****K8s DOCUMENT REALTIME****

|  |  |  |
| --- | --- | --- |
| **Topic** | **Detailed Explanation** | **Real-time Example** |
| **Introduction to Kubernetes** | Kubernetes is an open-source container orchestration platform. It automates deploying, scaling, and managing containerized apps. | Enterprises use Kubernetes to modernize legacy systems and enable rapid CI/CD pipelines. |
| **Monolithic vs Microservices Architecture** | Monolithic apps run as a single unit, scaling is difficult. Microservices split functions into independent containers managed by K8s. | Insurance companies migrating monolith core systems into microservices running in individual pods (claims, payments). |
| **Why Use Kubernetes?** | Benefits include automated deploy/rollback, self-healing, load balancing, and declarative Infrastructure as Code (IaC). | Banking apps using blue-green deployment to minimize downtime during updates. |
| **Kubernetes Architecture - Components** | Control plane (API server, etcd, scheduler, controller manager) manages cluster state; nodes run kubelet, kube-proxy & container runtime. | DevOps engineers use kubectl and automation scripts to deploy workloads onto cluster-managed nodes. |
| **Cluster Types** | Managed clusters (EKS, AKS), self-hosted clusters (installed independently), and hybrid/multi-cloud setups. | Multi-national firms deploying AKS in Azure Europe and EKS in AWS US regions. |
| **Self-hosted Cluster with Docker** | Install Kubernetes binaries on Linux VMs, Docker runtime, networking with Calico/CNI, joining worker nodes manually. | Small development teams running local K8s clusters for testing. |
| **Communicating with Control Plane** | Interaction via kubectl, APIs, or client libraries. | Jenkins pipelines execute kubectl apply commands to deploy new app versions. |
| **Manifest Files & YAML Format** | YAML or JSON files define K8s objects such as Pods, Services, Deployments in declarative fashion. | Deployment manifests scaling a web app to 3 replicas using kubectl apply. |
| **CNI & Containerd Layer Functionality** | CNI plugins (Calico, Weave) handle pod networking; containerd runtime manages image pulling, container lifecycle. | Troubleshooting pod communication by examining CNI plugin status; switching to containerd for better compliance. |
| **Pods** | Smallest K8s unit, groups one or more containers sharing network namespace. | Running Node.js app pod with sidecar container for log shipping (Fluentd). |
| **Database Pod Definition** | Pod running DB with persistent volumes via PVC and secret credentials. | PostgreSQL pod uses PersistentVolumeClaim for storage and Kubernetes Secrets for password management. |
| **Pod Lifecycle & Probes** | Pod states: Pending → Running → Succeeded/Failed; probes ensure health (liveness), readiness (traffic routing), startup sequencing. | Readiness probe prevents requests to payment API pod before DB connection is established. |
| **Labels & Selectors** | Labels attach metadata (key-value pairs) for selection and routing; e.g., environment=prod. | Scaling only frontend pods with label environment=prod via deployment selector. |
| **Requests & Limits** | Set CPU/memory requests and limits per pod to control resource allocation and QoS. | ML service pod requests 4 CPU units; limits set to 8 CPU for burst workloads. |
| **Pod Communication** | Pods communicate via internal DNS and ClusterIP services within the cluster. | Frontend pod calling backend pod using its service name (backend-service). |
| **Controllers** | Maintain desired number of pod replicas (ReplicaSets), manage stateful apps (StatefulSets), run node-wide pods (DaemonSets), schedule batch jobs (Jobs/CronJobs). | DaemonSet running Fluentd on each node collects logs; CronJob runs nightly DB backup. |
| **EKS/AKS Cluster** | Managed Kubernetes on AWS (EKS) or Azure (AKS) with integration for cloud services (IAM, ALB, Azure AD). | Enterprise uses AKS for compliance in Europe region with integration to Azure AD for authentication. |
