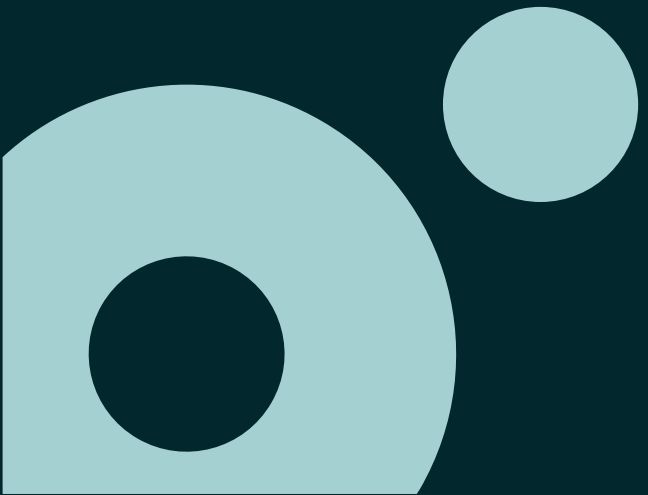
Containers & Kubernetes

Session #05



Container Orchestration

Kubernetes: Architecture

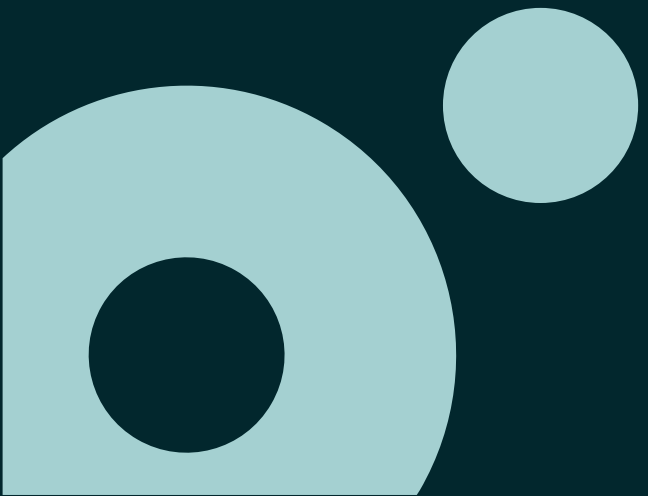
Kubectl

Pods

Namespaces

Lab

Container Orchestration



Motivation

Container Orchestration

- Containers brings benefits on development process, tests and deployment
- Portability between environments
- Higher productivity (less configuration needs)
- Less footprint with bigger density on hardware
- Resources isolation
- But brings several challenges to manage and operate!

Motivation

Container Orchestration

- All management, maintenance and operation can be done manually but could be a crazy task!
- To have more agility, automation and ease on these tasks, **orchestration** is the key
- Main orchestration features
  - Scheduling
  - Affinity
  - Monitoring
  - Failover
  - Scalability
  - Networking
  - Service Descovery
  - Application upgrades

Motivation

Container Orchestration

- Scheduling
  - Container provisioning using nodes metrics and containers requests
- Affinity
  - Specific configuration for provisioning about availability/performance
- Monitoring
  - Detect and fix failures on a reactive/preventive way
- Failover
  - Re-provision faulty instances
  - Re-provision instances to healthy machines

moOngy.

Motivation

Container Orchestration

- Scalability
  - Add/remove instances to meet demand
- Networking
  - Networking overlay for container communication
  - Allow inbound/outbound communication with the cluster
- Service Discovery
  - Enable containers to locate each other
- Application Upgrades
  - Avoid downtime and automatically rollback

Solutions

Container Orchestration

- Several options on the market
  - Docker Swarm
  - Apache Mesos
  - Hashicorp Nomad
  - Rancher
- Kubernetes is the de-facto orchestration solution on the market nowadays

Kubernetes

Container Orchestration

- Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation.
- 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".
- Google open-sourced the Kubernetes project in 2014. Donated to Cloud Native Computing Foundation (CNCF) in 2015

