Kubernetes

Container Orchestration

Container orchestration is all about managing the life cycles 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

Container Orchestration Tools

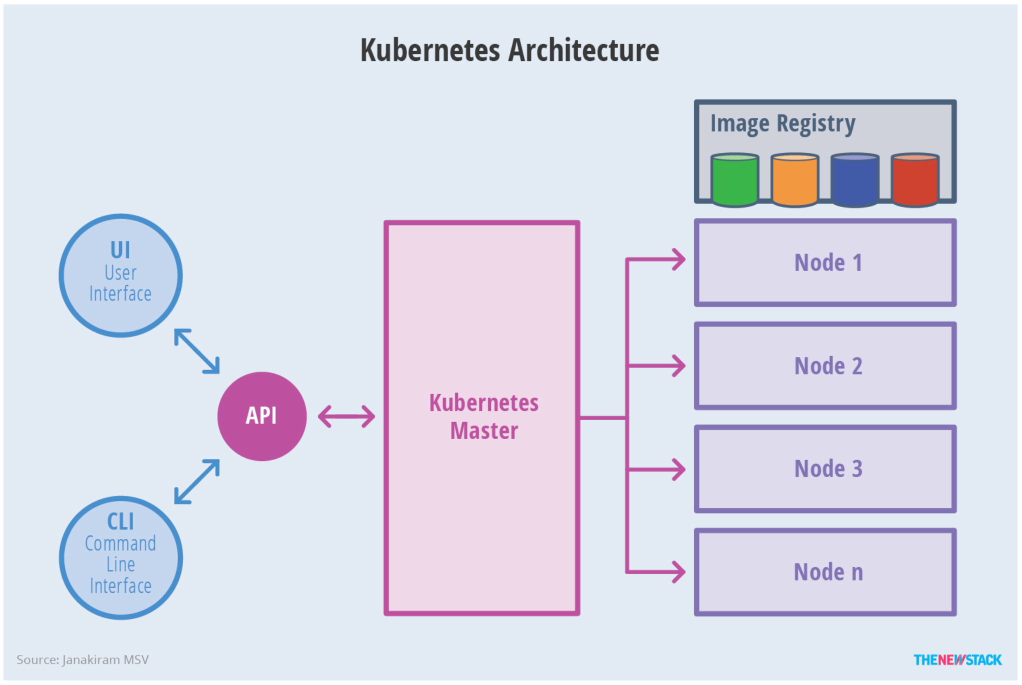
- Docker Swarm - Docker Swarm provides native clustering functionality for Docker containers, which lets you turn a group of Docker engines into a single, virtual Docker engine.
- Mesosphere Marathon - Marathon is a container orchestration framework for Apache Mesos that is designed to launch long-running applications. It offers key features for running applications in a clustered environment.
- Kubernetes - Kubernetes is an orchestration system for Docker containers. It handles scheduling and manages workloads based on user-defined parameters.

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

Why Kubernetes

- Agile application creation and deployment: Increased ease and efficiency of container image creation compared to VM image use.
- Continuous development, integration, and deployment: Provides for reliable and frequent container image build and deployment with quick and easy rollbacks (due to image immutability).
- Dev and Ops separation of concerns: Create application container images at build/release time rather than deployment time, thereby decoupling applications from infrastructure.
- Observability Not only surfaces OS-level information and metrics, but also application health and other signals.
- Environmental consistency across development, testing, and production: Runs the same on a laptop as it does in the cloud.
- Cloud and OS distribution portability: Runs on Ubuntu, RHEL, CoreOS, on-prem, Google Kubernetes Engine, and anywhere else.
- Application-centric management: Raises the level of abstraction from running an OS on virtual hardware to running an application on an OS using logical resources.
- Loosely coupled, distributed, elastic, liberated micro-services: Applications are broken into smaller, independent pieces and can be deployed and managed dynamically – not a monolithic stack running on one big single-purpose machine.
- Resource isolation: Predictable application performance.
- Resource utilization: High efficiency and density.



