MLOps

Tuur Vanhoutte

September 27, 2021

Contents

1 Introduction

- Deployment of deep learning models in an API
- Deploying services to Kubernetes with Docker
- Version Controlling with Git (to blame those who ruined the code)
- Automation pipelines to prepare data, train models and deploy applications
  - Automated delivery (ready to deploy)
  - Automated deployment (CI / CD)

1.1 Why do I need deployment automation

- Save time
- Increased accuracy
- Better documentation
- Autoscaling

1.2 What will we do

- Object classification with a CNN
- Deployed with FastAPI
- Scalable with Kubernetes
- Automated with CI/CD pipelines
  - Yes, also automated AI Training!

1.3 The AI pipeline

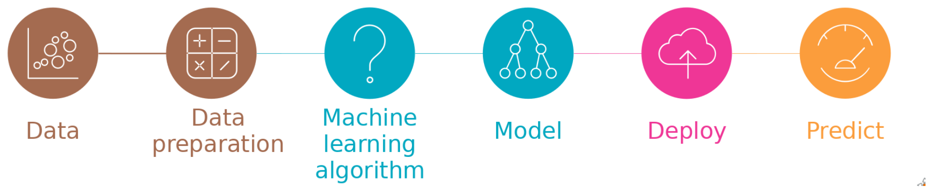


Figure 1: The complete AI pipeline

1.4 Teleport

How to connect to your dedicated virtual machine? **Teleport** allows engineers and security professionals to unify access for SSH servers, Kubernetes clusters, web applications, and databases across all environments.

1.4.1 Teleport Server Access

- One centralized login and user-management system
- Randomly generated passwords for your user account
- We manage all virtual machines your use can access

To **install and use Teleport**, see Leho.

2 FastAPI

TODO

3 Docker

TODO

4 Kubernetes

4.1 What is Kubernetes?

Definition 4.1 *Kubernetes is a* ***portable****,* ***extensible****,* ***open-source*** *platform for* ***managing containerized workloads and services****, that facilitates both* ***declarative configuration*** *and* ***automation****. It has a large, rapidly growing ecosystem. Kubernetes services, support, and tools are widely available.*

- Service discovery and load balancing
- Storage orchestration
- Automated rollouts and rollbacks
- Self-healing
- Auto-scaling

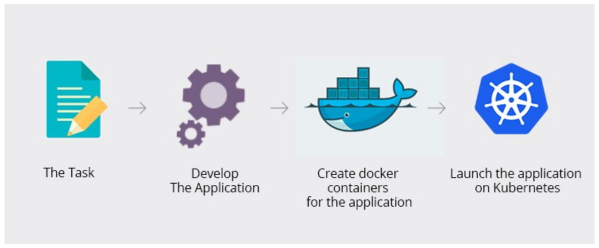

The Task
Develop The Application
Create docker containers for the application
Launch the application on Kubernetes

Figure 2: Kubernetes workflow

4.2 Architecture

A Kubernetes cluster consists of a set of worker machines, called nodes, that run containerized applications. Worker nodes host the Pods that are the components of the application workload. The control plane manages the worker nodes and the Pods in the cluster. In production environments, the control plane usually runs across multiple computers and a cluster usually runs multiple nodes, providing fault-tolerance and high availability.

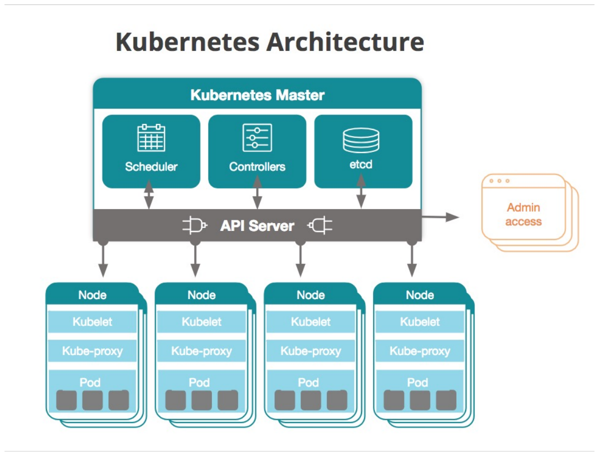


Figure 3: Architecture

4.2.1 Control Plane / Kubernetes Master

- Scheduler
  - Makes sure that the number of desired pods is always running
  - Keeps track of capacity and resources of nodes
  - Assigns work to nodes based on availability
- Controller
  - responsible for registering nodes and monitoring health
- etcd
  - A persistent and distributed key-value data store
  - Stores the state and configuration data for the entire cluster
- API-server
  - main access point to the control plane and master node

