SRE TRAINING (DAY 14) - KUBERNETES

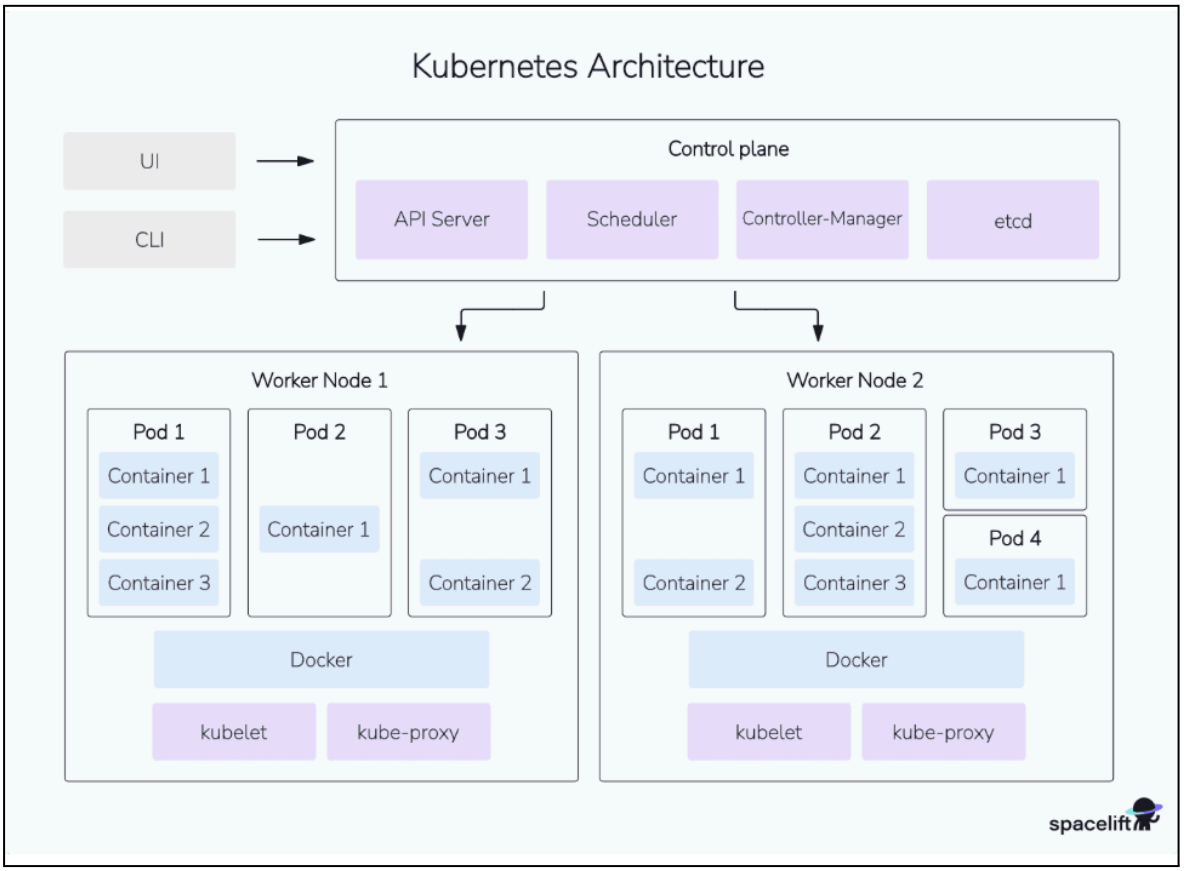
What is Kubernetes?

Kubernetes (K8s) is an **open-source** platform for **automating deployment, scaling, and management** of containerized applications.

Before Kubernetes, applications were deployed on virtual machines (VMs) or bare-metal servers. Containers (like Docker) improved efficiency, but managing multiple containers across different machines manually became complex.

Kubernetes solves this problem by:
✅ Automating container deployment & scaling.
✅ Managing networking & storage for containers.
✅ Ensuring high availability and fault tolerance.
✅ Rolling out updates & rollbacks easily

Kubernetes Architecture



1️⃣ Control Plane (Master Node)

The brain of Kubernetes that manages the cluster. It includes:

- **API Server**: The entry point for all Kubernetes commands (kubectl, Minikube, etc.).
- **Scheduler**: Assigns workloads (Pods) to worker nodes based on resource availability.
- **Controller Manager**: Monitors the cluster and ensures desired states are maintained.
- **etcd**: A distributed key-value store that keeps cluster state.

2️⃣ Worker Nodes

These are the **machines where your applications run**. Each node contains:

- **Kubelet**: A small agent that ensures Pods are running.
- **Container Runtime**: Runs the actual containers (e.g., Docker).
- **Kube Proxy**: Handles networking within the cluster.

