Kubernetes Networking

Agenda

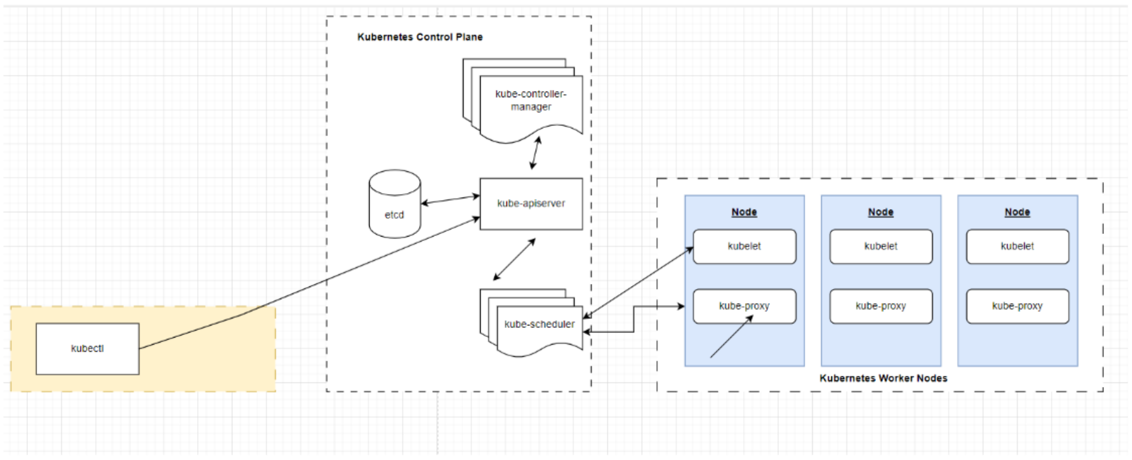
Backlog

Backlog

Kubernetes - Überblick

Aufbau Allgemein

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 (bereitgestellt vom kube-api-server)

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 (former: master node) - components

Worker Node - 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

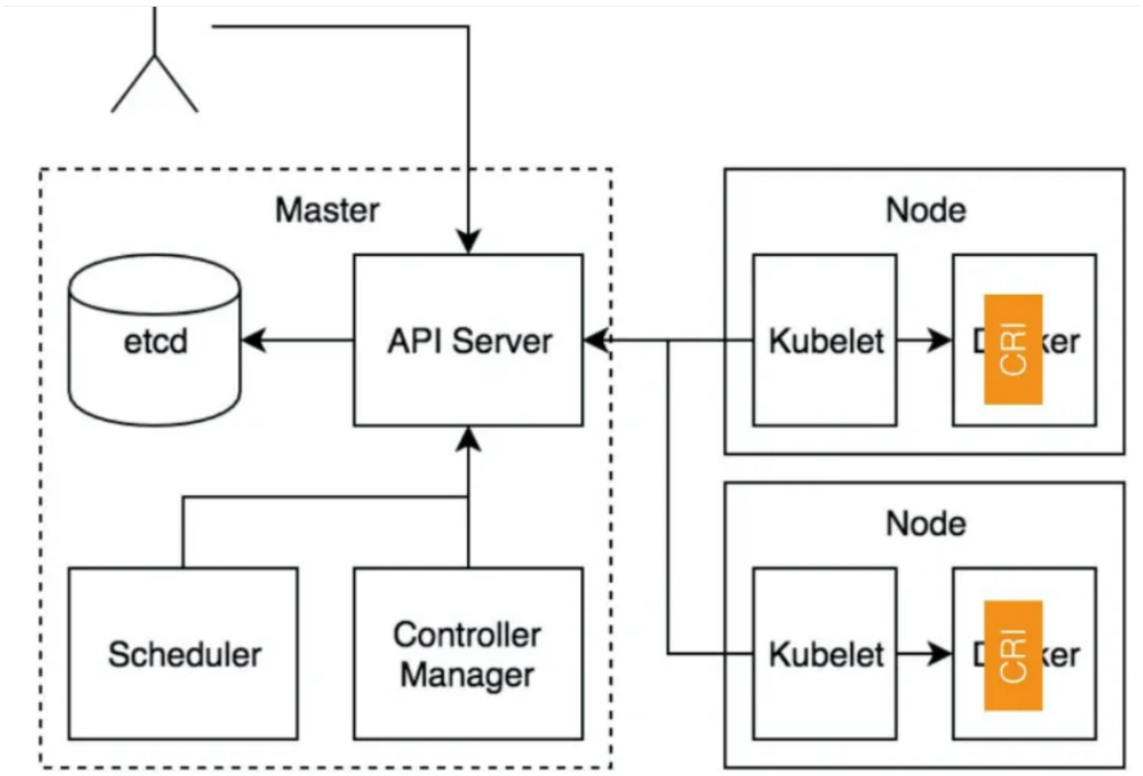
- https://www.redhat.com/de/topics/containers/kubernetes-architecture

Structure Kubernetes Deep Dive

- https://github.com/jmetzger/training-kubernetes-advanced/assets/1933318/1ca0d174-f354-43b2-81cc-67af8498b56c

CRI - Container Runtime interface

Where is it embedded



What is it for ?

