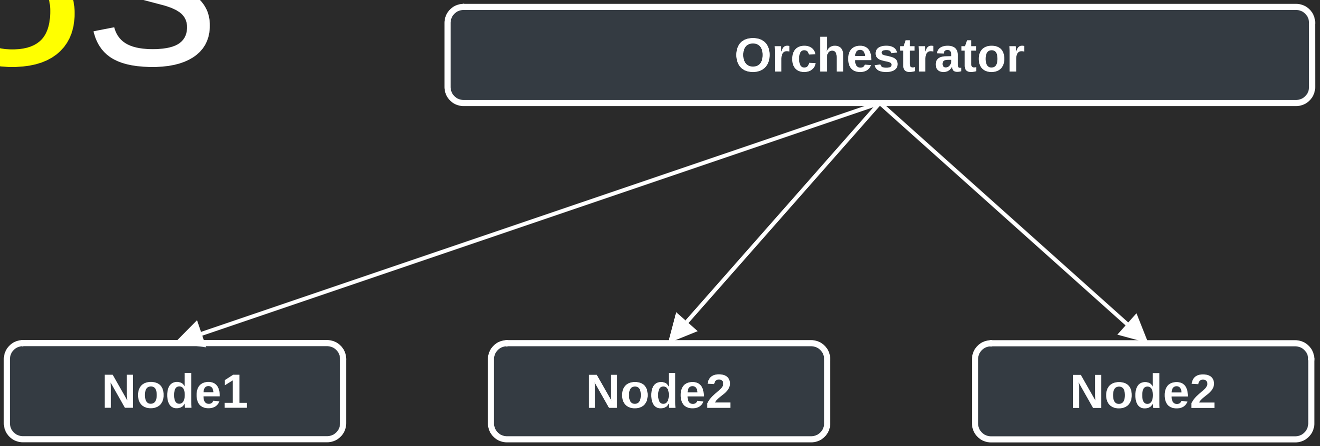
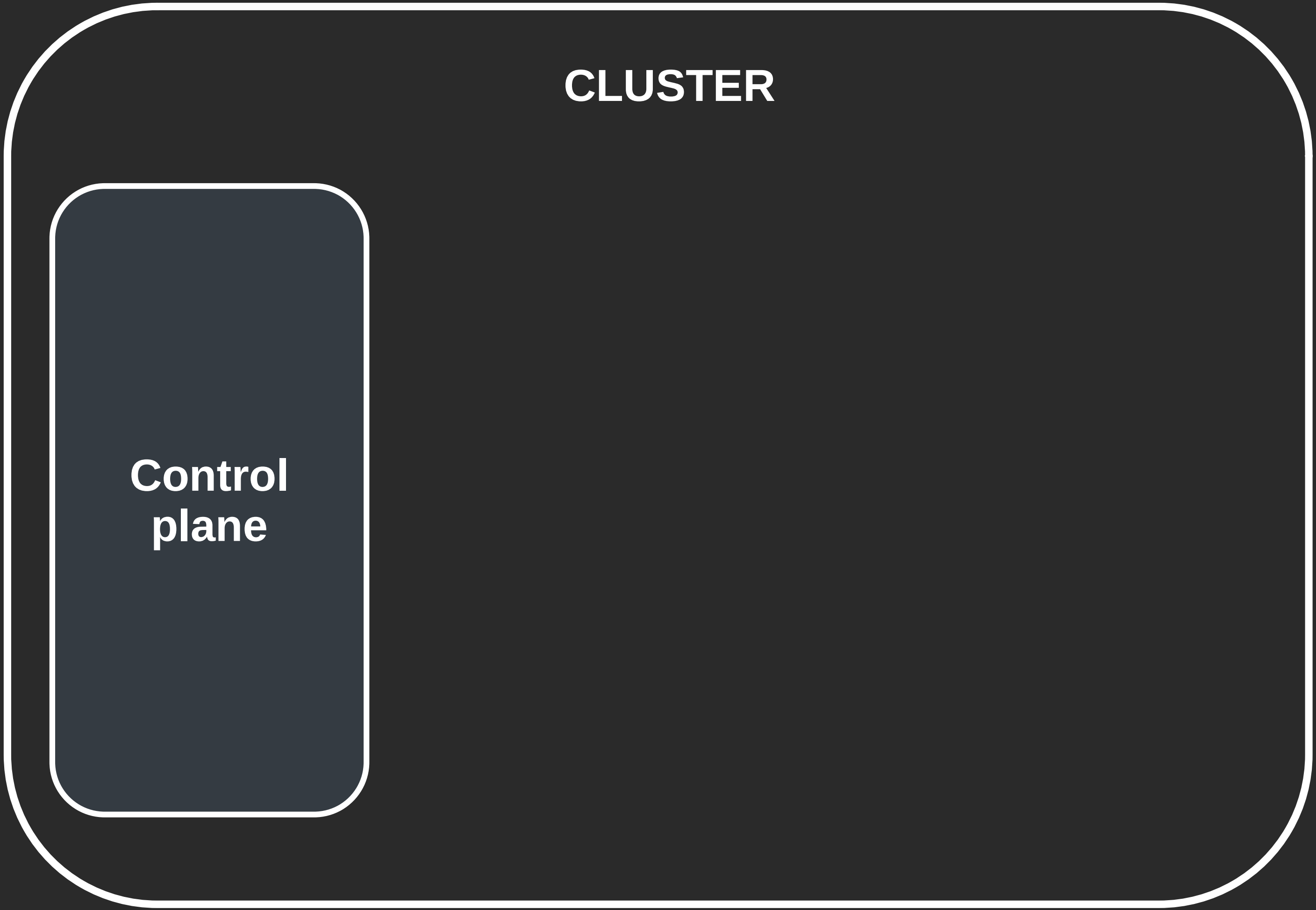
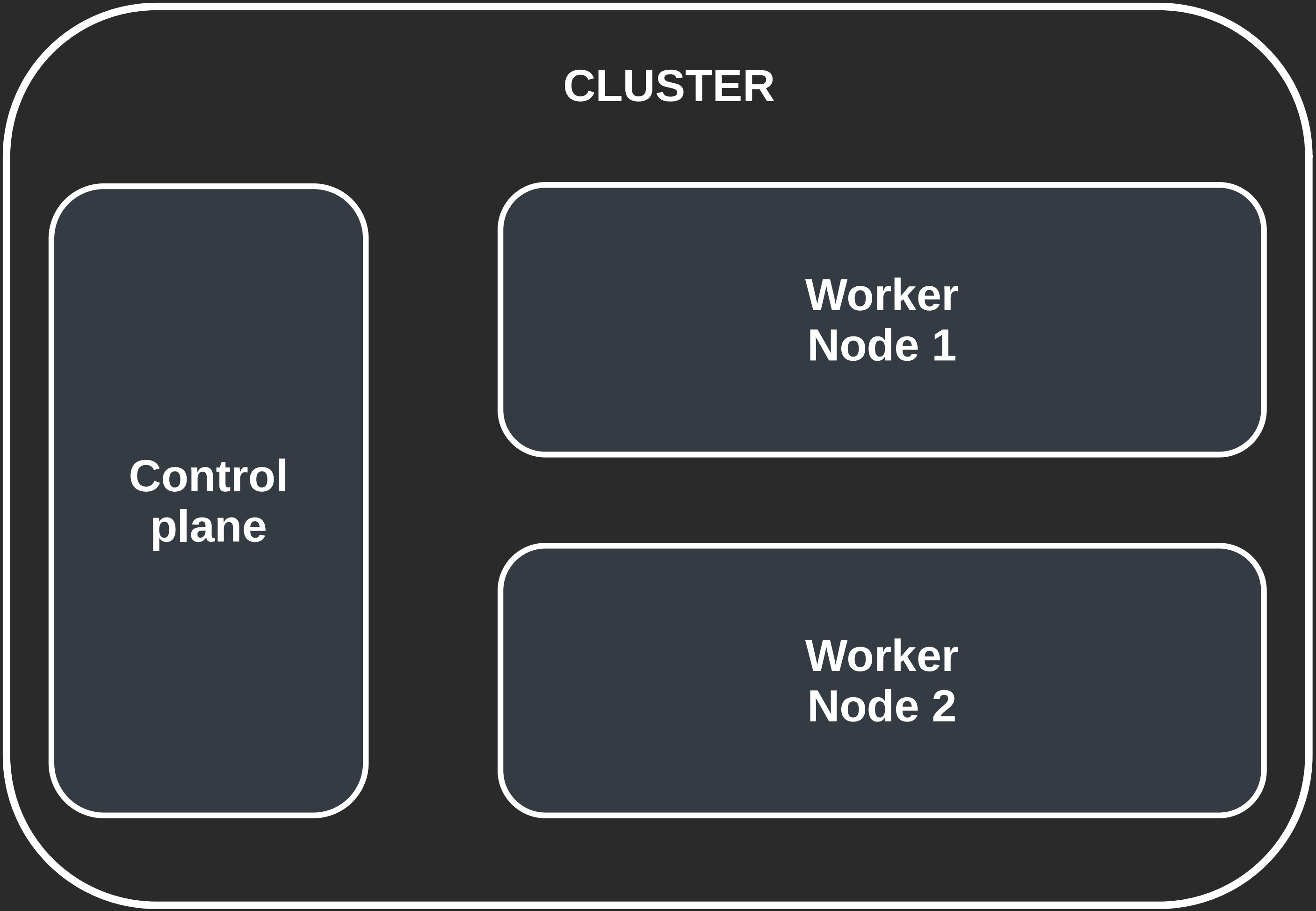
Kubernetes overview

