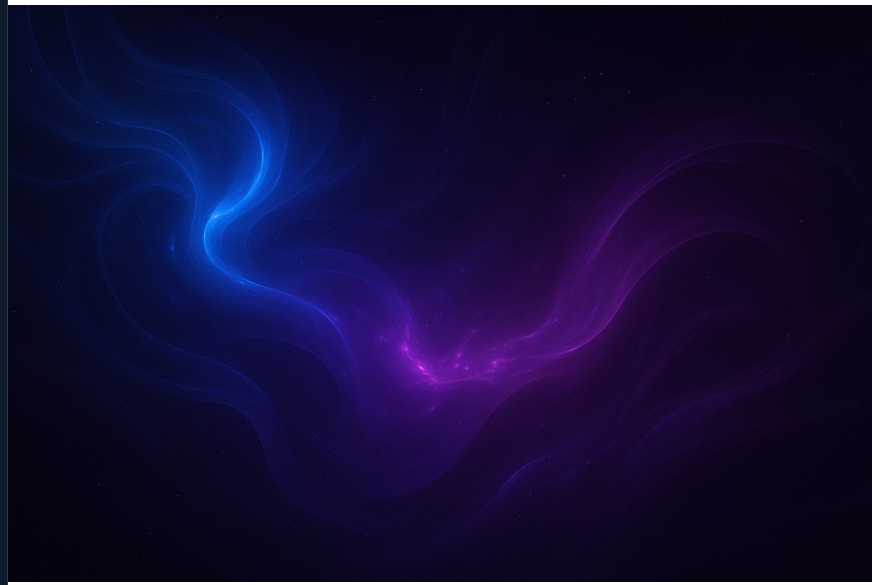
Kubernetes Administrator Bootcamp

Become a Kube Astronaut



20 September 2025

Course Overview

1. Fundamentals & Architecture
2. Objects & Workloads
3. Networking & Services
4. Storage & Configuration
5. Resource Management & Health
6. Scheduling & Placement
7. Autoscaling & Performance
8. Security & Access Control
9. Extensions & Observability
10. Installation & Troubleshooting

Why Kubernetes?

Scalability

Manage hundreds of containers across nodes, scale up/down based on demand, and maintain service availability.

