INTRODUCTION TO KUBERNETES

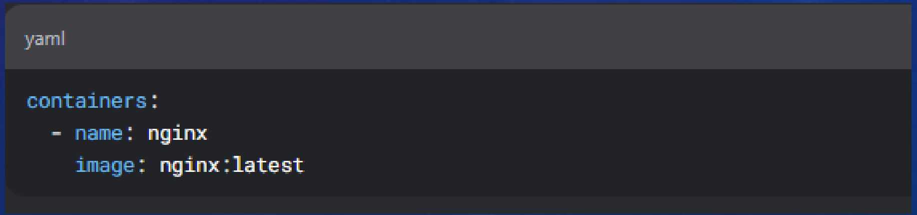
- Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications.
- Open-source platform for automating deployment, scaling, and management of containerized applications.
- Developed by Google, now maintained by the Cloud Native Computing Foundation (CNCF).
- Addresses challenges of running containers at scale.
- Core features:
  - Container orchestration
  - Self-healing (restarts, rescheduling)
  - Load balancing and service discovery
  - Automated rollouts and rollbacks

BENEFITS OF KUBERNETES

- Portability: Runs on any infrastructure (cloud, on-premises, hybrid).
- Scalability: Auto-scales based on CPU, memory, or custom metrics.
- Resilience: Self-healing for failed containers or nodes.
- Efficiency: Optimizes resource usage via scheduling. Ecosystem: Supports tools like Helm, Istio, and Prometheus.

KEY CONCEPTS AND TERMINOLOGY

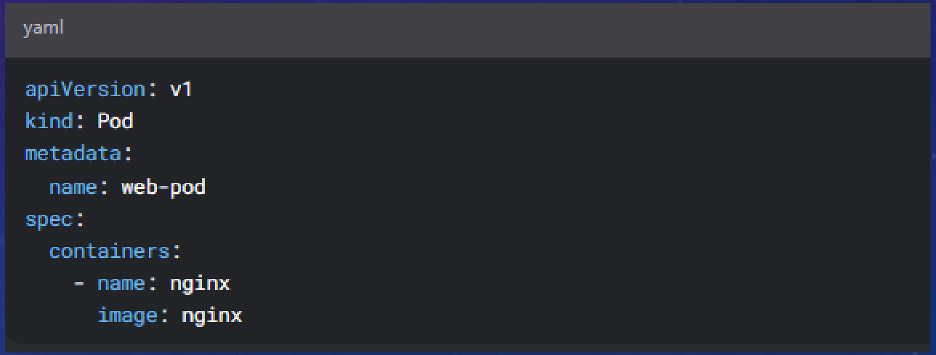
- Container
  - **Definition:**
    - A lightweight, standalone, executable software package that includes everything needed to run an application (code, runtime, libraries, dependencies).
    - Runs in isolation from other containers on the same host.
  - **Key Points:**
    - Built from **container images** (e.g., Docker images).
    - Shares the host OS kernel (unlike VMs).
    - Managed by container runtimes (e.g., Docker, containerd).
  - **Example:**



```yaml
containers:
  - name: nginx
    image: nginx:latest
```

KEY CONCEPTS AND TERMINOLOGY

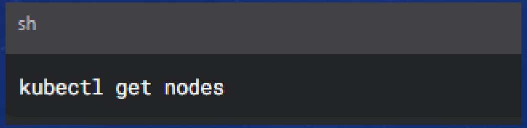
- **Pod**
  - **Definition:**
    - The **smallest deployable unit** in Kubernetes.
    - A logical group of **one or more containers** that share:
      - **Network namespace** (same IP/ports).
      - **Storage volumes**.
      - **Lifecycle** (start/terminate together).
  - **Key Points:**
    - Ephemeral (can be replaced anytime).
    - Typically runs a single main container + optional sidecar containers (e.g., log collectors).



```yaml
apiVersion: v1
kind: Pod
metadata:
  name: web-pod
spec:
  containers:
    - name: nginx
      image: nginx
```