Kubernetes: Architecture

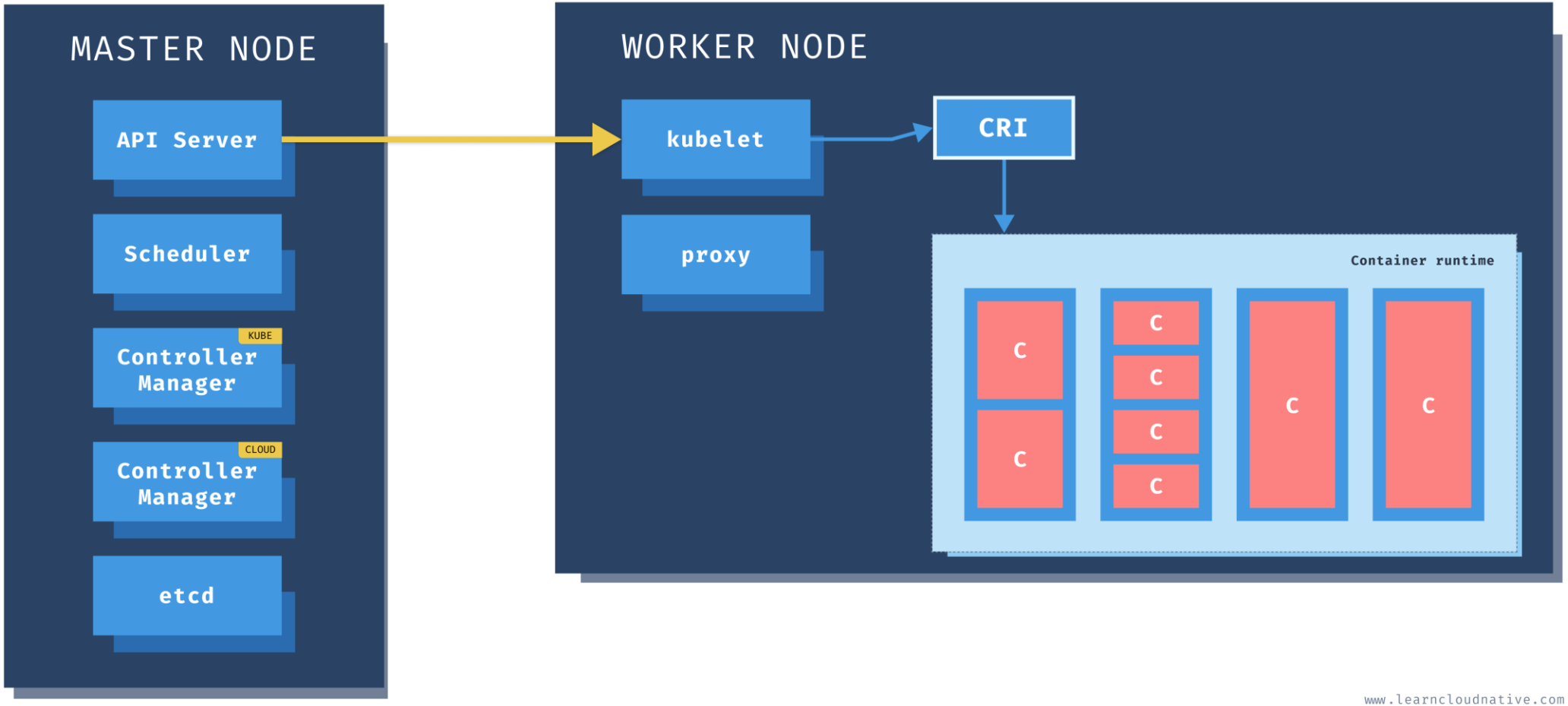
Kubernetes Cluster

Architecture

- A **Kubernetes cluster** consists of a set of machines (physical or virtual), called nodes
- **Master node(s)** (aka control plane) manages the worker nodes and the cluster
- **Worker node(s)** runs containerized workloads
- Worker nodes can be **heterogeneous** (small, large, GPUs, Linux, Windows, etc.)