| **Types of Services** | ClusterIP (internal), NodePort (open on node IP), LoadBalancer (public cloud LB), Headless (DNS with no cluster IP). | LoadBalancer service exposing web app to public users, internal APIs accessed via ClusterIP. |
| **Pod-Service Communication & Probes** | Services select pods by labels; readiness probes ensure only healthy pods get traffic. | LoadBalancer directs traffic to pods ready for new e-commerce release. |
| **Deployments** | Supports rolling updates, blue-green deployments with rollback; manages declarative app versions. | Rolling update of web app container image triggered from GitLab CI pipeline. |
| **Microservices with Kubernetes** | Microservices run as separate deployments communicating via DNS and services. | E-commerce split into independent microservices (checkout, catalog, user account). |
| **Ingress/Inggress Gateway** | Manages HTTP/S routing from outside world to services with path-based routing, TLS, authentication. | NGINX Ingress configuration for secure, authorized banking API access. |
| **StatefulSets** | Manage stateful applications requiring stable identities and persistent storage. | Cassandra cluster deployment where each pod maintains data via persistent volumes. |
| **Persistent Volumes & PVCs** | PVCs request storage, PVs provide physical storage resources for stateful workloads. | Video processing pods storing outputs to network-attached storage via PVCs. |
| **DaemonSets & Logging** | Ensures one pod runs on every node, e.g. for log collection or node monitoring. | Fluentd configured via DaemonSet collects logs from all cluster nodes. |
| **Jobs & CronJobs** | Run batch jobs or scheduled tasks (e.g., DB cleanup, backups). | CronJob manifest scheduling hourly backups of critical application databases. |
| **ConfigMaps** | Store application configuration separately for flexibility without application redeploy. | Feature toggle stored in ConfigMap updated independently of app. |
| **Secrets** | Securely store sensitive data such as passwords, API keys; mounted as env variables or volumes. | Application pod mounts DB credentials from Kubernetes Secrets rather than hardcoding. |
| **Annotations** | Metadata storage supporting debugging, monitoring, tooling info. | Annotating pods with git commit SHA for traceability in CI/CD process. |
| **Health Checks** | Liveness, readiness, and startup probes to detect unhealthy containers and ensure proper startup. | Payment service pod configured with startup and liveness probes to prevent downtime and retry failures. |
| **Scheduling Pods** | Uses affinity rules (node/pod affinity), taints, tolerations to control pod placement. | GPU workload pods scheduled with node affinity to nodes labeled ‘GPU=true’. |
| **Network Policies** | Define rules to allow/deny traffic between pods for security segmentation. | DB pod only accepts connections from authorized backend pods via NetworkPolicy. |
| **Cluster Administration Tasks** | Cluster scaling, node draining, log management, rolling upgrades, troubleshooting pod/network issues. | Automatically cordon and drain nodes during maintenance, restarting pods on healthy nodes. |
| **Authentication & Authorization (RBAC)** | Define roles and permissions to restrict access to cluster resources by users or services. | ‘Dev’ team given limited permissions to development namespaces only. |
| **Helm** | Package manager for Kubernetes, simplifies deployment and management of complex applications via charts. | Deploying ELK stack with a single Helm command supporting easy upgrades and rollbacks. |
| **Monitoring with Kubernetes** | Prometheus collects cluster metrics; Grafana visualizes dashboards; alerts notify SRE teams of issues. | Monitoring CPU, memory usage, and latency to detect anomalies in production clusters. |