KEY CONCEPTS AND TERMINOLOGY

- **Node**
  - **Definition:**
    - A **physical or virtual machine** (VM) that runs pods.
    - Part of a Kubernetes **cluster**.
  - **Types:**
    - **Worker Node:**
      - Runs pods (workloads).
      - Components:
        - **Kubelet** (communicates with the control plane).
        - **Kube-proxy** (manages networking).
        - **Container runtime** (e.g., Docker).
    - **Master Node (Control Plane):**
      - Manages the cluster (scheduling, scaling, etc.).
  - **Example Command:**

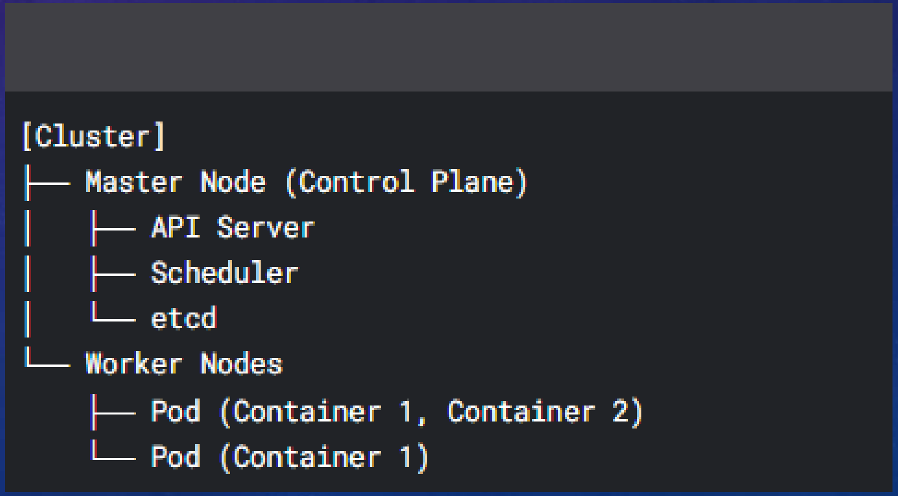


sh

```
kubectl get nodes
```

KEY CONCEPTS AND TERMINOLOGY

- **Cluster**
  - **Definition:**
    - A **set of nodes** (master + workers) that collectively run containerized applications.
  - **Key Components:**
    - **Control Plane (Master):**
      - API Server, Scheduler, Controller Manager, etcd.
    - **Worker Nodes:**
      - Execute workloads (pods).
  - **Analogy:**
    - A **cluster is like an orchestra** where:
      - The **master node** is the conductor.
      - **Worker nodes** are musicians (running pods).
- **Diagram:**



```
[Cluster]
├── Master Node (Control Plane)
│   ├── API Server
│   ├── Scheduler
│   └── etcd
└── Worker Nodes
    ├── Pod (Container 1, Container 2)
    └── Pod (Container 1)
```

KUBERNETES ARCHITECTURE OVERVIEW

- Consists of Control Plane and Node components.

CONTROL PLANE COMPONENTS

- - kube-apiserver
- - etcd
- - kube-scheduler
- - kube-controller-manager

CONTROL PLANE COMPONENTS

- kube-apiserver
  - Role:
    - The frontend of the Kubernetes control plane.
    - Exposes the Kubernetes API (RESTful interface) for all operations (e.g., kubectl commands).
  - Key Functions:
    - Validates and processes API requests.
    - Acts as a gateway to the cluster (authenticates users, authorizes requests).
    - Communicates with other components (e.g., etcd, kube-scheduler).
  - Analogy:
    - Like a reception desk that routes requests to the right department.

CONTROL PLANE COMPONENTS

- etcd
  - Role:
    - Kubernetes’ distributed key-value store (database).
    - Stores the entire cluster state (configurations, secrets, pod states, etc.).
  - Key Points:
    - Single source of truth for the cluster.
    - Highly available (typically deployed with 3+ nodes for redundancy).
    - Not directly accessed by users; only the kube-apiserver interacts with it.
  - Data Stored:
    - Cluster metadata (nodes, pods, services).
    - Desired state (YAML manifests).
    - Current state (actual pod status).
  - Analogy:
    - Like a filing cabinet that holds all cluster blueprints and records.

