Admissions & Support: support@mockattempt.com
Phone / WhatsApp: +91 98765 43210
Course Code: KUBE • **Updated:** 04 Nov 2026
https://mockattempt.com/courses/kubernetes-training

KUBE • CLOUD-NATIVE CONTAINER ORCHESTRATION (CKA / CKAD) **BEGINNER TO PRINCIPAL KUBERNETES & CLOUD-NATIVE ARCHITECT**

100% LIVE INDUSTRY EXPERT COHORT + INTERNSHIP LETTER

Kubernetes Training

Master Kubernetes 1.31 from Control Plane Reconciliation, etcd Raft Quorum & Multi-Container Pods to Rolling Deployments, Zero-Trust NetworkPolicies, CSI Dynamic PVCs, StatefulSets, Least-Privilege RBAC, HPA v2, Helm 3 & ArgoCD GitOps — with 200 Mock Tests & Voice MockAttempt AI Interviews at ₹4,499 (Incl. 18% GST).

Duration: 15–17 Weeks • Live + Hands-On Projects (212 Hrs) • **Modules:** 16 • **Live Topics:** 42 • **Subtopics:** 336+ • **Labs & Tasks:** 84 • **Mocks / AI Vivas:** 244 / 244 • **Projects:** 16 Module + 1 Live Client Prod

All-Inclusive Direct Fee (Incl. 18% GST • Zero Hidden Charges): Total Payable: ₹4,499 (Net Tuition ₹3,813 + Included 18% GST ₹686) • **Split Option:** 2 × ₹2,250 Split • **Effective ROI:** ~₹21/Guided Hr (~₹107/Live Topic)

Faculty: 100% Active Working IT Professionals • **Live Batch Tracks (7:00 AM – 8:00 PM IST Flexible Slots):** Weekday Live Online Batch (Mon – Fri • 1 Hr/Day • 7:00 AM – 8:00 PM) (Monday to Friday (1 Hour Live Session / Day) • 07:00 AM – 08:00 PM IST (Students Can Choose Any 1-Hour Slot Based on Trainer Availability) • 18–20 Weeks Fast-Track) | Weekend Live Online Batch (Sat – Sun • 2 Hrs/Day • 7:00 AM – 8:00 PM) (Saturday & Sunday (2 Hours Live Session / Day) • 07:00 AM – 08:00 PM IST (Students Can Choose Any 2-Hour Slot Based on Trainer Availability) • 20–22 Weeks Weekend Track)

Core Technologies & Engineering Stack: Kubernetes 1.31 (kubeadm / EKS / GKE) • kubectl, Kustomize & YAML Specs • Calico / Cilium CNI NetworkPolicies • CSI Dynamic Storage & StatefulSets • Least-Privilege RBAC & Pod Security • HPA v2, PDB & Helm 3 Charts • Prometheus, OpenTelemetry & ArgoCD

1. PROGRAM MASTERY OUTCOMES & ELIGIBILITY

End-to-End Engineering Outcomes:

1. Operate Kubernetes Control Plane (`kube-apiserver`, Raft `etcd`, `kube-scheduler`, `kube-controller-manager`) and worker `kubelet` CRI/CNI/CSI interfaces with confidence
2. Author zero-downtime `Deployment`s, `StatefulSet`s, `PodDisruptionBudget`s, readiness/liveness/startup probes, and dynamic CSI `PersistentVolumeClaim`s
3. Lock down multi-tenant namespaces using `ResourceQuota`, `LimitRange`, default-deny `NetworkPolicy`, KMS-encrypted `Secret`s, and least-privilege `RBAC`
4. Clear 244 Timed Mock Exams (10,000 Kubernetes CKA/CKAD MCQs) and 200 Spoken Voice + MockAttempt AI Cloud-Native Architecture Interviews
5. Control-plane & node internals (`kube-apiserver`, `etcd`, scheduler, `kubelet`, CNI/CSI) and declarative YAML/Helm design.
6. Production workload specs: Deployments, StatefulSets, DaemonSets, liveness/readiness/startup probes, and resource requests/limits.
7. Ingress TLS routing, Services, NetworkPolicies, Secrets/ConfigMaps, RBAC ServiceAccounts, and Pod Security Standards.
8. Autoscaling & HA: Horizontal Pod Autoscaler (HPA), PodDisruptionBudgets, topology spread constraints, and rolling rollbacks.

Eligibility & Prerequisites: Basic Linux CLI commands and introductory familiarity with Docker containers & YAML indentation • Any laptop/PC capable of running `kind`, `minikube`, `k3d`, or Docker Desktop Kubernetes locally • No prior Kubernetes industry experience required — Module 1 starts from environment setup, core mental models, and plain-English fundamentals. • Ideal for B.Tech / BCA / MCA / B.Sc IT students, non-CS beginners, career switchers, and working developers upgrading to 2026 production standards. • Commitment to attend live Weekday or Weekend classes, complete daily hands-on lab checklists, and practice topic-wise Mock Tests & AI Interviews.

2. DETAILED ZERO-TO-EXPERT MODULE, TOPIC & SUBTOPIC SYLLABUS (16 MODULES • 42 LIVE TOPICS • 336+ SUBTOPICS)

Progressive Zero-to-Expert Mastery Stages: Modules 1–4: Stage 1: Day-1 Foundations & Core Mental Models (8 Topics / 64 Subtopics) • Modules 5–8: Stage 2: Applied Engineering, Data & API Workflows (8 Topics / 64 Subtopics) • Modules 9–12: Stage 3: Internals, Concurrency, Security & Performance (12 Topics / 96 Subtopics) • Modules 13–16: Stage 4: Production Architecture, Testing, CI/CD & Expert Mastery (14 Topics / 112 Subtopics)

MODULE 1 OF 16 • STAGE 1 • 12 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 1: Declarative Desired-State Orchestration & Kubernetes Control-Plane Architecture

Understand level-triggered reconciliation loops, Kubernetes API object conventions (`apiVersion`, `kind`, `metadata`, `spec`, `status`), and how the Control Plane and Worker Nodes coordinate through `kube-apiserver`.

Objectives: ✓ Kubernetes Overview — Declarative Desired State, Level-Triggered Reconciliation & Cloud-Native Portability — Why container schedulers superseded Snowflake VMs, the Kubernetes API resource model (`apiVersion`, `kind`, `metadata`, `spec` vs cluster-managed `status`), self-healing control loops, bin-packing efficiency, and CNCF portability across EKS, GKE, AKS, and bare metal. • ✓ Kubernetes Architecture — `kube-apiserver`, Raft `etcd`, Scheduler, Controller Manager & `kubelet` — End-to-end Pod creation flow: `kubectl` -> `kube-apiserver` (AuthN -> AuthZ -> Admission Webhooks -> `etcd` commit) -> `kube-scheduler` (Filtering & Scoring nodes -> Binding) -> target node `kubelet` -> `containerd` CRI + CNI IP allocation. • ✓ Build, verify, and ship Module Capstone: HA 3-Control-Plane Production Cluster Topology & etcd Quorum Audit Spec.

1.1 Kubernetes Overview — Declarative Desired State, Level-Triggered Reconciliation & Cloud-Native Portability

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Declarative spec vs observed status & Level-triggered reconciliation loop • Core Syntax, CLI & Production APIs: apiVersion / kind / metadata / spec / status, kubectl api-resources, Optimistic Concurrency... • Deep-Dive Mechanics: Group / Version / Kind (GVK) & Etcd resourceVersion optimistic lock • Advanced Patterns: SharedInformer watch streams & CRI / CNI / CSI pluggable standards • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Canonical Declarative Kubernetes Object Skeleton (`v1/ConfigMap`) • Idiomatic KUBE Implementation: declarative YAML/Helm specs, Readiness/Startup probes, HPA & PDBs • Kubernetes Control Plane (apiserver, Raft etcd, Scheduler) & kubelet/CNI Execution & Memory Lifecycle for Kubernetes Overview

Purpose: To understand and apply Kubernetes Overview — Declarative Desired State, Level-Triggered Reconciliation & Cloud-Native Portability in real Kubernetes systems — specifical...

What Topic Is Next → Next Topic 1.2: Kubernetes Architecture — `kube-apiserver`, Raft `etcd`, Scheduler, Controller Manager & `kubelet`

1.2 Kubernetes Architecture — `kube-apiserver`, Raft `etcd`, Scheduler, Controller Manager & `kubelet`

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Kube-apiserver sole etcd gateway & Etcd Raft majority (N/2 + 1) quorum • Core Syntax, CLI & Production APIs: kube-apiserver, etcd (port 2379/2380), kube-scheduler, kube-controller-manager, kubelet (102... • Deep-Dive Mechanics: AuthN -> AuthZ -> Admission chain & Scheduler Filter & Score phases • Advanced Patterns: Kubelet PodSync & CRI containerd & Kube-proxy IPVS vs iptables mode • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Hardened `KubeletConfiguration` Spec Enforcing `systemd` Cgroups & Pod Caps • Idiomatic KUBE Implementation: declarative YAML/Helm specs, Readiness/Startup probes, HPA & PDBs • Kubernetes Control Plane (apiserver, Raft etcd, Scheduler) & kubelet/CNI Execution & Memory Lifecycle for Kubernetes Architecture

Purpose: To understand and apply Kubernetes Architecture — `kube-apiserver`, Raft `etcd`, Scheduler, Controller Manager & `kubelet` in real Kubernetes systems — specifically: End-...

What Topic Is Next → Module 1 Capstone (HA 3-Control-Plane Production Cluster Topology & etcd Quorum Audit Spec) → Next Topic 2.1: `kubectl` Mastery — Declarative `apply`, Server-Side Dry-Run, JSONPath & Ephemeral `kubectl debug`

Module 1 Hands-On Capstone Project (100 Marks): HA 3-Control-Plane Production Cluster Topology & etcd Quorum Audit Spec — Design a multi-AZ High-Availability Kubernetes Control Plane topology with stacked/external 3-node `etcd` Raft quorum, API server health probes, and CRI `containerd` worker node configuratio...

MODULE 2 OF 16 • STAGE 1 • 12 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 2: `kubectl` CKA Mastery, Multi-Container Pods, Probes & QoS Classes

Wield `kubectl` with surgical speed using JSONPath, server-side dry-runs, and `kubectl debug`, and engineer resilient Pods with Init/Sidecar containers, Startup/Readiness/Liveness probes, and QoS guarantees.

