Kubernetes (k8s)

Agenda

1. What is Kubernetes?
2. Kubernetes Concepts
3. Hands on Lab
4. Kubernetes – Architecture
5. Ressource Management

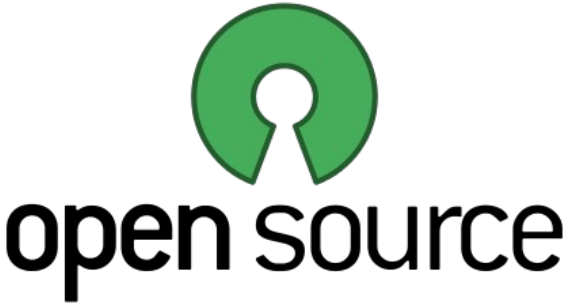
1

Kubernetes – What is it?

Kubernetes – What is it?

CLOUD NATIVE COMPUTING FOUNDATION

Kubernetes is a powerful open-source system, initially developed by Google, for managing containerized applications in a clustered environment. It aims to provide better ways of managing related, distributed components and services across varied infrastructure.



Google

Certified Kubernetes

1. **Consistency** users want consistency when interacting with any installation of Kubernetes.
2. **Timely Updates** vendors need to provide the latest version of Kubernetes yearly or more frequently, so you can be sure that you'll always have access to the latest features the community has been working hard to deliver.
3. **Confirmability** any end user can confirm that their distribution or platform remains conformant by running the identical open source conformance application (Sonobuoy) that was used to certify.

RANCHER

RED HAT® OPENSHIFT

https://www.cncf.io/certification/software-conformance/

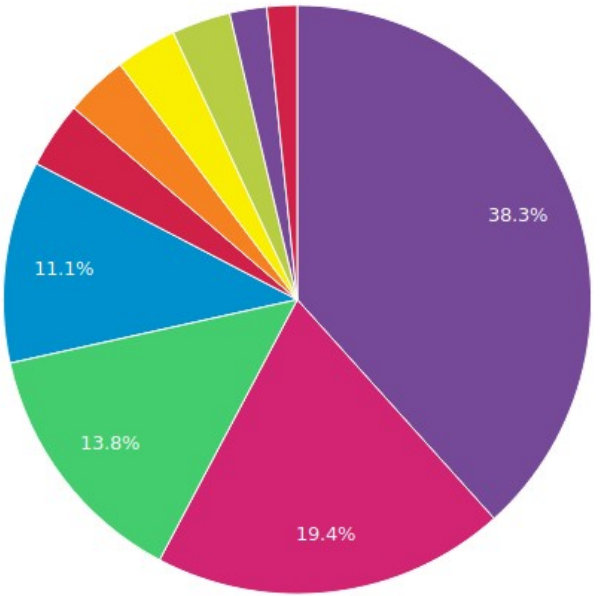
Kubernetes Community Contributions

Commits by Company

Show 10 entries Search

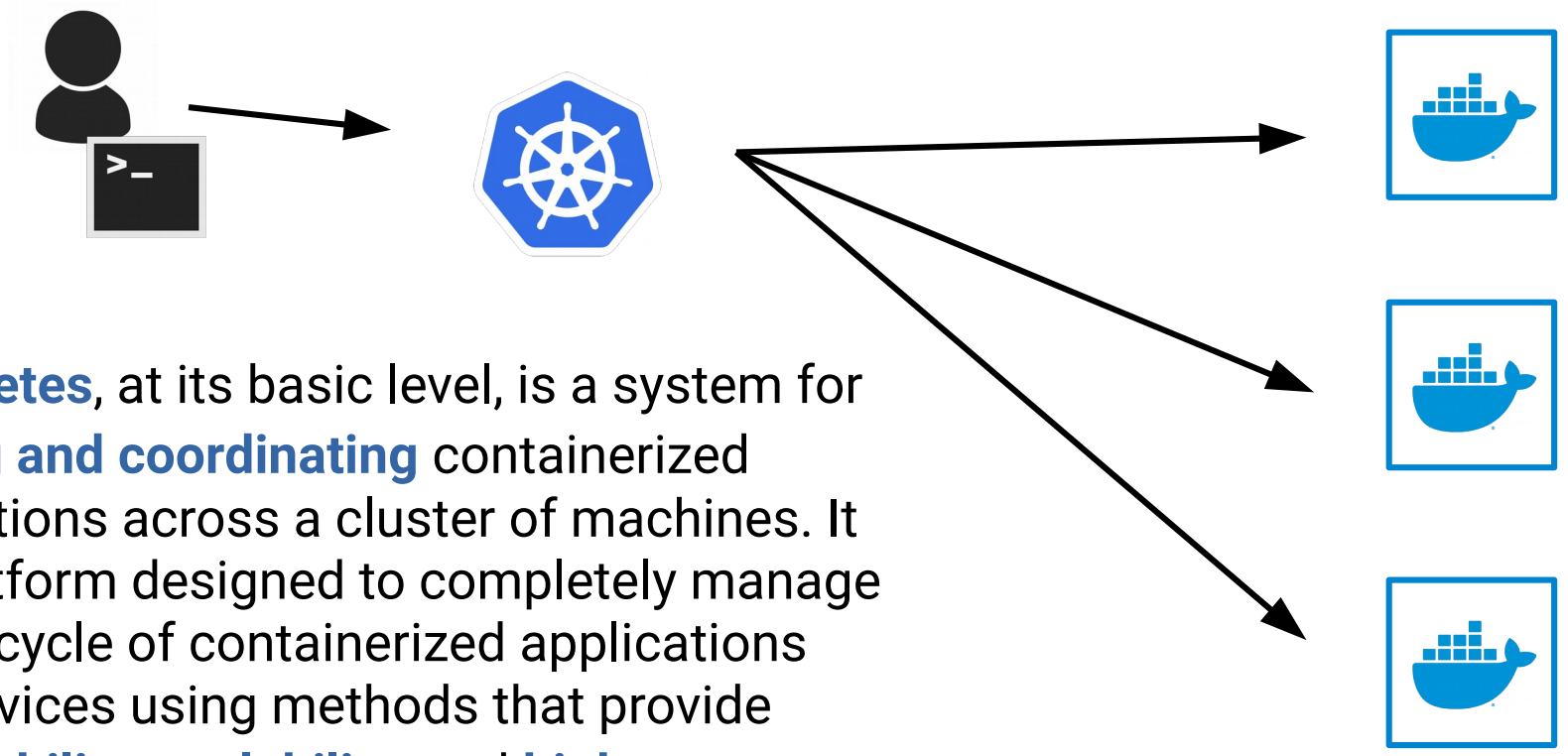
#	Company	Commits
	*independent	10780
1	Google	5462
2	Red Hat	3124
3	IBM	1021
4	VMware	965
5	Microsoft	946
6	Huawei	910
7	Ticket Master	571
8	Mirantis	467
9	Amazon	449

Showing 1 to 10 of 116 entries Previous Next



https://www.stackalytics.com/?release=all&project_type=kubernetes-group&metric=commits

Kubernetes – What is it?



