**Advantages of using the Kubernetes over Docker swam**

Docker has its own product/tool to manage, control, schedule, monitor, scale up and down the containers in different Docker hosts is called Docker swam.

Docker swam is also a Docker container orchestration tool, that is part of the Docker platform.

It helps you manage and coordinate the deployment, scaling, and management of containerized applications across multiple Docker hosts.

Docker Swarm and Kubernetes are both container orchestration tools, but Kubernetes is generally more feature-rich and widely adopted.

**Disadvantages of Using Docker swam over Kubernetes:**

* Docker doesn't handle very large, complex deployments as efficiently as Kubernetes
* Docker Swarm's storage management capabilities are less flexible then Kubernetes.
* Kubernetes provides more advanced options for managing persistent storage.
* Kubernetes has a much larger and more active community compared to Docker swam.
* Docker Swarm’s networking is simpler & less powerful then Kubernetes.
* Kubernetes has built-in Horizontal Pod Auto-scaling (HPA) and Vertical Pod Auto-scaling (VPA), allowing applications to scale based on CPU, memory, or custom metrics.
* Kubernetes is an open-source tool that support any containerization tools like Docker, podman, containerd, LXC and so on…
* No proper failover methods for Docker host. That means if Docker host fails then ower applications which are present in the container are go down.
* No proper auto-scaling feature for the Docker Swam.
* Docker swam supports smaller deployments.

**Difference between the Docker swam and Kubernetes:**

| **Feature** | **Docker Swarm** | **Kubernetes** |
| --- | --- | --- |
| **1. Scalability** | Supports scaling but limited for large, complex applications | Highly scalable, designed for large-scale enterprise deployments |
| **2. Ease of Use** | Easier to set up and manage | More complex setup with steep learning curve |
| **3. Networking** | Uses simple overlay networking | Advanced networking with service discovery, ingress, and DNS integration |
| **4. Auto-Scaling** | Manual scaling required | Supports automatic scaling based on resource usage |
| **5. Load Balancing** | Built-in basic load balancing | Advanced load balancing with external traffic management |
| **6. Storage & Persistence** | Limited persistent storage options | Strong support for persistent volumes and cloud storage integrations |

**Advantages of Using Kubernetes:**

1. It is advanced then the Docker swam.
2. A powerful auto-scaling capacity form complex deployments.
3. Kubernetes is an open-source tool that support any containerization tools like Docker, podman, containerd, LXC and so on…
4. Kubernetes has built-in Horizontal Pod Auto-scaling (HPA) and Vertical Pod Auto-scaling (VPA), allowing applications to scale based on CPU, memory, or custom metrics.
5. It has proper failover method for Docker host:

If any node in the cluster gets failed, then the master node (control panel) automatically detects it and reschedules the pods which present in the failed nods to a healthy working node.

1. Proper auto-scaling feature for the pods in the worker nodes.
2. Provides **Role-Based Access Control (RBAC), Network Policies, and Pod Security Policies** for better security.

**Conclusion**:

* Choose Docker Swarm for simple, small-scale deployments with quick setup.
* Choose Kubernetes for large-scale, production-grade applications with better automation, scalability, and cloud integration

**Service in Kubernetes:**

A **Service** in Kubernetes is an **abstraction that provides network access** to a set of Pods. Since Pods are **ephemeral** (can be created and destroyed dynamically).

That means services enable network access to a set of Pods, providing stable networking and load balancing, even as Pods are created, destroyed, or rescheduled. Services are a core concept in Kubernetes for enabling communication between different components of an application.

* We know that each pod in the Kubernetes assigned/have its own public IP, using this IP address we can access application and connect to DB (which is present in other pod). But one of the main disadvantage using IP is, when pod gets failed or destroyed, with the auto-scaling concept other pod will create with same copy of application with different IP address.
* As a result of changing of IP address of pods, we cannot access the application continuously without down time.
* That means, the IP is tied up with pods, if pod dies IP which is assigned to it also dies.
* So to overcome this above disadvantage we use the **“Service”** in Kubernetes of worker nodes. This services are not tied up with pods.

Types of Services:

1. ClusterIP (default).
2. Node port.
3. Load balancer.
4. Ingress
5. **ClusterIP:**

* This is the default Service type.
* It exposes the Service on an internal IP address within the cluster.
* ClusterIP Services are only accessible from within the cluster.
* This service cannot be accessed from outside the cluster.
* **Use Case:** They are primarily used for internal communication between applications

1. **Node port:**

* Exposes the Service on a static port on each **Node** in the cluster.
* NodePort Services allow external traffic to access the application via NodeIP:NodePort.
* The port range is **30000-32767**.
* It combines the functionality of ClusterIP, meaning it also has an internal cluster IP
* **Use Case**: Simple external access for development/testing.

