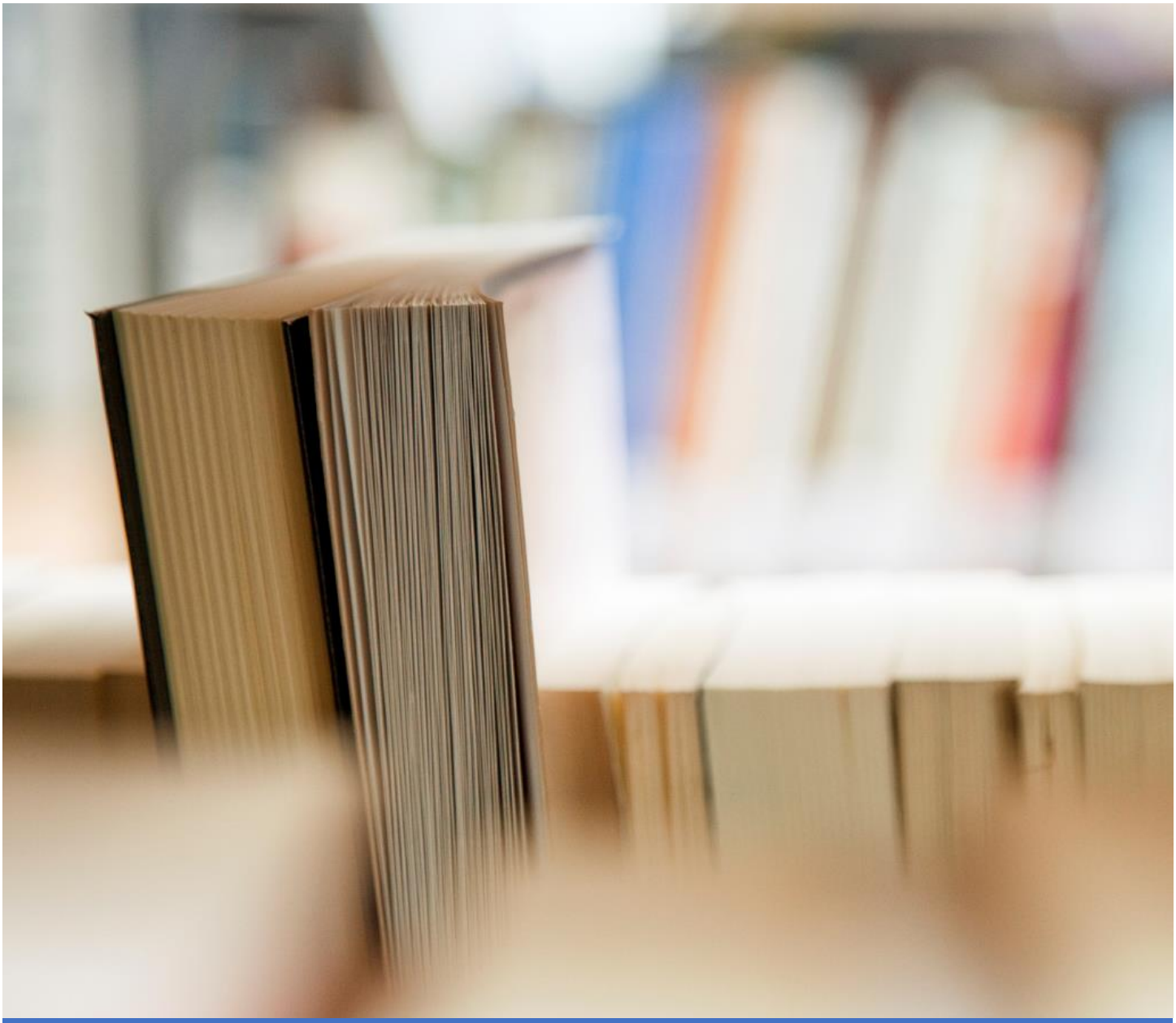


Kubernetes 101

Understanding Kubernetes

Contents

Kubernetes Concepts 2
Kubernetes Components and Architecture 3
Kubernetes High Level Architecture Diagram 3
Master Components 4
Node (worker) components 4
Kubectl 5
Reference 5