Kubernetes Cluster

Architecture

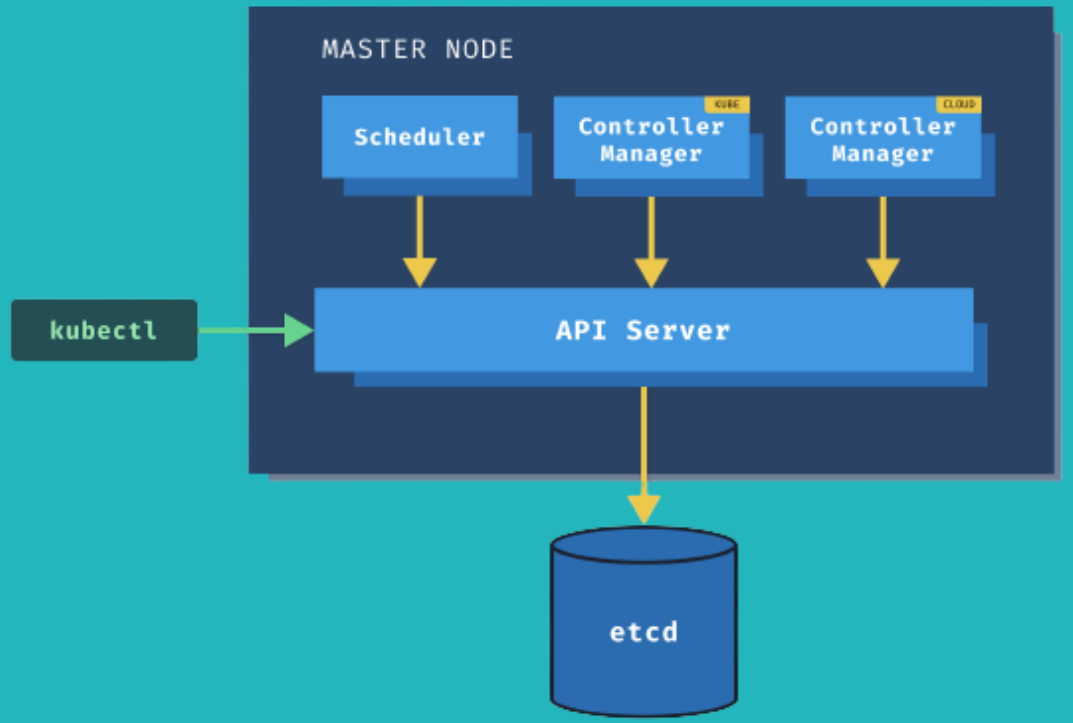


moOngy.

Master Nodes

Architecture

- **API Server:** exposes the Kubernetes API outside the cluster.
- **etcd:** Consistent and highly-available key value store used to store for all cluster data
- **Scheduler:** Watches for newly created Pods with no assigned node and selects a node for them to run on
- **Controller Manager:** Manages controller processes

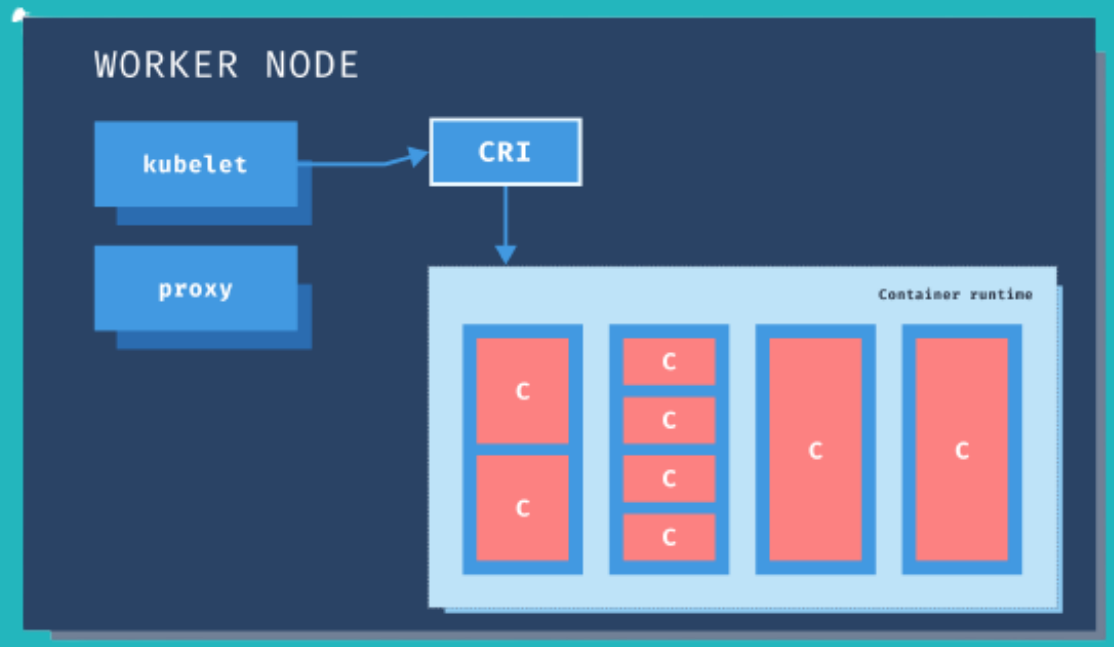


moOngy.