| **Q** | **Detailed Answer** | **Real-Time Example** |
| --- | --- | --- |
| **1. What happens if etcd is down?** | Workloads keep running since kubelet manages pods. But control plane is unavailable → no new deployments, scaling, or config changes. Recovery: restore etcd snapshot or quorum. | In prod, we restored etcd from nightly snapshots when quorum broke after node crash. Apps stayed up, but no deployments for ~15 mins. |
| **2. Deployment vs StatefulSet vs DaemonSet vs Job vs CronJob** | Deployment = stateless, scaling apps. StatefulSet = stateful apps with stable IDs/storage. DaemonSet = one pod per node. Job = run-to-completion tasks. CronJob = scheduled jobs. | Used StatefulSet for Kafka, DaemonSet for FluentD log collectors, CronJob for daily DB backup. |
| **3. ReplicaSet vs ReplicationController** | Both ensure replicas, but ReplicaSet supports set-based selectors and is newer. ReplicationController is legacy. | Migrated old RC workloads to ReplicaSet for flexible label selectors. |
| **4. How HPA works** | HPA queries metrics-server (CPU/memory/custom metrics). API Server → Controller Manager adjusts Deployment replica count. | Configured HPA to auto-scale pods when CPU > 70%. During traffic spikes, replicas doubled. |
| **5. Pod stuck in ContainerCreating** | Causes: ImagePullBackOff, PVC mount failure, CNI issue, node taints. Debug with kubectl describe pod. | In AWS EKS, pods stuck due to missing IAM role for EBS CSI driver → fixed IAM permissions. |
| **6. Pod CrashLoopBackOff** | Caused by bad probes, OOMKilled, misconfigured env vars. Debug with kubectl logs -p. | In a fintech app, readinessProbe failed (app startup 40s vs probe 10s) → pods kept restarting. Fixed probe delay. |
| **7. Node NotReady** | Reasons: kubelet crash, container runtime issues, disk/memory pressure. Fix: restart kubelet, drain node if needed. | During patching, kubelet failed due to disk full → cleared logs, restarted kubelet. |
| **8. Pod cross-node network issue** | Likely CNI misconfig, firewall rules, or DNS issues. Debug via ping, nslookup, check CNI logs. | Calico BGP misconfigured blocked cross-node pod traffic. Fixed with proper BGP peer config. |
| **9. Service types (ClusterIP/NodePort/LB/Ingress)** | ClusterIP = internal only. NodePort = exposed on all nodes. LoadBalancer = cloud external IP. Ingress = L7 routing + TLS. | In prod, Ingress + Cert-Manager used for TLS termination of APIs. |
| **10. Kube-proxy** | Runs on nodes, manages iptables/ipvs rules, routes Service traffic to pods. L4 only (not L7). | Debugged Service issue by inspecting kube-proxy logs → iptables not updated due to crash. |
| **11. DNS resolution** | CoreDNS pods handle DNS. Pods’ /etc/resolv.conf points to CoreDNS. Service names → ClusterIP. | Users couldn’t reach services due to CoreDNS crash → restarted pods + fixed ConfigMap. |
| **12. Service DNS issue** | Fix: check CoreDNS, kube-dns ConfigMap, kubelet --cluster-dns. | In on-prem K8s, CoreDNS couldn’t resolve external DNS due to firewall on UDP/53. |
| **13. emptyDir vs hostPath vs PV** | emptyDir = ephemeral storage. hostPath = node’s local FS. PV = external, persistent storage (EBS, Ceph, NFS). | Used PVs for MySQL DB pods to survive pod rescheduling. |
| **14. StatefulSet + storage** | Uses volumeClaimTemplates → unique PVCs for each pod → ensures persistent storage + stable identity. | Kafka StatefulSet with 3 brokers, each had its own PVC bound to different EBS volumes. |
| **15. PVC deletion** | Depends on reclaimPolicy: Retain (PV stays), Delete (PV removed), Recycle (deprecated). | In EKS, PV auto-deleted when PVC deleted → lost data; later switched to Retain. |
| **16. Secrets security** | Stored base64 in etcd. Must enable encryption-at-rest or external secrets (Vault, KMS). | Migrated Secrets to AWS KMS via ExternalSecrets Operator for compliance. |
| **17. Restrict pod traffic** | Use NetworkPolicies with ingress/egress rules. Without NP → all pods talk freely. | Applied NP to restrict frontend pods from accessing DB pods directly. |
| **18. ServiceAccount + RBAC** | Pods authenticate with ServiceAccounts. RBAC binds Roles to SA for permissions. | Backend SA given access only to ConfigMaps; prevented accidental Secret access. |