Objectives: ✓ `kubectl` Mastery — Declarative `apply`, Server-Side Dry-Run, JSONPath & Ephemeral `kubectl debug` — kubeconfig contexts (`clusters`, `users`, `contexts`), why `kubectl apply` three-way merge / Server-Side Apply beats imperative `create`/`replace`, validating manifests via `--dry-run=server`, extracting fields with `-o jsonpath`, and debugging distroless Pods via `kubectl debug`. • ✓ Kubernetes Pods — Shared Network Namespace, Init/Sidecar Lifecycle, Probes & QoS Classes — Why the Pod is the atomic unit of scheduling: the `pause` infra container holding the shared Pod IP/`localhost` port space, sequential `initContainers`, `startupProbe` vs `readinessProbe` vs `livenessProbe`, graceful `SIGTERM` (`preStop` + `terminationGracePeriodSeconds`), and OOMKiller QoS tiers. • ✓ Build, verify, and ship Module Capstone: Zero-Root Hardened API Pod with Init Schema Check & Readiness Gate.

2.1 `kubectl` Mastery — Declarative `apply`, Server-Side Dry-Run, JSONPath & Ephemeral `kubectl debug`

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Kubeconfig cluster + user + context & Kubectl apply vs imperative create • Core Syntax, CLI & Production APIs: kubectl apply --server-side, kubectl diff -f, kubectl get pods -o jsonpath, kubectl logs -p,... • Deep-Dive Mechanics: --dry-run=client vs --dry-run=server & Kubectl logs --previous (-p) forensics • Advanced Patterns: -o jsonpath & custom-columns queries & Kubectl debug ephemeral containers • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Multi-Context `kubeconfig` Manifest Scoped to Production Namespace • Idiomatic KUBE Implementation: declarative YAML/Helm specs, Readiness/Startup probes, HPA & PDBs • Kubernetes Control Plane (apiserver, Raft etcd, Scheduler) & kubelet/CNI Execution & Memory Lifecycle for `kubectl` Mastery

Purpose: To understand and apply `kubectl` Mastery — Declarative `apply`, Server-Side Dry-Run, JSONPath & Ephemeral `kubectl debug` in real Kubernetes systems — specifically: kube...

What Topic Is Next → Next Topic 2.2: Kubernetes Pods — Shared Network Namespace, Init/Sidecar Lifecycle, Probes & QoS Classes

2.2 Kubernetes Pods — Shared Network Namespace, Init/Sidecar Lifecycle, Probes & QoS Classes

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Pause container shared netns/localhost & Sequential initContainers execution • Core Syntax, CLI & Production APIs: v1/Pod, initContainers, readinessProbe, livenessProbe, startupProbe, resources.requests/limi... • Deep-Dive Mechanics: Readiness (traffic) vs liveness (restart) & CPU throttling vs Memory OOMKilled 137 • Advanced Patterns: Guaranteed / Burstable / BestEffort QoS & PreStop hook + SIGTERM grace period • Production Failure Mode & Pitfall Guard •

Hands-On Live Coding Lab: Production-Hardened Non-Root Pod (`Guaranteed` QoS + Probes) • Idiomatic KUBE Implementation: declarative YAML/Helm specs, Readiness/Startup probes, HPA & PDBs • Kubernetes Control Plane (apiserver, Raft etcd, Scheduler) & kubelet/CNI Execution & Memory Lifecycle for Kubernetes Pods

Purpose: To understand and apply Kubernetes Pods — Shared Network Namespace, Init/Sidecar Lifecycle, Probes & QoS Classes in real Kubernetes systems — specifically: Why the Pod is...

What Topic Is Next → Module 2 Capstone (Zero-Root Hardened API Pod with Init Schema Check & Readiness Gate) → Next Topic 3.1: Labels, Annotations & Selectors — Loose Coupling, Set-Based `matchExpressions` & Node Selection

Module 2 Hands-On Capstone Project (100 Marks): Zero-Root Hardened API Pod with Init Schema Check & Readiness Gate — Create a production Pod specification running as non-root (`runAsNonRoot: true`, read-only root filesystem), gated by an Init container dependency check and separate Startup, Readiness, and...

MODULE 3 OF 16 • STAGE 1 • 12 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 3: Labels, Set-Based Selectors, Multi-Tenant Namespaces & Resource Governance

Wire loosely coupled Kubernetes topologies with recommended `app.kubernetes.io/*` labels and selectors, and isolate multi-team clusters using Namespaces, `ResourceQuota`, and `LimitRange`.

Objectives: ✓ Labels, Annotations & Selectors — Loose Coupling, Set-Based `matchExpressions` & Node Selection — How Kubernetes controllers and Services dynamically discover Pods solely via key-value `labels` (never hardcoded IPs!), standard `app.kubernetes.io/*` taxonomy, equality (`=`, `!=`) vs set-based (`in`, `notin`, `exists`) selectors, and why `annotations` hold non-queryable metadata. • ✓ Kubernetes Namespaces — Scope Boundaries, CoreDNS FQDN, `ResourceQuota` & `LimitRange` — Virtual sub-clusters inside a physical cluster: namespaced (`Pod`, `Service`, `Secret`) vs cluster-scoped (`Node`, `PersistentVolume`, `StorageClass`, `ClusterRole`) objects, cross-namespace CoreDNS resolution (`<svc>.<ns>.svc.cluster.local`), and quota enforcement. • ✓ Build, verify, and ship Module Capstone: Multi-Team Tenant Namespace Isolation Blueprint (`ResourceQuota` + `LimitRange`).

3.1 Labels, Annotations & Selectors — Loose Coupling, Set-Based `matchExpressions` & Node Selection [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Labels (indexed<=63c) vs annotations & App.kubernetes.io/* standard labels • Core Syntax, CLI & Production APIs: metadata.labels, metadata.annotations, selector.matchLabels, selector.matchExpressions, kube... • Deep-Dive Mechanics: Equality (=, !=) label selectors & Set-based (In, NotIn, Exists) selectors • Advanced Patterns: Immutable Deployment spec.selector & NodeSelector & nodeAffinity scheduling • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Network/Workload Set-Based Label Selector (`matchLabels` + `matchExpressions`) • Idiomatic KUBE Implementation: declarative YAML/Helm specs, Readiness/Startup probes, HPA & PDBs • Kubernetes Control Plane (apiserver, Raft etcd, Scheduler) & kubelet/CNI Execution & Memory Lifecycle for Labels

Purpose: To understand and apply Labels, Annotations & Selectors — Loose Coupling, Set-Based `matchExpressions` & Node Selection in real Kubernetes systems — specifically: How Kub...

What Topic Is Next → Next Topic 3.2: Kubernetes Namespaces — Scope Boundaries, CoreDNS FQDN, `ResourceQuota` & `LimitRange`

3.2 Kubernetes Namespaces — Scope Boundaries, CoreDNS FQDN, `ResourceQuota` & `LimitRange` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Namespaced vs cluster-scoped kinds & Svc.ns.svc.cluster.local CoreDNS FQDN • Core Syntax, CLI & Production APIs: v1/Namespace, v1/ResourceQuota, v1/LimitRange, kubectl api-resources --namespaced=true • Deep-Dive Mechanics: ResourceQuota aggregate tenant caps & LimitRange default requests/limits • Advanced Patterns: Pod Security Admission namespace labels & Finalizers stuck Terminating diagnosis • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Tenant `ResourceQuota` Enforcing Aggregate CPU, Memory & Pod Ceilings • Idiomatic KUBE Implementation: declarative YAML/Helm specs, Readiness/Startup probes, HPA & PDBs • Kubernetes Control Plane (apiserver, Raft etcd, Scheduler) & kubelet/CNI Execution & Memory Lifecycle for Kubernetes Namespaces

Purpose: To understand and apply Kubernetes Namespaces — Scope Boundaries, CoreDNS FQDN, `ResourceQuota` & `LimitRange` in real Kubernetes systems — specifically: Virtual sub-clus...

What Topic Is Next → Module 3 Capstone (Multi-Team Tenant Namespace Isolation Blueprint (`ResourceQuota` + `LimitRange`)) → Next Topic 4.1: Kubernetes ReplicaSets — Desired Replica Reconciliation, `ownerReferences` & Pod Isolation Debugging

Module 3 Hands-On Capstone Project (100 Marks): Multi-Team Tenant Namespace Isolation Blueprint (`ResourceQuota` + `LimitRange`) — Provision a governed `checkout-prod` tenant namespace that enforces strict aggregate CPU/Memory/PVC ceilings while injecting sane default container requests/limits automatically.

MODULE 4 OF 16 • STAGE 1 • 12 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 4: Self-Healing ReplicaSets, Zero-Downtime Deployments & Rollout Strategies

Understand `ownerReferences` reconciliation in `ReplicaSet`s and ship zero-downtime stateless microservices using `Deployment` `RollingUpdate` surge/unavailability budgets and instant rollbacks.

Objectives: ✓ Kubernetes ReplicaSets — Desired Replica Reconciliation, `ownerReferences` & Pod Isolation Debugging — How the `ReplicaSet` controller continuously compares matching active Pods against `spec.replicas`, `metadata.ownerReferences` garbage collection (`Cascade=background/foreground/orphan`), and isolating a misbehaving Pod live in production by removing its selector label. • ✓ Kubernetes Deployments — Zero-Downtime `RollingUpdate`, `maxSurge`/`maxUnavailable` & `rollout undo` — How `Deployment` orchestrates old and new `ReplicaSet`s during `RollingUpdate` vs `Recreate`, `maxSurge` & `maxUnavailable` math, `minReadySeconds`, `progressDeadlineSeconds`, `revisionHistoryLimit`, and instant rollback when a bad build fails readiness checks. • ✓ Build, verify, and ship Module Capstone: Zero-Dropped-Request Stateless Microservice Deployment Spec.