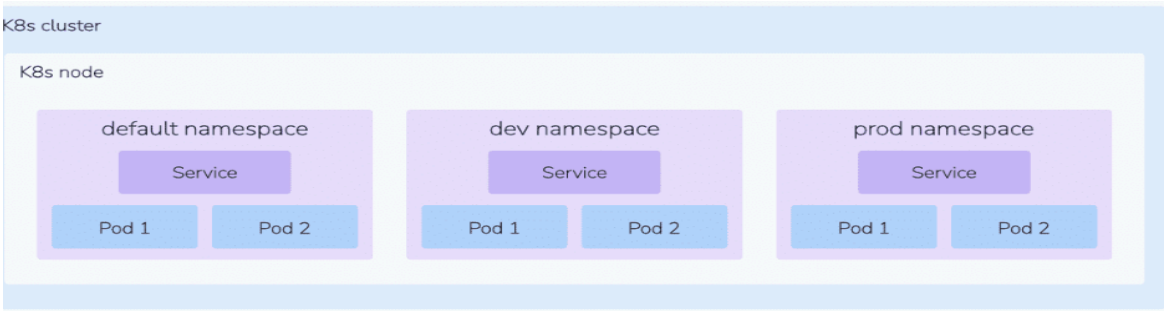
Basic terms and concepts

1. Kubernetes Nodes

Nodes represent the physical machines that form your Kubernetes cluster. They run the containers you create. Kubernetes tracks the status of your nodes and exposes each one as an object.

2. Namespaces

Kubernetes **namespaces** isolate different groups of resources. They avoid name collisions by scoping the visibility of your resources.



3. Pods

Pods are the fundamental compute unit in Kubernetes. A Pod is analogous to a container but with some key differences. Pods can contain multiple containers, each of which share a context. The entire Pod will always be scheduled onto the same node. The containers within a Pod are tightly coupled so you should create a new Pod for each distinct part of your application, such as its API and database.

4. Minikube

Minikube is a **lightweight Kubernetes tool** that lets you run Kubernetes on a single machine (your laptop or VM). It's used for **local development and testing** without needing a full cloud-based Kubernetes cluster.

5. Helm (Kubernetes Package Manager)

Helm is like a package manager (**apt, yum, npm**) but for **Kubernetes applications**. It simplifies installing, updating, and managing complex applications inside Kubernetes.

How Kubernetes Works Together

1️⃣ **Developer creates a Pod or Deployment** (via YAML or `kubectl`).

2️⃣ **Kubernetes schedules it** to a **Worker Node**.

3️⃣ **Kubelet on the Node** runs the container using Docker/containerd.

4️⃣ **Kube Proxy** ensures networking works inside the cluster.

5️⃣ **Services expose the application** (internally or externally).

6️⃣ **Helm** helps manage large-scale apps easily.

INSTALLATION

1. Installing Kubernetes Tools (kubectl, Minikube, Docker, Helm)

```
root@RheaAlisha:/# helm version
version.BuildInfo{Version:"v3.17.1", GitCommit:"980d8ac1939e39138101364400756af2bdee1da5", GitTreeState:"clean", GoVe
on:"go1.23.5"}
root@RheaAlisha:/# docker --version
Docker version 28.0.1, build 068a01e
root@RheaAlisha:/# kubectl version
Client Version: v1.32.2
Kustomize Version: v5.5.0
The connection to the server 127.0.0.1:32771 was refused - did you specify the right host or port?
root@RheaAlisha:/# minikube version
minikube version: v1.35.0
commit: dd5d320e41b5451cdf3c01891bc4e13d189586ed-dirty
```

- **Updated the system (apt-get update)**
- **Installed dependencies (apt-get install -y apt-transport-https ca-certificates curl software-properties-common gnupg2 conntrack)**
- **Installed Docker, added the current user to the docker group, and verified it with docker run hello-world.**
- **Downloaded kubectl and moved it to /usr/local/bin/ so it could be used globally.**
- **Installed Minikube and Helm for Kubernetes.**
- **Configured ~/.bashrc with Kubernetes aliases.**

2. Installing Docker

- **Any old Docker installation was removed to avoid conflicts.**
- **The official Docker repository was added.**
- **Docker was installed along with its dependencies.**
- **The user was added to the `docker` group for permission to run Docker without `sudo`.**
- **A test container (`hello-world`) was executed to verify the installation.**

```
root@RheaAlisha:/# sudo docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

3. Configured Docker to Start on WSL Startup

```
echo '# Start Docker daemon automatically when WSL starts
if [ -z "$(ps -ef | grep dockerd | grep -v grep)" ]; then
    sudo service docker start
fi' >> ~/.bashrc
```

4. Added Kubernetes Command Aliases

Aliases were set up to simplify frequently used Kubernetes commands.

