Kubernetes from Basic to Advanced

kubernetes

Session #02
Introduction to Kubernetes

kubernetes

Session Contents

- Kubernetes Clusters: OnPrem, Cloud, Local
- CLI Tooling
- Declarative Configuration
- Pods
- Namespace

kubernetes

Kubernetes Clusters: OnPrem, Cloud, Local

Kubernetes Clusters

- Set of nodes running Kubernetes agents
- Control Plane (Master nodes) and Nodes (Worker Nodes)
- High availability achieved running 1+ Control Plane Nodes
- Needs to have an external load balancer to achieve HA
- May have different nodes with different hardware
- Several layers of configuration and maintenance
  - VMs / Bare Metal Machines
  - Guest OS
  - Kubernetes software

kubernetes

Kubernetes Flavours

- Kubernetes Vanilla means no add-ons
- Several Flavours with different focus on added services
  - Suse Rancher
  - Red Hat OpenShift
  - VMWare Tanzu
  - Mirantis Kubernetes Engine, …
- All flavours add services on top of Kubernetes Vanilla meaning if you only use vanilla resources you are vendor-free

kubernetes

Kubernetes Clusters: OnPrem

- All node types needs to be created and maintain manually
- For Kubernetes Vanilla, all installation process must use **kubeadm** tool
- Requirements
  - One or more machines running a deb/rpm-compatible Linux OS; for example: Ubuntu or CentOS.
  - 2 GiB or more of RAM per machine--any less leaves little room for your apps.
  - At least 2 CPUs on the machine that you use as a control-plane node.
  - Full network connectivity among all machines in the cluster. You can use either a public or a private network.
- Control Plane nodes needs to be on Linux

kubernetes

Kubernetes Clusters: OnPrem

- Benefits
  - Full control on hardware and software
  - Allow making some changes on specific components (etcd, etc.)
- Challenges
  - Node autoscaling
  - External communication
  - Control Plane HA
  - Kuberentes version update

kubernetes

Kubernetes Clusters: Cloud

- Cloud Providers have their own flavours
  - Azure Kubernetes Service (AKS)
  - Google Kubernetes Engine (GKE)
  - Amazon Elastic Kubernetes Service (EKS)
  - DigitalOcean Kubernetes
- All are compatible with Kubernetes Vanilla
- Additional services may cause vendor-lock (like Azure AD integration)
  - Identity
  - Policies
  - Secrets Management, …

kubernetes

Kubernetes Clusters: Cloud

- Kubernetes as a Service
- Cloud Providers uses the concept of Managed Services
- On all of them, Control Plane nodes are totally managed by cloud provider
- Makes it easier to handle biggest challenges identified on OnPrem version
- Costs are mainly
  - Computing from worker nodes
  - Control Plane HA
  - In some cases, additional services may have additional costs

kubernetes

Cloud or OnPrem

- It depends ☺
  - Data sovereignty
  - Security concerns (can be argued)
- Why not hybrid?
- Most clouds have services to add OnPrem Kubernetes Clusters to be managed as a Cloud Resource (Azure Arc, Amazon EKS Anywhere, Google Anthos)
- Allow to easily use additional services like Policies

kubernetes

CLI Tooling

CLI Tooling

- Preferable way to interact with Kubernetes Clusters is using CLI (Command-Line Interface) tools
- Motivation
  - Allow to automatize interactions
  - Needs much less resources
  - Small attach surface

kubernetes

API Server

- API Server 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}`
- Some challenges
  - Manage REST requests
  - Authn/authz
- Kubernetes have a CLI tool to make this interaction easily: **kubectl**

kubernetes

Kubectl

- API Server 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}`
- Some challenges
  - Manage REST requests
  - Authn/authz
- Kubernetes have a CLI tool to make this interaction easily: **kubectl**
- First challenge, how to pronounce? ‘kubectl’: The definitive pronunciation guide

kubernetes

Kubectl: How to use?

```
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

kubernetes

Kubectl: Examples

- List all nodes