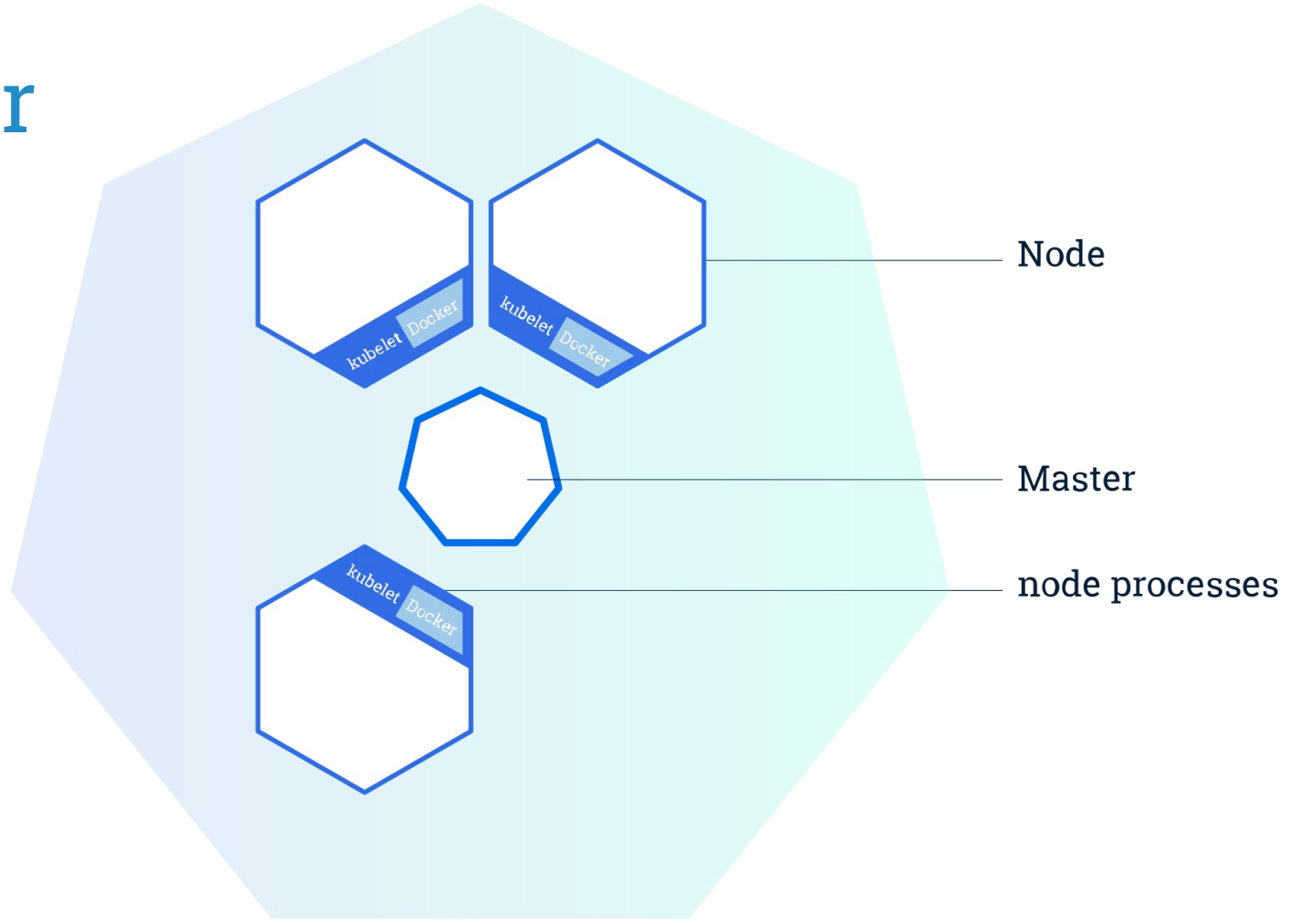
Kubernetes, at its basic level, is a system for **running and coordinating** containerized applications across a cluster of machines. It is a platform designed to completely manage the life cycle of containerized applications and services using methods that provide **predictability**, **scalability**, and **high availability**.

2

Kubernetes – Concepts

Kubernetes Objects

Cluster



Kubernetes Objects - 1

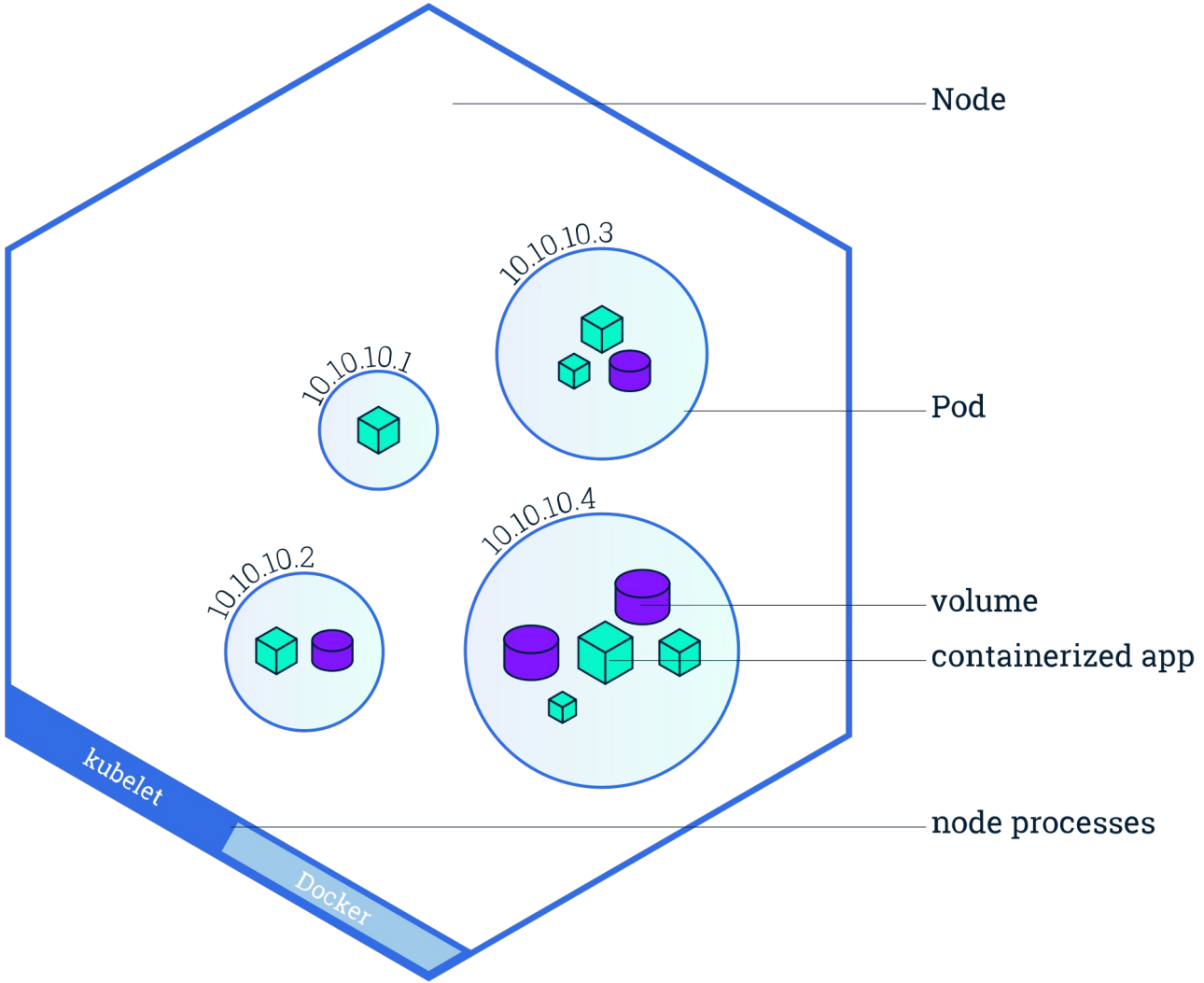
Pod is the smallest deployable unit on a Node. It's a group of containers which must run together. Quite often, but not necessarily, a Pod usually contains one container. https://kubernetes.io/docs/concepts/workloads/pods/pod-overview/