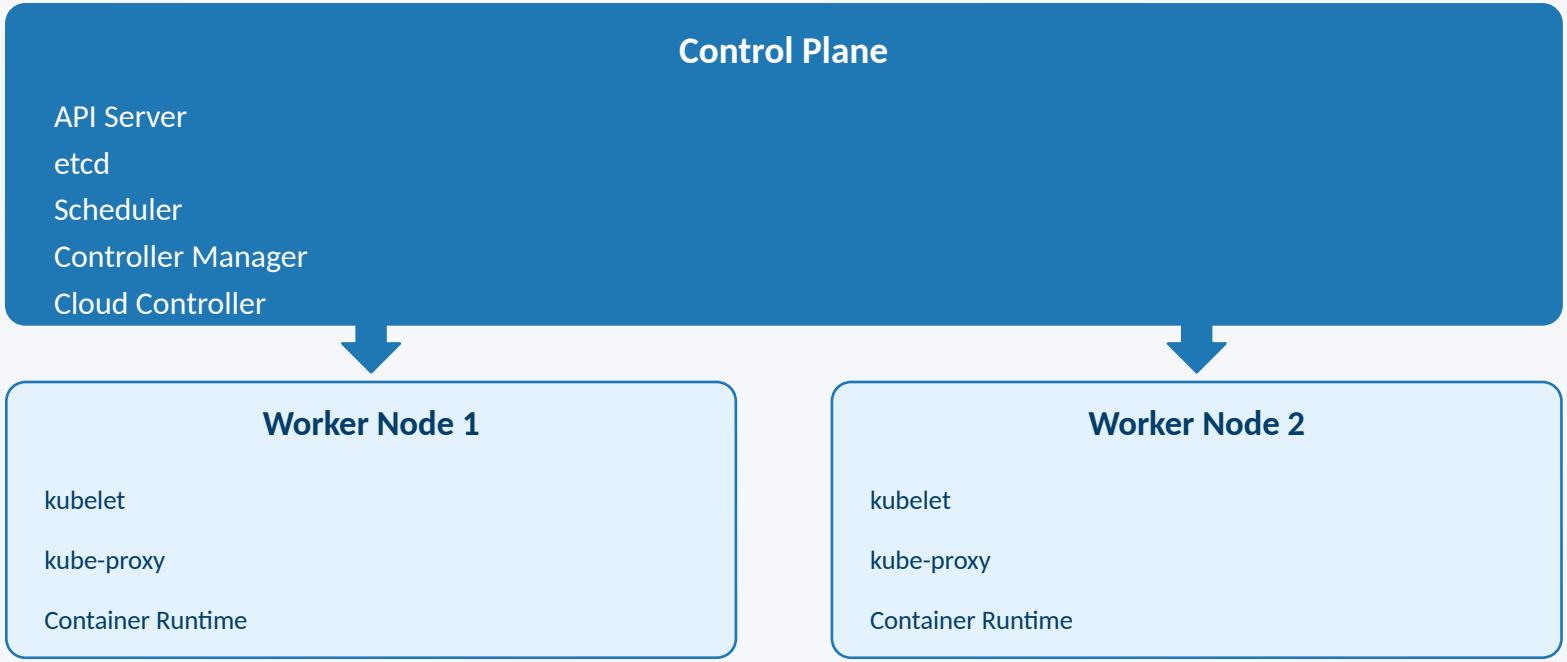
Reliability

Self-healing and rolling updates keep applications running with minimal downtime and consistent releases.

Portability

Runs on any infrastructure: on-prem, public cloud or hybrid. Declarative configs enable repeatable deployments.

Cluster Architecture



[1][2]

Control Plane Components

API Server
Exposes the Kubernetes API. All control plane components and external clients interact through it.

Scheduler
Watches for new Pods and assigns them to nodes based on resource requirements, constraints and affinity rules.

Cloud Controller
Integrates Kubernetes with cloud-provider APIs for load balancers, persistent volumes and node lifecycle.

etcd
Highly-available key-value store backing all cluster data and configuration.

Controller Manager
Runs controllers that handle routine tasks such as node health, job management and endpoint reconciliation.

[3]

Node Components

kubelet

Agent running on each node that ensures containers described in Pod specs are running and healthy.

kube-proxy

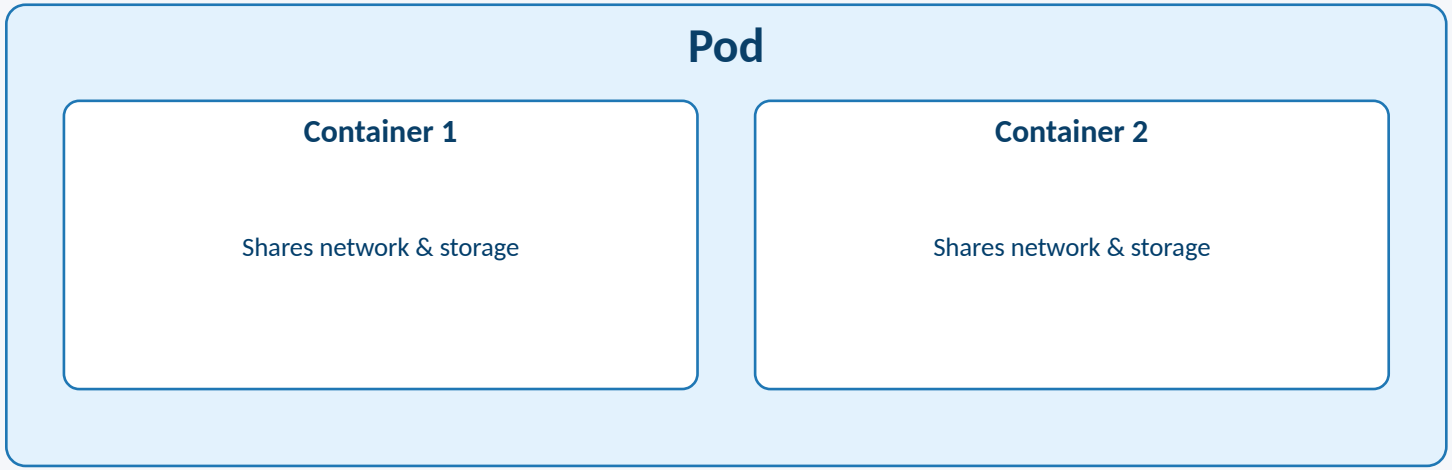
Implements the Service concept by managing network rules on each node and proxying requests to Pods.

