



FENIL GAJJAR

KUBERNETES

DAILY TASKS

 Welcome to the Kubernetes Series – A New Chapter in the DevOps Journey! 

· · KUBERNETES UNVEILED

THE FUTURE OF CONTAINER ORCHESTRATION

- COMPREHENSIVE GUIDE
- THEORY + PRACTICAL
- REAL TIME SCENARIO TASKS
- DAILY TASKS OF KUBERNETES SERIES

FOLLOW
FOR
MORE

CONTACT US:

fenilgajjar.devops@gmail.com





Getting Started with

Kubernetes :

The Backbone of Modern

DevOps 



Welcome to the Kubernetes Series – A New Chapter in the DevOps Journey!

First and foremost, **a massive thank you to each and every one of you** who supported me throughout my **AWS daily tasks series**. Over the past few weeks, I've shared hands-on insights, real-world implementations, and in-depth documentation covering **20+ AWS services**. Your engagement, feedback, and encouragement have made this journey truly incredible! 🙌

But as we all know, **DevOps is not just about AWS—it's a continuous journey of automation, scalability, and efficiency**. While AWS gave us a solid foundation, it's time to move forward into another game-changing technology—**Kubernetes!**



Why Kubernetes? Why Now?

In today's cloud-native world, businesses rely on **containers and microservices** to build scalable, resilient applications. But managing containers manually? That's where Kubernetes changes the game!

Kubernetes is the **backbone of modern container orchestration**, enabling seamless deployment, scaling, and management of containerized applications. **For any DevOps engineer, mastering Kubernetes is essential** for automating workflows, improving reliability, and optimizing resource usage.



What's Coming in This Series?

Just like my AWS series, I'll be breaking down **Kubernetes concepts into daily tasks**, ensuring both **theory and hands-on practicals** to build a strong understanding. Whether you're an absolute beginner or looking to enhance your skills, this series will cover:

- ✓ **Fundamentals of Kubernetes** – What it is, why it matters, and how it works
- ✓ **Core Kubernetes Components** – Pods, Deployments, Services, ConfigMaps, Secrets, and more
- ✓ **Hands-On Labs** – Step-by-step tasks with real-world applications
- ✓ **Best Practices & Use Cases** – Implementing Kubernetes efficiently in DevOps workflows

This is not just a tutorial—it's a structured, practical learning experience!




About This Document



This document serves as the **first step into Kubernetes—Introduction to Kubernetes**. It will help you:



- Understand **what Kubernetes is and why it's important** in DevOps.
- Learn about **its key features and benefits** over traditional deployment methods.
- Explore **real-world scenarios where Kubernetes plays a crucial role**.

This is your **starting point** to mastering Kubernetes! By the end of this document, you'll have a **clear vision** of why Kubernetes is a must-have skill for any DevOps professional and how it fits into modern cloud-native architectures.

Let's Build This Kubernetes Journey Together!

As we kick off this series, I'd love to hear your thoughts! **What Kubernetes topics are you most excited to learn?** Drop your suggestions, and let's make this series as valuable as possible. 

 **Missed my AWS tasks? No worries! You can still check them out on my LinkedIn profile.** 

So, **welcome aboard!** Let's dive deep into Kubernetes, one task at a time, and continue this **DevOps journey together!**  



Life Before Kubernetes: The Challenges of Traditional Deployment

Before Kubernetes emerged as the gold standard for container orchestration, application deployment and management faced significant hurdles. Organizations relied on manual processes, rigid infrastructure, and inefficient scaling strategies, making deployments complex and time-consuming. Let's explore the key challenges faced before Kubernetes revolutionized the industry.



Traditional Deployment on Physical Servers

Before virtualization and containers, applications were deployed directly on physical servers. This led to:

- **Inefficient Resource Utilization** – Each server ran a single application or multiple applications with conflicting resource needs, leading to over-provisioning or resource starvation.
- **Scalability Issues** – Scaling required manually provisioning new servers, which was expensive and slow.
- **Application Conflicts** – Running multiple applications on the same machine often caused compatibility issues due to shared libraries and dependencies.