Service is used to define a logical set of Pods and related policies used to access them. https://kubernetes.io/docs/concepts/services-networking/service/

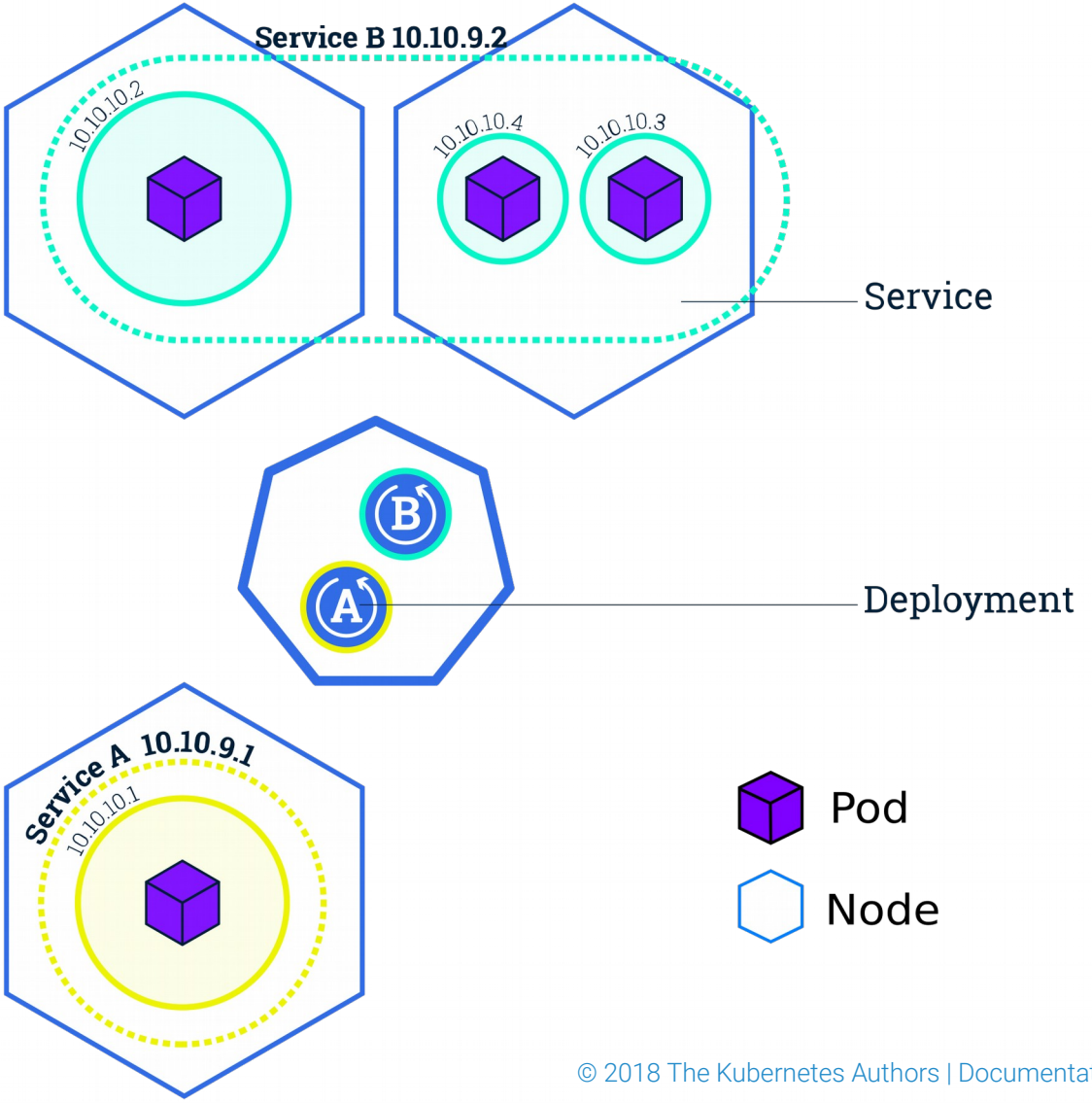
Volume is essentially a directory accessible to all containers running in a Pod. https://kubernetes.io/docs/concepts/storage/volumes/

Namespaces are virtual clusters backed by the physical cluster. https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/

