Tutorial 5: Kubernetes

Question Set:

Q1. Please describe what Kubernetes (K8s) is.

ANSWER:

Kubernetes [1] is an open-source container orchestration system for automating computer application deployment, scaling, and management. It was originally designed by Google and is now maintained by the Cloud Native Computing Foundation. It aims to provide a "platform for automating deployment, scaling, and operations of application containers across clusters of hosts". It works with a range of container tools and runs containers in a cluster, often with images built using Docker.

Kubernetes defines a set of building blocks ("primitives"), which collectively provide mechanisms that deploy, maintain, and scale applications based on CPU, memory or custom metrics. Kubernetes is loosely coupled and extensible to meet different workloads. This extensibility is provided in large part by the Kubernetes API, which is used by internal components as well as extensions and containers that run on Kubernetes. The platform exerts its control over compute and storage resources by defining resources as Objects, which can then be managed as such. Kubernetes follows the primary/replica architecture. The components of Kubernetes can be divided into those that manage an individual node and those that are part of the control plane.

Many cloud services offer a Kubernetes-based platform or infrastructure as a service (PaaS or IaaS) on which Kubernetes can be deployed as a platform-providing service. Many vendors (e.g., Google, Amazon) also provide their own branded Kubernetes distributions (e.g., Google Kubernetes Engine, aka GKE).

Q2. What are the motivations of using Kubernetes?

ANSWER:

In DevOps and Microservice architecture, container is a good way to guarantee your application and development more reliable and scalable, where you need to manage the multiple containers across many virtual machines (in a cloud environment) and ensure that there is no downtime. For example, if a container goes down, another container needs to start. If a service needs to be scaled out, more containers need to start. It’s impossible to manually handle these orchestration jobs 24*7 without problems using human labors. That's how Kubernetes comes to make our life easier. Kubernetes provides you with a framework to run distributed systems resiliently. It takes care of scaling and failover for your application, provides deployment patterns, and more. For example, Kubernetes can easily manage hundreds of docker images across a number of virtual machines (a K8s cluster) for your system.

The benefits of using Kubernetes are including but not limited to:

- Kubernetes can balance the traffic when a container is overloaded (too much traffic) to ensure the deployment is stable.

- Kubernetes allows you to automatically create new containers and remove the existing containers to achieve rollout/rollback, self-healing, and scaling operations.
- Kubernetes can fit containers with specified resource usage (e.g. how much CPU and RAM per container) to make the best use of your resources.
- Kubernetes has secure configuration management, which enables you to store and manage sensitive information (e.g., passwords, OAuth tokens, and SSH keys) without rebuilding your container images.

Q3. What is the purpose of the Reconciliation Loop?
ANSWER:

The Reconciliation Loop, also called a control loop, is the fundamental mechanism that Kubernetes (and other orchestrators) uses to keep the cluster reliable and consistent.

- What it is:

  The reconciliation loop is a continuous process where Kubernetes controllers monitor the system, compare the actual state of resources with the desired state defined by the user (usually through YAML manifests), and then take corrective actions if there is a mismatch.

- Purpose:

  The goal is to maintain self-healing and consistency in the cluster. Instead of requiring manual intervention every time something changes or fails, the reconciliation loop ensures that the system automatically moves back toward the desired configuration. This reduces human error and keeps applications running as expected.

- Core Steps:

  1. Observe – The controller checks the current state of resources (e.g., how many pods are running).
  2. Compare – It evaluates whether the current state matches the desired state defined by the user.
  3. Act – If there is a difference, the controller takes action to fix it (e.g., create, delete, or update resources).
     This cycle repeats continuously in the background.