1. **Load Balancer:**

* This type exposes the Service externally using a cloud provider's load balancer.
* It automatically provisions a load balancer in the cloud environment and assigns an external IP address to the Service.
* LoadBalancer Services are commonly used for exposing applications to the public internet.
* **Use Case:** Production environments on cloud platforms (AWS, GCP, and Azure).

1. **Ingress:**

* **It is used to manage external access** to services within a cluster.
* It allows you to define **HTTP(S) routing rules** to expose multiple services under a **single external IP** (using a reverse proxy).
* It provides routing rules to direct external traffic to different services based on hostnames or paths.
* Instead of using multiple **NodePort** or **LoadBalancer** services, **Ingress** provides a **centralized entry point** for traffic, handling **routing, SSL termination, and virtual hosting**.

Let’s we work with the Load balancer service in the cluster:

**Declarative way: (using YAML file)**

**Step1:** Create a .yaml file (for deploying a pod) by configuring it in such a way that it pulls the Nginx image from Docker Hub.

**Step2:** By using .yaml file create a pod.

**Step3:** Create one more .yaml file (for service) to expose this pod to the outside of the cluster.

(Or)

We can do the above task in simple way using imperative (CLI):

**Imperative way: (using bash, Azure CLI, PowerShell )**

**Command:** kubectl run <pod-name> --image <image-name>

Ex: kubectl run my-first-pod –image nginx.

**Command:** kubectl expose <pod-name> --type=LoadBalancer --name=<any-name>

Ex: kubectl expose my-first-pod --type=LoadBalancer --port=80--name my-first-service.

**Declarative way:**

**Step1:** Create a deployment.yaml file.

apiVersion: apps/v1

kind: Deployment

metadata:

name: my-deployment

spec:

replicas: 2 # Number of pods to run

selector:

matchLabels:

app: my-app # Select pods with this label

template: # Pod template

metadata:

labels:

app: my-app # Labels for the pods

spec: # Pod specification

containers:

- name: my-container

image: nginx:latest # Container image

ports:

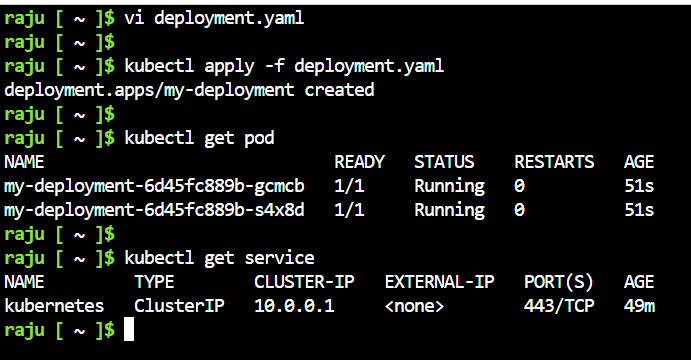
- containerPort: 80

Fig: deployment.yaml file.

**Step2**: Create the pods by deploying the .yaml file.

**Command:** kubctl apply –f <name-of the yaml file>

**EX:** kubectl apply -f deployment.yaml



**Step3:** Create one more yaml file to build the Load balancer service.

apiVersion: v1

kind: Service

metadata:

name: my-loadbalancer-service

spec:

type: LoadBalancer

ports:

- port: 80

targetPort: 80

protocol: TCP

selector:

app: my-app # Replace with your app's label

Fig: Service.yaml file (to create Loadbalancer service).