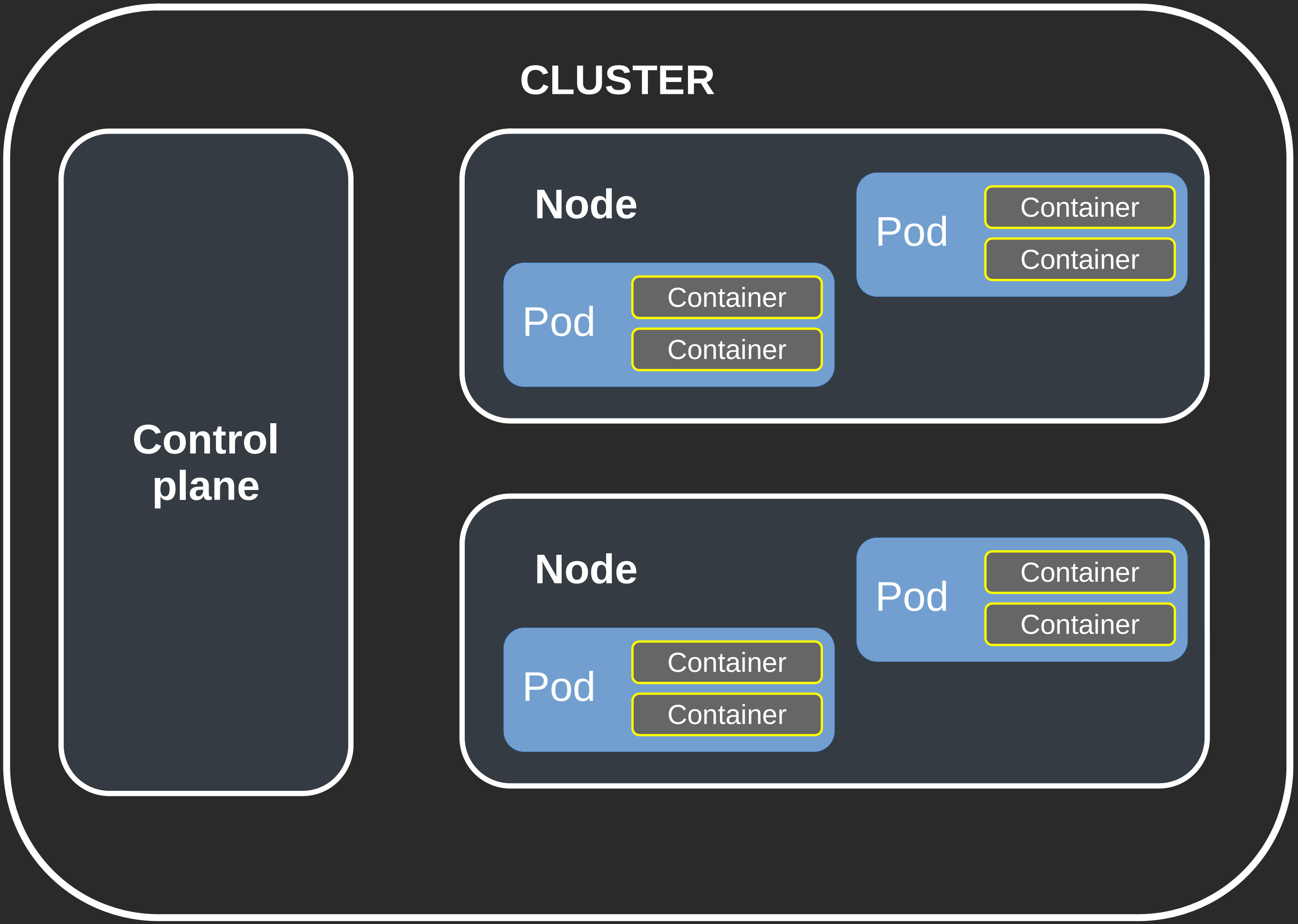
apriorit

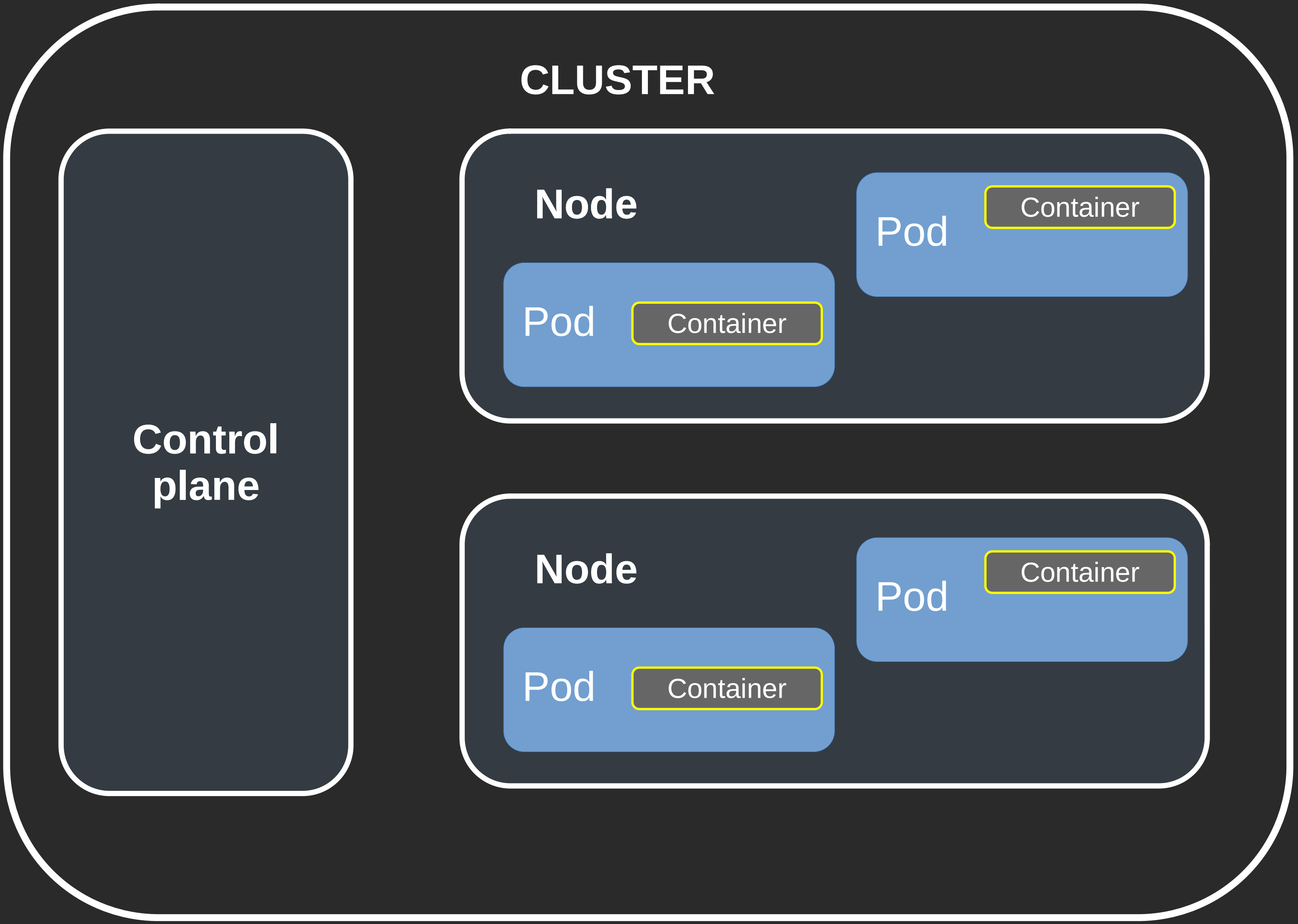
Kubernetes
k8s

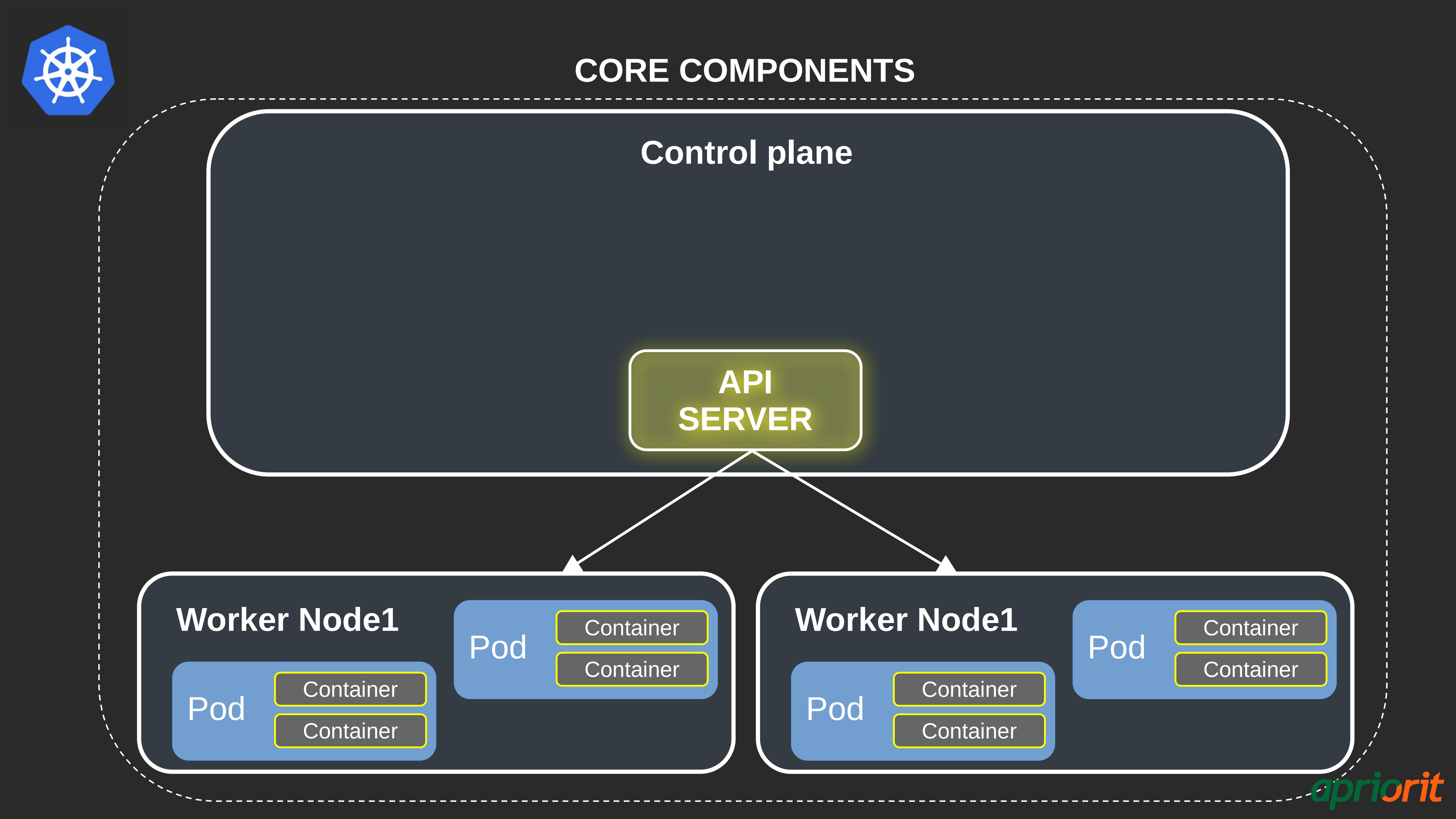


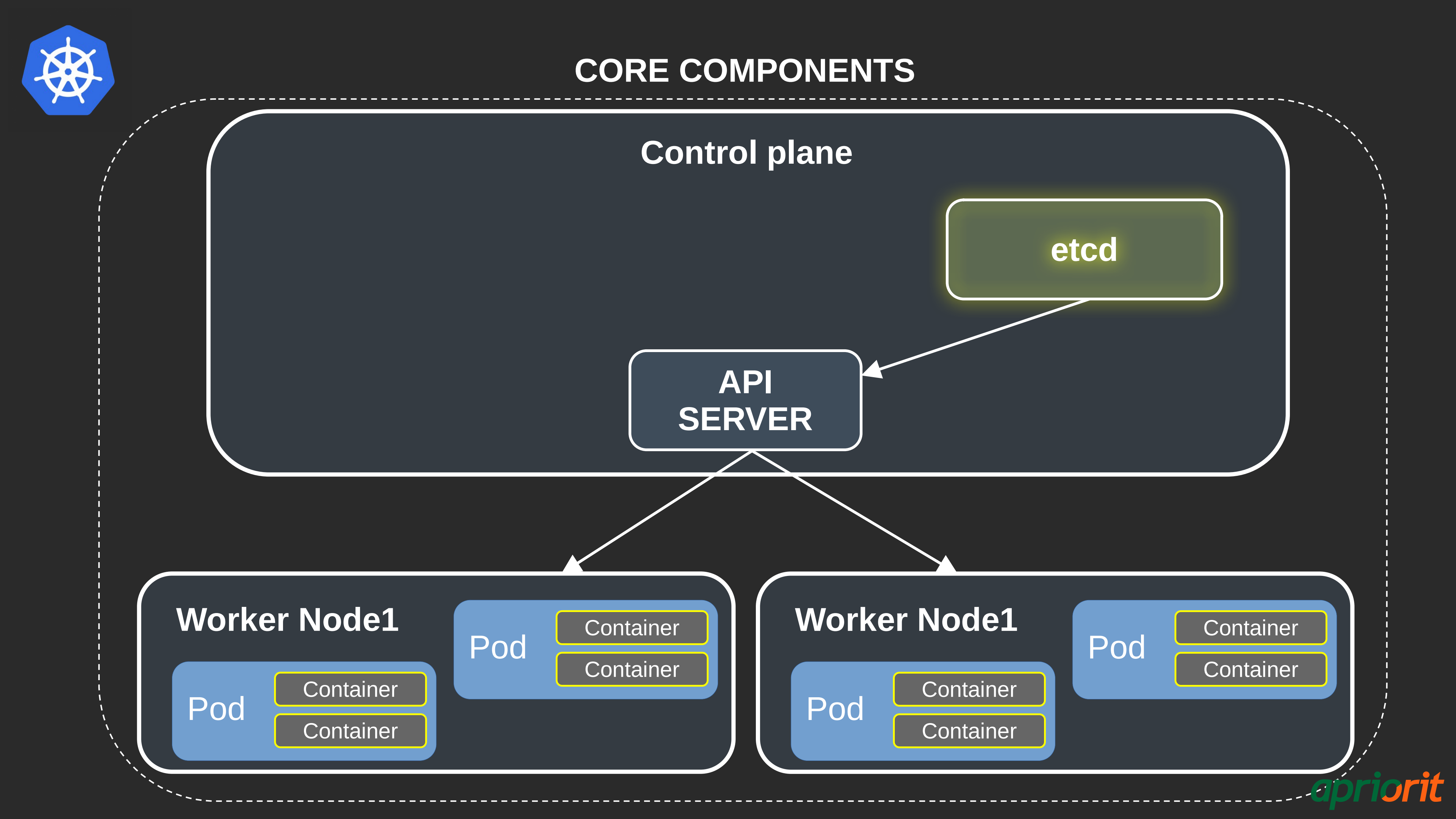

CLUSTER
Control plane

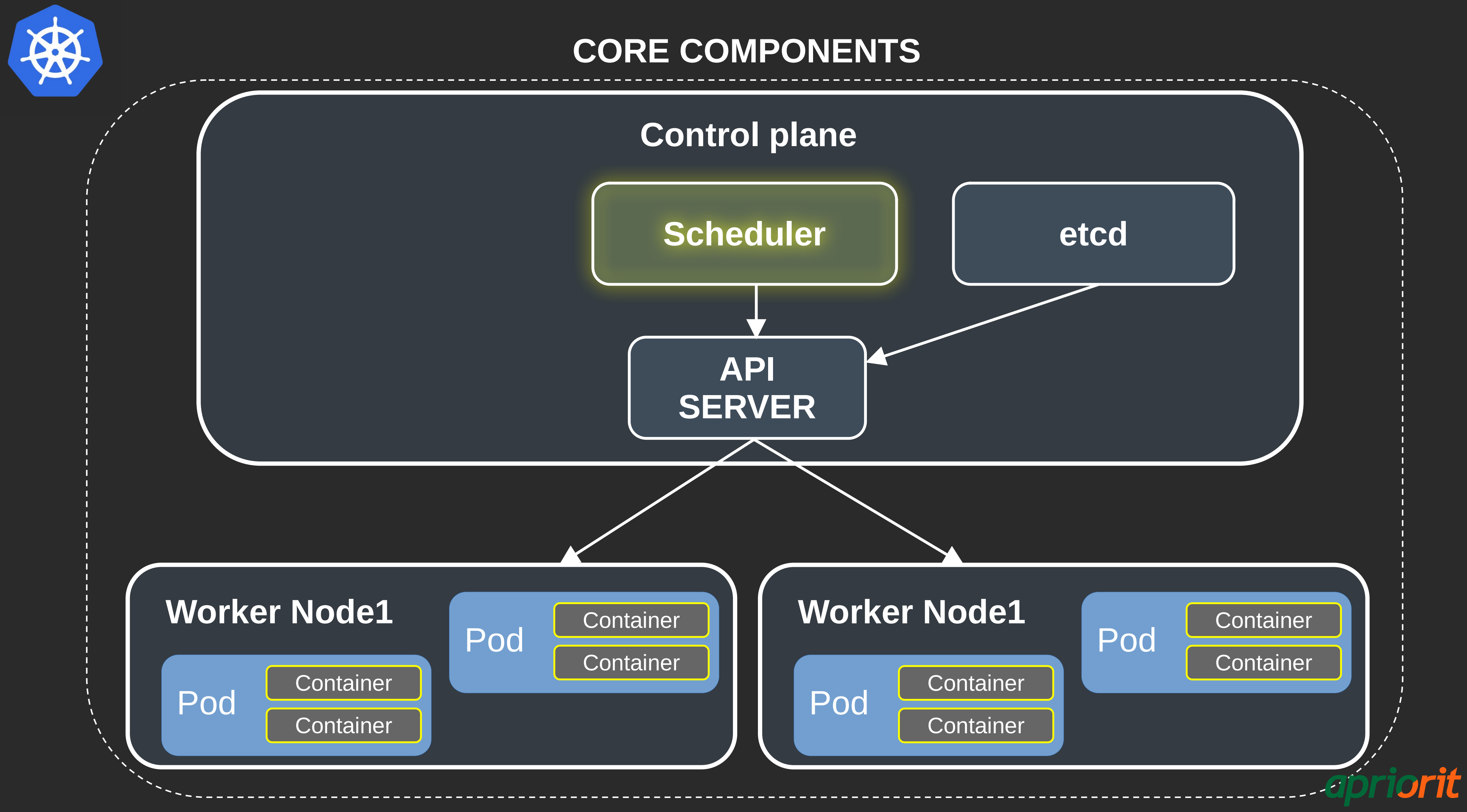