Example: A company running an e-commerce application had to dedicate entire physical servers to their web and database services. If traffic increased, they had to buy and configure more servers, leading to high costs and inefficiencies.



The Introduction of Virtual Machines (VMs) – A Partial Solution

With virtualization, multiple VMs could run on a single physical server using hypervisors like VMware, Hyper-V, or KVM. This improved resource utilization but introduced new challenges:

- **High Overhead** – Each VM had its own OS, consuming significant memory and processing power.
- **Complex Maintenance** – Managing updates, patches, and configurations across multiple VMs was still a manual process.
- **Slow Scaling** – Spinning up new VMs took time and required additional infrastructure provisioning.



Example: A company using VMs to host applications could isolate workloads, but managing hundreds of VMs manually became a bottleneck, requiring dedicated IT teams for provisioning and maintenance.



The Rise of Containers – A Step Forward, But Not Enough

Containers like **Docker** introduced lightweight, portable environments, bundling applications and their dependencies together. They solved many issues:

- **Improved Portability** – Containers ensured applications ran consistently across different environments (dev, test, production).
- **Better Resource Utilization** – Unlike VMs, containers shared the host OS, reducing overhead.
- **Faster Deployment** – Containers could be spun up in seconds, making deployment more efficient.

However, running containers at scale presented **new challenges**:

- **How to efficiently distribute containers across multiple servers?**
- **How to ensure failed containers restart automatically?**
- **How to handle networking and service discovery?**
- **How to scale applications dynamically based on demand?**

💡 *Example:* A tech startup adopted Docker to deploy microservices. Initially, managing a few containers was simple, but as their architecture grew to hundreds of containers, they faced difficulties in automation, scaling, and monitoring.

Before Kubernetes: How Did Google Manage Its Workloads?

Before Kubernetes was introduced in 2014, Google relied on **Borg**, an internal cluster management system that handled the deployment, scheduling, and resource allocation of applications across its massive infrastructure. **Borg was essentially the predecessor of Kubernetes**, providing Google with a powerful yet complex container orchestration solution.

How Did Google Operate Before Kubernetes?

Google has been running containerized workloads for over a decade. Long before the rise of Docker and Kubernetes, **everything at Google ran as a container**—but it was managed using Borg.

Borg played a crucial role in ensuring:

✅ **Scalability:** Efficiently managing thousands of workloads across Google's

global data centers.

✓ **High Availability:** Automatically rescheduling workloads in case of failures.

✓ **Resource Optimization:** Maximizing hardware utilization for better performance.

Challenges with Borg

While Borg was a robust system, it had several limitations:

⚠ **Complexity:** Required deep expertise, making it difficult for teams outside Google to use.

⚠ **Proprietary Nature:** It was an internal tool, unavailable to the public.

⚠ **Limited Flexibility:** Not designed to be a universally adaptable platform for enterprises.

The Birth of Kubernetes

Google took the best concepts from Borg and made them **open-source, flexible, and developer-friendly**—giving rise to Kubernetes. Unlike Borg, **Kubernetes was designed to be accessible to everyone**, enabling organizations worldwide to adopt container orchestration without the complexity of proprietary systems.

Today, Kubernetes has become the **industry standard** for managing containerized applications, bringing Google's expertise in large-scale infrastructure management to the global DevOps community.

🚀 **From Borg to Kubernetes—Google's innovation reshaped the future of cloud-native computing!**

Kubernetes: The Engine Powering Modern Cloud-Native Applications 🚀

In today's fast-paced world of software development, businesses need **speed, scalability, and reliability** to stay ahead. But managing applications efficiently, ensuring they scale on demand, and keeping them resilient against failures has always been a challenge.

