# **Kubernetes Theoretical Concepts**

## **Pods**

A Pod is the smallest and simplest unit in the Kubernetes object model. It represents a single instance of a running process in your cluster.

### Key Points:

* A Pod can contain one or more containers (most commonly one).
* All containers in a Pod:
  + Share the same network namespace (IP address and port space).
  + Can communicate with each other using localhost.
  + Share storage volumes if defined.
* Pods are ephemeral. If a Pod dies, Kubernetes does not restart it automatically unless it's managed by a higher-level object like a Deployment.
* Pods are typically created and managed by controllers (like Deployments, ReplicaSets, StatefulSets).

### Pod Lifecycle:

* Pending: The Pod has been accepted but not yet scheduled.
* Running: The Pod is bound to a node and containers are running.
* Succeeded: All containers have terminated successfully.
* Failed: One or more containers terminated with an error.
* Unknown: Pod state cannot be determined.

### Commands:

* Use command to install kubectx

*sudo apt install kubectx*

* To create a pod, navigate to a specific namespace or activate namespace using:

*kubens “namespace\_name”*

* To run the pod use command *kubectl run*

Example:

*kubectl run nginx –image=nginx* //the pod will be created

* *kubectl get pods*
* *kubectl get pods -n nginx* //where “nginx” is a namespace
* To delete the pod from default or current namespace:

*kubectl delete pod “pod\_name”*

* To delete the pod from a specific namespace:

*kubectl delete pod “pod\_name” -n “namespace”*

* To create pods make a *pod.yml* file
  + *kind: Pod*

*apiVersion: v1*

*metadata:*

*name: nginx-pod*

*namespace: nginx*

*spec:*

*contaienrs:*

*- name: nginx*

*image: nginx:latest*

*ports:*

*- containerPort: 80*

* Run below command to execute the file

*kubectl apply -f “file\_name”*

* To enter into the pod use command:

*kubectl exec -it nginx-pod -n nginx -- bash*

## **2. Namespaces**

A Namespace is a way to divide cluster resources between multiple users or teams. They are virtual clusters backed by the same physical cluster.

### Key Points:

* Namespaces are intended for environmental separation (e.g., dev, test, prod).
* Resources in one namespace cannot directly see or access resources in another namespace unless explicitly configured.
* Some resources like nodes, persistent volumes are not namespaced.
* By default, Kubernetes includes:
  + default: For objects without a specified namespace.
  + kube-system: For system components.
  + kube-public: Readable by all users.
* Namespaces help with:
  + Multi-tenancy
  + Resource Quotas
  + Isolation and Access Control

### Commands:

* If Minikube – *minikube start*
* *kubectl get namespaces*
* To get info about current serving cluster - *kubectl cluster-info*
* *kubectl create namespace my-namespace*
* Other way to create namespace via configuration file –

*apiVersion: v1*

*kind: ConfigMap*

*metadata:*

*name: mysql-config*

*namespace: my-namespace*

*data:*

*db\_url: mysqlservice.database*

where “mysql-service” is database service url and “database” is the namespace name

* To list all resources bound to a namespace – *kubectl api-resources –namespaced=true*
* To list all resources that are not bound to a namespace – *kubectl api-resources –namespaced=false*

Example – Volumes and nodes

* Creating and running namespace configuration file - *kubectl apply -f mysqlconfig.yml*
* *kubectl get configmap -n my-namespace*
* For easy handling install kubectx on machine
* *sudo apt install kubectx*
* to check all the namespaces –

*kubens* //the default namespace will be highlighted

* to set new active namespace –

*kubens “new\_namespace”*

* To check whether it is running or not:

*kubectl port-forward pod/nginx-pod 8080:80 -n nginx*

*curl 127.0.0.1:8080*

* To get info about a particular pod:

*kubectl describe pod/nginx-pod*

## **3. Deployments**

A Deployment is a higher-level object that manages Pods and ensures the desired number of Pods are always running.

### Key Points:

* A Deployment:
  + Creates ReplicaSets.
  + ReplicaSets create and manage the Pods.
* Used to:
  + Roll out new versions of an application.
  + Roll back to previous versions if needed.
  + Scale applications up or down.
* Supports rolling updates (zero downtime deployments).
* Automatically replaces Pods if they fail.