CLUSTER
Control plane
Worker Node 1
Worker Node 2

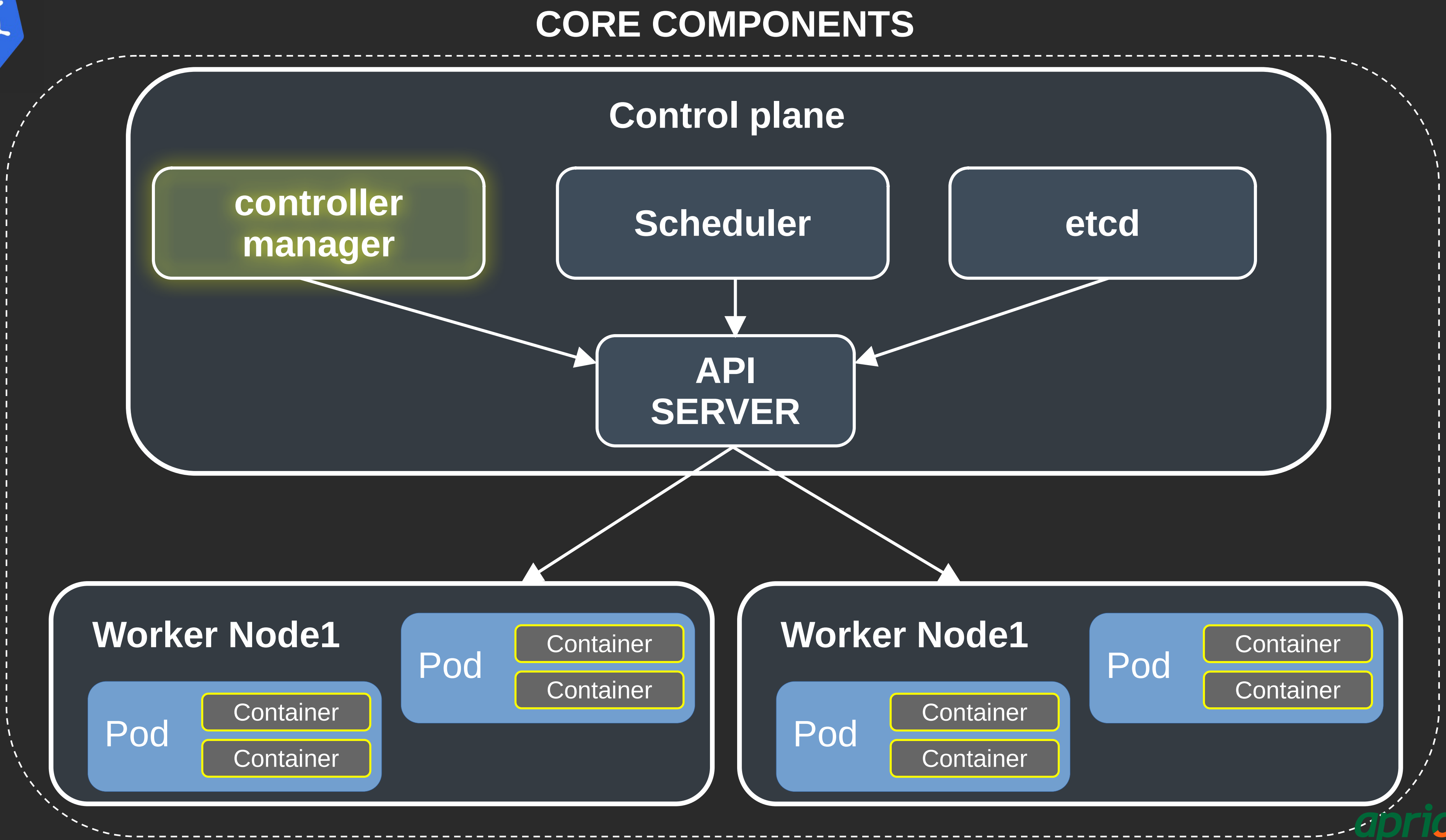

CLUSTER
Control plane
Node
Pod
Container
Container
Pod
Container
Container
Node
Pod
Container
Container
Pod
Container
Container


CLUSTER
Control plane
Node
Pod
Container