Pods



Services

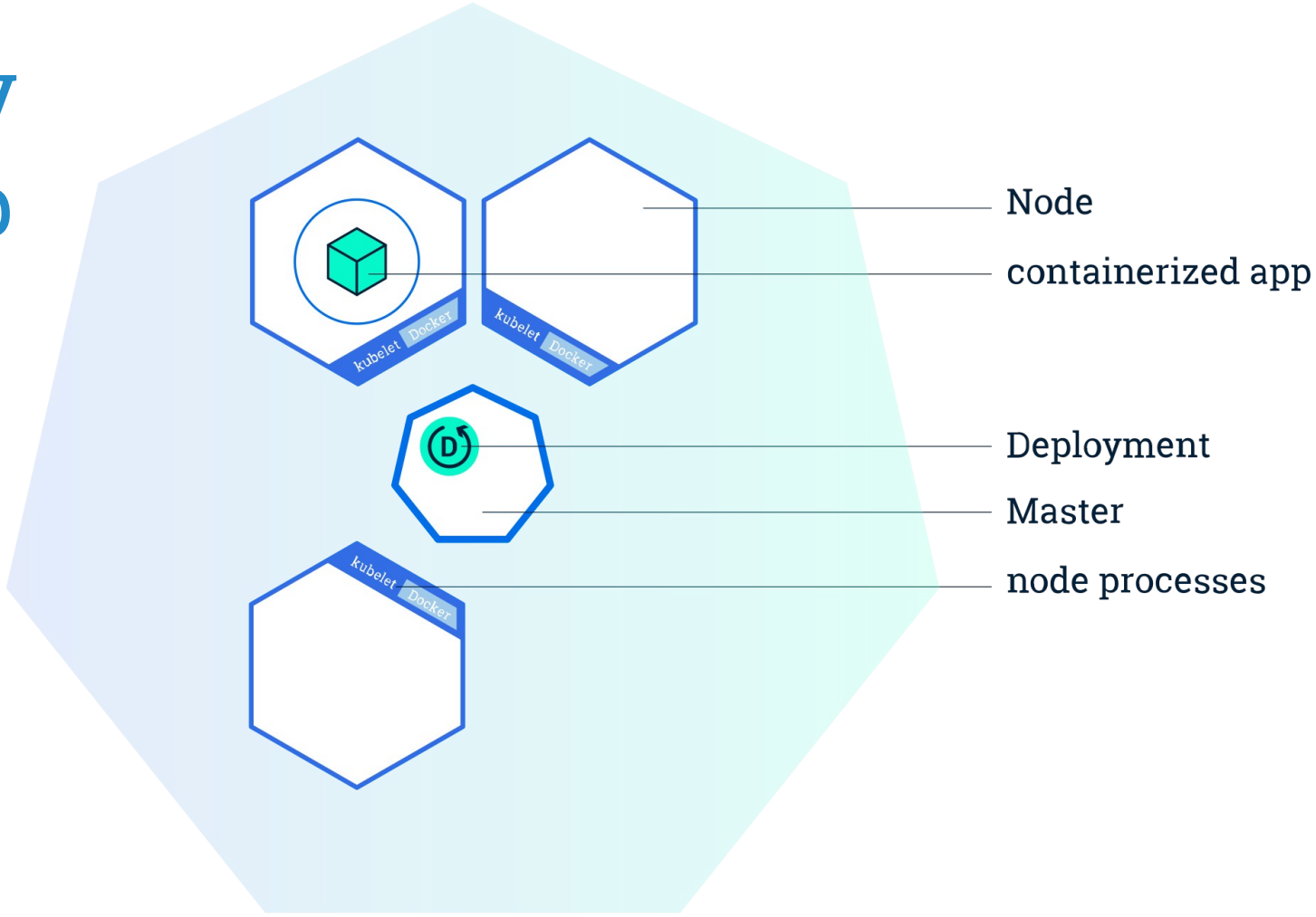


Kubernetes Objects - 2

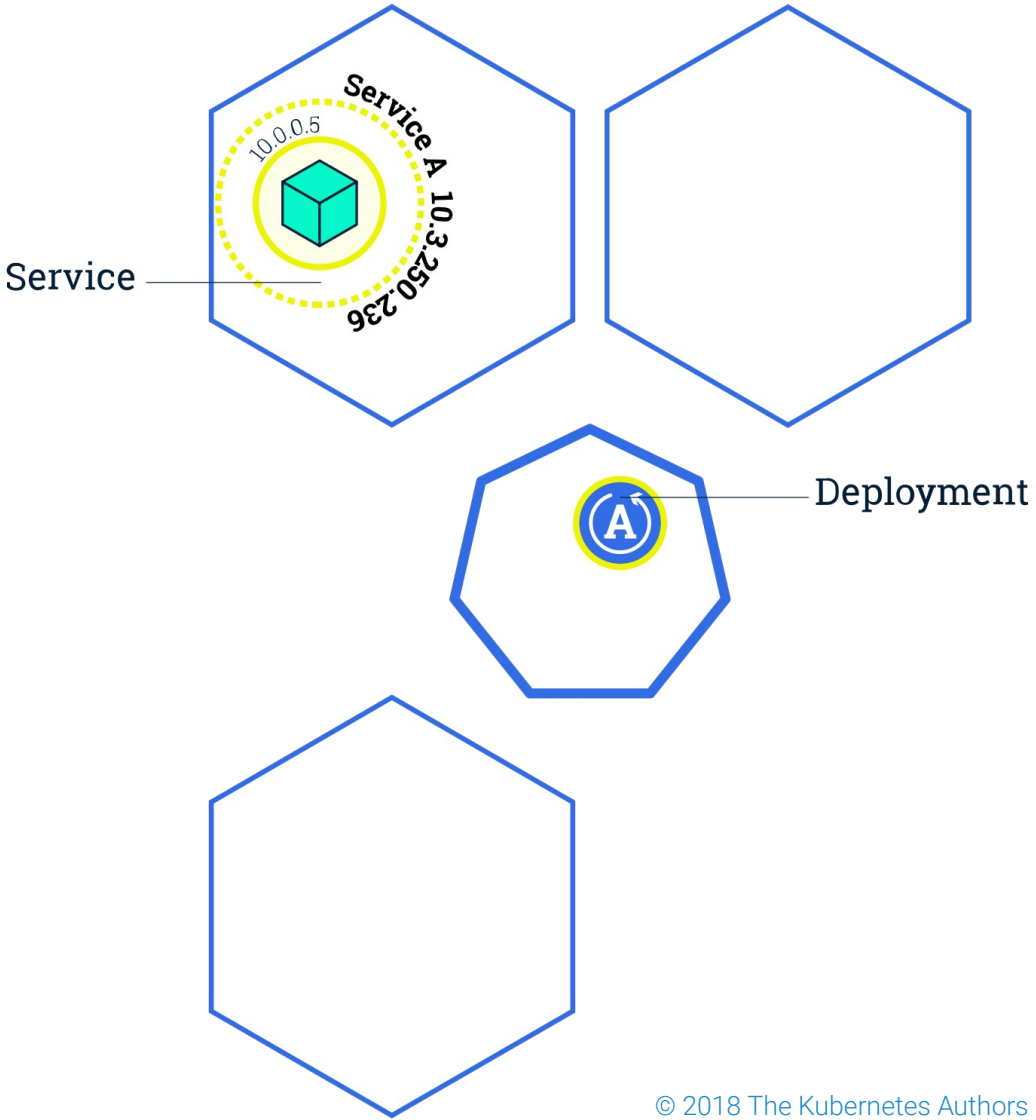
Replication Set is an object that defines a pod template and control parameters to scale identical replicas of a pod horizontally by increasing or decreasing the number of running copies.
https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/

Deployments are one of the most common workloads to directly create and manage. Deployments use replication sets as a building block, adding flexible life cycle management functionality to the mix.
https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