### Deployment Strategies:

* RollingUpdate (default): Gradually replaces old Pods with new ones.
* Recreate: Shuts down all old Pods before starting new ones.

### Commands:

* Create a deployment.yml file for configuration

*kind: Deployment*

*apiVersion: apps/v1*

*metadata:*

*name: nginx-deployment*

*namespace: nginx*

*spec:*

*replicas: 2*

*selector:*

*matchLabels:*

*app: nginx*

*template:*

*metadata:*

*name: nginx-dep-pod*

*labels:*

*app: nginx*

*spec:*

*containers:*

*- name: nginx*

*image: nginx:latest*

*port: 80*

* *kubectl apply -f deployment.yml*
* *kubectl get deployment -n nginx* //Use only when accessing from another namespace

*OR*

* *kubectl get deployment*
* *kubectl get pods*

*OR*

* *kubectl get pods -n nginx* //Use only when accessing from another namespace
* To scale up the deployments*:*

*kubectl scale deployment/nginx-deployment -n nginx --replicas=5* and then check the pods with the help of *kubectl get pods OR kubectl get pods -n nginx*

* Rolling updates commands:

*kubectl set image deployment/nginx-deployment -n nginx nginx=nginx:1.27.3*

1. *kubectl* – CLI Kubernetes
2. *set image* – command to update the image of running container
3. *deployment/nginx-deployment* – targeting the nginx-deployment in the cluster
4. *-n nginx* – specifies the namespace nginx
5. *Nginx nginx=nginx:1.27.3* – updates the container nginx in the deployment to use the nginx version 1.27.3
6. *kubectl get pods*

### Summary Table

| Concept | Purpose | Key Feature |
| --- | --- | --- |
| Pod | Smallest deployable unit | Can have one or more containers |
| Namespace | Logical separation in cluster | Isolation, multi-tenancy |
| Deployment | Manages Pod lifecycle | Scaling, updates, rollback support |

* kubectl context usually defaults to the default namespace.
* When creating resources, if namespace is not specified, it goes to the default namespace.
* You can switch namespaces using:
* kubectl config set-context --current --namespace=dev

## **4. Replicasets:**

## 1. What is a ReplicaSet?

A ReplicaSet (RS) is a Kubernetes controller that ensures a specified number of pod replicas are running at any given time.

### Key Points:

* ReplicaSet maintains pod availability.
* If a pod crashes or is deleted, the ReplicaSet automatically creates a replacement pod.
* ReplicaSets are often not created directly by users. They are automatically created and managed by Deployments.
* Standalone ReplicaSets can be used, but Deployments are preferred as they offer more advanced features like rolling updates and rollbacks.

## 2. Why Use ReplicaSets?

* Ensures high availability by maintaining the desired number of pods.
* Provides self-healing: replaces failed pods automatically.
* Useful for scaling applications (adjust the number of replicas easily).

### 3. Core Concepts

### Selector

* Used to match the ReplicaSet to the pods it should manage (based on labels).
* The ReplicaSet monitors only the pods with matching labels.

### Labels

* Attached to pods.
* Identifies which pods belong to which ReplicaSet.

### Template

* Describes how the new pods should be created (just like in a Deployment).
* Includes the pod specification: image, container, ports, etc.

## 4. ReplicaSet YML Example

## *kind: ReplicaSet*

## *apiVersion: apps/v1*

## *metadata:*

## *name: nginx-replicasets*

## *namespace: nginx*

## *spec:*

## *replicas: 2*

## *selector:*

## *matchLabels:*

## *app: nginx*

## *template:*

## *metadata:*

## *name: nginx-rep-pod*

## *labels:*

## *app: nginx*

## *spec:*

## *containers:*

## *- name: nginx*

## *image: nginx:latest*

## *ports:*

## *- containerPort: 80*

### Explanation:

* replicas: 3 → Desired number of pods.
* selector → Matches pods with label app: nginx.
* template → Describes the pod to be created.