Pod
Container
Node
Pod
Container
Pod
Container

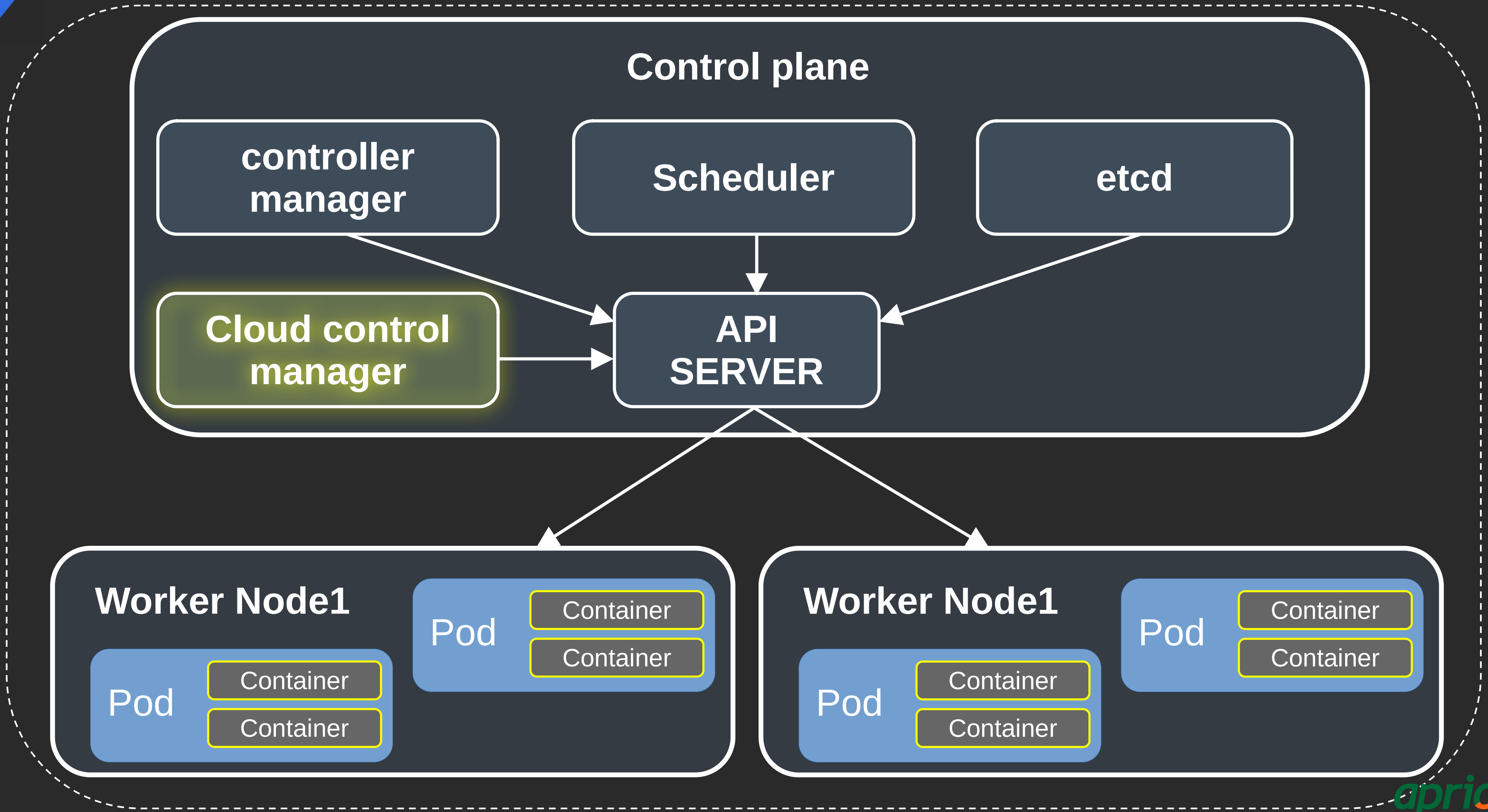

CORE COMPONENTS
Control plane
API
SERVER
Worker Node1
Pod
Container
Container
Pod
Container
Container
Worker Node1
Pod
Container
Container
Pod
Container
Container
apriorit