Container Runtime

Software such as containerd or CRI-O that runs and manages containers as instructed by the kubelet.

[4]

Pods: Smallest Deployable Unit



Pods are the smallest deployable unit in Kubernetes. A pod represents one or more containers that share the same IP address, storage volumes and namespaces.

[5]

ReplicaSets & Deployments

ReplicaSet

Maintains a specified number of replica Pods

-
- Replaces Pods if they crash or are deleted
-
- Used by Deployments

Deployment

- Manages ReplicaSets and rollout strategies
- Performs rolling updates: at least 75% pods available, up to 125% pods during updates
- Enables rollbacks and versioning

[6]

StatefulSet, DaemonSet & Jobs

StatefulSet

Manages stateful applications with stable network identities and persistent volumes.

DaemonSet

Ensures a copy of a Pod runs on every node (or selected nodes). Useful for logging, monitoring agents.

Jobs/CronJobs

Runs one-off or scheduled tasks. Jobs ensure completion; CronJobs schedule repeating runs.

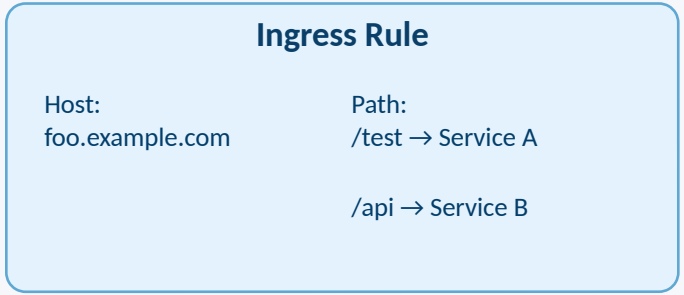
Services & Networking

ClusterIP	Exposes a service on a cluster-internal IP. Reachable only within the cluster (default).
NodePort	Exposes the service on each node’s IP and static port. Useful for simple external access.
LoadBalancer	Creates an external load balancer in cloud environments and assigns a public IP.
ExternalName	Maps the service to an external DNS name via a CNAME without proxying.

[7]

Ingress & HTTP Routing

Ingress is an API object that exposes HTTP and HTTPS routes from outside the cluster to Services within the cluster. It may provide load balancing, SSL/TLS termination and name-based virtual hosting.
An Ingress controller fulfills these rules. Only creating an Ingress resource has no effect without a controller.



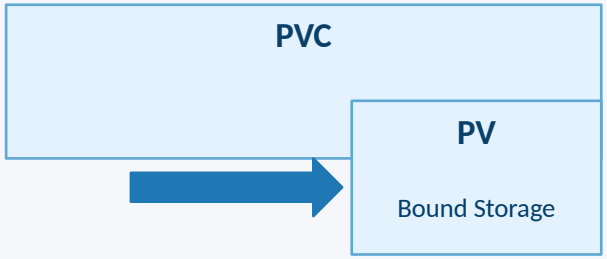
[8][9][10]

Persistent Volumes & Claims

PersistentVolume (PV)

- Cluster-level resource representing storage provisioned by administrators or dynamic provisioners.
- Lifecycle independent of Pods.
- PersistentVolumeClaim (PVC)

- User request for storage specifying size and access mode.
- Binds to a matching PV and consumed by a Pod like a node resource.



[11]

ConfigMaps & Secrets

ConfigMap

- Stores non-confidential key-value pairs.
- Decouples environment configuration from container images.
- Consumed via environment variables, command args or mounted files.