- Example:

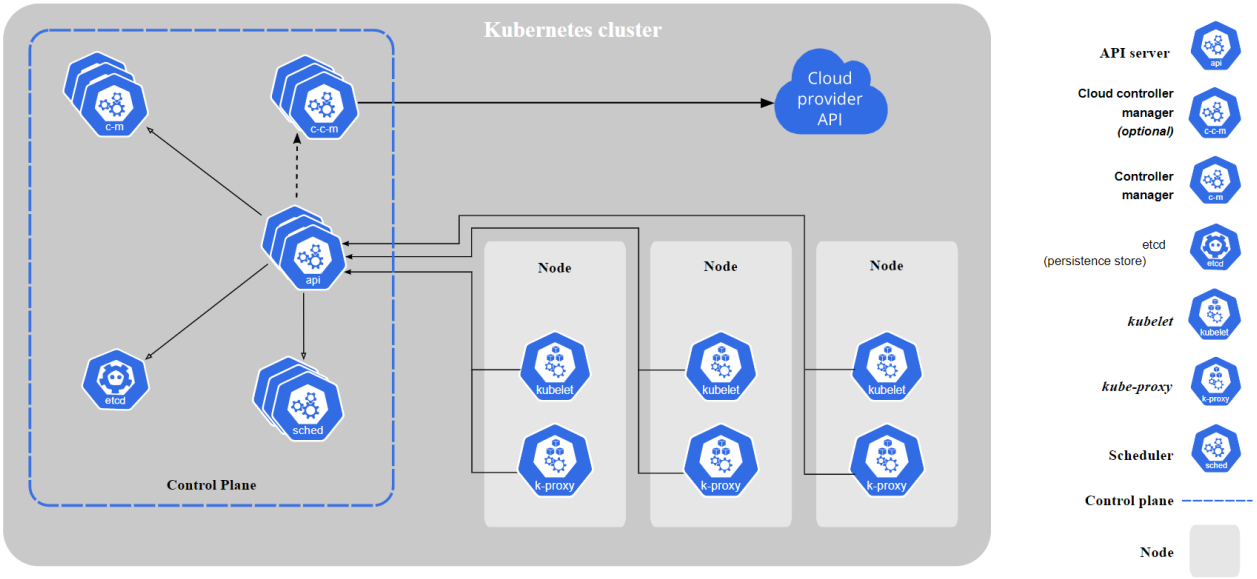
  Suppose a Deployment YAML specifies 3 replicas of an Nginx pod. If one pod crashes and only 2 remain:

  - The reconciliation loop observes the actual state (2 running pods).
  - It compares this with the desired state (3 pods).
  - It then acts by scheduling and starting a new pod, bringing the cluster back into alignment.

Through this mechanism, Kubernetes provides automation, resilience, and scalability, ensuring that workloads always match the user's intent.

Q4. Please describe the components and their functions in a Kubernetes cluster.

ANSWER:



A Kubernetes cluster consists of a control panel and a set of worker machines (called nodes) that run containerized applications. Every cluster has at least one worker node.

Control Panel and its components

The control plane's components make global decisions about the cluster (e.g., scheduling), as well as detecting and responding to cluster events (e.g., starting up a new pod when a deployment's replicas field is unsatisfied).

- kube-apiserver: is the main implementation of a Kubernetes API server.
- etcd: consistent and highly available key-value store used as Kubernetes' backing store for all cluster data.
- kube-scheduler: monitors newly created Pods with no assigned node and selects a node for them to run on. Some factors will be taken into account for scheduling decisions including individual and collective resource requirements, hardware/software/policy constraints, affinity and anti-affinity specifications, data locality, inter-workload interference, and deadlines.
- kube-controller-manager runs controller processes, which are separately running for different purposes.
    - Node controller: Responsible for noticing and responding when nodes go down.
    - Job controller: Watches for Job objects that represent one-off tasks, then creates Pods to run those tasks to completion.

    - Endpoints controller: Populates the Endpoints object (that is, joins Services & Pods).
    - Service Account & Token controllers: Create default accounts and API access tokens for new namespaces.
- cloud-controller-manager integrates cloud-specific control logic, allowing you to link your cluster into cloud provider's API, and separate out the components that interact with that cloud platform from components that only interact with your cluster. Note that if you are running Kubernetes on your own premises, or in a learning environment inside your own PC, the cluster does not have a cloud controller manager.