This is where **Kubernetes** steps in—a game-changing technology that redefines how applications are deployed, managed, and scaled in the cloud.

Why Was Kubernetes Created?

Before Kubernetes, deploying and managing applications was a manual, complex, and error-prone process. Companies either relied on **virtual machines (VMs)**—which were heavy, slow, and resource-hungry—or struggled with **manual container management**, which lacked automation.

● The Problem:

- Applications had to be manually deployed and monitored.
- Scaling required human intervention, leading to inefficiencies.
- System failures could take hours or days to recover.
- Running workloads across multiple clouds was nearly impossible.

● The Solution – Kubernetes!

Kubernetes was designed to eliminate these problems by **automating application deployment, scaling, and recovery**. Instead of manually managing every instance,

Kubernetes ensures your application always runs in the desired state—with minimal human effort.

How Kubernetes Works (In Simple Terms!)

Think of Kubernetes as an **intelligent traffic control system** for applications. Just like a smart city manages roads, traffic signals, and public transport to keep everything flowing smoothly, Kubernetes ensures that applications are:

- ✓ **Deployed correctly** – No more manual setups; Kubernetes does it for you.
- ✓ **Always available** – If something fails, Kubernetes restarts it automatically.
- ✓ **Scalable** – It adds or removes resources based on real-time demand.
- ✓ **Portable & flexible** – Works on any cloud (AWS, Azure, GCP) or even on-premises.

The Core of Kubernetes: How It Manages Everything

Kubernetes runs applications inside **containers**, grouping them into logical units called **Pods**. These Pods run on **Worker Nodes**, which are controlled by a central **Control Plane**.

- ♦ **Control Plane (The Brain):** Makes all the decisions—where to run applications, how to scale them, and how to handle failures.
- ♦ **Worker Nodes (The Workforce):** Run the actual applications inside containers.
- ♦ **Pods (The Heartbeat):** The smallest deployable units, housing one or more containers.

♦ **Services & Load Balancing (The Network Manager):** Ensures traffic flows efficiently to the right application instances.

🚀 **Everything in Kubernetes is automated, self-healing, and built for high availability.**

Why Kubernetes is a Game-Changer for DevOps?

Kubernetes is more than just a tool—it's a shift in **how modern applications are built and managed**. It brings:

✅ **Microservices & Cloud-Native Architecture** – Perfect for scalable, modular applications.

✅ **CI/CD & DevOps Automation** – Simplifies deployment pipelines and version control.

✅ **Multi-Cloud & Hybrid Compatibility** – Deploy apps across any cloud provider.

✅ **Resilience & Self-Healing** – If something breaks, Kubernetes fixes it automatically.

☀️ **In short, Kubernetes isn't just a technology—it's the foundation of the future of DevOps and cloud computing.**

How Kubernetes Works in Today's World 🌍🚀

In the modern era of cloud computing and DevOps, **Kubernetes has become the backbone of scalable, resilient, and automated application management**. From startups to tech giants, companies leverage Kubernetes to ensure applications run seamlessly across cloud and on-premise environments.

Let's break down how Kubernetes powers the modern world of software deployment and infrastructure management.

1 Kubernetes as the Operating System of the Cloud

Think of Kubernetes as the **operating system for cloud-native applications**. Just like an OS manages hardware resources for applications, Kubernetes manages infrastructure resources for **containerized workloads**.

✅ **Deploy Anywhere** – Kubernetes abstracts the underlying infrastructure, allowing applications to run **on any cloud provider (AWS, Azure, GCP) or even on-premises**.

✅ **Standardized Management** – Developers can define deployment configurations in YAML, and Kubernetes ensures the app runs exactly as intended.

✅ **Multi-Cloud & Hybrid Cloud** – Enterprises run Kubernetes clusters **across multiple clouds** for high availability and disaster recovery.

How Kubernetes Automates Application Lifecycle?

