Kubernetes Overview

안준환

Solutions Architect
Amazon Web Services

Agenda

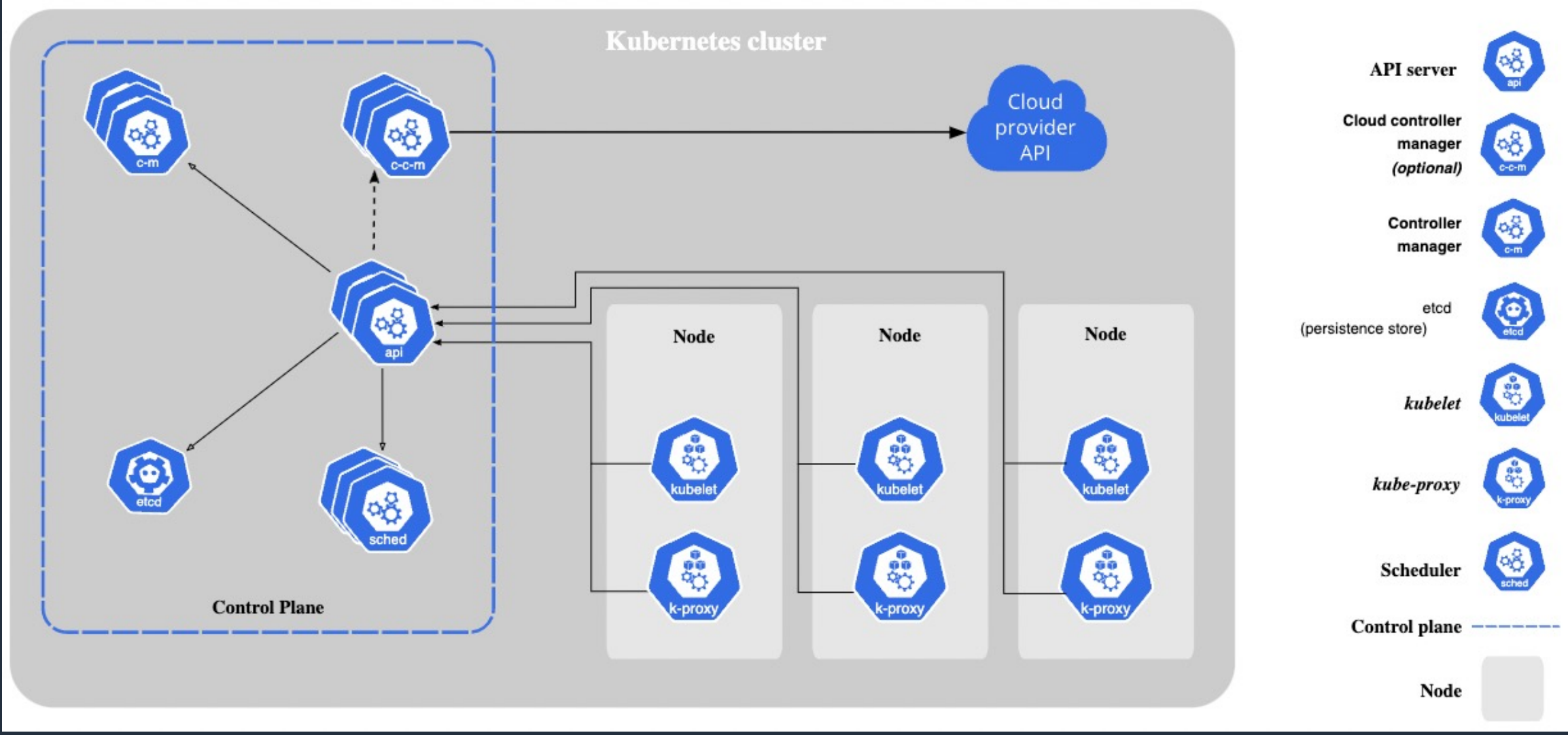
- Overview, Architecture
- Object & Controller
- Pod
- Pod Scheduling
- Network
- App Configuration, Secret
- Security

Kubernetes
Overview, Architecture

Kubernetes

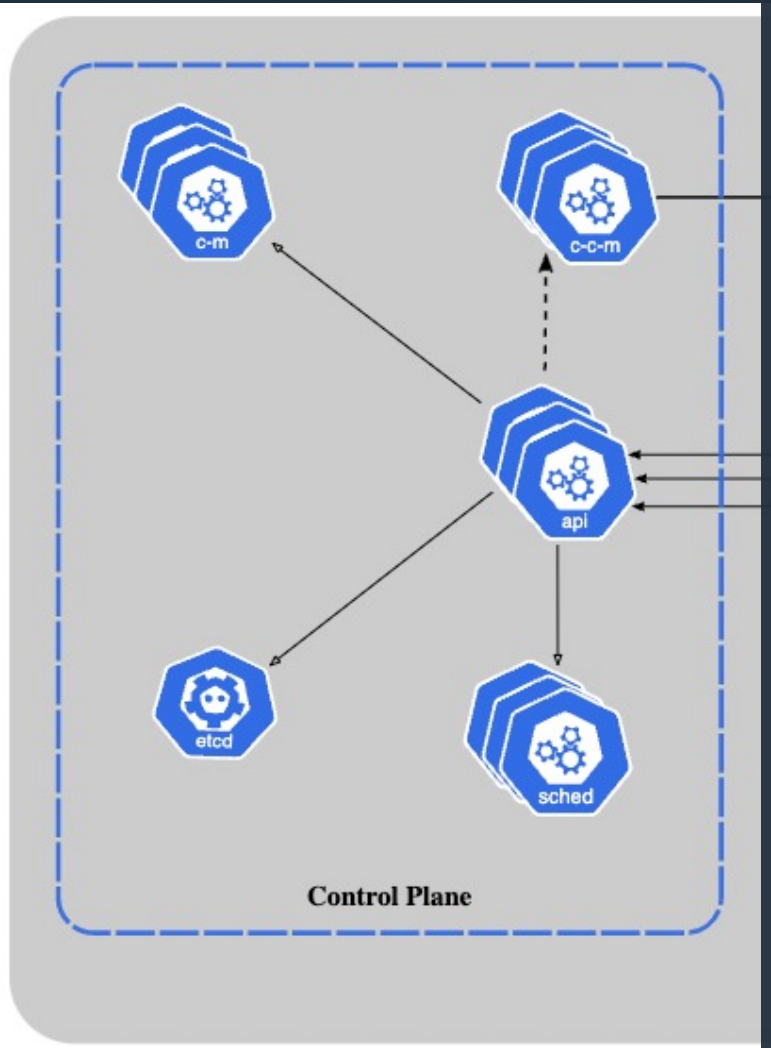
- Container orchestrator
  - Container management
  - Service discovery & load balancing
  - Automated rollouts & rollbacks
  - Self-healing
  - Storage orchestration
  - Secret & configuration management

Kubernetes Architecture



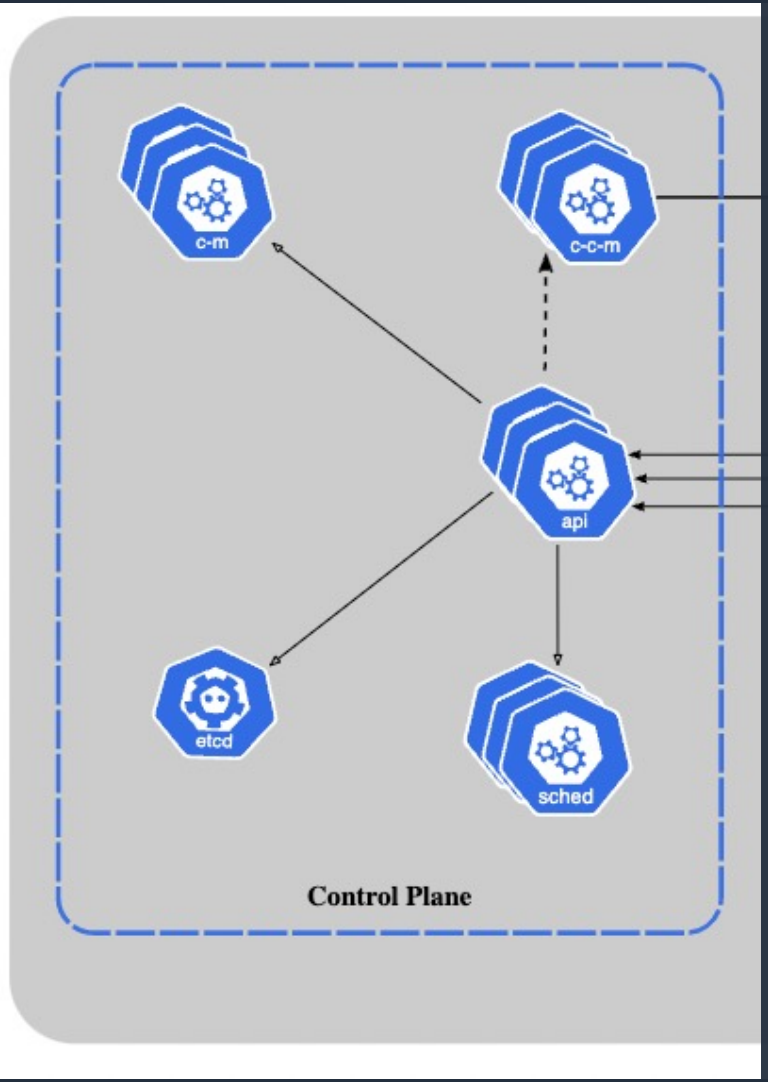
https://kubernetes.io/docs/concepts/overview/components/

Kubernetes Architecture / Control Plane



- etcd
  - Strong Consistent, 분산 Key-Value Store
  - K8s API Server의 Data Store 역할 수행
- K8s API Server
  - etcd를 기반으로 Kubernetes 관련 대부분의 정보를 Object 단위로 관리
  - K8s Client, Component의 요청 처리
  - 인증/인가 수행

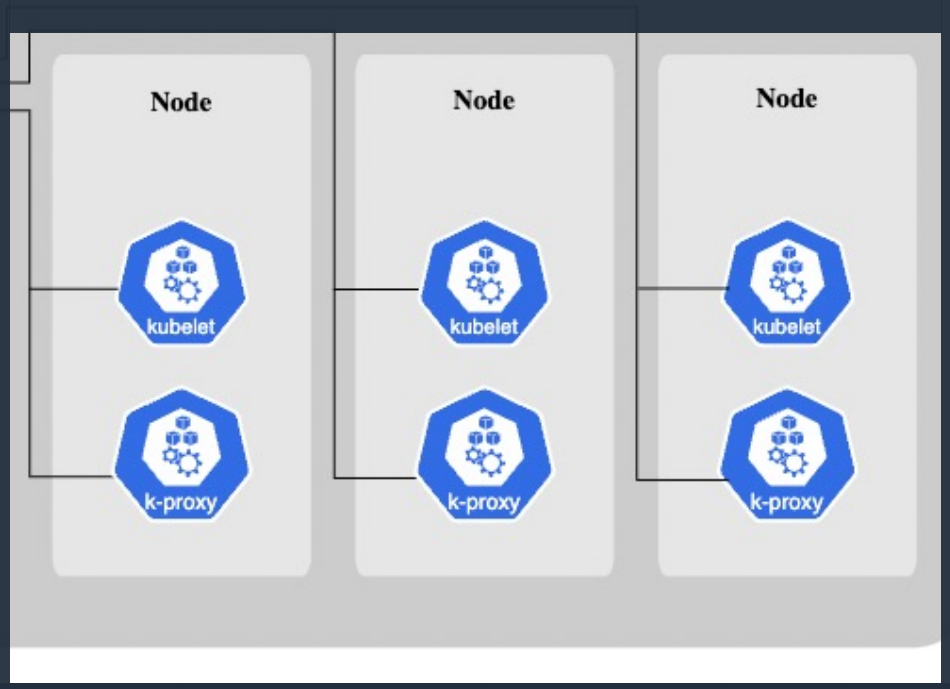
Kubernetes Architecture / Control Plane



- K8s Controller Manager
  - Controller의 집합
  - Controller는 K8s API Server로부터 Object 정보를 획득하여 Object 제어 역할 수행
- Cloud Controller Manager
  - Cloud Provider의 Resource 제어를 수행하는 Controller들의 집합
  - Ex) AWS ELB Controller, AWS EFS Controller
- Scheduler
  - 생성된 Pod를 Node에 배치하는 역할 수행

