Contents

1. Introduction

1.1. Purpose of the Thesis

1.2. Scope of the Thesis

1.3. Structure of the Thesis

In chapter Background and Related Work

2. Background and Related Work

2.1. Containerization

Contenerization is packaging an app along with all necessary runtime stuff like libraries, executables or asets into an object called "container". The main benefits of container are[1]:

- Protable and Flexible – container can be run on bare metal or virtual machine in cloud regardless of operating system. Only a container runtime software like Docker Engine or containerd is required, which allows to interact with host system.
- Lightweight – container is sharing operating system kernel with hostmachine, there is no need to install separate operating system inside
- Isolated – does not depends on host's environment or infrastructure
- Standarized – Open Container Initiative standarize runtime, image and distribution specifications

A container image is set of files and configuration needed to run a container. It is immutable, only new image can be crated with new changes. Consists of layers. The layer contain one modification made a image. All layers are cachable and can be reused when building an image. The mechanism is really usefull when compiling large application components inside one container[2].

2.2. Contianer Orchestration

Container orchestration is coordinated deploying, managing, networking, scaling and monitoring containers process. It automates and manages whole container's lifecycle, there is no need to worrying about of deployed app, orchestration software like Kubernetes will take care of its availability [1].

The Kubernetes Authors says: "The name Kubernetes originates from Greek, meaning helmsman or pilot. K8s as an abbreviation results from counting the eight letters between the

"K" and the "s"" [3]. K8s is open-source orchestration platform capable of managing containers [3]. Key functionalities are [3]:

- Automated rollouts and rollbacks – updates or downgrades version of deplotyed containers at controller rate, replacing containers incrementally
- Automatic bin packing – allows to specify exact resources needed by container (CPU, Memory) to fit on appropriate node
- Batch execution – possible to create sets of tasks which can be run without manual intervention
- Designed for extensibility – permits to add feautres using custom resource definitions without changing source code
- Horizontal scaling – scales (replicate) app based of its need for resources
- IPv4/IPv6 dual-stack – allocates IPv4 or IPv6 to pods and services
- Secret and configuration management – allows store, manage and update secrets. Containers do not have to be rebuilt to access updated credentials
- Self-healing – restarts crashed containers or by failure specified by user
- Service discovery and load balancing – advertises a container using DNS name or ip. Loadbalances traffic across all pods in deployment
- Storage orchestration – mounts desired storage like local or shipped by cloud provider and make it available for containers

Understanding Kubernetes workflow becomes significantly easier by familiarizing with its architecture, which will be discussed in the following seciton.

2.3. Kubernetes Architecture

A Kubernetes cluster is a group of machines that run containers and provide all the necessary services to enable communication between containers within the cluster, as well as access to the cluster from the outside. There are two types of components, a control plane and worker node. Minumum one of each is needed to run a container [4].

On figure 2.1 there is graphical representation of kubernetes cluster. Not all of components shown on figure are mandatory for kubernetes to work correctly. Take a look on control plane part, *cloud-controller-manager* might not be mandatory, in on-premise configurations where