Secret

- Stores sensitive data such as passwords, tokens or SSH keys.
- Reduces the risk of accidentally exposing confidential data in images or manifests.
- Similar usage to ConfigMaps but contents should be protected (RBAC, encryption at rest).

[12][13][14]

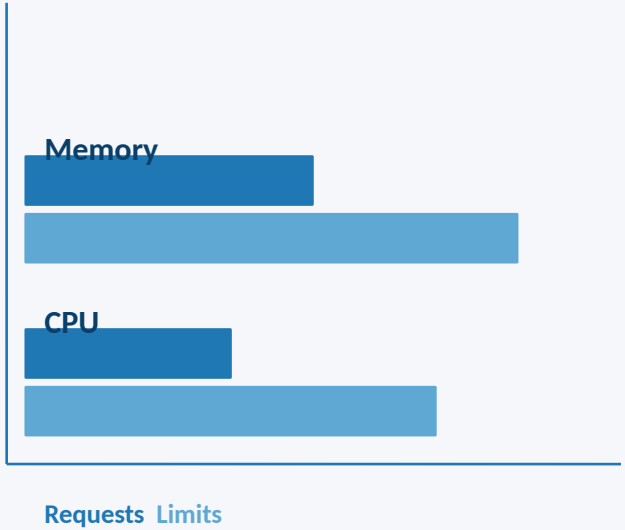
Resource Requests & Limits

Requests

- Minimum amount of CPU/Memory reserved by the scheduler for the Pod.
- Determines which node the Pod can be scheduled on.

Limits

- Maximum CPU/Memory usage enforced by the kubelet using cgroups.
- Memory limits trigger OOM kill on pressure; CPU limits throttle usage.
- If not specified, limit = request.



[15][16]

Pod Quality of Service (QoS)

Guaranteed	Every container has CPU & memory requests equal to limits; highest priority and reserved resources; cannot exceed limits.
Burstable	At least one container has a request but limits may be higher; gets guaranteed minimum resources but may use excess; intermediate eviction priority.
BestEffort	No requests or limits; uses resources only when available; lowest priority and first to be evicted under pressure.

[17]

Probes: Liveness, Readiness & Startup

Liveness

Detects if the application has entered a deadlock or crashed.

- When failed repeatedly, kubelet restarts the container.

Readiness

Determines when a container is ready to accept traffic.

- If fails, pod is temporarily removed from service endpoints.

Startup

Verifies the application has started.

- Disables liveness/readiness checks until it succeeds, avoiding premature restarts.

[18][19][20]

Scheduling: Labels & Affinity

nodeSelector

- Simplest way to constrain a pod to run only on nodes with matching labels.
- Example: nodeSelector: { disktype: ssd }

Node Affinity

- Offers more expressive placement constraints using required or preferred rules.
- Example: requiredDuringSchedulingIgnoredDuringExecution with label selector expressions.

[21][22]

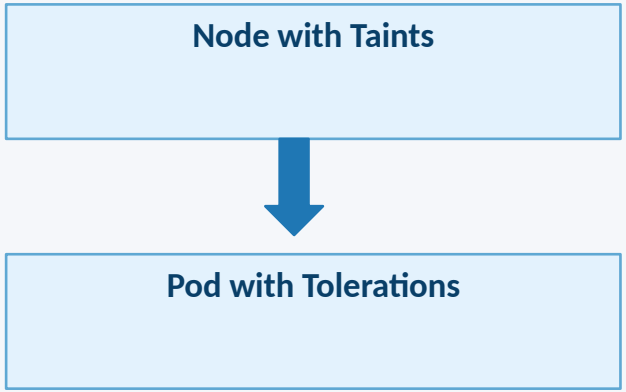
Taints & Tolerations

Taints

- Applied to nodes to repel pods.
- Consist of key, value and effect (NoSchedule, PreferNoSchedule, NoExecute).