Kubernetes Architecture / Data Plane

- kubelet
  - Node의 상태 보고, Pod 관리 수행
- kube-proxy
  - K8s Service를 활용하여 Pod 사이의 Traffic Load Balancing 수행
  - Linux Kernel의 iptables, IPVS를 활용하여 동작



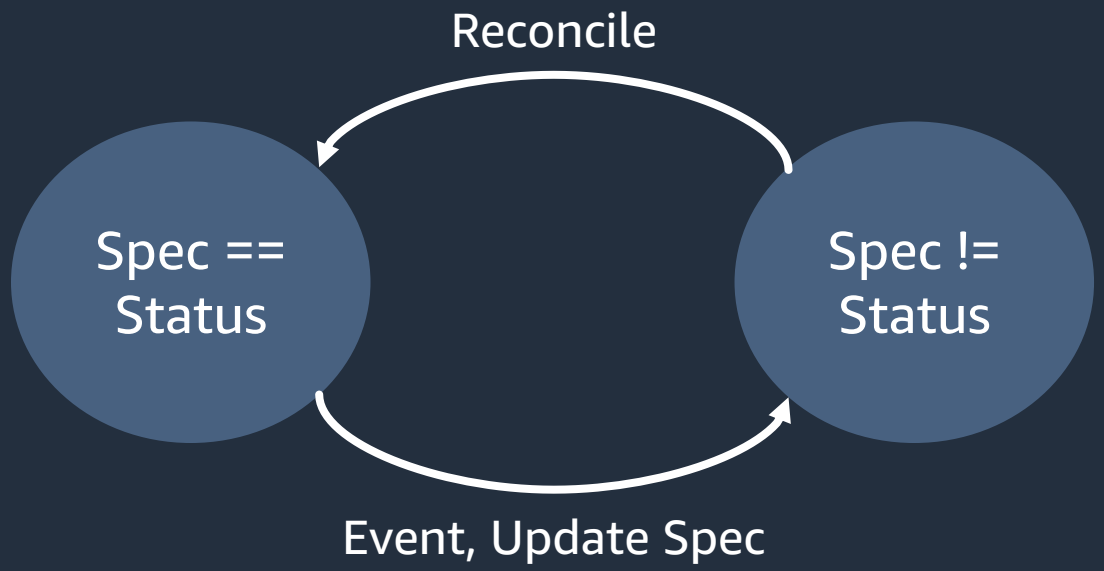
Kubernetes Object & Controller

Kubernetes Object

Kubernetes Objects

- Controller에 의해서 관리되는 최소 단위
- Object 구성 요소
  - Spec : Desired State
  - Status : Current State
- Namespaced Object
  - Namespace에 포함되는 Object.
  - Ex) Pod, Service, Deployment
- Non-namespaced Object
  - Namespace에 포함되지 않는 Object
  - Ex) Node, ClusterRole, ClusterRoleBinding
- CRD (Custom Resource Definition) 기능을 통해서 자신만의 Object 정의 가능

Kubernetes Controller

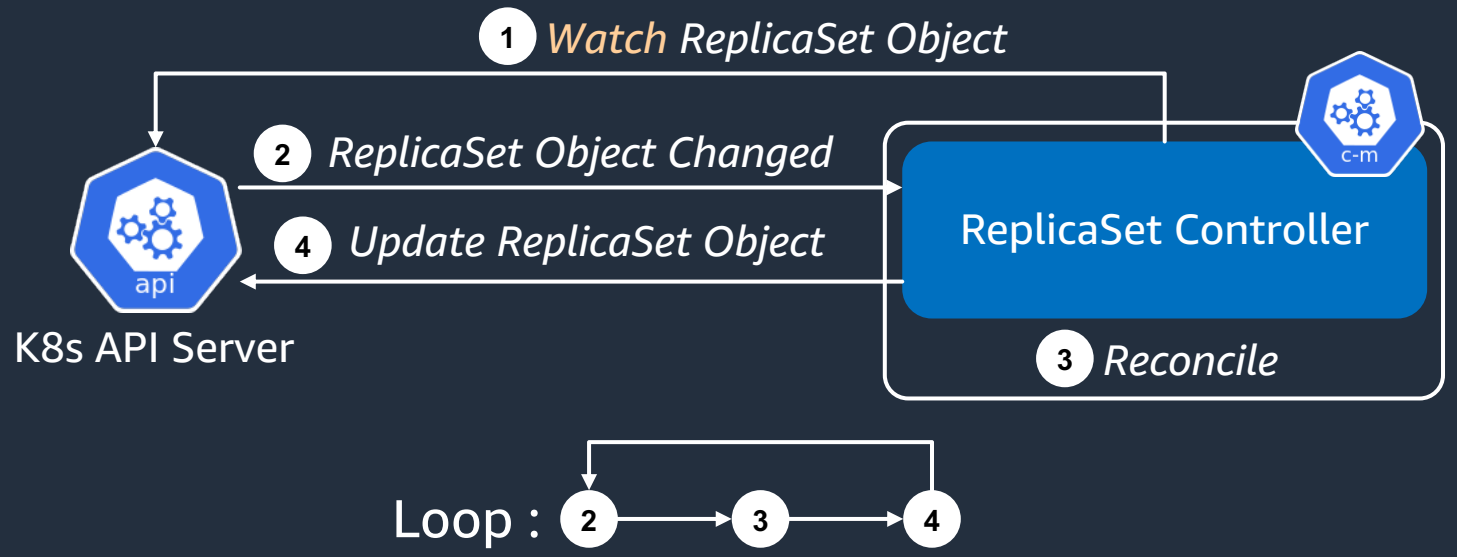


- Reconcile : Object의 Spec과 Status를 일치시키는 제어 과정
- User는 Spec (Desired State)만 명시하면 Controller가 스스로 제어하는 구조 때문에 Declarative (선언형) API로 분류

Example : ReplicaSet Object & Controller

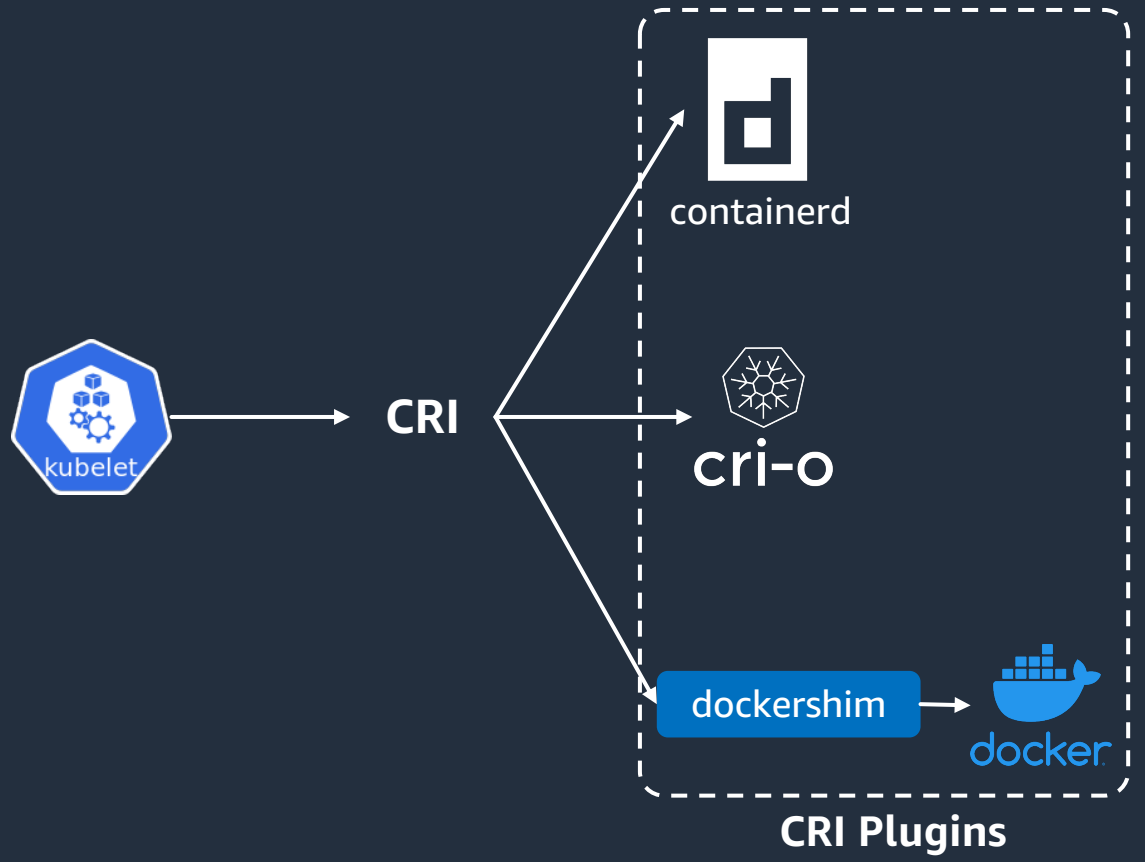
```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: example-deploy
...
spec:
  replicas: 10
...
status:
  replicas: 7
...
```

Example Deployment



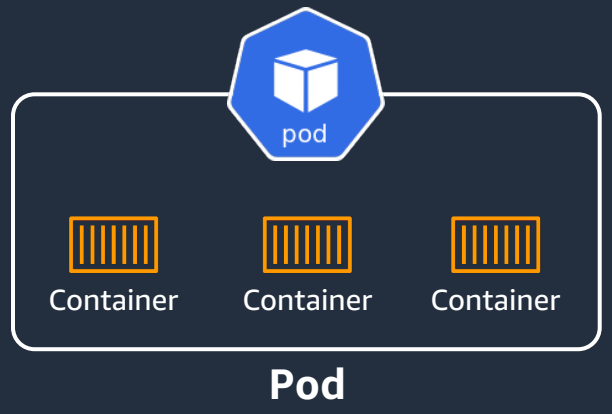
Kubernetes Pod

CRI (Container Runtime Interface)

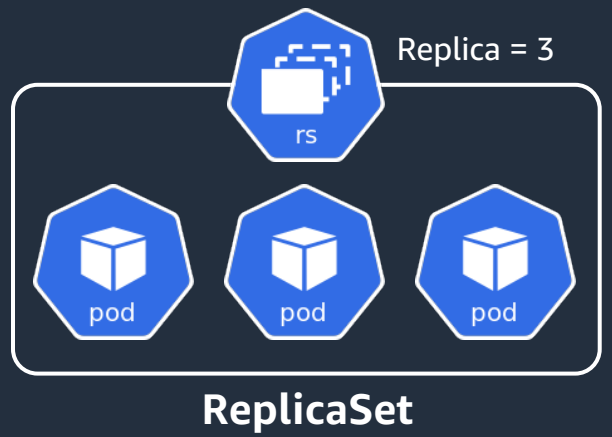


- K8s는 Container 관리를 직접 수행하지 않음
- K8s는 CRI를 통해서 CRI Plugin에게 Container 관리 위임
- Docker, dockershim은 Deprecated