Kubernetes Concepts

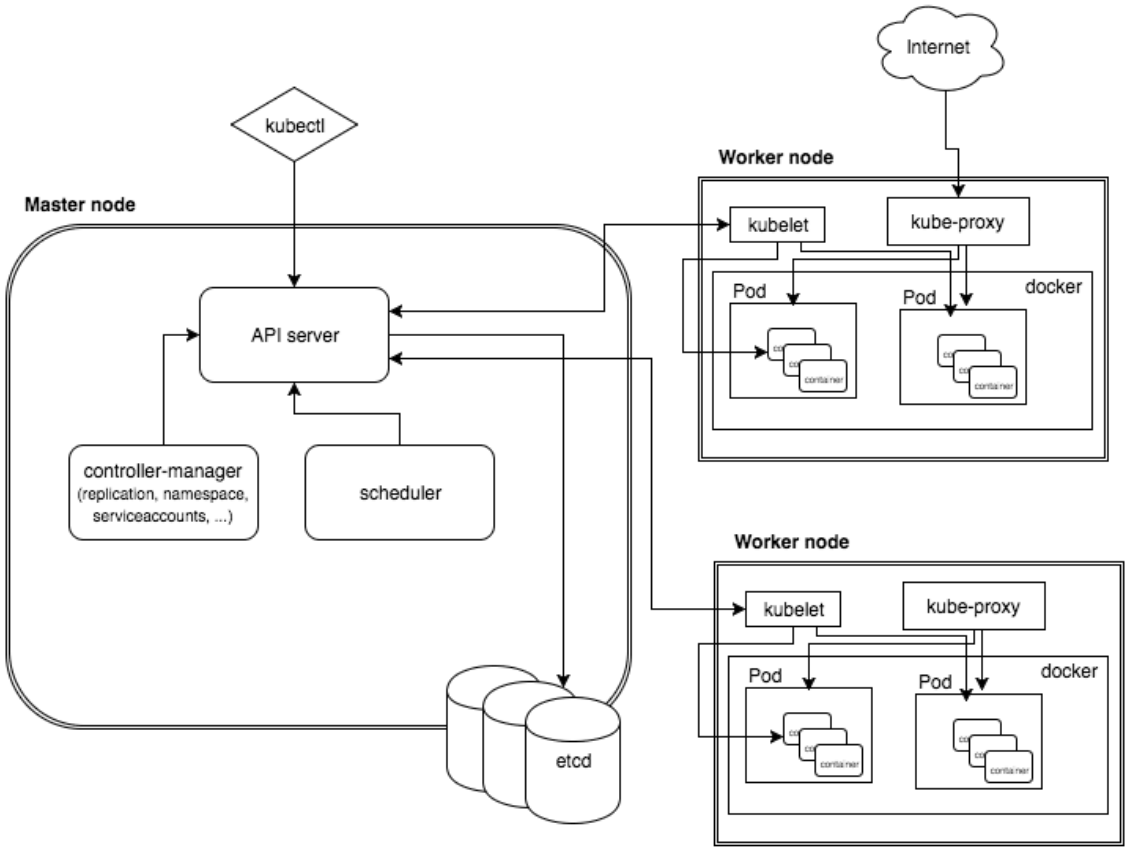
Making use of Kubernetes requires understanding the different abstractions it uses to represent the state of the system, such as services, pods, volumes, namespaces, and deployments.