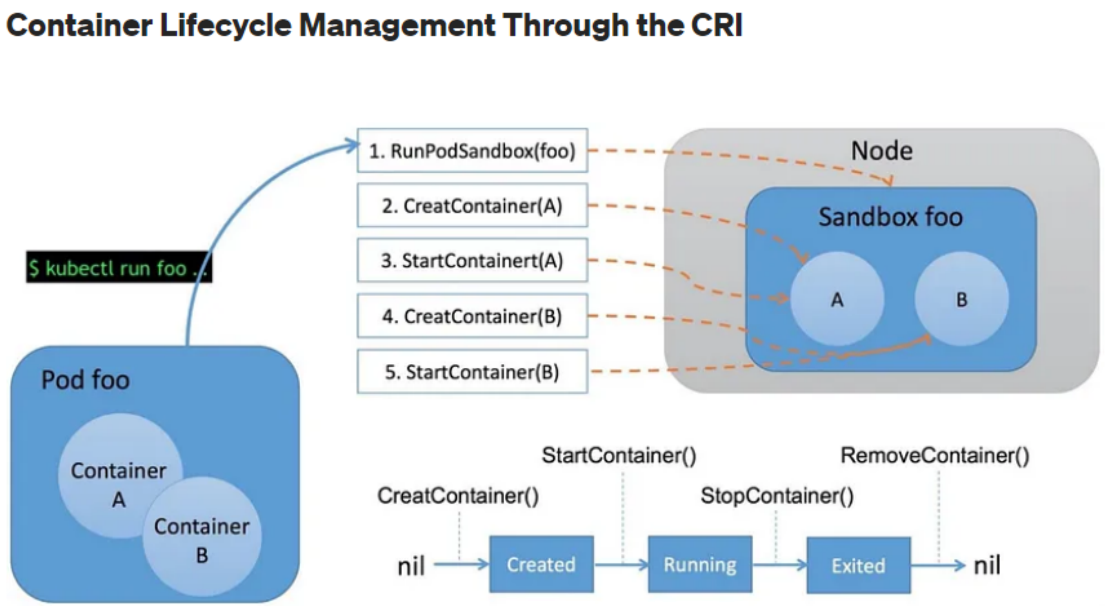
- Abstraction layer called by kubelet to make it possible to use other container runtimes
- The CRI uses gRPC as its communication protocol.

kubelet calls the CRI with its subcommands

- Expected commands are

```
Sandbox:
  Delete
  Create
  List
Image:
  Pull
  List
Container.
  Create
  Start
  Exec
```

Steps in the CRI



Ports und Protokolle

- https://kubernetes.io/docs/reference/networking/ports-and-protocols/

Kubernetes - Misc

Wann wird podIP vergeben ?

Example (that does work)

```
## Show the pods that are running
kubectl get pods

## Synopsis (most simplistic example
## kubectl run NAME --image=IMAGE_EG_FROM_DOCKER
## example
kubectl run nginx --image=nginx:1.23

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

Example (that does not work)

```
kubectl run foo2 --image=foo2
## ImageErrPull - Image konnte nicht geladen werden
kubectl get pods
## Weitere status - info
kubectl describe pods foo2
```

Ref:

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

Bash completion installieren

Walkthrough

```
## Eventuell, wenn bash-completion nicht installiert ist.
apt install bash-completion
source /usr/share/bash-completion/bash_completion
## is it installed properly
type _init_completion
```

```
## activate for all users
kubectl completion bash | sudo tee /etc/bash_completion.d/kubectl > /dev/null

## verifizieren - neue login shell
```

```
su -

## zum Testen
kubectl g<TAB>
kubectl get
```

Alternative für k als alias für kubectl

```
source <(kubectl completion bash)
complete -F __start_kubectl k
```

Reference

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

kubectl verbindung mit namespace einrichten

config einrichten

```
cd
mkdir .kube
cd .kube
cp -a /tmp/config config
ls -la
## nano config befüllen
## das bekommt ihr aus Eurem Cluster Management Tool
```

```
kubectl cluster-info
```

Arbeitsbereich konfigurieren

```
kubectl create ns jochen
kubectl get ns
kubectl config set-context --current --namespace jochen
kubectl get pods
```

vim support for yaml

Ubuntu (im Unterverzeichnis /etc/vim/vimrc.local - systemweit)