Tolerations

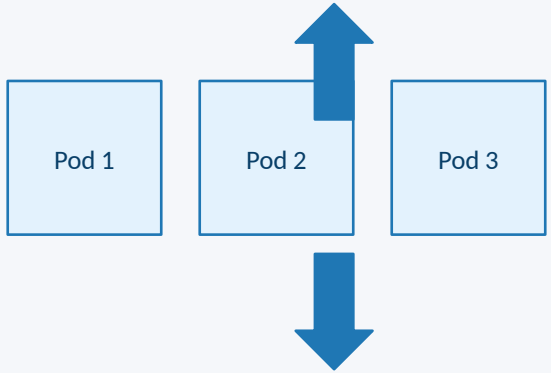
- Pods specify tolerations to match taints on nodes, allowing scheduling or continued running.
- Effect defines behaviour: NoSchedule prevents scheduling; PreferNoSchedule soft repels; NoExecute evicts existing pods.



[23]

Horizontal Pod Autoscaler (HPA)

- Automatically scales the number of pod replicas based on observed metrics (CPU, memory, custom metrics).
- Runs a control loop that periodically queries the metrics API and adjusts replicas accordingly.
- Works with Deployments, StatefulSets and other scalable resources.



[24]

Vertical & Node Autoscaling

Vertical Pod Autoscaler (VPA)

- Separate project that recommends and automatically updates CPU/Memory requests and limits for Pods.
- Modes: Auto (apply), Recreate (restart to apply), Initial (set once), Off.

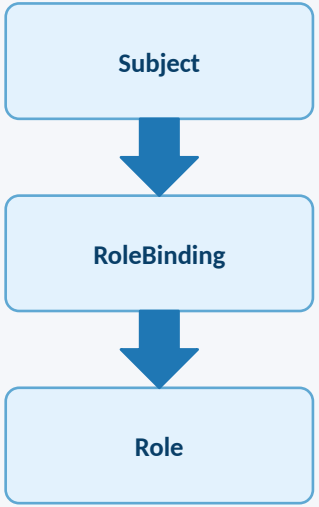
Node Autoscaling

- Cluster Autoscaler adds/removes nodes to meet pending Pod requirements.
- Karpenter auto-provisions nodes with just-in-time capacity and works across cloud providers.
- Works together with HPA to balance cost and capacity.

[25][26]

Role-Based Access Control (RBAC)

- Role: set of permissions within a namespace.
- ClusterRole: permissions across the entire cluster.
- RoleBinding: grants Role to users/groups/service accounts within a namespace.
- ClusterRoleBinding: binds a ClusterRole cluster-wide.



[27]

Service Accounts

- Non-human identities used by Pods and controllers to authenticate to the API server and external services.
- Namespaced and lightweight objects; each namespace has a default service account assigned to new Pods.
- Service accounts are distinct from user accounts and use tokens mounted into Pods for authentication.
- Use RBAC to assign least-privilege roles to service accounts.

[28]

Pod Security Standards & Admission

Privileged	Unrestricted policy allowing widest possible permissions; intended for trusted workloads.
Baseline	Minimally restrictive; prevents known privilege escalations while allowing common configurations.
Restricted	Heavily restricted; follows current best practices for pod hardening.

Pod Security Admission Modes

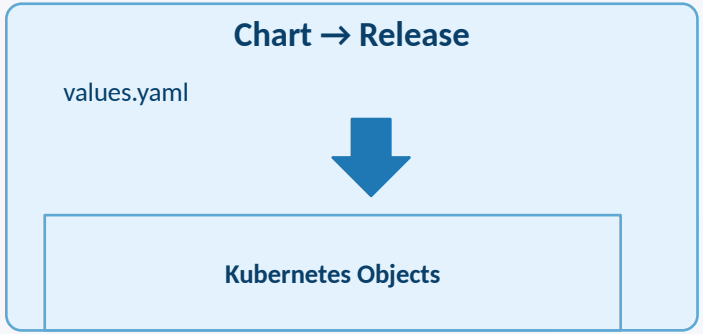
- enforce: reject pods that violate the selected policy.
- audit: allow the pod but record violation annotation in the audit log.
- warn: allow the pod but return a warning to the user.**Namespace labels**pod-security.kubernetes.io/<MODE>: <LEVEL>

Example: pod-security.kubernetes.io/enforce: restricted

[29][30][31]