```
kubectl get nodes
```

- Get more details on node node01

```
kubectl describe node node01
```

- List all nodes with more details

```
kubectl get node -o wide
```

- List all nodes and labels

```
kubectl get node -show-labels
```

kubernetes

Kubectl: How to reference resources?

- 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)

kubernetes

Kubectl: How to identify?

- **`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
- Uses the concept of contexts. A context makes a match between a cluster (API Server URL) and a user

Kubectl: Examples

- 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
```

kubernetes

Demo | kubectl

Krew: Plugin Manager for Kubectl

- Allow to search, install and update additional tools to use with kubectl
- 210 plugins available: https://krew.sigs.k8s.io/plugins/
- Maintenance done by a Kubernetes SIG (Special Interest Group)
- List installed plugins

```
kubectl krew list
```

- Install a plugin

```
kubectl krew install <package>
```

kubernetes

Krew: Plugin Manager for Kubectl

- cert-manager: Manage cert-manager resources inside your cluster
- ctx: Switch between contexts in your kubeconfig
- ingress-nginx: Interact with ingress-nginx
- kubesec-scan: Scan Kubernetes resources with kubesec.io.
- ns: Switch between Kubernetes namespaces
- resource-capacity: Provides an overview of resource requests, limits, and utilization

kubernetes

Declarative Configuration

Declarative Configuration

- 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.

kubernetes

YAML: Yet Another Markup Language

- 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
- When you create an YAML file to define Kubernetes resources, you're creating a Kubernetes Manifest File

kubernetes

YAML: Yet Another Markup Language

- 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 (---).

kubernetes

Manifest File

- 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

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
spec:
  containers:
  - image: nginx:1.18-alpine
    name: nginx-container
    ports:
    - containerPort: 80
```

kubernetes

Pods

What is a Pod?

- 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

kubernetes

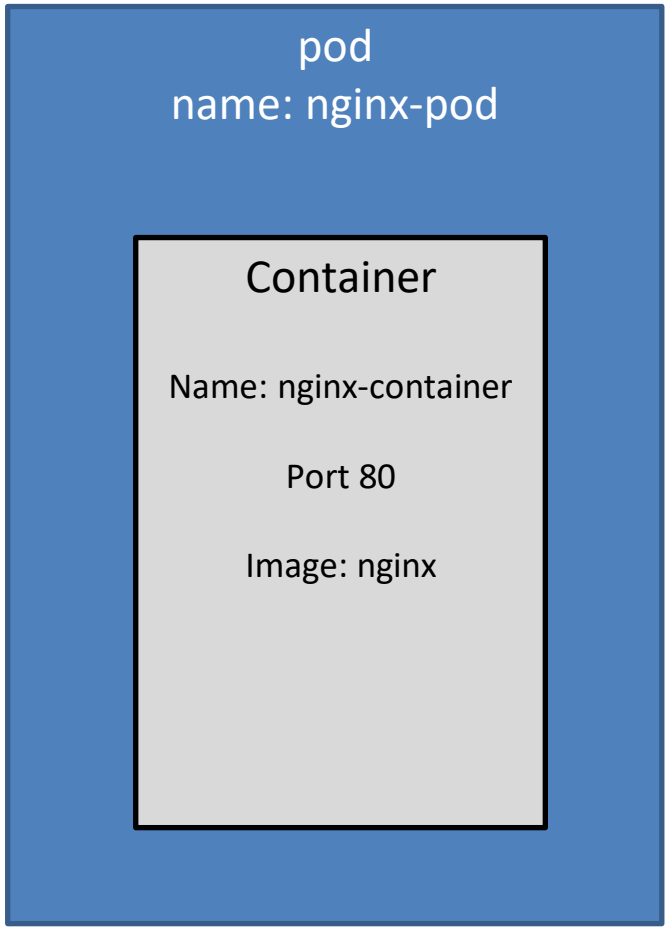
Pod Lifecycle

- 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

kubernetes

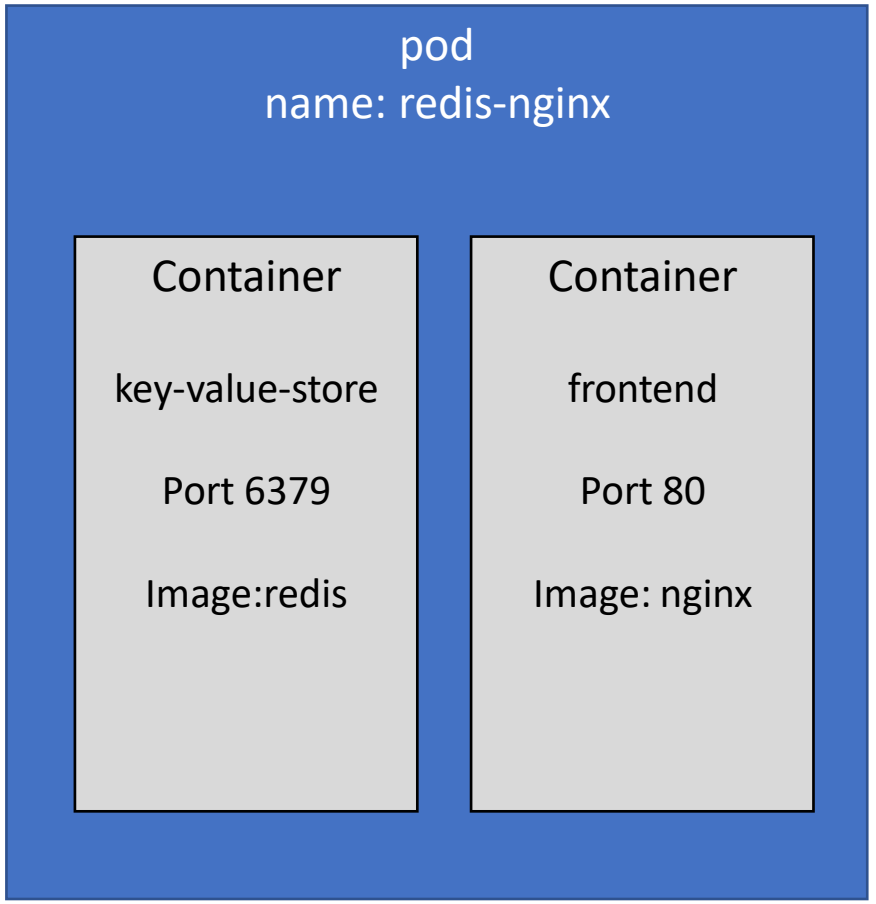
Pod with one container

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



Pod with two containers