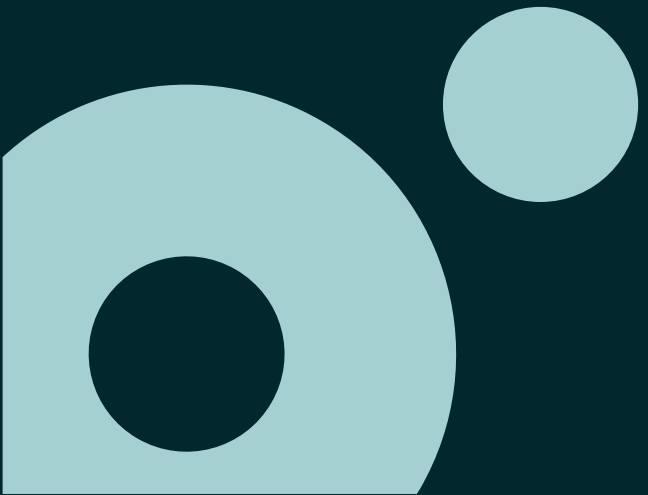
Worker Nodes

Architecture

- **Kubelet:** An agent that runs on each node in the cluster.
- **Kube-proxy:** A network proxy that runs on each node in your cluster, implementing part of the Kubernetes Service concept.
- **Container runtime (CRI):** The engine responsible for running containers.



Kubectl



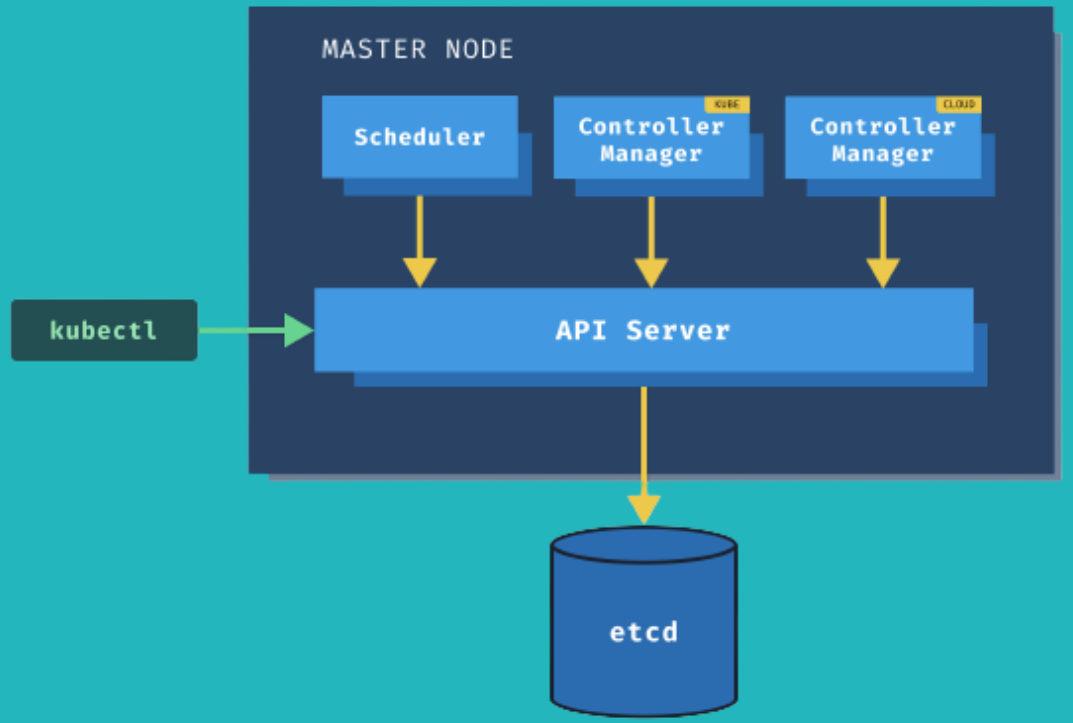
API Server

Kubectl

- Kubernetes is a “simple” REST API application, you can manage a cluster by making REST calls to the API Server.
- Example HTTP Request:

```
GET /api/v1/namespaces/default/pods/{name}
```

- However, it’s much easier to use the official Kubernetes client command-line utility → kubectl



How to use?

Kubectl

```
kubectl [command] [TYPE] [NAME] [flags]
```

- **`command`**: Operation that you want to perform on one or more resources, for example **`create`**, **`get`**, **`describe`**, **`delete`**.
- **`TYPE`**: Resource type. Case-insensitive and can specify the singular, plural, or abbreviated forms.
- **`NAME`**: Case-insensitive name of the resource. If the name is omitted, details for all resources are displayed
- **`flags`**: Optional flags. For example, **`-o`** allow to specify output type of the commands

Examples

Kubectl

- List all nodes

```
kubectl get nodes
```

- Get more details on node node01

```
kubectl describe node node01
```

- List all pods

```
kubectl get pods
```

- Delete pod pod-01

```
kubectl delete pod pod-01
```

- Execute a bash command in Pod pod-01

```
kubectl exec –it pod-01 -- bash
```

moOngy.

How to use?

