Kubernetes Training (3 days)

Agenda

1. Grundlagen
    - Welches System ? (minikube, micro8ks etc.)
    - Aufbau
2. Installation
    - Überblick
    - Linux Client aufsetzen
3. microk8s
    - Installation Ubuntu - snap
    - Patch to next major release - cluster
    - Remote-Verbindung zu Kubernetes (microk8s) einrichten
    - Create a cluster with microk8s
    - Arbeiten mit der Registry
4. kubectl
    - alle Ressourcen (Möglichkeiten) der Api anzeigen
    - pod starten mit beispiel
    - alle pods anzeigen
    - auf welcher Node läuft ein pod
    - pods löschen
    - Mit pod verbinden - terminal
5. Kubernetes
    - Deployments
6. Gitlab CI/CD
    - Predefined Variables
    - Kubernetes pipeline walkthrough
7. Ingress
    - Nginx Ingress Controller
8. Monitoring
    - Prometheus Operator
9. Logging
    - Logging on Node - Level (fluentd)
10. Netzwerk
    - Ist ipv6 mit ipsec möglich ? (Stand: Experimentell)
    - Metallb als LoadBalancer vor Ingress verwenden
    - Calico (Default Netzwerk-Dienst microk8s) - Netzwerk Policies
11. Shared Volumes
    - Shared Volumes with Rook & Ceph
    - Shared Volumes with nfs
12. Tipps & Tricks
    - kubectl-Autovervollständigung
    - Install plugin kubectl-convert
13. Examples
    - Kuard pod
    - Pod nginx port exposed
    - Deployment nginx
    - Ingress Nginx
    - Combind example in manifest (ubuntu-nginx)
    - Combined example in manifest (ubuntu-nginx-service)
14. Use Cases
    - Use Cases Foundation
    - Use Cases Kubernetes
    - Use Cases

Grundlagen

Welches System ? (minikube, micro8ks etc.)

General

```
kubernetes itself has not convenient way of doing specific stuff like
creating the kubernetes cluster.

So there are other tools/distri around helping you with that.
```

Kubeadm

General

- The official CNCF (https://www.cncf.io/) tool for provisioning Kubernetes clusters (variety of shapes and forms (e.g. single-node, multi-node, HA, self-hosted))
- Most manual way to create and manage a cluster

Disadvantages

- Plugins sind oftmals etwas schwierige zu aktivieren

microk8s

General

- Created by Canonical (Ubuntu)
- Runs on Linux
- Runs only as snap
- In the meantime it is also available for Windows/Mac
- HA-Cluster

Production-Ready ?

- Short answer: YES

```
Quote canonical (2020):

MicroK8s is a powerful, lightweight, reliable production-ready Kubernetes distribution. It is an enterprise-grade Kubernetes
distribution that has a small disk and memory footprint while offering carefully selected add-ons out-the-box, such as Istio,
Knative, Grafana, Cilium and more. Whether you are running a production environment or interested in exploring K8s, MicroK8s
serves your needs.

Ref: https://ubuntu.com/blog/introduction-to-microk8s-part-1-2
```

Advantages

- Easy to setup HA-Cluster (multi-node control plane)
- Easy to manage

minikube

Disadvantages

- Not usable / intended for production

Advantages

- Easy to set up on local systems for testing/development (Laptop, PC)
- Multi-Node cluster is possible
- Runs und Linux/Windows/Mac
- Supports plugin (Different name ?)

k3s

kind (Kubernetes-In-Docker)

General

- Runs in docker container

For Production ?

```
Having a footprint, where kubernetes runs within docker
and the applikations run within docker as docker containers
it is not suitable for production.
```

Misc

```

Minikube
Minikube can run on Windows and MacOS, because it relies on virtualization (e.g. Virtualbox) to deploy a kubernetes cluster in a
Linux VM. You can also run minikube directly on linux with or without virtualization. It also has some developer-friendly
features, like add-ons.

Minikube is currently limited to a single-node Kubernetes cluster (for details, see this issue). Although, it is on their roadmap.
```