Pod, ReplicaSet

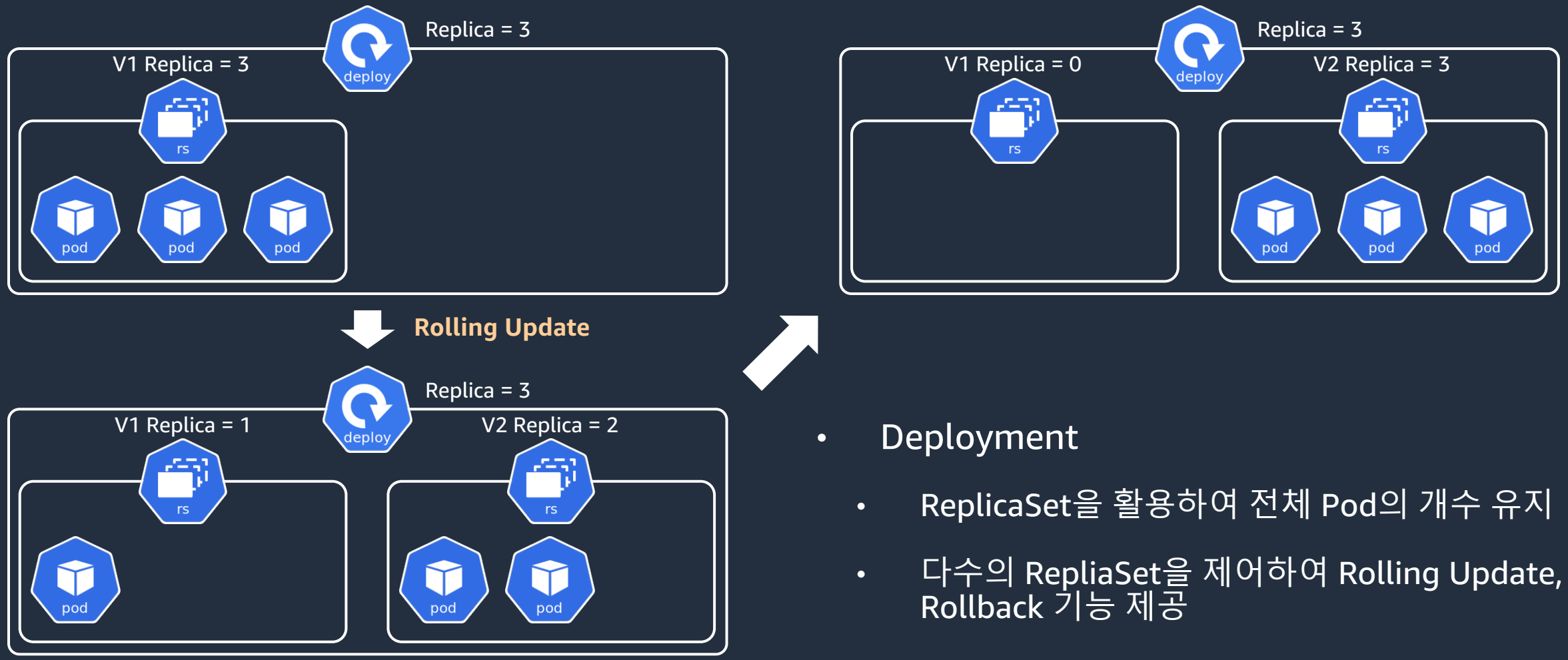


- Pod
  - K8s에서 배포, 관리 가능한 가장 작은 Computing Unit
  - 다수의 Container로 구성
  - Container 사이의 Network, Volume 공유



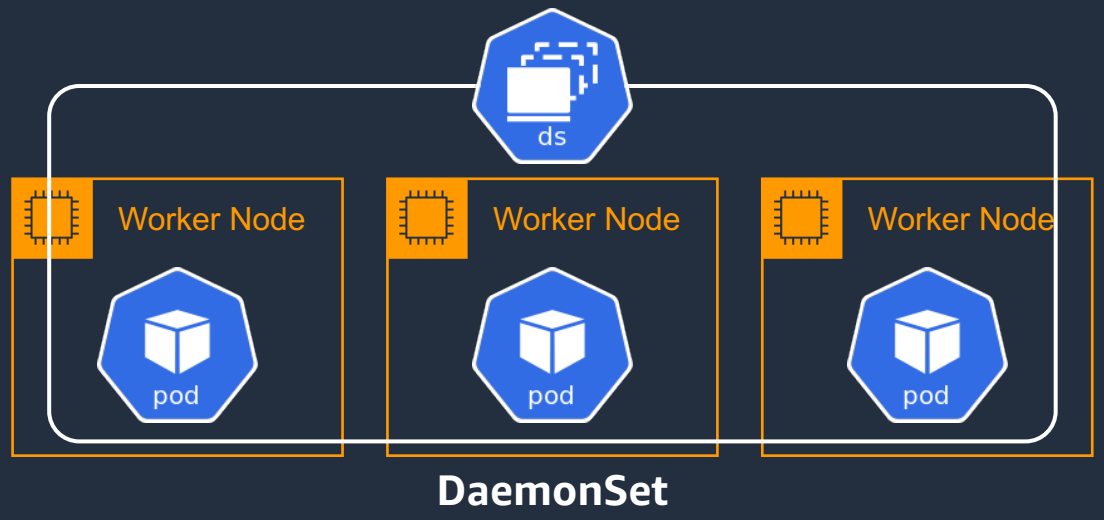
- ReplicaSet
  - Replica 개수 만큼 Pod의 개수를 유지
  - 일반적으로 직접 User가 이용하지 않고 Deployment를 통해서 이용

Deployment



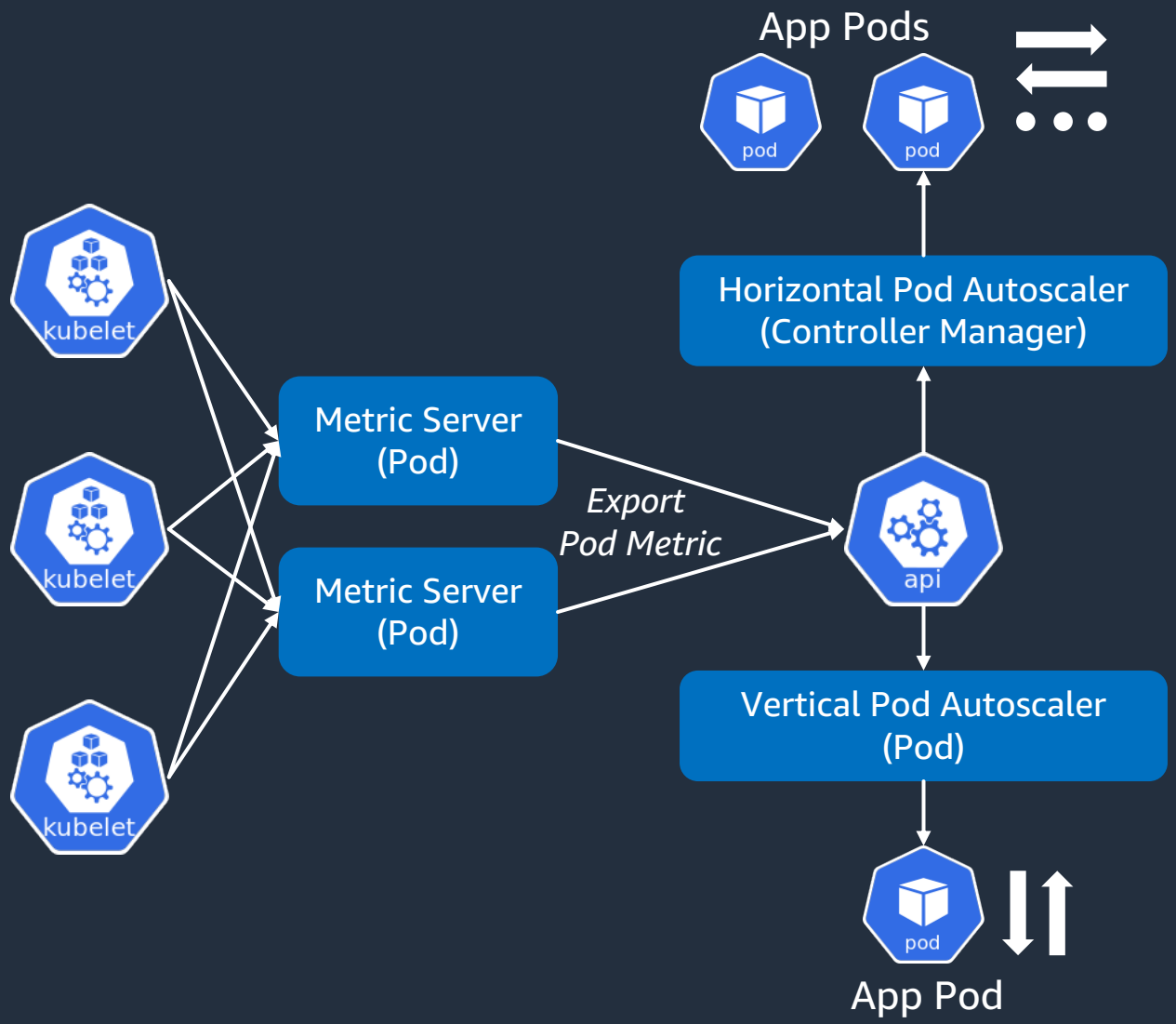
- Deployment
  - ReplicaSet을 활용하여 전체 Pod의 개수 유지
  - 다수의 RepliaSet을 제어하여 Rolling Update, Rollback 기능 제공

DaemonSet



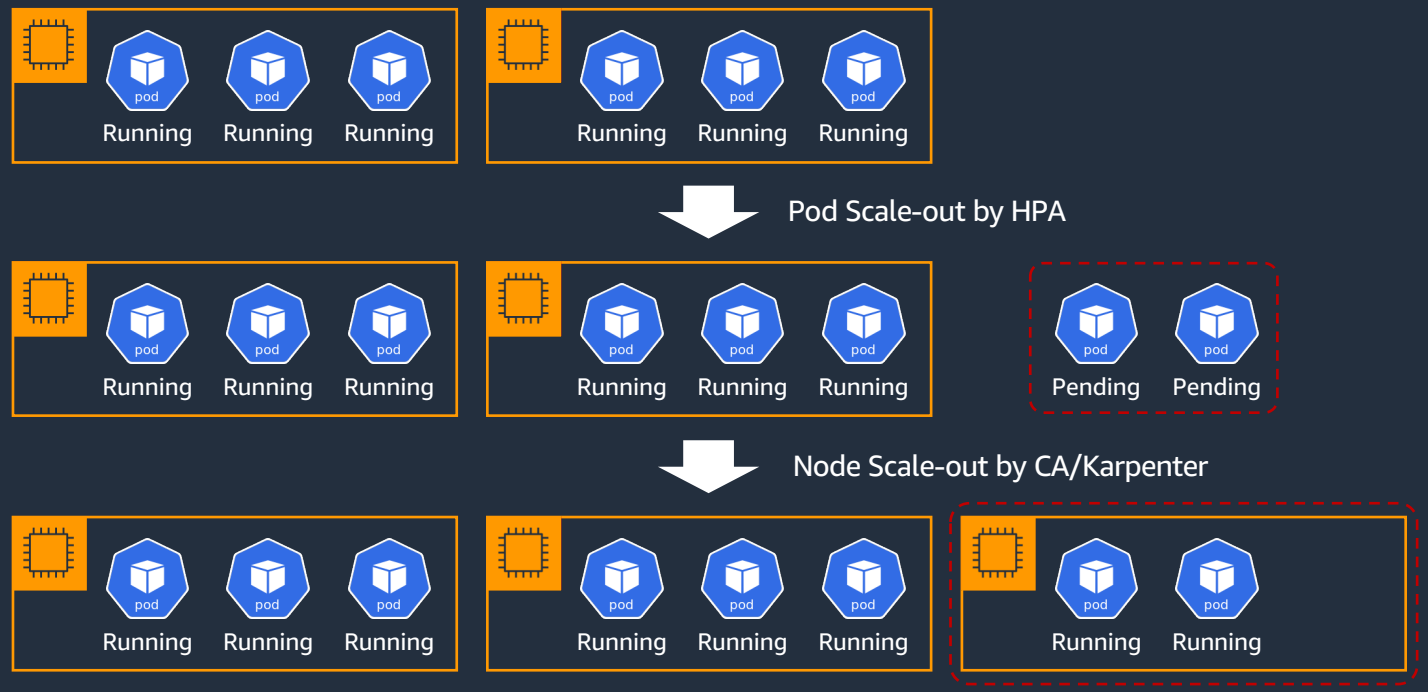
- DaemonSet
  - 모든 Node 또는 NodeSelector, NodeAffinity를 통해서 선택된 Node에 Pod를 한개씩 배포
  - kube-proxy, CNI Plugin등 각 Node마다 배포되어야 하는 Component의 경우 DaemonSet을 이용하여 배포