Kubectl

- You can perform an action on several resource using only one command even on resources from different types
- Resources from same type

```
kubectl get pod pod-01 pod-02
```

- Resources from different types

```
kubectl get pod/pod-01 node/node01
```

kubectl Cheat Sheet | Kubernetes

Kubectl Reference Docs (kubernetes.io)

moOngy.

kubeconfig

Kubectl

- **`kubeconfig`** is a file used to organize access to several cluster usually stored at ~/.kube/config
- Needs to be kept on a secure place since have complete information about authn/authz of a user to a cluster
- This file should never be included on a repo or used for CI/CD process due to security reasons

kubeconfig

Kubectl

- Get all clusters configuration available

```
kubectl config view
```

- Get actual context

```
kubectl config current-context
```

- Set another context

```
kubectl config use-context my-cluster-name
```

moOngy. | Demo: Kubectl

Declarative Configuration

Kubectl

- An **imperative configuration** explicitly instructs a system on the steps to take to achieve a desired outcome (like using Docker commands):
  - Connect to container registry
  - Pull desired image
  - Create container
  - Start container
- A **declarative configuration** specifies a final, or desired state of an object, and lets the system determine what steps to take to achieve that state.
- The Kubernetes control plane continually and actively manages every object's **actual state to match the desired state** you supplied.

Declarative Configuration using YAML

Kubectl

- REST API applications like Kubernetes API Server, exchange data using JSON format
- When using kubectl, you provide a desired state configuration using the YAML markup language (Yet Another Markup Language)
- kubectl converts your YAML to JSON when communicate with the Kubernetes API Server
- YAML uses indentation instead nested curly brackets ({}) to create hierarchy.