Kubernetes automates **everything** related to application management—from deployment to scaling and self-healing. Here's how:

Deployment & Orchestration:

- Developers package applications into **Docker containers** and define how they should run in Kubernetes YAML files.
- Kubernetes **deploys** these containers as **Pods** across multiple servers (**Worker Nodes**).
- Using **ReplicaSets**, Kubernetes ensures the right number of instances are always running.

Auto-Scaling Based on Demand:

- Kubernetes dynamically **scales applications** up or down based on traffic.
- Uses **Horizontal Pod Autoscaler (HPA)** to add more instances when demand rises.
- Uses **Vertical Pod Autoscaler (VPA)** to allocate more CPU/RAM to running workloads if needed.

Self-Healing & High Availability:

- If a container crashes, **Kubernetes automatically restarts it**.
- If a Node fails, **Kubernetes reschedules workloads** onto a healthy Node.
- **Load Balancing** ensures incoming traffic is distributed across healthy application instances.

Rolling Updates & Zero Downtime Deployments:

-
- Kubernetes updates applications **without downtime** using **Rolling Updates**.
 - If a new deployment has issues, it rolls back automatically.
 - Canary deployments allow testing new versions with a small audience before a full rollout.

3 How Kubernetes Powers Modern Applications?

- ◆ **Netflix, Spotify, and YouTube** – Scale streaming services automatically based on user demand.
- ◆ **E-commerce platforms like Amazon & Shopify** – Handle peak sales events (Black Friday, Diwali sales) seamlessly.
- ◆ **FinTech & Banking** – Ensure security and compliance while running containerized applications across multiple clouds.
- ◆ **AI & Machine Learning** – Run AI/ML workloads efficiently using Kubernetes clusters with GPU acceleration.
- ◆ **IoT & Edge Computing** – Deploy Kubernetes clusters on edge devices for real-time processing.

4 Kubernetes in DevOps & CI/CD Pipelines

Kubernetes is **deeply integrated into DevOps workflows**, making software delivery faster and more efficient.

✓ **CI/CD Pipelines** – Jenkins, GitHub Actions, or GitLab CI/CD automate deployments into Kubernetes.

✓ **GitOps Approach** – Tools like ArgoCD and Flux ensure that infrastructure

matches Git repository configurations.

✅ **Observability & Monitoring** – Prometheus, Grafana, and Loki provide real-time insights into application health and performance.

✅ **Security & Compliance** – Kubernetes integrates with security tools (Trivy, Falco, Kyverno) to ensure compliance and vulnerability scanning.

Why Do We Need Kubernetes When We Already Have Docker Swarm? 🤔

Docker Swarm and Kubernetes both manage containerized applications, but **Kubernetes became the industry standard for container orchestration**. While Docker Swarm is simpler, it **lacks the advanced automation, scalability, and flexibility that Kubernetes provides**.

Let's dive into why Kubernetes is the preferred choice over Docker Swarm in today's world.

1 Kubernetes vs. Docker Swarm – The Core Differences

- ♦ **Docker Swarm** is a lightweight **container orchestration tool** that comes built-in with Docker. It allows you to deploy and manage containers across multiple hosts.
- ♦ **Kubernetes** is a **full-fledged container orchestration platform** with built-in **scalability, automation, and enterprise-grade features**.

While both are used for managing containerized applications, **Kubernetes offers more flexibility, automation, and robustness.**

2 Key Reasons Why Kubernetes is Preferred Over Docker Swarm

Advanced Scaling & Load Balancing

✓ Docker Swarm: **Basic scaling** – manually scale services with `docker service scale`

✓ Kubernetes: **Automated scaling** – **Horizontal Pod Autoscaler (HPA)** scales services automatically based on CPU/memory usage.

♦ **Example:** If traffic spikes on an e-commerce app, **Kubernetes automatically adds more instances**, while Docker Swarm requires manual intervention.

