auto Scaling Group with the Cluster
- Use the latest EKS Optimized AMI (Amazon Linux 2, Ubuntu 18.04 LTS, or Custom AMI)
- Monitors for health issues and provides errors/alerts via the Console or Cloudwatch
- Automatically drains nodes during termination using the K8s API
- Supports rolling updates of AMIs and respects Pod Disruption Budget where possible

Update Process

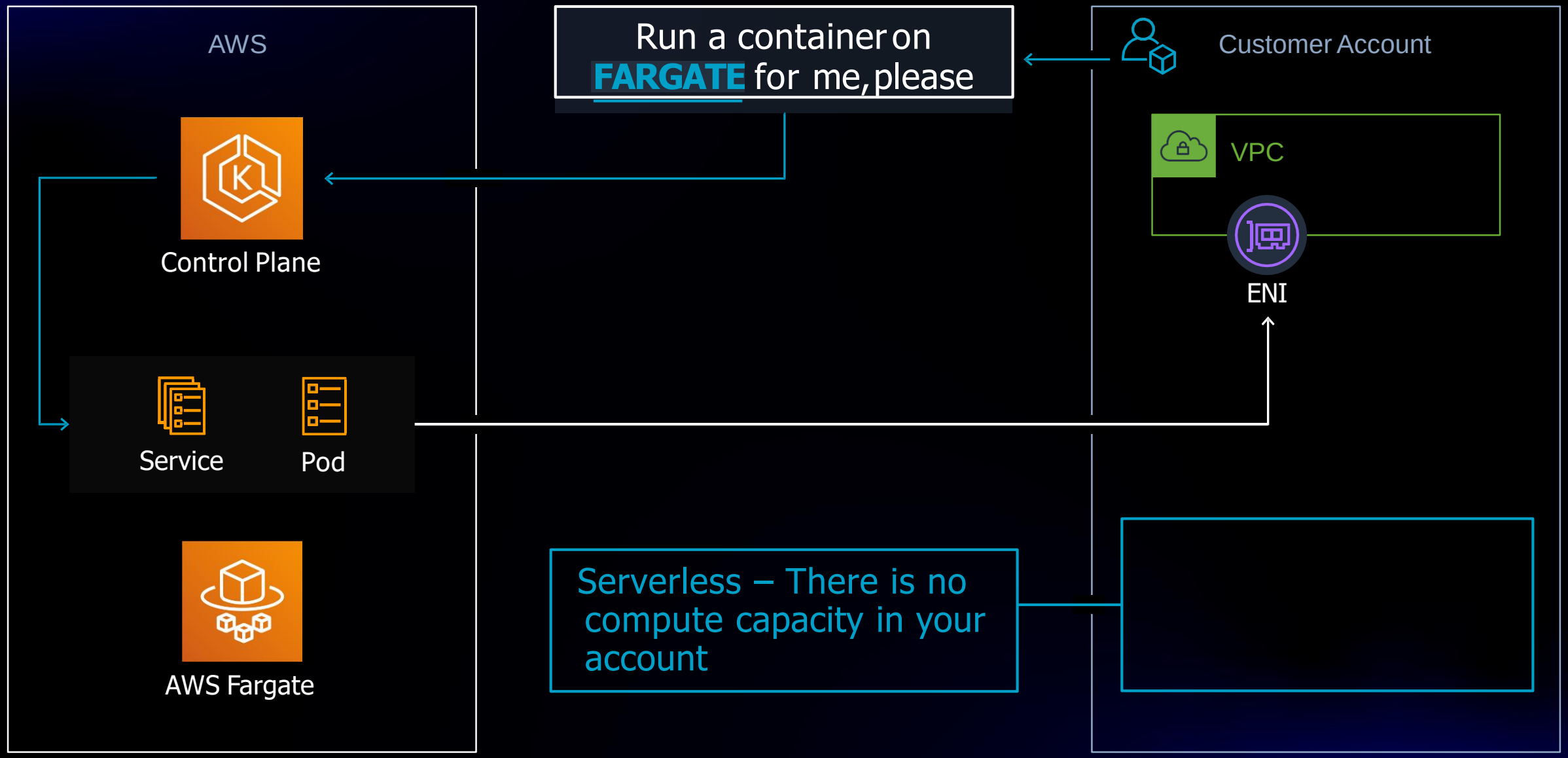
1. Creates A New Launch Template Version & Updates ASG
2. Bumps Auto Scaling Group By 1 (up to 2x #Azs)
3. Taints all old AMI nodes with *NoSchedule*.
4. Drains a random Node with old AMI, cordons it after all pods evicted
5. Send termination request to ASG for the cordonded node
6. Resets Auto Scaling Group to previous value

Repeat wile there are Nodes with previous AMI

EKS with EC2



EKS with Fargate



Using Fargate with EKS

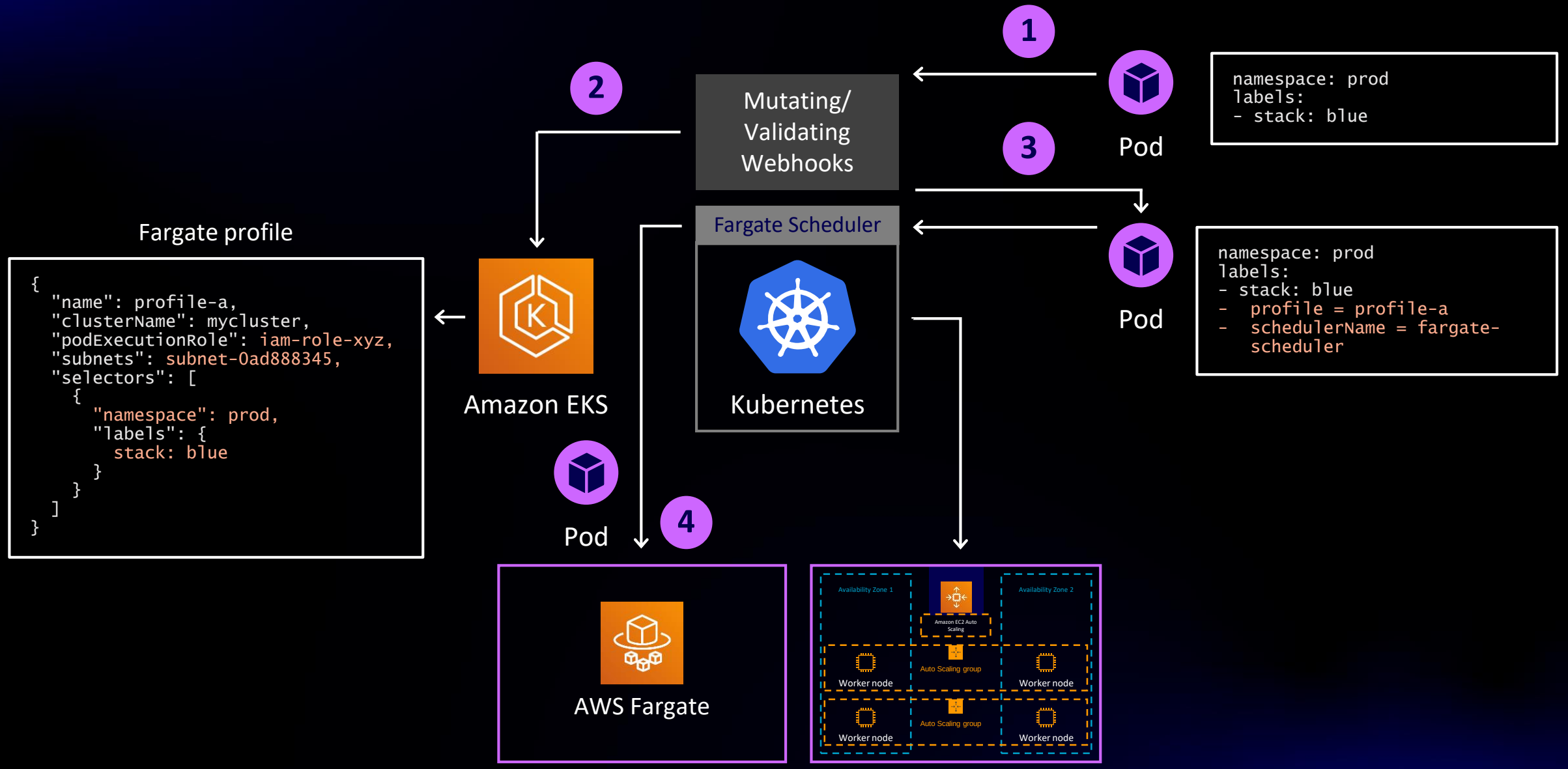
Fargate profile is your interface to configure Fargate on an EKS cluster

- EKS API that lets you define what pods should run on Fargate
- Other configurations such as subnets and IAM roles

Fargate scheduler

- Under the hood implementation
- Runs alongside the standard Kubernetes scheduler

Simplified deployment flow



Value Proposition

Applications not infrastructure

- Use the pod spec and don’t worry about nodes

Fargate Considerations

No cluster scaling

- We scale the clusters, you scale the applications

No host container runtime patching

- We patch the hosts, you focus on containers

Pod isolation

- Secure by default. Containers are isolated in their own VM

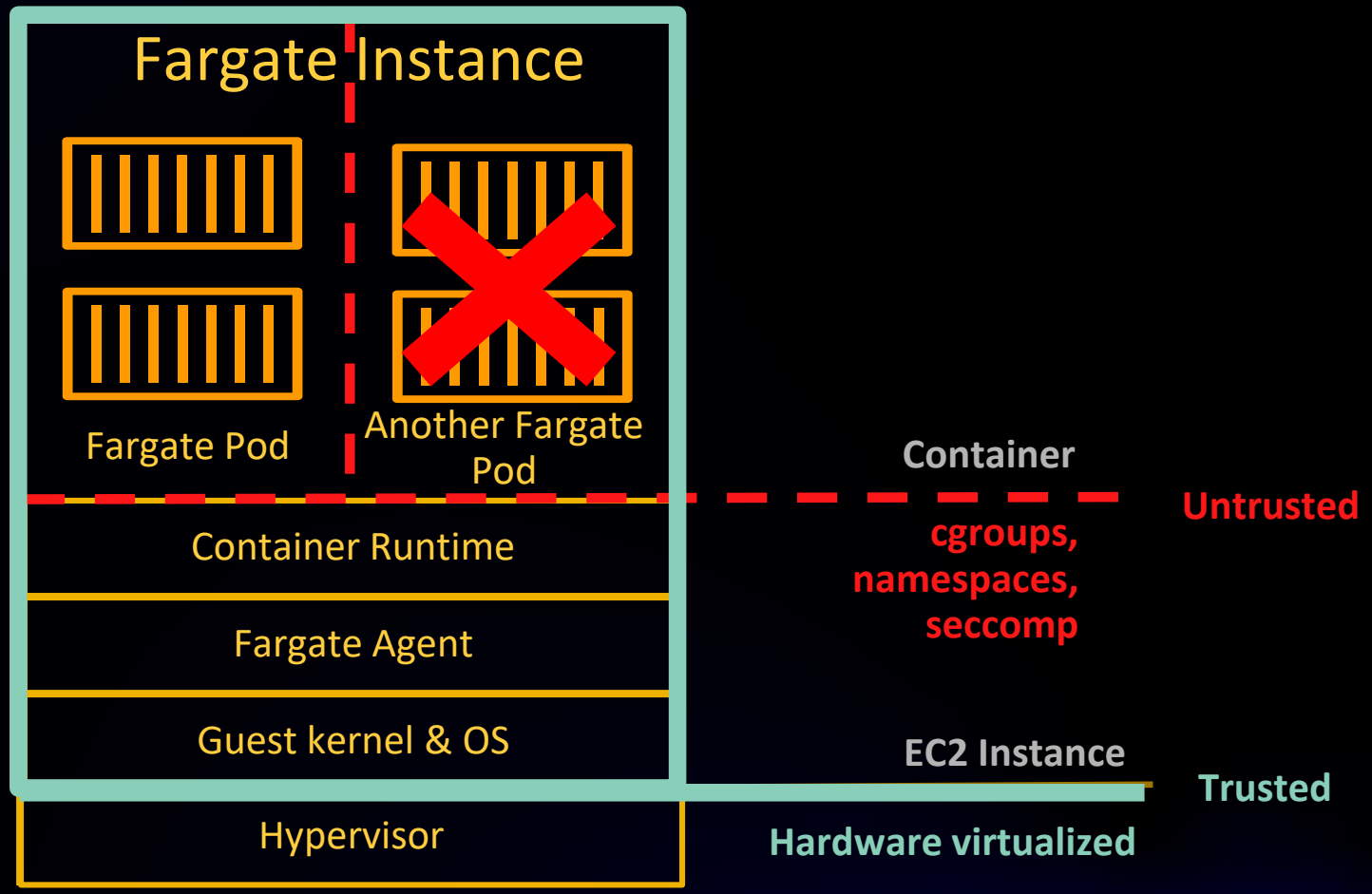
Streamlined updates

- Update cluster and simply relaunch pods

aws

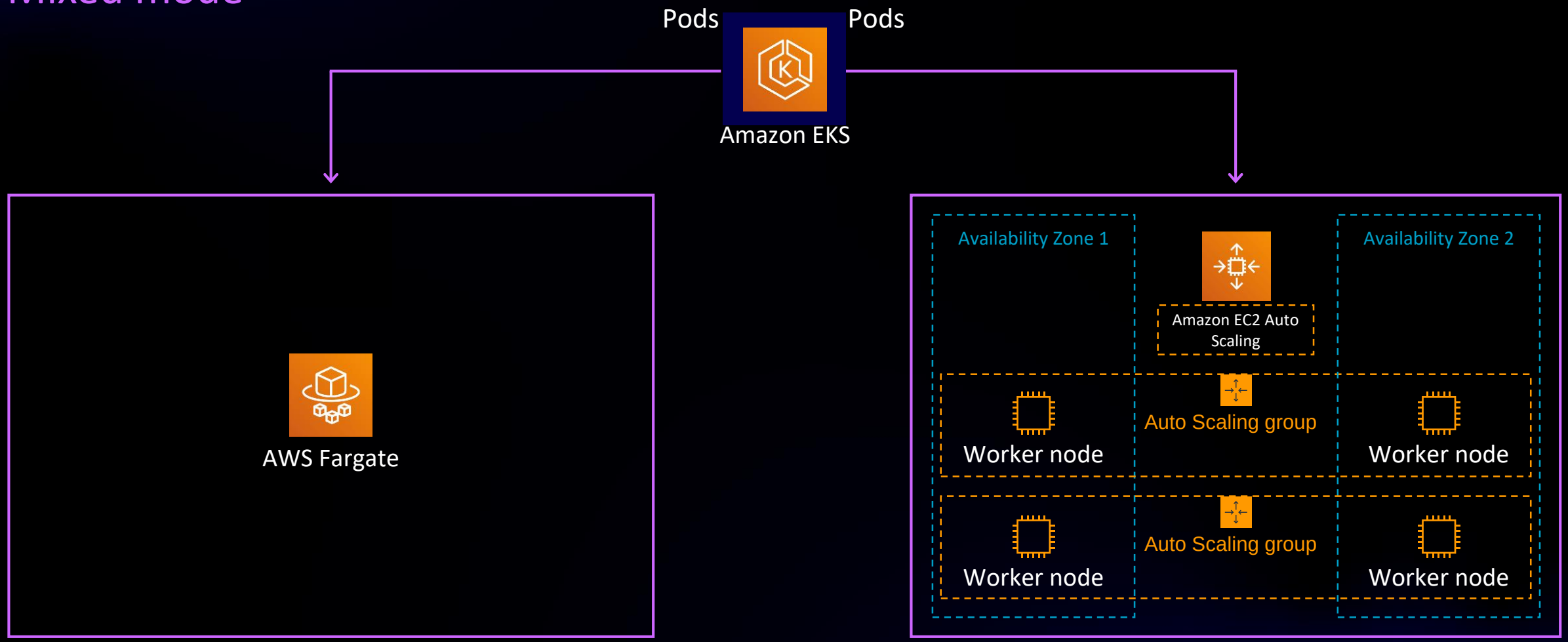
Fargate Security Model

One & only one pod per EC2 instance



EKS data plane options

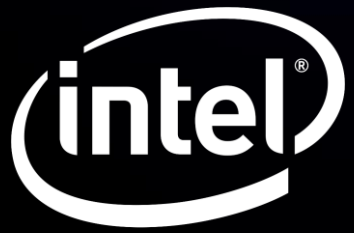
Mixed mode



Serverless data plane

EC2 data plane

Broadest choice of processors



Intel® Xeon Scalable processors

AMD EPYC processors

AWS Graviton processors

AWS Container Storage Interfaces (CSI)

Amazon EBS

Attach Amazon Elastic Block Store (EBS) volumes to containers for persistent data storage.

kubernetes-sigs/aws-ebs-csi-driver

Amazon EFS

Attach Amazon Elastic File System (EFS) file systems to multiple containers across availability zones.

kubernetes-sigs/aws-efs-csi-driver

Amazon FSx for Lustre

Attach Amazon FSx for Lustre posix compliant file systems for machine learning and big data containers.

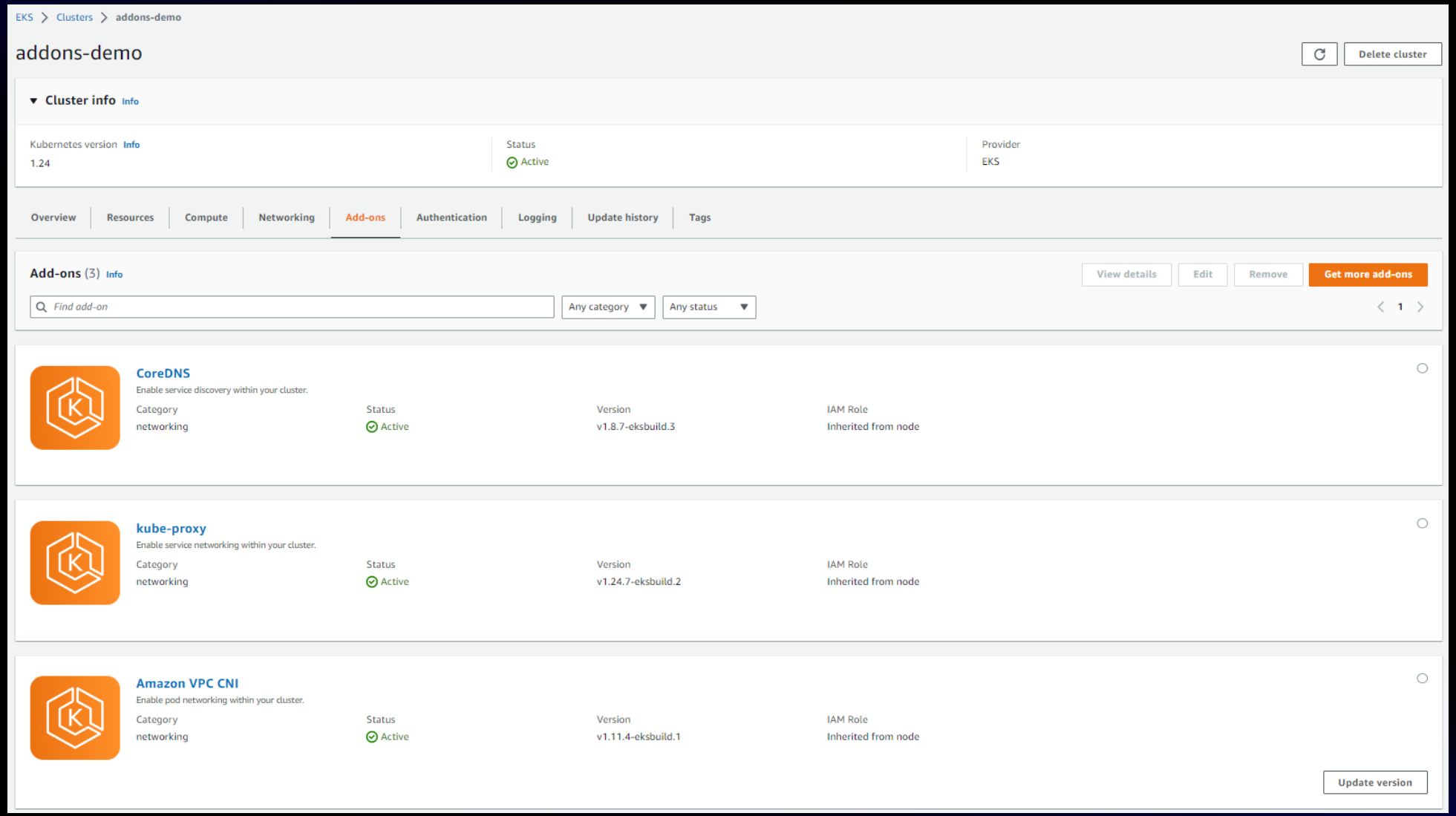
kubernetes-sigs/aws-fsx-csi-driver

Amazon EKS add-ons

- Curated list of operational cluster software
- Built and tested together to work with Amazon EKS
- Enable during cluster creation or any time
- Single operation enable, disable, and update
- Uses worker node IAM role by default
- IAM role for service accounts for finer granularity
- Requires minimum Kubernetes v1.18 and eks.3

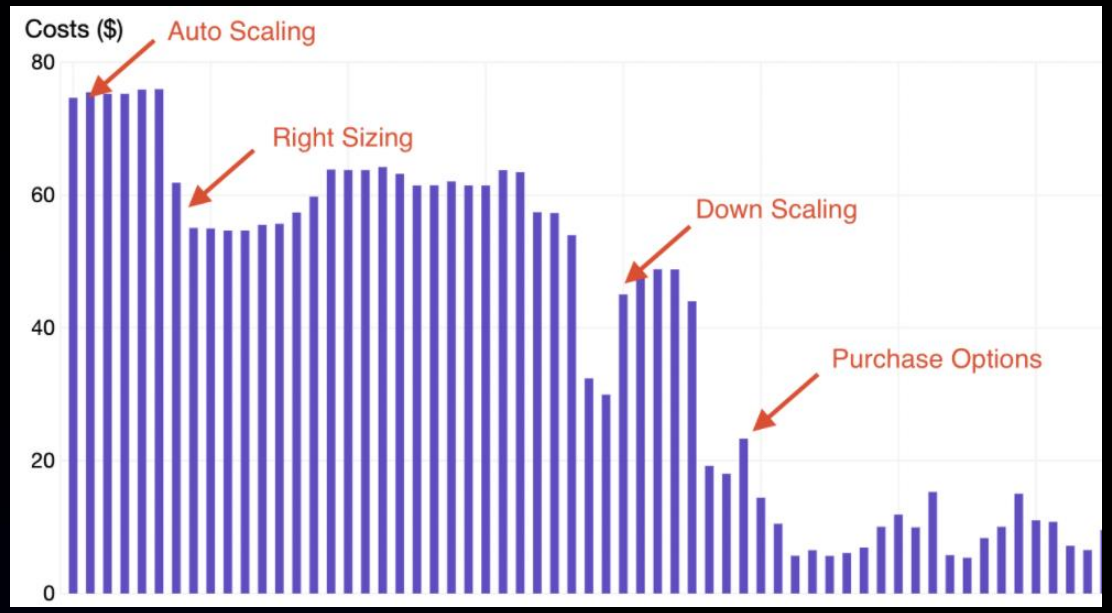
https://docs.aws.amazon.com/eks/latest/userguide/eks-add-ons.html

Amazon EKS add-ons in console



How can I reduce costs in my cluster?