Declarative Configuration using YAML

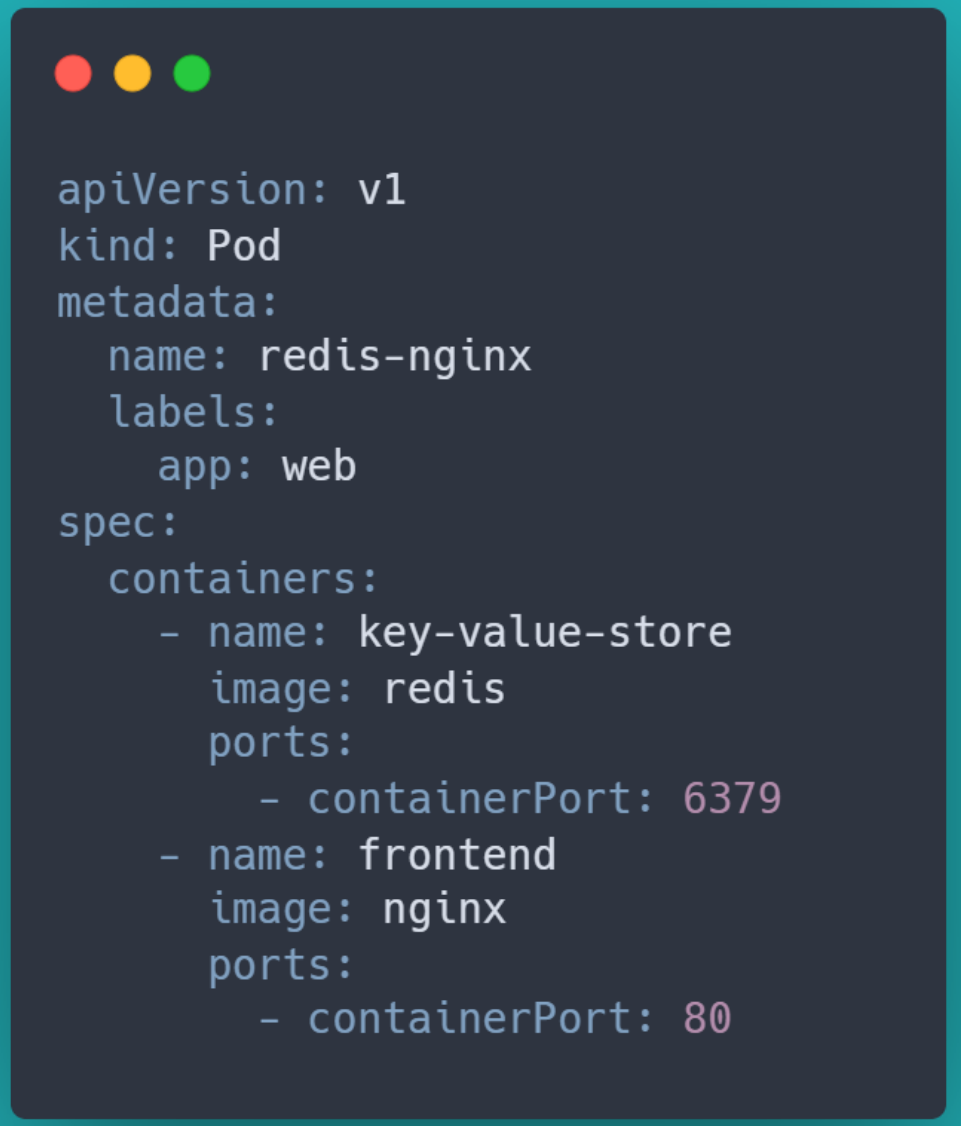
Kubectl

- Whitespace indentation is used for create file structure
- Tab characters are not allowed as part of that indentation
- Comments begin with the number sign(#) until the end of the line
- List members are denoted by a leading hyphen (-)
- An associative array entry is represented using colon space in the form key: value with one entry per line.
- Strings are ordinarily unquoted but may be enclosed in double-quotes ("), or single-quotes (‘).
- Multiple documents with single streams are separated with 3 hyphens (---).

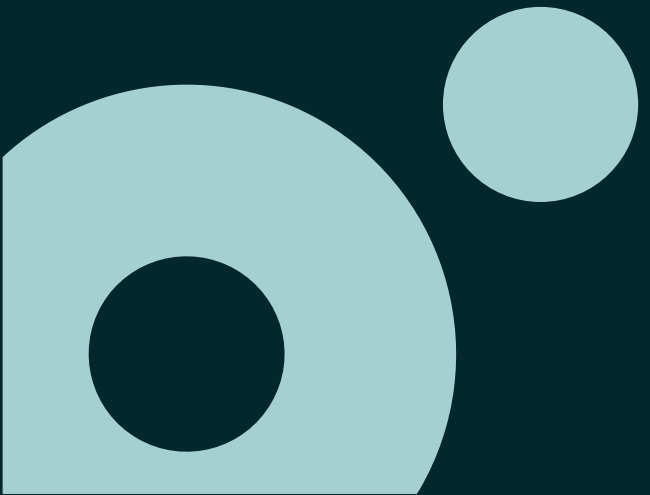
K8S Manifest File

Kubectl

- **apiVersion**: API group and version of the API you're calling to create this object
- **kind**: Object you want to create
- **metadata**: Data that helps uniquely identify the object, including a name string, UID, and optional namespace
- **spec** (most objects): Desired state for the object



Pods



What is a pod?

Pods

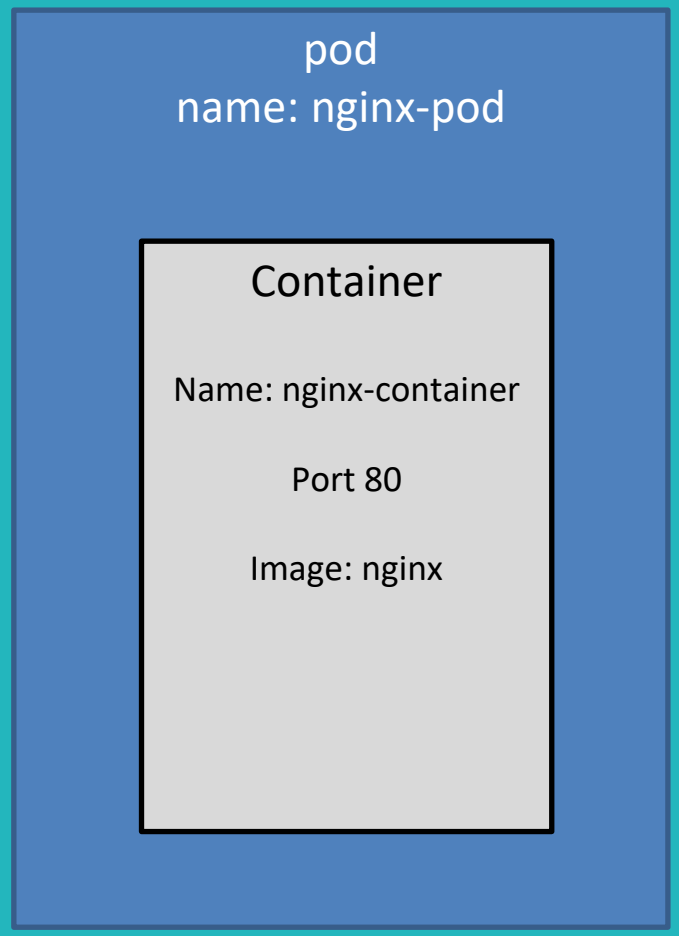
- Pods are the smallest deployable compute units you can create and manage in Kubernetes
- A Pod can manage one or more containers, with shared storage (volumes), environment variables, network resources, and a specification for how to run the containers
- Containers running in a Pod share the same IP and ports and communicate using native inter-process communication channels or localhost.
- Pods are immutable - if any change is made to the Pod specification (spec), a new Pod is created and then the old Pod is deleted