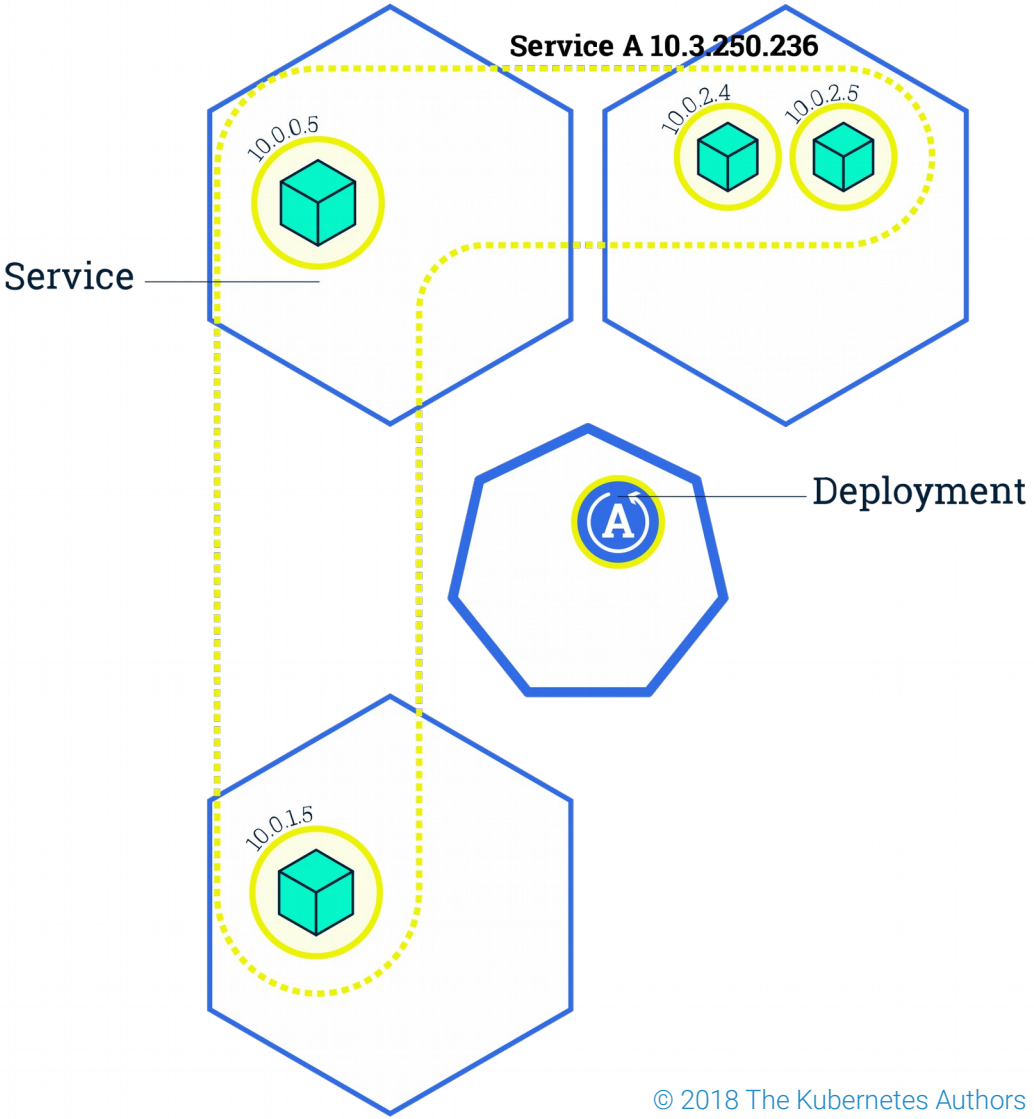
Deploy an app



Scaling



Scaling



Communication

Kubernetes – Communication

- Highly-coupled container to container communication
- Pod to pod communication
- Pod to service communication
- External to service communication

Kubernetes – Network Model

- All containers can communicate with each other without NAT
- All nodes can communicate with containers without NAT
- The IP address a container sees for itself is the same address everyone else sees

https://kubernetes.io/docs/concepts/cluster-administration/networking/#the-kubernetes-network-model

Perstistent Storage

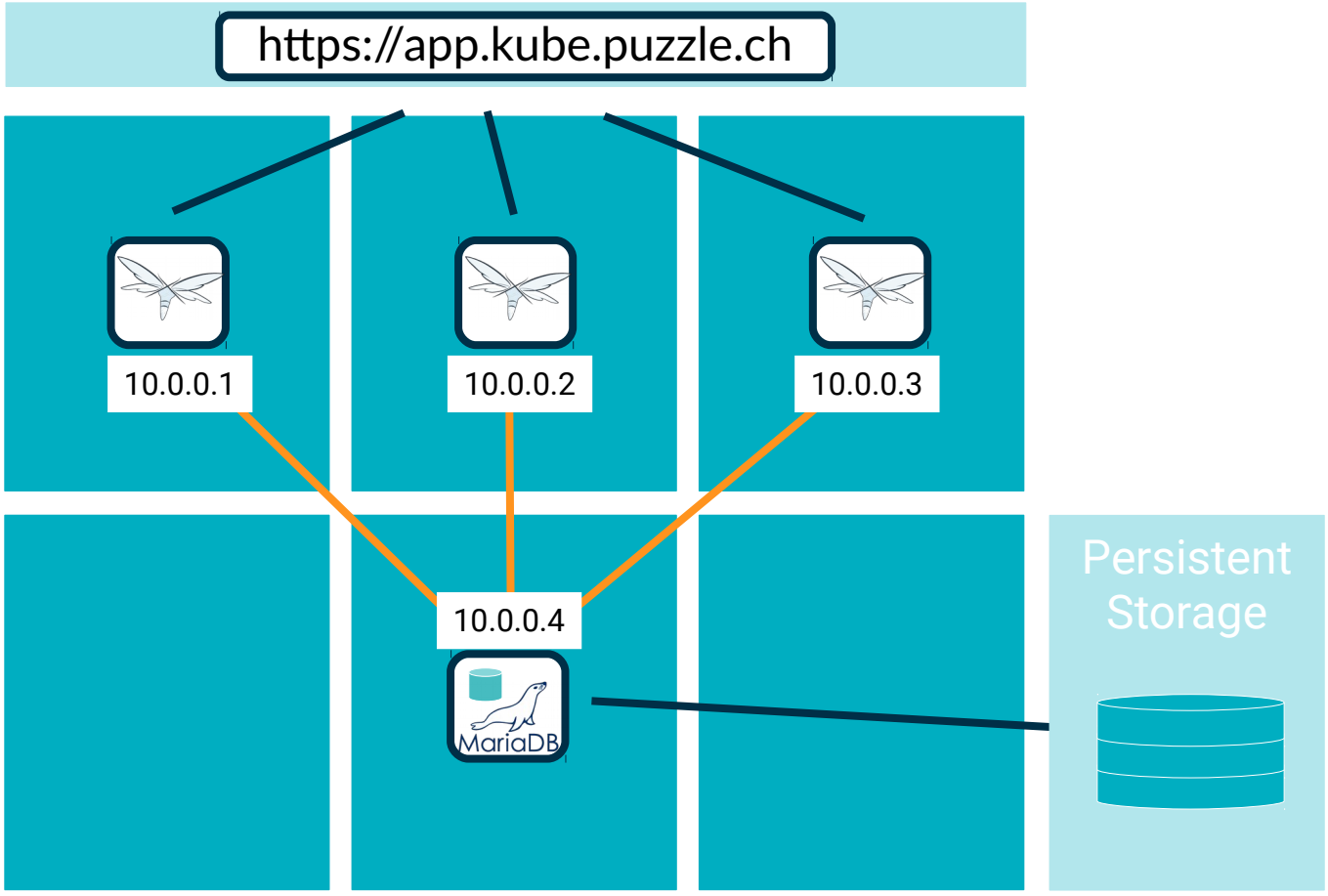
Persistent Storage

A PersistentVolume (PV) is a piece of storage in the cluster that has been provisioned by an administrator or dynamically provisioned using Storage Classes. It is a resource in the cluster just like a node is a cluster resource. PVs are volume plugins like Volumes, but have a lifecycle independent of any individual pod that uses the PV. This API object captures the details of the implementation of the storage, be that NFS, iSCSI, or a cloud-provider-specific storage system.