Pod Auto-scailing



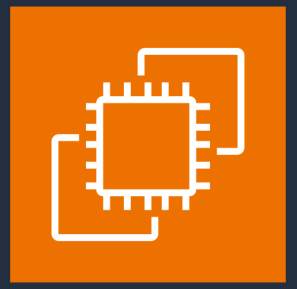
- HPA (Horizontal Pod Autoscaler)
  - Metric Server가 수집한 Pod의 Metric을 기반으로 필요에 따라서 Pod의 개수 변경
  - Controller Manager에 기본 포함
- VPA (Vertial Pod Autoscaler)
  - Metric Server가 수집한 Pod의 Metric을 기반으로 필요에 따라서 Pod의 Spec 변경
  - Pod의 Spec 변경시 Pod 재시작 필요
  - VPA 대신 HPA 이용 권장 (Metric Server 제외)
  - 별도 설치 필요, 고가용성을 위해서 2개 이상의 Pod 이용 권장
- Metric Server
  - kubelet으로부터 Pod Metric을 수집하여 K8s API Server로 Pod Metric Export 수행
  - 별도 설치 필요, 고가용성을 위해서 2개의 Pod 이용 권장

Cluster Auto-scailing



- Pod Scale-out
  - Pod의 부하가 증가하면 HPA (Horizontal Pod Autoscaler)에 의해서 Pod 개수 증가
- Cluster Scale-out
  - Pending 상태의 Pod가 존재하고 원인이 Node 부족이라면, CA (Cluster Autoscaler) 또는 Karpenter가 Node 생성 수행

Cluster Auto-scaler



Cluster Autoscaler

- 1세대 Cluster Autoscaler
- 많은 Reference 및 사례 존재
- 다양한 Cloud 환경에서 이용 가능

Karpenter

- 2세대 EC2 Node Manager
  (Node 관리 + Node Scaling 수행)
- 빠른 Auto-scailing
- Instance Type 및 On-demand/Spot Instance를 손쉽게 혼용하여 구성 가능
- EKS Cluster의 Context를 파악하여 Node Scaling 수행 (Node Selector, Pod Affinify, Topology Constratins, EBS Volume AZ)

Kubernetes Pod Scheduling

Node Label, Taint

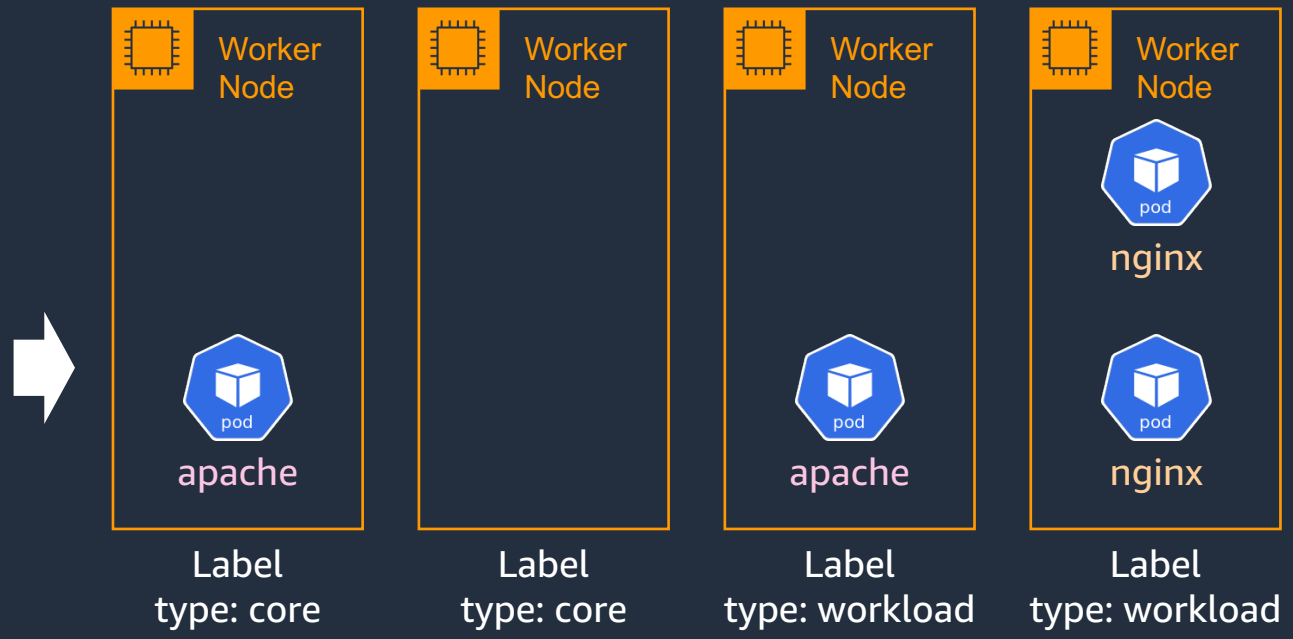
```
apiVersion: v1
kind: Node
metadata:
  labels:
    beta.kubernetes.io/arch: amd64
    beta.kubernetes.io/instance-type: m5.xlarge
    beta.kubernetes.io/os: linux
    eks.amazonaws.com/capacityType: ON_DEMAND
    eks.amazonaws.com/nodegroup: workshop-20240402072252983700000011
    eks.amazonaws.com/nodegroup-image: ami-0bf9fddc2187f96ec
    eks.amazonaws.com/sourceLaunchTemplateId: lt-0ad0ec6a63b781ed1
    eks.amazonaws.com/sourceLaunchTemplateVersion: "1"
    failure-domain.beta.kubernetes.io/region: ap-northeast-2
    failure-domain.beta.kubernetes.io/zone: ap-northeast-2a
    k8s.io/cloud-provider-aws: dc96c64d18c2675494b9609a385118ac
    kubernetes.io/arch: amd64
    kubernetes.io/hostname: ip-10-0-73-23.ap-northeast-2.compute.internal
    kubernetes.io/os: linux
    node.kubernetes.io/instance-type: m5.xlarge
    topology.ebs.csi.aws.com/zone: ap-northeast-2a
    topology.kubernetes.io/region: ap-northeast-2
    topology.kubernetes.io/zone: ap-northeast-2a
  name: ip-10-0-73-23.ap-northeast-2.compute.internal
...
```

```
spec:
  taints:
  - effect: NoExecute
    key: type
    value: core
```

Node Selector

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:  # Pod Template
    metadata:
      labels:
        app: nginx
    spec:
      nodeSelector:
        type: workload
...
```

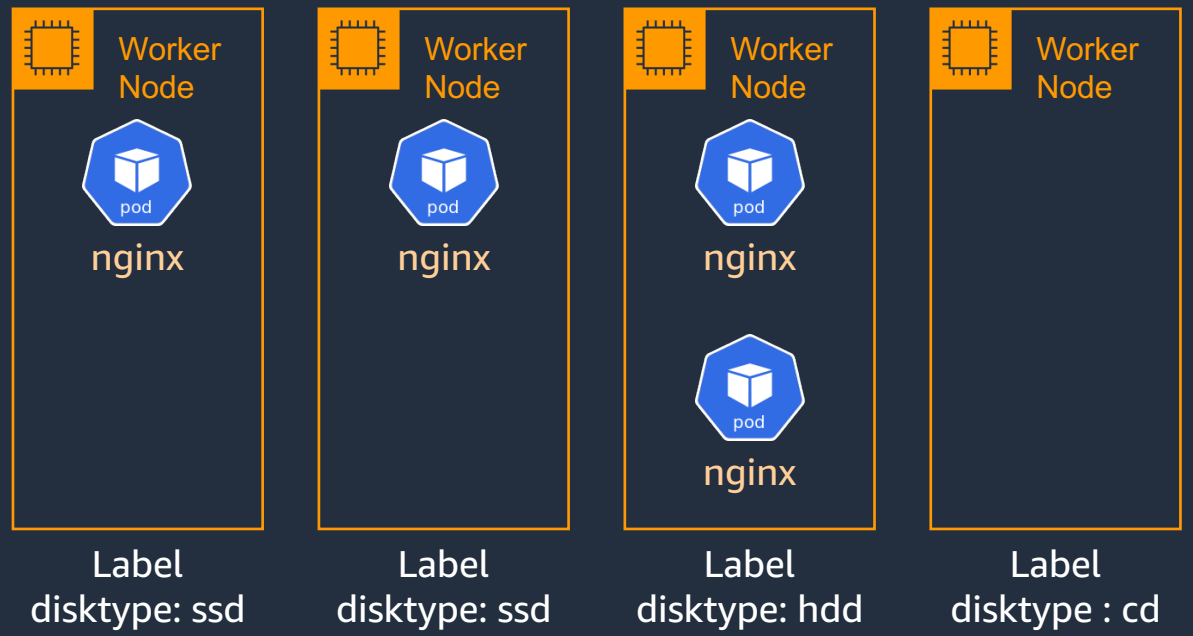
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: apache
  labels:
    app: apache
spec:
  replicas: 2
  selector:
    matchLabels:
      app: apache
  template:  # Pod Template
    metadata:
      labels:
        app: apache
...
```



- Node의 Label을 활용하여 어느 Node에서 Pod를 동작시킬지 결정
- Pod <-> Node 사이의 Hard Affinity 설정

Node Affinity

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 4
  selector:
    matchLabels:
      app: nginx
  template:  # Pod Template
    metadata:
      labels:
        app: nginx
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: disktype
                operator: In
                values:
                - ssd
                - hdd
```

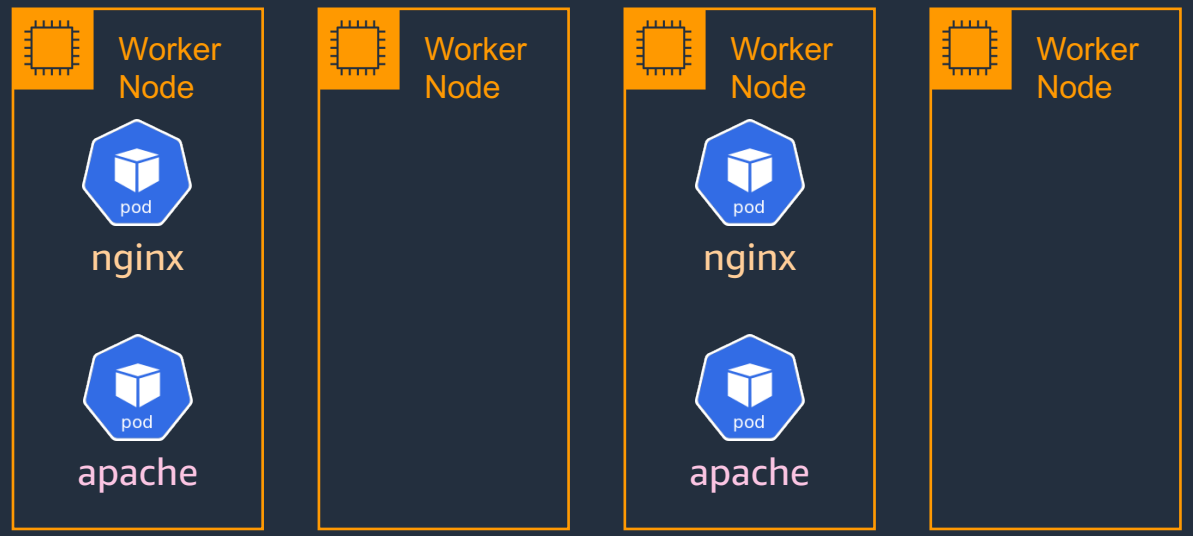


- Node Selector의 확장 Version
- Pod <-> Node 사이의 Soft/Hard Affinity 설정
  - Soft : preferredDuringSchedulingIgnoredDuringExecution
  - Hard : requiredDuringSchedulingIgnoredDuringExecution
- 다양한 Label 선택 기능 제공

Inter-Pod Affinity

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:  # Pod Template
    metadata:
      labels:
        app: nginx
    spec:
      affinity:
        podAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - topologyKey: kubernetes.io/hostname
            labelSelector:
              matchLabels:
                app: apache
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: apache
  labels:
    app: apache
spec:
  replicas: 2
  selector:
    matchLabels:
      app: apache
  template:  # Pod Template
    metadata:
      labels:
        app: apache
...
```