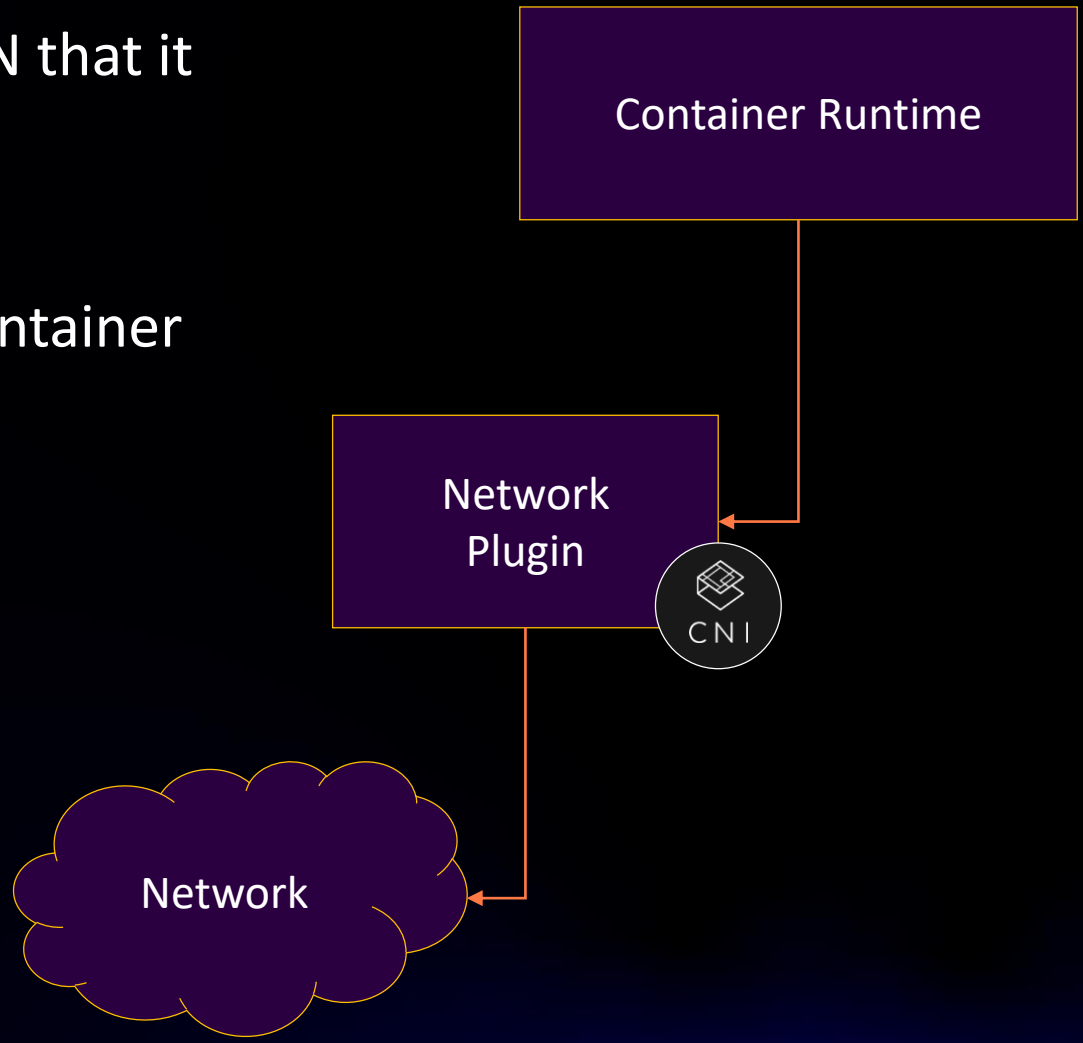
1. Use Spot Instances with managed node groups and save up to 90% off on-demand pricing
2. Use Cluster Autoscaler priority expander to prioritize lower-cost node groups
3. Utilize Savings Plan for Amazon EC2/Fargate
4. Move to latest generation EC2 instances
5. Move workloads to EC2 Graviton
6. Try out Karpenter in large clusters with fast scaling requirements



Networking

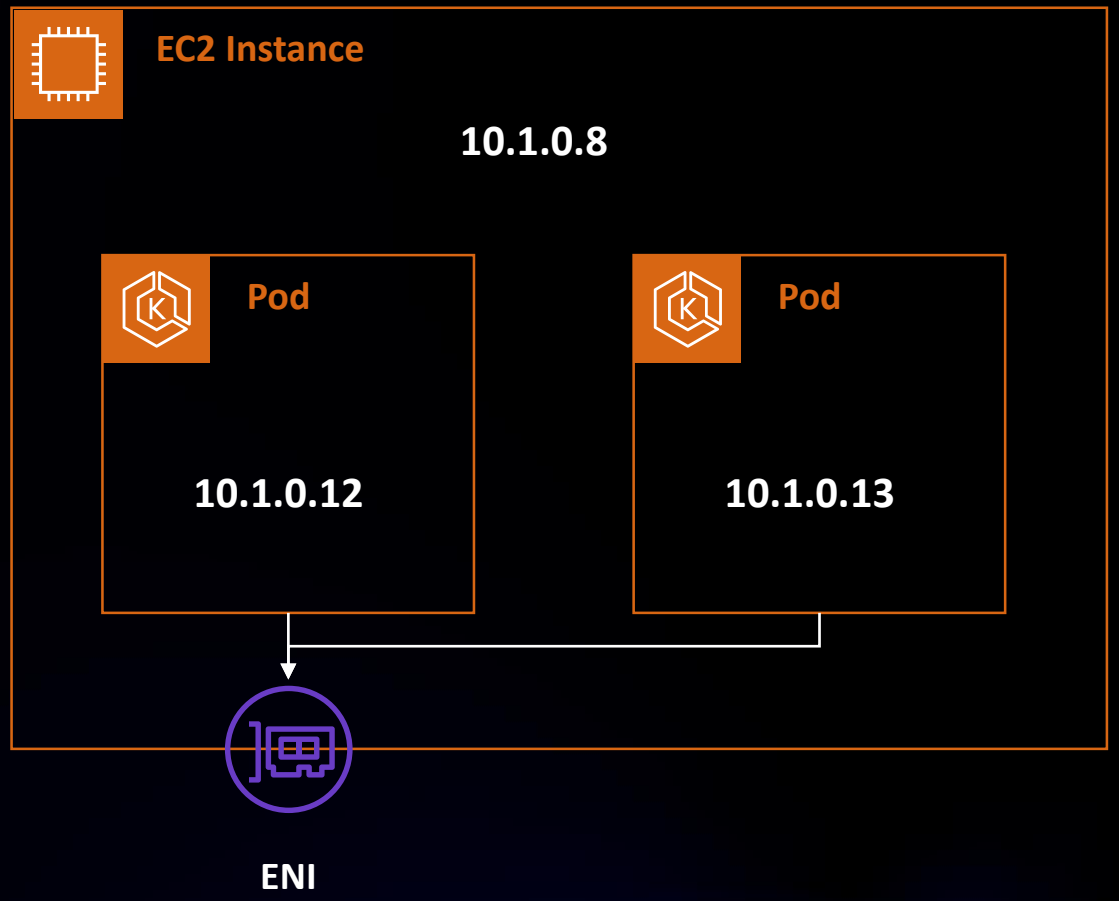
What Is CNI

- 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



VPC CNI Default Configuration

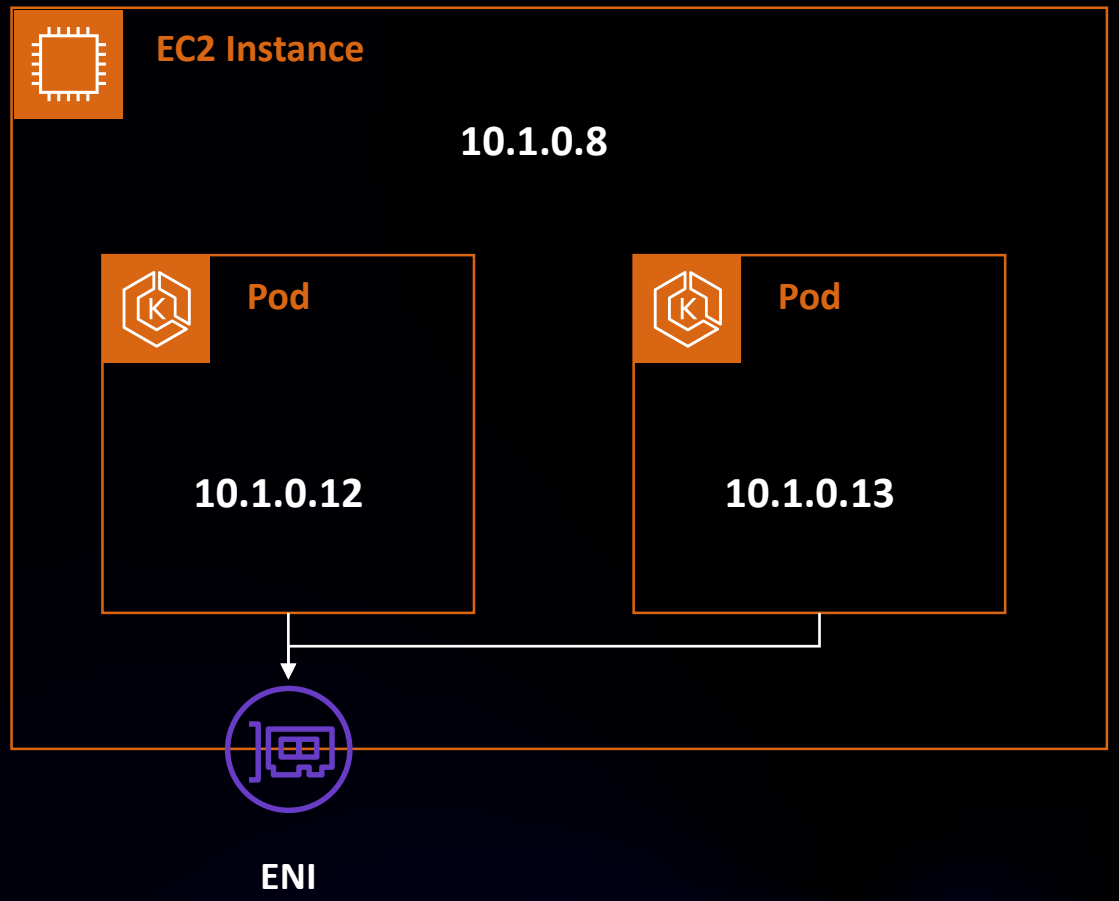
- Native VPC networking via Kubernetes CNI Plugin
- Pod IPs are from the same pool of IP addresses as the VPC
- Simple and secure routing between pods and AWS services.
- Supports Calico Network Policy and EC2 Security groups



aws/amazon-vpc-cni-k8s

VPC CNI Default Configuration

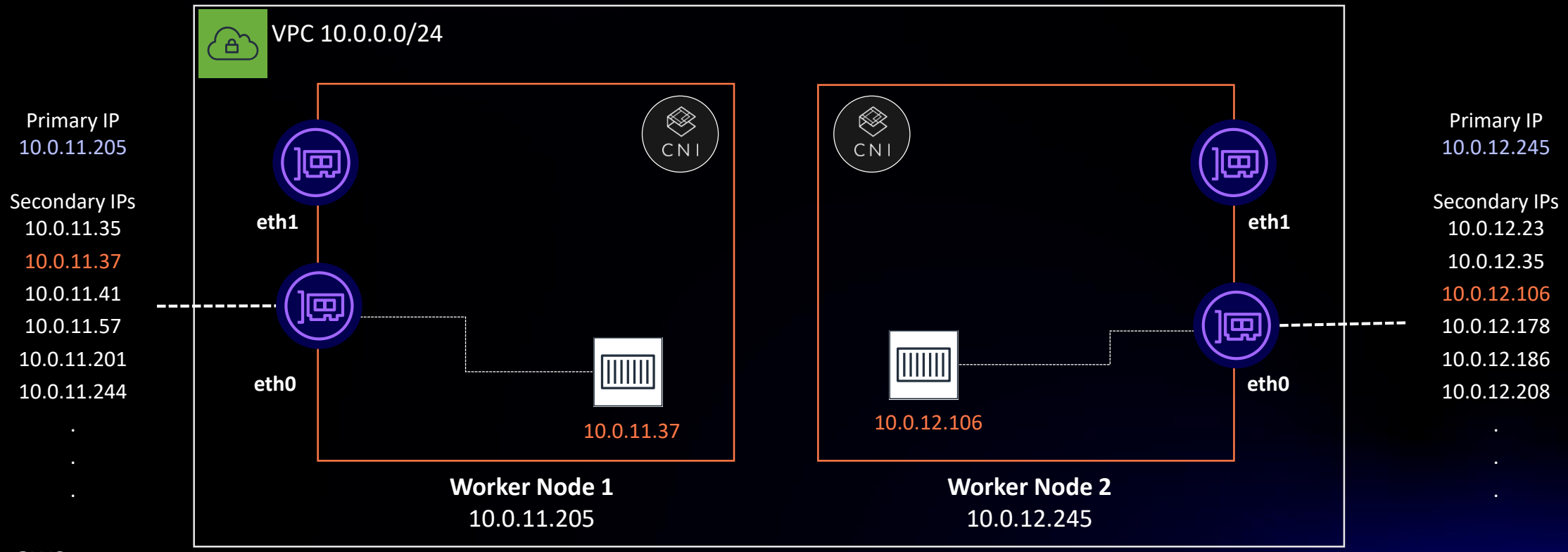
- Native VPC networking via Kubernetes CNI Plugin
- Pod IPs are from the same pool of IP addresses as the VPC
- Simple and secure routing between pods and AWS services.
- Supports Calico Network Policy and EC2 Security groups



aws/amazon-vpc-cni-k8s

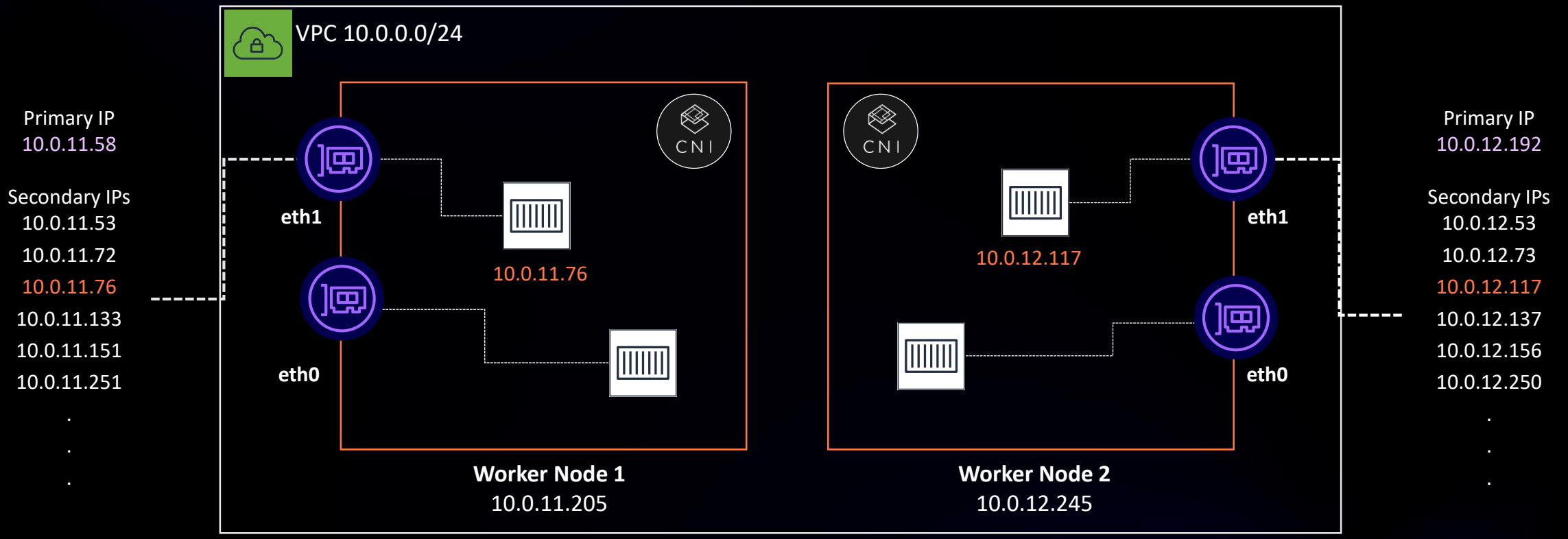
VPC CNI Default Behavior

- Attaches an additional, unused ENI to each node and fills with Secondary IP Addresses
- Keeps a “warm pool” of Secondary IP addresses (from active ENI and unused ENI)
- Pods scheduled on a node are assigned IP addresses from the pool
- When IP addresses on an active ENI are exhausted, another unused ENI is attached



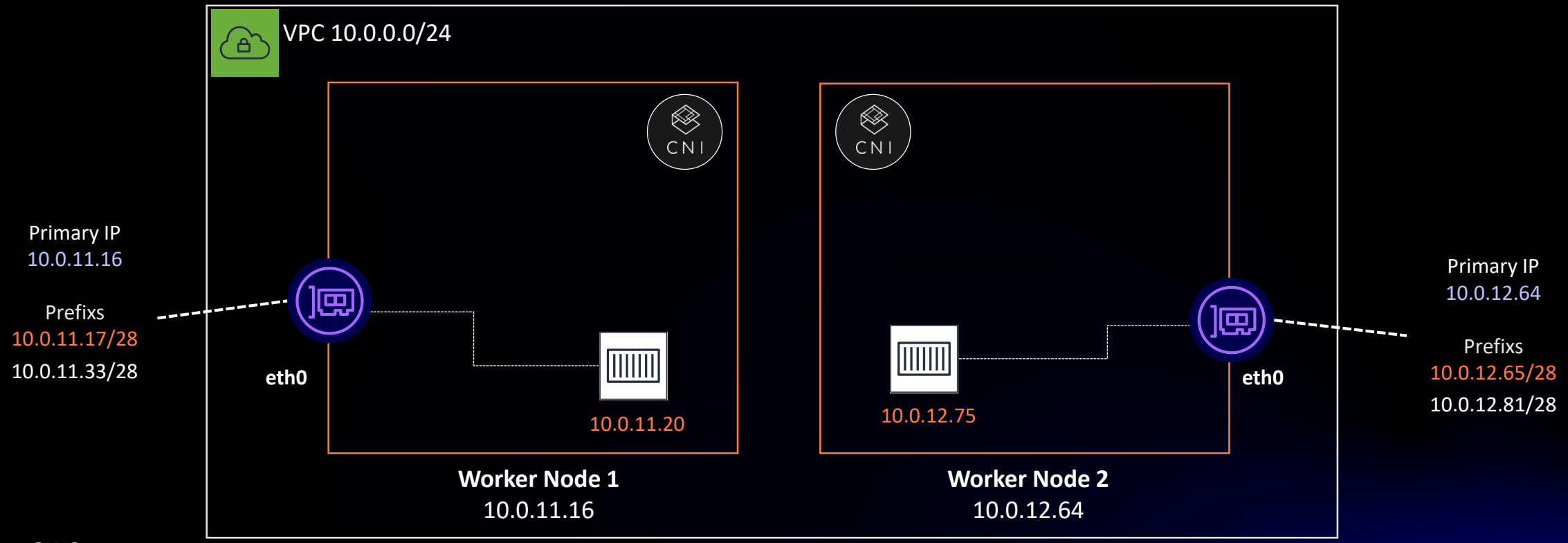
Pod Networking with VPC CNI Plugin

Pods IPs picked from secondary IP addresses allocated to primary and secondary ENIs



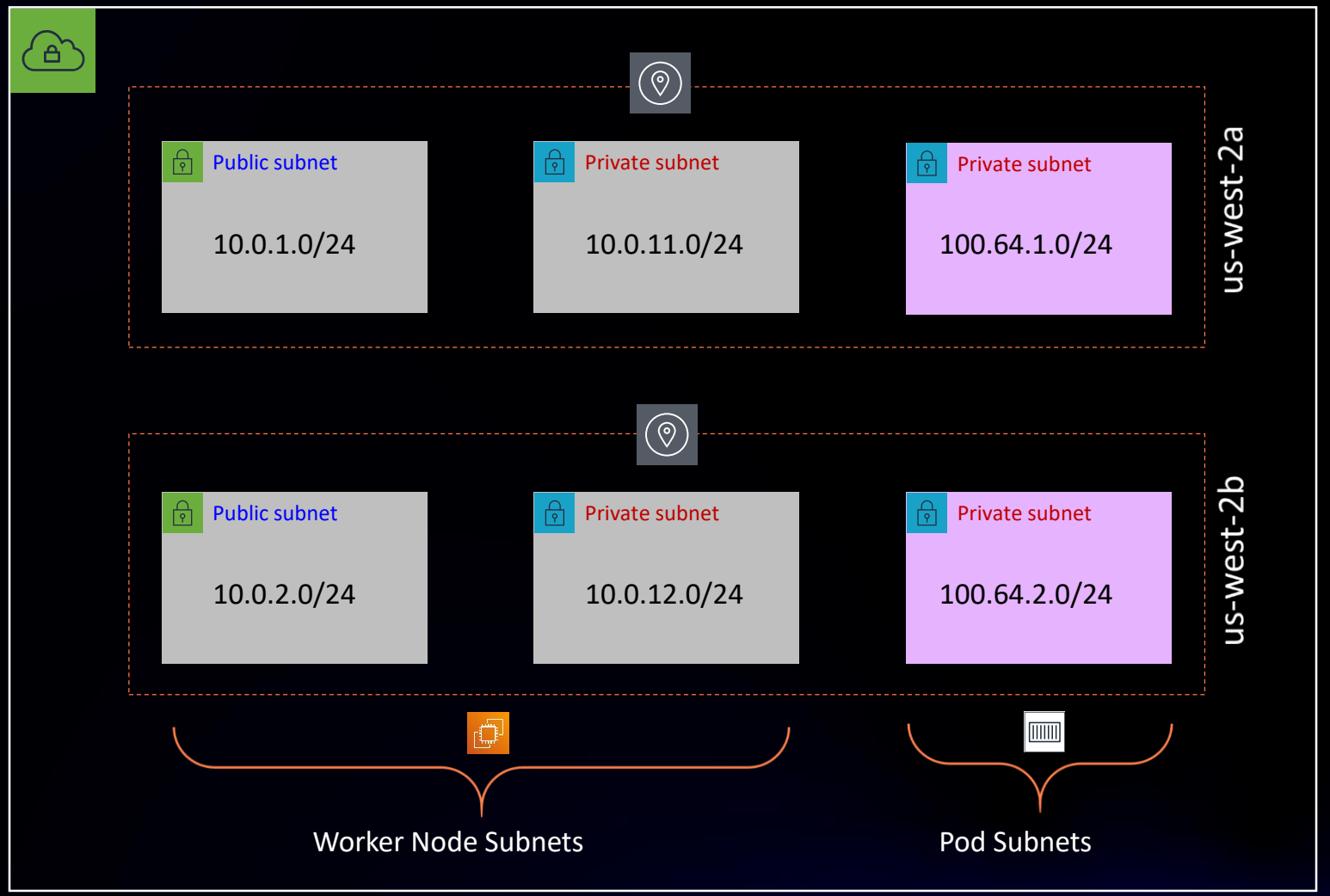
VPC CNI Prefix Assignment Default Behavior

- Attaches an additional, unused /28 prefix (IPv4) or /80 prefix (IPv6) to an ENI of the node
- Keeps a “warm pool” of IP addresses (from active prefix and unused prefix)
- When IP addresses on an active prefix are exhausted, another unused prefix is attached
- New ENI will only be allocated after all the prefixes allocated to existing ENIs are exhausted



Custom Pod Networking with VPC CNI Plugin

- Primary CIDR 10.0.0.0/16
- Attach secondary CIDR 100.64.0.0/16
- Setup subnets with secondary CIDR
- Separate address spaces for nodes and pods
  - Worker nodes → primary CIDR
  - Pod → secondary CIDR

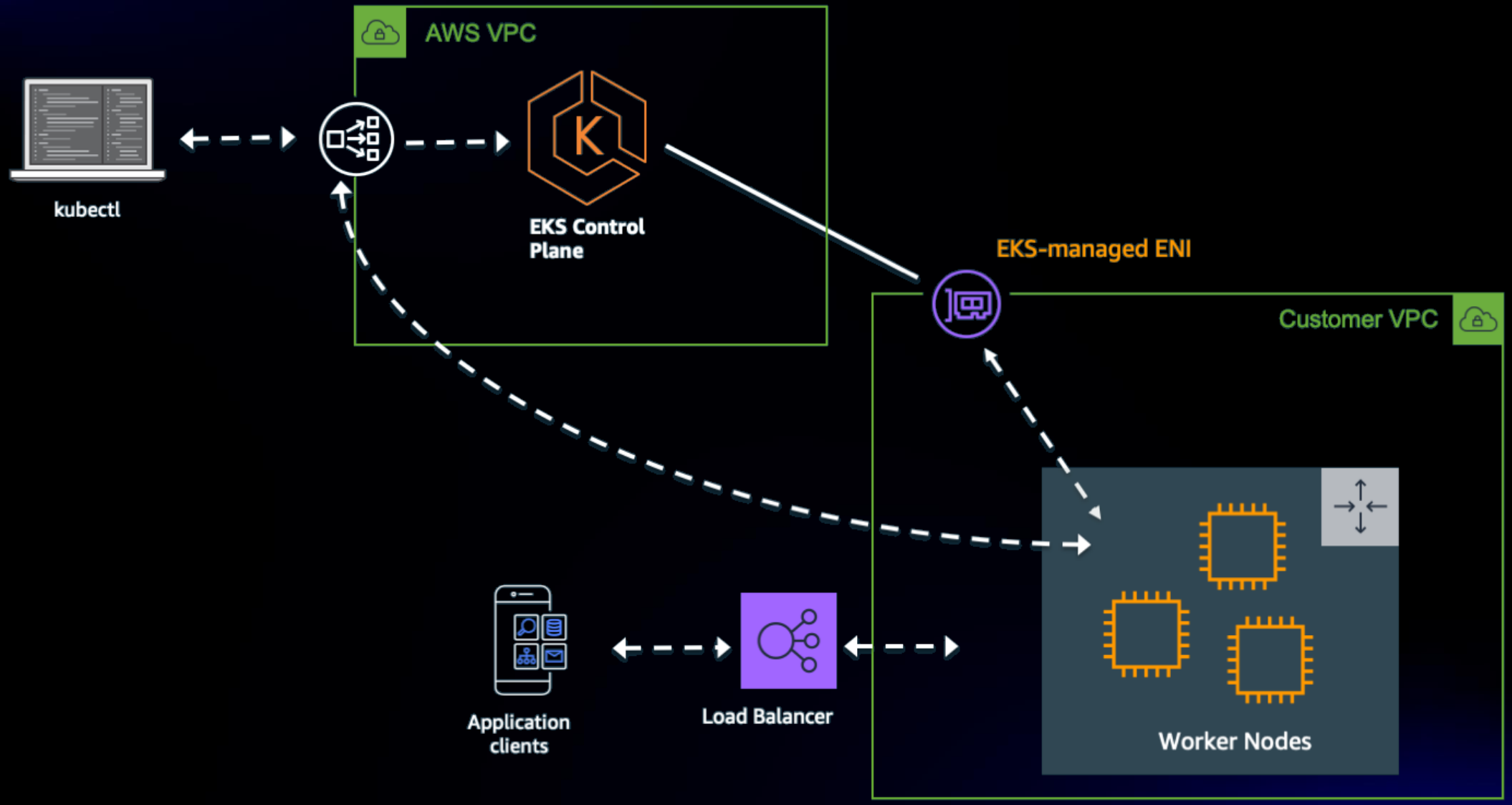


Maximum Pods per Instance Type (IPv4)

- Max pods per worker node is dependent on configuration or choice of:
  - EC2 instance type
  - Prefix Assignment
  - Custom Networking
- Kubernetes Best Practices recommends no more than 110 Pods per node

Instance type	Maximum network interfaces	Private IPv4 addresses per interface	Maximum pods WITHOUT Prefix Assignment	Maximum pods WITH Prefix Assignment
c5.large	3	10	29	110
c5.2xlarge	4	15	58	110
t3.small	3	4	11	110
c5.9xlarge	8	30	234	250

EKS Cluster Architecture



AWS Load Balancer Controller

AWS Load Balancer Controller is a controller to help manage Elastic Load Balancers for a Kubernetes cluster.