CONTROL PLANE COMPONENTS

- kube-scheduler
  - Role:
    - Assigns pods to worker nodes based on resource requirements, policies, and constraints.
  - Decision Factors:
    - Node resources (CPU/RAM).
    - Affinity/anti-affinity rules.
    - Taints/tolerations.
    - Pod priority.
  - Workflow:
    - Watches for newly created pods (via kube-apiserver).
    - Selects the best node for the pod.
    - Informs kube-apiserver (which updates etcd).
  - Analogy:
    - Like a wedding planner assigning guests to tables.

CONTROL PLANE COMPONENTS

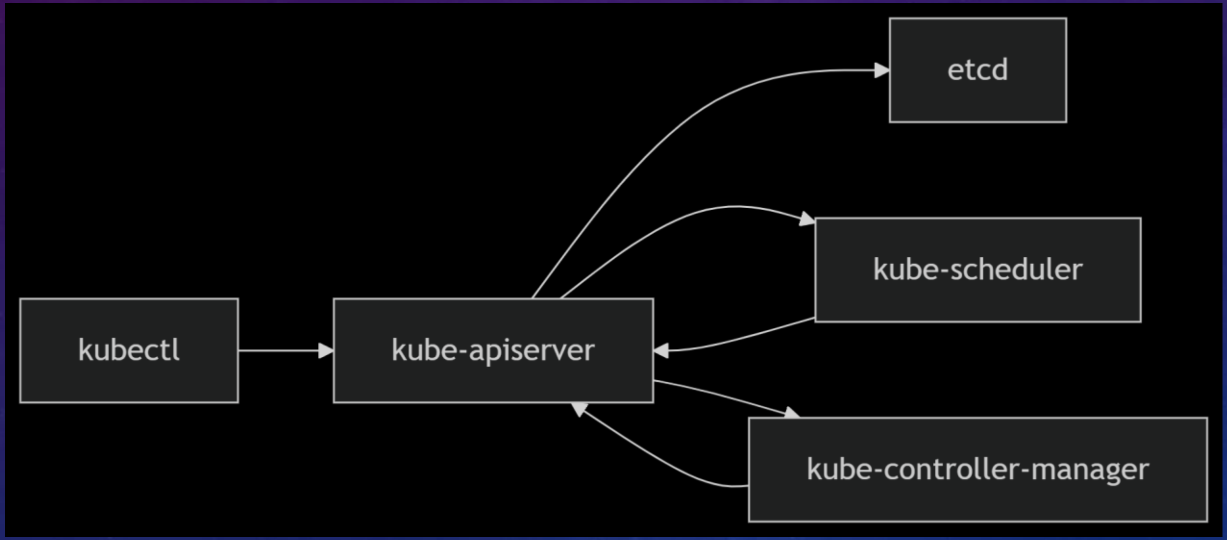
- kube-controller-manager
  - Role:
    - Runs controller processes that regulate the cluster state.
    - Ensures the actual state matches the desired state (declared in YAML).
  - Key Controllers:

Controller	Function
Node Controller	Monitors node status (up/down).
Deployment Controller	Manages replica sets for rolling updates.
Endpoint Controller	Updates services with pod IPs.
Namespace Controller	Manages lifecycle of namespaces.

  - Workflow:
    - Detects changes (e.g., a pod crashes).
    - Takes corrective action (e.g., respawns the pod).
  - Analogy:
    - Like a thermostat that keeps the room temperature constant.

Control Plane Workflow

User runs kubectl apply -f pod.yaml.
kube-apiserver validates the request → writes to etcd.
kube-scheduler assigns the pod to a node.
kube-controller-manager ensures the pod stays running.