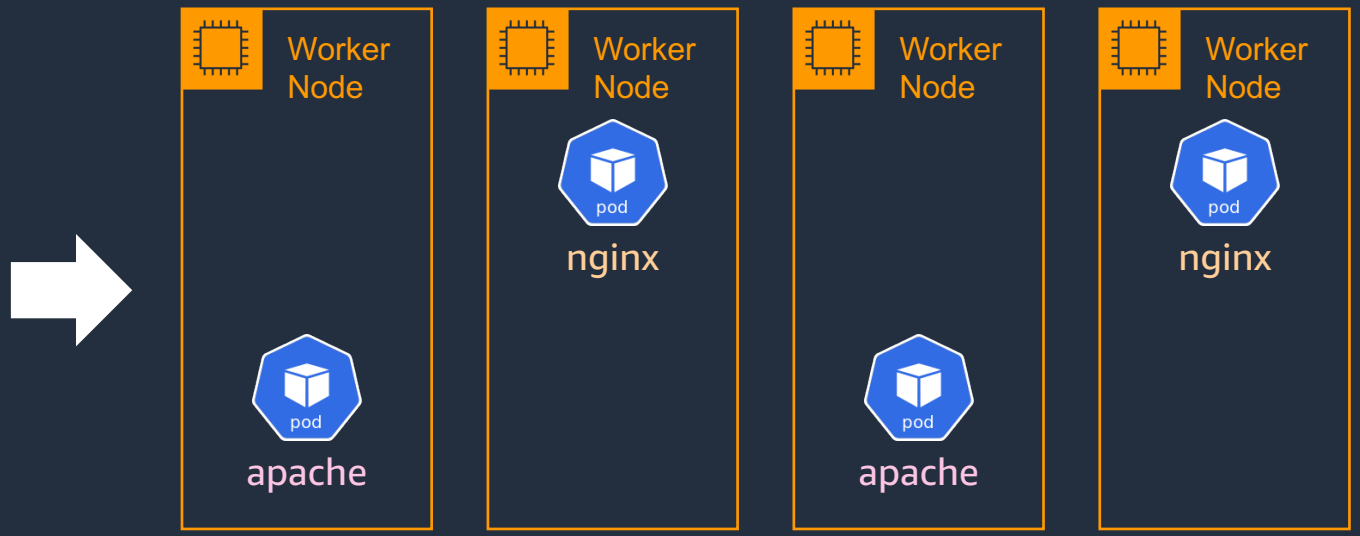


- Pod <-> Pod 사이의 Soft/Hard Affinity 설정
- 다양한 Label 선택 기능 제공

(Pod) Anti-affinity

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:  # Pod Template
    metadata:
      labels:
        app: nginx
    spec:
      affinity:
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - topologyKey: kubernetes.io/hostname
            labelSelector:
              matchLabels:
                app: apache
```

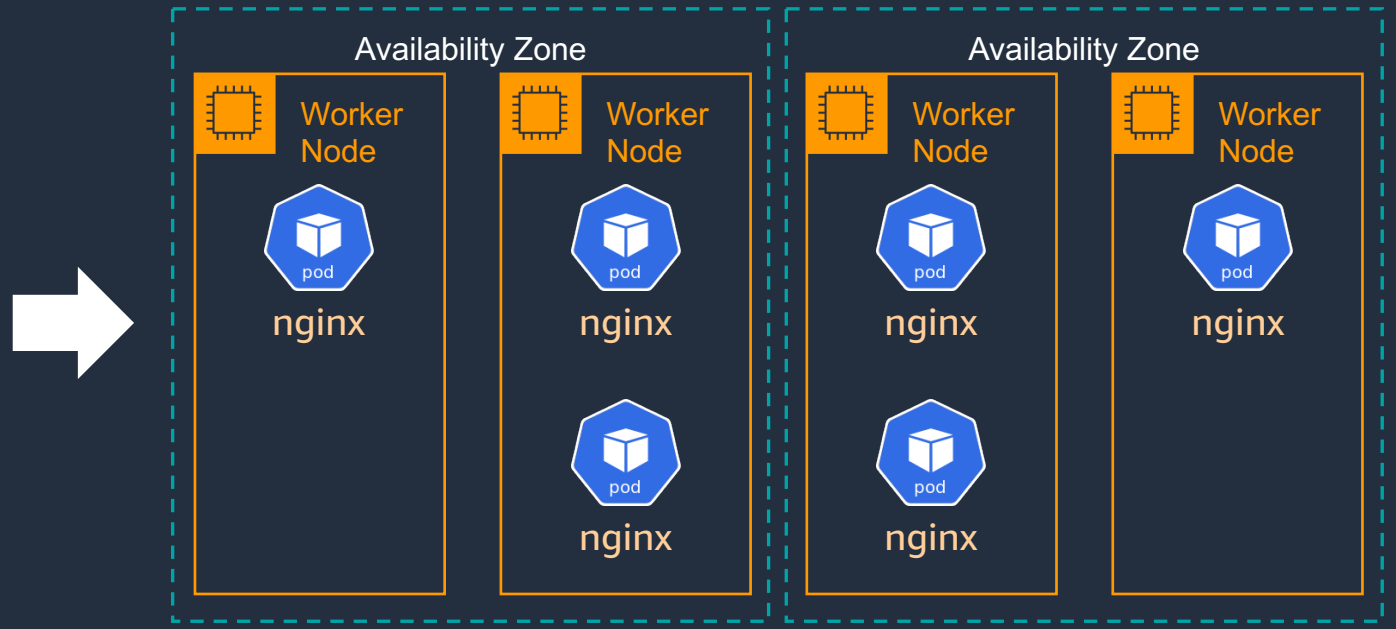
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: apache
  labels:
    app: apache
spec:
  replicas: 2
  selector:
    matchLabels:
      app: apache
  template:  # Pod Template
    metadata:
      labels:
        app: apache
...
```



- Pod <-> Pod 사이의 Soft/Hard Anti-affinity 설정
- 다양한 Label 선택 기능 제공
- Self Label 지정을 통해서 모든 Node에 고르게 분배되도록 설정 가능

Topology Spread Constraints

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 6
  selector:
    matchLabels:
      app: nginx
  template:  # Pod Template
    metadata:
      labels:
        app: nginx
    spec:
      topologySpreadConstraints
      - maxSkew: 1
        topologyKey: topology.kubernetes.io/zone
        whenUnsatisfiable: DoNotSchedule
        labelSelector:
          matchLabels:
            app: nginx
```

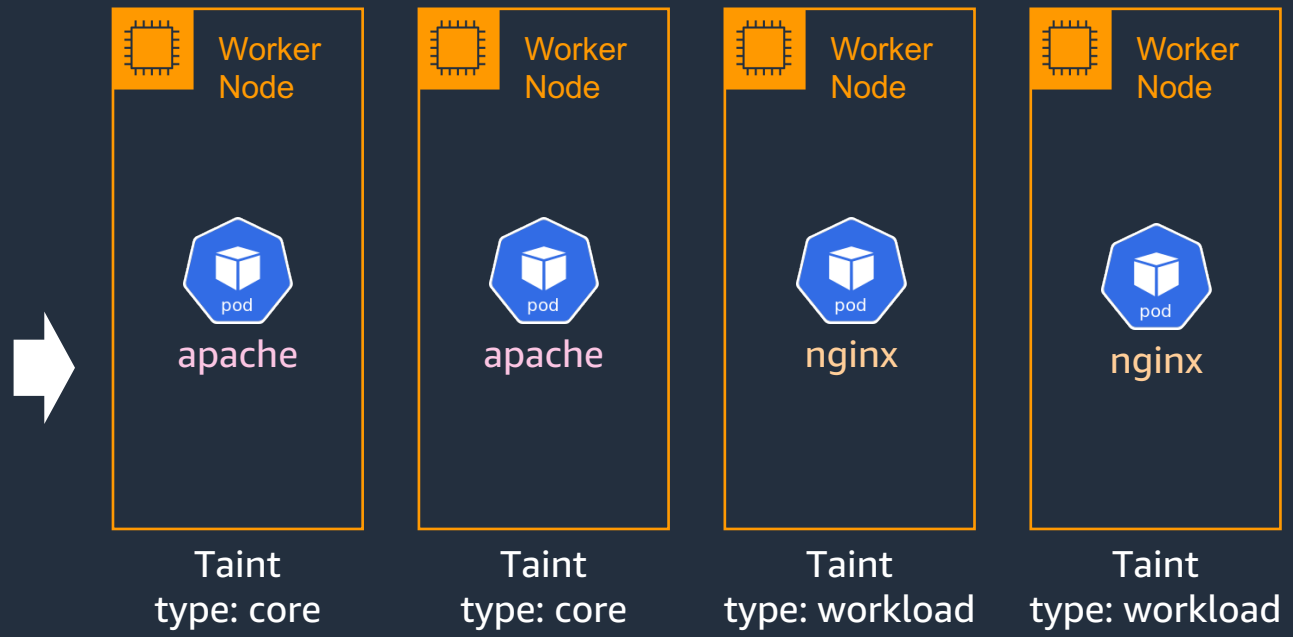


- Topology (Host, Zone, Region)을 인식하여 Pod 분배

Taint, Toleration

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:  # Pod Template
    metadata:
      labels:
        app: nginx
    spec:
      tolerations:
      - key: "type"
        operator: "Eqaul"
        value: "workload"
        effect: "NoExecute"
...
```