Lifecycle

Pods

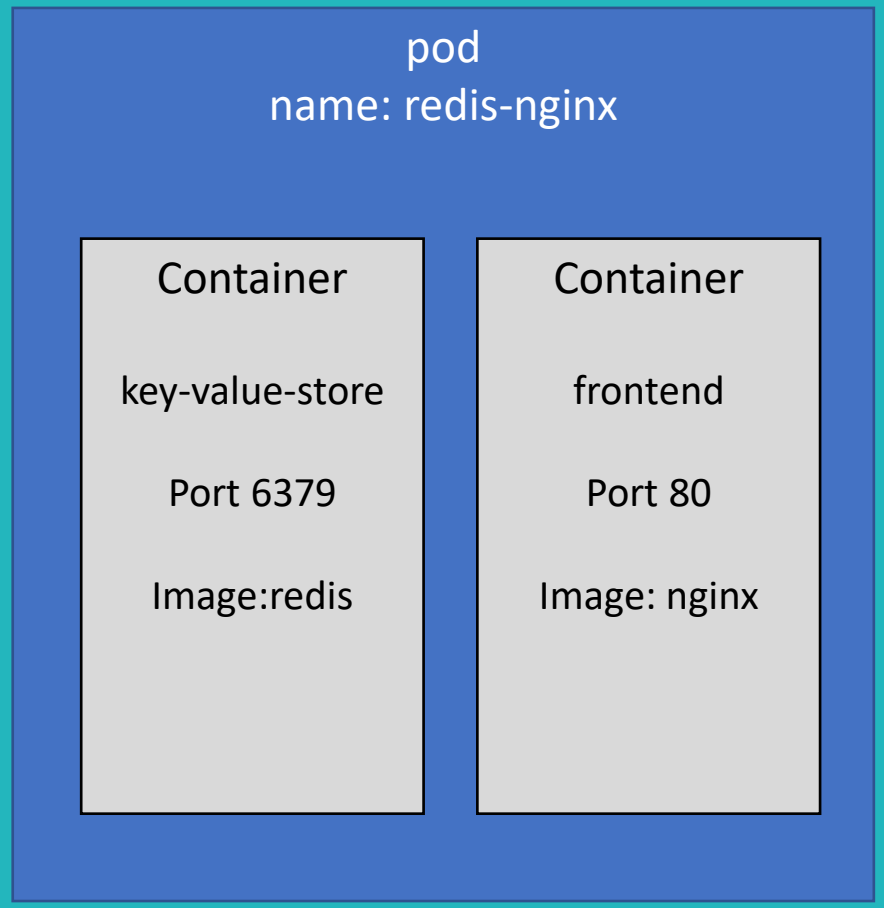
- Pod specification on a YAML file used by kubectl to ask the cluster to schedule the pod
- API Server add configuration in ETCD on a persistent way
- Scheduler finds a new pod maps to best available node
- Kubelet (on worker node) gets a notification about provisioning the pod and starts to create the associated containers
- Docker (or container runtime) creates new instances
- All pod status are saved on ETCD

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
spec:
  containers:
  - image: nginx
    name: nginx-container
    resources: {}
    ports:
    - containerPort: 80
```



moOngy.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: redis-nginx
  labels:
    app: web
spec:
  containers:
    - name: key-value-store
      image: redis
      ports:
        - containerPort: 6379
    - name: frontend
      image: nginx
      ports:
        - containerPort: 80
```



moOngy.

How to access pods

Pods

- Get access to nginx-pod pod

```
kubectl –it exec nginx-pod -- sh
```

- Get access to container frontend on redis-nginx pod

```
kubectl exec –it redis-nginx –c frontend -- bash
```

- Port forwarding to port 80 on nginx-pod pod

```
kubectl port-forward nginx-pod 8080:80
```

- Port forwarding to port 80 on redis-nginx pod

```
kubectl port-forward redis-nginx 8080:80
```

Handle resources

Pods

- A good practice when deploying pods on Kubernetes is to define the resources that will be used by it
- Kubernetes uses 2 concepts
  - Requests: Amount of resources used by scheduler to define which node could best fit. This amount is always reserved for the pod
  - Limits: Maximum amount of resources a pod can use.
- You can define request and limits for CPU and memory (in the future, for GPU too)

