Devops Shack

FUNDAMENTAL KUBERNETES

A PRACTICAL HELPBOOK

By DevOps Shack

[Click here for DevSecOps & Cloud DevOps Course]

DevOps Shack

Welcome to the world of Kubernetes — the open-source platform revolutionizing how applications are deployed, managed, and scaled in today's cloud environments. Whether you're a developer, system administrator, or DevOps engineer, having a solid grasp of Kubernetes fundamentals is essential for modern software delivery.

This helpbook breaks down the core Kubernetes concepts into simple, digestible explanations paired with real-world examples. You'll build a strong foundation that empowers you to confidently work with Kubernetes clusters and deploy containerized applications smoothly and efficiently.

What's Inside This Helpbook?

- **Pods & Nodes:** Understand the smallest unit and the worker machines in Kubernetes.
- **Cluster & Control Plane:** Learn how nodes work together and how Kubernetes manages the entire system.
- **Deployments & ReplicaSets:** Manage app lifecycle and ensure reliability with multiple pod copies.
- **Services & Ingress:** Discover how networking and external access are handled.
- **ConfigMaps & Secrets:** Safely manage configuration data and sensitive information.
- **Namespaces, Labels, & Selectors:** Organize and dynamically group Kubernetes resources.
- **Resource Management:** Control CPU and memory usage for efficient operation.
- **Advanced Concepts:** Dive into Helm, CRDs, Operators, DaemonSets, and more to automate and extend Kubernetes capabilities.

- **Monitoring & Troubleshooting:** Use logs, events, and rolling updates to maintain healthy, highly available applications.

1. Pod – The Basic Building Block of Kubernetes

“Think of a Pod as a **shared room** where one or more roommates (containers) live together and share utilities like Wi-Fi (network) and a kitchen (storage).”

A **Pod** is the smallest deployable unit in Kubernetes. It wraps one or more containers that need to work closely together. These containers share the same network IP and can communicate easily. Pods make it easy to manage containerized applications.

🧠 **Example:**
A web server (NGINX) and a log collector (Fluentd) container can be placed in a single pod so they can share logs and network seamlessly.

📌 **Key Points:**

- Shares storage and network
- Typically contains one container
- Automatically replaced if it fails

☸ 2. Node – The Worker Bee of Kubernetes

“Imagine a Node as a **factory worker** on a production line. Each worker (node) takes tasks (pods) from the manager (control plane) and gets them done.”

A **Node** is a physical or virtual machine that runs your applications in the form of pods. It contains everything needed to run them — a container runtime, kubelet agent, and kube-proxy.

🧠 **Example:**
If you're deploying a blog website, Node-1 might run the front-end UI pod, and Node-2 may run the database pod.

📌 **Key Points:**

- Managed by the control plane
- Can be a cloud VM or a physical server

- Runs one or more pods

🧱 3. Cluster – The Whole Construction Site

"A Cluster is like an **entire construction site** filled with workers (nodes) and a site manager (control plane) orchestrating the work."

A **Kubernetes Cluster** is a group of nodes (machines) working together to run your apps. The control plane decides what should run and where. It ensures everything stays healthy and balanced.

🧠 **Example:**
You may have 3 nodes — one runs a web app, one runs a database, and another stores uploaded files. The cluster manages all of them as one system.

📌 **Key Points:**

- Combines many nodes into one system
- Provides scalability and fault tolerance
- Central to Kubernetes functioning

🚀 4. Deployment – The App Launcher

"A Deployment is like a **mission control center** that ensures your app is always up and running in the correct number of copies."

A **Deployment** manages Pods and ensures the desired number of them are always running. It handles rolling updates, version control, and recovery from failure.

🧠 **Example:**
You want to run 3 instances of your web app. A deployment ensures that exactly 3 pods are always active, replacing any that crash.

📌 **Key Points:**

- Manages pod creation and updates
- Supports rollbacks and rollouts
- Maintains app availability

📦 5. ReplicaSet – The Pod Multiplier

“Think of a ReplicaSet as a **photocopier** that ensures there are always exactly X copies of a document (Pod).”