## 5. Important Commands

| Command | Purpose |
| --- | --- |
| *kubectl apply -f replicaset.yml* | Create a ReplicaSet |
| *kubectl get rs* | List all ReplicaSets |
| *kubectl describe rs nginx-replicaset* | Show detailed ReplicaSet info |
| *kubectl scale rs nginx-replicaset --replicas=5* | Scale ReplicaSet to 5 pods |
| *kubectl delete rs nginx-replicaset* | Delete the ReplicaSet |

## 6. How ReplicaSets Work Internally

1. When you create a ReplicaSet, Kubernetes compares the desired replicas with the currently running pods.
2. If fewer pods are running → New pods are created.
3. If more pods are running → Extra pods are terminated.
4. ReplicaSets continuously monitor the state to maintain the desired number.

## 7. Difference Between ReplicaSet and Deployment

| Feature | ReplicaSet | Deployment |
| --- | --- | --- |
| Purpose | Maintain pod replicas | Manage ReplicaSets, support updates and rollbacks |
| Rolling Updates | Not supported | Supported |
| Rollbacks | Not supported | Supported |
| Manual Scaling | Supported | Supported |
| Preferred Use | Rarely used directly | Commonly used |

### Summary:

* Deployment → Full lifecycle manager (recommended)
* ReplicaSet → Just keeps pods alive and at desired count

### 8. Self-Healing Example

* Desired replicas: 3
* If one pod is deleted → ReplicaSet immediately creates a new pod to restore the count to 3.
* ReplicaSet constantly watches the pods with matching labels.

### 9. Selector and Label Relationship

* The selector in ReplicaSet is what tells it which pods to manage.
* Only pods with labels matching the selector will be controlled by the ReplicaSet.

Example:

selector:

matchLabels:

app: nginx

* All pods with the label app: nginx will be controlled by this ReplicaSet.

### 10. Summary Table

| Concept | Purpose | Key Feature |
| --- | --- | --- |
| Pod | Smallest deployable unit | Runs one or more containers |
| Namespace | Logical separation in cluster | Isolation, multi-tenancy |
| Deployment | Manages Pod lifecycle | Rolling updates, rollback |
| ReplicaSet | Maintains desired number of pods | Self-healing, scaling |

### Additional Notes:

* Do not directly update pods created by a ReplicaSet. The RS will detect a difference and create new pods to match the original specification.
* ReplicaSets are almost always used as part of a Deployment in real-world Kubernetes setups.

## **5. Daemonsets:**

### 1. What is a DaemonSet?

A DaemonSet is a Kubernetes controller that ensures a copy of a specific pod runs on every node (or on selected nodes) in the cluster.

Key Points:

* Ensures one pod per node.
* Automatically runs the pod on new nodes when they are added to the cluster.
* Commonly used for infrastructure-related tasks that need to run across all nodes.

### 2. Why Use DaemonSets?

DaemonSets are typically used to deploy:

* Log collection agents (e.g., Fluentd, Logstash)
* Monitoring agents (e.g., Prometheus Node Exporter)
* Networking components (e.g., Calico, Cilium)
* Storage daemons (e.g., GlusterFS, Ceph)
* Security agents (e.g., anti-virus scanners, intrusion detection)

### 3. How DaemonSets Work

* When a new node is added to the cluster → The DaemonSet automatically schedules a pod on it.
* When a node is removed → The pod is also removed.
* DaemonSets run one pod per node by default (unless node selectors or taints/tolerations are used to control where pods can run).

### 4. DaemonSet YAML Example

## *kind: DaemonSet*

## *apiVersion: apps/v1*

## *metadata:*

## *name: nginx-daemonsets*

## *namespace: nginx*

## *spec:*

## *selector:*

## *matchLabels:*

## *app: nginx*

## *template:*

## *metadata:*

## *name: nginx-dmn-pod*

## *labels:*

## *app: nginx*

## *spec:*

## *containers:*

## *- name: nginx*

## *image: nginx:latest*

## *ports:*

## *- containerPort: 80*

### Explanation:

* kind: DaemonSet → This resource is a DaemonSet.
* selector → Matches pods that the DaemonSet should manage.
* template → Describes the pod that should be created on each node.

