Kubernetes basics

Discussed topics

- Containers orchestration
- Kubernetes
- Kubernetes architecture components
- Pods
- Pod basic commands
- Deployments
- Daemonsets, Jobs & CronJobs
- Labels & filtering
- Namespaces
- Services

Containers orchestration

Containers

In order to follow this presentation, basic understanding of containers and docker usage is required.

If this kind of knowledge is missing, the reader is invited to first, check the containers presentation before proceeding any further.

Containers limits

When working with containers multiple limits can be observed:

- What happens if the host machine is unstable ?
- What happens if the containers use all of the available cpu or memory ?
- Which machine is best suited to run the containers ?
- What if one container is no longer enough to handle the load ?
- What is responsible for restarting an unhealthy container ?
- What is an unhealthy container ?
- How to access a given service in a container ?
- How to define communication rules between containers ?

Containers orchestration

Containers orchestration is the automated management of the operational loads related to running containers.

Examples of what this operational load includes:

- Scheduling
- Scaling
- Health monitoring
- Network access

Kubernetes

Kubernetes

Kubernetes is the most popular container orchestration system.

Kubernetes is open-source & each public cloud providers offers his own managed service:

- GCP: Google Kubernetes Engine (GKE & GKE autopilot)
- AWS: Elastic Kubernetes Service (EKS)
- Microsoft Azure: Azure Kubernetes Service (AKS)

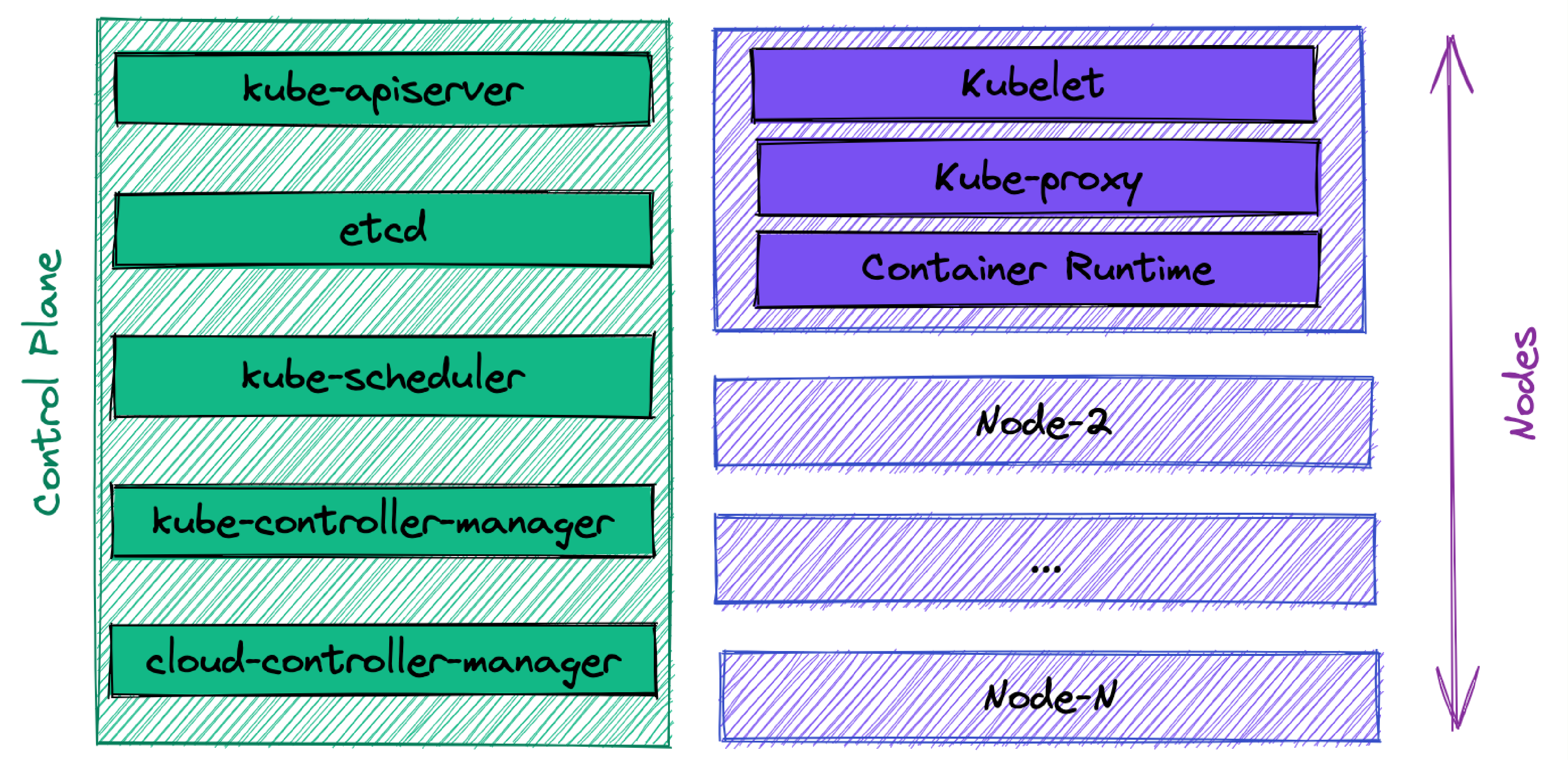
Kubernetes features

Some of kubernetes basic features are:

- Running containers on a cluster (multiple machines connected by network)
- Containers health monitoring & auto recovery
- Node health monitoring & container re-scheduling
- Service discovery

Kubernetes architecture

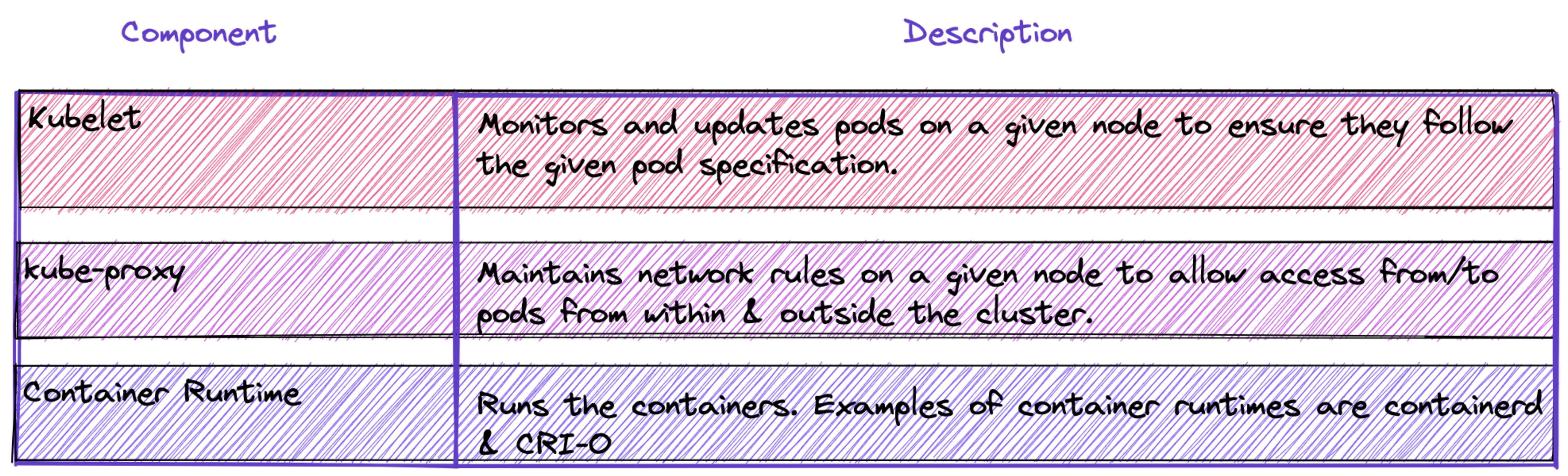
Kubernetes components (1/3)



Kubernetes components (2/3)

Component	Description
kube-apiserver	API: CRUD operations on kubernetes objects
etcd	Database: Key/value store to hold cluster data
kube-scheduler	Scheduler: responsible for scheduling pods & assigning them to nodes
kube-controller-manager	Controller aggregation: -> Nodes: supervise nodes for availability -> Endpoints: handles endpoints -> ...
cloud-controller-manager	Controller: responsible for interactions with cloud APIs -> Request an additional disk -> Reduce the number of machines -> ...

Kubernetes components (3/3)



Component	Description
Kubelet	Monitors and updates pods on a given node to ensure they follow the given pod specification.
kube-proxy	Maintains network rules on a given node to allow access from/to pods from within & outside the cluster.
Container Runtime	Runs the containers. Examples of container runtimes are containerd & CRI-O

Pods

Pods

Pods are units containing one or multiple containers.

Pods are kubernetes objects that are created by calling the kubernetes API.

Pods follow the structure below:

```yaml
apiVersion: v1
kind: Pod
metadata:
  # ... ignored for now
  name: nginx # Pod name
spec:
  containers:
  - image: nginx:stable # Container image
    name: nginx # Container name
  # ... ignored for now
```

Running a simple Pod

In order to run a simple pod we can do the following:

```
$ cat hands-on/yaml-samples/pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-from-yaml
spec:
  containers:
  - image: nginx:stable
    name: nginx-from-yaml
$ kubectl apply -f hands-on/yaml-samples/pod.yaml
pod/my-nginx created
```

In docker, this would have been equivalent to:

```
$ docker run -d -it nginx:stable
```

Pod basic commands

Please proceed to hands-on pod-basic-commands.md.

Deployments

Deployments can be viewed as managed pods.

In order to better understand the previous statement, please proceed to hands-on pods-vs-deployments.md.