- It satisfies Kubernetes Ingress resources by provisioning Application Load Balancers.
- It satisfies Kubernetes Service resources by provisioning Network Load Balancers.

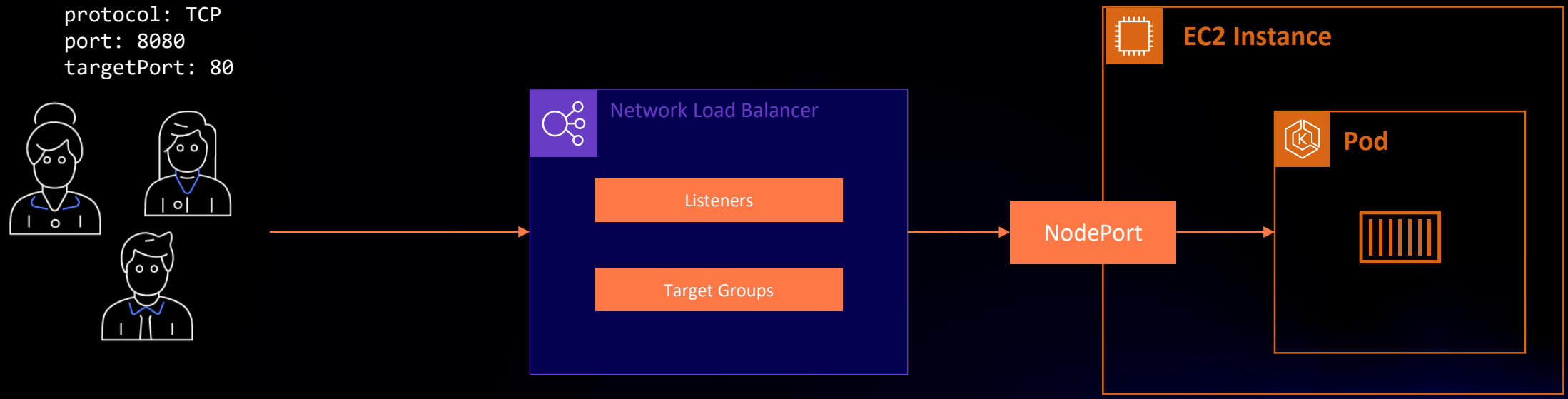
Just annotate services/ingress
NLB/ALB provisioned on your behalf

https://kubernetes-sigs.github.io/aws-load-balancer-controller/latest/

Service Type Load Balancer

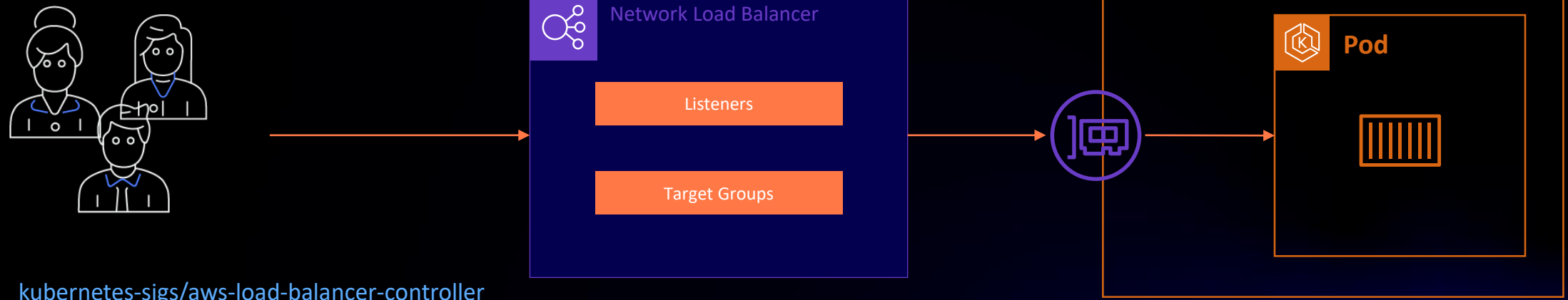
```
kind: Service
apiVersion: v1
metadata:
    name: nginx-service
    annotations:
       service.beta.kubernetes.io/aws-load-balancer-type: "external”
       service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: "instance”
       service.beta.kubernetes.io/aws-load-balancer-scheme: “internet-facing”
spec:
    type: LoadBalancer
    selector:
        app: nginx
    ports:
        - name: http
          protocol: TCP
          port: 8080
          targetPort: 80
```

Network Load Balancers are supported using an annotation on the service.



AWS Load Balancer Controller – Network Load Balancer

```
kind: Service
apiVersion: v1
metadata:
    name: nginx-service
    annotations:
       service.beta.kubernetes.io/aws-load-balancer-type: "external"
       service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: "ip”
      service.beta.kubernetes.io/aws-load-balancer-scheme: “internet-facing”
spec:
    type: LoadBalancer
    selector:
        app: nginx
    ports:
        - name: http
          protocol: TCP
          port: 8080
          targetPort: 80
```

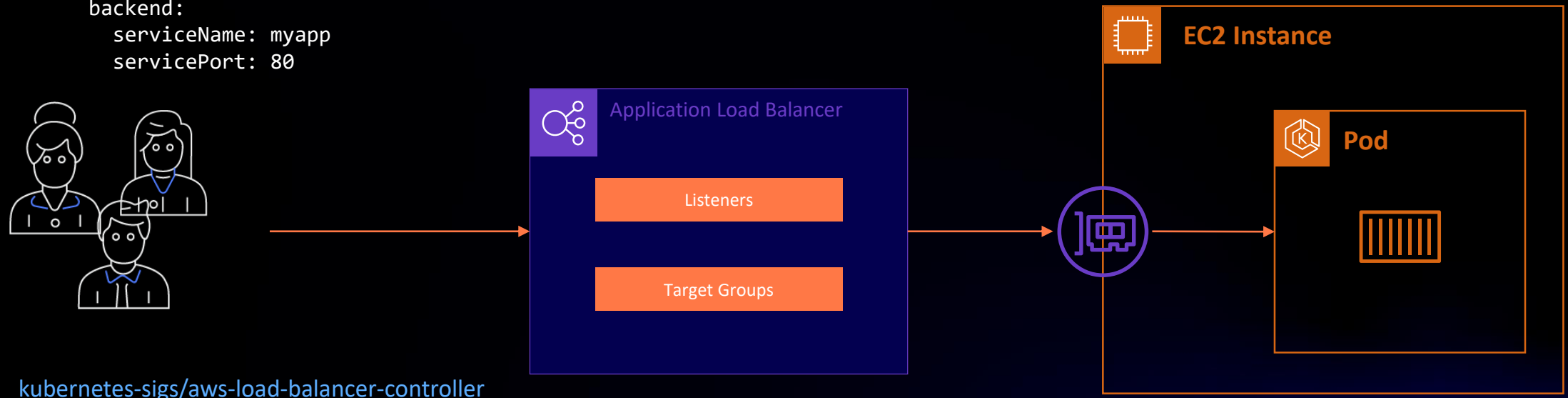


[kubernetes-sigs/aws-load-balancer-controller]

AWS Load Balancer Controller – Application Load Balancer

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: myapp-ingress
  annotations:
    kubernetes.io/ingress.class: alb
    alb.ingress.kubernetes.io/scheme: internet-facing
  labels:
    app: nginx
spec:
  rules:
    - http:
        paths:
          - path: /*
            backend:
              serviceName: myapp
              servicePort: 80
```

Application Load Balancers are created using an Ingress Resource.



[kubernetes-sigs/aws-load-balancer-controller]

Installing a Network Policy Provider on Kubernetes

- Firewalling within Kubernetes is controlled by NeworkPolicies. You first need to add a Network Policy Provider to EKS / Kubernetes in order to use Network Policies. A popular one covered in our documentation is Calico.
- https://docs.aws.amazon.com/eks/latest/userguide/calico.html

Controlling Networking Traffic within a Namespace

You can control network traffic between pods using Network Policies deployed within the cluster.

```
apiVersion: projectcalico.org/v3
kind: NetworkPolicy
metadata:
    name: allow-tcp-6379
spec:
    selector: app == 'redis'
    ingress:
        - action: Allow
          protocol: TCP
          source:
              selector: app == 'api'
          destination:
                    ports: [6379]
```

https://docs.aws.amazon.com/eks/latest/userguide/calico.html

aws

Controlling Networking Traffic by IP

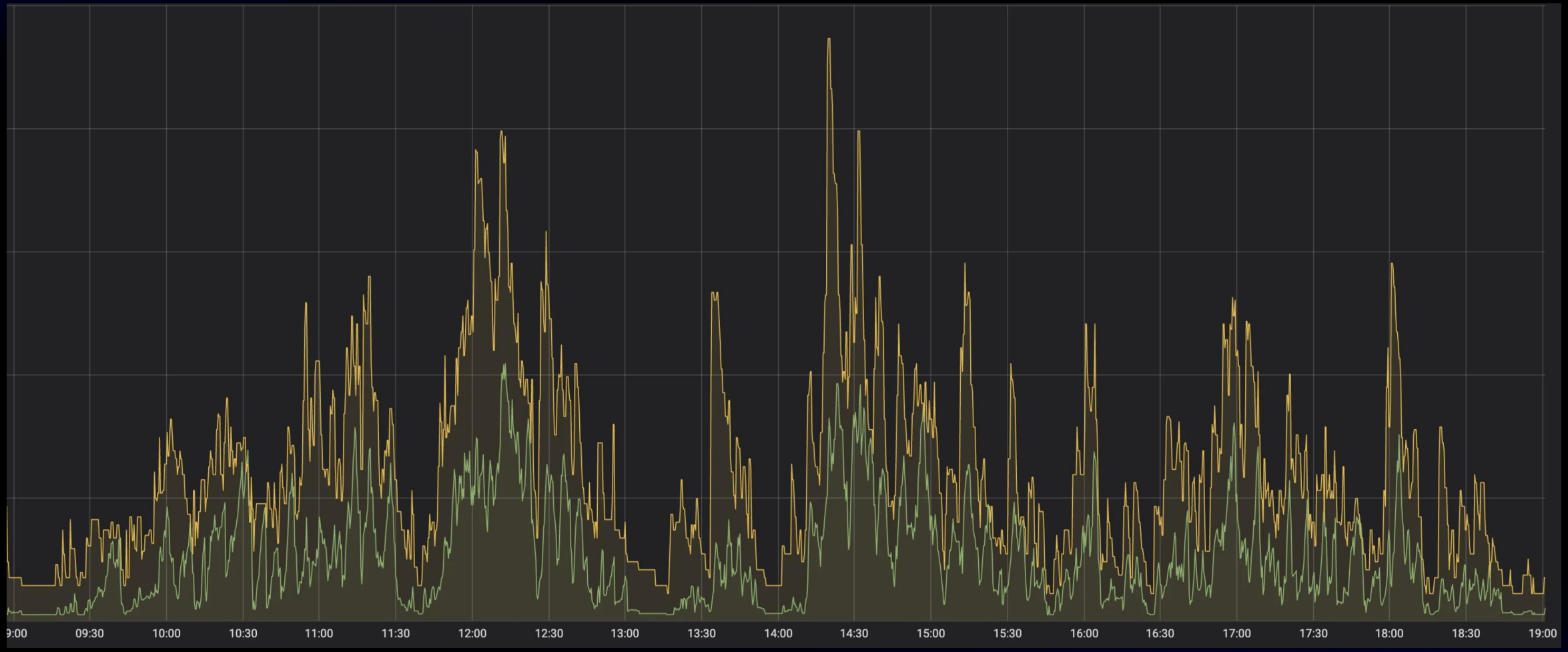
You can control access to external networks.

```
apiVersion: projectcalico.org/v3
kind: NetworkPolicy
metadata:
    name: allow-egress-internal
spec:
    selector: color == 'redis'
    types:
        - Egress
    egress:
        - action: Deny
          destination:
              nets: [0.0.0.0/0]
        - action: Allow
          destination:
              nets: [10.1.0.0/16]
```

https://docs.aws.amazon.com/eks/latest/userguide/calico.html

Autoscaling

Ressources needed will evolve with time



Kubernetes Autoscaling

Pod-based scaling (application layer)

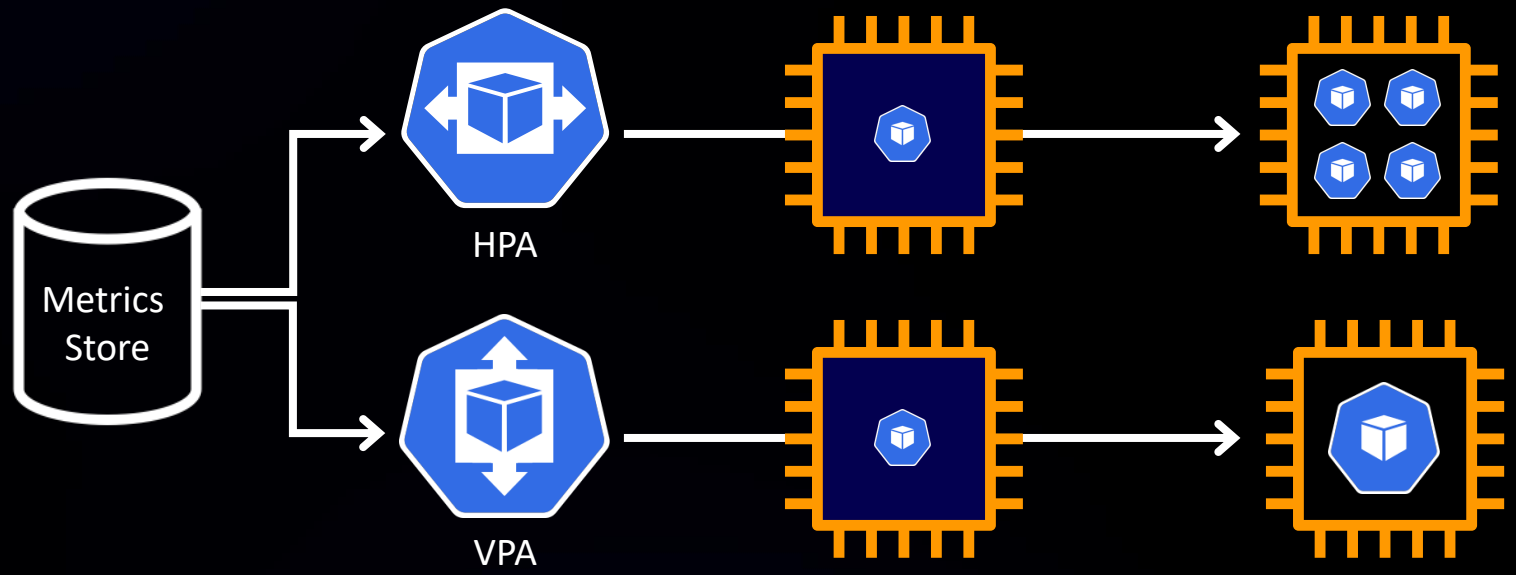
1. HPA – Horizontal Pod Autoscaler
2. VPA – Vertical Pod Autoscaler

Node-based scaling (infrastructure layer)

1. CA – Cluster Autoscaler
2. Karpenter

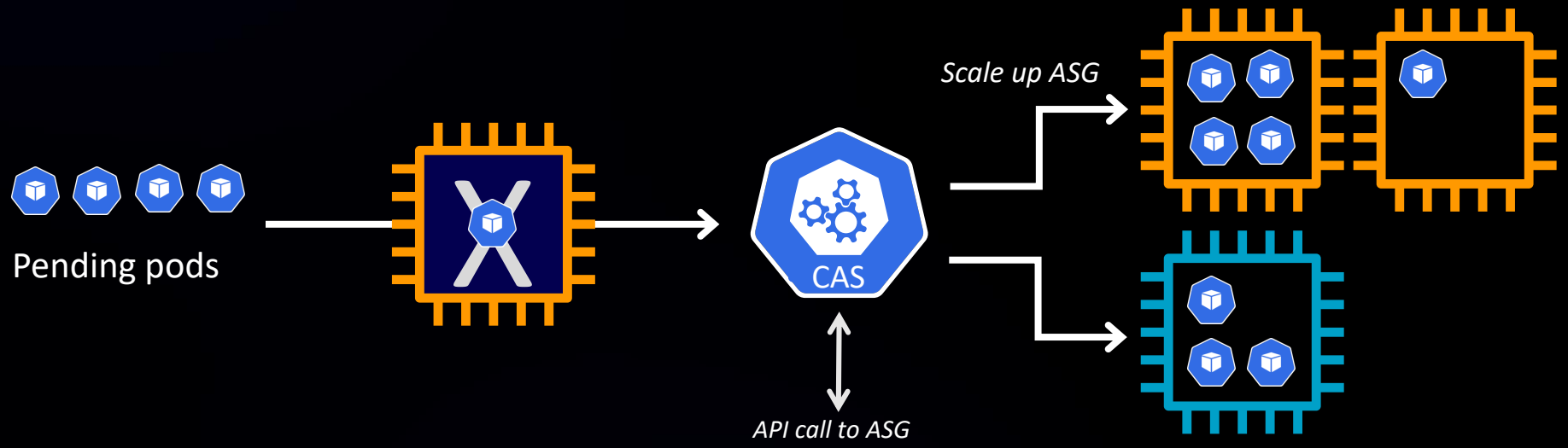
Pod-based scaling (application layer)

1. **Horizontal Pod Autoscaling (HPA)**
2. **Vertical Pod Autoscaling (VPA)**

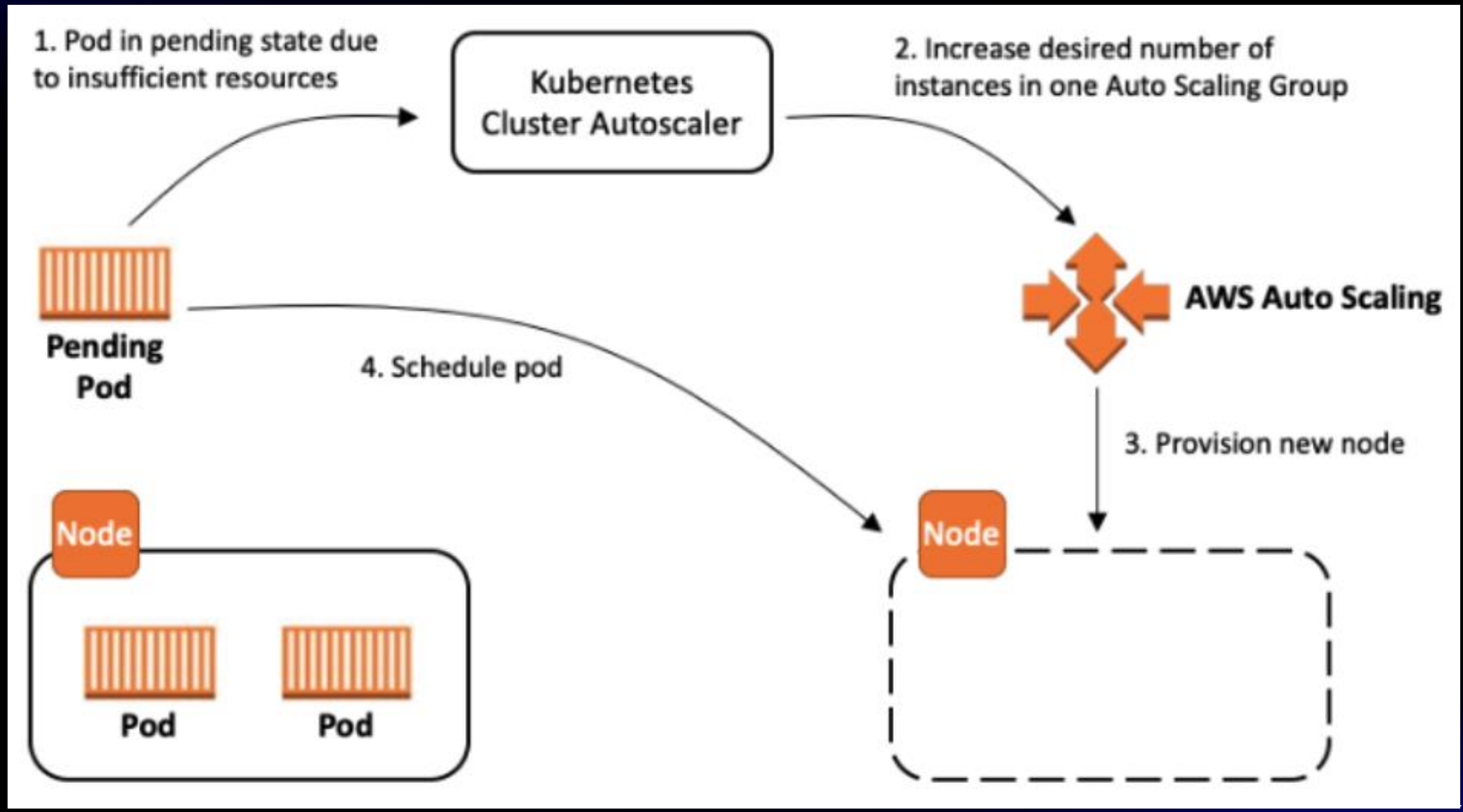


Node-based scaling (infrastructure layer)

Cluster Autoscaler (CAS)



Cluster Autoscaler - Diagram



How CA Works

1. CA doesn't look at memory or CPU available when it triggers
2. CA reacts to events and checks for any un-schedulable Pods, every 10 seconds
3. Pods are un-schedulable when the scheduler is unable to find a node to accommodate
4. CA triggers the provisioning of the new compute node, and then leaves it up to the kube-scheduler to schedule the pod(s) on the new node

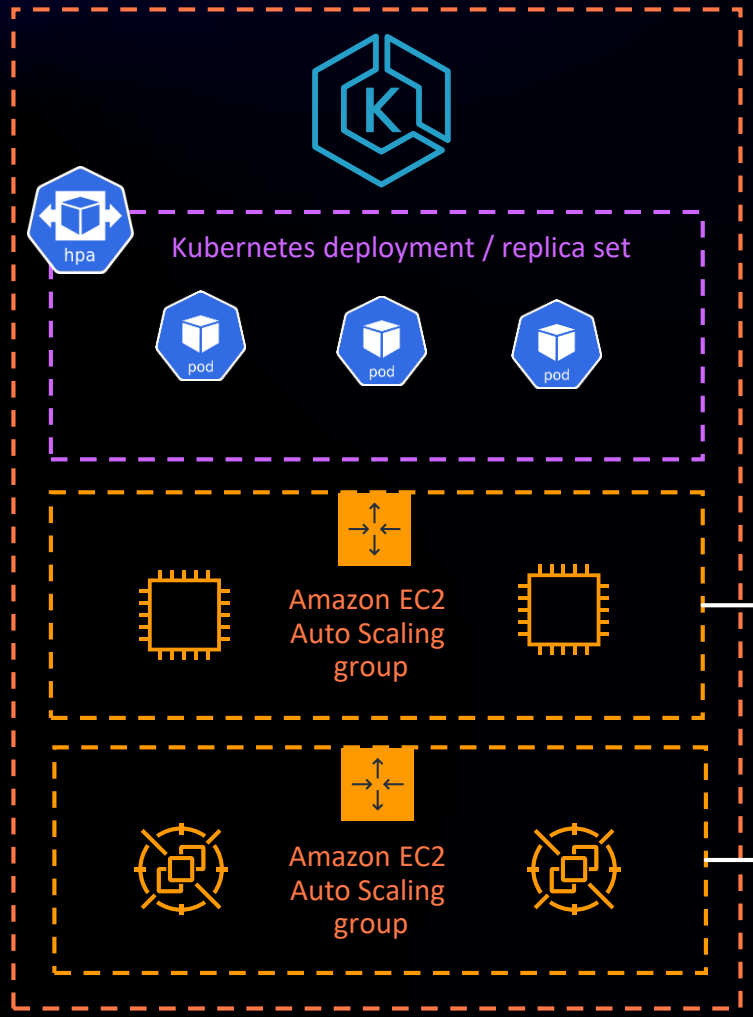
CA Uses Scheduling Simulation

- Unlike HPA and VPA, CA doesn't depend on load metrics. Instead, it's based on scheduling simulation and declared Pod requests. It's a best practice to enable CA whenever you are using either HPA or VPA. This practice ensures that if your Pod autoscalers determine that you need more capacity, your underlying infrastructure grows accordingly.
- Cluster Autoscaler expects requested nodes to appear within 15 minutes (configured by `--max-node-provision-time flag`.) After this time, if they are still unregistered, it stops considering them in simulations and may attempt to scale up a different group if the pods are still pending. It will also attempt to remove any nodes left unregistered after this time.

CA – Install/Configure

- kubectl or helm
- Use IAM Roles for Service Accounts (IRSA) and OIDC Identity Provider
  - Create IRSA Role with `eksctl` and `--asg-access`
- Install Prometheus to get CA metrics
- Match versions between CA & K8s
- Prevent eviction with annotations – `cluster-autoscaler.kubernetes.io/safe-to-evict: "false”`
- Make sure ASG auto-discovery is configured - `--node-group-auto-discovery=asg:tag=k8s.io/cluster-autoscaler/enabled,k8s.io/cluster-autoscaler/<CLUSTER_NAME>`

Scaling the Application and Infrastructure



Horizontal pod autoscaler

Scale number of pods in a deployment on the basis of observed CPU utilization or application-provided metrics

Kubernetes Cluster Autoscaler

- Scale-out Amazon EC2 Auto Scaling groups of nodes matching pod scheduling requirements when pods are in a *pending* state because of insufficient resources
- Scale in when nodes are underutilized and important pods can be rescheduled somewhere else

Auto Scaling group tags:

```
k8s.io/cluster-autoscaler/enabled: true
k8s.io/cluster-autoscaler/${cluster_name}: owned
```

CA Expanders

```
--expander=least-waste
```

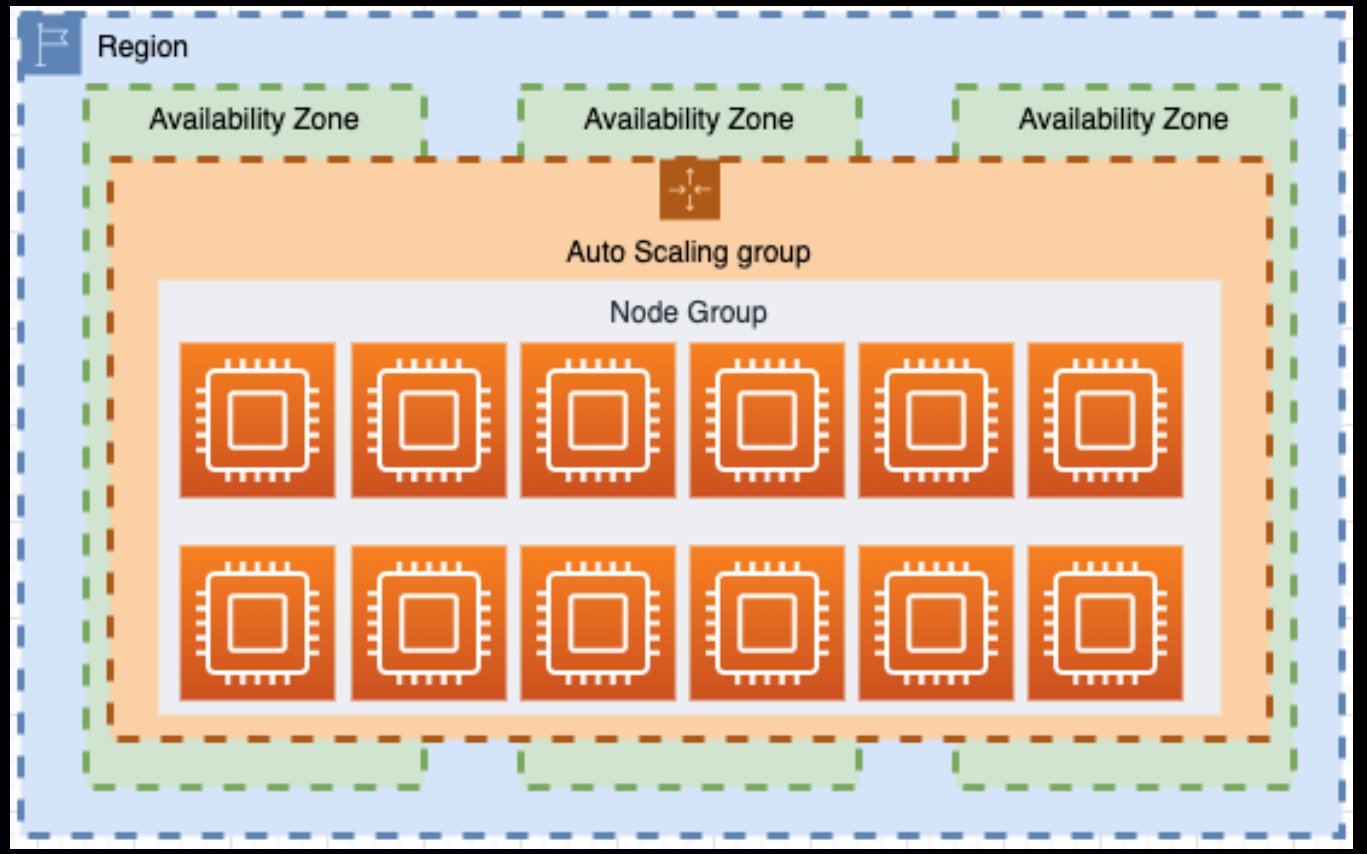
- Provide different strategies for selecting the node group to which new nodes will be added
- random - [DEFAULT] - used when no need for the node groups to scale differently
- most-pods - selects the node group that would be able to schedule the most pods when scaling up.
  - Useful when using nodeSelector to make sure certain pods land on certain nodes.

CA Expanders

- least-waste - selects the node group that will have the least idle CPU (if tied, unused memory) after scale-up.
  - This is useful when you have different classes of nodes, for example, high CPU or high memory nodes, and only want to expand those when there are pending pods that need a lot of those resources.
- price - select the node group that will cost the least and, at the same time, whose machines would match the cluster size.
  - Currently it works only for GCE and GKE
- priority - selects the node group that has the highest priority assigned by the user.

Multi-AZ Setup

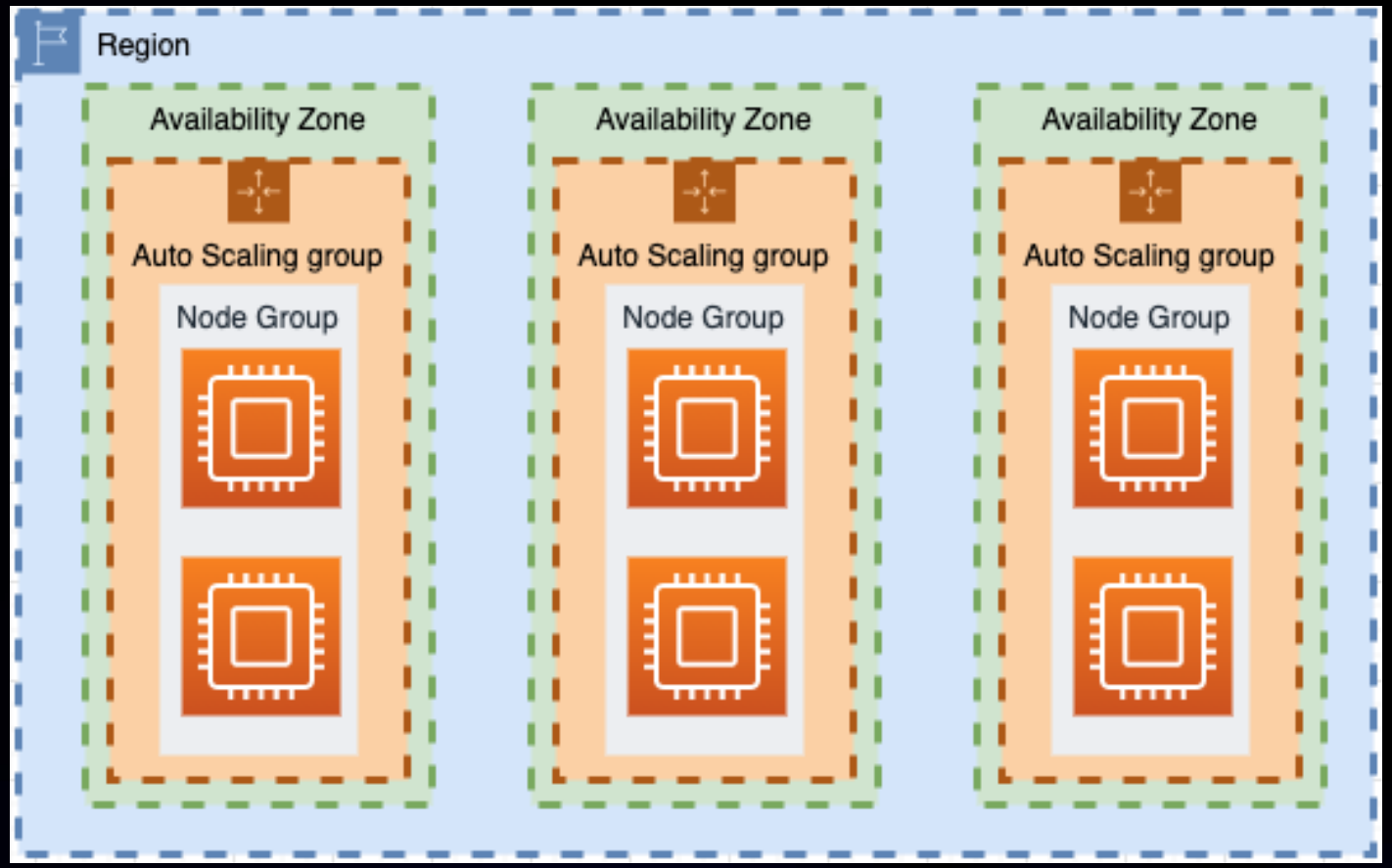
- Cluster setup with 1 node group, 1 ASG across 3 AZs



Each node group maps to a single Auto Scaling group. However, Cluster Autoscaler requires all instances within a node group to share the same number of vCPU and amount of RAM.

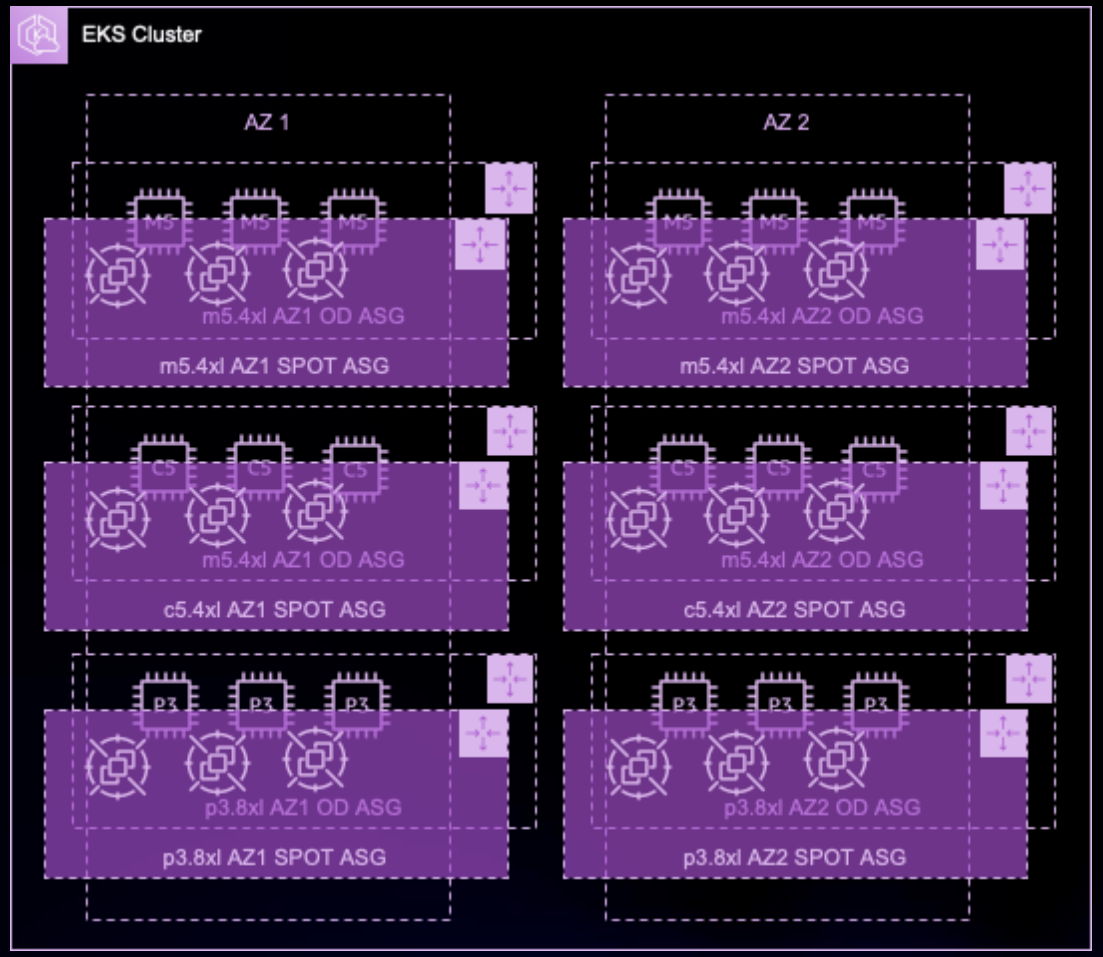
Multi-AZ Considerations - EBS

- EBS and PVC – Stateful Apps
- Could Choose AZ-Bounded ASGs
- Separate Kubernetes deployments for each AZ.



Node Group and Auto Scaling Group Sprawl

- Different workloads need different compute resources
- Not all workloads need to be isolated in separate clusters
- Balancing the needs of different workloads adds complexity

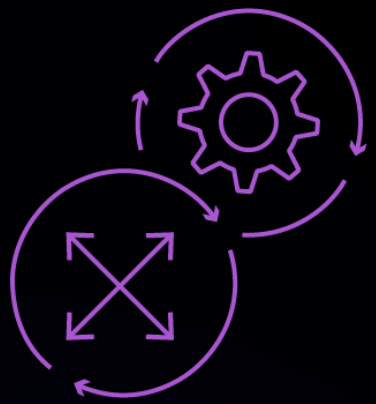


What is Karpenter?

Karpenter is an ***open-source***, ***flexible***, and ***high-performance*** Kubernetes cluster autoscaler.

Open source and Kubernetes-native

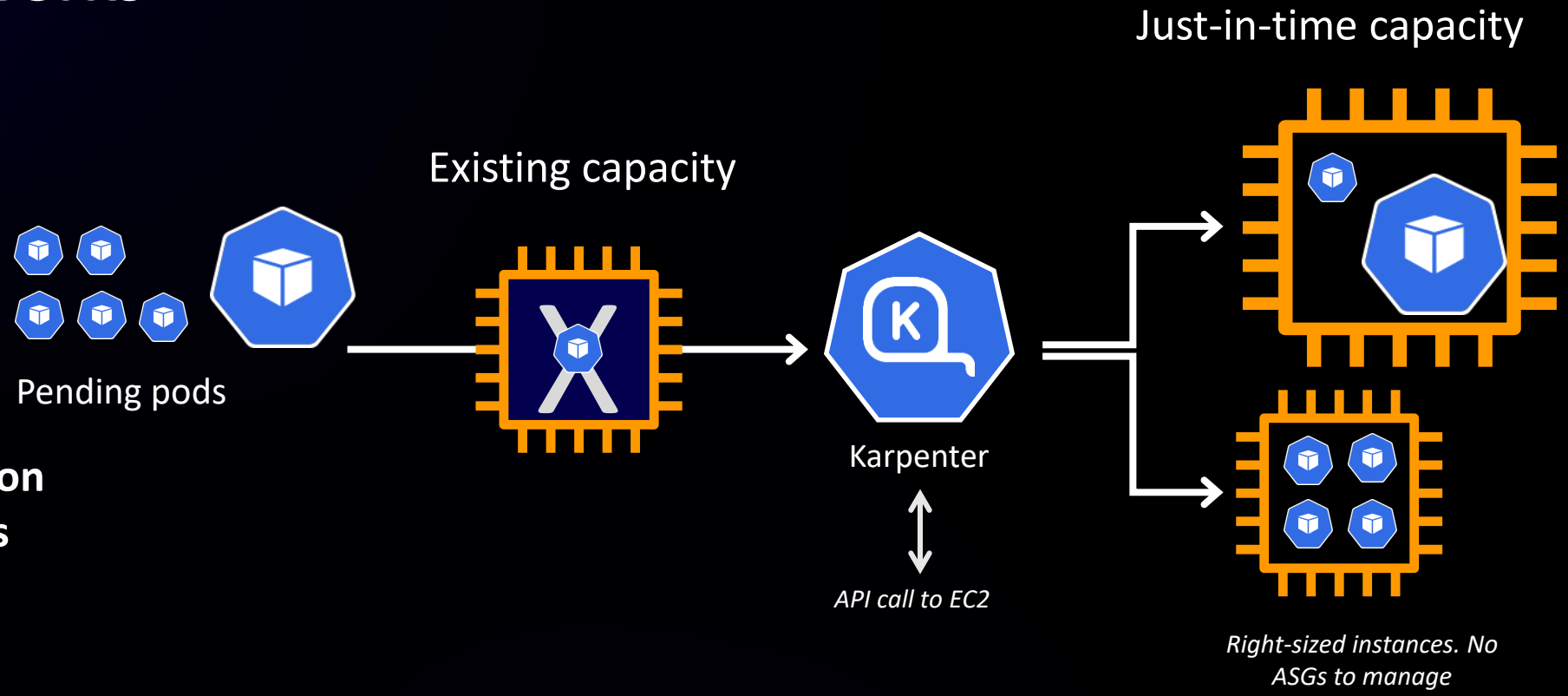
Dynamic, group-less node provisioning



Automatic node sizing

High-performance at scale

How Karpenter Works



Karpenter

Automate instance selection based on workloads needs

Deeply integrated with EC2

EC2 Fleet API, no ASGs

Deeply Kubernetes native

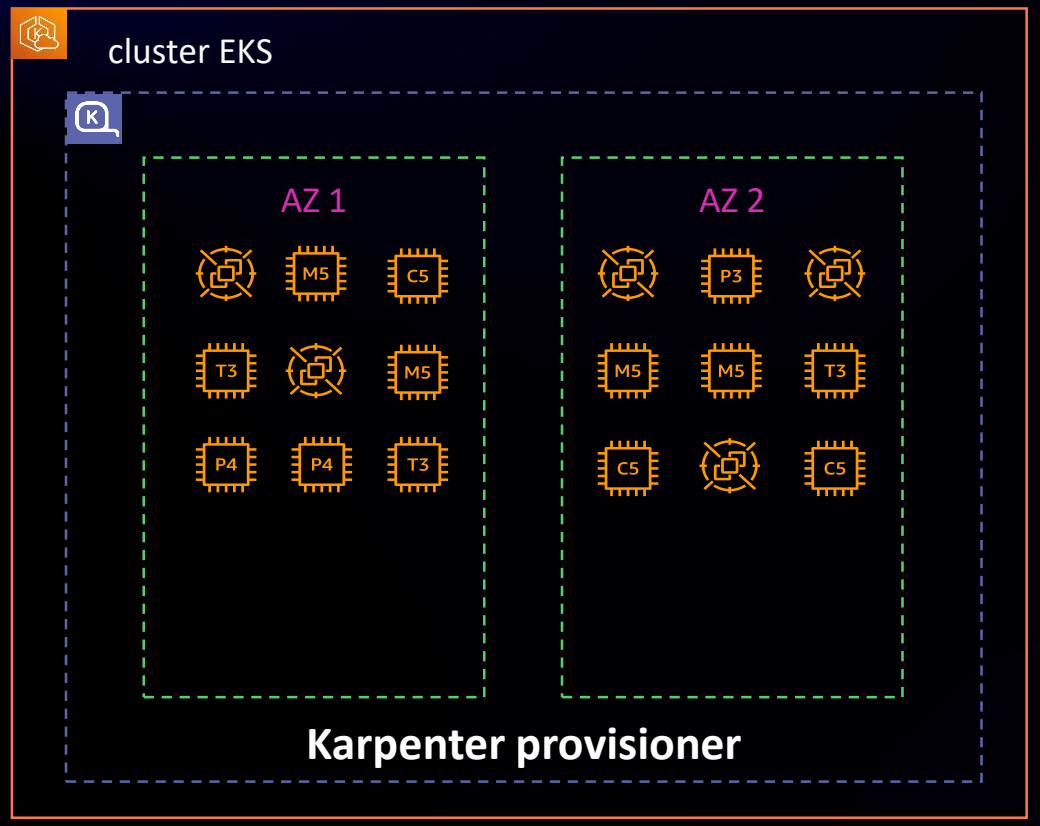
Watch API, Labels, Finalizers

Automated instance selection

Matches workload needs to instance type

aws

Flexible Cluster Auto Scaling



Compute is best fit to cluster workloads automatically

Declarative k8s custom resource definition (CRD) configuration