| **19. Namespace isolation** | Use ResourceQuotas, RBAC, and NetworkPolicies. | In multi-team cluster, applied quotas to prevent one team from consuming all CPU. |
| **20. Taints & Tolerations** | Taint repels pods. Toleration allows pod to schedule on tainted node. | Tainted GPU nodes with gpu=true:NoSchedule, only ML pods had tolerations. |
| **21. Node vs Pod Affinity** | Node Affinity = schedule pods to specific nodes. Pod Affinity = co-locate pods. Pod Anti-Affinity = spread pods apart. | Applied Anti-Affinity to ensure replicas of payment service spread across AZs. |
| **22. Pod unschedulable** | Causes: resource shortage, taints, affinity mismatch. Debug via kubectl describe pod. | Pod unschedulable due to high CPU requests > node capacity. Fixed by right-sizing requests. |
| **23. Rolling updates** | Deployment → creates new ReplicaSet, gradually replaces pods. Controlled by maxUnavailable & maxSurge. | Configured maxSurge=1, maxUnavailable=0 for zero downtime API upgrades. |
| **24. Zero downtime deployment** | Ensure readinessProbe works. Configure surge and unavailable limits. Use PreStop hook. | API had downtime because probes failed → fixed with proper grace period + PreStop sleep. |
| **25. Rollback** | kubectl rollout undo deployment. Also check history via kubectl rollout history. | Rolled back microservice after CPU leak was introduced in new image. |
| **26. Blue/Green vs Canary** | Blue/Green = two environments, switch traffic instantly. Canary = gradual rollout. | Used Canary with 10% → 50% → 100% rollout for new backend API. |
| **27. Monitoring cluster usage** | Use kubectl top (metrics-server) or Prometheus + Grafana. | Created Grafana dashboards showing CPU/memory per namespace. |
| **28. OOMKilled pod** | Pod exceeded memory limit. Fix by increasing limits, tuning app, or right-sizing requests. | Java service crashed due to heap issues; increased pod memory to 2Gi + tuned JVM flags. |
| **29. Logs collection** | kubectl logs for single pod. Cluster-wide → EFK, Loki, or FluentBit. | Centralized all logs in EFK for auditing and debugging issues. |
| **30. API server down** | Existing workloads continue. No new changes/deployments. Fix by restarting/recovering API server. | API server crashed due to out-of-memory → fixed by tuning --max-requests-inflight. |
| **31. Scheduler crash** | Existing workloads continue. New pods stay in Pending. Fix → restart scheduler. | During upgrade, scheduler down for 2 mins → pending pods later scheduled automatically. |
| **32. CNI vs CSI** | CNI = networking plugins (Calico, Flannel, Cilium). CSI = storage plugins (EBS, Ceph, NFS). | Used AWS EBS CSI driver for dynamic PVC provisioning. |
| **33. Pod Disruption Budget** | Ensures min pods available during voluntary disruptions (drain, upgrade). | Applied PDB minAvailable=2 for payment pods during node upgrades. |
| **34. Cluster-wide performance issues** | Check etcd latency, API server metrics, kubelet logs, node resource usage. | etcd degraded under load → fixed by adding dedicated SSD storage. |
| **35. Compliance/security enforcement** | Use Pod Security Admission, OPA Gatekeeper, Kyverno, admission controllers. | Blocked privileged containers via OPA Gatekeeper policies. |
| **36. Multi-cluster management** | Options: Federation (kubefed), service mesh (Istio multi-cluster), or Cluster API. | Managed 3 EKS clusters via ArgoCD + Istio multi-cluster for service mesh. |
| **37. Service load balancing issue** | Debug via kubectl get endpoints, check kube-proxy logs, NodePort reachability. | Service had no endpoints due to selector mismatch → fixed Deployment labels. |
| **38. InitContainer failing** | Debug via kubectl logs -c <init-container>. Often dependency or mount issues. | Init container failed because DB not reachable → added readiness gate. |
| **39. Namespace resource hog** | Fix via ResourceQuotas, LimitRanges, node isolation. | One team used 90% CPU → applied quota to restrict them. |
| **40. Cluster upgrade downtime** | Upgrade control plane first, then nodes. Use surge upgrades, drain/cordon, PDBs. | During EKS upgrade, used rolling node group upgrade with PDB → zero downtime. |