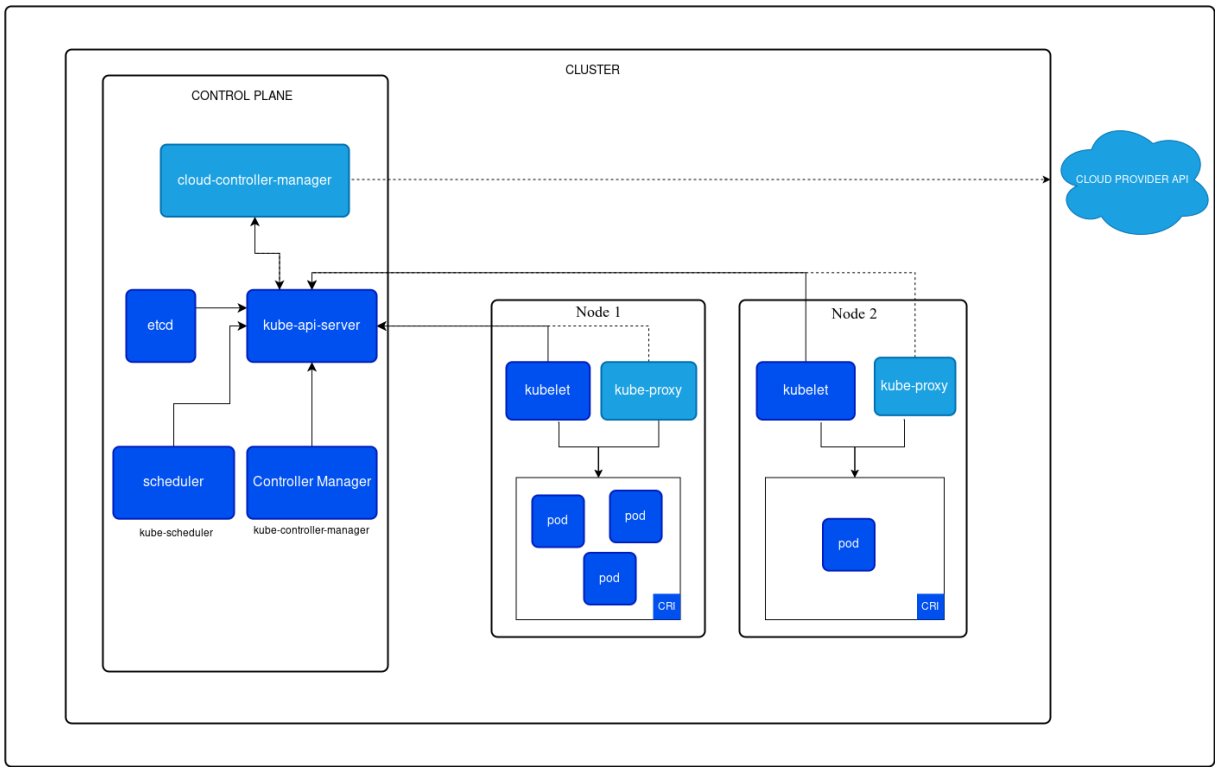

Figure 2.1: Kubernetes Cluster Architecture [4]

interacting with cloud provider is not needed. On the right side of figure in node representation is *kube-proxy* component, which is not mandatory as some networking plugins can provide own implementation of proxy [4]. This is example of "Designed for extensibility", where Kubernetes can acquire 3rd-party features without changing its source code [3].

2.3.1. Control Plane

Control plane is like a brain in Kubernetes cluster. To interact with use kubcetl, asks *kube-apiserver*It is responsible for communication with worker nodes running pods, a smallest unit managed by K8s that has containers inside.

cloud-controller-manager

This component allows connect Kubernetes cluster and interact with cloud provider's API. It is combined with kube-controller-manager as single binary and can be replicated. This is the only component that talks to the cloud provider, separating other components from direct communication with the cloud. When running without cloud envirnoment this component is absent [4].

etcd

Etcd is an open source distributed key-value store service often used in distributed systems. It is responsible for maintaining both the current state and its previous version in its persistent memory [4][5].

kube-apiserver

Exposes Kubernetes API to interact with a cluster. Takes responsibility for handling all requests from components and users. This is the component which answers for cluster administrator requests sent by kubectl [4].

kube-controller-manager

Component which run controller processes. Its compiled binary consists of multiple controllers. Example controllers are [4]:

- Node controller – observes worker nodes if are up and running
- Job controller – responsible for batch execution jobs
- EndpointSlice controller – connects services with pods

More controller names can be found in kubernetes source code.

kube-scheduler

Takes care of pods which are not assigned to a worker node yet. kube-scheduler is looking for node that meets pod's scheduling requirements and fit a pod on that node. Such a node is called feasible node [6].

2.3.2. Nodes

All of below mentioned components run on every node in a cluster.

