

Kubernetes Fundamentals

- [Kubernetes Resources](#)
- [Kubernetes Architecture](#)
- [Kubernetes API](#)
- [Containers](#)
- [Scheduling](#)

Kubernetes Resources

In Kubernetes, resources are like building blocks used to create and manage your cluster. Each resource is an object with metadata (such as names and labels) and a desired state that defines its behavior. All objects in Kubernetes are managed by an API and stored in the etcd database.

For example, a Pod resource defines how to run a group of containers, while a Service resource manages network access to those containers. Everything you can manage with `kubectl` or the Kubernetes API - like workloads, storage, or configuration - is a resource, making them essential for defining and controlling your cluster's behavior.

The following command lists all the resources that can be managed in a Kubernetes cluster, including their names, short names, API versions, and whether they are namespaced:

```
kubectl api-resources
```

The most relevant resources for the KCNA exam are marked red

Resource Overview

You can link to the table [here](#)

Core Resources

- **Pod**
- **Service**
- **ConfigMap**
- **Secret**
- **Namespace**
- PersistentVolume
- PersistentVolumeClaim
- ServiceAccount
- Binding
- ComponentStatus
- **Endpoints**
- Event
- LimitRange

- PodTemplate
- ReplicationController
- ResourceQuota

Workloads

- **Deployment**
- **ReplicaSet**
- **DaemonSet**
- StatefulSet
- **CronJob**
- **Job**
- ControllerRevision

Networking

- **Ingress**
- NetworkPolicy
- IngressClass

RBAC (Role-Based Access Control)

- **Role**
- **RoleBinding**
- ClusterRole
- ClusterRoleBinding

Storage

- **StorageClass**
- CSIDriver
- CSINode
- CSIStorageCapacity
- VolumeAttachment

Admission Control

- MutatingWebhookConfiguration
- ValidatingWebhookConfiguration

API Extensions

- CustomResourceDefinition
- APIService

Authentication and Authorization

- TokenReview
- LocalSubjectAccessReview
- SelfSubjectAccessReview
- SelfSubjectRulesReview
- SubjectAccessReview

Autoscaling

- **HorizontalPodAutoscaler**

Certificates

- CertificateSigningRequest

Coordination

- Lease

Discovery

- EndpointSlice

Events

- Event

Flow Control

- FlowSchema
- PriorityLevelConfiguration

Node Management

- RuntimeClass

Policies

- PodDisruptionBudget

- PodSecurityPolicy

Scheduling

- PriorityClass

Kubernetes Architecture

Control Plane:

1. API Server (kube-apiserver)
2. etcd
3. Controller Manager (kube-controller-manager)
4. Scheduler (kube-scheduler)
5. Cloud Controller Manager (optional)

Node (Worker Node):