4.1 Kubernetes ReplicaSets — Desired Replica Reconciliation, `ownerReferences` & Pod Isolation Debugging [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Desired vs observed replica loop & OwnerReferences controller binding • Core Syntax, CLI & Production APIs: apps/v1 ReplicaSet, spec.replicas, spec.selector, metadata.ownerReferences • Deep-Dive Mechanics: Quarantine Pod via label removal & Cascading vs orphan (--cascade=orphan) • Advanced Patterns:

Deployment -> ReplicaSet -> Pod hierarchy & Pod-template-hash label uniqueness • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Declarative `apps/v1` ReplicaSet Spec with Pinned Digest/Version • Idiomatic KUBE Implementation: declarative YAML/Helm specs, Readiness/Startup probes, HPA & PDBs • Kubernetes Control Plane (apiserver, Raft etcd, Scheduler) & kubelet/CNI Execution & Memory Lifecycle for Kubernetes ReplicaSets

Purpose: To understand and apply Kubernetes ReplicaSets — Desired Replica Reconciliation, `ownerReferences` & Pod Isolation Debugging in real Kubernetes systems — specifically: Ho...

What Topic Is Next → Next Topic 4.2: Kubernetes Deployments — Zero-Downtime `RollingUpdate`, `maxSurge`/`maxUnavailable` & `rollout undo`

4.2 Kubernetes Deployments — Zero-Downtime `RollingUpdate`, `maxSurge`/`maxUnavailable` & `rollout undo`

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: RollingUpdate vs Recreate strategy & MaxUnavailable: 0 + maxSurge: 1 • Core Syntax, CLI & Production APIs: apps/v1 Deployment, strategy.rollingUpdate, kubectl rollout status / history / undo / restar... • Deep-Dive Mechanics: Immutable image tags (never :latest) & RevisionHistoryLimit & rollout undo • Advanced Patterns: ProgressDeadlineSeconds stuck alert & TopologySpreadConstraints AZ balance • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Zero-Downtime `Deployment` (`maxUnavailable: 0` + Pinned Image) • Idiomatic KUBE Implementation: declarative YAML/Helm specs, Readiness/Startup probes, HPA & PDBs • Kubernetes Control Plane (apiserver, Raft etcd, Scheduler) & kubelet/CNI Execution & Memory Lifecycle for Kubernetes Deployments

Purpose: To understand and apply Kubernetes Deployments — Zero-Downtime `RollingUpdate`, `maxSurge`/`maxUnavailable` & `rollout undo` in real Kubernetes systems — specifically: Ho...

What Topic Is Next → Module 4 Capstone (Zero-Dropped-Request Stateless Microservice Deployment Spec) → Next Topic 5.1: Kubernetes Services — `ClusterIP`, `NodePort`, `LoadBalancer`, Headless DNS & `EndpointSlice`s

Module 4 Hands-On Capstone Project (100 Marks): Zero-Dropped-Request Stateless Microservice Deployment Spec — Author a 4-replica stateless `Deployment` manifest engineered to survive both rolling image upgrades and voluntary node drains with **zero** HTTP `502/504` connection resets.

MODULE 5 OF 16 • STAGE 2 • 12 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 5: Service Discovery, EndpointSlices, Ingress/Gateway & Zero-Trust NetworkPolicies

Route east-west and north-south traffic cleanly via `ClusterIP`, Headless Services, `EndpointSlice`s, and Gateway/Ingress controllers, and enforce microsegmentation with CNI `NetworkPolicy`.

Objectives: ✓ Kubernetes Services — `ClusterIP`, `NodePort`, `LoadBalancer`, Headless DNS & `EndpointSlice`s — Why ephemeral Pod IPs necessitate stable virtual Service IPs (`ClusterIP`), mapping `port` -> named `targetPort`, `EndpointSlice` scalability (100 endpoints per slice vs monolithic `Endpoints`), Headless `clusterIP: None` returning direct Pod A/AAAA records, and Ingress / Gateway API L7 routing. • ✓ Kubernetes Network Policies — CNI Enforcement, Default-Deny Posture & DNS Egress Rules — How Kubernetes defaults to a flat allow-all network, why `NetworkPolicy` requires a policy-capable CNI (Calico, Cilium, Weave — NOT basic `flannel` alone!), additive union semantics of multiple policies, `namespaceSelector` + `podSelector` AND vs OR YAML list traps, and CoreDNS egress rules. • ✓ Build, verify, and ship Module Capstone: Zero-Trust 3-Tier Microsegmentation (`Service` + Default-Deny `NetworkPolicy`).

5.1 Kubernetes Services — `ClusterIP`, `NodePort`, `LoadBalancer`, Headless DNS & `EndpointSlice`s

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: ClusterIP virtual IP + CoreDNS & Port vs container targetPort • Core Syntax, CLI & Production APIs: v1/Service (ClusterIP, NodePort, LoadBalancer, clusterIP: None), discovery.k8s.io/v1 Endpoin... • Deep-Dive Mechanics: EndpointSlice 100-endpoint sharding & Headless Service (clusterIP: None) • Advanced Patterns: ExternalTrafficPolicy: Local client IP & L4 Service vs L7 Gateway API / Ingress • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Internal `ClusterIP` Service Mapping Port `443` to Named `https-api` • Idiomatic KUBE Implementation: declarative YAML/Helm specs, Readiness/Startup probes, HPA & PDBs • Kubernetes Control Plane (apiserver, Raft etcd, Scheduler) & kubelet/CNI Execution & Memory Lifecycle for Kubernetes Services

Purpose: To understand and apply Kubernetes Services — `ClusterIP`, `NodePort`, `LoadBalancer`, Headless DNS & `EndpointSlice`s in real Kubernetes systems — specifically: Why ephe...

What Topic Is Next → Next Topic 5.2: Kubernetes Network Policies — CNI Enforcement, Default-Deny Posture & DNS Egress Rules

5.2 Kubernetes Network Policies — CNI Enforcement, Default-Deny Posture & DNS Egress Rules

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: CNI enforcement requirement (Cilium/Calico) & Default-deny podSelector: {} baseline • Core Syntax, CLI & Production APIs: networking.k8s.io/v1 NetworkPolicy, podSelector, policyTypes: [Ingress, Egress], namespaceSe... • Deep-Dive Mechanics: Additive union (no explicit deny rules) & AND vs OR YAML list-item hyphen trap • Advanced Patterns: UDP+TCP 53 CoreDNS egress rule & Kubernetes.io/metadata.name ns label • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Zero-Trust `NetworkPolicy` Restricting Ingress + Allowing DNS Egress • Idiomatic KUBE Implementation: declarative YAML/Helm specs, Readiness/Startup probes, HPA & PDBs • Kubernetes Control Plane (apiserver, Raft etcd, Scheduler) & kubelet/CNI Execution & Memory Lifecycle for Kubernetes Network Policies

Purpose: To understand and apply Kubernetes Network Policies — CNI Enforcement, Default-Deny Posture & DNS Egress Rules in real Kubernetes systems — specifically: How Kubernetes d...

What Topic Is Next → Module 5 Capstone (Zero-Trust 3-Tier Microsegmentation (`Service` + Default-Deny `NetworkPolicy`)) → Next Topic 6.1: Kubernetes Volumes — Ephemeral `emptyDir`, `tmpfs` Memory, Projected Volumes & `subPath` Caveats

Module 5 Hands-On Capstone Project (100 Marks): Zero-Trust 3-Tier Microsegmentation (`Service` + Default-Deny `NetworkPolicy`) — Secure a `payments-prod` namespace with default-deny ingress/egress posture, allowing only `api-gateway` Pods to call `payment-svc:8443` and allowing `payment-svc` egress strictly to `postgr...

MODULE 6 OF 16 • STAGE 2 • 12 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 6: Ephemeral Volumes, CSI StorageClasses & Dynamic PersistentVolumeClaims

Manage container filesystems safely from ephemeral `emptyDir` scratch spaces and `ConfigMap` projections to cloud CSI dynamic provisioning (`StorageClass`, `PersistentVolume`, `PersistentVolumeClaim`).

Objectives: ✓ Kubernetes Volumes — Ephemeral `emptyDir`, `tmpfs` Memory, Projected Volumes & `subPath` Caveats — Why writable container overlay layers vanish on container restart whereas Pod `emptyDir` survives container crashes for the lifetime of the Pod, RAM-backed `emptyDir.medium: Memory`, `projected` volumes combining `ConfigMap` + `Secret` + `serviceAccountToken`, and why `hostPath` is banned in restricted security policies. • ✓ Persistent Storage & PVCs — CSI `StorageClass`, `AccessModes`, `Retain` vs `Delete` & `WaitForFirstConsumer` — Decoupling developer storage requests (`PersistentVolumeClaim`) from cluster infrastructure (`PersistentVolume` + CSI `StorageClass`), block `ReadWriteOnce` (`RWO`) vs shared NFS/EFS `ReadWriteMany` (`RWX`), `reclaimPolicy: Retain` vs `Delete`, and `WaitForFirstConsumer` multi-AZ scheduling. • ✓ Build, verify, and ship Module Capstone: AZ-Aware Dynamic CSI Storage Class & Expandable PVC Architecture.

6.1 Kubernetes Volumes — Ephemeral `emptyDir`, `tmpfs` Memory, Projected Volumes & `subPath` Caveats [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Container restart vs Pod emptyDir life & EmptyDir.medium: Memory + sizeLimit • Core Syntax, CLI & Production APIs: spec.volumes[].emptyDir (sizeLimit, medium: Memory), projected, configMap, containers[].volu... • Deep-Dive Mechanics: Projected volume token/config/secret & SubPath disables live ConfigMap reload • Advanced Patterns: ReadOnlyRootFilesystem + /tmp emptyDir & HostPath node-escape security hazard • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Hardened Read-Only Root Container Using Bounded `emptyDir` `/tmp` • Idiomatic KUBE Implementation: declarative YAML/Helm specs, Readiness/Startup probes, HPA & PDBs • Kubernetes Control Plane (apiserver, Raft etcd, Scheduler) & kubelet/CNI Execution & Memory Lifecycle for Kubernetes Volumes

Purpose: To understand and apply Kubernetes Volumes — Ephemeral `emptyDir`, `tmpfs` Memory, Projected Volumes & `subPath` Caveats in real Kubernetes systems — specifically: Why wr...