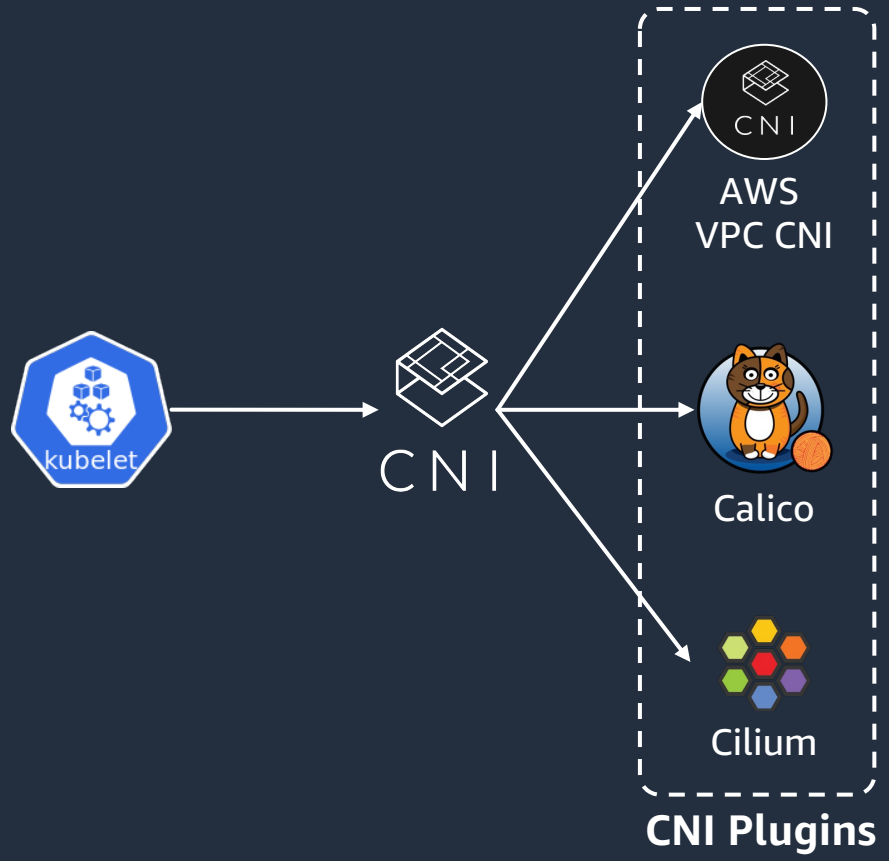
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: apache
  labels:
    app: apache
spec:
  replicas: 2
  selector:
    matchLabels:
      app: apache
  template:  # Pod Template
    metadata:
      labels:
        app: apache
    spec:
      tolerations:
      - key: "type"
        operator: "Eqaul"
        value: "core"
        effect: "NoExecute"
...
```



- Taint가 설정된 Node에 Pod가 동작하기 위해서는 반드시 Toleration 필요

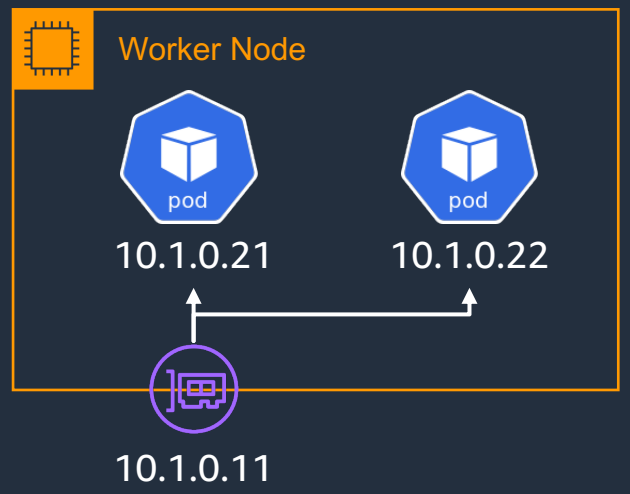
Kubernetes Network

Pod Network, CNI (Container Network Interface)

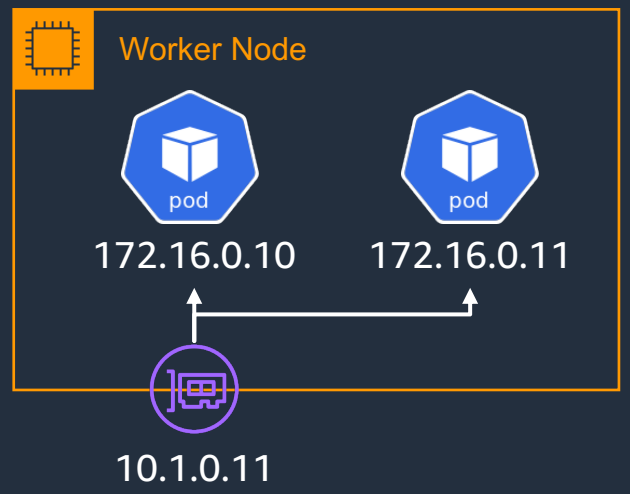


- K8s는 Pod Network를 직접 관리하지 않음
- K8s는 CNI를 통해서 CNI Plugin에게 Pod Network 관리 위임

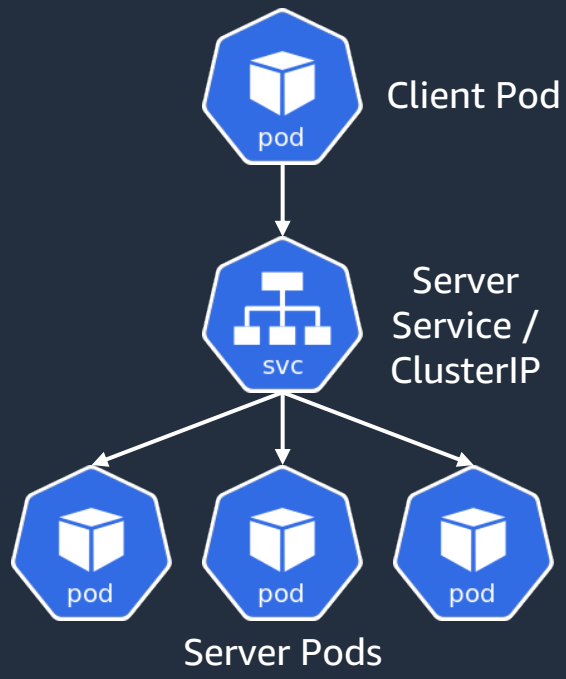
Node, Pod 동일 Network
- Pod, Node Network : 10.1.0.0/24
- AWS VPC CNI, Calico, Cilium



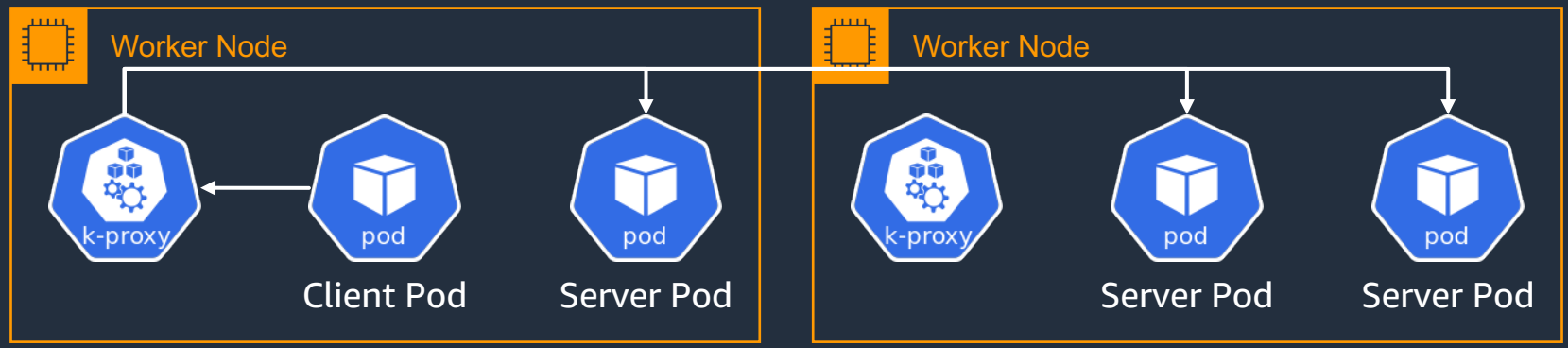
Node, Pod 별도 Network
- Node Network : 10.1.0.0/24
- Pod Network : 172.16.0.0/24
- Calico, Cilium



Service / ClusterIP

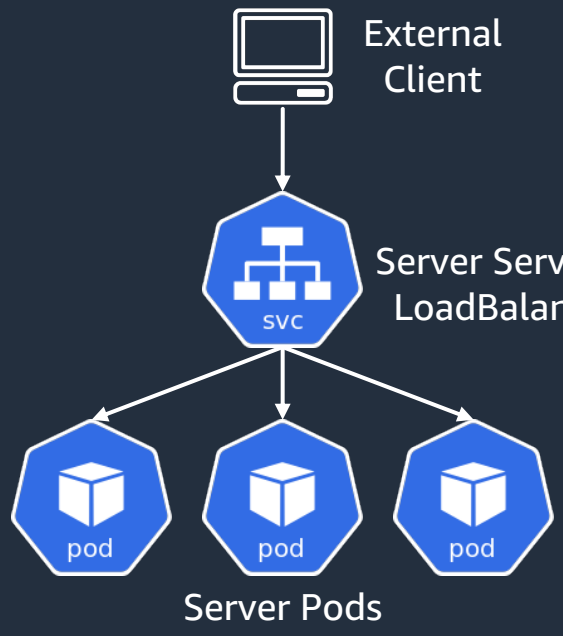


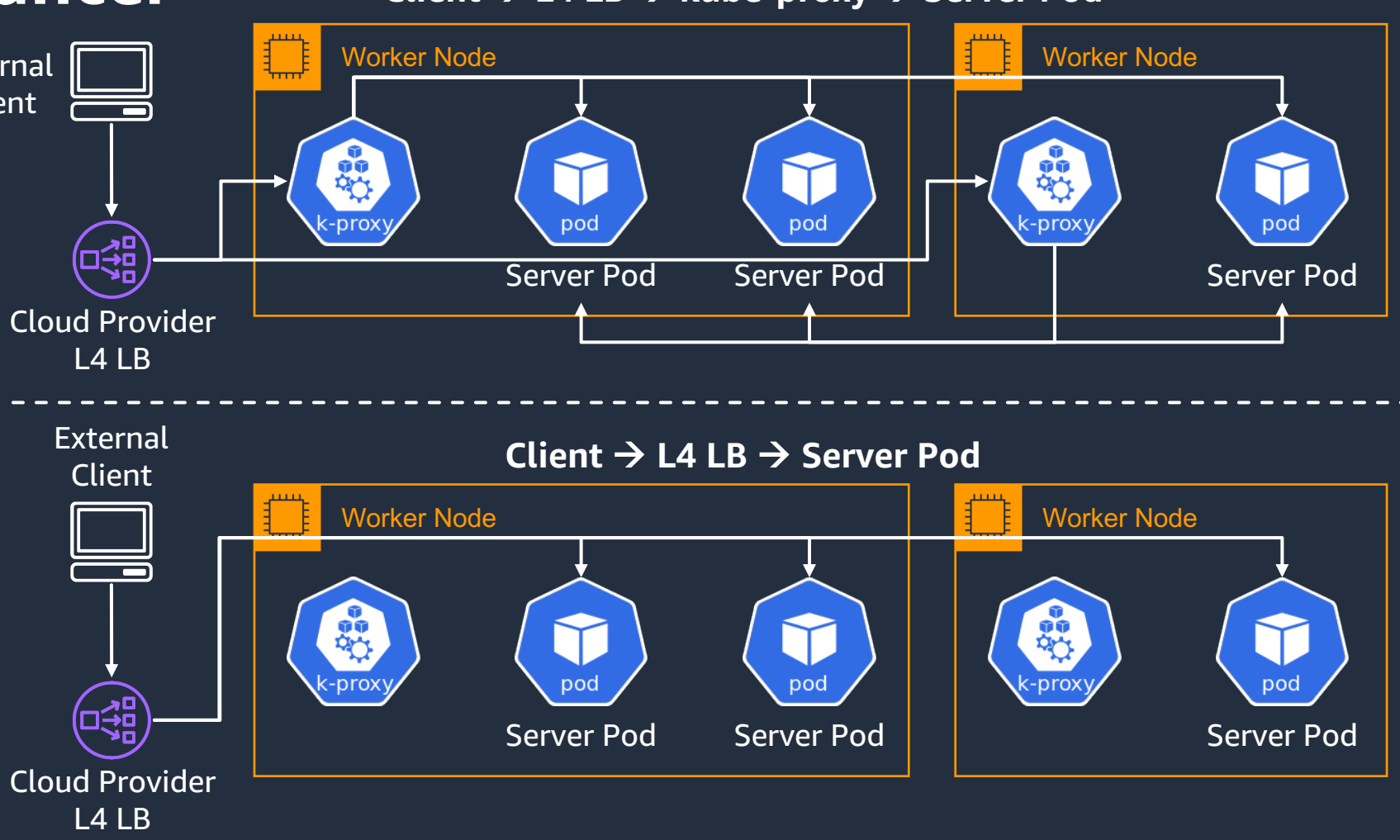
Service / ClusterIP



- Service
  - Pod의 Network Group
  - kube-proxy를 기반으로 L4 Traffic Load Balancing 수행 가능
  - Cluster 내부에서 Service의 ClusterIP를 통해서 Pod에 접근 가능