### 5. Key Characteristics of DaemonSets

| Feature | Description |
| --- | --- |
| Pod Scheduling | One pod per node automatically |
| New Node Behavior | Automatically schedules a pod on the new node |
| Node Removal Behavior | Pod is automatically removed |
| Manual Scaling | Not applicable (automatically scales per node) |
| Deletion Behavior | Deletes all DaemonSet pods when DaemonSet is deleted |

### 6. Important DaemonSet Commands

| Command |  |  | Purpose |
| --- | --- | --- | --- |
| *kubectl apply -f daemonset.yml* |  |  | Create the DaemonSet |
| *kubectl get daemonset* |  |  | List all DaemonSets |
| *kubectl describe daemonset nginx-daemonset* |  |  | Get detailed DaemonSet information |
| *kubectl delete daemonset nginx-daemonset* |  |  | Delete the DaemonSet and its pods |
| *kubectl get pods -o wide* |  |  | See on which nodes the pods are running |

### USES OF ALL SETS:

DAEMONSETS: At least one replica should be running on each and every node.

REPLICASETS: It will create the replicas of pods according to the number you give.

STATEFULSETS: It will keep the same states of all the replicas.

DEPLOYMENT: Rolling updates, scaling, auto scaling auto healing is done here.

## **6. JOBS**

### 1. Kubernetes Job – Overview

A Job in Kubernetes ensures that a specified number of pods successfully complete a task.  
Unlike Deployments or DaemonSets that run continuously, a Job runs to completion.

### Key Points:

* Runs pods to completion.
* Ensures the desired number of completions (successful runs).
* Useful for one-time tasks (e.g., data processing, backups, batch jobs).
* If a pod fails, the Job automatically starts a new pod to retry the task.
* Jobs can run single or parallel tasks.

### Job YAML Example:

*kind: Job*

*apiVersion: batch/v1*

*metadata:*

*name: demo-job*

*namespace: nginx*

*spec:*

*completions: 1*

*parallelism: 1*

*template:*

*metadata:*

*name: demo-job-pod*

*labels:*

*app: batch-task*

*spec:*

*containers:*

*- name: batch-container*

*image: busybox:latest*

*command: ["sh","-c","echo Jobs demo. && sleep 10"]*

*restartPolicy: Never*

### Job Key Fields:

| Field | Purpose |
| --- | --- |
| completions | Number of successful runs required |
| backoffLimit | Number of retries on pod failure |
| restartPolicy | Should always be Never for Jobs |
| template | Pod specification |

### 2. Job Behavior & Execution

| Situation | Job Response |
| --- | --- |
| Pod completes successfully | Counts toward completions |
| Pod fails | Retries (up to backoffLimit) |
| All completions met | Job marked as successful |
| All retries exhausted | Job marked as failed |

### 3. Important Job Commands

| Command | Purpose |
| --- | --- |
| *kubectl apply -f job.yml* | Create the Job |
| *kubectl get jobs* | List all Jobs |
| *kubectl describe job example-job* | Detailed Job info |
| *kubectl delete job example-job* | Delete the Job |
| *kubectl get pods --selector=job-name=example-job* | List pods related to the Job |
| *kubectl logs pod/[pod\_name]* | To check the logs of pods |

### 4. Job Parallelism & Completion Modes

Jobs can run:

* Single Pod Jobs: Runs one pod to completion.
* Parallel Jobs:
  + Run multiple pods simultaneously.
  + Use parallelism and completions fields.

### spec:

parallelism: 3

completions: 6

3 pods run simultaneously, job finishes after 6 completions.

### 5. Advanced Job Configurations

* RestartPolicy: Never → The pod will not be restarted by itself.
* ActiveDeadlineSeconds: Optional time limit for the job.
* TTLSecondsAfterFinished: Automatically cleans up job after completion.

### 6. When to Use a Job

* Batch Processing
* Data Migrations
* Database Backups
* Image Processing
* One-time Data Analysis

## 7. Kubernetes CronJob

A CronJob in Kubernetes is like a Linux cron—it runs Jobs on a scheduled, recurring basis.

### Key Points:

* Runs Jobs on a predefined schedule.
* Schedule format uses cron syntax (minute, hour, day, etc.).
* Automatically creates a new Job at each scheduled time.
* Each scheduled Job runs independently.