What Topic Is Next → Next Topic 6.2: Persistent Storage & PVCs — CSI `StorageClass`, `AccessModes`, `Retain` vs `Delete` & `WaitForFirstConsumer`

6.2 Persistent Storage & PVCs — CSI `StorageClass`, `AccessModes`, `Retain` vs `Delete` & `WaitForFirstConsumer` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: CSI StorageClass dynamic provisioning & RWO (Node) vs RWX (Multi-Node) vs RWOP • Core Syntax, CLI & Production APIs: storage.k8s.io/v1 StorageClass, v1/PersistentVolume, v1/PersistentVolumeClaim, VolumeSnapsho... • Deep-Dive Mechanics: ReclaimPolicy: Retain vs Delete & VolumeBindingMode: WaitForFirstConsumer • Advanced Patterns: AllowVolumeExpansion online resize & Kubernetes.io/pvc-protection finalizer • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: AZ-Safe CSI `StorageClass` + `PersistentVolumeClaim` Spec • Idiomatic KUBE Implementation: declarative YAML/Helm specs, Readiness/Startup probes, HPA & PDBs • Kubernetes Control Plane (apiserver, Raft etcd, Scheduler) & kubelet/CNI Execution & Memory Lifecycle for Persistent Storage

Purpose: To understand and apply Persistent Storage & PVCs — CSI `StorageClass`, `AccessModes`, `Retain` vs `Delete` & `WaitForFirstConsumer` in real Kubernetes systems — specific...

What Topic Is Next → Module 6 Capstone (AZ-Aware Dynamic CSI Storage Class & Expandable PVC Architecture) → Next Topic 7.1: Stateful Applications & `StatefulSet` — Sticky Ordinals, Headless Peer DNS & `volumeClaimTemplates`

Module 6 Hands-On Capstone Project (100 Marks): AZ-Aware Dynamic CSI Storage Class & Expandable PVC Architecture — Configure a production gp3/SSD `StorageClass` with `WaitForFirstConsumer` topology binding, online volume expansion (`allowVolumeExpansion: true`), and `Retain` reclaim protection for statef...

MODULE 7 OF 16 • STAGE 2 • 12 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 7: StatefulSets for Distributed Databases & KMS-Hardened Secrets Management

Run clustered stateful systems (Kafka, Redis, PostgreSQL) with sticky network/storage identities using `StatefulSet` + Headless Services, and secure sensitive credentials with KMS-encrypted `Secret`s and External Secrets.

Objectives: ✓ Stateful Applications & `StatefulSet` — Sticky Ordinals, Headless Peer DNS & `volumeClaimTemplates` — Why clustered databases/brokers cannot share a single `Deployment` PVC, how `StatefulSet` guarantees stable ordinal Pod names (`kafka-0`, `kafka-1`), predictable peer DNS (`kafka-0.kafka-hl.ns.svc.cluster.local`), dedicated per-ordinal PVCs via `volumeClaimTemplates`, and `OrderedReady` vs `Parallel` lifecycle. • ✓ Kubernetes Secrets — Base64 != Encryption, `etcd` KMS Encryption at Rest, `immutable: true` & CSI Vault Sync — Why standard `Secret` `data` is merely Base64-encoded (NOT encrypted by default in `etcd`!), enabling `EncryptionConfiguration` with cloud KMS v2 providers, `Opaque` vs `kubernetes.io/tls` vs `dockerconfigjson` types, setting `immutable: true` to cut kubelet watch load, and External Secrets Operator. • ✓ Build, verify, and ship Module Capstone: 3-Node Clustered StatefulSet with Sticky `volumeClaimTemplates` & Immutable TLS Secret.

7.1 Stateful Applications & `StatefulSet` — Sticky Ordinals, Headless Peer DNS & `volumeClaimTemplates` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Stable ordinal identity (pod-0, pod-1) & Governing Headless Service peer DNS • Core Syntax, CLI & Production APIs: apps/v1 StatefulSet, spec.serviceName, spec.volumeClaimTemplates, podManagementPolicy: Order... • Deep-Dive Mechanics: VolumeClaimTemplates per-replica PVC & PVCs retained on scale-down/delete • Advanced Patterns: OrderedReady vs Parallel management & No unsafe --force delete split-brain • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Production 3-Node `StatefulSet` with `volumeClaimTemplates` • Idiomatic KUBE Implementation: declarative YAML/Helm specs, Readiness/Startup probes, HPA & PDBs • Kubernetes Control Plane (apiserver, Raft etcd, Scheduler) & kubelet/CNI Execution & Memory Lifecycle for Stateful Applications

Purpose: To understand and apply Stateful Applications & `StatefulSet` — Sticky Ordinals, Headless Peer DNS & `volumeClaimTemplates` in real Kubernetes systems — specifically: Why...

What Topic Is Next → Next Topic 7.2: Kubernetes Secrets — Base64 != Encryption, `etcd` KMS Encryption at Rest, `immutable: true` & CSI Vault Sync

7.2 Kubernetes Secrets — Base64 != Encryption, `etcd` KMS Encryption at Rest, `immutable: true` & CSI Vault Sync [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Base64 encoding != encryption & Etcd KMS v2 EncryptionConfiguration • Core Syntax, CLI & Production APIs: v1/Secret (stringData, data, type: kubernetes.io/tls, immutable: true), apiserver.config.k8s... • Deep-Dive Mechanics: Tmpfs RAM-backed Secret volume mounts & Immutable: true watch optimization • Advanced Patterns: RBAC list/watch exposes Secret data & External Secrets Operator / SealedSecrets • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Immutable TLS / Application `Secret` Using `stringData` • Idiomatic KUBE Implementation: declarative YAML/Helm specs, Readiness/Startup probes, HPA & PDBs • Kubernetes Control Plane (apiserver, Raft etcd, Scheduler) & kubelet/CNI Execution & Memory Lifecycle for Kubernetes Secrets

Purpose: To understand and apply Kubernetes Secrets — Base64 != Encryption, `etcd` KMS Encryption at Rest, `immutable: true` & CSI Vault Sync in real Kubernetes systems — specific...

What Topic Is Next → Module 7 Capstone (3-Node Clustered StatefulSet with Sticky `volumeClaimTemplates` & Immutable TLS Secret) → Next Topic 8.1: Kubernetes Jobs & CronJobs — `completions`, `parallelism`, `concurrencyPolicy` & TTL Cleanup

Module 7 Hands-On Capstone Project (100 Marks): 3-Node Clustered StatefulSet with Sticky `volumeClaimTemplates` & Immutable TLS Secret — Deploy a 3-replica clustered data store `StatefulSet` with deterministic DNS peers (`store-0.store-headless`), dedicated per-ordinal SSD PVCs, and an immutable TLS Secret mounted read-only.

MODULE 8 OF 16 • STAGE 2 • 12 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 8: Batch Jobs, Scheduled CronJobs & Least-Privilege RBAC Security

Execute reliable parallel batch & scheduled workloads with `Job`s and `CronJob`s (`concurrencyPolicy`, `ttlSecondsAfterFinished`), and lock down human and workload identities using RBAC (`Role`, `ClusterRole`, `ServiceAccount`).

Objectives: ✓ Kubernetes Jobs & CronJobs — `completions`, `parallelism`, `concurrencyPolicy` & TTL Cleanup — Running finite run-to-completion tasks (`restartPolicy: OnFailure | Never`), work-queue vs `completionMode: Indexed` parallel Jobs, bounding retries (`backoffLimit`, `activeDeadlineSeconds`), auto-pruning finished Pods (`ttlSecondsAfterFinished`), and `CronJob` `Allow` vs `Forbid` vs `Replace` overlap rules. • ✓ Kubernetes RBAC — `Role` vs `ClusterRole`, `ServiceAccount` Tokens, Privilege Escalation & `auth can-i` — Additive allow-only authorization: Subjects (`User`, `Group`, `ServiceAccount`), Verbs (`get`, `list`, `watch`, `create`, `patch`, `delete`), reusing a `ClusterRole` inside a single namespace via a `RoleBinding`, time-bound projected `TokenRequest` JWTs, and verifying access with `kubectl auth can-i --as`. • ✓ Build, verify, and ship Module Capstone: Idempotent Nightly Ledger Settlement `CronJob` + Zero-Wildcard `RBAC` Suite.

8.1 Kubernetes Jobs & CronJobs — `completions`, `parallelism`, `concurrencyPolicy` & TTL Cleanup

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: RestartPolicy: OnFailure | Never rule & Completions vs parallelism workers • Core Syntax, CLI & Production APIs: batch/v1 Job, batch/v1 CronJob, completions, parallelism, backoffLimit, concurrencyPolicy, t... • Deep-Dive Mechanics: CompletionMode: Indexed ($JOB_COMPLETION_INDEX) & BackoffLimit & activeDeadlineSeconds • Advanced Patterns: TtlSecondsAfterFinished GC cleanup & CronJob concurrencyPolicy: Forbid • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Non-Overlapping Timezone-Pinned `batch/v1` `CronJob` Spec • Idiomatic KUBE Implementation: declarative YAML/Helm specs, Readiness/Startup probes, HPA & PDBs • Kubernetes Control Plane (apiserver, Raft etcd, Scheduler) & kubelet/CNI Execution & Memory Lifecycle for Kubernetes Jobs

Purpose: To understand and apply Kubernetes Jobs & CronJobs — `completions`, `parallelism`, `concurrencyPolicy` & TTL Cleanup in real Kubernetes systems — specifically: Running fi...

What Topic Is Next → Next Topic 8.2: Kubernetes RBAC — `Role` vs `ClusterRole`, `ServiceAccount` Tokens, Privilege Escalation & `auth can-i`

8.2 Kubernetes RBAC — `Role` vs `ClusterRole`, `ServiceAccount` Tokens, Privilege Escalation & `auth can-i`

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Additive allow-only RBAC model & Role (namespaced) vs ClusterRole • Core Syntax, CLI & Production APIs: rbac.authorization.k8s.io/v1 Role, ClusterRole, RoleBinding, ClusterRoleBinding, v1/ServiceA... • Deep-Dive Mechanics: RoleBinding + ClusterRole reuse pattern & AutomountServiceAccountToken: false • Advanced Patterns: Pods/exec & secrets escalation risks & Kubectl auth can-i --as verification • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Least-Privilege `Role` + `RoleBinding` Scoped to ConfigMaps Only • Idiomatic KUBE Implementation: declarative YAML/Helm specs, Readiness/Startup probes, HPA & PDBs • Kubernetes Control Plane (apiserver, Raft etcd, Scheduler) & kubelet/CNI Execution & Memory Lifecycle for Kubernetes RBAC

Purpose: To understand and apply Kubernetes RBAC — `Role` vs `ClusterRole`, `ServiceAccount` Tokens, Privilege Escalation & `auth can-i` in real Kubernetes systems — specifically:...