Kubernetes Architecture

1. Master Components
    a. Master components provide the cluster’s control plane.
    b. Master components make global decisions about the cluster (for example, scheduling), and detecting and responding to cluster events (starting up a new pod when a replication controller’s ‘replicas’ field is unsatisfied).
    c. Master components can be run on any machine in the cluster.
    d. Master components can be api server, etcd, scheduler.
2. Node components
    a. Node components run on every node, maintaining running pods and providing the Kubernetes runtime environment.
    b. Node components can be kubelet, proxy and container runtime

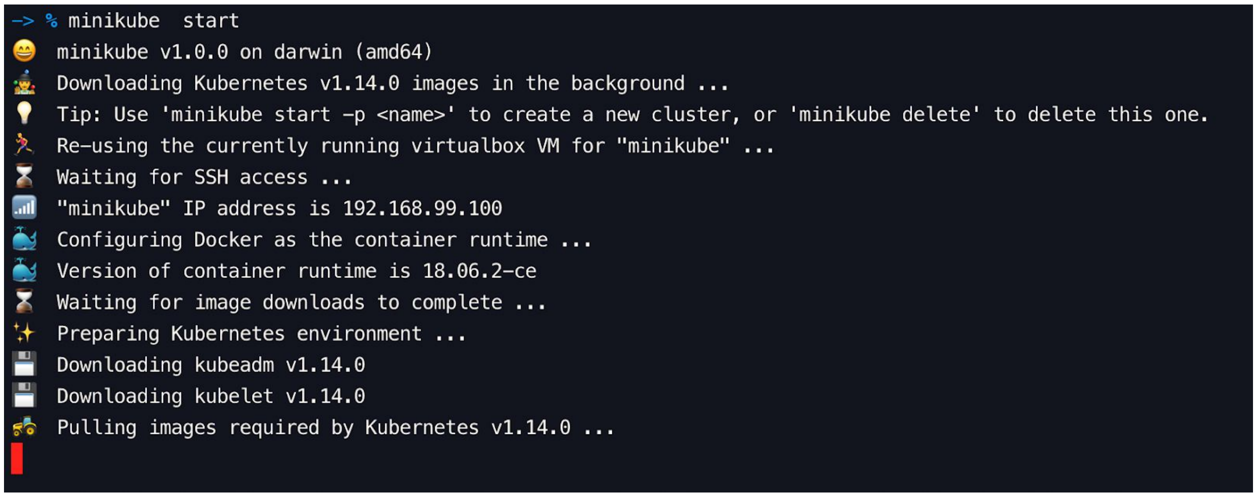
Working with Kubernetes

- To work with Kubernetes, you use Kubernetes API objects to describe your cluster’s desired state
- You describe what applications or other workloads you want to run, what container images they use, the number of replicas, what network and disk resources you want to make available, and more.
- You set your desired state by creating objects using the Kubernetes API, typically via the command-line interface, kubectl.
- You can also use the Kubernetes API directly to interact with the cluster and set or modify your desired state.

Environment Setup - Kubectl CLI

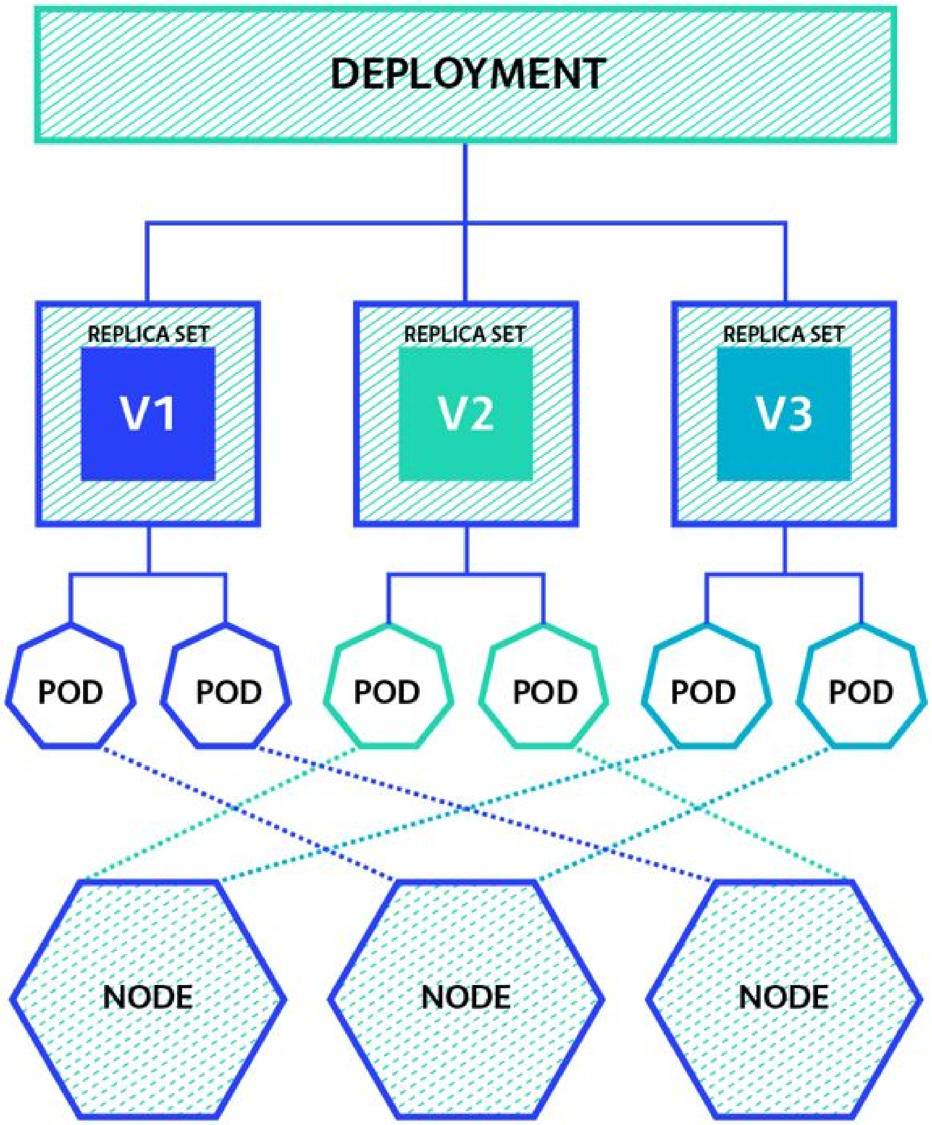
- Use the Kubernetes command-line tool, kubectl, to deploy and manage applications on Kubernetes.
- Using kubectl, you can inspect cluster resources;
- With Kubectl you can also create, delete, and update components within your kubernetes cluster
- Kubectl command allows us to work in two methods:
    a. Declarative.
    b. Ad Hoc commands.
- When working in declarative way we will create a YAML file that will describe our kubernetes components and kubectl will be used to deploy, update or remove them from our cluster.
- When working in Ad Hoc commands we will run specific commands against our kubernetes cluster.