- **Pod** - generally refers to one or more containers that should be controlled as a single application. A pod encapsulates application containers, storage resources, a unique network ID and other configuration on how to run the containers.
- **Service** - pods are volatile, that is Kubernetes does not guarantee a given physical pod will be kept alive (for instance, the replication controller might kill and start a new set of pods). Instead, a service represents a logical set of pods and acts as a gateway, allowing (client) pods to send requests to the service without needing to keep track of which physical pods actually make up the service.
- **Volume** - similar to a container volume in Docker, but a Kubernetes volume applies to a whole pod and is mounted on all containers in the pod. Kubernetes guarantees data is preserved across container restarts. The volume will be removed only when the pod gets destroyed. Also, a pod can have multiple volumes (possibly of different types) associated.
- **Namespace** - a virtual cluster (a single physical cluster can run multiple virtual ones) intended for environments with many users spread across multiple teams or projects, for isolation of concerns. Resources inside a namespace must be unique and cannot access resources in a different namespace. Also, a namespace can be allocated a resource quota to avoid consuming more than its share of the physical cluster's overall resources.
- **Deployment** - describes the desired state of a pod or a replica set, in a yaml file. The deployment controller then gradually updates the environment (for example, creating or deleting replicas) until the current state matches the desired state specified in the deployment file. For example, if the yaml file defines 2 replicas for a pod but only one is currently running, an extra one will get created. Note that replicas managed via a deployment should not be manipulated directly, only via new deployments.

Kubernetes Components and Architecture

Kubernetes follows a client-server architecture. It's possible to have a multi-master setup (for high availability), but by default there is a single master server which acts as a controlling node and point of contact. Please note that for AKS/EKS/GKS, only a single master node is allowed. The master server (Node) consists of various components including a kube-apiserver, an etcd storage, a kube-controller-manager, a cloud-controller-manager, a kube-scheduler, and a DNS server for Kubernetes services. Node components include kubelet and kube-proxy on top of Docker.

Kubernetes High Level Architecture Diagram



Master Components

Below are the main components found on the master node:

- **etcd cluster** - a simple, distributed key value storage which is used to store the Kubernetes cluster data (such as number of pods, their state, namespace, etc), API objects and service discovery details. It is only accessible from the API server for security reasons. etcd enables notifications to the cluster about configuration changes with the help of watchers. Notifications are API requests on each etcd cluster node to trigger the update of information in the node's storage.
- **kube-apiserver** - Kubernetes API server is the central management entity that receives all REST requests for modifications (to pods, services, replication sets/controllers and others), serving as frontend to the cluster. Also, this is the only component that communicates with the etcd cluster, making sure data is stored in etcd and is in agreement with the service details of the deployed pods.
- **kube-controller-manager** - runs a number of distinct controller processes in the background (for example, replication controller controls number of replicas in a pod, endpoints controller populates endpoint objects like services and pods, and others) to regulate the shared state of the cluster and perform routine tasks. When a change in a service configuration occurs (for example, replacing the image from which the pods are running, or changing parameters in the configuration yaml file), the controller spots the change and starts working towards the new desired state.
- **cloud-controller-manager** - is responsible for managing controller processes with dependencies on the underlying cloud provider (if applicable). For example, when a controller needs to check if a node was terminated or set up routes, load balancers or volumes in the cloud infrastructure, all that is handled by the cloud-controller-manager.
- **kube-scheduler** - helps schedule the pods (a co-located group of containers inside which our application processes are running) on the various nodes based on resource utilization. It reads the service's operational requirements and schedules it on the best fit node. For example, if the application needs 1GB of memory and 2 CPU cores, then the pods for that application will be scheduled on a node with at least those resources. The scheduler runs each time there is a need to schedule pods. The scheduler must know the total resources available as well as resources allocated to existing workloads on each node.

Node (worker) components

Below are the main components found on a (worker) node:

- **kubelet** - the main service on a node, regularly taking in new or modified pod specifications (primarily through the kube-apiserver) and ensuring that pods and their containers are healthy and running in the desired state. This component also reports to the master on the health of the host where it is running.
- **kube-proxy** - a proxy service that runs on each worker node to deal with individual host subnetting and expose services to the external world. It performs request forwarding to the correct pods/containers across the various isolated networks in a cluster.

Kubectl

kubectl command is a line tool that interacts with kube-apiserver and send commands to the master node. Each command is converted into an API call.

Reference

https://github.com/cloudmustafa/DockerOnAzureAKS