**COMMANDS WITH EXPLANATION :**

**1.Basic Cluster Information**

| **Command** | **Explanation** | **Real-Time Example** |
| --- | --- | --- |
| kubectl cluster-info | Shows cluster control plane & service endpoints. | During cluster troubleshooting, checked if API server and DNS were reachable. |
| kubectl get nodes | Lists all worker/master nodes. | Used to verify node status (Ready / NotReady) after patching. |
| kubectl describe node <node> | Shows node capacity, labels, taints, allocatable resources. | Debugged scheduling issue → found node tainted with NoSchedule. |
| kubectl top nodes | Shows CPU/memory usage of nodes (needs metrics-server). | Found one node over-utilized (95% CPU), drained it, and rescheduled pods. |

## ****2. Pods Management****

| **Command** | **Explanation** | **Real-Time Example** |
| --- | --- | --- |
| kubectl get pods | Lists pods in default namespace. | Checked if pods were stuck in CrashLoopBackOff. |
| kubectl get pods -n <namespace> -o wide | Lists pods with node/IP info. | Debugged networking by checking pod-to-node mapping. |
| kubectl describe pod <pod> | Detailed info (events, probes, mounts). | Found pod failing due to PVC not bound. |
| kubectl logs <pod> | Get pod logs (default container). | Checked Nginx error logs when ingress was failing. |
| kubectl logs <pod> -c <container> | Logs of specific container in multi-container pod. | Debugged sidecar logging agent. |
| kubectl exec -it <pod> -- /bin/sh | Opens interactive shell inside pod. | Ran curl from pod to test service connectivity. |
| kubectl delete pod <pod> | Deletes a pod (will be recreated if controlled by Deployment/RS). | Forced restart of app pod after config update. |
| kubectl cp <pod>:/path/file ./local/ | Copy file from pod to local. | Retrieved logs from container when cluster logging was down. |

1. **Deployments & ReplicaSets**

| **Command** | **Explanation** | **Real-Time Example** |
| --- | --- | --- |
| kubectl create deployment myapp --image=nginx | Creates deployment with nginx image. | Used to quickly test cluster after node upgrade. |
| kubectl get deployments | List deployments. | Verified number of replicas running. |
| kubectl describe deployment <name> | Shows rollout strategy, events. | Debugged rollout stuck due to readiness probe failures. |
| kubectl scale deployment myapp --replicas=5 | Scales replicas. | Scaled payment service from 2→5 pods during traffic spike. |
| kubectl rollout status deployment myapp | Check rollout status. | Waited for canary deployment pods to be ready. |
| kubectl rollout undo deployment myapp | Rollback to previous version. | Rolled back due to high error rate in new release. |

## ****4. Services & Networking****

| **Command** | **Explanation** | **Real-Time Example** |
| --- | --- | --- |
| kubectl get svc | List services. | Verified ClusterIP assigned to backend service. |
| kubectl describe svc <svc> | Shows selectors, ports, endpoints. | Found misconfigured selector → service had no endpoints. |
| kubectl get endpoints <svc> | Shows which pods are behind service. | Debugged service with no pod endpoints. |
| kubectl port-forward svc/myapp 8080:80 | Forwards local port to service. | Accessed internal DB web UI without exposing via LoadBalancer. |
| kubectl exec -it <pod> -- nslookup myservice | DNS resolution inside pod. | Debugged CoreDNS misconfiguration. |

## ****5. ConfigMaps & Secrets****

| **Command** | **Explanation** | **Real-Time Example** |
| --- | --- | --- |
| kubectl create configmap app-config --from-file=config.properties | Creates ConfigMap from file. | Loaded DB connection config into app pods. |
| kubectl get configmaps | List ConfigMaps. | Verified configs before redeploy. |
| kubectl describe configmap app-config | Show ConfigMap contents. | Checked DB connection string in config. |
| kubectl create secret generic db-secret --from-literal=username=admin --from-literal=password=pass123 | Creates secret. | Created MySQL credentials secret. |
| kubectl get secrets | List secrets. | Verified AWS credentials secret for external S3 bucket. |
| kubectl describe secret db-secret | Metadata only (not values). | Confirmed secret mounted to pod. |

