Container orchestration

Container orchestration is all about managing the lifecycles of containers, especially in large, dynamic environments. Software teams use container orchestration to control and automate many tasks:

- Provisioning and deployment of containers
- Redundancy and availability of containers
- Scaling up or removing containers to spread application load evenly across host infrastructure
- Movement of containers from one host to another if there is a shortage of resources in a host, or if a host dies
- Allocation of resources between containers
- External exposure of services running in a container with the outside world
- Load balancing of service discovery between containers
- Health monitoring of containers and hosts
- Configuration of an application in relation to the containers running it

How does container orchestration work?

- When you use a container orchestration tool, like Kubernetes or Docker Swarm (more on these shortly), you typically describe the configuration of your application in a YAML or JSON file, depending on the orchestration tool.

- These configurations files (for example, docker-compose.yml) are where you tell the orchestration tool where to gather container images (for example, from Docker Hub), how to establish networking between containers, how to mount storage volumes, and where to store logs for that container.

- Typically, teams will branch and version control these configuration files so they can deploy the same applications across different development and testing environments before deploying them to production clusters.

- Containers are deployed onto hosts, usually in replicated groups. When it's time to deploy a new container into a cluster, the container orchestration tool schedules the deployment and looks for the most appropriate host to place the container based on predefined constraints (for example, CPU or memory availability).

- You can even place containers according to labels or metadata, or according to their proximity in relation to other hosts—all kinds of constraints can be used.

- Once the container is running on the host, the orchestration tool manages its lifecycle according to the specifications you laid out in the container's definition file (for example, its Dockerfile).
- The beauty of container orchestration tools is that you can use them in any environment in which you can run containers. And containers are supported in just about any kind of environment these days, from traditional on-premise servers to public cloud instances running in Amazon Web Services (AWS), Google Cloud Platform (GCP), or Microsoft Azure. Additionally, most container orchestration tools are built with Docker containers in mind.

Container Orchestration Tools:

Amazon ECS — The Amazon EC2 Container Service (ECS) supports Docker containers and lets you run applications on a managed cluster of Amazon EC2 instances.

Azure Container Service (ACS) — ACS lets you create a cluster of virtual machines that act as container hosts along with master machines that are used to manage your application containers.

Cloud Foundry's Diego — Diego is a container management system that combines a scheduler, runner, and health manager. It is a rewrite of the Cloud Foundry runtime.

CoreOS Fleet — Fleet is a container management tool that lets you deploy Docker containers on hosts in a cluster as well as distribute services across a cluster.

Docker Swarm — Docker Swarm provides native clustering functionality for Docker containers, which lets you turn a group of Docker engines into a single, virtual Docker engine.

Google Container Engine — Google Container Engine, which is built on Kubernetes, lets you run Docker containers on the Google Cloud platform. It schedules containers into the cluster and manages them based on user-defined requirements.

Kubernetes — Kubernetes is an orchestration system for Docker containers. It handles scheduling and manages workloads based on user-defined parameters.

Mesosphere Marathon — Marathon is a container orchestration framework for Apache Mesos that is designed to launch long-running applications. It offers key features for running applications in a clustered environment.

Openshift

Selecting a container runtime for use with Kubernetes Interfaces

- Native
- Docker
- rktnetes
- CRI
- cri-containerd
- rktlet
- cri-o
- frakti

OCI (Open Container Initiative) Compatible Runtimes Containers

- bwrap-oci
- crun
- railcar
- rkt
- runc
- runlxc
- Virtual Machines
- Clear Containers
- runv