What Topic Is Next → Module 8 Capstone (Idempotent Nightly Ledger Settlement `CronJob` + Zero-Wildcard `RBAC` Suite) → Next Topic 9.1: Kubernetes Autoscaling — `HPA v2` Stabilization Windows, VPA, KEDA & Karpenter Node Autoscaling

Module 8 Hands-On Capstone Project (100 Marks): Idempotent Nightly Ledger Settlement `CronJob` + Zero-Wildcard `RBAC` Suite — Create a timezone-explicit nightly settlement `CronJob` preventing overlapping runs (`concurrencyPolicy: Forbid`) paired with a dedicated, least-privilege `ServiceAccount` audited via `kubec...

MODULE 9 OF 16 • STAGE 3 • 12 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 9: Horizontal/Cluster Autoscaling (`HPA v2` / Karpenter) & Helm 3 Packaging

Scale Pods elastically on CPU, memory, and custom queue metrics with `autoscaling/v2` `HPA` stabilization windows + Karpenter node provisioning, and package multi-environment releases with Helm 3.

Objectives: ✓ Kubernetes Autoscaling — `HPA v2` Stabilization Windows, VPA, KEDA & Karpenter Node Autoscaling — The 3 autoscaling dimensions: Horizontal Pod Autoscaler (`HPA v2`: `desiredReplicas = ceil[currentReplicas * (currentMetric / desiredMetric)]`), Event-Driven `KEDA` scaling to/from zero on Kafka/SQS lag, Vertical Pod Autoscaler (`VPA`), and Cluster Autoscaler / Karpenter provisioning EC2/cloud nodes for `Pending` Pods. • ✓ Kubernetes Helm 3 — Charts, Go Templates (`values.yaml`), OCI Registries & `--atomic` Rollbacks — Why Helm 3 eliminated Tiller (pure client-side library obeying user RBAC!), Chart structure (`Chart.yaml`, `values.yaml`, `templates/`, `_helpers.tpl`), `sha256sum` ConfigMap checksum annotations to trigger Deployment rollouts automatically, OCI chart registries, and `helm upgrade --install --atomic`. • ✓ Build, verify, and ship Module Capstone: Anti-Flapping `HPA v2` + Atomic Multi-Environment Helm 3 Chart Package.

9.1 Kubernetes Autoscaling — `HPA v2` Stabilization Windows, VPA, KEDA & Karpenter Node Autoscaling

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: HPA ceil(current * actual / target) math & Mandatory resources.requests denominator • Core Syntax, CLI & Production APIs: autoscaling/v2 HorizontalPodAutoscaler, metrics-server, behavior.scaleUp / scaleDown.stabili... • Deep-Dive Mechanics: ScaleDown stabilizationWindowSeconds & KEDA event-driven queue/lag scaling • Advanced Patterns: VPA vs HPA CPU conflict rule & Karpenter / Cluster Autoscaler Pending Pods • Production Failure Mode & Pitfall Guard • Hands-

On Live Coding Lab: Anti-Flapping `autoscaling/v2` `HorizontalPodAutoscaler` Spec • Idiomatic KUBE Implementation: declarative YAML/Helm specs, Readiness/Startup probes, HPA & PDBs • Kubernetes Control Plane (apiserver, Raft etcd, Scheduler) & kubelet/CNI Execution & Memory Lifecycle for Kubernetes Autoscaling

Purpose: To understand and apply Kubernetes Autoscaling — `HPA v2` Stabilization Windows, VPA, KEDA & Karpenter Node Autoscaling in real Kubernetes systems — specifically: The 3 a...

What Topic Is Next → Next Topic 9.2: Kubernetes Helm 3 — Charts, Go Templates (`values.yaml`), OCI Registries & `--atomic` Rollbacks

9.2 Kubernetes Helm 3 — Charts, Go Templates (`values.yaml`), OCI Registries & `--atomic` Rollbacks [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Tiller-less Helm 3 RBAC security & Chart.yaml + values.yaml + templates/ • Core Syntax, CLI & Production APIs: helm template, helm lint, helm diff, helm upgrade --install --atomic --wait, helm rollback <... • Deep-Dive Mechanics: ToYaml | nindent whitespace safety & Checksum/config sha256sum rollout hook • Advanced Patterns: Helm upgrade --install --atomic & --history-max release Secret cap • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Production `Chart.yaml` Spec (`apiVersion: v2` Application Chart) • Idiomatic KUBE Implementation: declarative YAML/Helm specs, Readiness/Startup probes, HPA & PDBs • Kubernetes Control Plane (apiserver, Raft etcd, Scheduler) & kubelet/CNI Execution & Memory Lifecycle for Kubernetes Helm 3

Purpose: To understand and apply Kubernetes Helm 3 — Charts, Go Templates (`values.yaml`), OCI Registries & `--atomic` Rollbacks in real Kubernetes systems — specifically: Why Hel...

What Topic Is Next → Module 9 Capstone (Anti-Flapping `HPA v2` + Atomic Multi-Environment Helm 3 Chart Package) → Next Topic 10.1: Kubernetes Monitoring — `cAdvisor` vs `kube-state-metrics`, Prometheus `ServiceMonitor` & OTel Collector

Module 9 Hands-On Capstone Project (100 Marks): Anti-Flapping `HPA v2` + Atomic Multi-Environment Helm 3 Chart Package — Package a production microservice as a reusable Helm 3 chart with `prod-values.yaml`, PodDisruptionBudget, and an `autoscaling/v2` `HPA` tuned for rapid burst scale-out and anti-flapping gra...

MODULE 10 OF 16 • STAGE 3 • 12 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 10: Cloud-Native Observability (`Prometheus` / `OpenTelemetry`) & GitOps CI/CD (`ArgoCD`)

Instrument clusters end-to-end with `kube-state-metrics`, cAdvisor, Prometheus `ServiceMonitor`s, and OpenTelemetry, and ship pull-based GitOps continuous delivery with ArgoCD / Flux.

Objectives: ✓ Kubernetes Monitoring — `cAdvisor` vs `kube-state-metrics`, Prometheus `ServiceMonitor` & OTel Collector — The cloud-native observability triad: `cAdvisor` (embedded in `kubelet` for container CPU throttling `container_cpu_cfs_throttled_periods_total` & RSS memory), `kube-state-metrics` (API object state: `CrashLoopBackOff`, `Pending` PVCs, `Deployment` replica mismatches), DaemonSet log shipping, and OpenTelemetry Collector. • ✓ Kubernetes GitOps CI/CD — Push vs Pull (`ArgoCD` / `Flux`), Image Digest Pinning & Progressive Delivery — Separating CI (test, Trivy scan, Cosign sign, push immutable image digest, open PR updating GitOps manifest repo) from pull-based CD (`ArgoCD` running inside the cluster reconciling Git -> cluster with `selfHeal` & `prune`), plus Argo Rollouts Canary / Blue-Green progressive delivery. • ✓ Build, verify, and ship Module Capstone: Prometheus `ServiceMonitor` SLO Alerting & Pull-Based ArgoCD GitOps Pipeline.

10.1 Kubernetes Monitoring — `cAdvisor` vs `kube-state-metrics`, Prometheus `ServiceMonitor` & OTel Collector [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: CAdvisor (cgroup usage) vs kube-state-metrics & Container_memory_working_set_bytes OOM • Core Syntax, CLI & Production APIs: monitoring.coreos.com/v1 ServiceMonitor & PrometheusRule, kube-state-metrics, container_memo... • Deep-Dive Mechanics: Container_cpu_cfs_throttled_periods_total & Prometheus ServiceMonitor CRD discovery • Advanced Patterns: Cardinality explosion label discipline & OpenTelemetry Collector DaemonSet/Gateway • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Prometheus Operator `ServiceMonitor` Scraping `https-metrics` • Idiomatic KUBE Implementation: declarative YAML/Helm specs, Readiness/Startup probes, HPA & PDBs • Kubernetes Control Plane (apiserver, Raft etcd, Scheduler) & kubelet/CNI Execution & Memory Lifecycle for Kubernetes Monitoring

Purpose: To understand and apply Kubernetes Monitoring — `cAdvisor` vs `kube-state-metrics`, Prometheus `ServiceMonitor` & OTel Collector in real Kubernetes systems — specifically...

What Topic Is Next → Next Topic 10.2: Kubernetes GitOps CI/CD — Push vs Pull (`ArgoCD` / `Flux`), Image Digest Pinning & Progressive Delivery

10.2 Kubernetes GitOps CI/CD — Push vs Pull (`ArgoCD` / `Flux`), Image Digest Pinning & Progressive Delivery [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: CI (build/scan/sign) vs GitOps CD split & Pull-based ArgoCD inside cluster boundary • Core Syntax, CLI & Production APIs: argoproj.io/v1alpha1 Application, syncPolicy.automated (prune: true, selfHeal: true), Kustom... • Deep-Dive Mechanics: Automated prune: true & selfHeal: true & Separate app code vs GitOps config repo • Advanced Patterns: Cosign / Kyverno image signature policy & Argo Rollouts Canary metric analysis • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Declarative `ArgoCD` `Application` Enforcing Automated `selfHeal` & `prune` • Idiomatic KUBE Implementation: declarative YAML/Helm specs, Readiness/Startup probes, HPA & PDBs • Kubernetes Control Plane (apiserver, Raft etcd, Scheduler) & kubelet/CNI Execution & Memory Lifecycle for Kubernetes GitOps CI/CD

Purpose: To understand and apply Kubernetes GitOps CI/CD — Push vs Pull (`ArgoCD` / `Flux`), Image Digest Pinning & Progressive Delivery in real Kubernetes systems — specifically:...