Helm – Kubernetes Package Manager

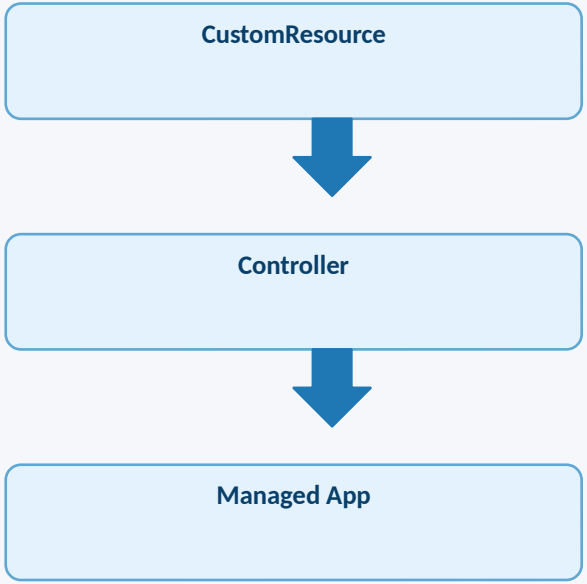
- Helm uses Charts to package all Kubernetes resources needed to deploy an application.
- Simplifies installation, upgrades and rollback of complex apps; handles dependencies and values templating.
- Supports chart repositories and sharing of community charts.



[32][33]

Operators: Extend Kubernetes

- Operators are software extensions that use Custom Resources (CRs) to manage applications and their components.
- They follow Kubernetes principles: controllers watching desired state vs. actual state (control loop).
- Examples: automate application deployment, backup/restore, upgrades and leader election.



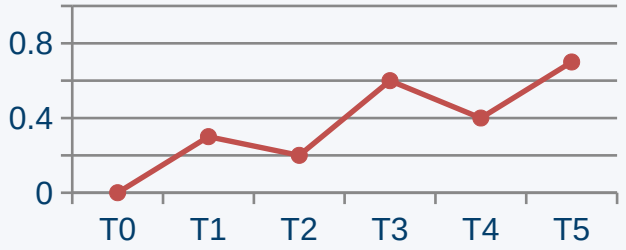
Observability with Prometheus & Grafana

Prometheus

- Open-source systems monitoring and alerting toolkit originally built at SoundCloud.
- Collects metrics as time-series data, identified by metric names and labels.
- Pulls metrics from targets via HTTP and stores them locally; provides PromQL for flexible queries.

Grafana

- Open-source analytics and visualization platform.
- Connects to multiple data sources including Prometheus, InfluxDB and Elasticsearch.
- Enables interactive dashboards, alerts and plug-in extensibility.



[35][36][37]

Debugging & Troubleshooting

kubectl describe

Fetches detailed information about Pods, including status, resource requests/limits, events and conditions. Useful for diagnosing scheduling and lifecycle issues.

kubectl logs

Prints stdout/stderr from containers. Use --previous to view logs from crashed containers.

kubectl exec

Runs commands inside a running container for investigation. For example: kubectl exec -it <pod> -- sh.

kubectl debug

Creates an Ephemeral container or copies a Pod with modified settings when your container lacks debugging utilities.

Troubleshooting Workflow

1. Check Pod status and events with kubectl get & describe.
2. Inspect container logs. Use --previous for crash loops.
3. Exec into the container for manual inspection.
4. Create debug Pods when images lack troubleshooting tools.

[38][39][40][41]

Cluster Installation with kubeadm

1. Prepare machines (install container runtime, kubeadm, kubelet, kubectl).
2. Initialize control plane: kubectl init … – downloads and installs control plane components and performs pre-checks.
3. Save kubeadm join command and configure kubeconfig (admin.conf) for kubectl access.
4. Deploy a CNI plugin: kubectl apply -f <podnetwork>.yaml.
5. Join worker nodes: kubeadm join <control-plane-host>:<port> --token <token> --discovery-token-ca-cert-hash <hash>.

Notes & Cautions

- kubeadm init runs prechecks and generates admin.conf and super-admin.conf; do not share these files.
- Always record the join command and secure the bootstrap token.
- Install only one CNI plugin per cluster; ensure Pod network CIDR does not overlap with host networks.