4.2.2 Node

A node is a virtual machine. It has at least two tasks:

- Kubelet
  - Manages the state of a node
    - Starting, stopping and maintaining application containers with instructions from the control plane
  - Collects performance and health information of node, pods and containers

        * Shares those with the Control Plane

- Kube-proxy
  - Network proxy
  - Manages the Virtual IP addresses of pods and services
  - Also works as a load balancer for services running on a node
- Pod

4.3 Objects

Kubernetes objects are persistent entities in the Kubernetes system. Kubernetes uses these entities to represent the state of your cluster. Specifically, they can describe:

- What containerized applications are running (and on which nodes)
- The resources available to those applications
- The policies around how those applications behave, such as restart policies, upgrades, and fault-tolerance

A Kubernetes object is a "record of intent"–once you create the object, the Kubernetes system will constantly work to ensure that object exists. By creating an object, you're effectively telling the Kubernetes system what you want your cluster's workload to look like; this is your cluster's desired state.

4.3.1 Building blocks

- **Pods**
- **Services** and EndPoints
- **Deployments**
- ReplicaSets
- DaemonSets
- StatefulSets
- **Ingress**
- PersistentVolumes and PersistentVolumeClaims
- ConfigMaps and Secrets

4.3.2 Pods

Definition 4.2 *In Kubernetes, the* ***Pod*** *serves as a kind of basic, functional unit. A pod is a set of containers, along with their shared volumes.*

As Kubernetes' scheduler creates and deletes application pods unexpectedly, you should not rely on a particular pod. However, you do need to be able to access your application in a predictable manner. And to do that, Kubernetes provides the simplest form of load balancing traffic, namely a Service.

4.3.3 Services

Definition 4.3 *A Kubernetes Service is an abstraction which defines a logical set of Pods and a policy by which to access them - sometimes called a micro-service. The set of Pods targeted by a Service is (usually) determined by a Label Selector.*

Services have IP addresses (used internally by Kubernetes) which are relatively stable. When a program element needs to make use of the functions abstracted by the service, it makes a request to the service, rather than an individual pod. The service then acts as a dispatcher, assigning a pod to handle the request. Thus, a client never connects to a container, but to a Pod, through a Service.

4.3.4 Deployments and ReplicaSets

Definition 4.4 *A deployment is a YAML object that defines the pods and the number of container instances, called replicas, for each pod. You define the number of replicas you want to have running in the cluster via a ReplicaSet.*

Definition 4.5 *A ReplicaSet is part of the deployment object. So, for example, if a node running a pod dies, the replica set will ensure that another pod is scheduled on another available node.*

4.3.5 Daemonsets

Definition 4.6 *A DaemonSet ensures that all (or some) Nodes run a copy of a Pod. As nodes are added to the cluster, Pods are added to them. As nodes are removed from the cluster, those Pods are garbage collected. Deleting a DaemonSet will clean up the Pods it created.*

Some typical uses of a DaemonSet are:

- running a cluster storage daemon, such as glusterd, ceph, on each node.
- running a logs collection daemon on every node, such as fluentd or filebeat.

4.3.6 StatefulSets

Definition 4.7 *Like a Deployment, a StatefulSet manages Pods that are based on an identical container spec. Unlike a Deployment, a StatefulSet maintains a sticky identity for each of their Pods. These pods are created from the same spec, but are not interchangeable: each has a persistent identifier that it maintains across any rescheduling.*

If you want to use storage volumes to provide persistence for your workload, you can use a StatefulSet as part of the solution. Although individual Pods in a StatefulSet are susceptible to failure, the persistent Pod identifiers make it easier to match existing volumes to the new Pods that replace any that have failed.

4.3.7 PersistentVolumes and PersistentVolumeClaims

PersistentVolumes

- Define a storage volume in the cluster
- Independent lifecycle
- Otherwise: ephemeral data inside the pods

4.3.8 ConfigMaps and Secrets

- Decouple configurations from hard-coded environment variables
- Separate Kubernetes object to share with Pods
- Difference: Secrets hold confidential information
- Example:
  - Different configurations for Development, Staging and Production environments, quickly interchangeable
  - Storing passwords and using it in your applications

4.4 Load balancer

L4 Load Balancing (TCP)

- Only available on Kubernetes Cloud Providers (GCP, AWS, Azure . . .)