What Topic Is Next → Module 10 Capstone (Prometheus `ServiceMonitor` SLO Alerting & Pull-Based ArgoCD GitOps Pipeline) → Next Topic 11.1: Ingress Controllers & Modern Kubernetes Gateway API (HTTPRoute)

Module 10 Hands-On Capstone Project (100 Marks): Prometheus `ServiceMonitor` SLO Alerting & Pull-Based ArgoCD GitOps Pipeline — Define a Prometheus Operator `ServiceMonitor` scraping `/metrics` plus an ArgoCD `Application` manifest enforcing automated Git-to-cluster drift remediation (`selfHeal: true`).

MODULE 11 OF 16 • STAGE 3 • 16 LIVE + LAB HOURS • 4 TOPICS (32 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Gateway API / Ingress TLS, Zero-Trust NetworkPolicies & CoreDNS

Production-grade Kubernetes Training engineering module covering Gateway API / Ingress TLS, Zero-Trust NetworkPolicies & CoreDNS with hands-on labs and capstone build.

Objectives: ✓ Ingress Controllers & Modern Kubernetes Gateway API (HTTPRoute) — Purpose: Route multi-domain HTTPS traffic with canary header/weight splitting and cert-manager TLS. • ✓ Default-Deny NetworkPolicies (Ingress + Egress Pod Isolation) — Purpose: Ensure a compromised frontend pod can never connect directly to database pods. • ✓ CoreDNS Service Discovery, Headless Services & Topology-Aware Routing — Purpose: Resolve internal cluster DNS cleanly and keep pod traffic inside the same cloud AZ. • ✓ Debugging Packet Drops, Ephemeral Debug Containers & EndpointsSlices — Purpose: Diagnose 502/504 cluster networking mysteries rapidly with kubectl debug.

11.1 Ingress Controllers & Modern Kubernetes Gateway API (HTTPRoute) [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Modern Kubernetes Gateway API (HTTPRoute) • Integration with Default • Live Debugging, Telemetry & Technical Interview Trade • What Is Ingress Controllers & Modern Kubernetes Gateway API (HTTPRoute) & Why Industry Teams Use It • In Kubernetes Training, Ingress Controllers & Modern Kubernetes Gateway API (HTTPRoute) exists for one clear real • Even if you come from a non • Runnable Code Implementation (YAML)

Purpose: To understand and apply Ingress Controllers & Modern Kubernetes Gateway API (HTTPRoute) in real Kubernetes systems — specifically: Route multi-domain HTTPS traffic with c...

What Topic Is Next → Next Topic 11.2: Default-Deny NetworkPolicies (Ingress + Egress Pod Isolation)

11.2 Default-Deny NetworkPolicies (Ingress + Egress Pod Isolation) [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Edge • Integration with CoreDNS Service Discovery, Headless Services & Topology • Live Debugging, Telemetry & Technical Interview Trade • What Is Default • In Kubernetes Training, Default • Even if you come from a non • Runnable Code Implementation (YAML)

Purpose: To understand and apply Default-Deny NetworkPolicies (Ingress + Egress Pod Isolation) in real Kubernetes systems — specifically: Ensure a compromised frontend pod can nev...

What Topic Is Next → Next Topic 11.3: CoreDNS Service Discovery, Headless Services & Topology-Aware Routing

11.3 CoreDNS Service Discovery, Headless Services & Topology-Aware Routing [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Headless Services • Integration with Debugging Packet Drops, Ephemeral Debug Containers & EndpointsSlices • Live Debugging, Telemetry & Technical Interview Trade • What Is CoreDNS Service Discovery, Headless Services & Topology • In Kubernetes Training, CoreDNS Service Discovery, Headless Services & Topology • Even if you come from a non • Runnable Code Implementation (YAML)

Purpose: To understand and apply CoreDNS Service Discovery, Headless Services & Topology-Aware Routing in real Kubernetes systems — specifically: Resolve internal cluster DNS clea...

What Topic Is Next → Next Topic 11.4: Debugging Packet Drops, Ephemeral Debug Containers & EndpointsSlices

11.4 Debugging Packet Drops, Ephemeral Debug Containers & EndpointsSlices [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Ephemeral Debug Containers • Integration with Ingress Controllers & Modern Kubernetes Gateway API (HTTPRoute) • Live Debugging, Telemetry & Technical Interview Trade • What Is Debugging Packet Drops, Ephemeral Debug Containers & EndpointsSlices & Why Industry Teams Use It • In Kubernetes Training, Debugging Packet Drops, Ephemeral Debug Containers & EndpointsSlices exists for one clear r... • Even if you come from a non • Runnable Code Implementation (YAML)

Purpose: To understand and apply Debugging Packet Drops, Ephemeral Debug Containers & EndpointsSlices in real Kubernetes systems — specifically: Diagnose 502/504 cluster networkin...

What Topic Is Next → Module 11 Capstone (Multi-Namespace TLS-Routed & Network-Isolated K8s Mesh) → Next Topic 12.1: Roles, ClusterRoles, ServiceAccounts & Disabling Auto-Mounted Tokens

Module 11 Hands-On Capstone Project (100 Marks): Multi-Namespace TLS-Routed & Network-Isolated K8s Mesh — Design, implement, test, and document 'Multi-Namespace TLS-Routed & Network-Isolated K8s Mesh' applying every production concept from Module 11 (Gateway API / Ingress TLS, Zero-Trust Network...

MODULE 12 OF 16 • STAGE 3 • 16 LIVE + LAB HOURS • 4 TOPICS (32 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

RBAC Least Privilege, Pod Security Standards, External Secrets & Workload IAM

Production-grade Kubernetes Training engineering module covering RBAC Least Privilege, Pod Security Standards, External Secrets & Workload IAM with hands-on labs and capstone build.

Objectives: ✓ Roles, ClusterRoles, ServiceAccounts & Disabling Auto-Mounted Tokens — Purpose: Enforce strict per-team and per-microservice API server permissions. • ✓ Pod Security Admission (Restricted Profile), SecurityContext & ReadOnlyRootFS — Purpose: Block privileged containers, root UIDs, and Linux capability escalations cluster-wide. • ✓ External Secrets Operator (AWS Secrets Manager / Vault) & KMS Encryption — Purpose: Sync rotated cloud secrets into pods securely without committing base64 YAML secrets to Git. • ✓ IRSA / Workload Identity Federation for Cloud SDK Access — Purpose: Grant pods fine-grained S3/SQS access via OIDC tokens with zero static cloud keys.

12.1 Roles, ClusterRoles, ServiceAccounts & Disabling Auto-Mounted Tokens [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of ClusterRoles • Integration with Pod Security Admission (Restricted Profile), SecurityContext & ReadOnlyRootFS • Live Debugging, Telemetry & Technical Interview Trade • What Is Roles, ClusterRoles, ServiceAccounts & Disabling Auto • In Kubernetes Training, Roles, ClusterRoles, ServiceAccounts & Disabling Auto • Even if you come from a non • Runnable Code Implementation (YAML)

Purpose: To understand and apply Roles, ClusterRoles, ServiceAccounts & Disabling Auto-Mounted Tokens in real Kubernetes systems — specifically: Enforce strict per-team and per-mi...

What Topic Is Next → Next Topic 12.2: Pod Security Admission (Restricted Profile), SecurityContext & ReadOnlyRootFS

12.2 Pod Security Admission (Restricted Profile), SecurityContext & ReadOnlyRootFS [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of SecurityContext • Integration with External Secrets Operator (AWS Secrets Manager / Vault) & KMS Encryption • Live Debugging, Telemetry & Technical Interview Trade • What Is Pod Security Admission (Restricted Profile), SecurityContext & ReadOnlyRootFS & Why Industry Teams Use It • In Kubernetes Training, Pod Security Admission (Restricted Profile), SecurityContext & ReadOnlyRootFS exists for on... • Even if you come from a non • Runnable Code Implementation (YAML)

Purpose: To understand and apply Pod Security Admission (Restricted Profile), SecurityContext & ReadOnlyRootFS in real Kubernetes systems — specifically: Block privileged containe...

What Topic Is Next → Next Topic 12.3: External Secrets Operator (AWS Secrets Manager / Vault) & KMS Encryption

12.3 External Secrets Operator (AWS Secrets Manager / Vault) & KMS Encryption [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of KMS Encryption • Integration with IRSA / Workload Identity Federation for Cloud SDK Access • Live Debugging, Telemetry & Technical Interview Trade • What Is External Secrets Operator (AWS Secrets Manager / Vault) & KMS Encryption & Why Industry Teams Use It • In Kubernetes Training, External Secrets Operator (AWS Secrets Manager / Vault) & KMS Encryption exists for one cle... • Even if you come from a non • Runnable Code Implementation (YAML)

Purpose: To understand and apply External Secrets Operator (AWS Secrets Manager / Vault) & KMS Encryption in real Kubernetes systems — specifically: Sync rotated cloud secrets int...

What Topic Is Next → Next Topic 12.4: IRSA / Workload Identity Federation for Cloud SDK Access

12.4 IRSA / Workload Identity Federation for Cloud SDK Access [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Edge • Integration with Roles, ClusterRoles, ServiceAccounts & Disabling Auto • Live Debugging, Telemetry & Technical Interview Trade • What Is IRSA / Workload Identity Federation for Cloud SDK Access & Why Industry Teams Use It • In Kubernetes Training, IRSA / Workload Identity Federation for Cloud SDK Access exists for one clear real • Even if you come from a non • Runnable Code Implementation (YAML)

Purpose: To understand and apply IRSA / Workload Identity Federation for Cloud SDK Access in real Kubernetes systems — specifically: Grant pods fine-grained S3/SQS access via OIDC...

What Topic Is Next → Module 12 Capstone (Hardened CIS-Compliant Production Kubernetes Cluster) → Next Topic 13.1: Authoring Reusable Helm Charts, Values Schema Validation & Hooks

Module 12 Hands-On Capstone Project (100 Marks): Hardened CIS-Compliant Production Kubernetes Cluster — Design, implement, test, and document 'Hardened CIS-Compliant Production Kubernetes Cluster' applying every production concept from Module 12 (RBAC Least Privilege, Pod Security Standards, E...

MODULE 13 OF 16 • STAGE 4 • 16 LIVE + LAB HOURS • 4 TOPICS (32 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Helm Charts, Kustomize Overlays, ArgoCD GitOps & Progressive Rollouts

Production-grade Kubernetes Training engineering module covering Helm Charts, Kustomize Overlays, ArgoCD GitOps & Progressive Rollouts with hands-on labs and capstone build.