```
apiVersion: karpenter.sh/v1alpha5
kind: Provisioner
spec:
  consolidation:
    enabled: true
  requirements:
      - key: karpenter.k8s.aws/instance-family
          operator: NotIn
          Values: [a1,c1,c3,inf1,t3,t2]
      - key: karpenter.k8s.aws/instance-generation
          operator: Gt
          values:
          - '2'
       - key: kubernetes.io/arch
          operator: In
          values:
          - amd64
       - key: karpenter.sh/capacity-type
          operator: In
          value : [on-demand, spot]
  taints:
      - effect: NoSchedule
      key: karpenter
      value: 'true'
  ttlSecondsUntilExpired: 2592000
  weight: 1
```

aws

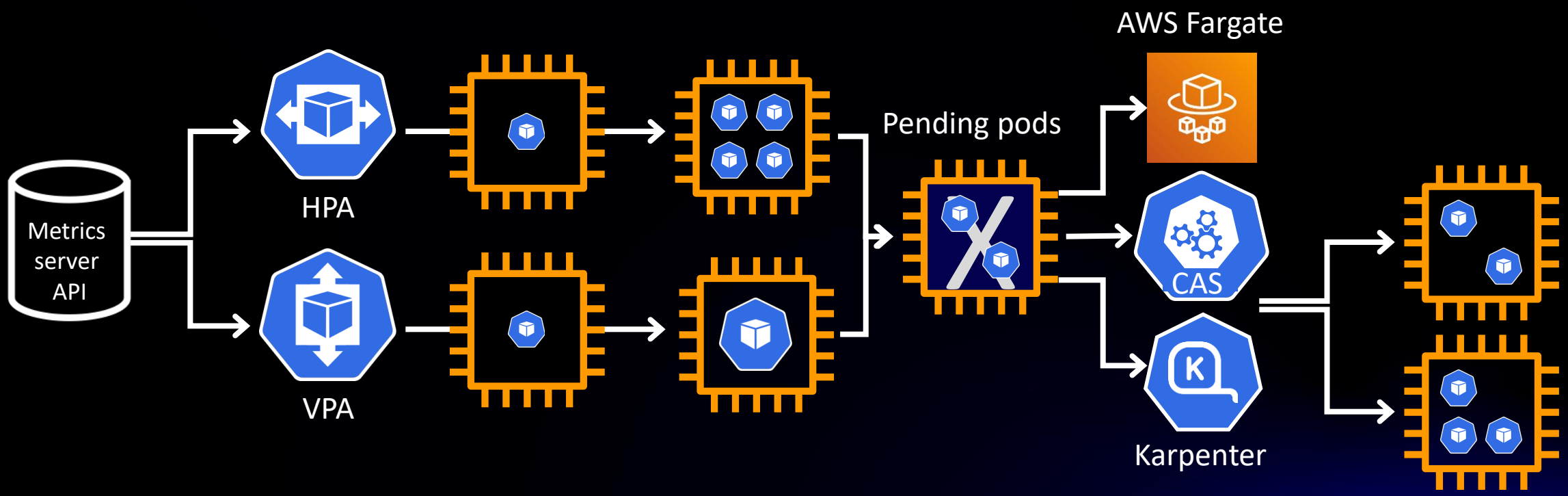
How do I right-size compute capacity?

Scale pods horizontally/vertically based on usage

Set pod resource requests close to actual utilization

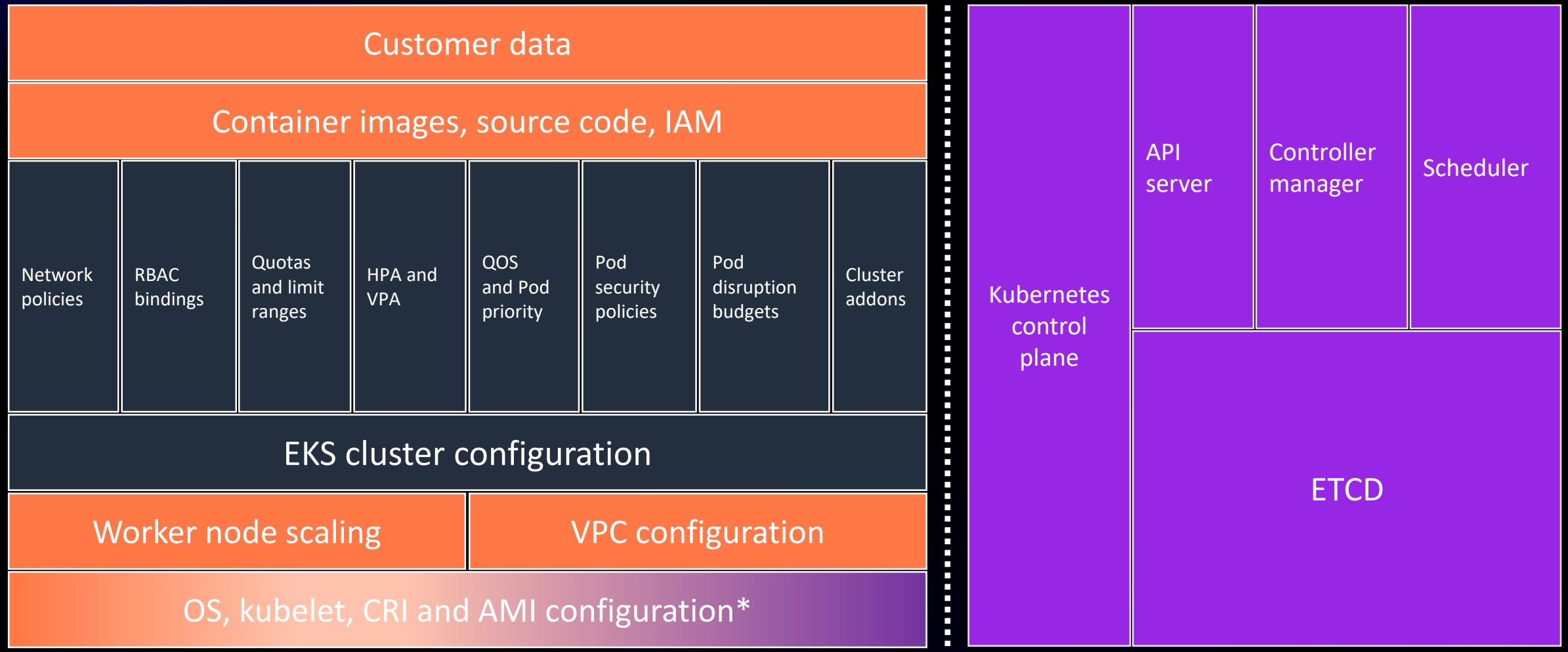
Scale node resources based on pod requirements

Limit overprovisioning of compute capacity

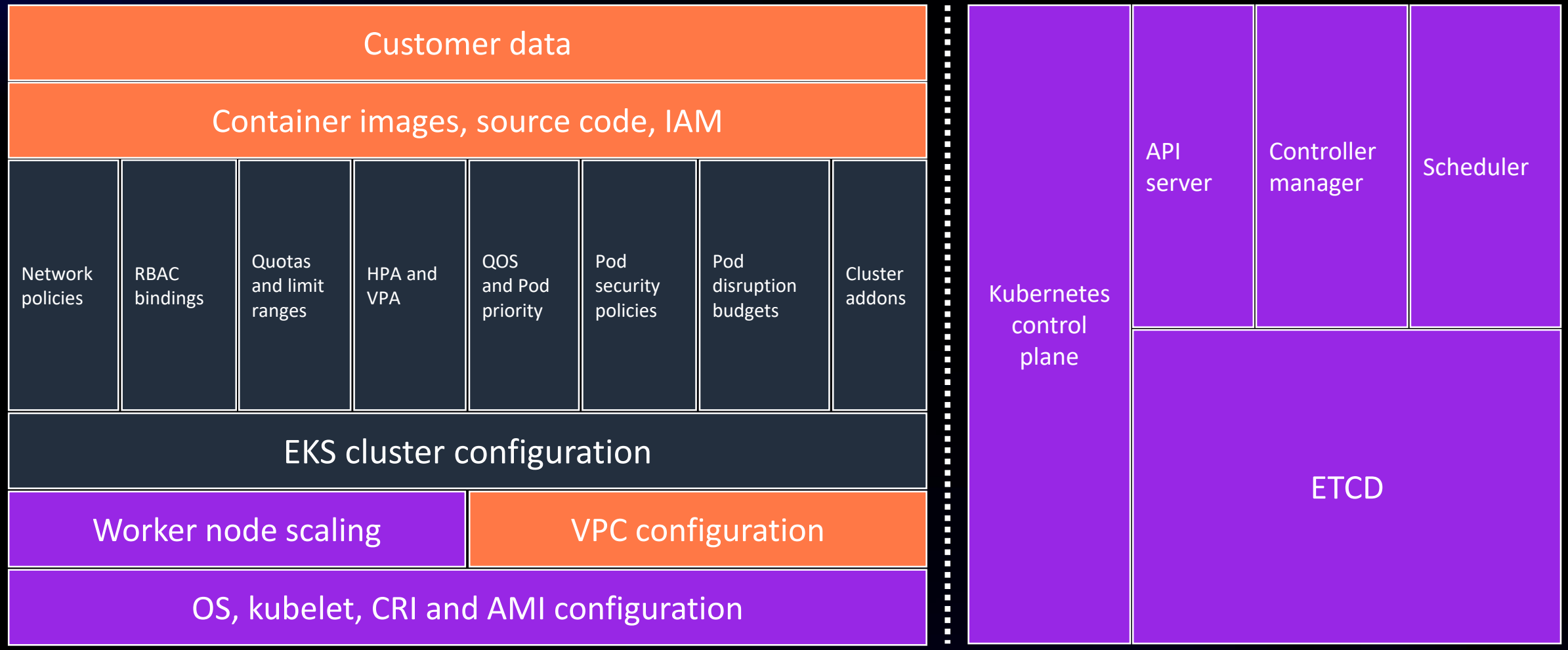


Security

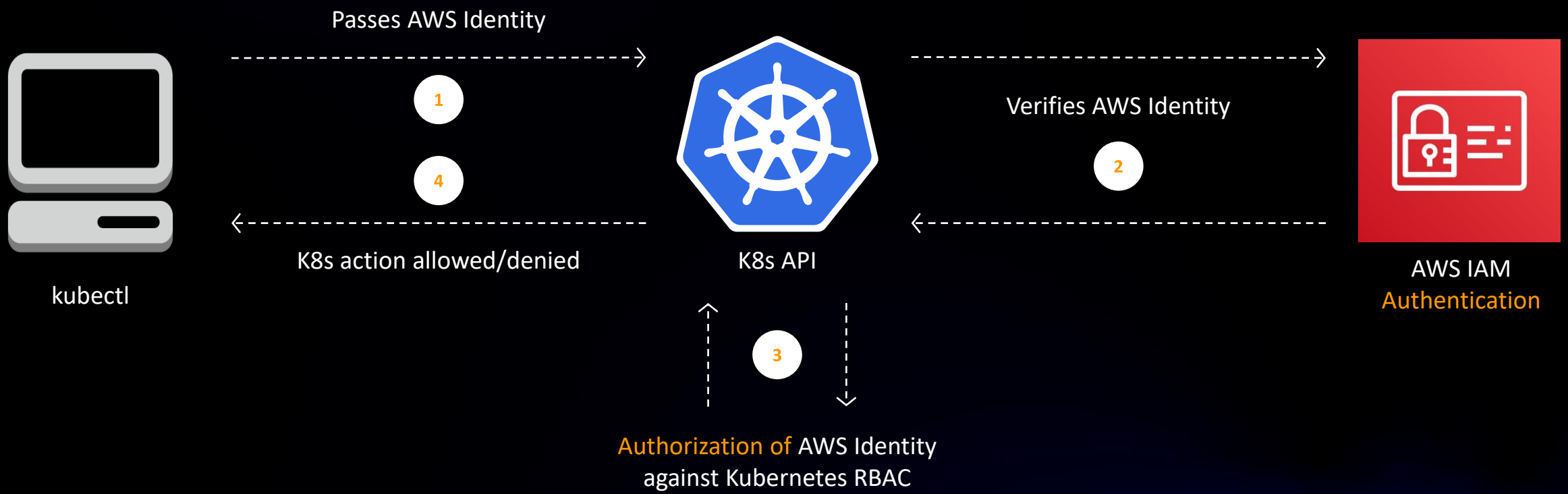
EKS with managed node groups



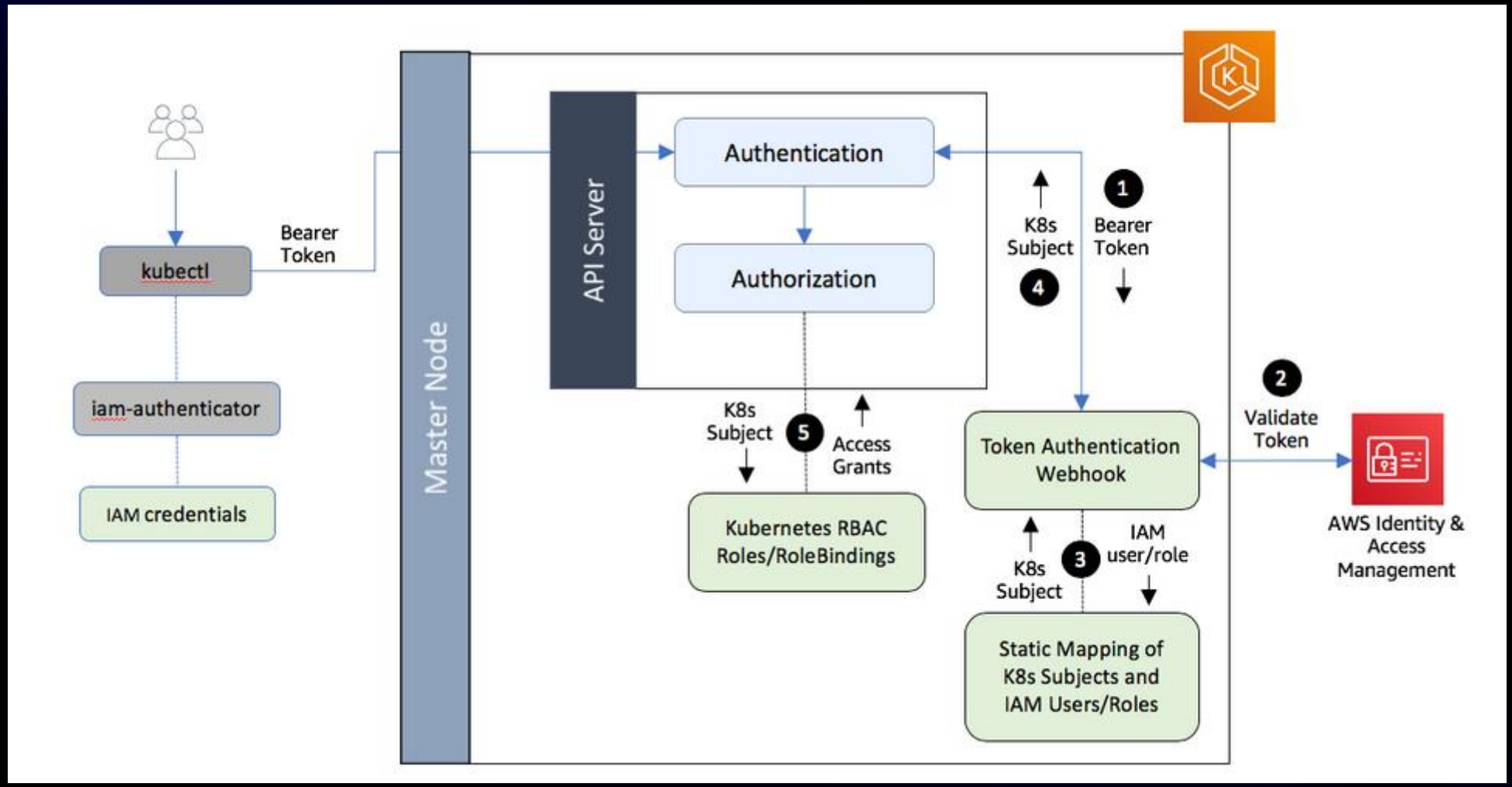
EKS AWS Fargate



EKS: IAM Authentication + kubectl



Authentication and Authorization of a client in an Amazon EKS cluster



https://aws.amazon.com/blogs/containers/kubernetes-rbac-and-iam-integration-in-amazon-eks-using-a-java-based-kubernetes-operator/

How do I manage user access to the cluster?

Authentication

AWS IAM

- No need to maintain separate identity store
- Assume IAM roles for simplified multi-user access control
- Built on open-source AWS IAM Authenticator for Kubernetes
- Integrated with cloud-hosted EKS Kubernetes console

OpenID Connect (OIDC)

- Use your organization's existing identity management system
- Built on open standards
- Use as alternative or in addition to IAM users/roles
- Simplified migration from self-managed Kubernetes and EKS Anywhere

Authorization

Kubernetes role-based access control (RBAC)

Managing Authentication via IAM

Configure the AWS Authenticator configuration map

Consider eksctl to configure it

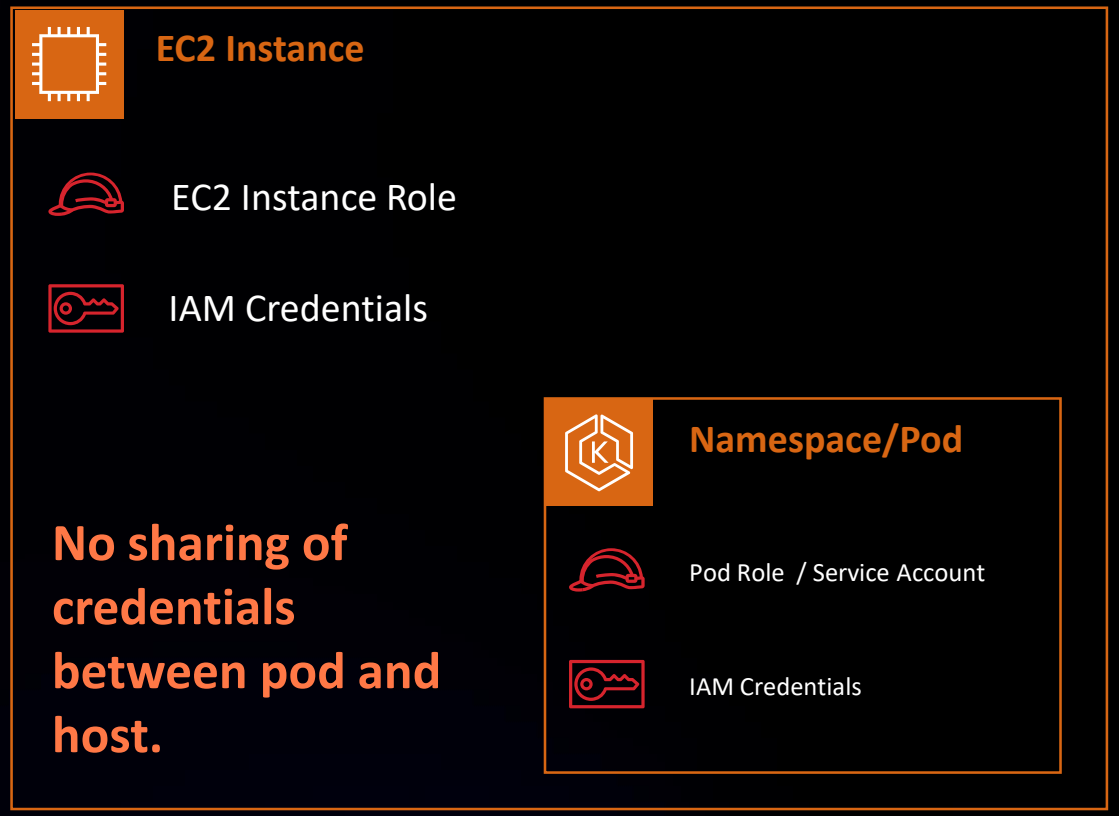
```
apiVersion: v1
kind: ConfigMap
metadata:
  name: aws-auth
  namespace: kube-system
data:
  mapRoles: |
    - rolearn: <node group role>
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes
    - rolearn: arn:aws:iam::123456789012:role/K8s-Admin
      username: iam-admin-{{SessionName}}
      groups:
        - iam:admin
    - rolearn: arn:aws:iam::123456789012:role/K8s-PowerUser
      username: iam-power-user
      groups:
        - iam:power-user
    - rolearn: arn:aws:iam::123456789012:role/K8s-ReadOnly
      username: iam-read-only
      groups:
        - iam:read-only
```

IAM Roles for Service Accounts

Map IAM Roles to Kubernetes Service Accounts

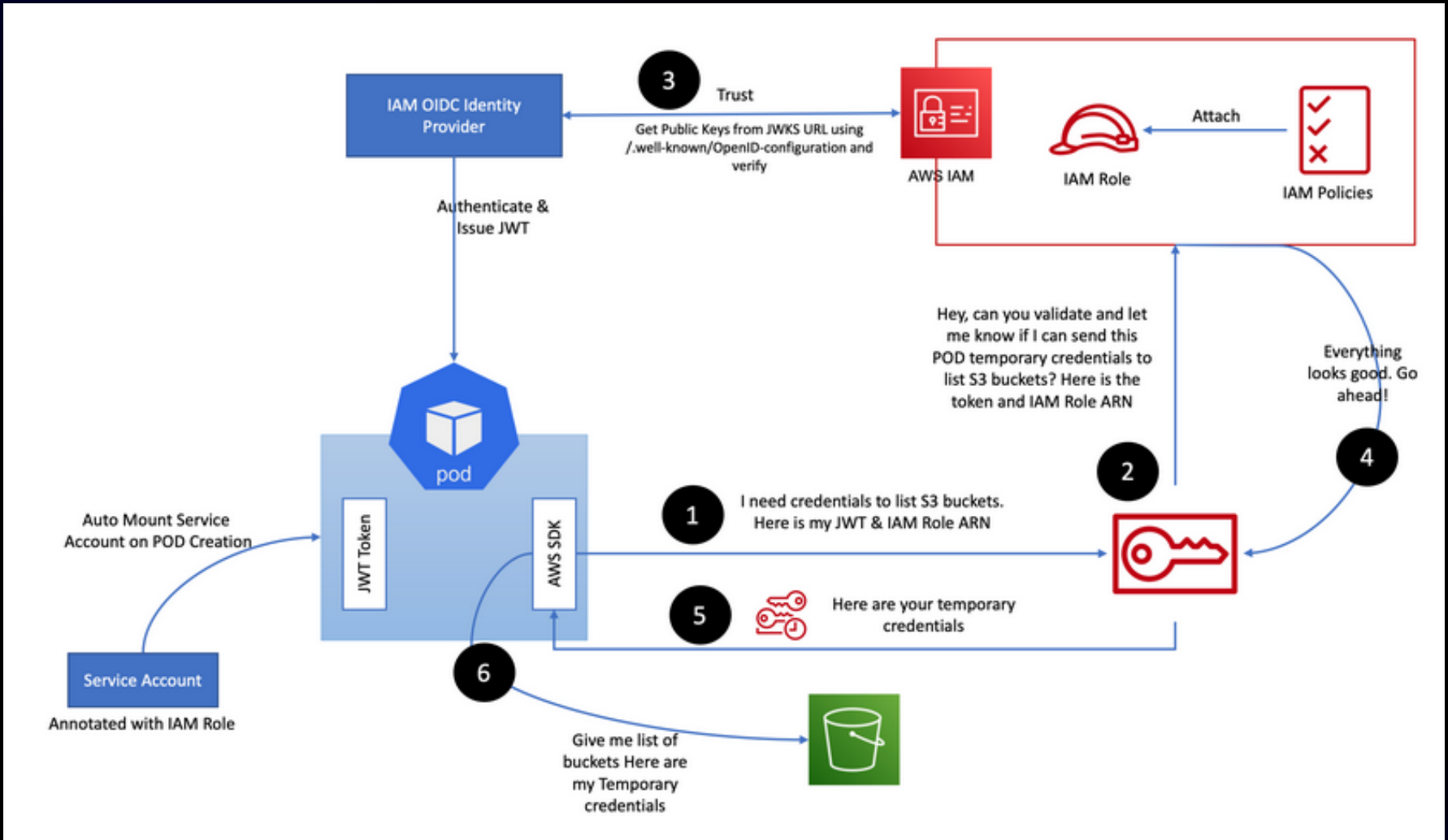
Enable least privilege for Kubernetes Pods

Natively supported by EKS and available as open source



aws/amazon-eks-pod-identity-webhook

Diving into IAM Roles for Service Accounts



https://aws.amazon.com/blogs/containers/diving-into-iam-roles-for-service-accounts/

IAM Roles for Service Accounts

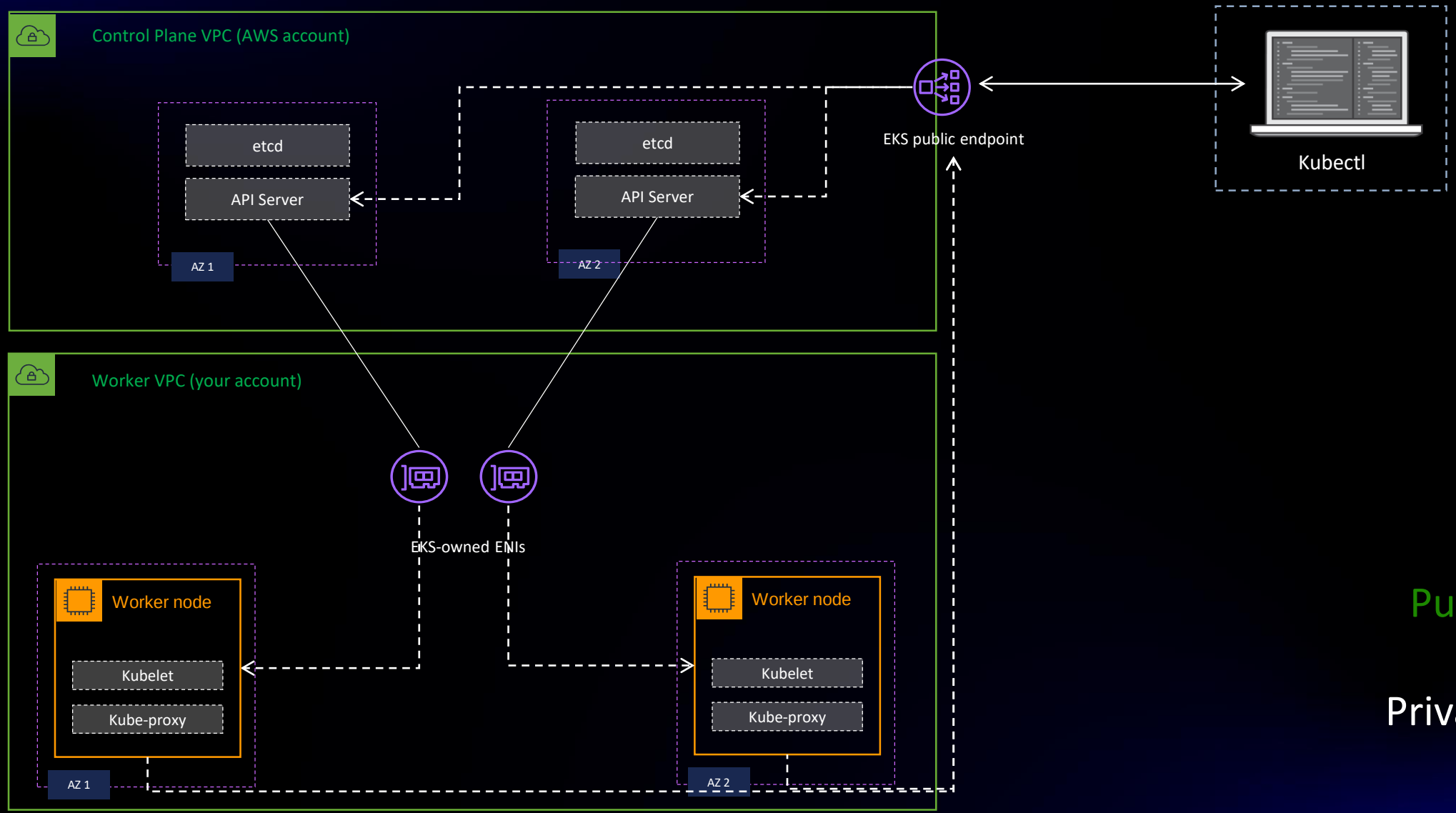
Associate the EKS OIDC server with IAM