```
hi CursorColumn cterm=NONE ctermbg=lightred ctermfg=white
autocmd FileType y?ml setlocal ts=2 sts=2 sw=2 ai number expandtab cursorline cursorcolumn
```

Testen

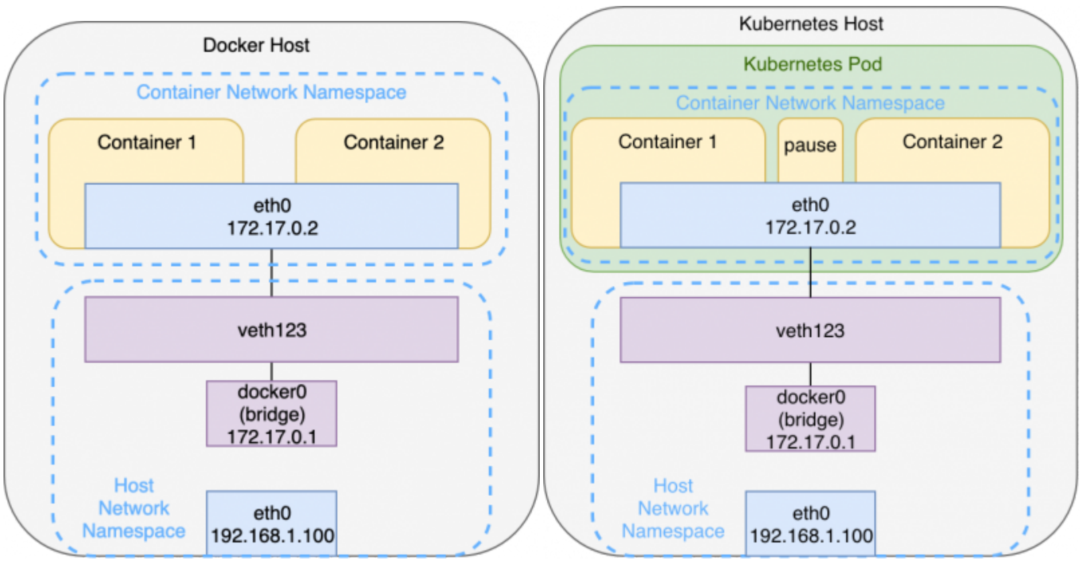
```
vim test.yml
Eigenschaft: <return> # springt eingerückt in die nächste Zeile um 2 spaces eingerückt

## evtl funktioniert vi test.yml auf manchen Systemen nicht, weil kein vim (vi improved)
```

Kubernetes - Netzwerk (CNI's) / Mesh

Netzwerk Interna

Network Namespace for each pod

Overview



General

- Each pod will have its own network namespace
  - with routing, networkdevices
- Connection to default namespace to host is done through veth - Link to bridge on host network
  - similar like on docker to docker0

```
 Each container is connected to the bridge via a veth-pair. This interface pair functions like a virtual point-to-point ethernet
connection and connects the network namespaces of the containers with the network namespace of the host
```

- Every container is in the same Network Namespace, so they can communicate through localhost
  - Example with hashicorp/http-echo container 1 and busybox container 2

Pod-To-Pod Communication (across nodes)

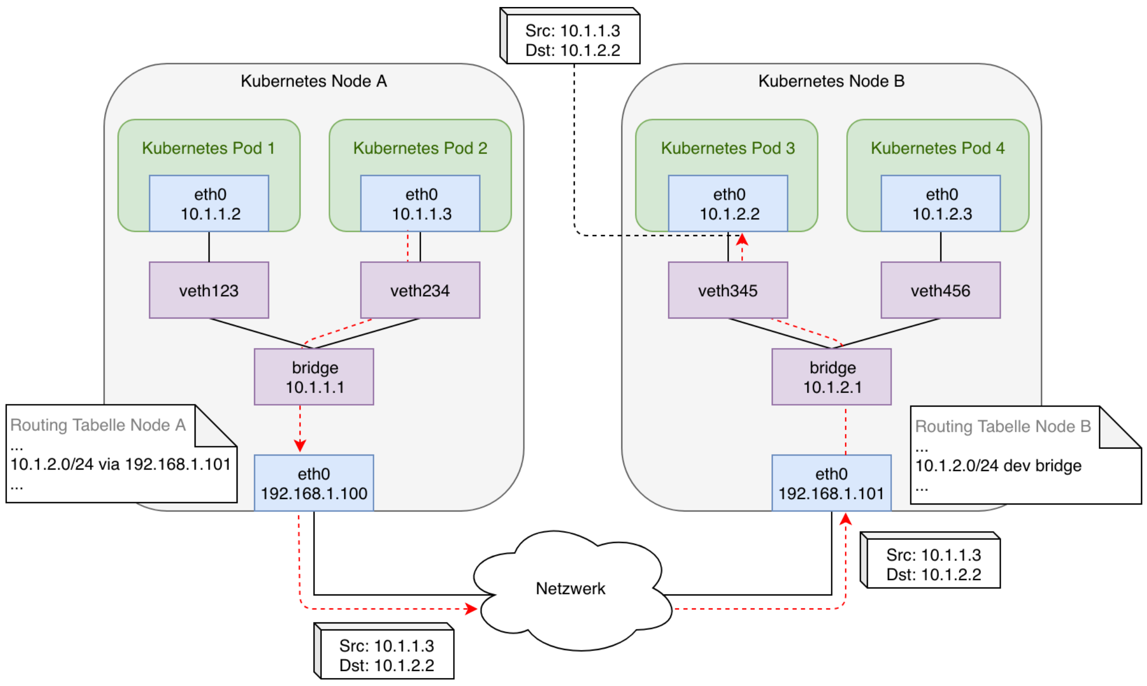
Prerequisites

- pods on a single node as well as pods on a topological remote can establish communication at all times
- Each pod receives a unique IP address, valid anywhere in the cluster. Kubernetes requires this address to not be subject to network address translation (NAT)
- Pods on the same node through virtual bridge (see image above)

General (what needs to be done) - and could be done manually

- local bridge networks of all nodes need to be connected
- there needs to be an IPAM (IP-Address Managemenet) so addresses are only used once
- The need to be routes so, that each bridge can communicate with the bridge on the other network
- Plus: There needs to be a rule for incoming network
- Also: A tunnel needs to be set up to the outside world.

General - Pod-to-Pod Communication (across nodes) - what would need to be done



General - Pod-to-Pod Communication (side-note)

- This could of cause be done manually, but it is too complex
- So Kubernetes has created an Interface, which is well defined
  - The interface is called CNI (common network interface)
  - Funtionally is achieved through Network Plugin (which use this interface)
    - e.g. calico / cilium / weave net / flannel

CNI

- CNI only handles network connectivity of container and the cleanup of allocated resources (i.e. IP addresses) after containers have been deleted (garbage collection) and therefore is lightweight and quite easy to implement.
- There are some basic libraries within CNI which do some basic stuff.

Hidden Pause Container

What is for ?

- Holds the network - namespace for the pod
- Gets started first and falls asleep later
- Will still be there, when the other containers die