[42][43][44]

High Availability & Upgrades

High Availability

- Run multiple control-plane nodes behind a load balancer.
- Use stacked etcd cluster for durability (or external managed etcd).
- Ensure quorum and monitor etcd health.

Upgrades & Maintenance

- Use kubeadm upgrade to incrementally upgrade control plane and nodes.
- Drain nodes before upgrading to move Pods to other nodes (kubectl drain).
- Back up etcd regularly and test restore procedures.

Mission Accomplished

KUBERNETES
ADMINISTRATOR
BOOTCAMP
BECOME A KUBE ASTRONAUT
September 20, 2025

You are now equipped to administer Kubernetes clusters!

- Understand cluster architecture and core components.
- Deploy and scale workloads using Pods, Deployments, and Services.
- Manage storage, configuration and secrets effectively.
- Tune performance through resource management and autoscaling.
- Secure your cluster with RBAC, Pod Security and admission controls.
- Extend Kubernetes with Helm and Operators and monitor with Prometheus & Grafana.
- Install and troubleshoot clusters with kubeadm.

Next Steps:

Explore advanced networking and security features, contribute to the Kubernetes community, and consider pur
Kubernetes Administrator (CKA) certification.

[46][47][48][49][50][51][52][53][54]
[55][56][57][58][59]

Imperative vs Declarative

Imperative

- kubectl create/delete operations act directly on cluster objects
- kubectl run nginx --image=nginx --restart=Never
- kubectl create deployment web --image=nginx

Declarative

- Define desired state in YAML manifests (Pod, Deployment, Service)
- kubectl apply -f pod.yaml
- kubectl apply -f deployment.yaml
- kubectl apply -f service.yaml

[60][61]

Exercises: Cluster & Nodes

- Start a local cluster: minikube start or kind create cluster
- List nodes: kubectl get nodes
- Inspect node details: kubectl describe node <node-name>
- Label a node: kubectl label node <node-name> disktype=ssd
- Taint a node: kubectl taint node <node-name> key=value:NoSchedule

[62][63]

Exercises: Pods & Deployments

Pods

- kubectl run nginx --image=nginx --restart=Never
- kubectl get pods
- Create pod.yaml with apiVersion, kind, metadata & spec
- kubectl apply -f pod.yaml

Deployments

- kubectl create deployment web --image=nginx --replicas=3
- kubectl scale deployment/web --replicas=5
- kubectl set image deployment/web nginx=nginx:1.17
- Define deployment.yaml and apply: kubectl apply -f deployment.yaml
- kubectl rollout history deployment/web
- kubectl rollout undo deployment/web

[64][65]

Exercises: Services & Ingress

Services

- Expose deployment: kubectl expose deployment web --port=80 --target-port=80 --type=ClusterIP
- NodePort service: kubectl expose deployment web --port=80 --target-port=80 --type=NodePort
- List services: kubectl get services
- Define service.yaml and apply: kubectl apply -f service.yaml

Ingress

- Create ingress.yaml with host and path rules
- Apply: kubectl apply -f ingress.yaml
- Check ingress: kubectl get ingress
- Describe ingress: kubectl describe ingress <name>

Exercises: Storage & Configuration

ConfigMaps & Secrets

- kubectl create configmap app-config --from-literal=APP_MODE=production
- kubectl create secret generic db-secret --from-literal=password=Pa$$w0rd
- kubectl get configmap,secret
- kubectl describe configmap app-config
- Define configmap.yaml/secret.yaml & apply: kubectl apply -f configmap.yaml

Persistent Storage

- Define pv.yaml for static PersistentVolume
- Define pvc.yaml for PersistentVolumeClaim
- Apply PV & PVC: kubectl apply -f pv.yaml && kubectl apply -f pvc.yaml
- View storage: kubectl get pv,pvc

[67][68][69]

Exercises: Resource Management & Health

Requests & Limits

- Set resources in pod spec (requests & limits) and apply via YAML
- kubectl set resources deployment/web --limits=cpu=500m,memory=256Mi --requests=cpu=250m,memory=128Mi
- kubectl describe pod <pod> (view QoS class & resource usage)
- kubectl top pods (requires metrics-server)

