**Docker Swarm Setup and Scaling**

**Lecture Notes**

**Introduction**

Docker Swarm is a native clustering and orchestration tool for Docker that allows you to manage a group of machines as a single unit. It enables high availability, scalability, and fault tolerance for containerized applications.

**Introduction to Container Orchestration**

**What is Container Orchestration?**

Container orchestration is the process of automating container deployment, scaling, networking, and management. It ensures that applications run efficiently in distributed environments.

**Example Scenario**

Imagine a large e-commerce platform handling millions of users. Instead of manually deploying and managing containers for different services (e.g., product catalog, payment processing, user authentication), a container orchestrator ensures:

* Automatic scaling based on demand.
* Self-healing of failed containers.
* Efficient distribution of workloads across multiple servers.

Conclusion:

* If traffic increases, Kubernetes or Swarm can **scale up** by adding more containers.
* If a node fails, these orchestration tools **self-heal** by shifting workloads to healthy nodes.
* They also handle **rolling updates** and **service discovery** to keep applications running smoothly.

**2️. Traditional Deployment vs. Container Orchestration**

**Traditional Deployment Process**

Before containers, applications were deployed in two ways:

1. **Physical Servers:** Each application had dedicated hardware, leading to poor resource utilization.
2. **Virtual Machines (VMs):** Allowed running multiple applications on one server, but each VM had a full OS, consuming significant resources.

**Why Move to Containers?**

* Containers share the host OS, reducing overhead.
* Faster startup time compared to VMs.
* Easily portable across different environments.

However, managing containers manually at scale is complex—this is where container orchestration comes in.

**Why Docker Swarm?**

**What is Docker Swarm?**

Docker Swarm is Docker’s native container orchestration tool. It allows managing multiple Docker hosts as a single cluster.

**Advantages of Docker Swarm**

* **Native to Docker:** Works seamlessly with existing Docker tools.
* **Easy Setup:** Requires fewer steps than alternatives like Kubernetes.
* **Integrated Load Balancing:** Automatically distributes network traffic among services.
* **Declarative Service Model:** Allows defining desired states (e.g., replica count, networks).

**Why Not Kubernetes?**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Docker Swarm** | **Kubernetes** |
| Complexity | Simple setup | More complex |
| Scaling | Manual & basic auto-scaling | Advanced auto-scaling |
| Load Balancing | Built-in | External solutions (Ingress, Service Mesh) |
| Ecosystem | Limited ecosystem | Large ecosystem |

* Kubernetes is **better for large-scale** enterprise applications.
* Swarm is **ideal for simpler, Docker-native deployments.**

**Alternatives to Docker Swarm**

1. **Kubernetes** - Most popular orchestration tool, powerful but complex.
2. **Amazon ECS (Elastic Container Service)** - Managed by AWS, integrates well with AWS infrastructure.
3. **Nomad by HashiCorp** - Lightweight orchestrator with support for both containers and non-container workloads.
4. **Apache Mesos** - Suitable for large-scale cluster management.

Each tool has different use cases depending on complexity, infrastructure, and scalability requirements.

**1️. Setting Up Ubuntu Servers**

We need three Ubuntu machines:

* **1 Manager Node** (Coordinates and schedules tasks)
* **2 Worker Nodes** (Execute assigned tasks)

**Setting Hostnames**

Each node should have a unique hostname for identification.

**Command**

sudo hostnamectl set-hostname <hostname>

**Example**

On the manager node:

sudo hostnamectl set-hostname manager-node

On worker nodes:

sudo hostnamectl set-hostname worker1

sudo hostnamectl set-hostname worker2

**Verify Hostname**

hostname

Expected output:

manager-node

**2️. Install Docker on All Nodes**

Run these commands on **ALL NODES** (Manager + Workers).

**Install Dependencies**

sudo apt update && sudo apt install -y apt-transport-https ca-certificates curl software-properties-common

**Explanation:**

* apt update: Updates the package list to ensure we get the latest versions.
* apt install -y ...: Installs essential dependencies:
  + apt-transport-https: Allows downloading packages over HTTPS.
  + ca-certificates: Ensures secure HTTPS communication.
  + curl: A tool for downloading files.
  + software-properties-common: Adds repository management tools.