## ****6. Namespaces & Contexts****

| **Command** | **Explanation** | **Real-Time Example** |
| --- | --- | --- |
| kubectl get namespaces | List namespaces. | Checked staging vs production namespaces. |
| kubectl create namespace dev | Create namespace. | Created isolated namespace for testing. |
| kubectl config get-contexts | Shows available cluster contexts. | Verified if I’m on prod or staging before deployment. |
| kubectl config use-context prod | Switch context. | Prevented accidental deployment to wrong cluster. |

## ****7. Resource Management (YAML)****

| **Command** | **Explanation** | **Real-Time Example** |
| --- | --- | --- |
| kubectl apply -f app.yaml | Apply changes (create/update). | Used GitOps pipeline to apply manifests. |
| kubectl delete -f app.yaml | Delete resources from file. | Removed old service + pods. |
| kubectl get -f app.yaml | List resources defined in file. | Checked if YAML resources were applied correctly. |
| kubectl diff -f app.yaml | Shows changes before applying. | Checked if rollout would affect replicas or probes. |

## ****8. Node & Scheduling****

| **Command** | **Explanation** | **Real-Time Example** |
| --- | --- | --- |
| kubectl cordon <node> | Marks node unschedulable. | Used before draining node for patch upgrade. |
| kubectl drain <node> --ignore-daemonsets | Evicts pods from node safely. | Migrated pods before rebooting node. |
| kubectl uncordon <node> | Makes node schedulable again. | Brought node back after OS patch. |
| kubectl taint nodes <node> key=value:NoSchedule | Prevents pods without toleration from running. | Tainted GPU nodes so only ML jobs scheduled. |
| kubectl label nodes <node> disktype=ssd | Add label to node. | Scheduled DB pods on SSD-backed nodes. |

## ****9. Monitoring & Debugging****

| **Command** | **Explanation** | **Real-Time Example** |
| --- | --- | --- |
| kubectl top pods | Show pod CPU/mem usage. | Found pod consuming 2Gi memory when limit was 1Gi → OOMKilled. |
| kubectl get events | Show recent cluster events. | Saw pod scheduling failures due to resource shortage. |
| kubectl describe <resource> | Detailed info with events. | Debugged service not binding to endpoints. |
| kubectl get all -n <ns> | Show all resources in namespace. | Checked complete state of staging namespace. |
| kubectl debug node/<node> --image=busybox | Run debug pod on node (v1.20+). | Debugged node network connectivity issues. |

## ****10. Advanced Commands****

| **Command** | **Explanation** | **Real-Time Example** |
| --- | --- | --- |
| kubectl rollout history deployment <name> | Shows rollout history. | Verified rollback revision for microservice. |
| kubectl edit deployment <name> | Opens editor to live-edit resource. | Hot-fixed wrong image tag. |
| kubectl patch deployment <name> -p '{"spec":{"replicas":3}}' | Patch resource inline. | Increased replicas without editing YAML. |
| kubectl api-resources | Lists all resource types supported by cluster. | Used to confirm CRDs installed for Istio. |
| kubectl explain pod.spec.containers | Documentation for resource fields. | Checked correct syntax for probes. |
| kubectl get pods --field-selector=status.phase=Pending | Filter pods by status. | Found pods pending due to no available nodes. |

## ****11. Jobs & CronJobs****

| **Command** | **Explanation** | **Real-Time Example** |
| --- | --- | --- |
| kubectl create job backup --image=busybox -- echo "Backup done" | Run one-time job. | Used for DB migration task. |
| kubectl get jobs | List jobs. | Checked success/failure of migration. |
| kubectl create cronjob db-backup --image=mysql:5.7 --schedule="0 0 \* \* \*" | Daily scheduled job. | Automated nightly DB backups. |

## ****12. RBAC & Security****

| **Command** | **Explanation** | **Real-Time Example** |
| --- | --- | --- |
| kubectl create role pod-reader --verb=get,list --resource=pods -n dev | Create role. | Limited dev access to pods only. |
| kubectl create rolebinding pod-read-bind --role=pod-reader --user=dev-user -n dev | Bind role to user. | Gave developer read-only pod access. |
| kubectl auth can-i get pods --as dev-user -n dev | Check permissions. | Verified RBAC for new dev-user. |