```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
```



Kubectl: Work with pods

- Get all pods

```
kubectl get pods
```

- Get additional details from pods

```
kubectl get pods -o wide
```

- Get pod named **nginx-pod**

```
kubectl get pod nginx-pod
```

- Inspect pod named **nginx-pod**

```
kubectl describe pod nginx-pod
```

kubernetes

Kubectl: Interact with 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
```

kubernetes

Pods: Handle Resources

- A good practice when deploying pods on Kubernetes is to define the resources that will be used by it
- Kubernetes uses 2 concepts: Requests and Limits
- 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 CPUs, memory and GPUs

kubernetes

Pods: Self Healing

- 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

kubernetes

Demo | Pods

Namespaces

Namespaces

- Kubernetes supports multiple virtual clusters backed by the same physical cluster
- These virtual clusters are called namespaces and creates logical partitions inside your cluster
- Namespaces provide a scope for names
- Names of resources need to be unique within a namespace, but not across namespaces.
- Namespaces can be used to
  - Allowing/denying network communications
  - Define RBAC permissions
  - Define resource quotas

kubernetes

Namespaces are mandatory?

- Usually, your resources are created in the context of a namespace
- "**`default`**" namespace is created on a empty cluster
- Your **`kubectl`** commands uses the namespace you set on your **`kubeconfig`** to be the default
- You can use **`kubectl ns`** plugin to easily change it
- When you want to not change the default, you need to use the flag **`-n`** in each command

kubernetes

Namespaces are mandatory?

- When a resource is not namespaced, means that is a cluster resource

- 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

kubernetes

Kubectl: Work with namespaces

- Get all namespaces

```
kubectl get namespaces
```

- Get all namespaces using shortname

```
kubectl get ns
```

- Get list of pods from namespace **my-ns**

```
kubectl get pod -n my-ns
```

- Delete **my-ns** namespace

```
kubectl delete ns my-ns
```

kubernetes

Namespaces: Why to use?

- Namespaces can be used to define global policies
  - Allowing/denying network communications
  - Define RBAC permissions
  - Define resource quotas
- Namespaces are usually created 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

kubernetes

Namespaces: How to define?

- Namespaces are usually created 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

kubernetes

Demo | Namespaces

Questions?

kubernetes

Lab #01: Introduction to K8S

kubernetes