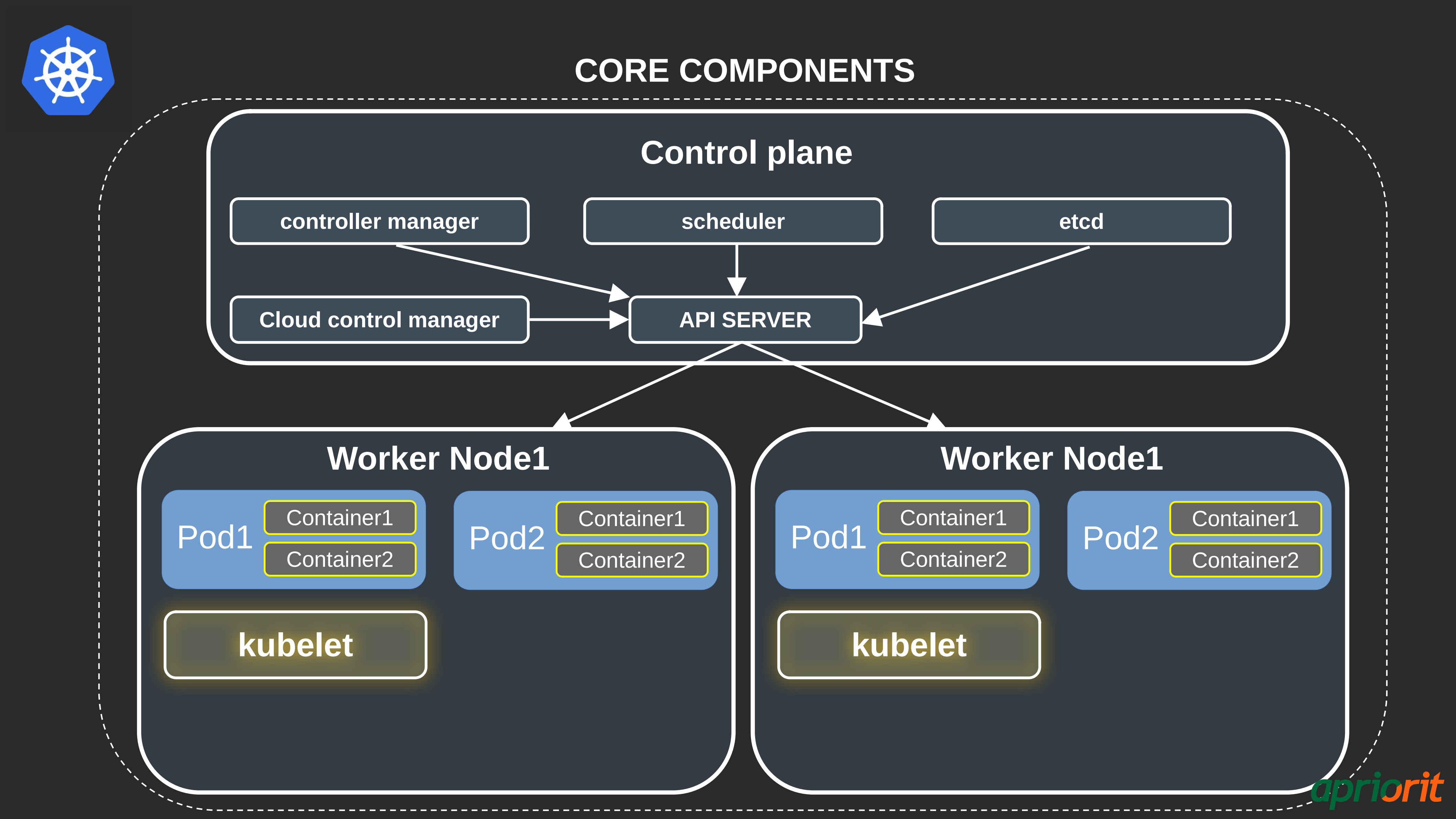

CORE COMPONENTS
Control plane
etcd
API SERVER
Worker Node1
Pod
Container
Container
Pod
Container
Container
Worker Node1
Pod
Container
Container
Pod
Container
Container
apriorit

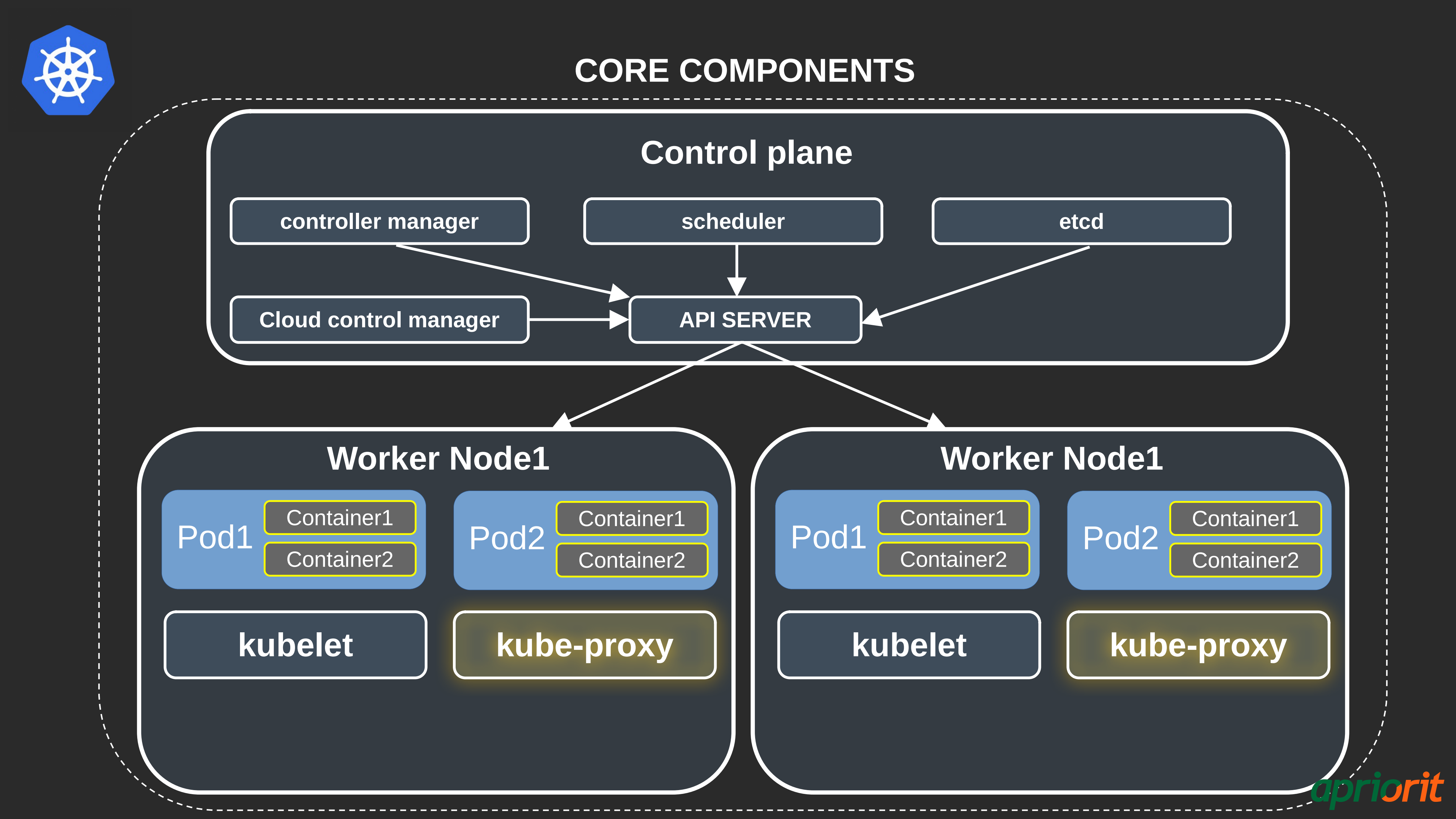

CORE COMPONENTS
Control plane
Scheduler
etcd
API
SERVER
Worker Node1
Pod
Container
Container
Pod
Container
Container
Worker Node1
Pod
Container
Container
Pod
Container
Container
apriorit