**Add Docker GPG Key**

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

**Explanation:**

* Downloads the **GPG key** used to verify Docker packages.
* curl -fsSL: Fetches the key securely.
* apt-key add -: Adds the key to the system’s trusted keys.

**Add Docker Repository**

sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable"

**Explanation:**

* Adds the **official Docker repository** to the system.
* $(lsb\_release -cs): Inserts the current Ubuntu version (e.g., focal for Ubuntu 20.04).
* stable: Ensures we get the latest stable version of Docker.

**Install Docker**

sudo apt update && sudo apt install -y docker-ce docker-ce-cli containerd.io

**Explanation:**

* docker-ce: Installs Docker Community Edition (CE).
* docker-ce-cli: Installs Docker command-line interface (CLI).
* containerd.io: Installs the container runtime.

**Enable and Start Docker**

sudo systemctl enable docker

sudo systemctl start docker

**Explanation:**

* systemctl enable docker: Ensures Docker starts automatically on boot.
* systemctl start docker: Starts the Docker service immediately.

**Verify Installation**

docker --version

**Expected Output**

Docker version 24.0.5, build 123abc456

**Explanation:**

* Checks if Docker is installed correctly.
* Displays the installed version and build details.

**3️. Initializing Docker Swarm**

Run **only on the manager node**.

docker swarm init --advertise-addr <MANAGER-IP>

**Why This Command?**

* Initializes a new Swarm cluster.
* --advertise-addr specifies the manager node's IP for communication.

Expected Output:

Swarm initialized: current node (abc123def) is now a manager.

**4️. Joining Worker Nodes**

Each worker node must join the swarm using a unique token generated by the manager.

**Command**

docker swarm join --token SWMTKN-1-xyz123 <MANAGER-IP>:2377

**Why This Command?**

* Uses a secure token to authenticate worker nodes.
* Connects workers to the swarm cluster.

Verify nodes in Swarm:

docker node ls

Expected Output:

ID HOSTNAME STATUS AVAILABILITY MANAGER STATUS

abc123def manager-node Ready Active Leader

xyz456ghi worker1 Ready Active

lmn789jkl worker2 Ready Active

**5️. Deploying an Nginx Service**

docker service create --name my-nginx --replicas 3 -p 80:80 nginx

**Why This Command?**

* --name my-nginx: Names the service "my-nginx".
* --replicas 3: Runs three instances of Nginx across available nodes.
* -p 80:80: Exposes the service on port 80.

**Verify Deployment**

docker service ls

Expected Output:

ID NAME MODE REPLICAS IMAGE

abcd1234 my-nginx replicated 3/3 nginx:latest

**6️. Checking Container Distribution**

docker service ps my-nginx

Expected Output:

ID NAME IMAGE NODE DESIRED STATE CURRENT STATE

xyz123 my-nginx.1 nginx:latest worker1 Running Running 1 min ago

abc456 my-nginx.2 nginx:latest worker2 Running Running 1 min ago

mno789 my-nginx.3 nginx:latest manager Running Running 1 min ago

**7️. Testing Traffic Distribution**

Find IP addresses:

ip a | grep inet

Make HTTP requests:

curl http://<NODE-IP>

Expected: **Nginx default page appears for all nodes.**

**8️. Scaling the Service**

Increase replicas dynamically:

docker service scale my-nginx=5

Verify distribution:

docker service ps my-nginx

**9️. Auto-Scaling Configuration**

Create autoscaler-config.yml:

version: '3'

services:

autoscaler:

image: containrrr/autoscaler

environment:

- SWARM\_MANAGER\_ADDR=<MANAGER-IP>:2377

- SCALE\_THRESHOLD=75

- SCALE\_DOWN\_THRESHOLD=30

- MIN\_REPLICAS=2

- MAX\_REPLICAS=10

deploy:

placement:

constraints:

- node.role == manager

Deploy auto-scaler:

docker stack deploy -c autoscaler-config.yml autoscaler

Simulate high traffic:

ab -n 10000 -c 100 http://<MANAGER-IP>/

**Removing the Swarm**

docker service rm my-nginx

docker swarm leave --force