```
cd
mkdir -p manifests
cd manifests
mkdir pausetest
cd pausetest
nano 01-nginx.yml
```

```
## vi nginx-static.yml

apiVersion: v1
kind: Pod
metadata:
  name: nginx-pausetest
  labels:
    webserver: nginx:1.21
spec:
  containers:
  - name: web
    image: nginx
```

```
kubectl apply -f .
## als root auf dem worker node
ctr -n k8s.io c list | grep pause
```

References

- https://www.inovex.de/de/blog/kubernetes-networking-part-1-en/
- https://www.inovex.de/de/blog/kubernetes-networking-2-calico-cilium-weavenet/

Wirkweise cni

Ablauf

- Containerd ruft CNI plugin über subcommandos: ADD, DEL, CHECK, VERSION auf (mehr subcommandos gibt es nicht)
- Was gemacht werden soll wird über JSON-Objekt übergeben
- Die Antwort kommt auch wieder als JSON zurück

Plugins die Standardmäßig schon da sind

- https://www.cni.dev/plugins/current/

CNI-Provider

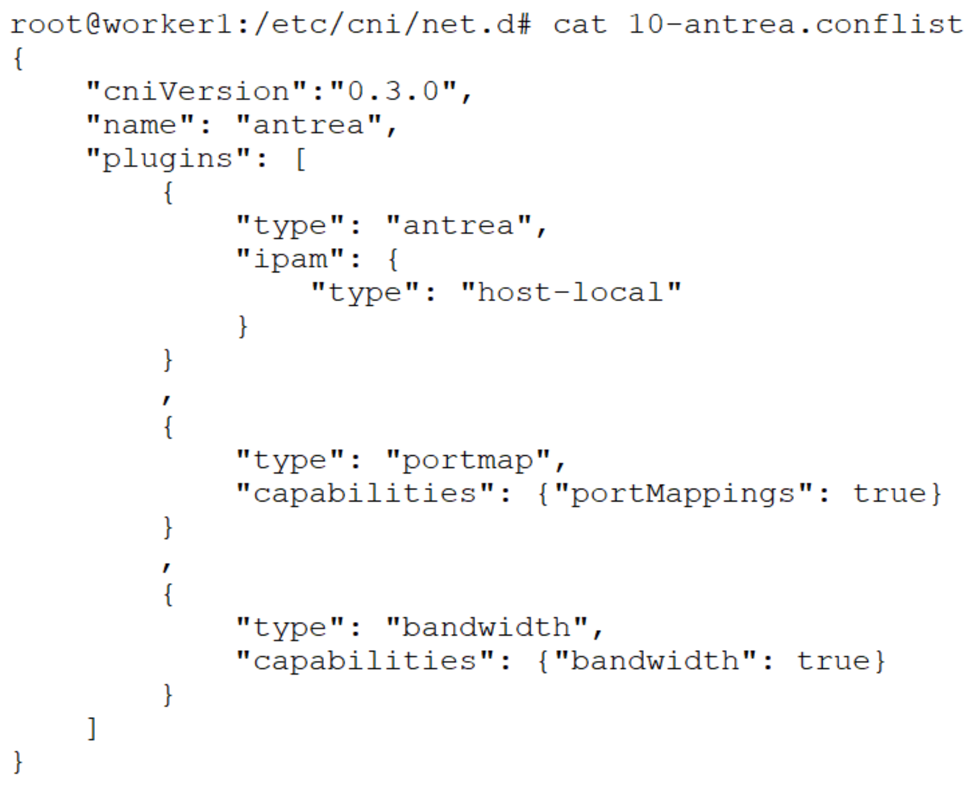
- Ein Kubernetes-Cluster braucht immer ein CNI-Provider, sonst funktioniert die Kmmunikation nicht und die Nodes im Cluster stehen auf NotReady
- Beispiele: Calico, WeaveNet, Antrea, Cilium, Flannel

IPAM - IP Address Management

- Ziel ist, dass Adressen nicht mehrmals vergeben werden.
- Dazu wird ein Pool bereitgestellt.
- Es gibt 3 CNI IPAM - Module:
  - host-local
  - dhcp
  - static