NODE COMPONENTS

- - kubelet
- - kube-proxy
- - Container Runtime

NODE COMPONENTS

- kubelet
  - Role:
    - The primary "node agent" that runs on every worker node.
    - Ensures containers are running in a Pod as expected.
  - Key Responsibilities:
    - Receives Pod specs from the API server (via kube-apiserver).
    - Manages the lifecycle of Pods (start/stop/restart containers).
    - Reports node and Pod status back to the control plane.
    - Executes health checks (liveness/readiness probes).
  - Critical Behavior:
    - Only manages containers created by Kubernetes.
    - Does NOT manage containers created directly (e.g., via Docker CLI).
  - Analogy:
    - Like a foreman on a construction site, ensuring workers (containers) follow the blueprint (Pod spec)

NODE COMPONENTS

- kube-proxy
  - Role:
    - Maintains network rules on nodes to enable communication to/from Pods.
    - Implements Kubernetes Service concepts (ClusterIP, NodePort, LoadBalancer).
  - How It Works:
    - Uses iptables (default) or IPVS to forward traffic to Pods.
    - Ensures each Pod gets a unique IP (even across nodes).
    - Handles load balancing for Service traffic.
  - Key Rules It Manages:
    - "When traffic arrives for Service X, route it to Pod Y."
    - "When a Pod dies, update routing rules."
  - Analogy:
    - Like a traffic cop directing network packets to the right Pods.

NODE COMPONENTS

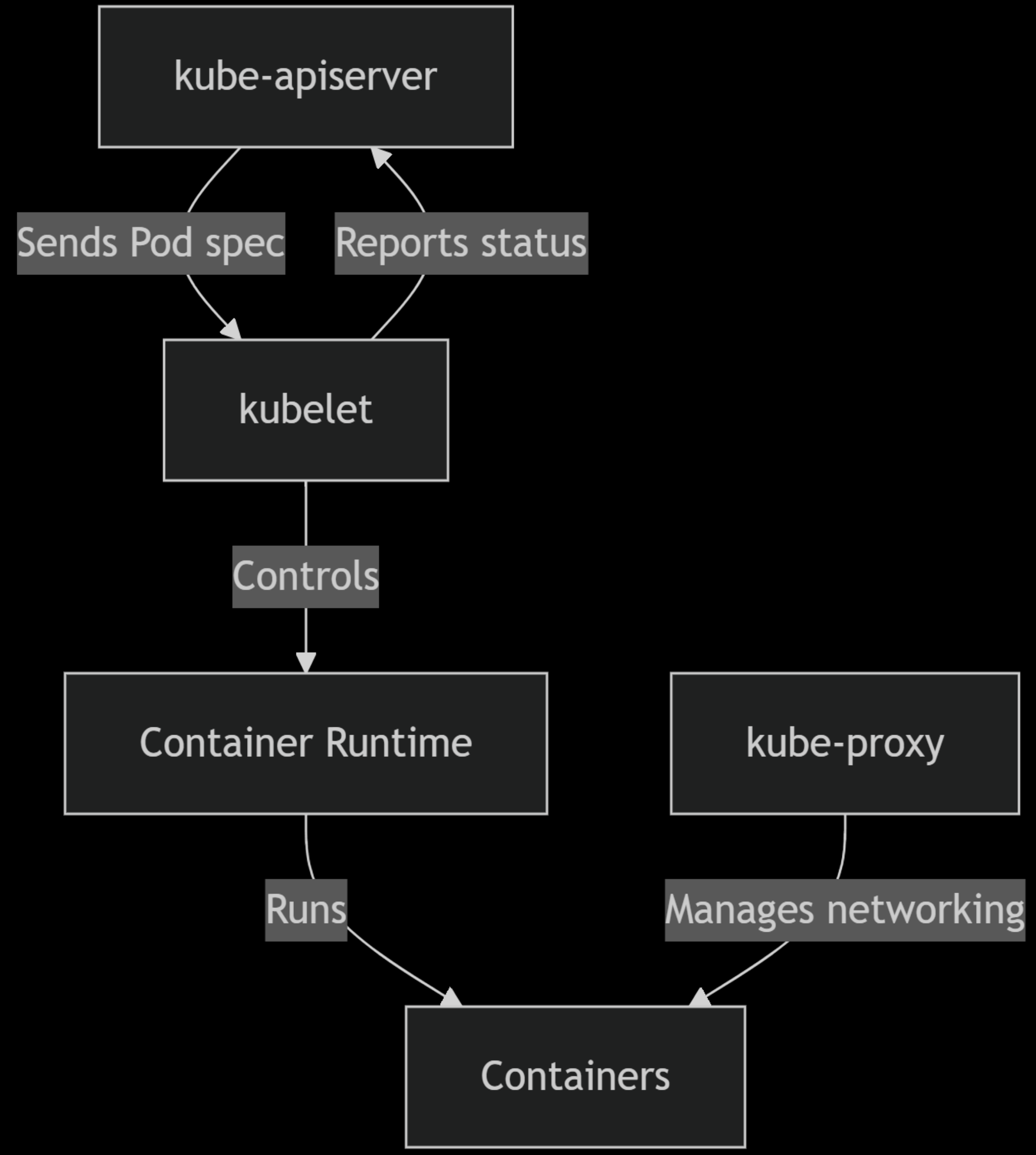
- Container Runtime
  - Role:
    - The underlying software that executes containers (runs the processes inside containers).
    - Interfaces with the OS kernel to provide isolation (cgroups, namespaces).
  - Common Runtimes:

Runtime	Description
containerd	Default in modern Kubernetes (lightweight, part of Docker).
Docker (deprecated)	Older versions used Docker Engine (now replaced with containerd).

  - Key Tasks:
    - Pulling container images from registries.
    - Running/stopping containers.
    - Managing storage/network for containers.
  - Analogy:
    - Like a construction worker who actually builds the house (container) from the blueprint (image).

HOW THEY WORK TOGETHER

- Control Plane schedules a Pod to a Node.
- kubelet (on that Node) receives the Pod spec.
- Container Runtime creates the containers.
- kube-proxy sets up networking rules for the Pod.



UNDERSTANDING CLUSTERING

- A cluster is a set of nodes that run containerized applications managed by Kubernetes.

KUBERNETES NETWORKING BASICS

- - Each Pod gets a unique IP
- - Communication within cluster and outside
- - Service discovery

POD LIFECYCLE AND COMMUNICATION

- Pod is the smallest deployable unit in Kubernetes.
- Pods can communicate via localhost within themselves.

INTRODUCTION TO SERVICES

- - ClusterIP
- - NodePort
- - LoadBalancer

DEPLOYMENTS AND REPLICASETS

- Deployment manages ReplicaSets which manage Pods.

NAMESPACES AND RESOURCE MANAGEMENT

- Namespaces divide cluster resources between users.
- Use Resource Quotas and Limits for control.

KUBERNETES YAML FILES

- Kubernetes objects are defined in YAML: apiVersion, kind, metadata, spec

INSTALLING KUBERNETES

- - Minikube (for local)
- - kubeadm (for production-like setups)

SETTING UP A LOCAL CLUSTER

- Install Minikube, start with `minikube start`

SETTING UP A MULTI-NODE CLUSTER

- Use `kubeadm init` and `kubeadm join` commands.

BASIC KUBECTL COMMANDS

- `kubectl get`, `describe`, `create`, `apply`, `delete`

MONITORING AND LOGGING BASICS

- - `kubectl logs`
- - Metrics Server
- - Prometheus/Grafana

SCALING APPLICATIONS

- `kubectl scale deployment <name> --replicas=3`

COMMON TROUBLESHOOTING TIPS

- - Check pod status with `kubectl get pods`
- - Logs and events
- - Describe resources

SUMMARY AND LEARNING RESOURCES

- Review key points.
- Resources: kubernetes.io, Katacoda, Play with Kubernetes.