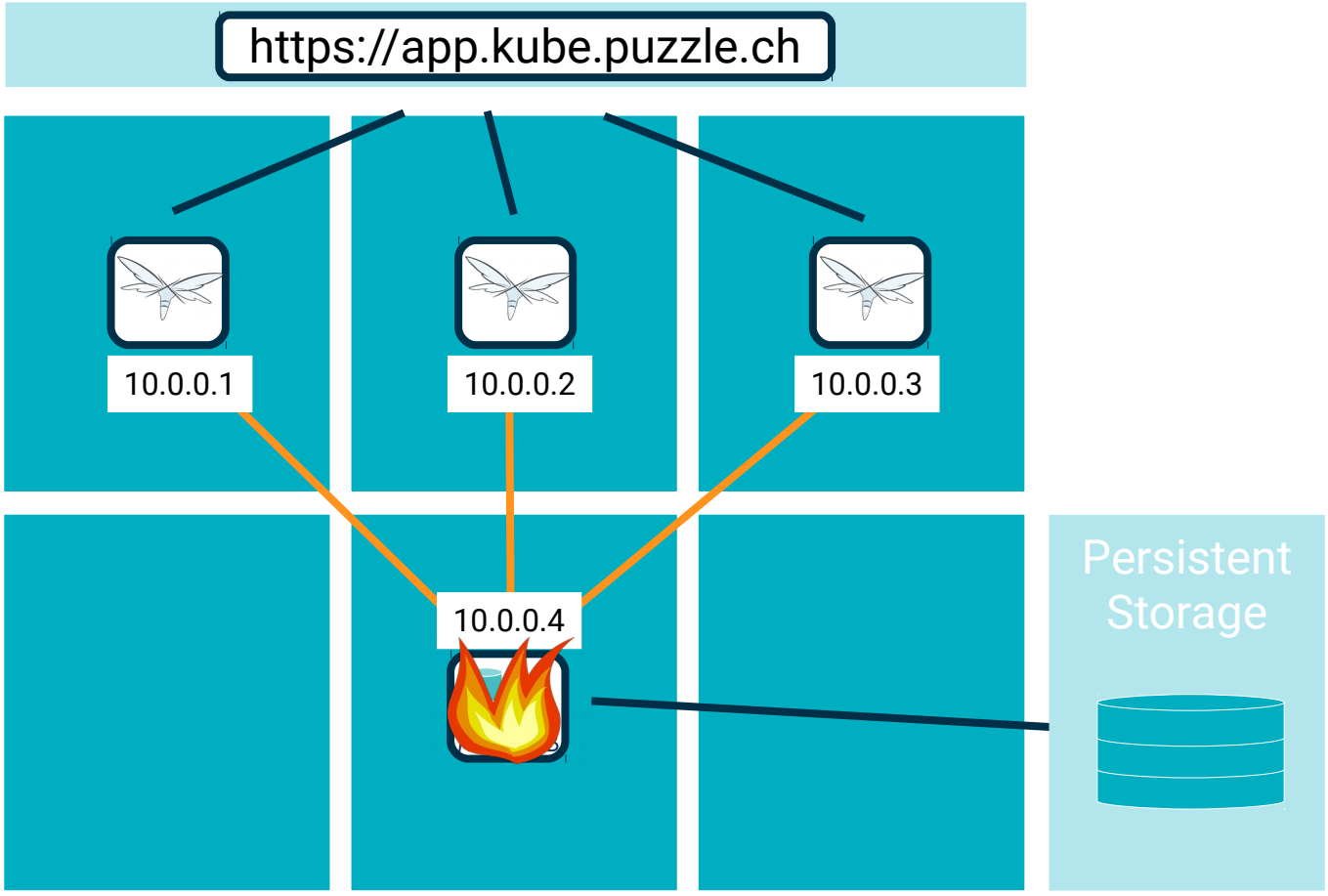
A PersistentVolumeClaim (PVC) is a request for storage by a user. It is similar to a pod. Pods consume node resources and PVCs consume PV resources. Pods can request specific levels of resources (CPU and Memory). Claims can request specific size and access modes (e.g., can be mounted once read/write or many times read-only).