```
eksctl utils associate-iam-oidc-provider --region=$AWS_REGION --cluster=$CLUSTER --approve
```

Create Role & Service Account

```
eksctl create iamserviceaccount --cluster $CLUSTER \
  --name ebs-csi-controller-irsa \
  --namespace kube-system \
  --attach-policy-arn $EBS_CNI_POLICY_ARN \
  --override-existing-serviceaccounts \
  --approve
```

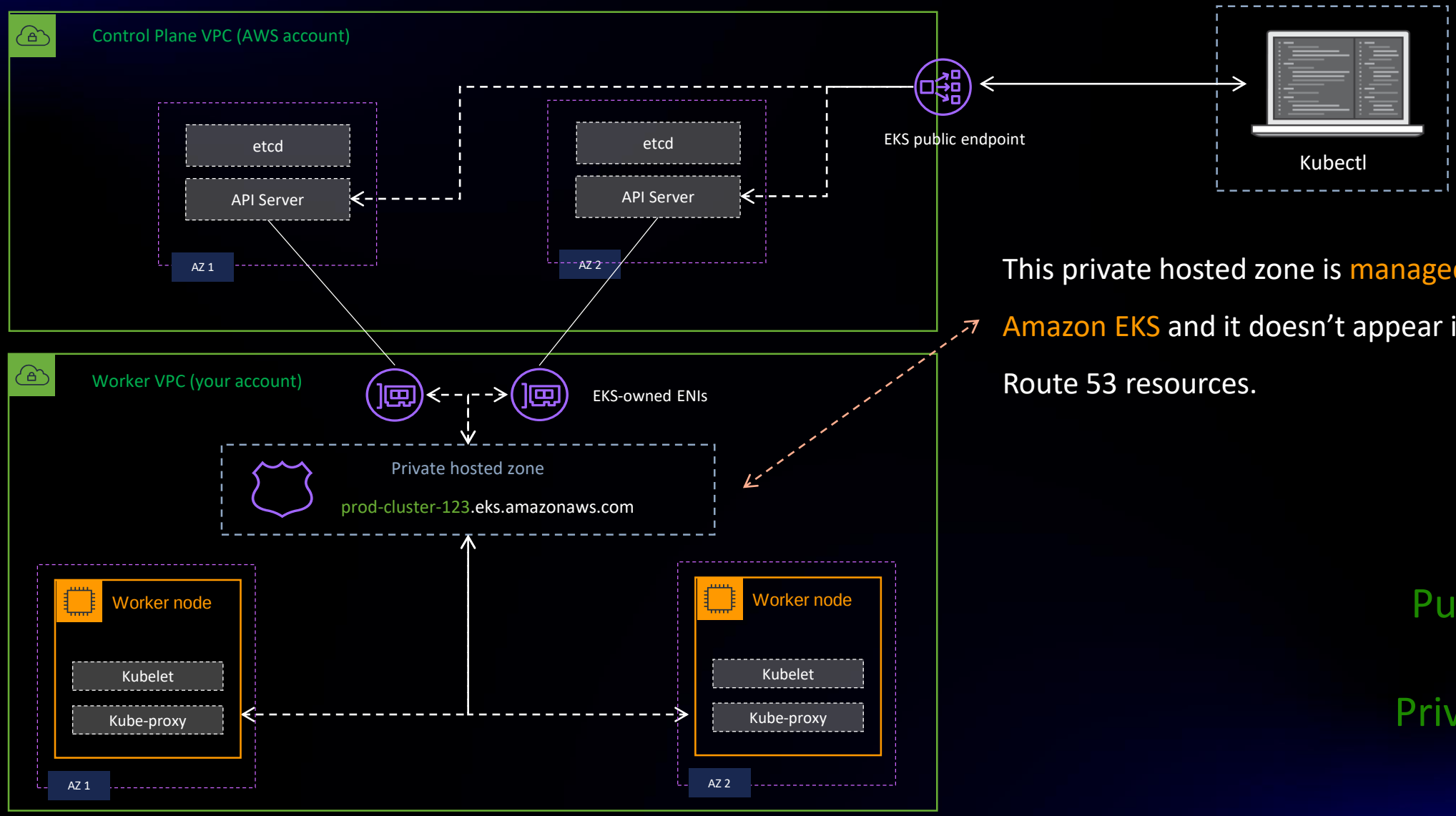
EKS API Endpoints



Public == true

Private == false

EKS API Endpoints

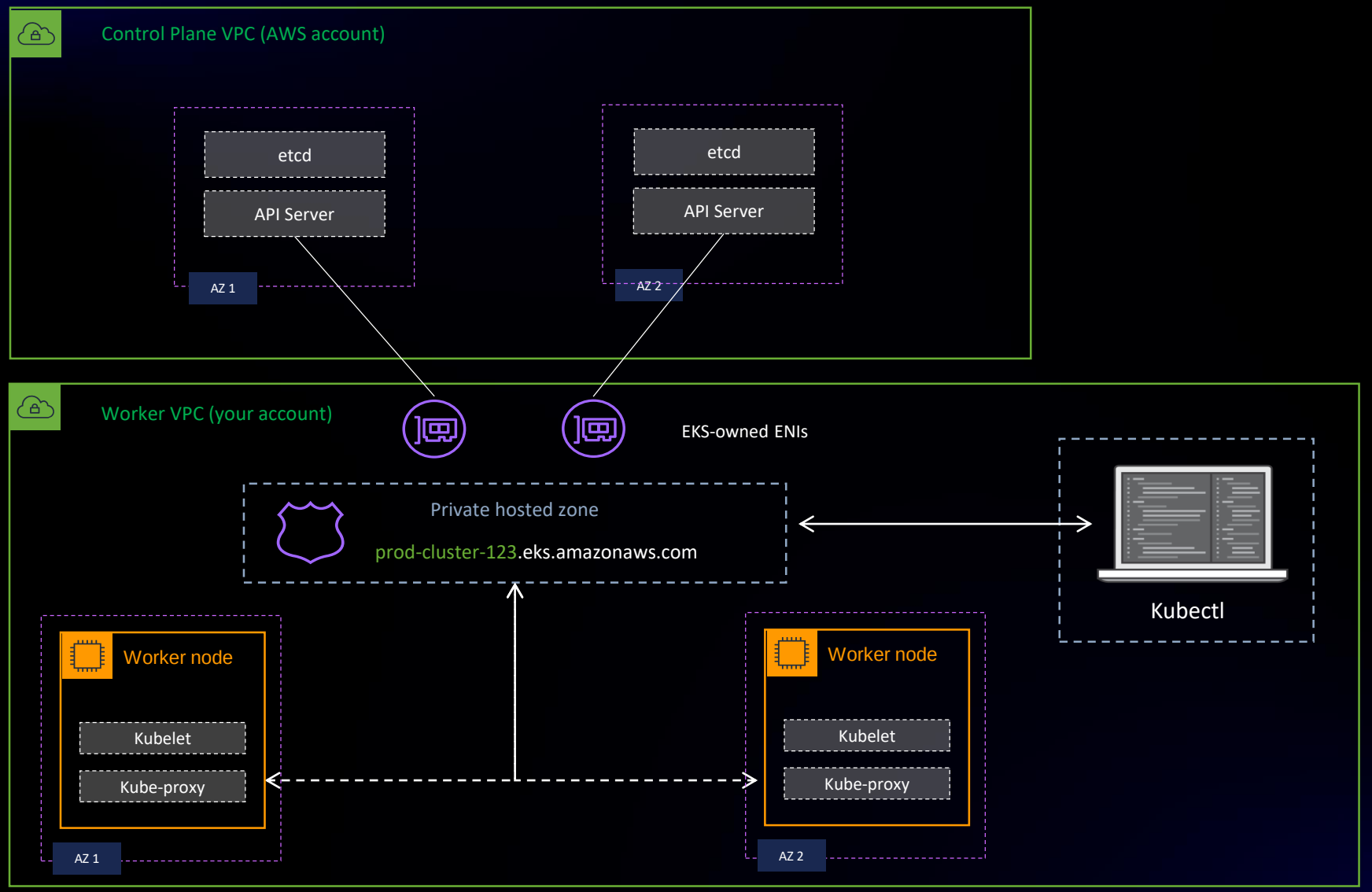


This private hosted zone is managed by Amazon EKS and it doesn't appear in your Route 53 resources.

Public == true

Private == true

EKS API Endpoints

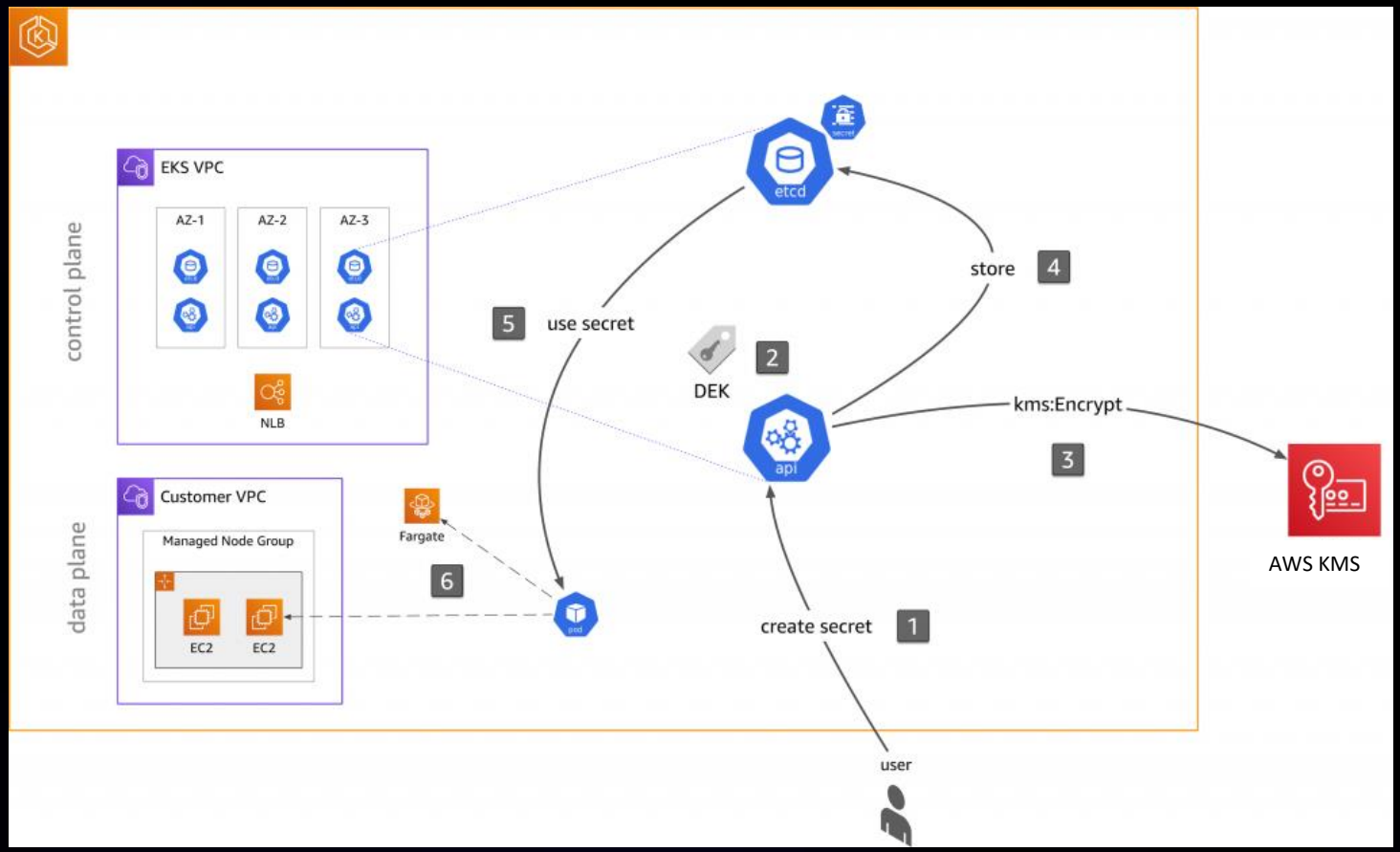


Public == false

Private == true

How do I secure sensitive cluster data?

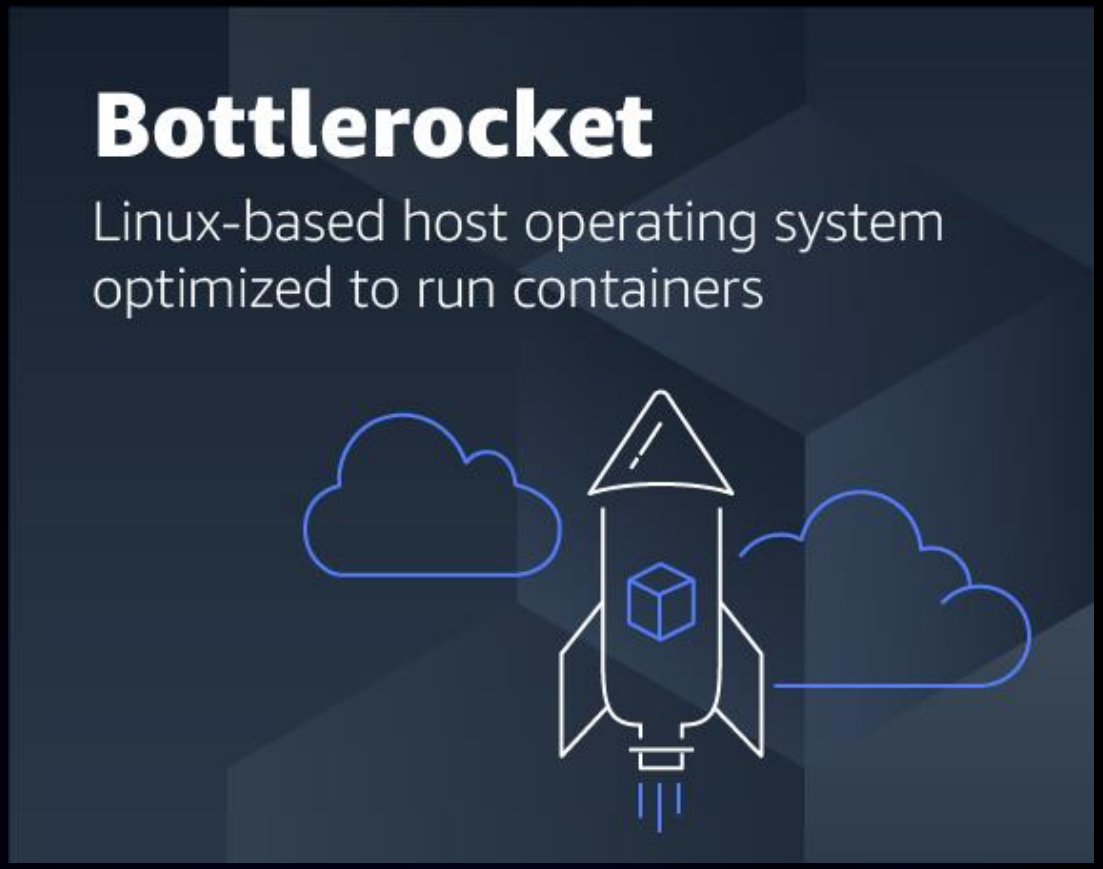
- Encrypt your secrets with AWS Key Management Service (AWS KMS)
- Store secrets outside the cluster and retrieve using AWS Secrets Manager CSI driver
- Use AWS Certificate Manager Private CA Issuer cert-manager plugin to generate TLS certificates



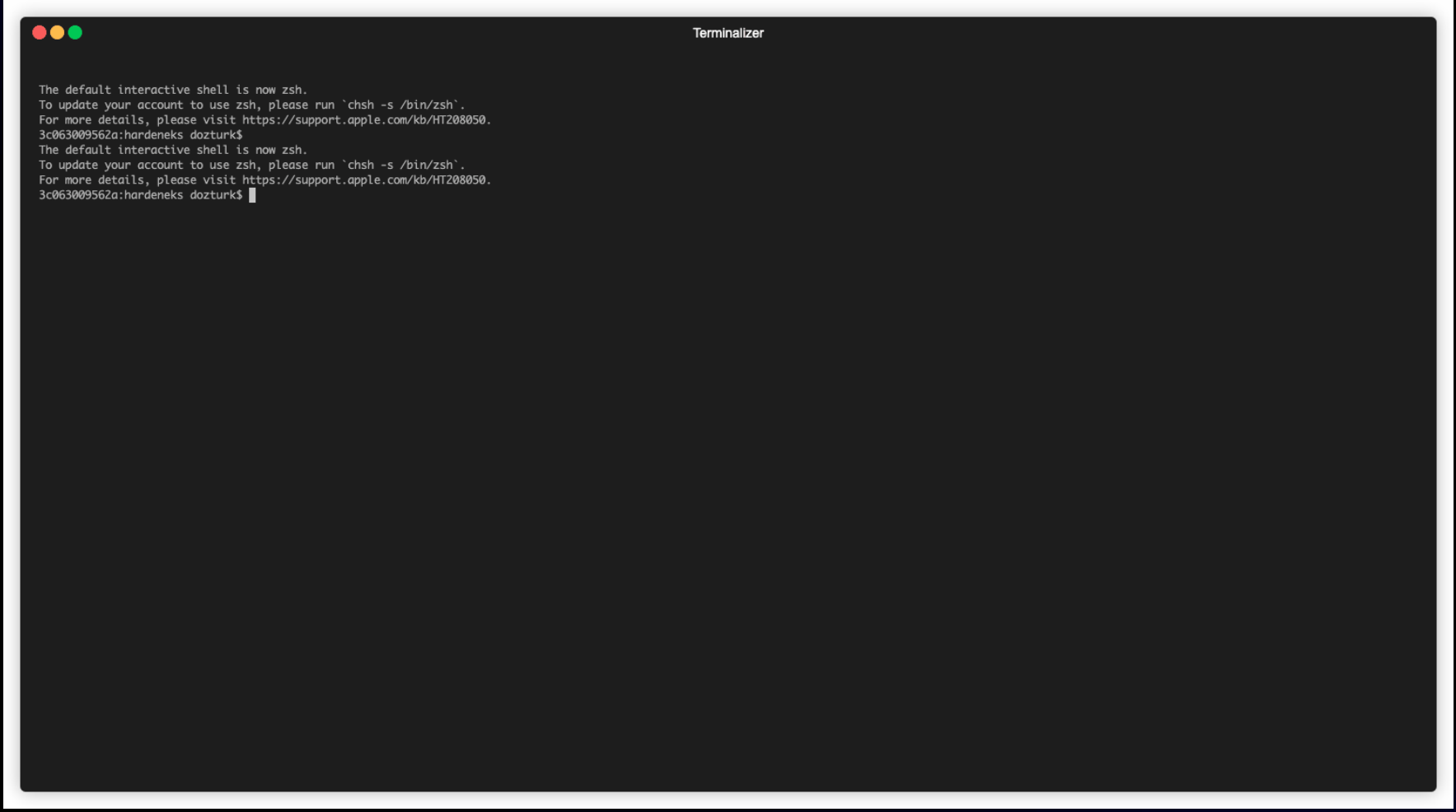
https://aws.github.io/aws-eks-best-practices/security/docs/data/

How do I run secure worker nodes?

1. Use an OS optimized for containers, such as Bottlerocket
2. Treat worker nodes as immutable
3. Use AWS Systems Manager instead of SSH
4. Validate custom AMIs with CIS Amazon EKS Benchmark
5. Or, pass off this responsibility to AWS Fargate



HardenEKS



What is Amazon GuardDuty?

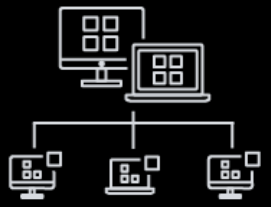
Amazon GuardDuty is a threat detection service that uses **machine learning**, anomaly detection, and **integrated threat intelligence** to identify and prioritize potential threats.

One-step activation

Continuous monitoring of AWS accounts and resources

Global coverage with regional results

Detect known and unknown threats

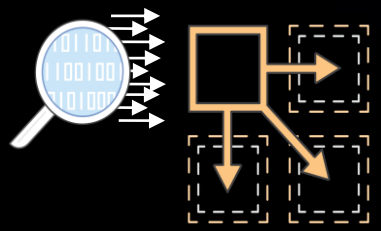


Enterprise-wide consolidation & management

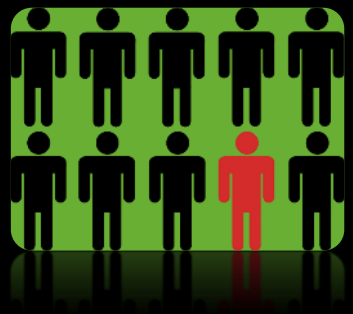
How GuardDuty works

EKS protection in Amazon GuardDuty

With a single click Amazon GuardDuty for Kubernetes protection will continuously monitor Amazon Elastic Kubernetes Service (Amazon EKS) cluster control plane activity by analyzing the EKS audit logs (across all accounts).

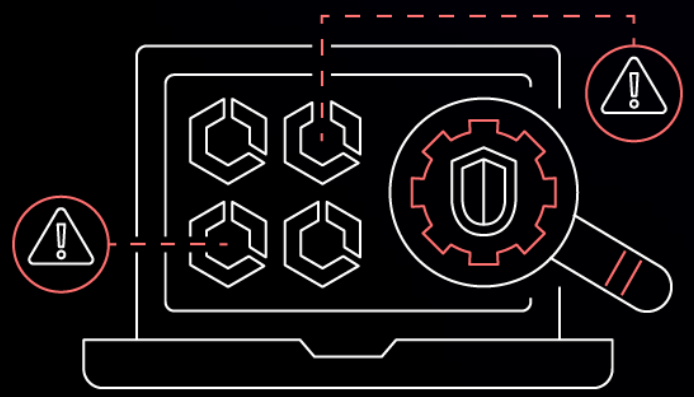


Using threat intelligence it will detect and report findings for Amazon EKS clusters that are accessed by known malicious actors or from Tor nodes, API operations performed by anonymous users that might indicate a misconfiguration, and misconfigurations that can result in unauthorized access to Amazon EKS clusters.