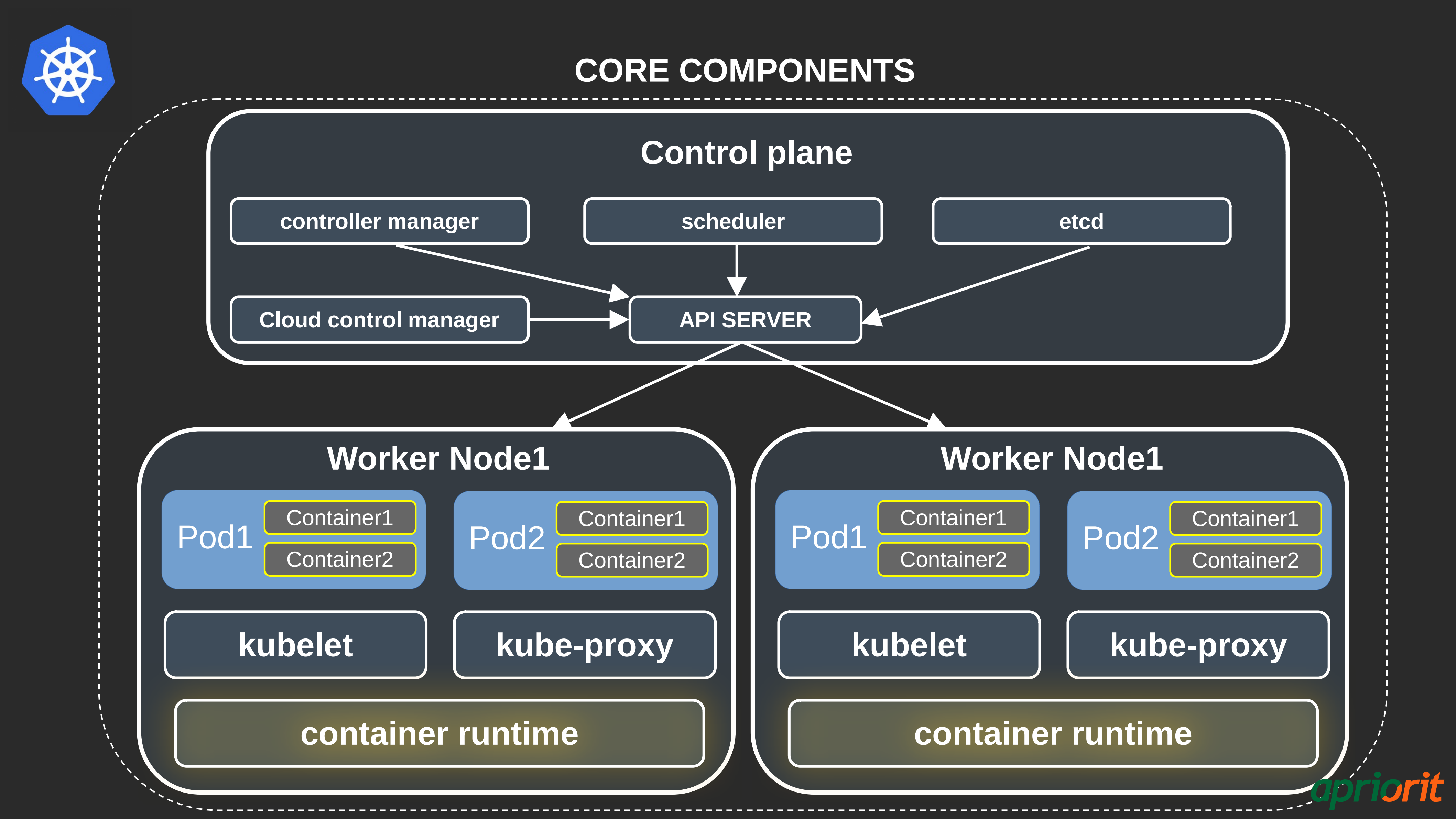

CORE COMPONENTS
Control plane
controller manager
Scheduler
etcd
API SERVER
Worker Node1
Pod
Container
Container
Pod
Container
Container
Worker Node1
Pod
Container
Container
Pod
Container
Container

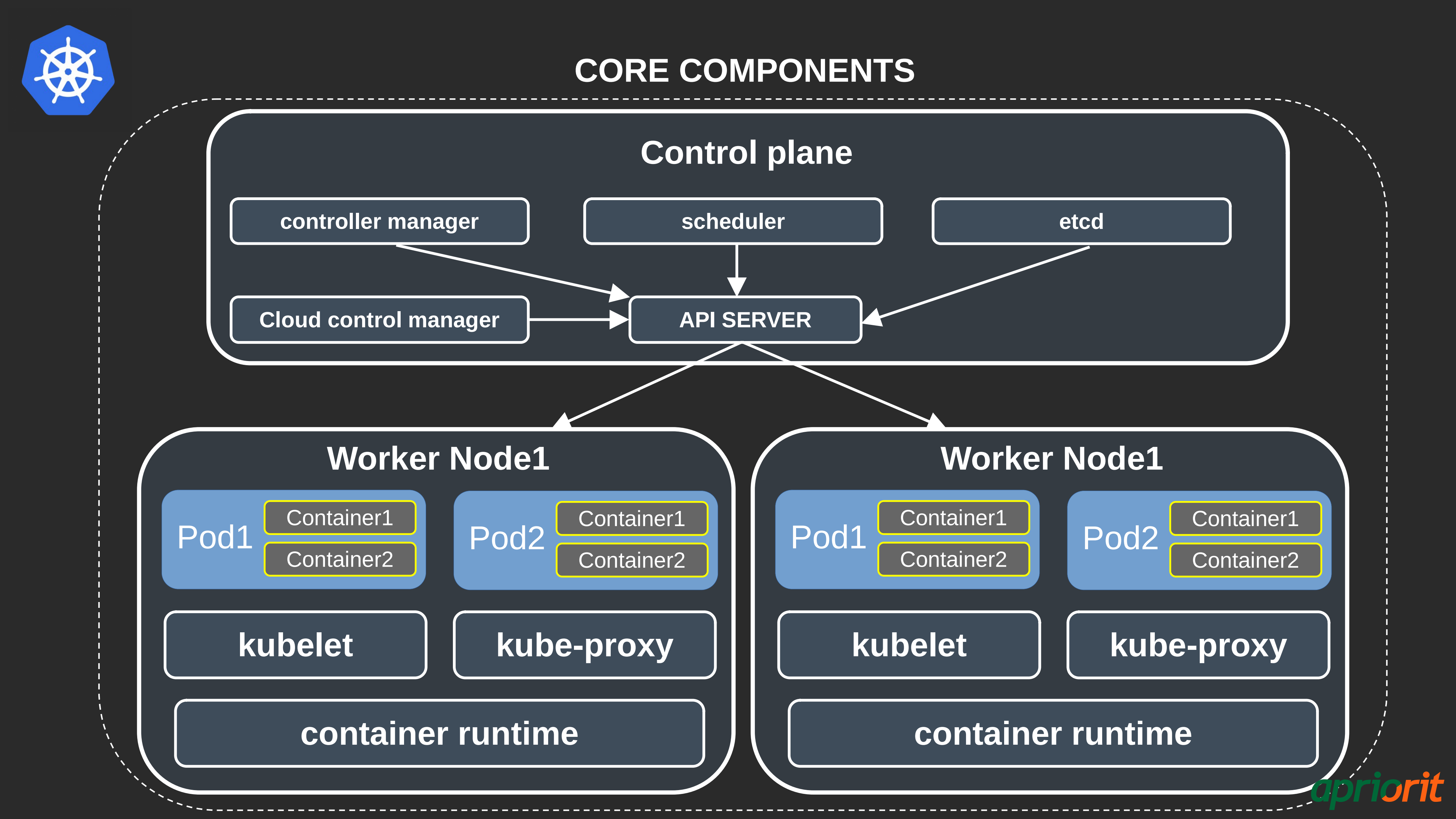
CORE COMPONENTS




CORE COMPONENTS
Control plane
controller manager
scheduler
etcd
Cloud control manager
API SERVER
Worker Node1
Pod1
Container1
Container2
Pod2
Container1
Container2
kubelet
Worker Node1
Pod1
Container1
Container2
Pod2
Container1
Container2
kubelet
apriorit


CORE COMPONENTS
Control plane
controller manager
scheduler
etcd
Cloud control manager
API SERVER
Worker Node1
Pod1
Container1
Container2
Pod2
Container1
Container2
kubelet
kube-proxy
Worker Node1
Pod1
Container1
Container2
Pod2
Container1
Container2
kubelet
kube-proxy
apriorit


CORE COMPONENTS
Control plane
controller manager
scheduler
etcd
Cloud control manager
API SERVER
Worker Node1
Pod1
Container1
Container2
Pod2
Container1
Container2
kubelet
kube-proxy
container runtime
Worker Node1
Pod1
Container1
Container2
Pod2
Container1
Container2
kubelet
kube-proxy
container runtime
apriorit


CORE COMPONENTS
Control plane
controller manager
scheduler
etcd
Cloud control manager
API SERVER
Worker Node1
Pod1
Container1
Container2
Pod2
Container1
Container2
kubelet
kube-proxy
container runtime
Worker Node1
Pod1
Container1
Container2
Pod2
Container1
Container2
kubelet
kube-proxy
container runtime
apriorit

Core Components

A Kubernetes cluster consists of a control plane and one or more worker nodes. In production environments, the control plane usually runs across multiple computers and a cluster usually runs multiple nodes, providing fault-tolerance and high availability.

Control Plane Components Manage the overall state of the cluster:

kube-apiserver
The core component server that exposes the Kubernetes HTTP API

etcd
Consistent and highly-available key value store for all API server data

kube-scheduler
Looks for Pods not yet bound to a node, and assigns each Pod to a suitable node.

kube-controller-manager
Runs controllers to implement Kubernetes API behavior.

cloud-controller-manager (optional)
Integrates with underlying cloud provider(s).

Node Components Run on every node, maintaining running pods and providing the Kubernetes runtime environment:

kubelet
Ensures that Pods are running, including their containers.

kube-proxy (optional)
Maintains network rules on nodes to implement Services.