https://kubernetes.io/docs/concepts/storage/persistent-volumes/

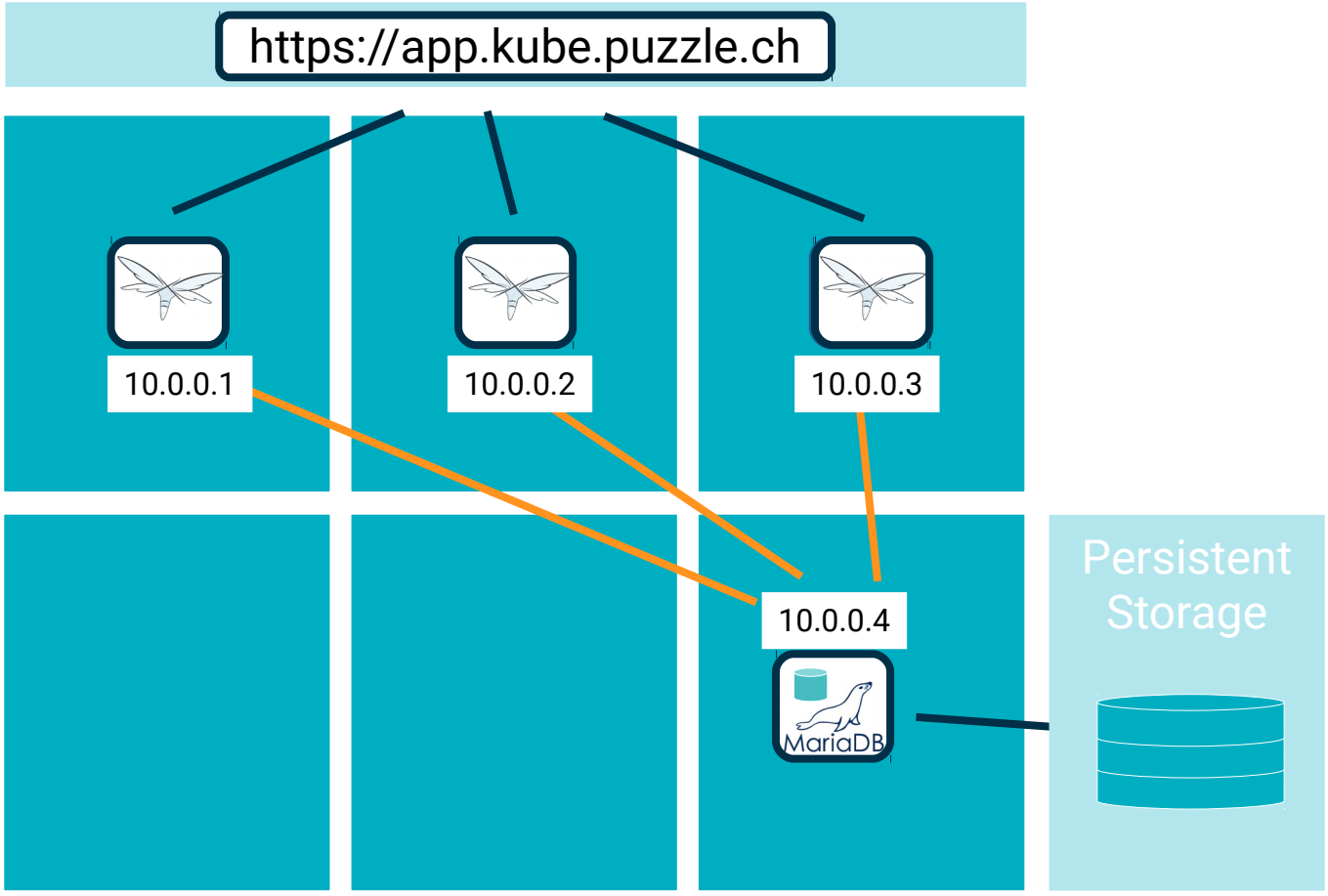
Persistent storage example



Persistent storage example



Persistent storage example



RedHat OpenShift vs. vanilla Kubernetes

RedHat OpenShift vs. Kubernetes

- RedHat OpenShift only on RHEL, Kubernetes on any OS
- OpenShift templates are less flexible than Kubernetes Helm charts
- Routers on OpenShift vs. Ingress on Kubernetes
  - OpenShift with HAProxy or F5 BIG-IP
  - Kubernetes with more Options (NGINX, traefik, HAProxy etc)
- A different approach to deployments. *Deployment (Controller)* vs. *DeploymentConfig* (controller by ded. Pods, Hooks/Trigger)

RedHat OpenShift vs. Kubernetes

- OpenShift has more strict security policies than default Kubernetes
  - Most of container images available on Docker Hub won't run on OpenShift, as it forbids to run a container as root and even many of official images don't meet this requirement.
- Better management of container images
  - ImageChange Trigger, chained builds
- Integrated Image Registry
- Bundeld with Logging & Monitoring

RedHat OpenShift vs. Kubernetes

- OpenShift projects are more than Kubernetes namespaces
  - trivial things such as description and display name
  - Default Objects (Roles, Default Quotas, etc)
- OpenShift is easier for beginners
  - oc has support for logging to OpenShift cluster - with kubectl you need to obtain your credentials and create your kubeconfig
  - oc lets you switch between projects/namespaces while kubectl doesn't (you need exernal tools such as *kubens*/*kubectx*)
  - oc allows you to build a container image from a source and then deploy it onto environments with a single command! (*oc new-app*)

RedHat OpenShift vs. Kubernetes

- Integrated CI/CD with Jenkins
- Source-to-image (S2I) / Build Pipeline

3

Hands On Lab

Hand on Lab

URLs

https://github.com/puzzle/kubernetes-techlab/tree/rancherversion
https://github.com/puzzle/kubernetes-techlab/archive/rancherversion.zip
https://rancher.puzzle.ch

Accounts

Username : LDAP Account
Password : ...
Project : Team [1-15]
Cluster : techlab

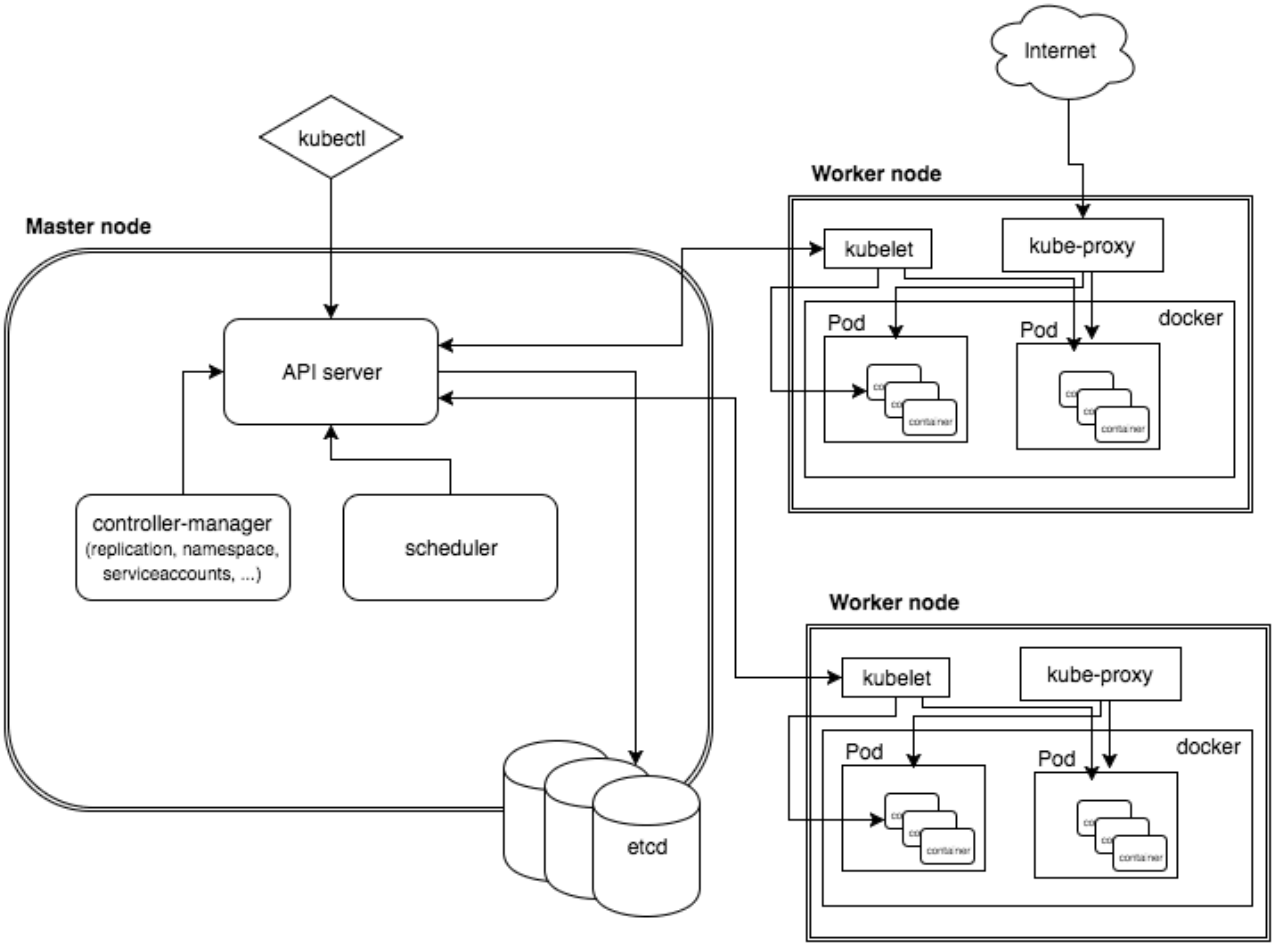
Remarks

* Create Namespaces via Rancher WebGUI, otherwise they are not assigned to your project (or ask your teacher to assign the namespace to your project)

4

Kubernetes – Architecture

High Level Architecture



Master Node

The master node is responsible for the management of Kubernetes cluster. This is the entry point of all administrative tasks. The master node is the one taking care of orchestrating the worker nodes, where the actual services are running.