L7 Load Balancing

- Can redirect traffic to specific workloads based on request
- Ingress

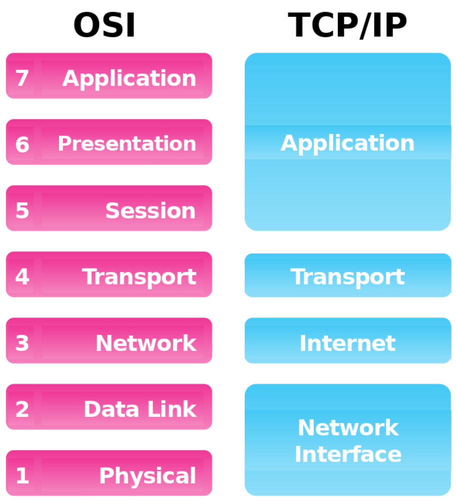


Figure 4: Layers in OSI & TCP/IP

- Kube-proxy
  - Load distribution
  - Default: Randomly route to IP addresses based on iptables
  - Previously: Round-robin
- Ingress
  - Routes traffic based on request rules configured
  - Deploy a Controller (like nginx)

- Deploy Resources
- Tip: Use Rancher to deploy Ingress Controllers and Resources (it has a UI for that)

4.5 Autoscaler

Kubernetes uses the horizontal pod autoscaler (HPA) to monitor the resource demand and automatically scale the number of replicas. When changes in replica count are required, the number of replicas is increased or decreased accordingly.

You define the minimum and maximum number of replicas that can run. You also define the metric to monitor and base any scaling decisions on, such as CPU usage.

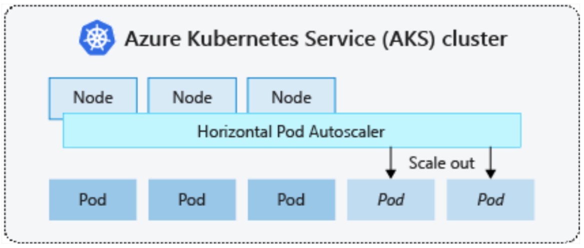

Azure Kubernetes Service (AKS) cluster
Node
Node
Node
Horizontal Pod Autoscaler
Scale out
Pod
Pod
Pod
Pod
Pod

Figure 5: Autoscaler

4.6 YAML file

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2 # tells deployment to run 2 pods
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

- apiVersion
  - Which version of the Kubernetes API you're using to create this object.
- kind
  - What kind of object you want to create

- metadata
  - Data that helps uniquely identify the object, including the name string, UID and optional namespace
- spec
  - What state you desire for the object

4.7 Helm

Definition 4.8 *Helm is an application package management registry for Kubernetes, maintained by the CNCF. Helm "charts" are pre-configured software application resources you can download and deploy and in your Kubernetes environment.*

According to a 2018 CNCF survey, 68% of respondents said Helm was the preferred package management tool for Kubernetes applications. Helm charts can help DevOps teams come up to speed more quickly with managing applications in Kubernetes; it allows them leverage existing charts that they can share, version, and deploy into their dev and production environments.

- Package manager
- We can create custom Helm charts

```
    spec:
      containers:
      - args:
        - uwsgi
        - --ini
        - app.ini
        env:
        {{- range $key, $value := .Values.env.webapp }}
        - name: {{ $key }}
          value: {{ $value | quote }}
        {{- end }}
        image:{{ .Values.repository }}
        /basicwebapp:{{ .Values.app.version.webapp }}
        name: {{ $name }}
        resources: {}
      restartPolicy: Always
```

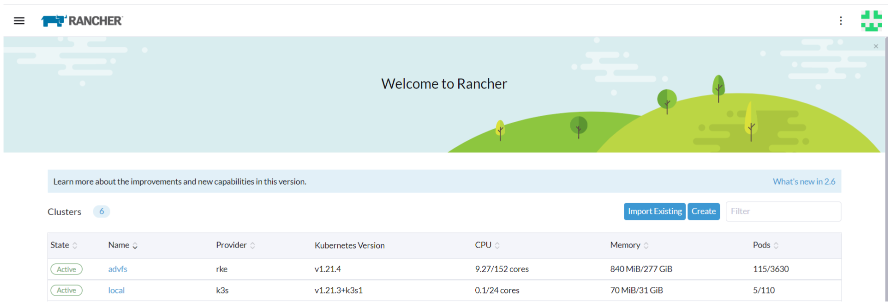
- Organised in template.yaml files
- Placeholders inside yaml files
- Values.yaml to fill placeholders
- Installing and upgrading versions of a release through CLI
- Loops, conditionals, variables, filters, values and more . . .

5 Rancher

Definition 5.1 *Rancher is a complete software stack for teams adopting containers.*

It addresses the operational and security challenges of managing multiple Kubernetes clusters, while providing DevOps teams with integrated tools for running containerized workloads.