By using machine learning (ML) models, GuardDuty can identify patterns consistent with privilege-escalation techniques, such as a suspicious launch of a container with root-level access to the underlying Amazon Elastic Compute Cloud (Amazon EC2) host

Introducing Amazon GuardDuty EKS Runtime Monitoring



- Protect your Amazon EKS workloads with a new GuardDuty security agent that adds visibility into individual Kubernetes **container runtime activities**, such as file access, process execution, and network connections.
- **Continuously monitor** all Amazon EKS clusters across your organization with a few steps in the GuardDuty console.
- **Over two dozen new threat detections** leveraging high fidelity system-level events, tailored for known container attack techniques, and providing container-level context.

Detect threats at each layer of Kubernetes deployment on Amazon EKS

VPC Flow Logs and DNS logs continue to be monitored: defense in depth

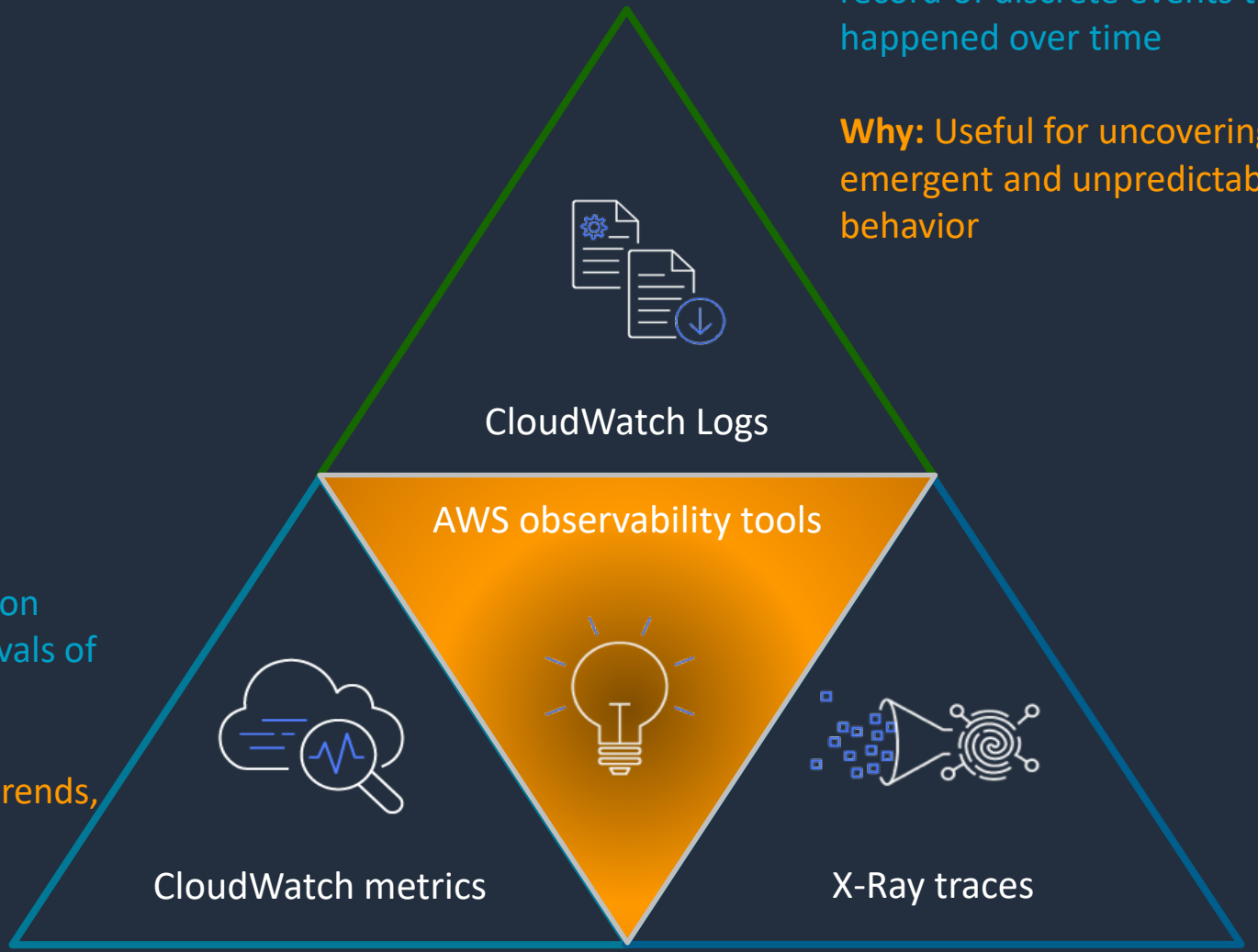
Observability

What is Observability?

AWS Observability Tools

What: Immutable, timestamped record of discrete events that happened over time

Why: Useful for uncovering emergent and unpredictable behavior



What: Numeric representation of data measured over intervals of time

Why: Useful for identifying trends, mathematical modeling, and prediction

What: Representation of a series of related distributed events that encode the end-to-end request flow through a distributed system

Why: Provides visibility into both the path traversed by a request as well as the structure of a request

Metrics

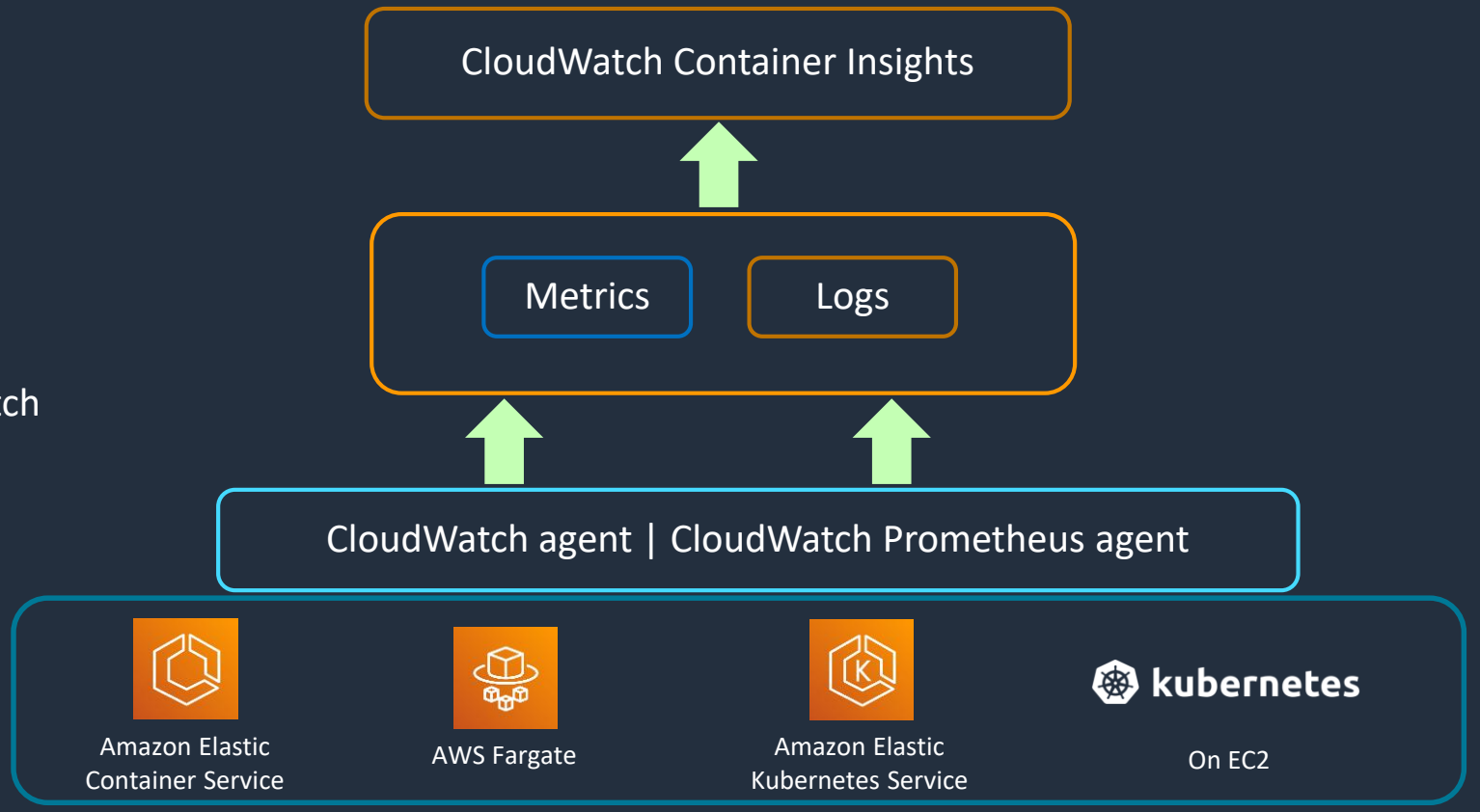
CloudWatch Container Insights

Built-in dashboards to see performance metrics for cluster resources at different levels

Out of the box dashboards for popular workloads such as AppMesh, Java/JMX, NGINX, HAProxy etc

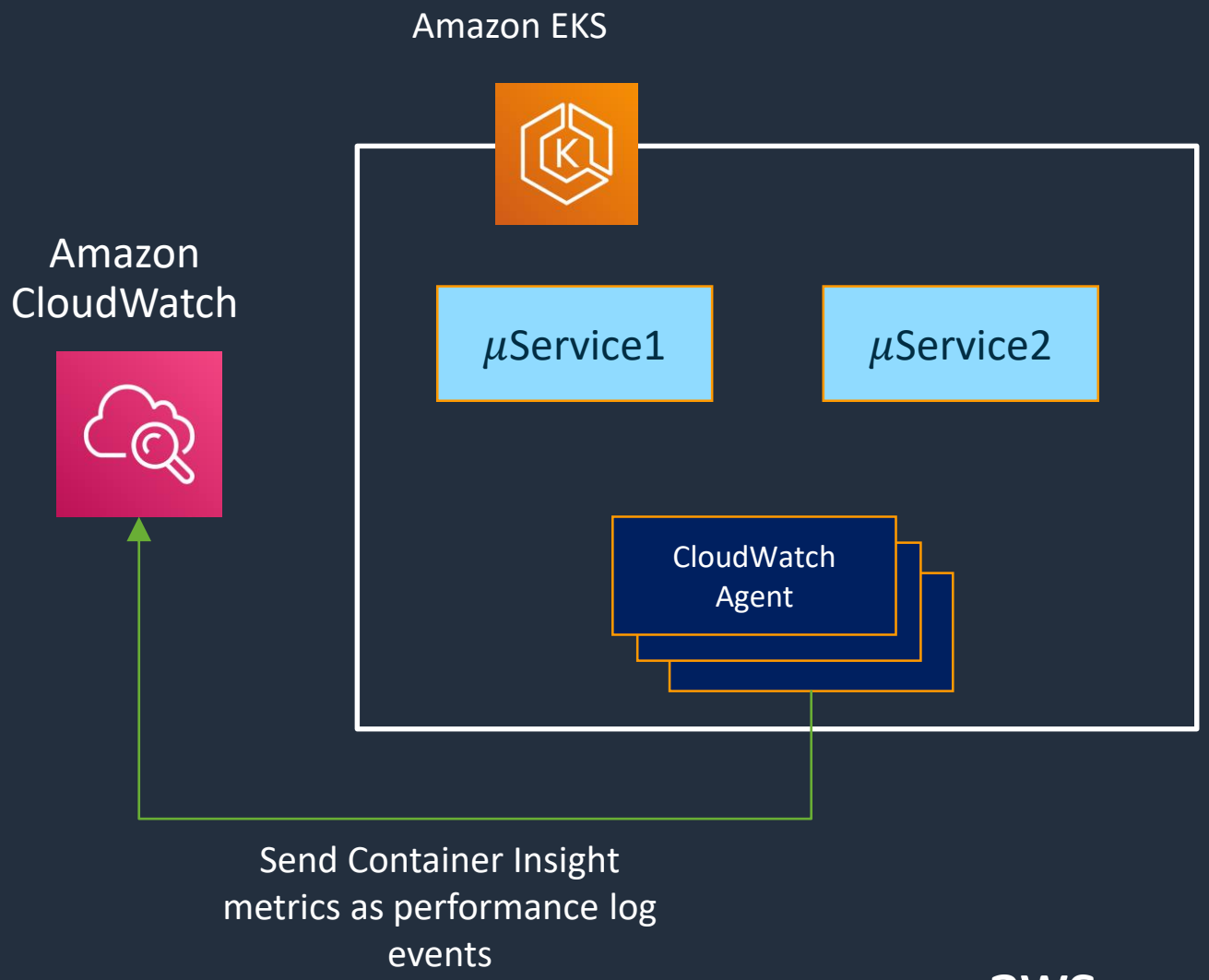
Collect Prometheus metrics from workloads

Amazon CloudWatch



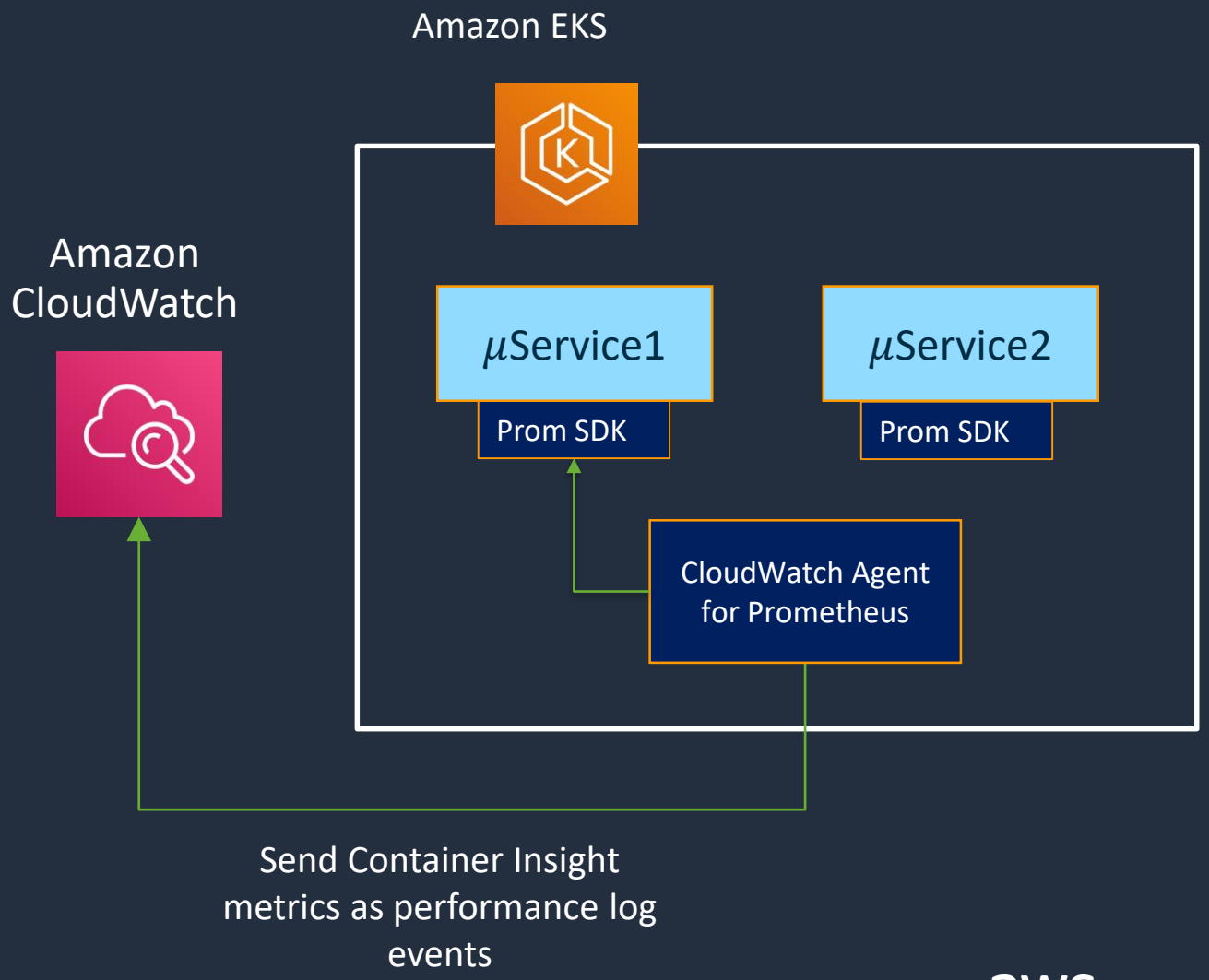
CloudWatch Container Insights

- Collect, aggregate and summarize metrics and logs from containerized applications
- Collect collect instance-level metrics such as CPU, memory, disk and network usage
- Operational data collected as performance log events with EMF from which metrics are extracted



CloudWatch Container Insights for Prometheus

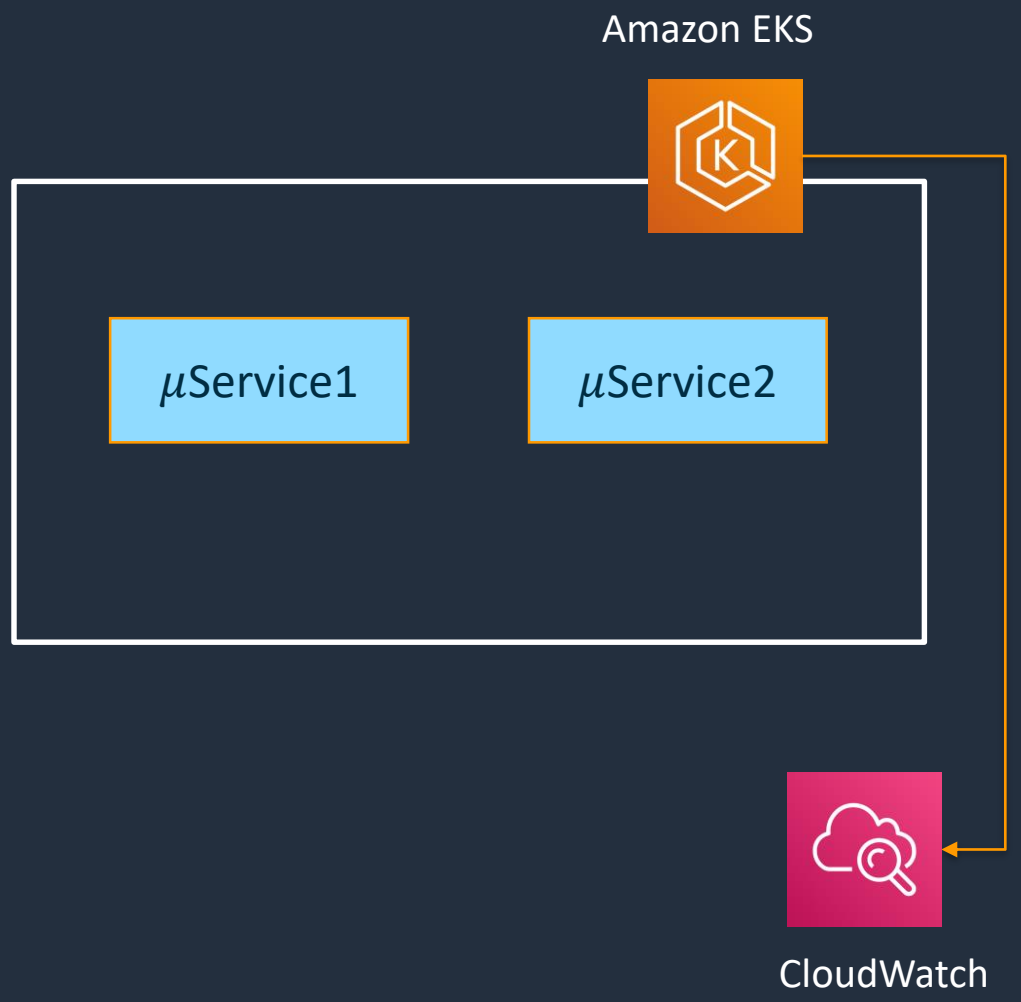
- Collect, aggregate and summarize metrics and logs from containerized applications
- Collect collect instance-level metrics such as CPU, memory, disk and network usage
- Operational data collected as performance log events with EMF from which metrics are extracted



Logs

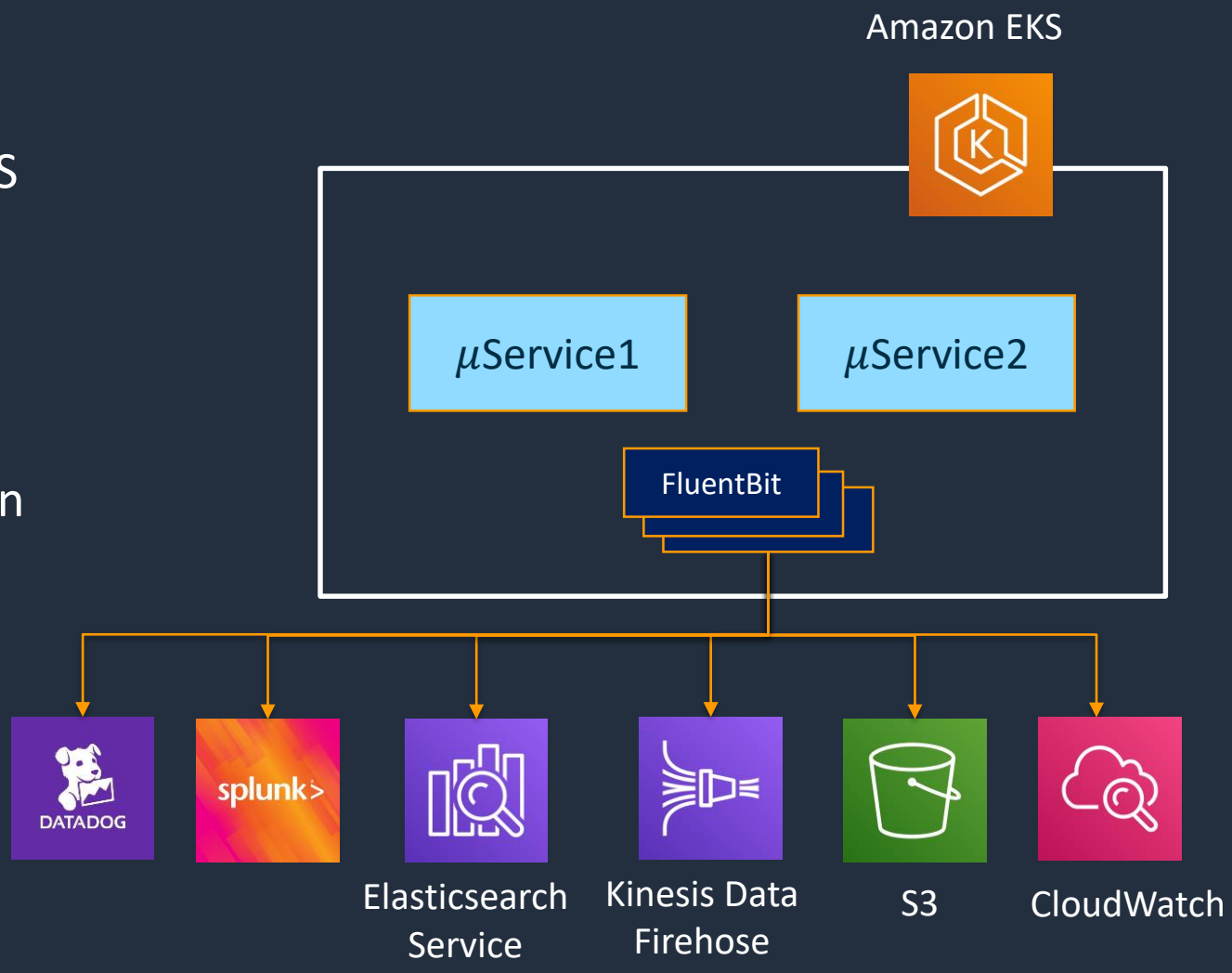
CloudWatch Logs for Amazon EKS

- Audit and diagnostic logs from Amazon EKS Control Plane can be sent to CloudWatch