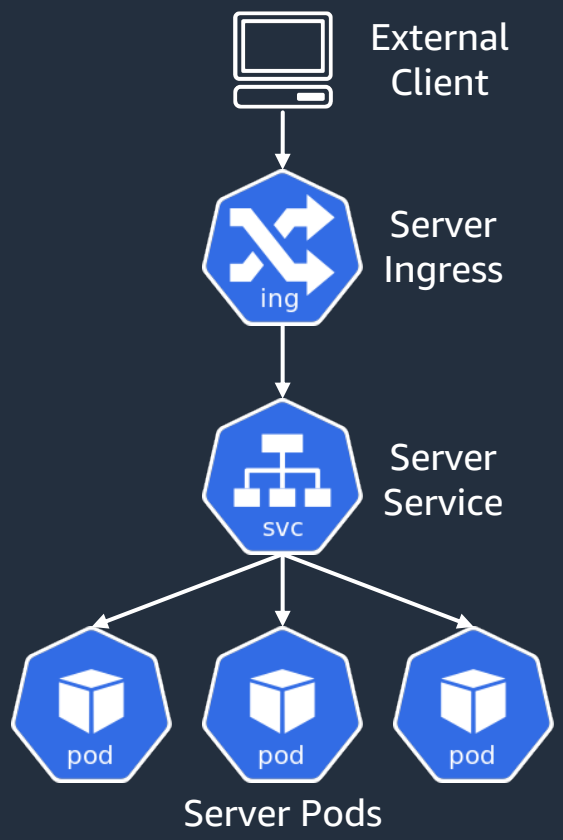
Service / LoadBalancer



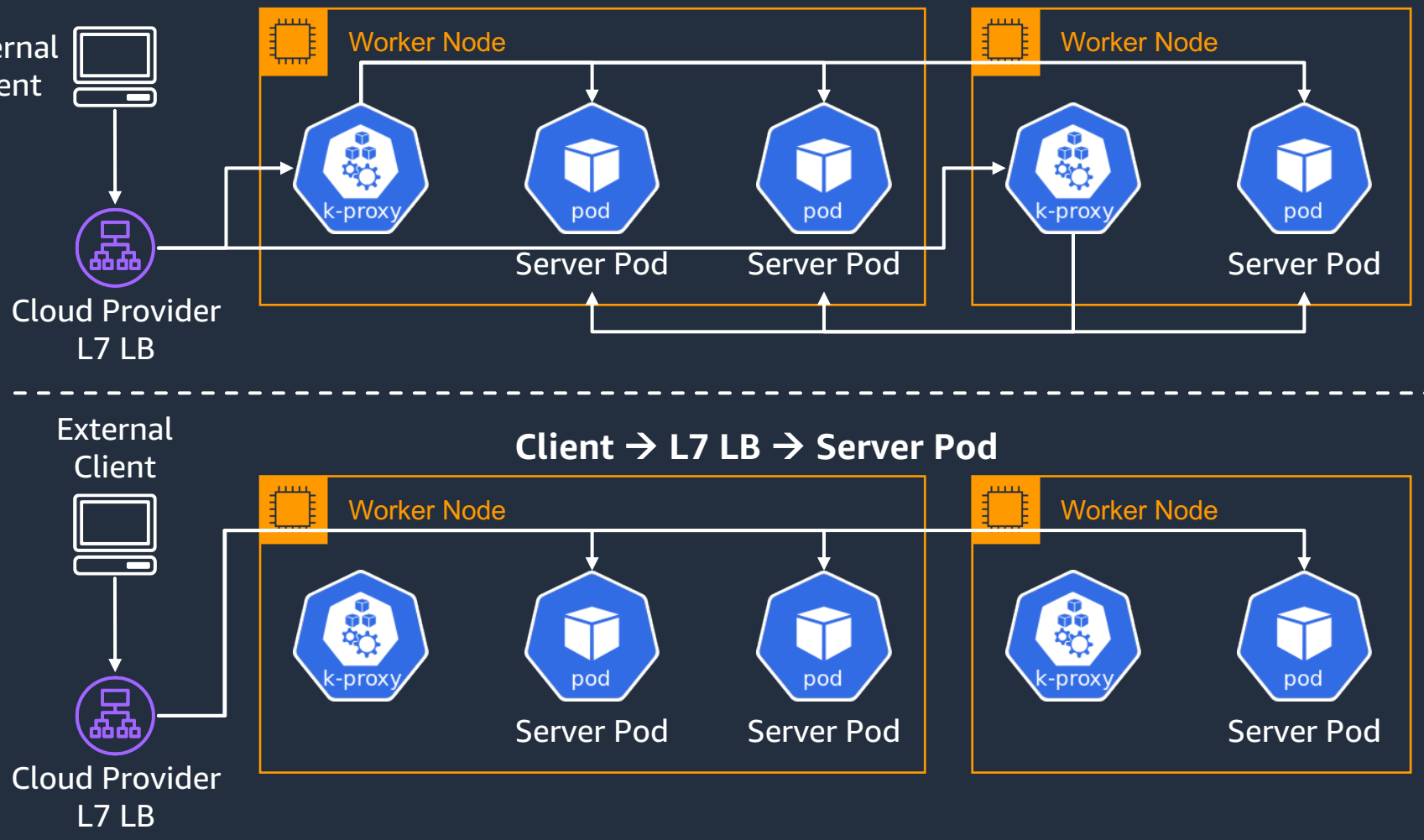


- Service LoadBalancer
  - LoadBalancer를 통해서 Cluster 외부에서 접근 가능
  - Cloud Provider에서 제공하는 LB Controller 설치 필요
  - 설정에 따라서 Traffic이 kube-proxy를 경유하거나 경유하지 않을 수 있음

Ingress / Cloud Provider L7



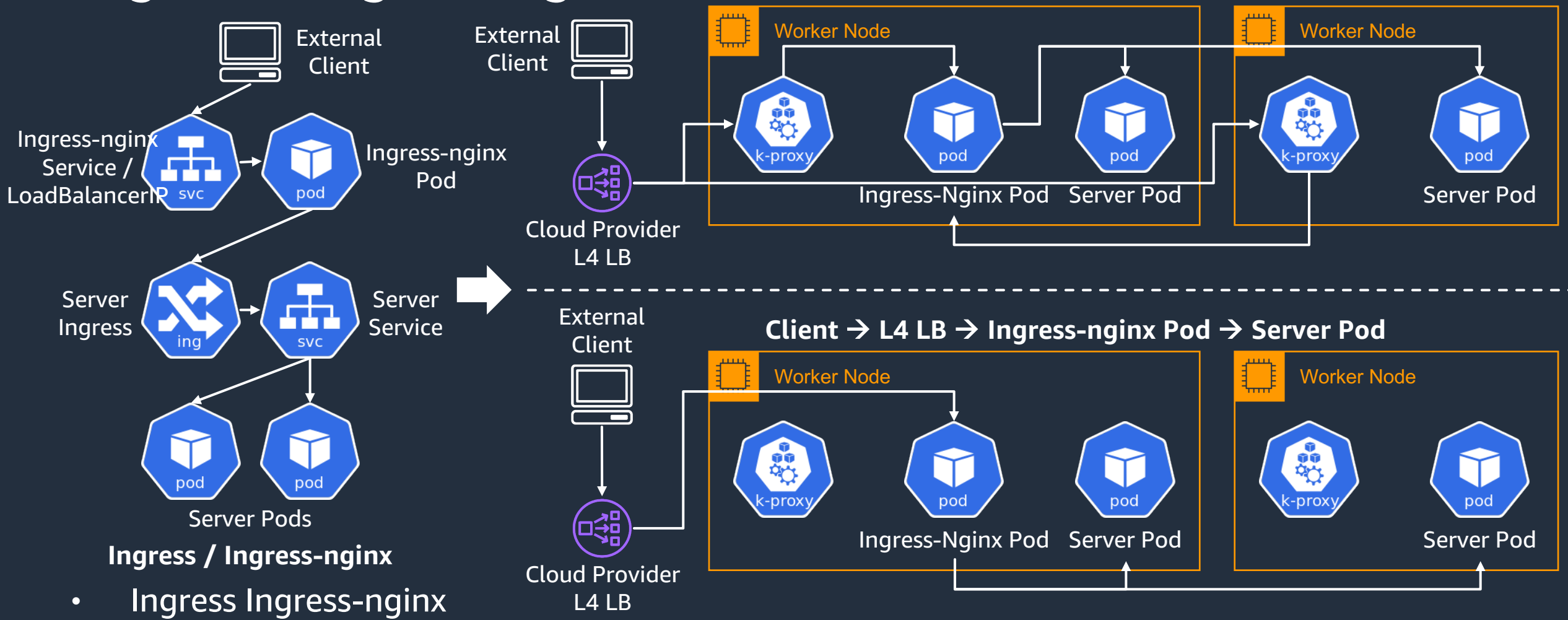
Ingress / Cloud Provider L7



- Ingress
  - Cluster 외부에서 접근 가능한 L7 Network Rule 설정
  - Cloud Provider에서 제공하는 Ingress Controller 설치 필요
  - 설정에 따라서 Traffic이 kube-proxy를 경유하거나 경유하지 않을 수 있음

Ingress / Ingress-nginx L7

Client → L4 LB → kube-proxy → Ingress-nginx Pod → Server Pod



Ingress / Ingress-nginx

- Ingress Ingress-nginx
  - Ingress-nginx Controller 설치 필요
  - Cloud Provider의 L4 Load Balancer를 통해서 Ingress-nginx로 Traffic 유입
  - 설정에 따라서 Traffic이 kube-proxy를 경유하거나 경유하지 않을 수 있음

Kubernetes App Configuration, Secret

ConfigMap

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: game-demo
data:
  player_initial_lives: "3"
  ui_properties_file_name: "user-interface.properties"
  game.properties: |
    enemy.types=aliens,monsters
    player.maximum-lives=5
  user-interface.properties: |
    color.good=purple
    color.bad=yellow
    allow.textmode=true
```