- Simple, intuitive UI to get started with Kubernetes
- Multi-cloud and hybrid-cloud Kubernetes solutions
- Fast way to set up on-premises Kubernetes clusters
- Includes CI/CD pipelines for DevOps operations

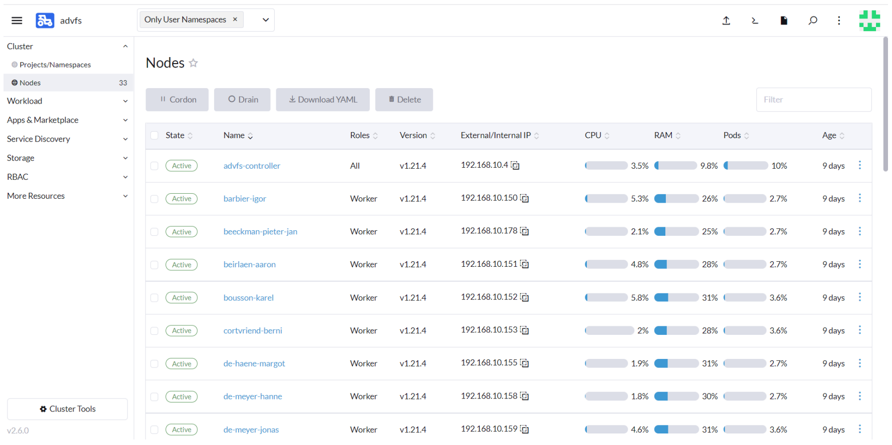


Learn more about the improvements and new capabilities in this version.

Clusters 6

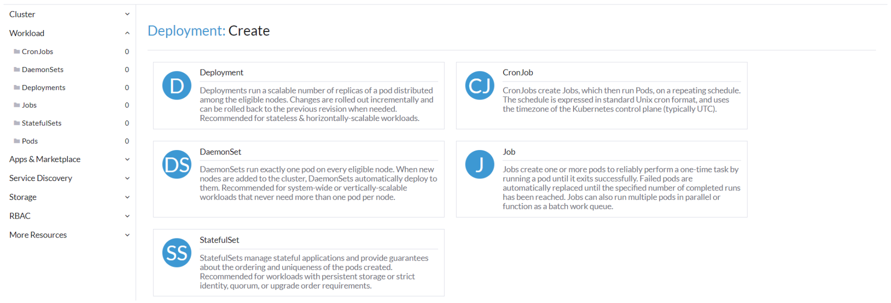
State	Name	Provider	Kubernetes Version	CPU	Memory	Pods
Active	advfs	rke	v1.21.4	9.27/152 cores	840 MiB/277 GiB	115/3630
Active	local	k3s	v1.21.3+k3s1	0.1/24 cores	70 MiB/31 GiB	5/110

Figure 6: Rancher welcome page



State	Name	Roles	Version	External/Internal IP	CPU	RAM	Pods	Age
Active	advfs-controller	All	v1.21.4	192.168.10.4	3.5%	9.8%	10%	9 days
Active	barbier-igor	Worker	v1.21.4	192.168.10.150	5.3%	26%	2.7%	9 days
Active	beeckman-pieter-jan	Worker	v1.21.4	192.168.10.178	2.1%	25%	2.7%	9 days
Active	beirlaen-aaron	Worker	v1.21.4	192.168.10.151	4.8%	28%	2.7%	9 days
Active	bousson-karel	Worker	v1.21.4	192.168.10.152	5.8%	31%	3.6%	9 days
Active	cortvriend-berni	Worker	v1.21.4	192.168.10.153	2%	28%	3.6%	9 days
Active	de-haene-margot	Worker	v1.21.4	192.168.10.155	1.9%	31%	2.7%	9 days
Active	de-meyer-hanne	Worker	v1.21.4	192.168.10.158	1.8%	30%	2.7%	9 days
Active	de-meyer-jonas	Worker	v1.21.4	192.168.10.159	4.6%	31%	3.6%	9 days

Figure 7: List of nodes



Deployment: Create

Deployment
Deployments run a scalable number of replicas of a pod distributed among the eligible nodes. Changes are rolled out incrementally and can be rolled back to the previous revision when needed. Recommended for stateless & horizontally-scalable workloads.