K3s
K3s runs on any Linux distribution without any additional external dependencies or tools. It is marketed by Rancher as a lightweight Kubernetes offering suitable for edge environments, IoT devices, CI pipelines, and even ARM devices, like Raspberry Pi's. K3s achieves its lightweight goal by stripping a bunch of features out of the Kubernetes binaries (e.g. legacy, alpha, and cloud-provider-specific features), replacing docker with containerd, and using sqlite3 as the default DB (instead of etcd). As a result, this lightweight Kubernetes only consumes 512 MB of RAM and 200 MB of disk space. K3s has some nice features, like Helm Chart support out-of-the-box.

Unlike the previous two offerings, K3s can do multiple node Kubernetes cluster. However, due to technical limitations of SQLite, K3s currently does not support High Availability (HA), as in running multiple master nodes. The K3s team plans to address this in the future.

Now, on to some honorary mentions...

Kind
Kind (Kubernetes-in-Docker), as the name implies, runs Kubernetes clusters in Docker containers. This is the official tool used by Kubernetes maintainers for Kubernetes v1.11+ conformance testing. It supports multi-node clusters as well as HA clusters. Because it runs K8s in Docker, kind can run on Windows, Mac, and Linux.

Kind is optimized first and foremost for CI pipelines, so it may not have some of the developer-friendly features of other offerings.

Desktop Docker
Docker for Mac/Windows now ships with a bundled Kubernetes offering.

However:

Kubernetes versions are tightly coupled with the Docker version (i.e. Docker stable channel ships with K8s v1.10. If you want K8s v1.13, you need to switch to Docker edge channel).

Not as easy to destroy and start a new K8s cluster. AFAIK, you would have to disable Kubernetes and re-enable it through the Docker desktop app preferences.

Kubeadm

... And there are plenty more that I do not remember off the top of my head.

So, it all depends on what you want to get out these tools and out of Kubernetes.

Are you a developer who wants a simple K8s cluster and don't need or care about multi-node/HA features?

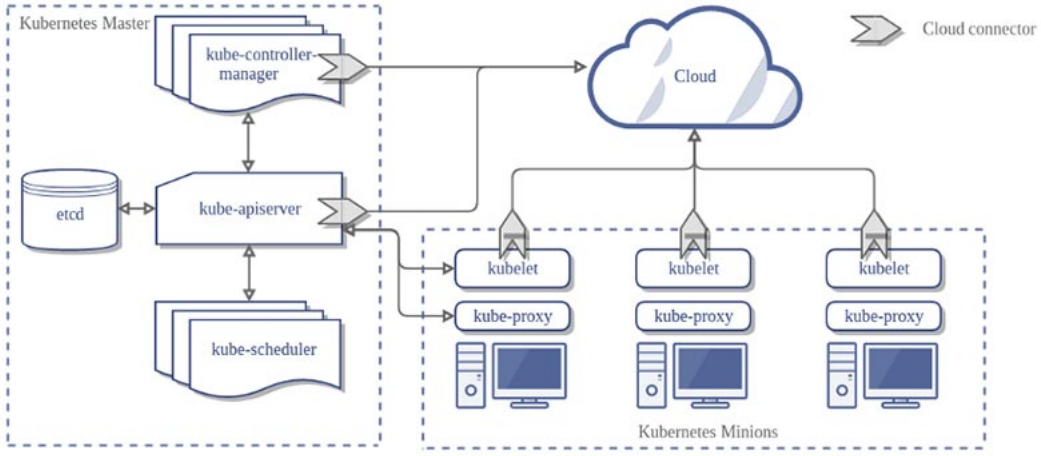
Are you a developer working on a distributed application that runs on Kubernetes and need to test various failure scenarios (e.g. node failure)?