```
echo '# Kubernetes aliases
alias k="kubectl"
alias kgp="kubectl get pods"
alias kgs="kubectl get services"
alias kgd="kubectl get deployments"
alias kgn="kubectl get nodes"
alias kga="kubectl get all"' >> ~/.bashrc
```

RUNNING THE FIRST APPLICATION

For this we ran a kubernetes script. This script creates and deploys a minimal Kubernetes application using Flask, Docker, and Minikube.

Key Steps in the Script

1. Setting Up the Project Structure

- **Created a directory: `~/mini-k8s-demo/{app,k8s}`**
- **Organized files into:**
  - `app/` → **Application code**
  - `k8s/` → **Kubernetes deployment files**

2. Creating a Flask Web Application

- **Developed a Python Flask application (`app.py`)**
- **Created a `requirements.txt` file with Flask dependency.**

```python
# Simple Flask application for Kubernetes demonstration
from flask import Flask, jsonify
import os
import socket
import datetime

# Initialize Flask application
app = Flask(__name__)

# Read configuration from environment variables (will be set via Kubernetes ConfigMap)
APP_NAME = os.environ.get('APP_NAME', 'mini-k8s-demo')
APP_VERSION = os.environ.get('APP_VERSION', '1.0.0')

# Track request count to demonstrate statelessness in Kubernetes
request_count = 0

@app.route('/')
def index():
    """Main page showing container/pod information"""
    global request_count
    request_count += 1
```

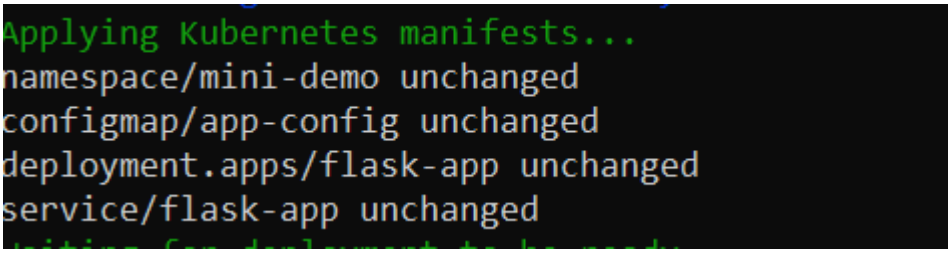
3. Creating a Docker Image for the Application

- **Created a `Dockerfile` with:**
  - **Python 3.9 slim as the base image**
  - **Installed dependencies (`pip install -r requirements.txt`)**
  - **Exposed port 5000**
  - **Set the default command to run Flask (`CMD ["python", "app.py"]`)**

```
Configuring Docker to use Minikube's Docker daemon...
Building Docker image...
[+] Building 10.5s (11/11) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 929B
 => [internal] load metadata for docker.io/library/python:3.9-slim
 => [auth] library/python:pull token for registry-1.docker.io
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/5] FROM docker.io/library/python:3.9-slim@sha256:d1fd807555208707ec95b284af
 => [internal] load build context
 => => transferring context: 1.99kB
 => CACHED [2/5] WORKDIR /app
 => [3/5] COPY requirements.txt .
 => [4/5] RUN pip install --no-cache-dir -r requirements.txt
 => [5/5] COPY app.py .
 => exporting to image
 => => exporting layers
 => => writing image sha256:dae1d470129c223728becc132def6c3cd41e6113fea80d8ba650fc
 => => naming to docker.io/library/mini-k8s-demo:latest
```

4. Creating Kubernetes Deployment Files

- **Namespace (`namespace.yaml`)**
  - **Created a new namespace mini-demo to isolate the application.**
- **ConfigMap (`configmap.yaml`)**
  - **Stored environment variables APP_NAME and APP_VERSION.**
- **Deployment (`deployment.yaml`)**
  - **Created a ReplicaSet with 2 Pods using `mini-k8s-demo:latest` image.**
  - **Set resource limits (`cpu: 200m`, `memory: 128Mi`).**
  - **Defined liveness and readiness probes (`/api/health`).**
- **Service (`service.yaml`)**
  - **Created a NodePort service to expose the app at `http://<minikube-ip>:30080`.**



5. Automating Deployment with a Shell Script (`deploy.sh`)

- **Checked if Minikube was running and started it if needed.**
- **Configured Docker to use Minikube's internal registry.**
- **Built the Docker image (`mini-k8s-demo:latest`).**
- **Applied all Kubernetes manifests (`kubectl apply -f k8s/`).**
- **Waited for deployment to complete (`kubectl rollout status`).**
- **Printed the Minikube service URL for accessing the app.**