Liveness/Readiness Probes

- Add livenessProbe/readinessProbe/startupProbe in container spec
- kubectl apply -f deployment-probes.yaml
- kubectl describe pod <pod> to view probe status

[70][71][72]

Exercises: Scheduling & Placement

Labels & Node Selectors

- Label nodes: kubectl label nodes <node> disktype=ssd zone=east
- Add nodeSelector to Pod: nodeSelector: { disktype: ssd }
- kubectl apply -f pod-selector.yaml

Affinity & Taints

- Define nodeAffinity/podAffinity/podAntiAffinity in Pod spec
- kubectl taint nodes <node> dedicated=database:NoSchedule
- Add tolerations: tolerations: [{ key: 'dedicated', operator: 'Equal', value: 'database', effect: 'NoSchedule' }]
- Apply Pod with tolerations: kubectl apply -f pod-tolerations.yaml

[73][74]

Exercises: Autoscaling & Performance

Horizontal Pod Autoscaler

- kubectl autoscale deployment web --min=2 --max=5 --cpu-percent=50
- kubectl get hpa
- Define hpa.yaml with target metrics and apply

Load Generation & VPA

- Generate load: kubectl run -i --tty load-gen --image=busybox -- /bin/sh
- Use stress tools inside the container to increase CPU usage
- Delete load generator: kubectl delete pod load-gen
- Install VPA components and apply vpa.yaml (optional)
- Use cluster autoscaler/Karpenter for node scaling (cloud-provider specific)

[75][76][77]

Exercises: Security & RBAC

RBAC

- Create service account: kubectl create serviceaccount viewer
- Define Role manifest (read pods) and apply
- Bind role: kubectl create rolebinding read-pods-binding --role=pod-reader --serviceaccount=default:viewer --namespace=default
- Test access: kubectl auth can-i list pods --as system:serviceaccount:default:viewer

Pod Security

- Label namespace to enforce baseline: kubectl label --overwrite ns dev pod-security.kubernetes.io/enforce=baseline
- Audit with restricted profile: kubectl label --overwrite ns dev pod-security.kubernetes.io/audit=restricted
- Warn on violations: kubectl label --overwrite ns dev pod-security.kubernetes.io/warn=baseline
- Verify labels: kubectl describe ns dev

[78][79][80][81]

Exercises: Extensions & Observability

Helm & Operators

- helm repo add bitnami https://charts.bitnami.com/bitnami
- helm install my-nginx bitnami/nginx
- helm upgrade my-nginx bitnami/nginx --set service.type=NodePort
- helm uninstall my-nginx
- Install operator: kubectl apply -f operator.yaml
- List custom resources: kubectl get crd
- Apply custom resource instance: kubectl apply -f cr-instance.yaml

Observability

- helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
- helm install kube-prom prometheus-community/kube-prometheus-stack
- kubectl get pods -n kube-prometheus-stack
- kubectl port-forward svc/kube-prometheus-stack-grafana -n kube-prometheus-stack 3000:80
- View logs: kubectl logs <pod>
- Follow logs: kubectl logs -f <pod>

[82][83][84][85]

Exercises: Installation & Troubleshooting

Cluster Installation

- kubeadm init --pod-network-cidr=192.168.0.0/16
- mkdir -p $HOME/.kube && cp /etc/kubernetes/admin.conf $HOME/.kube/config
- kubectl get nodes
- kubeadm join <master-ip>:6443 --token <token> --discovery-token-ca-cert-hash sha256:<hash>

Maintenance & Troubleshooting

- cordon & drain node: kubectl cordon <node>, kubectl drain <node>
- Upgrade plan/apply: kubeadm upgrade plan; kubeadm upgrade apply
- Describe resources: kubectl describe pod <pod>
- View logs: kubectl logs <pod> [--previous]
- Execute into container: kubectl exec -it <pod> -- /bin/sh
- Debug pod with temporary container: kubectl debug <pod> --image=busybox

[86][87][88][89][90][91]