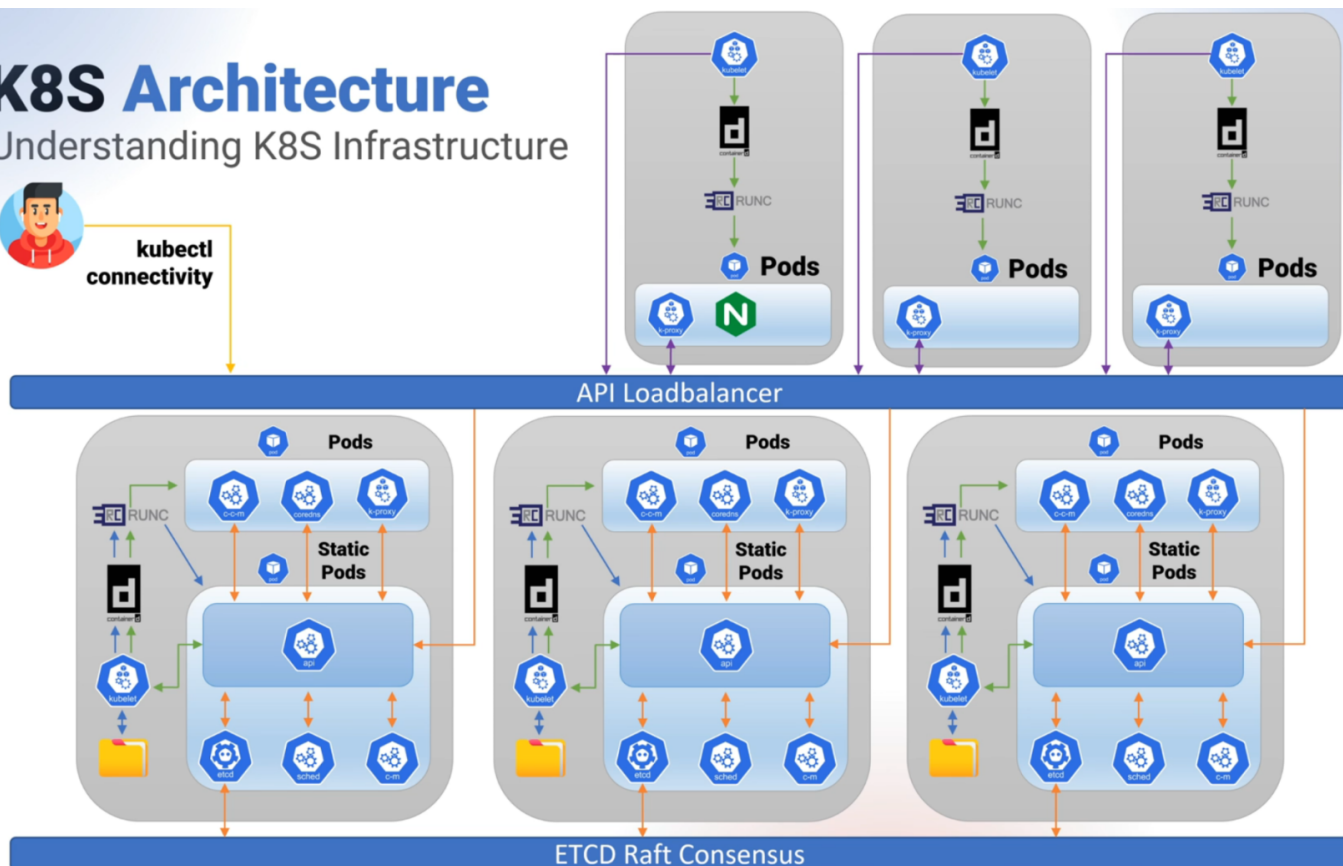
1. Kubelet
2. Kube Proxy
3. Container Runtime (z. B. containerd, CRI-O, Docker)

K8S Architecture

Understanding K8S Infrastructure



kubectl
connectivity



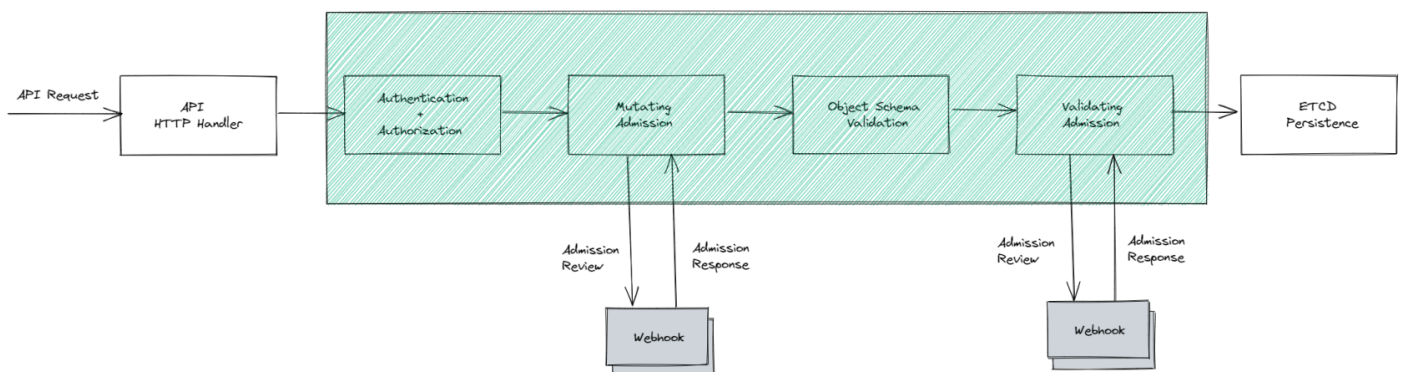
Kubernetes API

Kubernetes is an object based system - everything is managed and stored in objects, and it also provides the tools to manage those objects.