Objectives: ✓ Authoring Reusable Helm Charts, Values Schema Validation & Hooks — Purpose: Template standardized microservice deployments across 20+ engineering squads. • ✓ Kustomize Base + Dev/Stage/Prod Patch Overlays — Purpose: Manage environment replica/resource differences cleanly without copy-pasting YAML. • ✓ Declarative GitOps Sync, Drift Self-Healing & App-of-Apps in ArgoCD — Purpose: Make Git the single auditable source of truth with 1-click rollbacks. • ✓ Blue/Green & Automated Metric-Analysis Canary Releases (Argo Rollouts) — Purpose: Auto-abort a rollout within 30 seconds if Prometheus 5xx error rates spike.

13.1 Authoring Reusable Helm Charts, Values Schema Validation & Hooks [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Values Schema Validation • Integration with Kustomize Base + Dev/Stage/Prod Patch Overlays • Live Debugging, Telemetry & Technical Interview Trade • What Is Authoring Reusable Helm Charts, Values Schema Validation & Hooks & Why Industry Teams Use It • In Kubernetes Training, Authoring Reusable Helm Charts, Values Schema Validation & Hooks exists for one clear real • Even if you come from a non • Runnable Code Implementation (YAML)

Purpose: To understand and apply Authoring Reusable Helm Charts, Values Schema Validation & Hooks in real Kubernetes systems — specifically: Template standardized microservice dep...

What Topic Is Next → Next Topic 13.2: Kustomize Base + Dev/Stage/Prod Patch Overlays

13.2 Kustomize Base + Dev/Stage/Prod Patch Overlays [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Edge • Integration with Declarative GitOps Sync, Drift Self • Live Debugging, Telemetry & Technical Interview Trade • What Is Kustomize Base + Dev/Stage/Prod Patch Overlays & Why Industry Teams Use It • In Kubernetes Training, Kustomize Base + Dev/Stage/Prod Patch Overlays exists for one clear real • Even if you come from a non • Runnable Code Implementation (YAML)

Purpose: To understand and apply Kustomize Base + Dev/Stage/Prod Patch Overlays in real Kubernetes systems — specifically: Manage environment replica/resource differences cleanly...

What Topic Is Next → Next Topic 13.3: Declarative GitOps Sync, Drift Self-Healing & App-of-Apps in ArgoCD

13.3 Declarative GitOps Sync, Drift Self-Healing & App-of-Apps in ArgoCD [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Drift Self • Integration with Blue/Green & Automated Metric • Live Debugging, Telemetry & Technical Interview Trade • What Is Declarative GitOps Sync, Drift Self • In Kubernetes Training, Declarative GitOps Sync, Drift Self • Even if you come from a non • Runnable Code Implementation (YAML)

Purpose: To understand and apply Declarative GitOps Sync, Drift Self-Healing & App-of-Apps in ArgoCD in real Kubernetes systems — specifically: Make Git the single auditable sourc...

What Topic Is Next → Next Topic 13.4: Blue/Green & Automated Metric-Analysis Canary Releases (Argo Rollouts)

13.4 Blue/Green & Automated Metric-Analysis Canary Releases (Argo Rollouts) [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Automated Metric • Integration with Authoring Reusable Helm Charts, Values Schema Validation & Hooks • Live Debugging, Telemetry & Technical Interview Trade • What Is Blue/Green & Automated Metric • In Kubernetes Training, Blue/Green & Automated Metric • Even if you come from a non • Runnable Code Implementation (YAML)

Purpose: To understand and apply Blue/Green & Automated Metric-Analysis Canary Releases (Argo Rollouts) in real Kubernetes systems — specifically: Auto-abort a rollout within 30 s...

What Topic Is Next → Module 13 Capstone (GitOps Multi-Environment Delivery Platform with ArgoCD) → Next Topic 14.1: Requests vs Limits QoS Classes, VPA & CPU-Throttling Prevention

Module 13 Hands-On Capstone Project (100 Marks): GitOps Multi-Environment Delivery Platform with ArgoCD — Design, implement, test, and document 'GitOps Multi-Environment Delivery Platform with ArgoCD' applying every production concept from Module 13 (Helm Charts, Kustomize Overlays, ArgoCD GitOp...

MODULE 14 OF 16 • STAGE 4 • 16 LIVE + LAB HOURS • 4 TOPICS (32 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

HPA/KEDA Autoscaling, Karpenter Nodes, PDBs, Stateful CSI & Prometheus

Production-grade Kubernetes Training engineering module covering HPA/KEDA Autoscaling, Karpenter Nodes, PDBs, Stateful CSI & Prometheus with hands-on labs and capstone build.

Objectives: ✓ Requests vs Limits QoS Classes, VPA & CPU-Throttling Prevention — Purpose: Right-size pod memory/CPU scientifically and prevent OOMKilled evictions. • ✓ HPA + Event-Driven KEDA (Scale on Kafka Lag / SQS Depth) & Karpenter — Purpose: Scale pods on real queue backlog and provision right-sized spot/on-demand nodes in 40s. • ✓ PodDisruptionBudgets, TopologySpreadConstraints & Graceful preStop Hooks — Purpose: Survive AZ outages and node upgrades with zero user-visible downtime. • ✓ CSI Persistent Storage Snapshots (Velero) & kube-prometheus-stack Alerting — Purpose: Back up stateful volumes/cluster manifests and monitor golden SRE signals.

14.1 Requests vs Limits QoS Classes, VPA & CPU-Throttling Prevention [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of VPA • Integration with HPA + Event • Live Debugging, Telemetry & Technical Interview Trade • What Is Requests vs Limits QoS Classes, VPA & CPU • In Kubernetes Training, Requests vs Limits QoS Classes, VPA & CPU • Even if you come from a non • Runnable Code Implementation (YAML)

Purpose: To understand and apply Requests vs Limits QoS Classes, VPA & CPU-Throttling Prevention in real Kubernetes systems — specifically: Right-size pod memory/CPU scientificall...

What Topic Is Next → Next Topic 14.2: HPA + Event-Driven KEDA (Scale on Kafka Lag / SQS Depth) & Karpenter

14.2 HPA + Event-Driven KEDA (Scale on Kafka Lag / SQS Depth) & Karpenter [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Karpenter • Integration with PodDisruptionBudgets, TopologySpreadConstraints & Graceful preStop Hooks • Live Debugging, Telemetry & Technical Interview Trade • What Is HPA + Event • In Kubernetes Training, HPA + Event • Even if you come from a non • Runnable Code Implementation (YAML)

Purpose: To understand and apply HPA + Event-Driven KEDA (Scale on Kafka Lag / SQS Depth) & Karpenter in real Kubernetes systems — specifically: Scale pods on real queue backlog a...

What Topic Is Next → Next Topic 14.3: PodDisruptionBudgets, TopologySpreadConstraints & Graceful preStop Hooks

14.3 PodDisruptionBudgets, TopologySpreadConstraints & Graceful preStop Hooks [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of TopologySpreadConstraints • Integration with CSI Persistent Storage Snapshots (Velero) & kube • Live Debugging, Telemetry & Technical Interview Trade • What Is PodDisruptionBudgets, TopologySpreadConstraints & Graceful preStop Hooks & Why Industry Teams Use It • In Kubernetes Training, PodDisruptionBudgets, TopologySpreadConstraints & Graceful preStop Hooks exists for one cle... • Even if you come from a non • Runnable Code Implementation (YAML)

Purpose: To understand and apply PodDisruptionBudgets, TopologySpreadConstraints & Graceful preStop Hooks in real Kubernetes systems — specifically: Survive AZ outages and node up...

What Topic Is Next → Next Topic 14.4: CSI Persistent Storage Snapshots (Velero) & kube-prometheus-stack Alerting

14.4 CSI Persistent Storage Snapshots (Velero) & kube-prometheus-stack Alerting [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of kube • Integration with Requests vs Limits QoS Classes, VPA & CPU • Live Debugging, Telemetry & Technical Interview Trade • What Is CSI Persistent Storage Snapshots (Velero) & kube • In Kubernetes Training, CSI Persistent Storage Snapshots (Velero) & kube • Even if you come from a non • Runnable Code Implementation (YAML)

Purpose: To understand and apply CSI Persistent Storage Snapshots (Velero) & kube-prometheus-stack Alerting in real Kubernetes systems — specifically: Back up stateful volumes/clu...

What Topic Is Next → Module 14 Capstone (Auto-Scaling High-Availability Production Cluster Stack) → Next Topic 15.1: Pod Security Standards (Restricted), Non-Root SecurityContexts & Read-Only Root Filesystems

Module 14 Hands-On Capstone Project (100 Marks): Auto-Scaling High-Availability Production Cluster Stack — Design, implement, test, and document 'Auto-Scaling High-Availability Production Cluster Stack' applying every production concept from Module 14 (HPA/KEDA Autoscaling, Karpenter Nodes, PDBs,...

MODULE 15 OF 16 • STAGE 4 • 14 LIVE + LAB HOURS • 3 TOPICS (24 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Kubernetes Zero-Trust Security, Pod Security Standards & Autoscaling (HPA/KEDA)

Self-explanatory list-and-bullet production mastery module covering Kubernetes Zero-Trust Security, Pod Security Standards & Autoscaling (HPA/KEDA) in Kubernetes Training, paired with hands-on labs, capstone repository, Topic Mock Tests, and Spoken MockAttempt AI Viva.

Objectives: ✓ Pod Security Standards (Restricted), Non-Root SecurityContexts & Read-Only Root Filesystems — Purpose: Block container privilege escalation and enforce least-privilege RBAC ServiceAccounts. • ✓ Default-Deny NetworkPolicies, TLS Gateway API / Ingress & External Secrets Sync — Purpose: Isolate east-west pod traffic and inject rotated cloud secrets without plaintext YAML. • ✓ Requests/Limits Right-Sizing, HPA/KEDA Event Autoscaling & PodDisruptionBudgets — Purpose: Scale pods smoothly on CPU or queue depth while guaranteeing zero downtime during node drains. • ✓ Build, verify, and document 'Hardened Non-Root Namespace with NetworkPolicies, PSS Restricted & HPA/PDB' with clean Git commits and edge-case protection. • ✓ Clear 100% of Topic Timed Mock Tests and Spoken Voice + MockAttempt AI Technical Interviews.