```
* IPAM: IP address allocation
dhcp : Runs a daemon on the host to make DHCP requests on behalf of a container
host-local : Maintains a local database of allocated IPs
static : Allocates static IPv4/IPv6 addresses to containers
```

Beispiel json für antrea (wird verwendet beim Aufruf von CNI)



```
root@worker1:/etc/cni/net.d# cat 10-antrea.conflist
{
    "cniVersion":"0.3.0",
    "name": "antrea",
    "plugins": [
        {
            "type": "antrea",
            "ipam": {
                "type": "host-local"
            }
        }
        ,
        {
            "type": "portmap",
            "capabilities": {"portMappings": true}
        }
        ,
        {
            "type": "bandwidth",
            "capabilities": {"bandwidth": true}
        }
    ]
}
```

Übersicht Netzwerke

CNI

- Common Network Interface
- Feste Definition, wie Container mit Netzwerk-Bibliotheken kommunizieren

Docker - Container oder andere

- Container wird hochgefahren -> über CNI -> zieht Netzwerk - IP hoch.

- Container witd runtergahren -> uber CNI -> Netzwerk - IP wird released

Welche gibt es ?

- Flannel
- Canal
- Calico
- Cilium
- Antrea (vmware)
- Weave Net

Flannel

Generell

- Flannel is a CNI which gives a subnet to each host for use with container runtimes.

Overlay - Netzwerk

- virtuelles Netzwerk was sich oben drüber und eigentlich auf Netzwerkebene nicht existiert
- VXLAN

Vorteile

- Guter einfacher Einstieg
- reduziert auf eine Binary flanneld

Nachteile

- keine Firewall - Policies möglich
- keine klassichen Netzwerk-Tools zum Debuggen möglich.

Guter Einstieg in flannel

- https://mvallim.github.io/kubernetes-under-the-hood/documentation/kube-flannel.html

Canal

General

- Auch ein Overlay - Netzwerk
- Unterstützt auch policies
- Kombination aus Flannel (Overlay) und den NetworkPolicies aus Calico

Calico

calico

Komponenten

Calico API server

- Lets you manage Calico resources directly with kubectl.

Felix

```
Main task: Programs routes and ACLs, and anything else required on the host to provide desired connectivity for the endpoints on
that host. Runs on each machine that hosts endpoints. Runs as an agent daemon.
```

BIRD

- Gets routes from Felix and distributes to BGP peers on the network for inter-host routing. Runs on each node that hosts a Felix agent. Open source, internet routing daemon.

confd

```
Monitors Calico datastore for changes to BGP configuration and global defaults such as AS number, logging levels, and IPAM
information. Open source, lightweight configuration management tool.

Confd dynamically generates BIRD configuration files based on the updates to data in the datastore. When the configuration file
changes, confd triggers BIRD to load the new files
```

Dikastes

```
Enforces NetworkPolicy for istio service mesh
```

CNI plugin

Datastore plugin

IPAM plugin

kube-controllers

```
Main task: Monitors the Kubernetes API and performs actions based on cluster state. kube-controllers.

