Container Orchestration

Imagine you have lots of containers (each running a part of your application) on different computers or servers. Now, managing all these containers—starting them, stopping them, scaling them, handling their communication can become very complex.

Container Orchestration is the process of automatically managing and coordinating all these containers. It's like a manager that ensures everything runs smoothly, without you needing to manually start or stop containers every time.

Without orchestration, managing containers can be like managing a bunch of individual apps without any central control.You'd have to manually start containers, connect them, and make sure they stay healthy. Orchestration automates all of this so everything just works on its own!

Popular Container Orchestration Tools

1. Docker Swarm

- What it is: A simple tool that helps you manage Docker containers across multiple servers.
- Advantage: Easy to use: Very simple to set up and manage, especially if you're already using Docker.
- Disadvantage: Limited features, Not as powerful or feature-rich as Kubernetes for managing large-scale applications.

2. Kubernetes

- What it is: A powerful system to manage containers at scale. It automates deployment, scaling, and operations of containerized applications.
- Advantage: Highly scalable: Great for large, complex applications and can automatically scale as your app grows.
- Disadvantage: Complex to set up, Kubernetes can be difficult to configure and manage, especially for beginners.

3. Apache Mesos

- What it is: A system that manages not only containers but also virtual machines and other applications across a cluster of machines.
- Advantage: Supports everything, Can handle containers and non-containerized apps (like VMs).
- Disadvantage: Overkill for small projects, It's more complex than needed for small, container-only setups.

Kubernetes

Kubernetes is a container archistration technology that manages and deploys thousands of containers in cluster.

Kubernetes (also called K8s) is a tool that manages containers.

If Docker helps you run a single container, Kubernetes helps you run many containers across many machines, and keep them organized, available, and scalable.

Imagine you run a pizza shop:

- One oven (Docker) can bake pizzas (your app).
- But if you have 100 customers, one oven isn't enough.
- You now need many ovens, workers, delivery, and a manager to:
    - Assign tasks
    - Replace broken ovens
    - Scale up/down based on orders
    - Make sure no one goes hungry

Kubernetes is that manager.

It:

- Keeps your app running
- Handles more customers (traffic)
- Fixes problems automatically
- Scales the app up or down
-

Kubernetes architecture

1. Node : A Node is a physical machine or a virtual machine (VM) where Kubernetes runs your containers. Node as a worker in the Kubernetes system that actually runs the apps you deploy.

2. Cluster : Group of nodes, working together as team. K8S need atleast one node, but for high availability and fault tolerance, you usually have more than one Node in a cluster.If one Node fails, another Node can take over to keep the applications running.

3. Master Node (Control Plane) : The Master Node is the brain of your Kubernetes cluster. It manages and controls the state of the cluster. The Master Node is where you interact with Kubernetes (via the API Server).

4. Components of Kubernetes :

   a. **API Server**: The API Server is the gateway through which you interact with Kubernetes. All requests go through the API Server, which then communicates with the other components of the cluster.

   b. **etcd**: It is a key-value store that keeps all the cluster data. It stores the state of the cluster and its configurations.

   c. **Kubelet**: An agent that runs on each node, ensuring the containers (pods) are running and healthy.

   d. **Container Runtime**: The software responsible for running containers (e.g., Docker, containerd).

   e. **Controller Manager**: Ensures the desired state of the cluster is maintained, and fixes any issues (like replacing crashed pods).

   f. **Scheduler**: Decides which node should run a pod based on available resources (like CPU, memory).

Imagine you're deploying a simple app on Kubernetes with 3 copies of the app (3 pods):

1. You send a request to the API Server to create 3 pods.
2. The API Server tells the Controller Manager to create the pods.

3. The Scheduler decides which Nodes should run the pods based on available resources.
4. The Kubelet on each Node starts the containers (with help from the Container Runtime).
5. The Controller Manager ensures 3 pods are always running (if one crashes, it's automatically replaced).
6. All this information is stored and tracked in etcd, so Kubernetes always knows what's going on.

Kubernetes + Docker

- Docker: runs one container
- Kubernetes: runs lots of containers across many computers

They work together:

- You build your app with Docker
- You run & manage it at scale with Kubernetes

---

developed by google one of the imp orchestration container + orchestration