Container runtime

Node's key component, has ability to run, execute commands, manage and delete containers in efficient way [4].

kube-proxy

Create networking rules which allows to communicate with Pods from outside cluster. If available kube-proxy uses operating system packet filtering to create set of rules. It is also able to forward traffic by itself. This component is optional, can be replaced with different one if desired one implements key features. [4].

kubelet

It is responsible for managing containers inside pod on its node. Uses Container Runtime Interface to communicate with containers [4] [7].

2.3.3. Objects

Namespace

The purpose of namespace object is to isolate group of resources like pods, deployments, services etc. in a cluster. It helps to organise cluster into virtual sub areas of working space. If *Service* is created in some custom namespace <service-name>.<namespace-name>.svc.cluster.local DNS entry within cluster is created [8].

Pods

Pods are the smallest deployable objects in Kubernetes. It contains in one or more containers, which can communicate with each other using localhost interface. Since they share IP address, the can not use the same ports. It is really useful, when our service consist of two apps which are coupled together. For example there is a pod which has two containers, one responsible for compiling a code, second one is creating cache entry from compiled object and uploads to some data storage. It has more sense, as sharing data among containers in a pod is rather easier than on node between pods. Scaling is simpler as replicating one pod instead of two. Moreover communication between apps happends using localhost, in scenario where there are two pods with one container, ClusterIP *Service* is needed. However the most common approach is to run one container per pod, where pod is just managing wrapper for contenerized app. Also rather then creating pod directly it is more common to use workload resource like *Deployment* [9].

ReplicaSet

Basically *ReplicaSet* consists of pod template, and runs desired number of pods [10].

Deployment

Deployment is a higher level abstraction over *ReplicaSet*, that manages its lifecycle. It provides more features like rollingback an app, as it keeps history of configurations [11].

DaemonSet

Running pods using DaemonSet guarantee that every node will have copy of desirded pod (if resource requirements are met etc.). It has ability to automatically add or remove pods, if number of nodes changes. The typical usege is creating monitoring pod on every node [12].

StatefulSet

StatefulSet unlike *Deployment* is stateful. It saves an identity of each pod and if e.g. some persistent storage is assigned to specific e.g. database pod and it dies, kubernetes will recreate pod on the same node as it was previously [13].

Job

Runs pod that does one task and exists. Kubernetes will retry execution if pod fails specific number of tries set in its configuration [14].

CronJob

Behaviours like *Job*, but is able to run regularly every given time for tasks like database backups or log rotation [15].

Service

Service exposes an application running inside a cluster by using an enpoint. As a pod is ephemeral resource and its address changes from time to time (e.g. when pod is recreated) it better to create dns name that resolves IP address. Moreover service will not advertise unhealthy pods. Usually service exposes one port per service, but for example web app might expose http and https ports. There are four types of services [16].

1. ClusterIP – makes one pod available to other inside cluster by exposing application using inter-cluter IP address. Although its oriented to be accessible within the cluster, objets like *Ingress* or *Gateway API* can expose service to the outside.

2. NodePort – by default allocates port (from range 30000-32767) to publish service on every node's ip. In this scenario every node on specified port acts like a proxy to deployed app.

3. LoadBalancer – kubernetes does not provide loadbalancer by default and when creating such a service it interacts with cloud provider to create external service for traffic balancing. Loadbalancer can be installed inside cluster.

4. ExternalName – allows pods inside Kubernetes to access external service using defined name rather than using IP address

LOCAL TRAFFIC POLICY

Ingress

Ingress is an object that manages outside cluster access to services inside a cluster. It is a single point of entry to route traffic to specified pod based on configuration. This is only high abstract object that specifies routing rules in cluster. Real functionalities are provided by an

Ingress Controller. Nowadays the development of Ingress is frozen, Kubernetes authors pay attention to its successor a *Gateway API* [17].