Container runtime
Software responsible for running containers.

https://kubernetes.io/docs/concepts/architecture/

Why has Kubernetes become popular?

Automation and scalability: Kubernetes allows easy to scale of applications according to demand without the need for manual infrastructure adjustments.

Infrastructure independence: Kubernetes can run on various platforms, including cloud providers (AWS, Google Cloud, Azure), on-premise data centers, or hybrid environments. This enables organizations to migrate between infrastructures without changing their application architecture.

High availability and reliability: Kubernetes ensures automatic recovery of containers in case of failure, restarts pods, and manages resources, enhancing the stability and availability of applications. This makes it a reliable platform for production environments.

Microservices architecture: Kubernetes is ideal for deploying microservices, where applications are split into separate, independent services that interact with each other. It manages the network interactions between these services and allows easy updates and scaling of each service individually.

Declarative management: Kubernetes uses a declarative approach to infrastructure management, where users describe the desired state of the cluster (e.g., how many pods should be running), and Kubernetes automatically works to achieve that state. This simplifies the management of complex systems.

Automatic updates and rollbacks: Kubernetes supports automatic deployments of new application versions with built-in rollback capabilities in case of failures, making the update process more reliable and safer.

Cloud-native scalability: Kubernetes is designed with cloud scalability in mind. It enables efficient use of resources and helps reduce costs by scaling applications according to actual demand.

Support for DevOps practices: Kubernetes makes it easier to implement CI/CD (continuous integration and delivery), allowing for the automation of testing, deployment, and updates of applications. This makes it a key tool for teams following DevOps practices.

Extensibility and flexibility: Kubernetes is built to be customizable and extensible to meet the needs of specific companies or projects. For instance, new networking plugins, storage systems, or other functionalities can be added to the platform.

What are the disadvantages of Kubernetes?

Complex setup and management: Kubernetes is a complex system that requires deep knowledge for configuration and management. It demands expertise in containerization, networking, security, and distributed systems, making it challenging for newcomers to quickly adopt.

Steep learning curve: Mastering Kubernetes takes time. It is not a tool that can be learned quickly, especially considering the complexity of setting up clusters, managing network policies, and automating tasks.

Infrastructure costs: Kubernetes requires significant resources to run efficiently, especially at scale. This can include costs for servers, networking infrastructure, and storage. Additional expenses for maintenance and monitoring may also arise.

Resource-intensive: Kubernetes has a high overhead on system resources, particularly in smaller clusters or on low-powered machines. This can lead to inefficient resource use or the need to allocate additional capacity for optimal performance.

Support for complex network configurations: Although Kubernetes supports various network plugins, setting up and maintaining complex network policies and topologies can be challenging, especially in cloud environments or when integrating with existing networks.

Security: While Kubernetes offers built-in security mechanisms like Secrets and Service Accounts, it can be vulnerable if misconfigured. Proper setup of access control, encryption, authentication, and other security settings is critical.

Upgrades and maintenance: Upgrading Kubernetes clusters can be a difficult process, especially in large production environments. Poorly executed upgrades can lead to service outages or even failures across the entire cluster.

Debugging complexity: Troubleshooting issues in Kubernetes can be challenging due to its distributed nature and many components. Logging and monitoring often require additional tools like Prometheus or the ELK stack.

Need for additional tools: Kubernetes is rarely used in isolation — full deployment and management typically require extra tools (e.g., Helm for managing charts, Prometheus for monitoring, Fluentd for logging). This adds further complexity.

Overkill for small projects: For small projects, Kubernetes can be overly complex and heavyweight.

Kubernetes objects

- Pod
- Service
- Namespace
- ReplicaSet
- Deployment
- StatefulSet
- DaemonSet
- Job
- CronJob
- ConfigMap
- Secrets
- Ingress
- PersistentVolumeClaim
- PersistentVolume
- ServiceAccount
- Horizontal Pod Autoscaler
- Vertical Pod Autoscaler
- Taints and Tolerations

Kubernetes objects

Pod	DaemonSet	PersistentVolumeClaim
Service	Job	PersistentVolume
Namespace	CronJob	ServiceAccount
ReplicaSet	ConfigMap	Horizontal Pod Autoscaler
Deployment	Secrets	Vertical Pod Autoscaler
StatefulSet	Ingress	Taints and Tolerations

apriorit

Kubernetes objects

Pod	DaemonSet	PersistentVolumeClaim
Service	Job	PersistentVolume
Namespace	CronJob	ServiceAccount
ReplicaSet	ConfigMap	Horizontal Pod Autoscaler
Deployment	Secrets	Vertical Pod Autoscaler
StatefulSet	Ingress	Taints and Tolerations

apriorit

Kubernetes objects

- Pod
- Service
- Namespace
- ReplicaSet
- Deployment
- StatefulSet
- DaemonSet
- Job
- CronJob
- ConfigMap
- Secrets
- Ingress
- PersistentVolumeClaim
- PersistentVolume
- ServiceAccount
- Horizontal Pod Autoscaler
- Vertical Pod Autoscaler
- Taints and Tolerations

Kubernetes objects

Pod	DaemonSet	PersistentVolumeClaim
Service	Job	PersistentVolume
Namespace	CronJob	ServiceAccount
ReplicaSet	ConfigMap	Horizontal Pod Autoscaler
Deployment	Secrets	Vertical Pod Autoscaler
StatefulSet	Ingress	Taints and Tolerations

Tools used to create local Kubernetes clusters

Minikube is a lightweight Kubernetes implementation that creates a virtual machine (VM) locally and runs a single-node Kubernetes cluster inside it. https://minikube.sigs.k8s.io/docs/

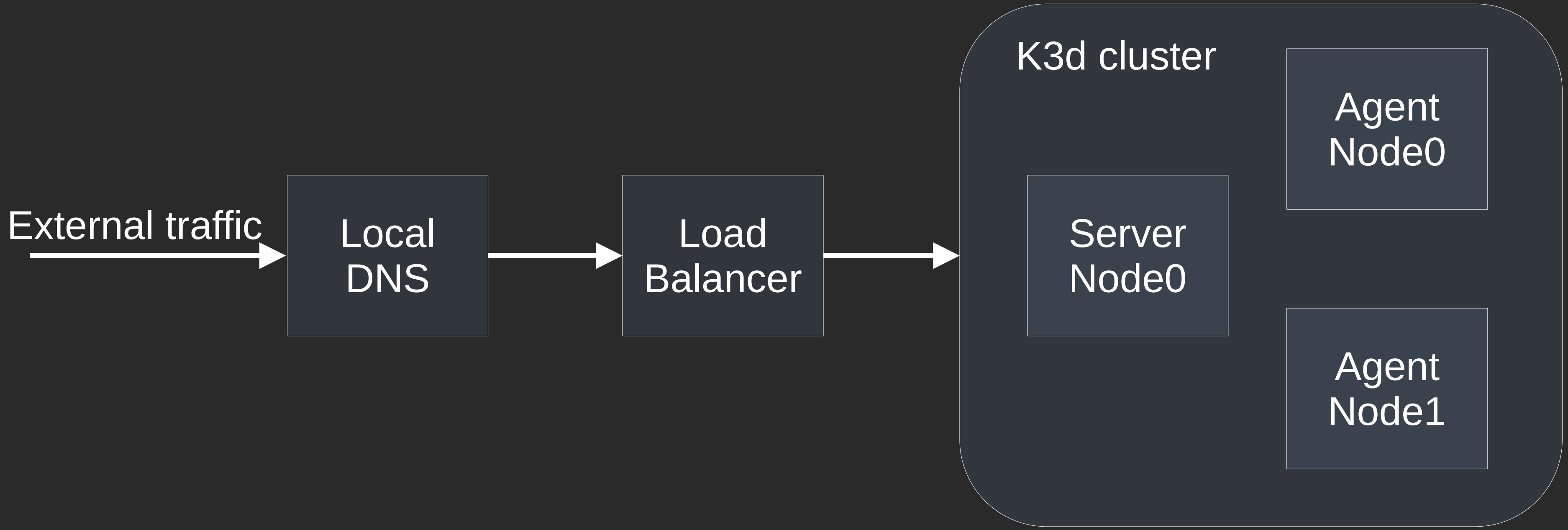
K3s is a lightweight, certified Kubernetes distribution developed by Rancher Labs. It is optimized for lower resource environments and is especially suitable for IoT devices or edge computing. https://docs.k3s.io/installation

K3d is a wrapper that runs K3s in Docker containers, enabling easy and fast local Kubernetes clusters. https://k3d.io/v5.7.4/#learning

kind (Kubernetes in Docker) - is a tool that runs Kubernetes clusters in Docker containers. https://kind.sigs.k8s.io/