Each object contains of three parts:

1. **Metadata:** Information about the object, like its name and labels.
2. **Specification (spec):** What you want Kubernetes to do—your "desired state."
For example, "run 3 replicas of my app."
3. **Status:** What's actually happening right now—Kubernetes updates this as it works to match the current state to the desired state.

This interaction with objects happens through the Kubernetes API, which is the core interface of Kubernetes. The Kubernetes API provides a universal way for users, automation tools, and Kubernetes itself to interact with the objects stored in the system. It allows you to create, update, delete, and retrieve information about the various object resources (like Pods, Deployments, Services, etc.) within a Kubernetes cluster.



Containers

- **What is a Container?**

- Lightweight and portable.
- OS-level virtualization (compared to VMs).
- Runs applications and dependencies in an isolated environment.

- **Container Runtime:**

- OCI standards for container image format.
- Examples: `containerd`, `CRI-O`, `runc`, Docker.
- How Kubernetes interacts with container runtimes through the Container Runtime Interface (CRI).

- **Container Images:**

- Layers and image composition.
- Building images using Dockerfile.
- Storing and retrieving images from registries (Docker Hub, Artifact Registry, etc.).

- **Container Lifecycle:**

- Creation, running, stopping containers.
- Restart policies in Kubernetes (Always, OnFailure, Never).

- **Security:**

- Image scanning and vulnerability management.
- Running containers with least privilege (user ID and group).
- Container isolation using namespaces and cgroups.

- **Networking:**

- Container Network Interface (CNI) plugins.
- Port mapping and exposing services.
- Container-to-container and container-to-host communication.

- **Storage:**

- Ephemeral storage (temporary storage for containers).
- Persistent storage (using volumes in Kubernetes).
- Mounting and sharing volumes between containers.

Scheduling

- **What is Scheduling?**

- Process of assigning Pods to Nodes in a Kubernetes cluster.

- **Kubernetes Scheduler:**

- The component responsible for deciding which Node a Pod should run on.
- Works based on the available resources and the Pod's resource requirements.

- **Node Selection:**

- Factors considered by the scheduler:
 - CPU, memory resources.
 - Node affinity.
 - Taints and tolerations.
 - Pod affinity and anti-affinity.
 - Resource requests and limits.

- **Scheduling Policies:**

- Default scheduling behavior.
- Priority scheduling (priority classes).
- Custom scheduling policies (e.g., using `kube-scheduler` with custom rules).

- **Affinity and Anti-Affinity:**

- **Node Affinity:** Constraints that allow or prevent Pods from being scheduled on specific nodes.
- **Pod Affinity:** Ensures that Pods are scheduled on nodes where other specific Pods are running.
- **Pod Anti-Affinity:** Ensures Pods are not scheduled on the same node as other specified Pods.

- **Taints and Tolerations:**

- **Taints:** Applied to Nodes to prevent Pods from being scheduled unless they tolerate the taint.
- **Tolerations:** Allow Pods to be scheduled on Nodes with specific taints.

- **Resource Requests and Limits:**

- **Requests:** Minimum amount of resources a Pod needs (scheduler uses this to place Pods).
- **Limits:** Maximum amount of resources a Pod can use (enforced during execution).

- **Preemption:**

- When a Pod with higher priority displaces lower-priority Pods to be scheduled.

- **Scheduler Extenders:**

- Custom schedulers or extender mechanisms to modify default scheduling decisions.

- **Pod Disruption Budgets (PDB):**

- Used to manage voluntary disruptions (like upgrades) to ensure availability during disruptions.