- Open CMD / Terminal:
    - Run minikube start



```
-> % minikube  start
😄  minikube v1.0.0 on darwin (amd64)
👷  Downloading Kubernetes v1.14.0 images in the background ...
💡  Tip: Use 'minikube start -p <name>' to create a new cluster, or 'minikube delete' to delete this one.
🏃  Re-using the currently running virtualbox VM for "minikube" ...
⌛  Waiting for SSH access ...
📶  "minikube" IP address is 192.168.99.100
🐳  Configuring Docker as the container runtime ...
🐳  Version of container runtime is 18.06.2-ce
⌛  Waiting for image downloads to complete ...
✨  Preparing Kubernetes environment ...
💾  Downloading kubeadm v1.14.0
💾  Downloading kubelet v1.14.0
🚜  Pulling images required by Kubernetes v1.14.0 ...
```

Kubernetes Components Hierarchy



Kubernetes Pods

- A pod (as in a pod of whales or pea pod) is a group of one or more containers (such as Docker containers), with shared storage/network, and a specification for how to run the containers.
- A pod’s contents are always co-located and co-scheduled, and run in a shared context.
- A pod models an application-specific “logical host” - it contains one or more application containers which are relatively tightly coupled — in a pre-container world, being executed on the same physical or virtual machine would mean being executed on the same logical host.

Kubernetes Pods Example

1. kubectl run nginx --image=nginx --generator=run-pod/v1
2. kubectl run redis --image=redis:5.0.4-alpine --generator=run-pod/v1
3. kubectl logs redis
4. kubectl describe pod redis

Kubernetes Deployment

- A Deployment controller provides declarative updates for Pods and ReplicaSets.
- You describe a desired state in a Deployment object, and the Deployment controller changes the actual state to the desired state at a controlled rate.
- You can define Deployments to create new ReplicaSets, or to remove existing Deployments and adopt all their resources with new Deployments.
- The following are typical use cases for Deployments:
  - Create a Deployment to rollout a ReplicaSet.
  - Change the state of pods group
  - Rollback to an earlier Deployment revision if the current state of the Deployment is not stable.
  - Scale up the Deployment to facilitate more load.
  - Use the status of the Deployment as an indicator that a rollout has stuck.
  - Clean up older ReplicaSets that you don’t need anymore.

Kubernetes Services

- A Kubernetes Service is an abstraction which defines a logical set of Pods and a policy by which to access them - sometimes called a micro-service.
- The set of Pods targeted by a Service is (usually) determined by a Label Selector (see below for why you might want a Service without a selector).

Kubernetes Workload Example

1. kubectl create deployment hello-node --image=gcr.io/hello-minikube-zero-install/hello-node
2. kubectl get deployments
3. kubectl describe deployment hello-node
4. kubectl get pods
5. kubectl scale deployment/hello-node --replicas=3
6. kubectl get all
7. kubectl expose deployment hello-node --type=LoadBalancer --port=8080
8. kubectl get services
9. minikube addons enable heapster