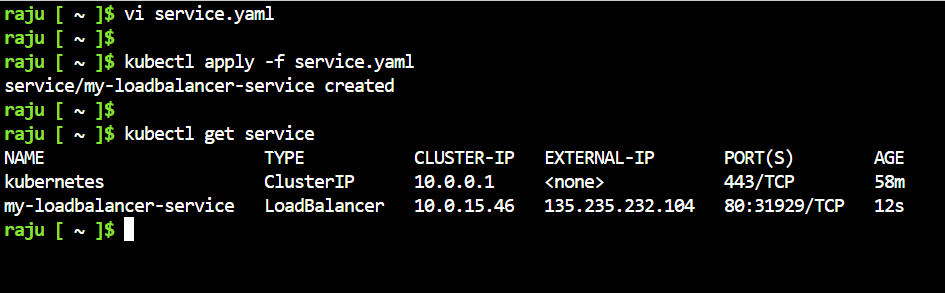
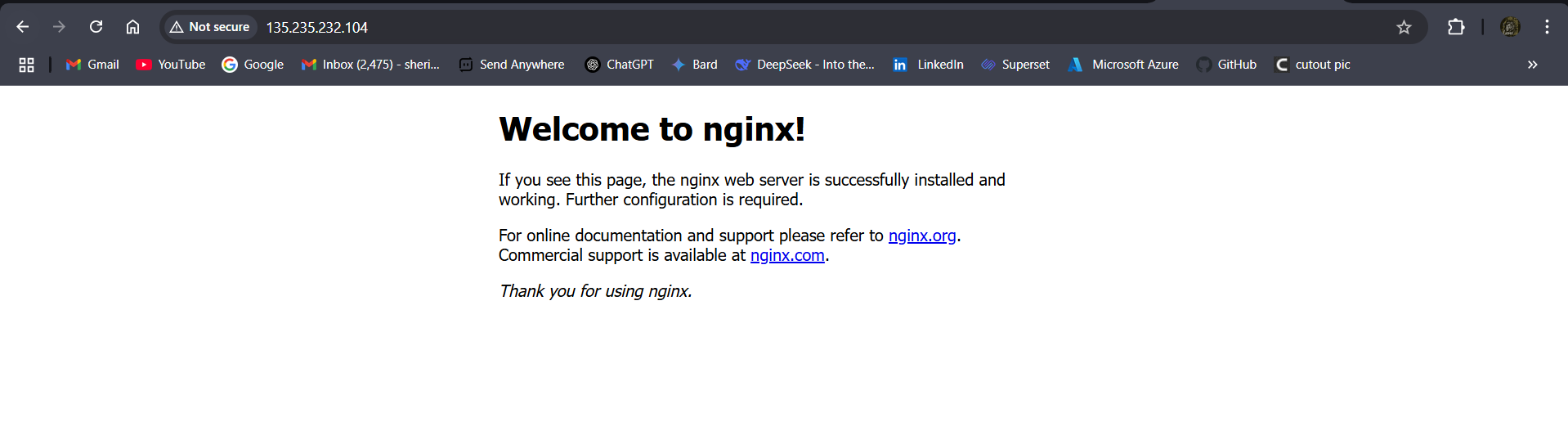


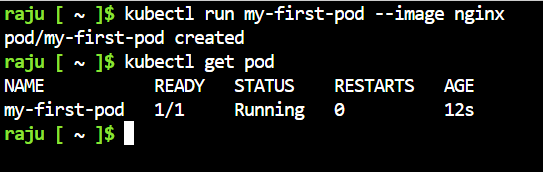
Fig: LoadBalancer service is created successfully.

By using the EXTERNAL-IP we can access this pod from the outside the cluster.

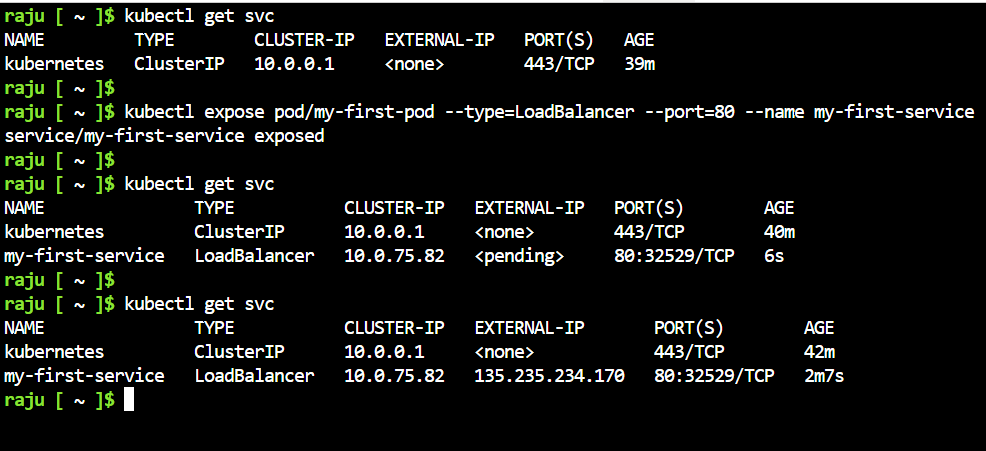


**Imperative (CLI) way:**

**Step1:** Create a pod using **Command:** kubectl run <pod-name> --image <image-name>

****

**Step2:** Create a service using **Command:** kubectl expose <pod-name> --type=LoadBalancer --port=80--name=<any-name>



By using the EXTERNAL-IP we can browse the Nginx page.