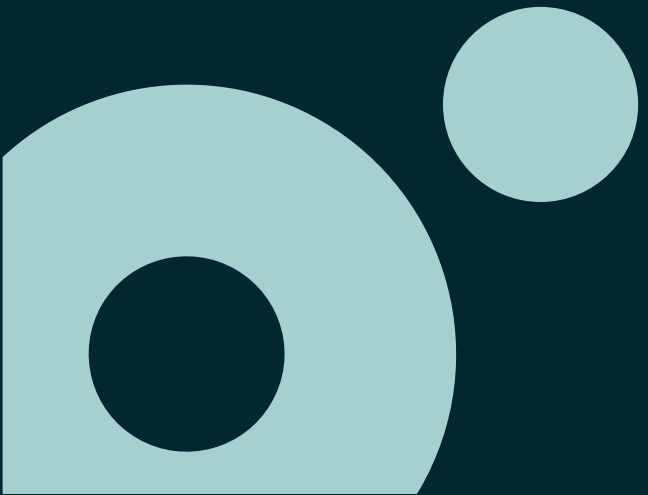
Some considerations

Pods

- A Pod don’t have any ability of self-healing
- Pods can be used directly but usually a controller is used to automatically manage your pods
- Each controller have specific way to control and manage their pods
  - ReplicaSets: Controls pods number of replicas
  - DaemonSets: Controls if one pods runs on each worker node
  - StatfulSets: Controls link between pod and persistent storage to handle pod state

moOngy. | Demo: Pods

Namespaces



What is a Namespace?

Namespaces

- Kubernetes supports multiple virtual clusters backed by the same physical cluster. These virtual clusters are called namespaces.
- Namespaces are intended for use in environments with many users spread across multiple teams, or projects.
- Namespaces provide a scope for names. Names of resources need to be unique within a namespace, but not across namespaces.
- Get a list of Namespaces

```
kubectl get namespaces
```

- Get a list of Pods in a namespace

```
kubectl get pods -n mynamespace
```

How to use?

Namespaces

- Namespaces can be used to define global network policies allowing/denying communications
- Namespaces can be used to isolate resources by:
  - Component Type – Ex: All backends in one namespace, all websites in another
  - Users – Ex: User rights/quotas can be limited by namespace
  - Environments – Ex: Dev resources can be in one namespace, QA in another
  - System Segment – Ex: Catalog microservices in one namespace, ordering in another
- To access resources across namespaces, use their FDQN:

```
curl catalog-service.mynamespace.svc.cluster.local
```

moOngy.

Mandatory usage of Namespaces?

Namespaces

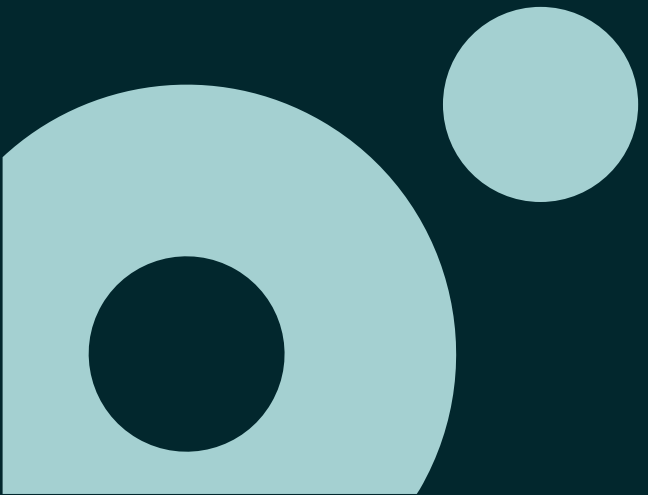
- Usually your resources are created in the context of a namespace
- “Default” namespace is used whenever you don’t explicitly set the namespace to use
- Get list of resources that aren’t namespaced scope

```
kubectl api-resources --namespaced=false
```

NAME	SHORTNAMES	APIVERSION	NAMESPACED	KIND
componentstatuses	cs	v1	false	ComponentStatus
namespaces	ns	v1	false	Namespace
nodes	no	v1	false	Node
persistentvolumes	pv	v1	false	PersistentVolume
mutatingwebhookconfigurations		admissionregistration.k8s.io/v1	false	MutatingWebhookConfiguration
validatingwebhookconfigurations		admissionregistration.k8s.io/v1	false	ValidatingWebhookConfiguration
customresourcedefinitions	crd,crds	apiextensions.k8s.io/v1	false	CustomResourceDefinition

moOngy. | Demo: Namespace

Lab



Lab 5: Introduction to Kubernetes

Github

[Lab 05 - Introduction to Kubernetes | docker-kubernetes-training (tasb.github.io)]

moOngy.
Minds on the move

Rua Sousa Martins, nº 10
1050-218 Lisboa | Portugal