A **ReplicaSet** ensures a specified number of pod replicas are running at any given time. If one pod crashes, it automatically creates a new one.

🧠 **Example:**
If you need 5 pods of your backend API for load balancing, the ReplicaSet keeps them running — even if one fails, another spins up.

📌 **Key Points:**

- Ensures high availability
- Automatically maintains pod count
- Works behind the scenes of a deployment

🌐 6. Service – The Stable Doorway to Your Pods

“A Service is like a **reception desk** that directs visitors to the right person (pod), even if the staff keeps changing.”

A **Service** provides a stable endpoint to access a group of pods, regardless of changes in their IP addresses. Since pods are ephemeral and can be recreated with different IPs, services ensure that internal or external users always reach the right destination.

🧠 **Example:**
Your frontend needs to talk to a backend API. Instead of tracking each pod's IP, it talks to the service, which routes requests to healthy backend pods.

📌 **Key Points:**

- Abstracts dynamic pod IPs
- Enables reliable communication
- Can be internal or exposed externally

🌍 7. Ingress – The Smart Gatekeeper

“Ingress is like a **smart security gate** that not only lets visitors in but also guides them to the right building (service) based on the request.”

An **Ingress** manages external HTTP and HTTPS traffic to Kubernetes services. It provides routing rules, SSL termination, and virtual hosting — all in one place.

🧠 **Example:**
You want requests to /shop to go to the shopping service and /blog to go to the blog service. Ingress makes this routing clean and efficient.

📌 **Key Points:**

- Routes external traffic based on URL or host
- Can handle TLS/SSL
- Reduces the need for multiple LoadBalancers

🗂️ 8. ConfigMap – The Configuration Organizer

“A ConfigMap is like a **settings file** outside the app code, so you can update behavior without redeploying the whole app.”

A **ConfigMap** stores non-sensitive configuration data (like environment variables, config files, etc.) separately from the container image. It lets you customize app behavior dynamically.

🧠 **Example:**
You can store a log level (DEBUG, INFO, ERROR) in a ConfigMap and change it without rebuilding your Docker image.

📌 **Key Points:**

- Stores environment configs
- Keeps app code and settings separate
- Dynamically injected into pods

🔐 9. Secret – The Secure Vault

"A Secret is like a **locked safe** that holds sensitive info like passwords and tokens — securely and discreetly."

A **Secret** stores confidential data like API keys, SSH credentials, or TLS certificates. Unlike ConfigMaps, Secrets are base64-encoded and managed securely by Kubernetes.

🧠 **Example:**
Instead of hardcoding your database password into the app, you store it in a Secret and mount it into the pod securely.

📌 **Key Points:**

- Stores sensitive info securely
- Encrypted at rest (if configured)
- Can be injected into pods via environment variables or volumes

🧭 10. Namespace – The Kubernetes Filing Cabinet

"A Namespace is like a **department folder** in a filing system — it keeps things organized and prevents clutter."

A **Namespace** allows you to divide a Kubernetes cluster into multiple logical environments. This helps separate applications, teams, or projects and apply policies like resource limits or access control per namespace.

🧠 **Example:**
Your dev team works in the dev namespace and your QA team in the qa namespace. Both can deploy apps without interfering with each other.

📌 **Key Points:**

- Logical separation within a cluster
- Enables multi-tenancy
- Useful for access control and quota management

🔧 11. Kubelet – The Node Supervisor

"The Kubelet is like a **floor manager** in a factory, making sure every machine (container) on its floor (node) is working as expected."

The **Kubelet** is an agent that runs on each node in the Kubernetes cluster. It ensures that containers described in the Pod specs are up and running. It communicates with the control plane to receive pod instructions and reports back the node's health and status.

🧠 **Example:**
If the control plane tells a node to run 2 containers for an app, the Kubelet ensures they are created and stays running — restarting if needed.