The tigera/kube-controllers container includes the following controllers:

Policy controller
Namespace controller
```

```
Serviceaccount controller
Workloadendpoint controller
Node controller
```

Typha

```
Typha maintains a single datastore connection on behalf of all of its clients like Felix and confd. It caches the datastore state
and deduplicates events so that they can be fanned out to many listeners.
```

calicoctl

- Wird heute selten gebraucht, da das meiste heute mit kubectl über den Calico API Server realisiert werden kann
- Früher haben die neuesten NetworkPolicies/v3 nur über calioctl funktioniert

Generell

- klassische Netzwerk (BGP) - kein Overlay
- klassische Netzwerk-Tools können verwendet werden.
- eBPF ist implementiert, aber muss aktiviert

Vorteile gegenüber Flannel

- Policy über Kubernetes Object (NetworkPolicies)

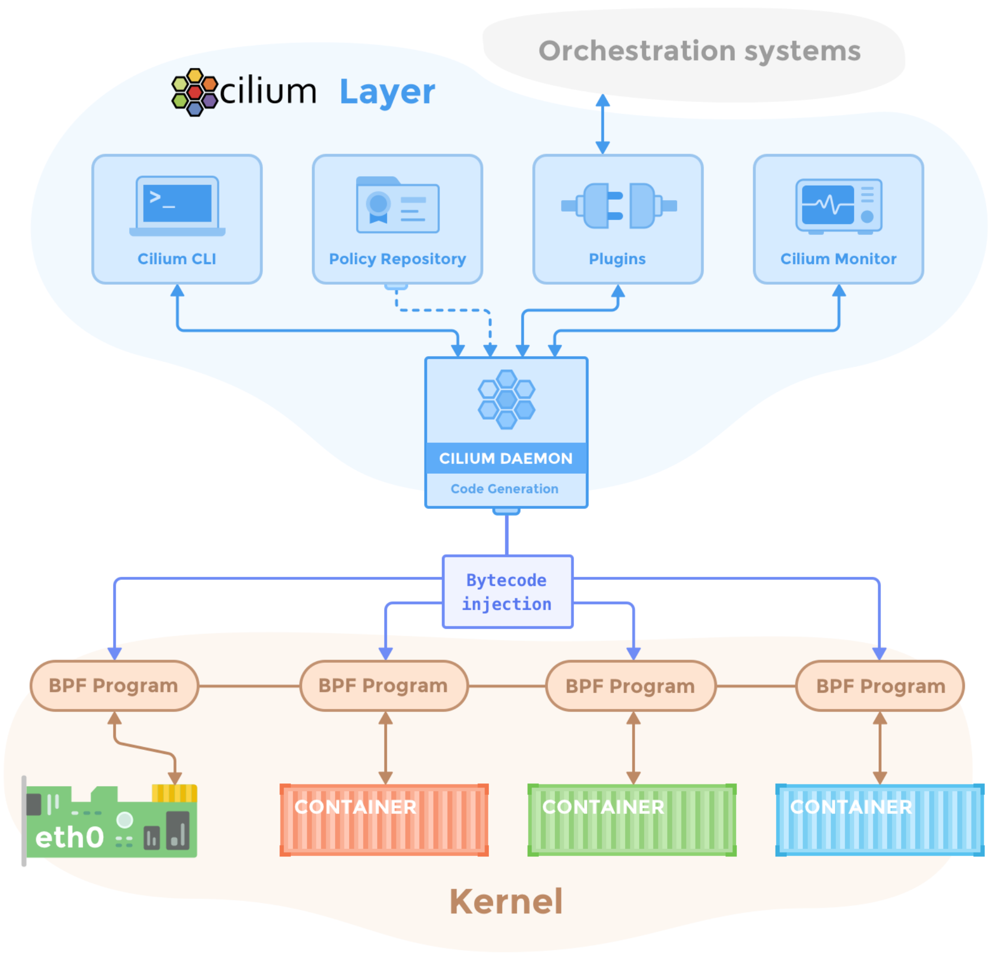
Vorteile

- ISTIO integrierbar (Mesh - Netz)
- Performance etwas besser als Flannel (weil keine Encapsulation)

Referenz

- https://projectcalico.docs.tigera.io/security/calico-network-policy

Cilium



Komponenten:

Cilium Agent

- Läuft auf jeder Node im Cluster
- Lauscht auf events from Orchestrierer (z.B. container gestoppt und gestartet)
- Managed die eBPF - Programme, die Linux kernel verwendet um den Netzwerkzugriff aus und in die Container zu kontrollieren

Client (CLI)

- Wird im Agent mit installiert (interagiert mit dem agent auf dem gleichen Node)
- Kann aber auch auf dem Client installiert werden auf dem kubectl läuft.

Cilium Operator

- Zuständig dafür, dass die Agents auf den einzelnen Nodes ausgerollt werden