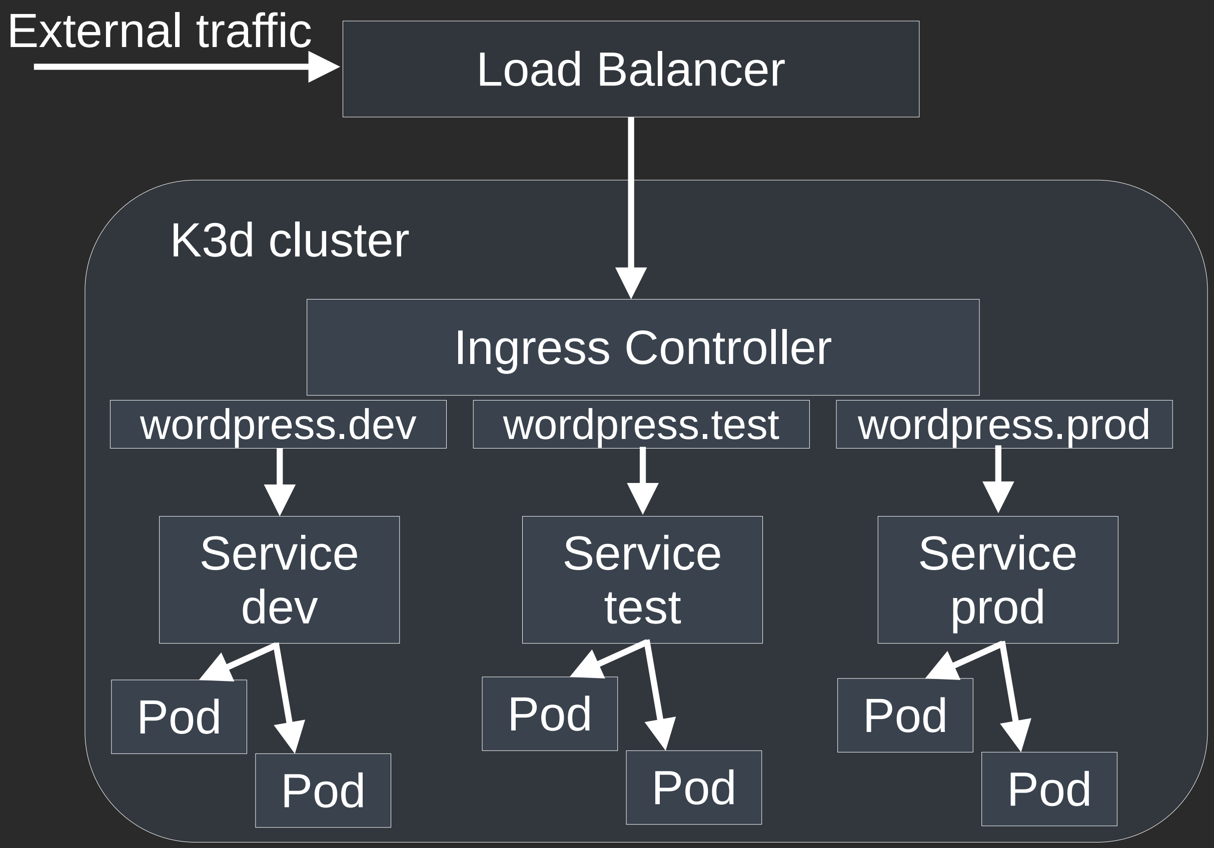
MicroK8s is a lightweight, production-grade Kubernetes distribution from Canonical, designed to run on any Linux environment, and also supports Windows and macOS. https://microk8s.io/docs/getting-started

Local k3d cluster Architecture

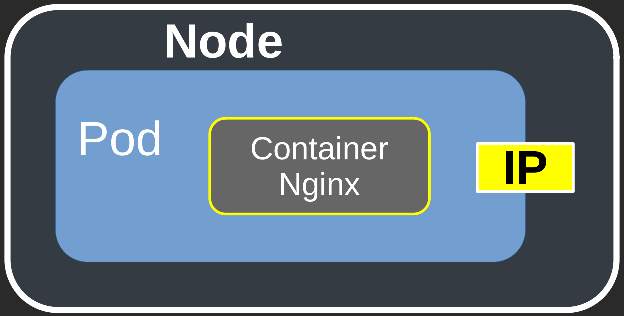



```
CONTAINER ID   IMAGE                            COMMAND                  CREATED          STATUS          PORTS                                                                                       NAMES
330a643c24a1   ghcr.io/k3d-io/k3d-proxy:latest  "/bin/sh -c nginx-pr…"   50 minutes ago   Up 50 minutes   0.0.0.0:80->80/tcp, :::80->80/tcp, 0.0.0.0:443->443/tcp, :::443->443/tcp, 127.0.0.1:6445->6443/tcp   k3d-k3dcluster-serverlb
d88ab5fed9e2   rancher/k3s:v1.30.1-k3s1         "/bin/k3d-entrypoint…"   50 minutes ago   Up 50 minutes                                                                                               k3d-k3dcluster-agent-1
ae6b1fff7cd9   rancher/k3s:v1.30.1-k3s1         "/bin/k3d-entrypoint…"   50 minutes ago   Up 50 minutes                                                                                               k3d-k3dcluster-agent-0
191373be3f28   rancher/k3s:v1.30.1-k3s1         "/bin/k3d-entrypoint…"   50 minutes ago   Up 50 minutes                                                                                               k3d-k3dcluster-server-0
```

Local k3d cluster Architecture



Pod – the smallest unit in Kubernetes



pod.yaml:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
    - image: nginx:latest
      name: nginx
      ports:
        - containerPort: 80
```

Base commands:

Deploy:

kubectl apply -f pod.yaml

Check status:

kubectl get pods

Get detailed information about the Pod:

kubectl describe pod my-pod

View logs:

kubectl logs my-pod

Exec into container:

kubectl exec -it my-pod -- /bin/sh

Delete:

kubectl delete pod my-pod

ReplicaSet

replicaset.yaml:

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: nginx-replicaset
  labels:
    name: app-nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: app-nginx
  template:
    metadata:
      labels:
        app: app-nginx
    spec:
      containers:
        - name: app-nginx
          image: nginx:stable-alpine
          resources:
            limits:
              memory: "128Mi"
              cpu: "500m"
          ports:
            - containerPort: 80
```

Base commands:

Deploy:

kubectl apply -f replicaset.yaml

Check status:

kubectl get pods

View ReplicaSets:

kubectl get replicaset

Scale the ReplicaSet:

kubectl scale replicaset nginx-replicaset --replicas=5

Describe the ReplicaSet:

kubectl describe replicaset nginx-replicaset

Delete a ReplicaSet:

kubectl delete replicaset nginx-replicaset

RBAC

RBAC ROLE

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: pod-reader
rules:
- apiGroups: [""] # "" indicates the core API group
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
```

RBAC

RBAC ClusterRole

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  # "namespace" omitted since ClusterRoles are not
namespaced
  name: secret-reader
rules:
- apiGroups: [""]
  #
  # at the HTTP level, the name of the resource for
accessing Secret
  # objects is "secrets"
  resources: ["secrets"]
  verbs: ["get", "watch", "list"]
```

RBAC

RoleBindings

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: pod-reader
rules:
- apiGroups: [""] # "" indicates the core API group
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
```

HELM

https://helm.sh/docs/intro/install/

Directory Structure

```
wordpress-chart/
├── Chart.yaml
├── values.yaml
├── templates/
│   ├── namespace.yaml
│   ├── secret.yaml
│   ├── storageclass.yaml
│   ├── pv.yaml
│   ├── pvc.yaml
│   ├── mysql-headless-service.yaml
│   ├── mysql-statefulset.yaml
│   ├── mysql-service.yaml
│   ├── wordpress-service.yaml
│   ├── wordpress-deployment.yaml
│   └── ingress.yaml
```

Home Work

- Install and configure a local Kubernetes cluster using k3d, or Kind, or Minikube.

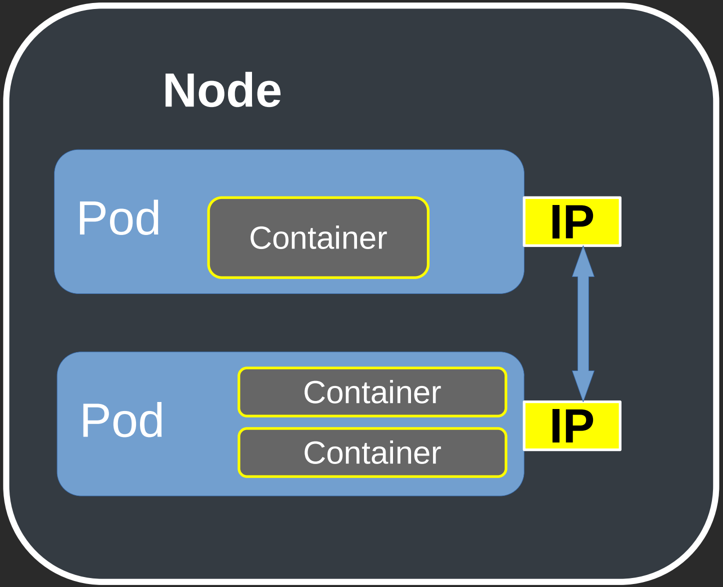
- Deploy a WordPress site with a MySQL database using Kubernetes resources such as Deployments, StatefulSets, Services, Ingress, and Persistent Volumes. The images should be pulled from Docker Hub.

- Use **kubectl logs** and **kubectl describe** to troubleshoot any issues with the WordPress deployment.

- The task is complete when the WordPress site is accessible through a browser.

- Push your files to the GitHub repository.

* Create a Terraform script to install a local Kubernetes cluster and deploy an application on it, using terraform and Helm.



Q & A