📌 **Key Points:**

- Runs on every node
- Ensures containers are healthy
- Communicates with control plane

💻 12. kubectl – The Kubernetes Command Line Interface

"kubectl is like your **remote control** for the Kubernetes cluster — it lets you deploy, inspect, and control everything from one place."

kubectl is the official command-line tool used to interact with your Kubernetes cluster. Whether it's checking pod status, scaling deployments, or updating configs, kubectl is your go-to command.

🧠 **Example:**
You can run kubectl get pods to see running pods, or kubectl apply -f app.yaml to deploy your app using a YAML file.

📌 **Key Points:**

- CLI for interacting with Kubernetes
- Supports debugging and management
- Works with YAML manifests

🧠 13. Control Plane – The Brain of the Cluster

"The Control Plane is like the **head office** — it makes all decisions, plans, and supervises the whole operation."

The **Control Plane** is the central component that manages the state of the Kubernetes cluster. It schedules workloads, manages the desired state, handles scaling, and monitors health.

🧠 **Example:**
If you ask for 5 replicas of your app, the control plane ensures that exactly 5 pods are running — spinning up more if one crashes.

📌 **Key Points:**

- Manages the entire cluster
- Includes API server, scheduler, controller manager, etcd
- Decides what runs where

🧮 14. Scheduler – The Matchmaker for Pods and Nodes

"The Scheduler is like a **dispatch officer**, assigning pods to the best available workers (nodes) based on their needs and availability."

The **Scheduler** watches for unscheduled pods and assigns them to suitable nodes. It considers resource needs, node health, affinity rules, and taints/tolerations.

🧠 **Example:**
A pod needing high memory will be scheduled on a node with enough available RAM. The scheduler makes this decision automatically.

📌 **Key Points:**

- Assigns pods to nodes
- Based on resources and rules
- Key to workload distribution

🛠️ 15. Controller Manager – The Auto-Pilot of Kubernetes

"The Controller Manager is like a **robotic assistant** that automatically fixes and maintains things to match the desired state."

The **Controller Manager** runs control loops that watch the cluster's state and make changes to move toward the desired state. It includes various controllers — like ReplicaSet, Node, and Job controllers.

🧠 **Example:**
If a node goes down and pods disappear, the controller manager starts new pods on healthy nodes to maintain the desired count.

📌 **Key Points:**

- Runs background controllers
- Maintains cluster health and state
- Automates fault recovery and scaling

🗄️ 16. Etcd – The Cluster's Memory Vault

"Etcd is like a **secure filing cabinet** that stores the brain's memory — the cluster's entire configuration and state."

etcd is a distributed key-value store used by Kubernetes to store all cluster data, such as node information, pod specs, secrets, and more. It is the **single source of truth** for the entire system.

🧠 **Example:**
If you deploy an app, the configuration and desired state are saved in etcd. When the control plane needs to check or update the cluster's state, it consults etcd.

📌 **Key Points:**

- Stores all cluster data
- Consistent and highly available
- Critical for Kubernetes operation

🚫☑️ 17. Taints & Tolerations – The Bouncers and Guest Passes

"Taints are like **VIP-only signs**, and Tolerations are like **guest passes** that let approved pods in."

Taints are applied to nodes to repel certain pods, while **Tolerations** allow specific pods to bypass those rules and get scheduled on tainted nodes.

🧠 **Example:**
A GPU node might be tainted so that only AI workloads with a matching toleration can be scheduled there.

📌 **Key Points:**

- Taints repel pods
- Tolerations let pods tolerate taints
- Help control pod placement

🏷️ 18. Labels & Selectors – The Tags & Filters of Kubernetes

"Labels are like **name tags**, and Selectors are like **search filters** to find the right group of resources."

Labels are key-value pairs attached to Kubernetes objects (like pods or services). **Selectors** allow you to find and group objects using these labels for operations like deployment, service targeting, or monitoring.

🧠 **Example:**
Label your frontend pods with app=frontend and use a selector to target them in your service or deployment strategy.

📌 **Key Points:**

- Labels tag objects
- Selectors match based on labels
- Used in grouping, filtering, and targeting

⚙️ 19. Resource Requests & Limits – The Resource Budget

"Think of Requests as a **minimum salary** and Limits as a **maximum budget cap** for your containers."