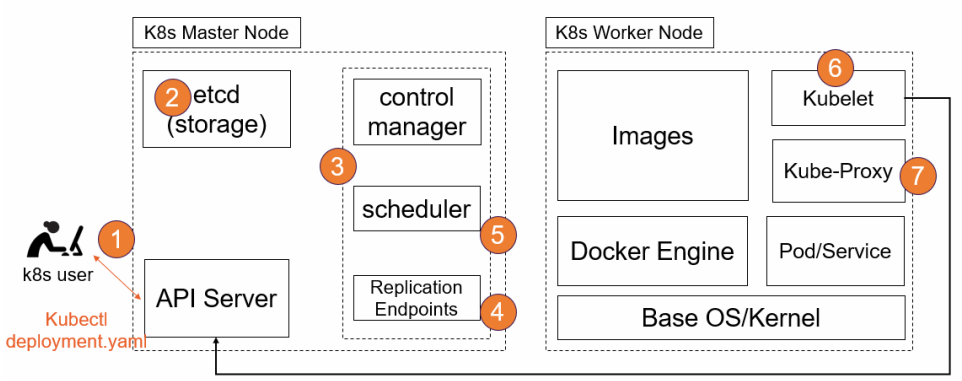
Node Components

Node components run on every node, maintaining running pods and providing the Kubernetes runtime environment.

- kubelet: runs on each node in the cluster. The kubelet takes a set of PodSpecs that are provided through various mechanisms and ensures that the containers described in those PodSpecs are running and healthy. The kubelet doesn't manage containers which were not created by Kubernetes.
- kube-proxy: is a network proxy that runs on each node in your cluster. kube-proxy maintains network rules on nodes. These network rules allow network communication to your Pods from network sessions inside or outside of your cluster.
- Container runtime engine is the software that is responsible for running containers. Kubernetes supports several container runtimes: Docker, containerd, CRI-O, and any implementation of the Kubernetes CRI (Container Runtime Interface).
- Addons and more:
    - Addons use Kubernetes resources (DaemonSet, Deployment, etc) to implement cluster features. Because these are providing cluster-level features, namespaced resources for addons belong within the kube-system namespace. Selected addons are described below; for an extended list of available addons, please see Addons.
    - Web UI (Dashboard): allows users to manage and troubleshoot applications running in the cluster, as well as the cluster itself.
    - Container Resource Monitoring: records generic time-series metrics about containers in a central database and provides a UI for browsing that data.
    - Cluster-level Logging is responsible for saving container logs to a central log store with search/browsing interface.

Q5. Please generally describe the workflow of Kubernetes.

ANSWER:



1. User via "kubectl" deploys a new application. Kubectl sends the request to the API Server.
2. API server receives the request and stores it in the data store (etcd). Once the request is written to data store, the API server is done with the request.
3. Watchers detects the resource changes and send a notification to controller to act upon it
4. Controller detects the new app and creates new pods to match the desired number# of instances. Any changes to the stored model will be picked up to create or delete Pods.
5. Scheduler assigns new pods to a Node based on a criterion. Scheduler makes decisions to run Pods on specific Nodes in the cluster. Scheduler modifies the model with the node information.
6. Kubelet on a node detects a pod with an assignment to itself and deploys the requested containers via the container runtime (e.g., Docker). Each Node watches the storage to see what pods it is assigned to run. It takes necessary actions on resource assigned to it like create/delete Pods.
7. Kubeproxy manages network traffic for the pods - including service discovery and load-balancing. Kubeproxy is responsible for communication between Pods that want to interact.

References

[1] Kubernetes, https://en.wikipedia.org/wiki/Kubernetes
[2] Official Documentation of Kubernetes, https://kubernetes.io/docs/home/
[3] Kube101 workshop, https://ibm.github.io/kube101/