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[7]:

- 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[6].

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 [7].

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"" [4]. K8s is open-source orchestration platform capable of managing containers [4]. Key functionalities are [4]:

- 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 [1].

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

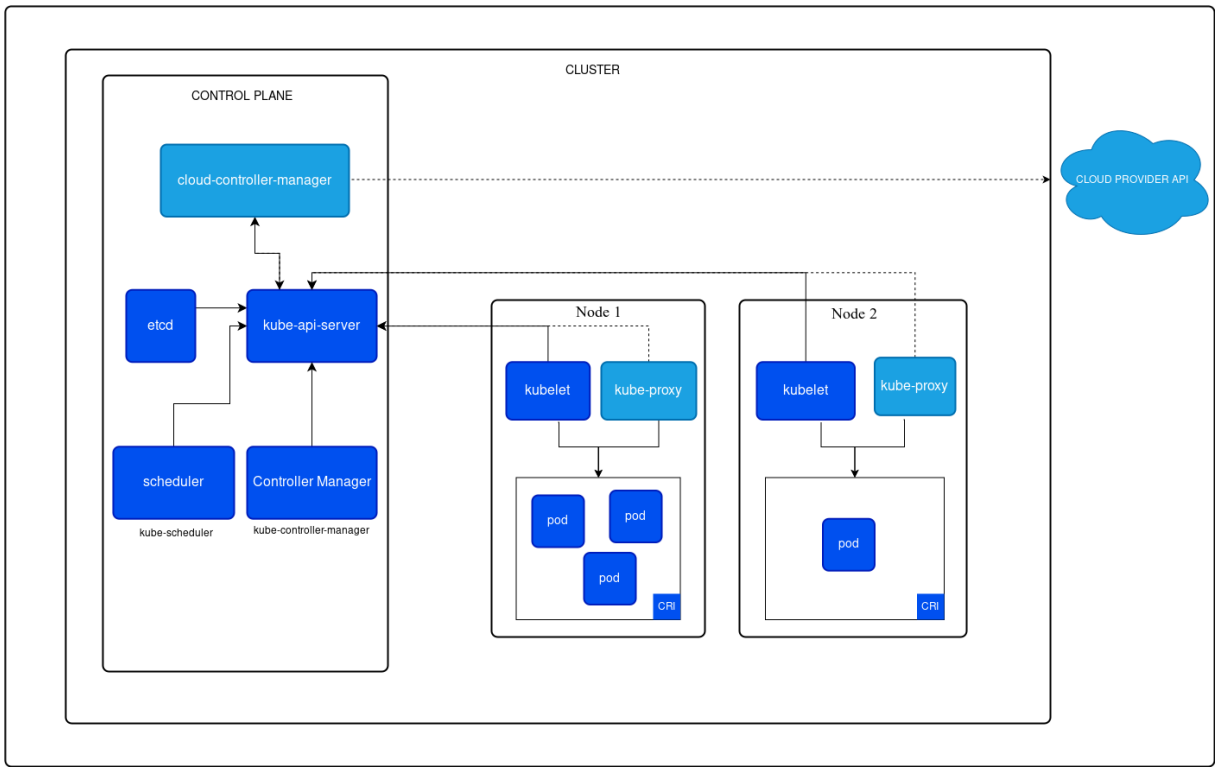

Figure 2.1: Kubernetes Cluster Architecture [1]

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 [1]. This is example of "Designed for extensibility", where Kubernetes can acquire 3rd-party features without changing its source code [4].

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 [1].

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 [1][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 [1].

kube-controller-manager

Component which run controller processes. Its compiled binary consists of multiple controllers. Example controllers are [1]:

- 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 [3].

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 [1].

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. [1].

kubelet

It is responsible for managing containers inside pod on its node. Uses Container Runtime Interface to communicate with containers [1] [2].

2.3.3. Objects

2.3.4. Interfaces

TEST

TEKST asdasdasda

2.4. Cluster Networking

Tekst

1. Tekst
2. Tekst

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

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

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

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

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

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

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