CloudWatch Logs for Amazon EKS

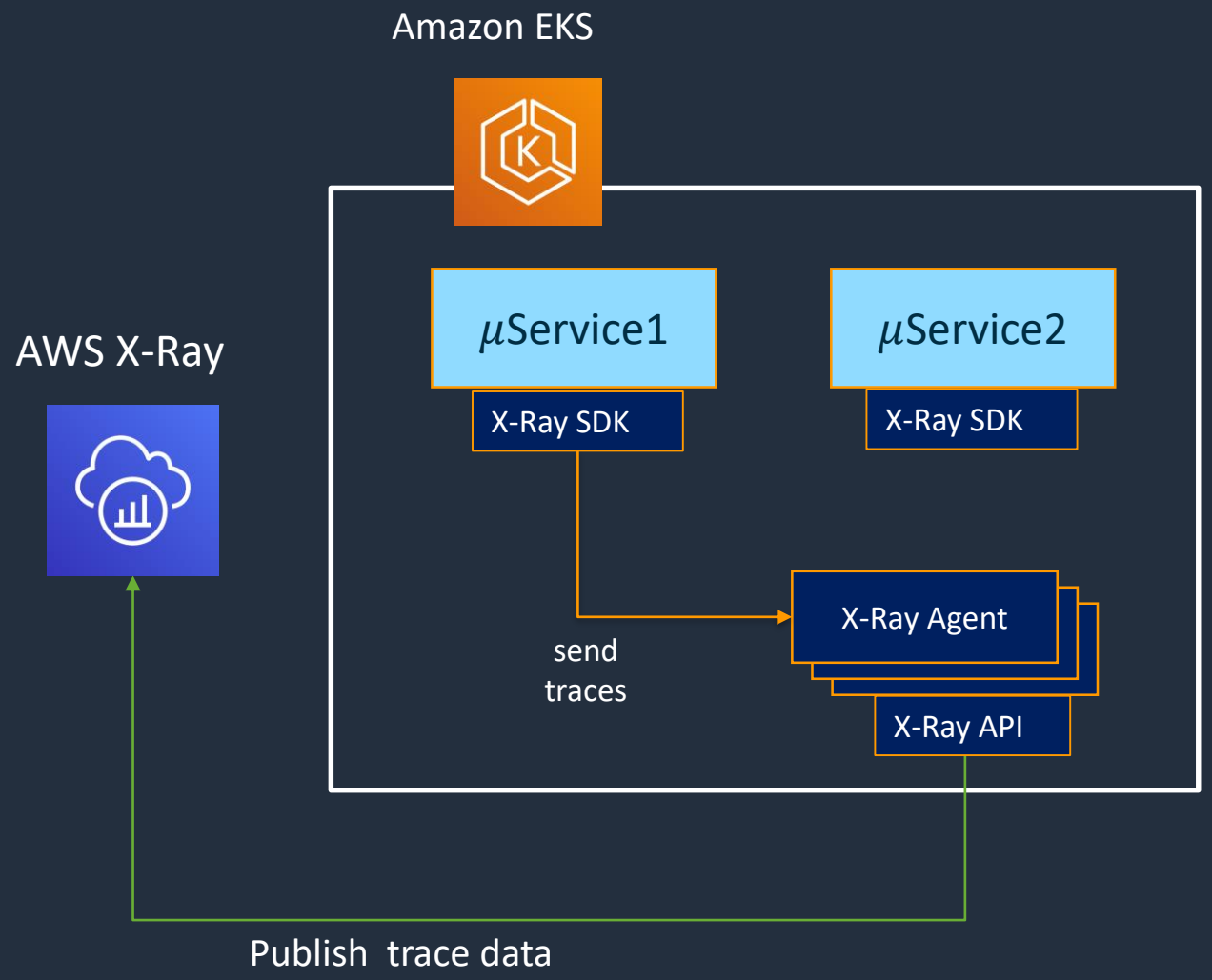
- Audit and diagnostic logs from Amazon EKS Control Plane can be sent to CloudWatch
- Use FluentBit to send application logs to destination of your choosing
- FluentBit-based logging is also supported in EKS on Fargate



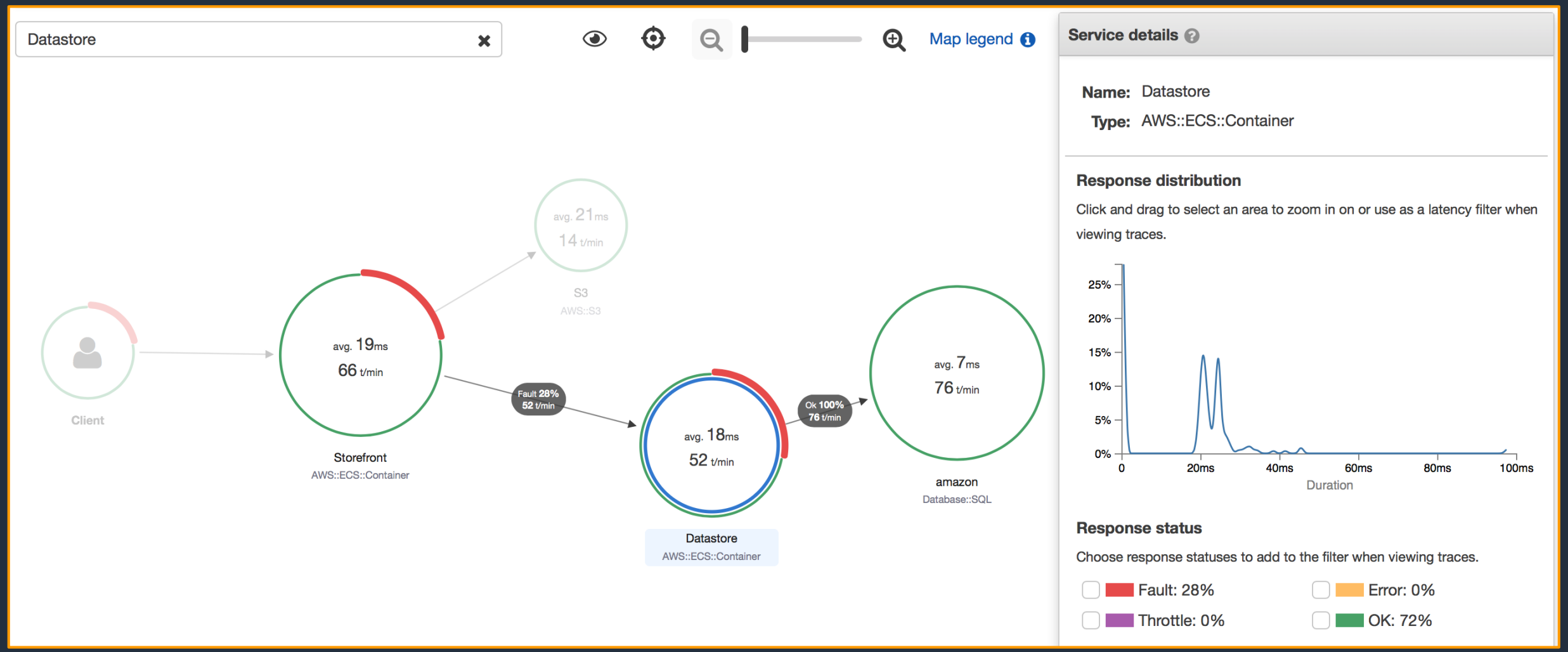
Traces

AWS X-Ray for Amazon ECS/EKS

- Microservices instrumented with X-Ray SDK send segment data to X-Ray agent in the cluster
- X-Ray agent buffers segments in a queue and uploads them to X-Ray in batches
- X-Ray groups segments that have a common request into traces which are used to generate a service graph that provides a visual representation of your application



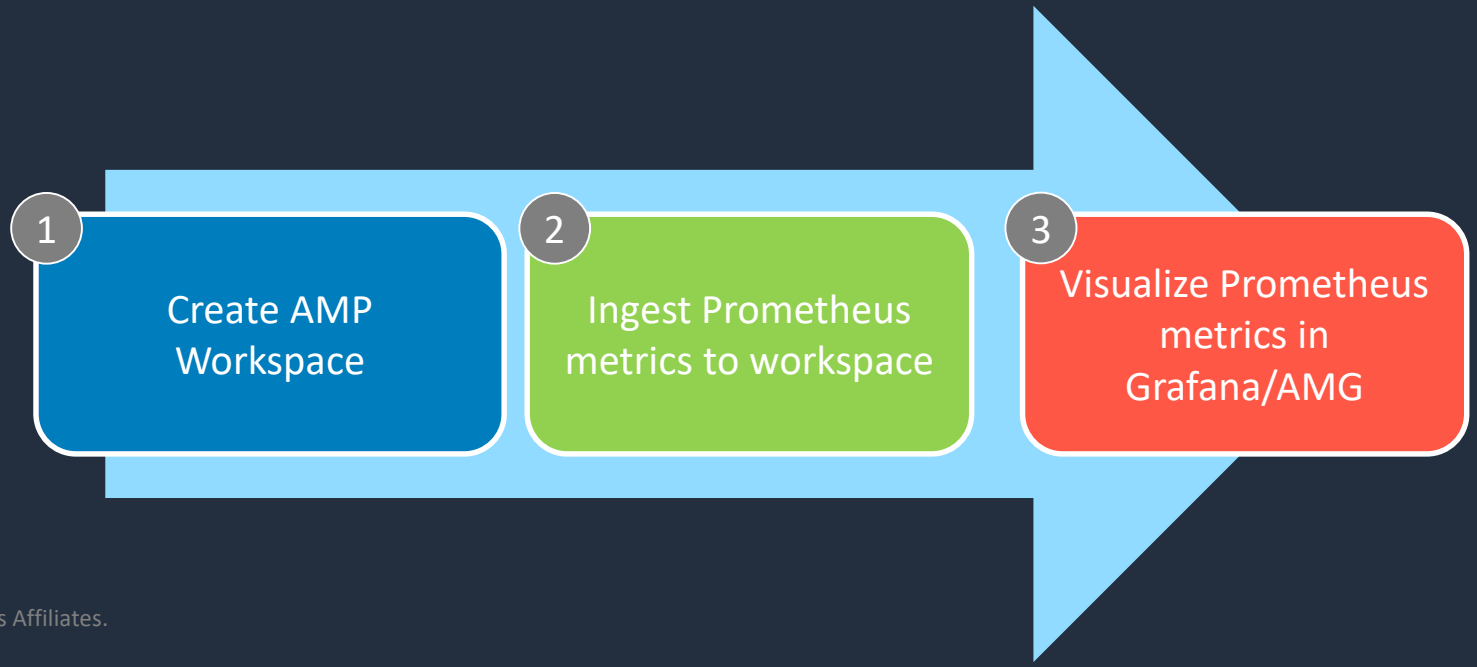
Identify Performance Bottlenecks



Open Source Observability Tools

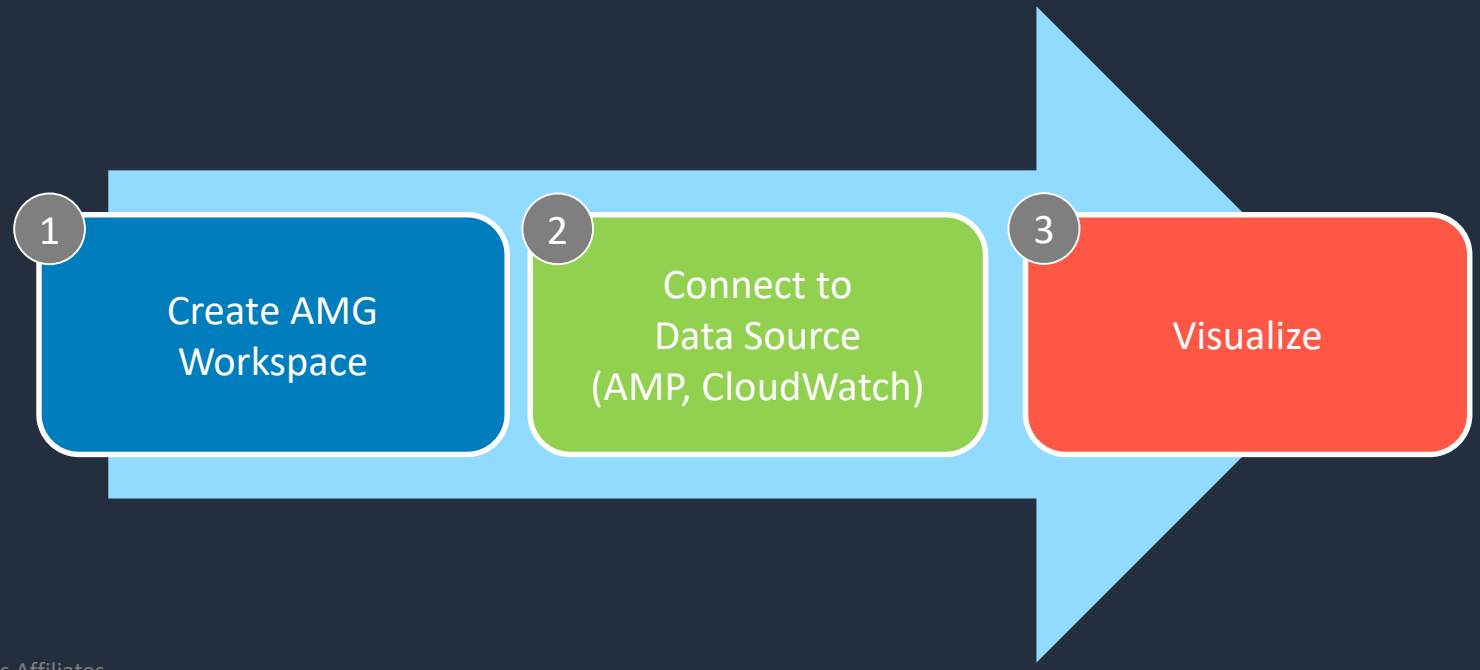
Amazon Managed Service for Prometheus

- Serverless Prometheus-compatible service for metrics to securely monitor container environments at scale
- Fully managed, secure, and highly available using multi-AZ deployments
- Use the same open source Prometheus data model and query language
- Improved scalability, availability, and security without managing the underlying infrastructure

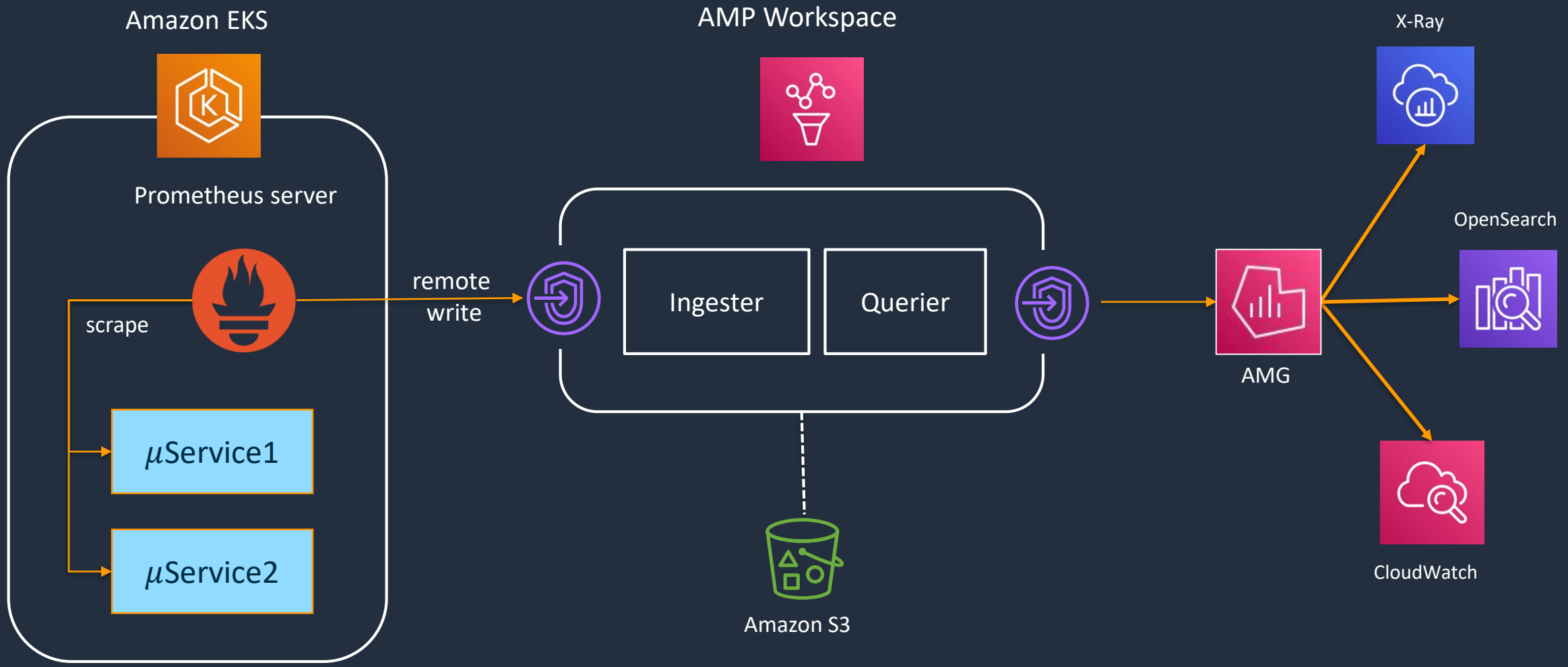


Amazon Managed Grafana

- Scalable, secure and highly available fully-managed Grafana service
- Analyze, monitor, and alarm across multiple data sources; native AWS as well as 3rd party
- Native integration with multiple AWS Services for enterprise-ready security
- Easily upgrade to Grafana Enterprise from AWS marketplace



Observability with AMP & AMG

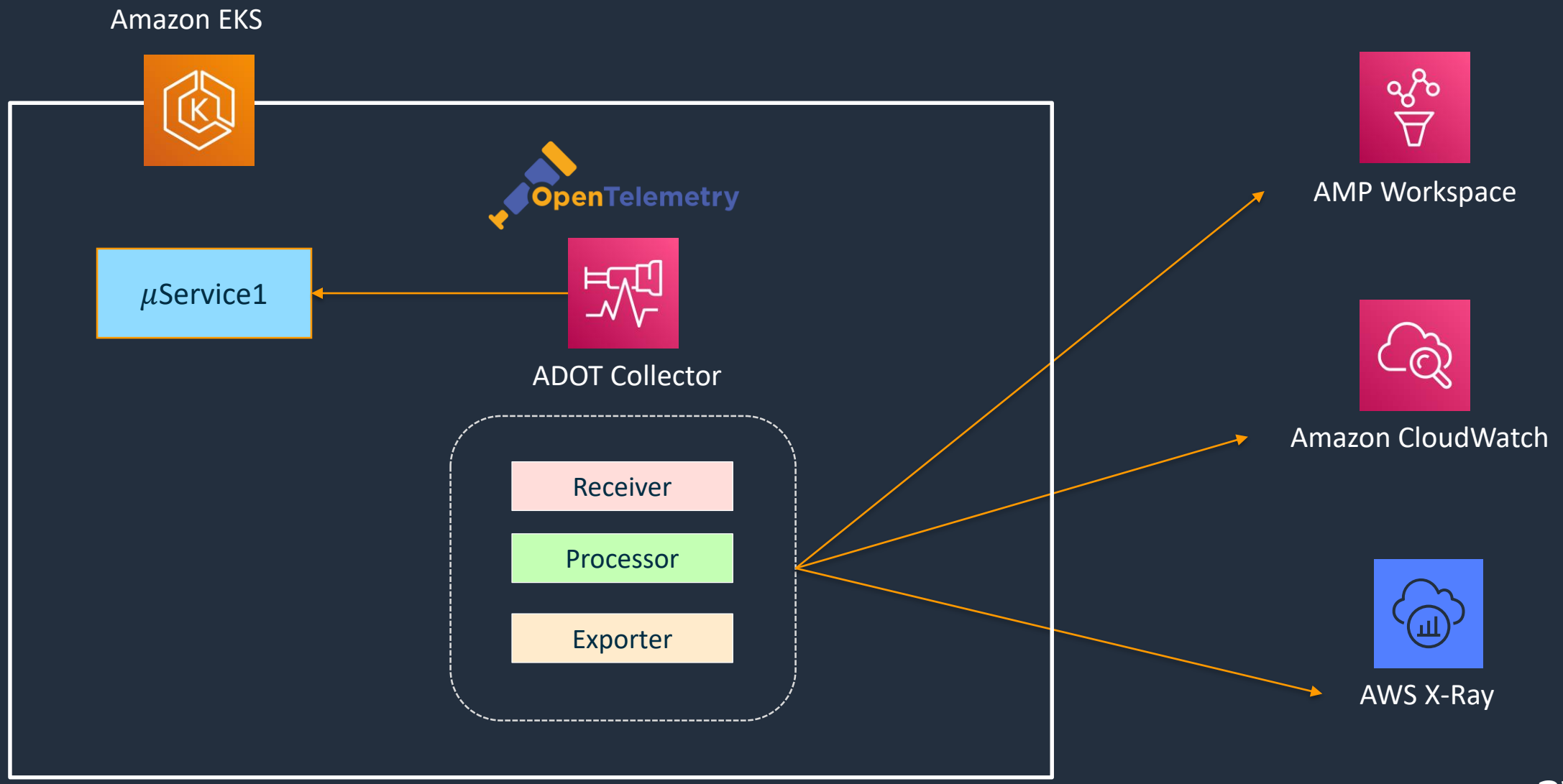


AWS Distro for Open Telemetry (ADOT)

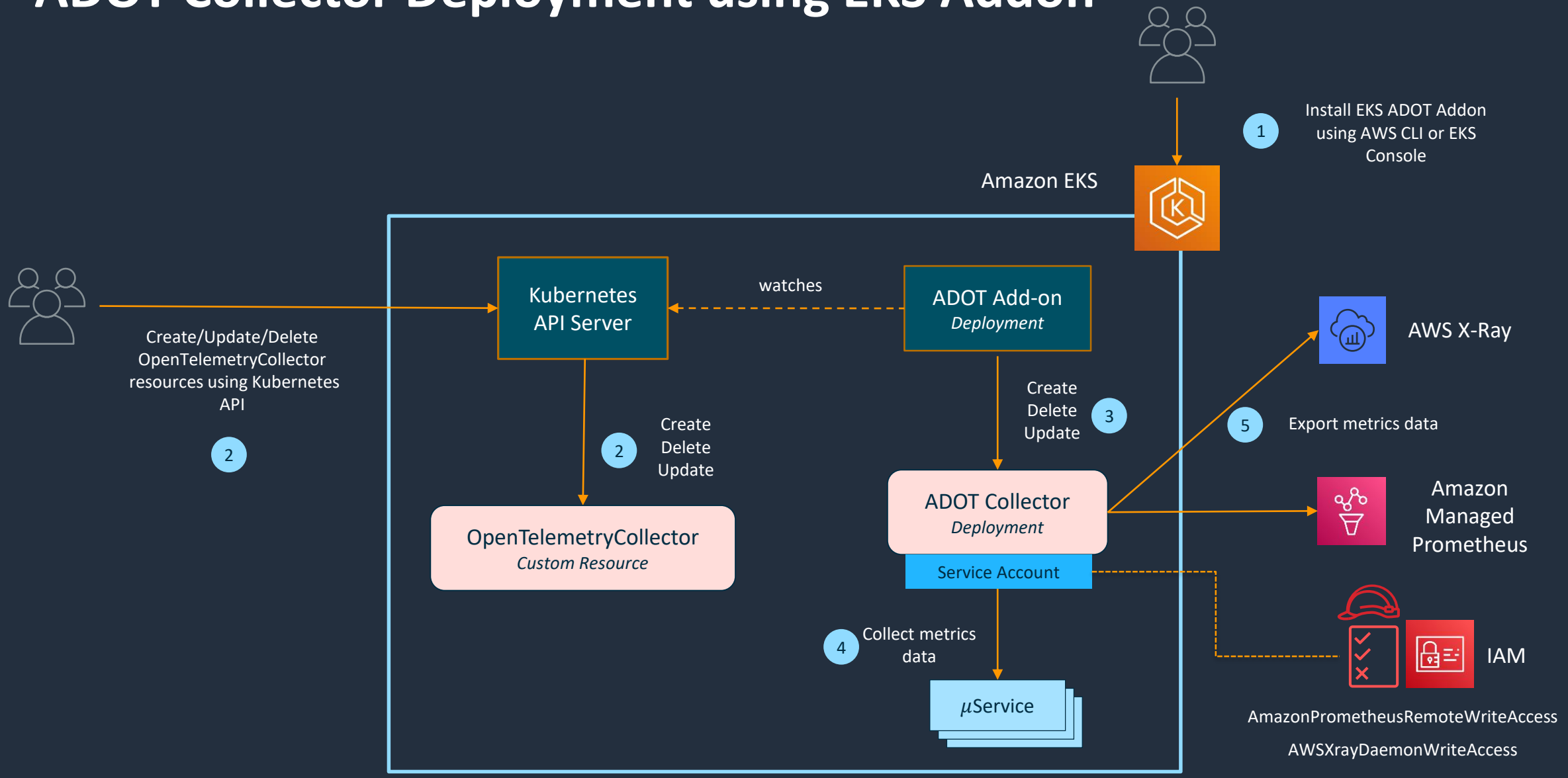
- OpenTelemetry provides open source APIs, libraries, and agents to collect distributed traces and metrics for application monitoring.

- AWS Distro for OpenTelemetry
  - Secure, production-ready AWS-supported distribution of OpenTelemetry project
  - Instrument your applications just once to send correlated metrics and traces to multiple monitoring solutions
  - Use auto-instrumentation agents to collect traces without changing your code

AWS Distro for OpenTelemetry (ADOT)



ADOT Collector Deployment using EKS Addon



Where can I learn more?

EKS Best Practices Guide

- Open-sourced best practices guide
- Check back often for new chapters

aws.github.io/aws-eks-best-practices

Resources

[EKS Best Practices Guide]

[Amazon EKS Compute Blog]

[Amazon EKS User Guide]

[Harden EKS]

[EKS Node Viewer]

[Kubernetes Operational View]

Thank You

Please take the survey: https://pulse.buildon.aws/survey/N9LLMZSA