ConfigMap Example

- ConfigMap
  - Non-confidential Key-Value Store
  - Pod에서 Env, Volume으로 접근 가능

```
apiVersion: v1
kind: Pod
metadata:
  name: configmap-demo-pod
spec:
  containers:
    - name: demo
      image: alpine
      command: ["sleep", "3600"]
      env:
        # Env with ConfigMap
        - name: PLAYER_INITIAL_LIVES
          valueFrom:
            configMapKeyRef:
              name: game-demo
              key: player_initial_lives
volumeMounts:
      - name: config
        mountPath: "/config"
        readOnly: true
  volumes:
  - name: config
    # Volume with ConfigMap
    configMap:
      name: game-demo
      items:
      - key: "game.properties"
        path: "game.properties"
      - key: "user-interface.properties"
        path: "user-interface.properties"
```

Pod with ConfigMap Example

https://kubernetes.io/docs/concepts/configuration/configmap/

Secret

```
apiVersion: v1
kind: Secret
metadata:
  name: secret-ssh-auth
data:
  ssh-privatekey: |
        MIIEpQIBAAKCAQEAulqb/Y ...
```

Secret Example

- Secret
  - Confidential Key-Value Store
  - Base64 Encoding 형태로 Data 저장 (암호화 X)
  - Pod에서 Env, Volume으로 접근 가능
  - 다양한 Type 제공
    - Opaque Secrets : 임의의 Key-Value Data 저장
    - Service Account : Service Account Token 저장
    - Docker Registry : Container Image Registry Credential 저장

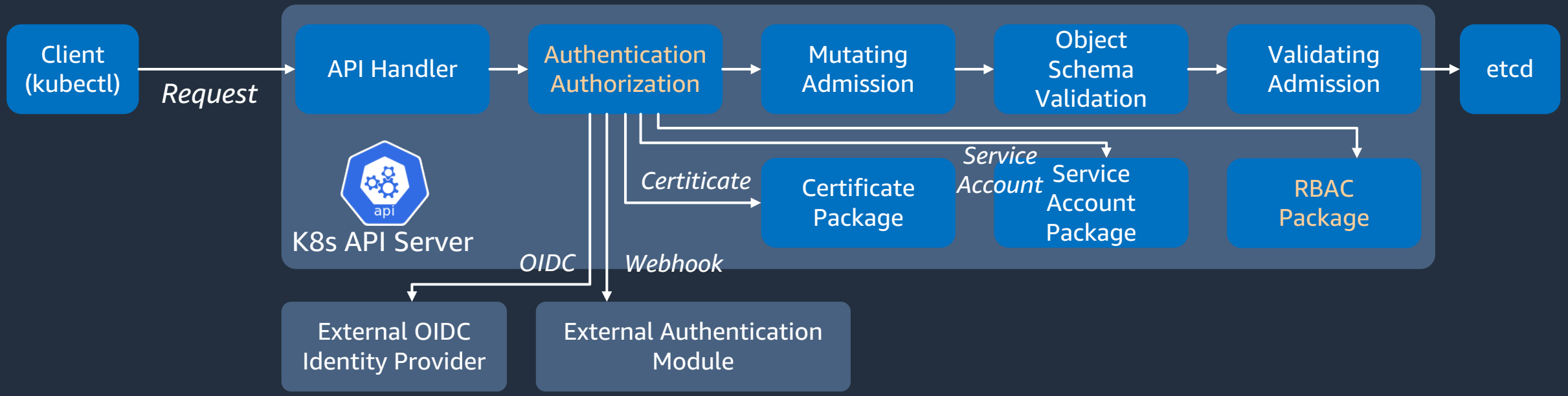
```
apiVersion: v1
kind: Pod
metadata:
  name: secret-test-pod
  labels:
    name: secret-test
spec:
  containers:
  - name: ssh-test-container
    image: mySshImage
    volumeMounts:
    - name: secret-volume
      mountPath: "/etc/secret-volume"
  env:
  - name: PRIVATE_KEY
    valueFrom:
      # Env with Secret
      secretKeyRef:
        name: ssh-key-secret
        key: ssh-privatekey
  volumes:
  - name: secret-volume
    # Volume with Secret
    secret:
      secretName: ssh-key-secret
```

Pod with Secret Volume Example

https://kubernetes.io/docs/concepts/configuration/secret

Kubernetes Security

Kubernetes 인증, 인가



- 인증
  - 다양한 방식의 인증 기법 제공
  - Ceritificate, Servic Account, OIDC, Webhook
- 인가
  - K8s에서 제공하는 RBAC 이용
  - K8s RBAC에서 허용되어도 Admission Controller에서 거부될 수 있음

Kubernetes RBAC Objects

Role

- 특정 Namespace 내부의 Namespaced Object를 대상으로 권한 명시

ClusterRole

- 모든 Namespace의 Namespaced Object를 대상으로 권한 명시
- Non-namespaced Object를 대상으로 권한 명시

RoleBinding

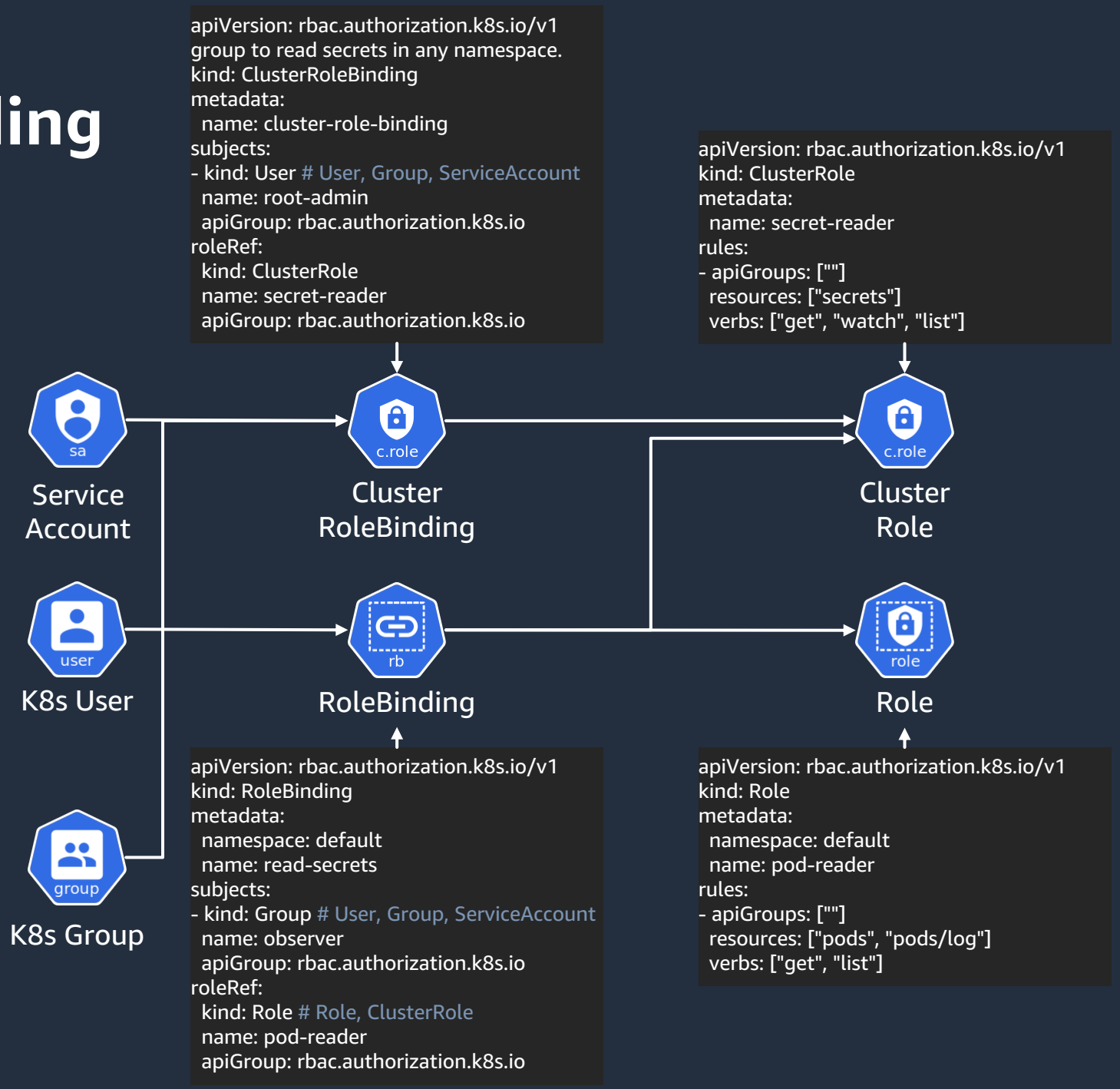
- 특정 Namespace안에서 Role을 User/Group/SA에 Binding 수행
- 특정 Namespace안에서 ClusterRole을 User/Group/SA에 Binding 수행 (ClusterRole에 명시된 Namespaced Object만 권한 부여)

ClusterRoleBinding

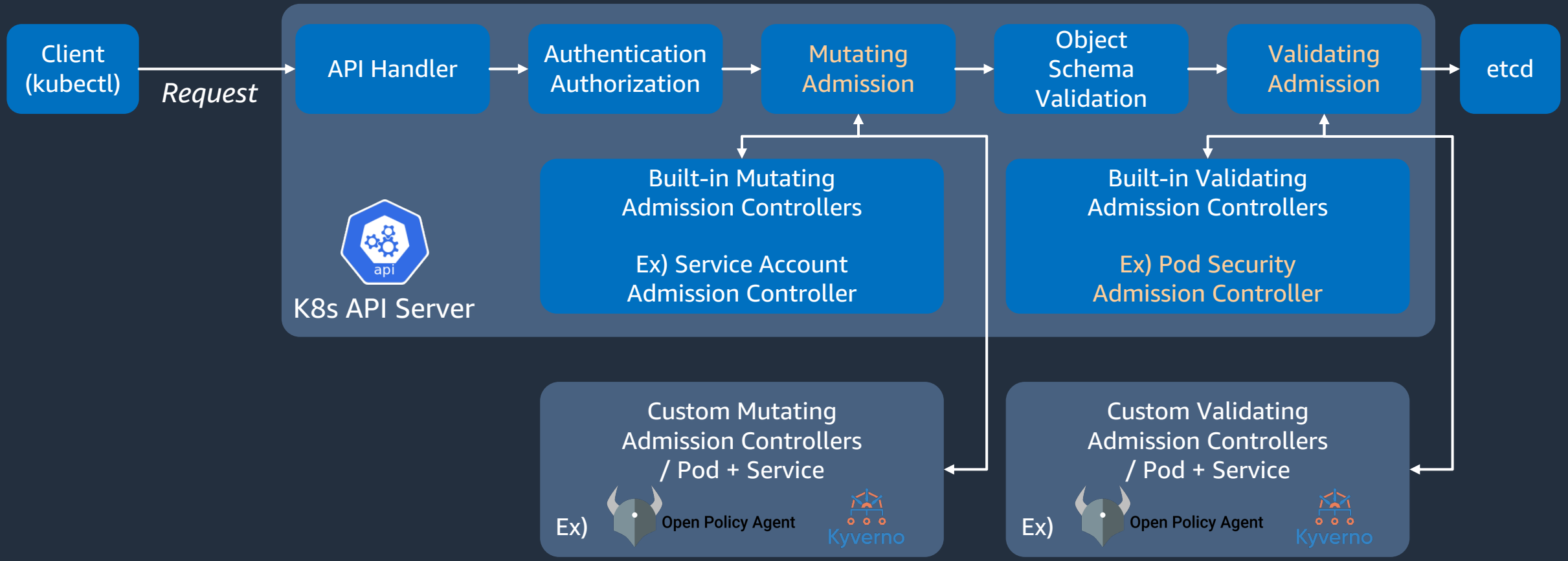
- 모든 Namespace안에서 ClusterRole을 User/Group/SA에 Binding 수행

Kubernetes RBAC Binding

- Service Account 인증
  → Service Account
- Certificate, OIDC, Webhook 인증
  → User, Group
- K8s User, K8s Group은 Object로 존재하지 않으며, K8s API Server 내부적으로 관리



Kubernetes Admission Controller Architecture



- Mutating Admission : Request 변환 및 Allow/Deny 수행
- Validating Admission : Request Allow/Deny 수행

Kubernetes Admission Controllers

Pod Security Policy (PSP)

- Built-in Controller
- Deprecated (v1.21 ~)

Pod Security Standard (PSS)

- Built-in Controller
- Pod Security Admission Controller를 통해 구현
- Pod Security Policy 대체

OPA

- Custom Controller
- Fine Grained
- Rego 문법 기반 Rule 작성

Kyverno

- Custom Controller
- Fine Grained
- YAML 문법 기반 Rule 작성

Thank you!