Resource Requests specify the minimum CPU and memory a container needs, while **Limits** define the maximum it can use. This helps the scheduler place workloads efficiently and prevents one container from hogging all resources.

🧠 **Example:**
You can request 200Mi of memory and limit it to 500Mi. If it crosses 500Mi, Kubernetes may throttle or kill the container.

📌 **Key Points:**

- Requests guide scheduling
- Limits cap resource usage
- Prevent noisy neighbor problems

📦 20. Helm – The App Store for Kubernetes

"Helm is like the **package manager** (apt, yum, npm) of Kubernetes — it installs complex apps with just a few commands."

Helm helps you define, install, and manage Kubernetes applications using packages called **charts**. These charts bundle all the required resources (Deployments, Services, ConfigMaps, etc.) into reusable templates.

🧠 **Example:**
Want to install WordPress with MySQL on your cluster? A single Helm chart can do it, handling everything from deployments to networking.

📌 **Key Points:**

- Simplifies deployment of complex apps
- Uses versioned charts
- Supports upgrades and rollbacks

🛠️ 21. CRD (Custom Resource Definition) – Build Your Own Kubernetes Object

"A CRD is like adding a **new item to the restaurant menu** — it lets you create custom Kubernetes objects tailored to your needs."

Custom Resource Definitions allow you to define your own resources beyond the built-in ones (like Pods or Services). Once registered, you can create and manage these custom objects just like any native Kubernetes object.

🧠 **Example:**
You can define a CRD called Database to represent your custom database deployments, and manage it with kubectl.

📌 **Key Points:**

- Adds custom types to Kubernetes
- Useful for extensions and domain-specific needs
- Managed like native resources

🤖 22. Operator – Automated App Caretaker

"An Operator is like a **robotic admin** that knows how to install, manage, and heal a complex application."

Operators use CRDs and controllers to automate the full lifecycle of applications. From provisioning and scaling to upgrades and recovery, Operators handle it without manual intervention.

🧠 **Example:**
A MySQL Operator can automatically create, back up, and recover databases across your cluster.

📌 **Key Points:**

- Built using CRDs + Controllers
- Automates complex app tasks
- Follows Kubernetes-native patterns

🔁 23. DaemonSet – One Pod per Node

"DaemonSets are like **security guards** placed at every entrance — they ensure one pod runs on every node."

A **DaemonSet** ensures that a copy of a pod runs on **all nodes** (or a subset). They're used for things like log collection, monitoring, or networking tools that must exist everywhere.

🧠 **Example:**
You could use a DaemonSet to run a log shipper like Fluentd on every node in your cluster.

📌 **Key Points:**

- Ensures uniform pod distribution
- Used for system-level services
- Automatically adjusts with node changes

📋 24. Logs & Events – The Cluster's Diary

"Logs and events are the **black box** of Kubernetes — helping you trace what happened, when, and why."

Logs show container output, and **Events** track what's happening in the cluster (like pod restarts, scheduling failures, etc.). Both are essential for troubleshooting and observability.

🧠 **Example:**
If a pod crashes, you can run kubectl logs <pod> to inspect its output and kubectl describe pod <pod> to check related events.

📌 **Key Points:**

- Logs show runtime output
- Events record cluster activity
- Vital for debugging and auditing

🔄 25. Rolling Updates & Rollbacks – Seamless App Upgrades

“Rolling Updates are like **smooth costume changes on stage** — updating your app without interrupting the show. Rollbacks are the quick rewind button if something goes wrong.”

Kubernetes supports **rolling updates** to upgrade your applications gradually by replacing pods one at a time, ensuring zero downtime. If the new version has issues, you can easily **rollback** to the previous stable version.

🧠 **Example:**
When you update a Deployment with a new container image, Kubernetes updates pods incrementally. If crashes happen, you can rollback to the last working version with a simple command.

📌 **Key Points:**

- Ensures zero downtime during updates
- Updates pods incrementally
- Easy rollback for quick recovery