Are you trying to learn about Kubernetes from a cluster administrator's perspective?

I hope you find this information helpful.

Aufbau

Schaubild



Komponenten / Grundbegriffe

Master (Control Plane)

Aufgaben

- Der Master koordiniert den Cluster
- Der Master koordiniert alle Aktivitäten in Ihrem Cluster
  - Planen von Anwendungen
  - Verwalten des gewünschten Status der Anwendungen
  - Skalieren von Anwendungen
  - Rollout neuer Updates.

Komponenten des Masters

ETCD

- Verwalten der Konfiguration des Clusters (key/value - pairs)

KUBE-CONTROLLER-MANAGER

- Zuständig für die Überwachung der Stati im Cluster mit Hilfe von endlos loops.
- kommuniziert mit dem Cluster über die kubernetes-api

KUBE-API-SERVER

- provides api-frontend for administration (no gui)
- Exposes an HTTP API (users, parts of the cluster and external components communicate with it)
- REST API

KUBE-SCHEDULER

- assigns Pods to Nodes.
- scheduler determines which Nodes are valid placements for each Pod in the scheduling queue (according to constraints and available resources)
- The scheduler then ranks each valid Node and binds the Pod to a suitable Node.
- Reference implementation (other schedulers can be used)

Nodes

- Nodes (Knoten) sind die Arbeiter (Maschinen), die Anwendungen ausführen
- Ref: https://kubernetes.io/de/docs/concepts/architecture/nodes/

Pod/Pods

- Pods sind die kleinsten einsetzbaren Einheiten, die in Kubernetes erstellt und verwaltet werden können.
- Ein Pod (übersetzt Gruppe) ist eine Gruppe von einem oder mehreren Containern
  - gemeinsam genutzter Speicher- und Netzwerkressourcen
  - Befinden sich immer auf dem gleich virtuellen Server

Control Plane Node (former: master) - components

Node (Minion) - components

General

- On the nodes we will rollout the applications

kubelet

```
Node Agent that runs on every node (worker)
Er stellt sicher, dass Container in einem Pod ausgeführt werden.
```

Kube-proxy

- Läuft auf jedem Node
- = Netzwerk-Proxy für die Kubernetes-Netzwerk-Services.
- Kube-proxy verwaltet die Netzwerkkommunikation innerhalb oder außerhalb Ihres Clusters.

Referenzen

- https://www.redhat.com/de/topics/containers/kubernetes-architecture

Installation

Überblick

Linux Client aufsetzen

```
## If you want to use ubuntu as your local kubectl client
## set it up like so (e.g. in virtualbox)



## hostnamectl set-hostname client.training.local
## will install latest version as snap
sudo snap install kubectl --classic
## show other versions
snap info kubectl

kubectl version --client

## vi ~/.bashrc
echo "alias k=kubectl" >> ~./bashrc
source ~/.bashrc
```

microk8s

Installation Ubuntu - snap

```
sudo snap install microk8s --classic
## Important enable dns // otherwice not dns lookup is possible
microk8s enable dns
microk8s status

## Execute kubectl commands like so
microk8s kubectl
microk8s kubectl cluster-info

## Make it easier with an alias
echo "alias kubectl='microk8s kubectl'" >> ~/.bashrc
source ~/.bashrc
kubectl
```

Patch to next major release - cluster

Remote-Verbindung zu Kubernetes (microk8s) einrichten

```
## On master-server get config
microk8s config > remote_config

## Download (scp config file) and store in .kube - folder
cd ~
mkdir .kube
cd .kube
scp master_server:/path/to/remote_config config

## now you can execute all kubectl commands, but they are executed against remote server
microk8s kubectl get pods
## or if using kubectl or alias
kubectl get pods

## if you want to use a different kube config file, you can do like so
kubectl --kubeconfig /home/myuser/.kube/myconfig
```