Self-Healing & Fault Tolerance

✓ Docker Swarm: **Recreates failed containers**, but lacks deep health checks.

✓ Kubernetes: **Self-healing mechanism** – if a container fails, **it restarts automatically** and reschedules workloads if a node crashes.

♦ **Example:** If a web server running on Docker Swarm crashes, **you must manually redeploy** it. In Kubernetes, it restarts **automatically** without downtime.



Rolling Updates & Canary Deployments



Docker Swarm: **Performs rolling updates but lacks rollback automation.**



Kubernetes: **Rolling updates with rollback support** – allows smooth updates without downtime.

- ♦ **Example:** If a new version of an application breaks in Swarm, **you need manual intervention** to roll back. Kubernetes **automatically detects issues and rolls back** to a stable version.



Security & Access Control



Docker Swarm: **Basic security** – uses TLS encryption but lacks Role-Based Access Control (RBAC).



Kubernetes: **Enterprise-grade security** – RBAC, Network Policies, Service Mesh integration (Istio, Linkerd).

- ♦ **Example:** In a multi-team enterprise environment, Kubernetes **restricts access to resources** based on user roles, whereas Swarm lacks fine-grained access control.



Multi-Cloud & Hybrid Deployments



Docker Swarm: **Limited support** for multi-cloud and hybrid environments.



Kubernetes: **Designed for multi-cloud** – easily runs on AWS, Azure, GCP, on-premises, and even at the edge.

♦ **Example:** If a business wants to **run applications across AWS and Azure**, Kubernetes **natively supports multi-cloud environments**, while Swarm has limitations.

3 When Would You Use Docker Swarm Over Kubernetes?


Although Kubernetes is more powerful, **Docker Swarm is still useful in some cases:**

- ✓ **Smaller teams & projects** – Swarm is simpler and faster to set up.
- ✓ **Rapid prototyping** – If you just need quick container orchestration without complex configurations.
- ✓ **Less overhead** – Swarm uses fewer resources, making it ideal for lightweight applications.

4 Final Verdict – Why Kubernetes Won the Battle?

Kubernetes is the preferred choice for **modern DevOps and enterprise applications** because of its:

- ✓ **Automation** – Self-healing, auto-scaling, and zero-downtime deployments.
- ✓ **Security** – RBAC, Network Policies, and Service Mesh support.
- ✓ **Flexibility** – Runs on any cloud, on-premises, or even at the edge.
- ✓ **Robust ecosystem** – Integrates with Helm, GitOps, and monitoring tools like Prometheus/Grafana.

 **For production-grade, scalable, and mission-critical applications, Kubernetes is the clear winner!**

How Do People Create a Kubernetes Cluster?


Setting up a **Kubernetes cluster** is the first step in deploying and managing containerized applications at scale. Depending on the use case, people create Kubernetes clusters using various methods, from local setups for learning to production-grade deployments in the cloud.

Let's explore the different ways Kubernetes clusters are created today!

1 Local Kubernetes Clusters for Learning & Development

For developers and learners, lightweight Kubernetes distributions are used to **test applications locally** before deploying to production.

- ♦ **Minikube** – Runs a single-node Kubernetes cluster on a local machine.
- ♦ **Kind (Kubernetes in Docker)** – Uses Docker containers to simulate a Kubernetes cluster.
- ♦ **MicroK8s** – A minimal Kubernetes distribution for local testing.

 **Use Case:** Ideal for developers who want to experiment with Kubernetes without setting up cloud infrastructure.

2 Self-Managed Kubernetes Clusters

For organizations that want full control over their infrastructure, Kubernetes can be installed on **bare-metal servers** or **virtual machines** using various tools:

- ♦ **Kubeadm** – A command-line tool to set up a cluster manually.
- ♦ **Kops (Kubernetes Operations)** – Automates cluster deployment on AWS.
- ♦ **Kubectrl** – A command-line tool to interact with Kubernetes clusters.