15.1 Pod Security Standards (Restricted), Non-Root SecurityContexts & Read-Only Root Filesystems

[65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for Pod Security Standards (Restricted), Non • Performance Optimization, Complexity & Latency Impact of Pod Security Standards (Restricted), Non • Live Debugging, Stack • Runnable Code Implementation (YAML) • Hands • Idiomatic KUBE Implementation: declarative YAML/Helm specs, Readiness/Startup probes, HPA & PDBs • Kubernetes Control Plane (apiserver, Raft etcd, Scheduler) & kubelet/CNI Execution & Memory Lifecycle for Pod Security Standards (Restricted) • Production Hardening & Security: least-privilege RBAC, Pod Security Standards, NetworkPolicies & Secrets CS...

Purpose: To understand and apply Pod Security Standards (Restricted), Non-Root SecurityContexts & Read-Only Root Filesystems in real Kubernetes systems — specifically: Block conta...

What Topic Is Next → Next Topic 15.2: Default-Deny NetworkPolicies, TLS Gateway API / Ingress & External Secrets Sync

15.2 Default-Deny NetworkPolicies, TLS Gateway API / Ingress & External Secrets Sync

[65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for Default • Performance Optimization, Complexity & Latency Impact of Default • Live Debugging, Stack • Runnable Code Implementation (YAML) • Hands • Idiomatic KUBE Implementation: declarative YAML/Helm specs, Readiness/Startup probes, HPA & PDBs • Kubernetes Control Plane (apiserver, Raft etcd, Scheduler) & kubelet/CNI Execution & Memory Lifecycle for Default-Deny NetworkPolicies • Production Hardening & Security: least-privilege RBAC, Pod Security Standards, NetworkPolicies & Secrets CS...

Purpose: To understand and apply Default-Deny NetworkPolicies, TLS Gateway API / Ingress & External Secrets Sync in real Kubernetes systems — specifically: Isolate east-west pod t...

What Topic Is Next → Next Topic 15.3: Requests/Limits Right-Sizing, HPA/KEDA Event Autoscaling & PodDisruptionBudgets

15.3 Requests/Limits Right-Sizing, HPA/KEDA Event Autoscaling & PodDisruptionBudgets

[65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for Requests/Limits Right • Performance Optimization, Complexity & Latency Impact of Requests/Limits Right • Live Debugging, Stack • Runnable Code Implementation (YAML) • Hands • Idiomatic KUBE Implementation: declarative YAML/Helm specs, Readiness/Startup probes, HPA & PDBs • Kubernetes Control Plane (apiserver, Raft etcd, Scheduler) & kubelet/CNI Execution & Memory Lifecycle for Requests/Limits Right-Sizing • Production Hardening & Security: least-privilege RBAC, Pod Security Standards, NetworkPolicies & Secrets CS...

Purpose: To understand and apply Requests/Limits Right-Sizing, HPA/KEDA Event Autoscaling & PodDisruptionBudgets in real Kubernetes systems — specifically: Scale pods smoothly on...

What Topic Is Next → Module 15 Capstone (Hardened Non-Root Namespace with NetworkPolicies, PSS Restricted & HPA/PDB) → Next Topic 16.1: GitOps Sync Loops with Argo CD, Helm Values Hygiene & Canary/Blue-Green Rollouts

Module 15 Hands-On Capstone Project (100 Marks): Hardened Non-Root Namespace with NetworkPolicies, PSS Restricted & HPA/PDB — Architect, implement, test, and document 'Hardened Non-Root Namespace with NetworkPolicies, PSS Restricted & HPA/PDB' applying every bullet-wise engineering concept from Module 15 (Kubernete...

MODULE 16 OF 16 • STAGE 4 • 14 LIVE + LAB HOURS • 3 TOPICS (24 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

GitOps Continuous Delivery (Argo CD), Cluster Troubleshooting & CKA/CKAD Prep

Self-explanatory list-and-bullet production mastery module covering GitOps Continuous Delivery (Argo CD), Cluster Troubleshooting & CKA/CKAD Prep in Kubernetes Training, paired with hands-on labs, capstone repository, Topic Mock Tests, and Spoken MockAttempt AI Viva.

Objectives: ✓ GitOps Sync Loops with Argo CD, Helm Values Hygiene & Canary/Blue-Green Rollouts — Purpose: Promote audited cluster changes automatically from Git with instant 1-click rollback. • ✓ Systematic Triage of CrashLoopBackOff, ImagePullBackOff, OOMKilled & CoreDNS Failures — Purpose: Debug broken pods, PVCs, endpoints, and CNI networking rapidly using kubectl & events. • ✓ Multi-AZ Topology Spread Constraints & Production Kubernetes Interviews — Purpose: Architect highly available stateful and stateless clusters and pass hands-on K8s scenarios. • ✓ Build, verify, and document 'Declarative Helm + Argo CD Rollout Pipeline & Live Cluster Failure Runbook' with clean Git commits and edge-case protection. • ✓ Clear 100% of Topic Timed Mock Tests and Spoken Voice + MockAttempt AI Technical Interviews.

16.1 GitOps Sync Loops with Argo CD, Helm Values Hygiene & Canary/Blue-Green Rollouts

[65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for GitOps Sync Loops with Argo CD, Helm Values Hygiene & Canary/Blue • Performance Optimization, Complexity & Latency Impact of GitOps Sync Loops with Argo CD, Helm Values Hygiene & Cana... • Live Debugging, Stack • Runnable Code Implementation (YAML) • Hands • Idiomatic KUBE Implementation: declarative YAML/Helm specs, Readiness/Startup probes, HPA & PDBs • Kubernetes Control Plane (apiserver, Raft etcd,

Scheduler) & kubelet/CNI Execution & Memory Lifecycle for GitOps Sync Loops with Argo CD • Production Hardening & Security: least-privilege RBAC, Pod Security Standards, NetworkPolicies & Secrets CS...

Purpose: To understand and apply GitOps Sync Loops with Argo CD, Helm Values Hygiene & Canary/Blue-Green Rollouts in real Kubernetes systems — specifically: Promote audited cluste...

What Topic Is Next → Next Topic 16.2: Systematic Triage of CrashLoopBackOff, ImagePullBackOff, OOMKilled & CoreDNS Failures

16.2 Systematic Triage of CrashLoopBackOff, ImagePullBackOff, OOMKilled & CoreDNS Failures [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for Systematic Triage of CrashLoopBackOff, ImagePullBackOff, OOMKilled & CoreD... • Performance Optimization, Complexity & Latency Impact of Systematic Triage of CrashLoopBackOff, ImagePullBackOff, O... • Live Debugging, Stack • Runnable Code Implementation (YAML) • Hands • Idiomatic KUBE Implementation: declarative YAML/Helm specs, Readiness/Startup probes, HPA & PDBs • Kubernetes Control Plane (apiserver, Raft etcd, Scheduler) & kubelet/CNI Execution & Memory Lifecycle for Systematic Triage of CrashLoopBackOff • Production Hardening & Security: least-privilege RBAC, Pod Security Standards, NetworkPolicies & Secrets CS...

Purpose: To understand and apply Systematic Triage of CrashLoopBackOff, ImagePullBackOff, OOMKilled & CoreDNS Failures in real Kubernetes systems — specifically: Debug broken pods...

What Topic Is Next → Next Topic 16.3: Multi-AZ Topology Spread Constraints & Production Kubernetes Interviews

16.3 Multi-AZ Topology Spread Constraints & Production Kubernetes Interviews [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for Multi • Performance Optimization, Complexity & Latency Impact of Multi • Live Debugging, Stack • Runnable Code Implementation (YAML) • Hands • Idiomatic KUBE Implementation: declarative YAML/Helm specs, Readiness/Startup probes, HPA & PDBs • Kubernetes Control Plane (apiserver, Raft etcd, Scheduler) & kubelet/CNI Execution & Memory Lifecycle for Multi-AZ Topology Spread Constraints • Production Hardening & Security: least-privilege RBAC, Pod Security Standards, NetworkPolicies & Secrets CS...

Purpose: To understand and apply Multi-AZ Topology Spread Constraints & Production Kubernetes Interviews in real Kubernetes systems — specifically: Architect highly available stat...

What Topic Is Next → Module 16 Capstone → End-of-Course Live Client Project: KubeScale Zero-Downtime Microservices Orchestration & GitOps Production Cluster

Module 16 Hands-On Capstone Project (100 Marks): Declarative Helm + Argo CD Rollout Pipeline & Live Cluster Failure Runbook — Architect, implement, test, and document 'Declarative Helm + Argo CD Rollout Pipeline & Live Cluster Failure Runbook' applying every bullet-wise engineering concept from Module 16 (GitOps Co...

3. LIVE CLIENT PRODUCTION PROJECT & CAREER SCOPE

End-of-Course Live Client Production Project + Official Internship Confirmation Letter

Project: KubeScale Zero-Downtime Microservices Orchestration & GitOps Production Cluster • **Confirmed Role:** Kubernetes & Cloud-Native SRE Intern

Domain & Production Infra: Enterprise E-Commerce Cloud Platform Client (Kubernetes Cluster + NGINX Ingress TLS + HPA + StatefulSets + Helm)

- Authored production Kubernetes manifests and Helm charts for Deployments, StatefulSets, ConfigMaps, and Secrets.
- Configured NGINX Ingress TLS termination, NetworkPolicies, RBAC service accounts, and Horizontal Pod Autoscalers.
- Implemented liveness/readiness/startup probes, PodDisruptionBudgets, and persistent volume storage classes.
- Executed zero-downtime rolling upgrades and observability verification on a live production client cluster.

Target Job Roles: Kubernetes & Cloud-Native Engineer, Site Reliability Engineer (SRE), DevOps & Container Platform Engineer, Infrastructure Software Engineer

Realistic Annual CTC Scope: **Junior Cloud-Native / DevOps Engineer (0–1 Yr):** ₹6.0 LPA – ₹11.5 LPA **Kubernetes DevOps / SRE Engineer (1–3 Yrs):** ₹12.5 LPA – ₹24.0 LPA **Principal Platform Architect / Staff SRE (3–6+ Yrs):** ₹25.0 LPA – ₹48.0+ LPA