Create a cluster with microk8s

Walkthrough

```
## auf master (jeweils für jedes node neu ausführen)
microk8s add-node

## dann auf jeweiligem node vorigen Befehl der ausgegeben wurde ausführen
## Kann mehr als 60 sekunden dauern ! Geduld...Geduld..Geduld
```

Ref:

- https://microk8s.io/docs/high-availability

Arbeiten mit der Registry

```
microk8s enable registry

mkdir ~/docker-images/ubuntu/
cd ~/docker-images/ubuntu
vi Dockerfile

FROM ubuntu:latest
RUN apt-get update
RUN apt-get install -y inetutils-ping
RUN apt-get install -y telnet
RUN echo 'while true; do sleep 15 ; done' > /bootstrap.sh
RUN chmod +x /bootstrap.sh

CMD /bootstrap.sh

docker build -t myubuntu .
docker images
docker tag myubuntu localhost:32000/myubuntu

## testrun,ob container im Hintergrund laufen kann
docker run -d localhost:32000/myubuntu
docker ps

## vernichten
docker rm -f <id-des-containers>

## manifests erstellen
vi ubuntu.yml
apiVersion: v1
kind: Pod
metadata:
  name: static-ubuntu

spec:
  containers:
  - name: ubuntu-os
    image: localhost:32000/myubuntu

## apply'en
kubectl apply -f ubuntu.yml
```

kubectl

alle Ressourcen (Möglichkeiten) der Api anzeigen

```
kubectl api-resources
```

pod starten mit beispiel

Example

```
## Synopsis (most simplistic example
## kubectl run NAME --image=IMAGE_EG_FROM_DOCKER
## example
kubectl run nginx --image=nginx

kubectl get pods
## on which node does it run ?
kubectl get pods -o wide
```

Ref:

- https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands#run

alle pods anzeigen

Basics

```
kubectl get pods
## aber auch get pod kann verwendet werden, gleiches Ergebnis
kubectl get pod
```

Erweitert

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

auf welcher Node läuft ein pod

```
## Hier hilft die Ausgabe -o wide
kubectl get pods -o wide
```

pods löschen

```
## Variante 1
kubectl delete pods/mypod

## Variante 2
kubectl apply -f pod-manifest.yml
kubectl get pods
## now delete
kubectl delete -f pod-manifest.yml
```

Mit pod verbinden - terminal

Connect to specific pod

```
kubectl exec -it ubuntu-deployment-557dfb95d4-jc57t -- bash
```

Connect to 1 random pod in deployment

```
### This connects to one random pod within deployment

## ubuntu-deployment was the name of the deployment
kubectl exec -it deploy/ubuntu-deployment -- bash
```

Kubernetes

Deployments

Example

```
## Deploy a sample from k8s.io
kubectl apply -f https://k8s.io/examples/controllers/nginx-deployment.yaml
```

Refs:

- https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

Gitlab CI/CD

Predefined Variables

- https://docs.gitlab.com/ee/ci/variables/predefined_variables.html

Kubernetes pipeline walkthrough

Walkthrough

```
1. Create new repo on gitlab
2. CI/CD workflow aktivieren, in dem wir auf das Menü CI/CD klicken
-> Get started with GitLab CI/CD -> Use Template

3. file .gitlab-ci.yml anpassen

variables:
   KUBECONFIG_SECRET: $KUBECONFIG_SECRET


build-version:       # This job runs in the build stage, which runs first.
  stage: build
  image:
    name: dtzar/helm-kubectl:3.7.1

  script:
    - echo "Show use our repo"
    - cd $CI_PROJECT_DIR
    - ls -la
    - kubectl version --client
    - echo "kubeconfig aufsetzen"
    - mkdir ~/.kube
    - echo "$KUBECONFIG_SECRET" > ~/.kube/config
    - ls -la ~/.kube/config
    - cat ~/.kube/config
    - kubectl cluster-info
    - kubectl get pods
    - echo "Deploying..."
    - kubectl apply -f manifests/deploy.yml
    - sleep 2
    - echo "And now..."
    - kubectl get pods

4. Zugangsdaten auf master-server auslesen und in den Zwischenspeicher kopieren

microk8s config

5. Im Repo und SETTINGS -> CI/CD -> Variables

variable
KUBECONFIG_SECRET
mit Inhalt aus 4. setzen

MASKED und PROTECTED Nicht aktivieren

Speicern

6. im repo folgende Datei anlegen.

## manifests/deploy.yml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: echo-server-deployment-from-gitlab
  labels:
    app: echo-server
spec:
  selector:
    matchLabels:
      app: echo-server
  replicas: 1
  template:
    metadata:
      annotations:
      labels:
        app: echo-server
    spec:
      containers:
        - name: echo-server
          image: hashicorp/http-echo
          imagePullPolicy: IfNotPresent
          args:
            - -listen=:8080
            - -text="hello NEWNEW world"
```