### CronJob YAML Example:

*apiVersion: batch/v1*

*kind: CronJob*

*metadata:*

*name: minute-backup*

*namespace: nginx*

*spec:*

*schedule: "\* \* \* \* \*"*

*jobTemplate:*

*spec:*

*template:*

*metadata:*

*name: minute-backup*

*labels:*

*app: minute-backup*

*spec:*

*containers:*

*- name: backup-container*

*image: busybox*

*command:*

*- sh*

*- -c*

*- >*

*echo "Backup Started" ;*

*mkdir -p /backups &&*

*mkdir -p /demo-data &&*

*cp -r /demo-data /backups &&*

*echo "Backup Completed" ;*

*volumeMounts:*

*- name: data-volume*

*mountPath: /demo-data*

*- name: backup-volume*

*mountPath: /backups*

*restartPolicy: OnFailure*

*volumes:*

*- name: data-volume*

*hostPath:*

*path: /demo-data*

*type: DirectoryOrCreate*

*- name: backup-volume*

*hostPath:*

*path: /backups*

*type: DirectoryOrCreate*

### CronJob Key Fields:

| Field | Purpose |
| --- | --- |
| schedule | Cron format schedule (e.g. "0 \* \* \* \*") |
| jobTemplate | The job definition to be run |
| successfulJobsHistoryLimit | Retain X successful jobs (optional) |
| failedJobsHistoryLimit | Retain X failed jobs (optional) |
| concurrencyPolicy | Allow, Forbid, Replace overlapping jobs |
| startingDeadlineSeconds | Deadline for starting missed job runs |

### 8. Cron Syntax Quick Reference

| Cron Syntax | Meaning |
| --- | --- |
| \* \* \* \* \* | Every minute |
| \*/5 \* \* \* \* | Every 5 minutes |
| 0 0 \* \* \* | Every day at midnight |
| 0 \* \* \* \* | Every hour |

### 9. Important CronJob Commands

| Command | Purpose |
| --- | --- |
| *kubectl apply -f cronjob.yml* | Create the CronJob |
| *kubectl get cronjobs* | List all CronJobs |
| *kubectl describe cronjob hello-cronjob* | Get detailed CronJob info |
| *kubectl delete cronjob hello-cronjob* | Delete the CronJob |
| *kubectl get jobs* | List Jobs created by the CronJob |
| *kubectl logs <job-pod-name>* | View logs from the job pod |

### 10. Concurrency Policies for CronJobs

| Policy | Behavior |
| --- | --- |
| Allow | Allows concurrent runs of the job |
| Forbid | Skips new job if the previous one is still running |
| Replace | Cancels the currently running job and starts a new one |

concurrencyPolicy: Forbid

11. Advanced CronJob Settings

* startingDeadlineSeconds: Time window in seconds to start a missed job if the CronJob controller was down.
* successfulJobsHistoryLimit: Limits how many successful jobs are kept in history.
* failedJobsHistoryLimit: Limits how many failed jobs are kept in history.
* Suspend: Temporarily disables the CronJob schedule.

### 12. Common CronJob Use Cases

* Daily Database Backups
* Periodic Log Rotation
* Regular Batch Data Imports
* Cleanup Scripts
* Scheduled Report Generation

### 13. Job vs CronJob vs Deployment vs DaemonSet Summary Table

| Feature | Job | CronJob | Deployment | DaemonSet |
| --- | --- | --- | --- | --- |
| Purpose | Run a task to completion | Run tasks on schedule | Long-running applications | One pod per node |
| Run Mode | Once | Recurring | Always running | Always running |
| Scheduling | Immediate | Cron-based | N/A | One per node |
| Scalability | Parallel Jobs supported | Creates Jobs automatically | Manual or auto scaling | One per node |
| Pod Restart Policy | Never | Never | Always | Always |

### 14. Quick Memory Recap:

* Job: Run once, wait for successful completion.
* CronJob: Run Jobs on a schedule.
* Deployment: Manage long-running apps, scaling and rolling updates.
* DaemonSet: Ensure a pod runs on every node.