- Es gibt ihn nur 1x im Cluster
- Ist unkritisch, sobald alles ausgerollt ist.
    - wenn dieser nicht läuft funktioniert das Networking trotzdem

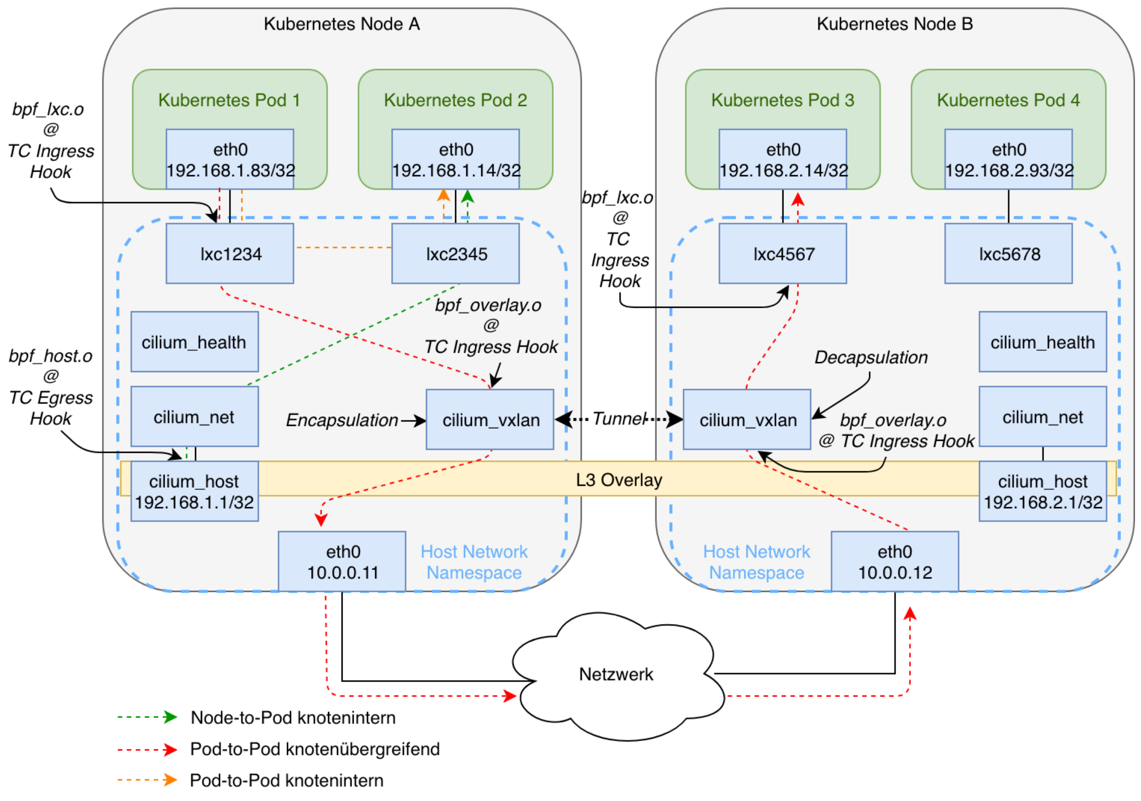
cilium CNI - Plugin

- Ist ein binary auf dem server (worker)
- wird durch die Container Runtime ausgeführt.
- cilium cni plugin interagiert mit der Cilium API auf dem Node

Datastore

- Daten werden per Default in CRD (Custom Resource Defintions) gespeichert
- Diese Resource Objekte werden von Cilium definiert und angelegt.
    - Wenn Sie angelegt sind, sind die Daten dadurch automatisch im etc - Speicher
    - Mit der weiteren Möglichkeit den Status zu speichern.
- Alternative: Speichern der Daten direkt in etcd

Generell



- Quelle: https://www.inovex.de/de/blog/kubernetes-networking-2-calico-cilium-weavenet/
- Verwendet keine Bridge sondern Hooks im Kernel, die mit eBPF aufgesetzt werden
    - Bessere Performance
- eBPF wird auch für NetworkPolicies unter der Haube eingesetzt
- Mit Ciliums Cluster Mesh lassen sich mehrere Cluster miteinander verbinden:

Vorteile

- Höhere Leistung mit eBPF-Ansatz. (extended Berkely Packet Filter)
    - JIT - Just in time compiled -
    - Bytecode wird zu MaschineCode kompiliert (Miniprogramme im Kernel)
- Ersatz für iptables (wesentlich schneller und keine Degredation wie iptables ab 5000 Services)

- Gut geeignet für größere Cluster

Weave Net

- Ähnlich calico
- Verwendet overlay netzwerk
- Sehr stabil bzgl IPV4/IPV6 (Dual Stack)
- Sehr grosses Feature-Set
- mit das älteste Plugin

DNS - Resolution - Services