7. in CI/CD - Menü -> Pipelines gucken, ob die Pipeline durchläuft und die detaillierte Ausgabe anzeigen

8. Änderung in deploy.yml durchführen.

z.B. Text: hello NEWNEW world in hello OLDNEW world ändern.

9. Prüfen ob neuer Pod erstellt wird durch überprüfen der Ausgabe in Pipelines

Ingress

Nginx Ingress Controller

Example redirecting/rewriting

- https://medium.com/ww-engineering/kubernetes-nginx-ingress-traffic-redirect-using-annotations-demystified-b7de846fb43d

Snippet: permanent redirect

```
apiVersion: v1
kind: Service
metadata:
  name: my-redirect-service
spec:
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376

---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
   annotations:
     nginx.ingress.kubernetes.io/permanent-redirect: https://www.google.de
     nginx.ingress.kubernetes.io/permanent-redirect-code: '308'
   name: redirect-google.de
spec:
   rules:
   - host: redirect.training.local
     http:
       paths:
       - backend:
           service:
             name: my-redirect-service
             port:
               number: 80
         path: /
         pathType: Prefix
```

Use multiple hosts

```
## this i yaml magic - yaml - ids
spec:
  rules:
  - host: foobar.com
    http: &http_rules
      paths:
      - backend:
          serviceName: foobar
          servicePort: 80
  - host: api.foobar.com
    http: *http_rules
  - host: admin.foobar.com
    http: *http_rules
  - host: status.foobar.com
    http: *http_rules
```

Full example

```
## remember to set in your hosts - file or properly in dns
apiVersion: v1
kind: Service
metadata:
  name: my-redirect-service
spec:
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376

---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
   annotations:
     nginx.ingress.kubernetes.io/permanent-redirect: https://www.google.de
```

```
    nginx.ingress.kubernetes.io/permanent-redirect-code: '308'
  name: redirect-google.de
spec:
  rules:
  - host: redirect.training.local
    http: &http_rules
      paths:
      - backend:
          service:
            name: my-redirect-service
            port:
              number: 80
        path: /
        pathType: Prefix
  - host: redirect2.training.local
    http: *http_rules
```

Ref:

- https://github.com/kubernetes/kubernetes/issues/43633

Documentation

- Github
- Using custom Template

Monitoring

Prometheus Operator

- https://prometheus.io/docs/introduction/overview/

Logging

Logging on Node - Level (fluentd)

- https://medium.com/kubernetes-tutorials/cluster-level-logging-in-kubernetes-with-fluentd-e59aa2b6093a

Netzwerk

Ist ipv6 mit ipsec möglich ? (Stand: Experimentell)

General

- Ist möglich, aber noch nicht produktionsreif (Stand 10/2021)
- Jedes Kubernetes-Cluster cluster braucht einen Netzwerk-Dienst
- By microk8s ist dieser Netzwerkdienst calico
- Dieser unterstützt experimentell ipv mit ipsec

Ref:

- https://docs.projectcalico.org/getting-started/kubernetes/vpp/ipsec
- https://docs.projectcalico.org/getting-started/kubernetes/vpp/getting-started

Metallb als LoadBalancer vor Ingress verwenden

Example:

```
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    address-pools:
    - name: default
      protocol: layer2
      addresses:
      - 203.0.113.10-203.0.113.15
```

Refs:

- https://kubernetes.github.io/ingress-nginx/deploy/baremetal/
- https://metallb.universe.tf/

Calico (Default Netzwerk-Dienst microk8s) - Netzwerk Policies

Unterstützt Netzwerk Policys

- Ich kann festlegen, welcher Pod mit welchem Pod kommunizieren

Ref:

- https://docs.oracle.com/de-de/iaas/Content/ContEng/Tasks/contengsettingupcalico.htm

Shared Volumes

Shared Volumes with Rook & Ceph

General

- cephfs arbeitet als Objektspeicher (RADOS)
- Es wird eine Schnittstelle als Blockspeicher zur Verfügung gestellt
- Ceph verwendet intern Monitoring - Server (MONs)
- Um ceph in Kubernetes auszurollen sind MON's und OSDs notwendig

Rook

- Rook liegt zwischen Ceph und Kubernetes und kümmert sich um dessen Verwaltung praktisch komplett automatisch.
- Ein Rook-Cluster mit laufendem CEPH entsteht, wenn
  - die vorgefertigten Rook-Definitionen aus dessen Git-Repository auf die Kubernetes-Instanz angewendet wird.

RADOS

- RADOS hat mehrere Komponenten
- Die Object Storage Daemons (OSDs) sind die Speichersilos in Ceph

Please do not/Be careful (as of: 2019/12), not sure, if this is still the case (2021/10)!

- Bitte nicht über HELM ausrollen, dies ist (Stand: 11/2021) nicht funktional komplett

Walkthrough

```
#### Achtung: Stand 2021/10 walkthrough funktioniert aktuell nicht

## Do this on your client
git clone https://github.com/rook/rook.git
cd rook/cluster/examples/kubernetes/ceph
kubectl create -f crds.yaml -f common.yaml -f operator.yaml

## Check if pod is running
kubectl -n rook-ceph get pod -l app=rook-ceph-operator

## First proceed if pod is running (last action)
kubectl create -f cluster.yaml
kubectl -n rook-ceph get pods
## Wait till all pods are ready

## Install the toolbox
kubectl create -f toolbox.yaml

kubectl -n rook-ceph exec -it deploy/rook-ceph-tools -- bash
```

Ref:

- https://faun.pub/deploy-rook-ceph-on-kubernetes-3a2252f3732e (best choice for walkthrough)
- https://www.linux-magazin.de/ausgaben/2019/12/rook-2/4/

Shared Volumes with nfs

Tipps & Tricks

kubectl-Autovervollständigung

Walkthrough

```
echo 'source <(kubectl completion bash)' >>~/.bashrc
source ~/.bashrc

## works like
kubectl TAB TAB
```

Ref:

- https://kubernetes.io/docs/tasks/tools/included/optional-kubectl-configs-bash-linux/

Install plugin kubectl-convert

Why ?

- It used to be part of kubernetes
- It was deprecated like so because it was not cleanly implemented
- Ref: https://github.com/kubernetes/kubectl/issues/725#:~:text=kubectl%20convert%20is%20deprecated,functionality%20is%20removed%20from%20kubectl%2(

What does it do ?

```
If converts old version of your manifests
to the versions that work on your server
```

Walkthrough

```
curl -LO https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl-convert
## verify checksum
curl -LO "https://dl.k8s.io/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl-convert.sha256"
echo "$(<kubectl-convert.sha256) kubectl-convert" | sha256sum --check
## install it to the appropriate place being in $PATH
sudo install -o root -g root -m 0755 kubectl-convert /usr/local/bin/kubectl-convert
## Test it
kubectl-convert --help
```