💡 **Use Case:** Used by teams that need **custom configurations, network policies, and full control** over the cluster setup.

3 Managed Kubernetes Services (Cloud-Based) ☁️

Most enterprises today **prefer cloud-managed Kubernetes services** for ease of deployment, scalability, and maintenance. Popular cloud providers offer fully managed Kubernetes clusters:

- ♦ **Amazon EKS (Elastic Kubernetes Service)** – Kubernetes on AWS
- ♦ **Google Kubernetes Engine (GKE)** – Kubernetes on Google Cloud
- ♦ **Azure Kubernetes Service (AKS)** – Kubernetes on Microsoft Azure
- ♦ **DigitalOcean Kubernetes (DOKS)** – Kubernetes on DigitalOcean

💡 **Use Case:** Best for businesses that need **high availability, auto-scaling, and built-in security** without managing the control plane manually.

4 Kubernetes in Hybrid & Multi-Cloud Environments 🌐

Organizations that operate across multiple clouds or on-premises use **hybrid cloud solutions** for Kubernetes:

- ♦ **Rancher** – Multi-cluster Kubernetes management.
- ♦ **OpenShift** – Enterprise Kubernetes by Red Hat.
- ♦ **VMware Tanzu** – Kubernetes for VMware environments.

💡 **Use Case:** Used by enterprises that need **Kubernetes across different cloud providers or on-premises data centers**.

Choosing the Right Way to Create a Kubernetes Cluster

- ✓ **For beginners & local testing:** Minikube, Kind, or MicroK8s
- ✓ **For full control:** Kubeadm, Kops, or self-hosted Kubernetes
- ✓ **For cloud-native applications:** AWS EKS, GKE, AKS, or other managed services
- ✓ **For hybrid/multi-cloud deployments:** Rancher, OpenShift, or VMware Tanzu

💡 **The choice depends on scalability, ease of management, and operational needs.**



Why Kubernetes is Essential for DevOps?

✓ **Container Orchestration** – Automates deployment, scaling, and management of containerized applications.

✓ **Infrastructure as Code (IaC)** – Works seamlessly with tools like Terraform and Helm for declarative infrastructure management.

✓ **Continuous Deployment & Delivery (CI/CD)** – Integrates with Jenkins, ArgoCD, and GitOps workflows for automated deployments.

✓ **Scalability & High Availability** – Ensures smooth traffic distribution and fault tolerance across multiple nodes.

✓ **Multi-Cloud & Hybrid Support** – Works across on-prem, cloud, and hybrid environments with providers like AWS (EKS), GCP (GKE), and Azure (AKS).

✓ **Security & Policy Enforcement** – Implements RBAC, network policies, and container runtime security for safe deployments.

🌟 Wrapping Up: The Beginning of an Exciting Journey!

We've just scratched the surface of **Kubernetes**, but this is only the beginning! 🚀

Through this document, we explored how the world worked before Kubernetes, why it became essential, how it operates today, and why it stands out among container orchestration tools. Kubernetes isn't just a technology—it's the backbone of **modern DevOps**, enabling automation, scalability, and efficiency like never before.

But this is just **Day 1** of our **Daily Kubernetes Tasks Series**! From here, we will dive deeper into practical implementations, real-world use cases, hands-on exercises, and best practices to master Kubernetes step by step. **Every day, a new concept, a new challenge, and a new learning experience!** 💡

📌 Stay Connected & Keep Learning!

Make sure to **follow me** so you don't miss out on the daily Kubernetes tasks and insights. 🚀 Let's grow together and build expertise in Kubernetes, one day at a time!

🔗 **Have suggestions or topics you'd love to see covered? Drop them in the comments!**

See you in the next post! **The journey has just begun.** 💪 🔥

#Kubernetes #DevOps #CloudNative #ContainerOrchestration
#100DaysOfKubernetes #LearningJourney