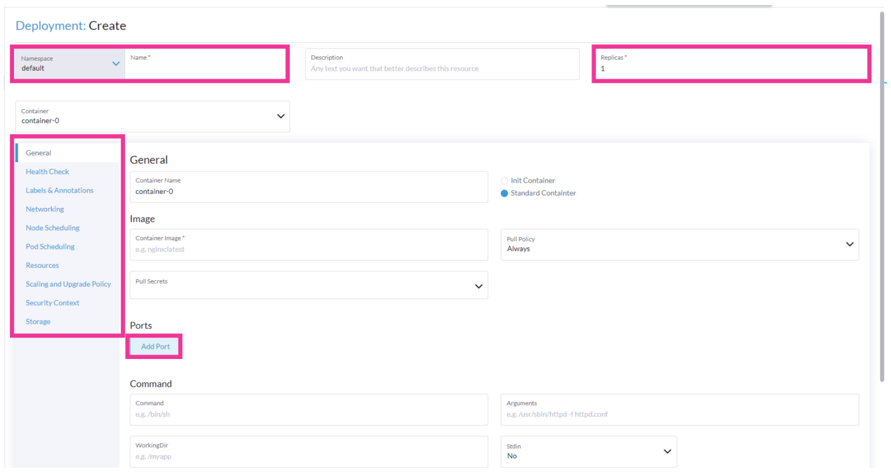
CronJob
CronJobs create Jobs, which then run Pods, on a repeating schedule. The schedule is expressed in standard Unix cron format, and uses the timezone of the Kubernetes control plane (typically UTC).

DaemonSet
DaemonSets run exactly one pod on every eligible node. When new nodes are added to the cluster, DaemonSets automatically deploy to them. Recommended for system-wide or vertically-scalable workloads that never need more than one pod per node.

Job
Jobs create one or more pods to reliably perform a one-time task by running a pod until it exits successfully. Failed pods are automatically replaced until the specified number of completed runs has been reached. Jobs can also run multiple pods in parallel or function as a batch work queue.

StatefulSet
StatefulSets manage stateful applications and provide guarantees about the ordering and uniqueness of the pods created. Recommended for workloads with persistent storage or strict identity, quorum, or upgrade order requirements.

Figure 8: Deployment



Deployment: Create

Namespace: default
Name*
Description: Any text you want that better describes this resource
Replicas*: 1

Container: container-0

General

Container Name: container-0
Init Container [✓ Standard Containter]

Image

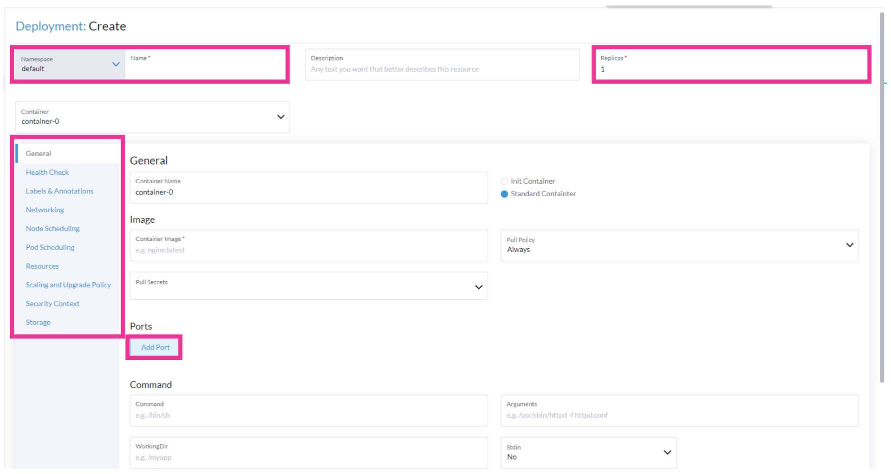
Container Image*: e.g. nginx:latest
Pull Policy: Always
Pull Secrets

Ports

Command

Command: e.g. /bin/sh
Arguments: e.g. /usr/sbin/httpd -f httpd.conf
WorkingDir: e.g. /myapp
Stdin: No

Figure 9: Creating a Deployment



Deployment: Create

Namespace: default
Name*
Description: Any text you want that better describes this resource
Replicas*: 1

Container: container-0

General

Container Name: container-0
Init Container [✓ Standard Containter]

Image

Container Image*: e.g. nginx:latest
Pull Policy: Always
Pull Secrets

Ports

Command

Command: e.g. /bin/sh
Arguments: e.g. /usr/sbin/httpd -f httpd.conf
WorkingDir: e.g. /myapp
Stdin: No

Figure 10: Creating a Deployment

5.1 What we learned

- Why Kubernetes is useful for microservices
- The Kubernetes Building Blocks
- How autoscaling in Kubernetes works
- Different kinds of Load Balancing options
- How to write a YAML file for a Kubernetes object
- What the advantages of Helm charts are
- What Rancher does for us

Setting up Rancher and Kubernetes: see labs