Examples

Kuard pod

kuard-pod.yml

```
apiVersion: v1
kind: Pod
metadata:
  name: kuard
spec:
  containers:
  - image: gcr.io/kuar-demo/kuard-amd64:1
    name: kuard
    ports:
    - containerPort: 8080
      name: http
      protocol: TCP
```

kuard-pod apply

```
kubectl apply -f kuard-pod.yml
```

Pod nginx port exposed

What is containerPort (from kubectl explain) ?

```
containerPort        <integer> -required-
     Number of port to expose on the pod's IP address. This must be a valid port
     number, 0 < x < 65536.
```

Walkthrough

```
vi nginx-static-expose.yml

apiVersion: v1
kind: Pod
metadata:
  name: nginx-static-web
  labels:
    webserver: nginx
spec:
  containers:
  - name: web
    image: nginx
    ports:
    - name: web
      containerPort: 80
      protocol: TCP
```

```
kubectl apply -f nginx-static-expose.yml
kubectl describe nginx-static-web
## show config
kubectl get pod/nginx-static-web -o yml
```

Adding a service

Deployment nginx

```
## vi nginx-deployment.yml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2 # tells deployment to run 2 pods matching the template
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80
```

```
kubectl apply -f nginx-deployment.yml
```

Ingress Nginx

Walkthrough

```
mkdir apple-banana-ingress

## apple.yml
## vi apple.yml
kind: Pod
apiVersion: v1
metadata:
  name: apple-app
  labels:
    app: apple
spec:
  containers:
    - name: apple-app
      image: hashicorp/http-echo
      args:
        - "-text=apple"
---

kind: Service
apiVersion: v1
metadata:
  name: apple-service
spec:
  selector:
    app: apple
  ports:
    - port: 80
    - targetPort: 5678 # Default port for image
```

```
kubectl apply -f apple.yml
```

```
## banana
## vi banana.yml
kind: Pod
apiVersion: v1
metadata:
  name: banana-app
  labels:
    app: banana
spec:
  containers:
    - name: banana-app
      image: hashicorp/http-echo
      args:
        - "-text=banana"

---

kind: Service
apiVersion: v1
metadata:
  name: banana-service
spec:
  selector:
    app: banana
  ports:
    - port: 80
      targetPort: 5678 # Default port for image
```

```
kubectl apply -f banana.yml
```

```
## Ingress
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: example-ingress
  annotations:
    ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - http:
      paths:
        - path: /apple
          backend:
```

```
            serviceName: apple-service
            servicePort: 80
        - path: /banana
          backend:
            serviceName: banana-service
            servicePort: 80
```

```
## ingress
kubectl apply -f ingress.yml
kubectl get ing
```

Reference

- https://matthewpalmer.net/kubernetes-app-developer/articles/kubernetes-ingress-guide-nginx-example.html

Find the problem

```
## Hints

## 1. Which resources does our version of kubectl support
## Can we find Ingress as "Kind" here.
kubectl api-ressources

## 2. Let's see, how the configuration works
kubectl explain --api-version=networking.k8s.io/v1 ingress.spec.rules.http.paths.backend.service

## now we can adjust our config
```

Solution

```
## in kubernetes 1.22.2 - ingress.yml needs to be modified like so.
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: example-ingress
  annotations:
    ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - http:
      paths:
        - path: /apple
          pathType: Prefix
          backend:
            service:
              name: apple-service
              port:
                number: 80
        - path: /banana
          pathType: Prefix
          backend:
            service:
              name: banana-service
              port:
                number: 80
```

Combind example in manifest (ubuntu-nginx)

Combined example in manifest (ubuntu-nginx-service)

Use Cases

Use Cases Foundation

- https://www.cncf.io/case-studies/

Use Cases Kubernetes

- https://kubernetes.io/case-studies/

Use Cases

- https://dzone.com/articles/how-big-companies-are-using-kubernetes