Master Node – Components - 1

API Server

The API server is the entry points for all the REST commands used to control the cluster. It processes the REST requests, validates them, and executes the bound business logic. The result state has to be persisted somewhere, and that brings us to the next component of the master node.

Etcd storage

etcd is a simple, distributed, consistent key-value store. It provides a REST API for CRUD operations as well as an interface to register watchers on specific nodes, which enables a reliable way to notify the rest of the cluster about configuration changes.

Master Node – Components - 2

scheduler

The deployment of configured pods and services onto the nodes happens thanks to the scheduler component.

The scheduler has the information regarding resources available on the members of the cluster, as well as the ones required for the configured service to run and hence is able to decide where to deploy a specific service.

controller-manager

A controller uses apiserver to watch the shared state of the cluster and makes corrective changes to the current state to change it to the desired one.

An example of such a controller is the Replication controller, which takes care of the number of pods in the system. The replication factor is configured by the user, and it's the controller’s responsibility to recreate a failed pod or remove an extra-scheduled one.

Worker Node

The pods are run here, so the worker node contains all the necessary services to manage the networking between the containers, communicate with the master node, and assign resources to the containers scheduled.

Worker Node – Components - 1

Docker

Docker runs on each of the worker nodes, and runs the configured pods. It takes care of downloading the images and starting the containers.

kubelet

kubelet gets the configuration of a pod from the apiserver and ensures that the described containers are up and running. This is the worker service that’s responsible for communicating with the master node.

It also communicates with etcd, to get information about services and write the details about newly created ones.

Worker Node – Components - 2

kube-proxy

kube-proxy acts as a network proxy and a load balancer for a service on a single worker node. It takes care of the network routing for TCP and UDP packets.

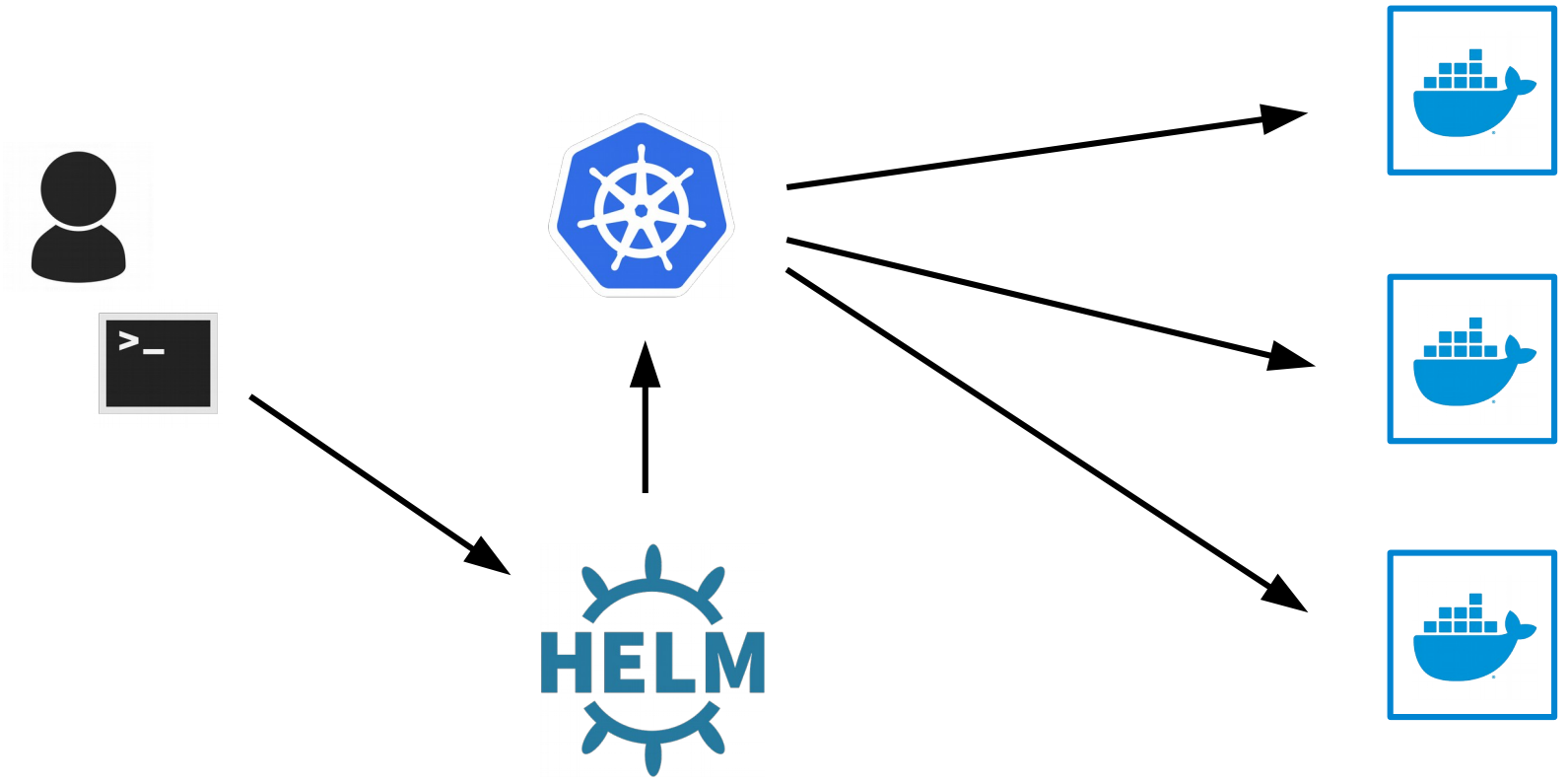
kubectl

And the final bit – a command line tool to communicate with the API service and send commands to the master node.

5

Resource Management

Kubernetes – Helm



Kubernetes – Helm

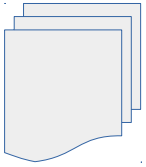
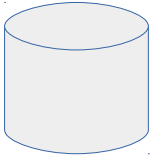
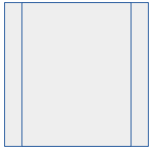


Chart: Kubernetes resources package



Repository: Collection of public charts



Release: Load Chart Instance

Kubernetes – Kustomize

```
~/someApp
├── base
│   ├── deployment.yaml
│   ├── kustomization.yaml
│   └── service.yaml
└── overlays
    ├── development
    │   ├── cpu_count.yaml
    │   ├── kustomization.yaml
    │   └── replica_count.yaml
    └── production
        ├── cpu_count.yaml
        ├── kustomization.yaml
        └── replica_count.yaml
```

Thank you!