Ingress Controller

Ingress Controller fulfills an *Ingress* and starts serving an application which performs configured rules. Any implementation has its own features, but common functionalities are L4/L7 loadbalancing, host and path based routing, SSL termination. This is the real application that runs in a pod. Ingress Controller have to be installed manually and is not part of Kubernetes, however the container orchestration tool developers maintain AWS, GCE, and nginx ingress controllers [17][18].

Gateway API

Gateway API vs API Gateway GatewayClass etc workflow

2.3.4. Interfaces

TEST

TEKST asdasdasda

2.4. Cluster Networking

Networking is a most important thing in Kubernetes, the whole point is to obtain reliable and robust communication among containers, pods, services, nodes and external systems in a cluter. There are four types of network communication problems to take into account [19]:

1. container-to-container – resolved by sharing network resources inside a pod
2. Pod-to-Pod –
3. Pod-to-Service – covered service type ClusterIP
4. External-to-Service – held by services type NodePort and Loadbalancer

Tekst

2.5. Container Network Interface (CNI)

2.6. Overview of Selected CNI Plugins

2.7. Related Work

DoTekst Na stronie http://kile.sourceforge.net/screenshots.php Tekst *Kile*, Tekst

Tekst

– Tekst

Bibliography

[1] Redhat Inc. What is containerization? https://www.redhat.com/en/topics/cloud-native-apps/what-is-containerization. Accessed, 07-Dec-2024.

[2] Docker Inc. What is an image? https://docs.docker.com/get-started/docker-concepts/building-images/. Accessed, 07-Dec-2024.

[3] The Kubernetes Authors. Overview. https://kubernetes.io/docs/concepts/overview/. Accessed, 07-Dec-2024.

[4] The Kubernetes Authors. Cluster Architecture. https://kubernetes.io/docs/concepts/architecture/. Accessed, 07-Dec-2024.

[5] etcd Authors. Data model. https://etcd.io/docs/v3.5/learning/data_model/. Accessed, 08-Dec-2024.

[6] The Kubernetes Authors. Kubernetes Scheduler. https://kubernetes.io/docs/concepts/scheduling-eviction/kube-scheduler/. Accessed, 08-Dec-2024.

[7] The Kubernetes Authors. Container Runtimes. https://kubernetes.io/docs/setup/production-environment/container-runtimes/. Accessed, 08-Dec-2024.

[8] The Kubernetes Authors. Namespaces. https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/. Accessed, 10-Dec-2024.

[9] The Kubernetes Authors. Pods. https://kubernetes.io/docs/concepts/workloads/pods/. Accessed, 08-Dec-2024.

[10] The Kubernetes Authors. ReplicaSet. https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/. Accessed, 10-Dec-2024.

[11] The Kubernetes Authors. Deployments. https://kubernetes.io/docs/concepts/workloads/controllers/deployment/. Accessed, 10-Dec-2024.

[12] The Kubernetes Authors. DaemonSet. https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/. Accessed, 10-Dec-2024.

[13] The Kubernetes Authors. StatefulSets. https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/. Accessed, 10-Dec-2024.

[14] The Kubernetes Authors. Jobs. https://kubernetes.io/docs/concepts/workloads/controllers/job/. Accessed, 10-Dec-2024.

[15] The Kubernetes Authors. CronJob. https://kubernetes.io/docs/concepts/workloads/controllers/cron-jobs/. Accessed, 10-Dec-2024.

[16] The Kubernetes Authors. Service. https://kubernetes.io/docs/concepts/services-networking/service/. Accessed, 10-Dec-2024.

[17] The Kubernetes Authors. Ingress. https://kubernetes.io/docs/concepts/services-networking/ingress/. Accessed, 10-Dec-2024.

[18] The Kubernetes Authors. Ingress Controllers. https://kubernetes.io/docs/concepts/services-networking/ingress-controllers/. Accessed, 10-Dec-2024.

[19] The Kubernetes Authors. Cluster Networking. https://kubernetes.io/docs/concepts/cluster-administration/networking/. Accessed, 10-Dec-2024.