```
kubectl run podtest --rm -ti --image busybox -- /bin/sh
If you don't see a command prompt, try pressing enter.
/ # wget -O - http://apple-service.jochen
Connecting to apple-service.jochen (10.245.39.214:80)
writing to stdout
apple-tln1
-                    100%
|*************************************************************************************************************|    11  0:00:00
ETA
written to stdout
/ # wget -O - http://apple-service.jochen.svc.cluster.local
Connecting to apple-service.jochen.svc.cluster.local (10.245.39.214:80)
writing to stdout
apple-tln1
-                    100%
|*************************************************************************************************************|    11  0:00:00
ETA
written to stdout
/ # wget -O - http://apple-service
Connecting to apple-service (10.245.39.214:80)
writing to stdout
apple-tln1
-                    100%
|*************************************************************************************************************|    11  0:00:00
ETA
written to stdout
```

Kubernetes NetworkPolicy

Einfache Übung Network Policy

Schritt 1: Deployment und Service erstellen

```
KURZ=jm
kubectl create ns policy-demo-$KURZ
```

```
cd
mkdir -p manifests
cd manifests
mkdir -p np
cd np
```

```
## nano 01-deployment.yml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 1
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.23
        ports:
        - containerPort: 80
```

```
kubectl -n policy-demo-$KURZ apply -f .
```

```
## nano 02-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx
spec:
  type: ClusterIP # Default Wert
  ports:
  - port: 80
    protocol: TCP
  selector:
    app: nginx
```

```
kubectl -n policy-demo-$KURZ apply -f .
```

Schritt 2: Zugriff testen ohne Regeln

```
## lassen einen 2. pod laufen mit dem auf den nginx zugreifen
kubectl run --namespace=policy-demo-$KURZ access --rm -ti --image busybox
```

```
## innerhalb der shell
wget -q nginx -O -
```

```
## Optional: Pod anzeigen in 2. ssh-session zu jump-host
kubectl -n policy-demo-$KURZ get pods --show-labels
```

Schritt 3: Policy festlegen, dass kein Zugriff erlaubt ist.

```
## nano 03-default-deny.yaml
## Schritt 2: Policy festlegen, dass kein Ingress-Traffic erlaubt
## in diesem namespace: policy-demo-$KURZ
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: default-deny
spec:
  podSelector:
    matchLabels: {}
```

```
kubectl -n policy-demo-$KURZ apply -f .
```

Schritt 3.5: Verbindung mit deny all Regeln testen

```
kubectl run --namespace=policy-demo-$KURZ access --rm -ti --image busybox
```

```
## innerhalb der shell
wget -q nginx -O -
```

Schritt 4: Zugriff erlauben von pods mit dem Label run=access (alle mit run gestarteten pods mit namen access haben dieses label per default)

```
## nano 04-access-nginx.yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: access-nginx
spec:
  podSelector:
    matchLabels:
      app: nginx
  ingress:
    - from:
      - podSelector:
          matchLabels:
            run: access
```

```
kubectl -n policy-demo-$KURZ apply -f .
```

Schritt 5: Testen (zugriff sollte funktionieren)

```
## lassen einen 2. pod laufen mit dem auf den nginx zugreifen
## pod hat durch run -> access automatisch das label run:access zugewiesen
kubectl run --namespace=policy-demo-$KURZ access --rm -ti --image busybox
```

```
## innerhalb der shell
wget -q nginx -O -
```

Schritt 6: Pod mit label run=no-access - da sollte es nicht gehen

```
kubectl run --namespace=policy-demo-$KURZ no-access --rm -ti --image busybox
```

```
## in der shell
wget -q nginx -O -
```

Schritt 7: Aufräumen

```
kubectl delete ns policy-demo-$KURZ
```

Ref:

- https://projectcalico.docs.tigera.io/security/tutorials/kubernetes-policy-basic

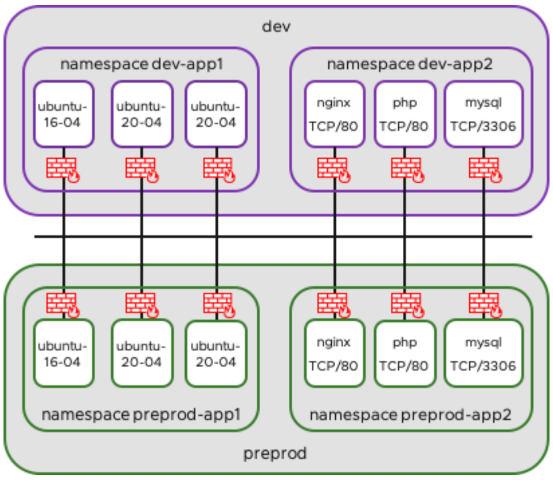
NetworkPolicy from IPBlock

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 1
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: traefik/whoami
        ports:
        - containerPort: 80
---
## nano 02-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx
spec:
  type: NodePort # Default Wert
  ports:
  - port: 80
    protocol: TCP
  selector:
    app: nginx
---
## nano 03-default-deny.yaml
## Schritt 2: Policy festlegen, dass kein Ingress-Traffic erlaubt
## in diesem namespace: policy-demo-$KURZ
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: default-deny
spec:
  podSelector:
    matchLabels: {}
---
## nano 05-from-access.yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: access-nginx
spec:
  podSelector:
    matchLabels:
      app: nginx
  ingress:
    - from:
      - ipBlock:
          cidr: 138.197.181.70/32
```

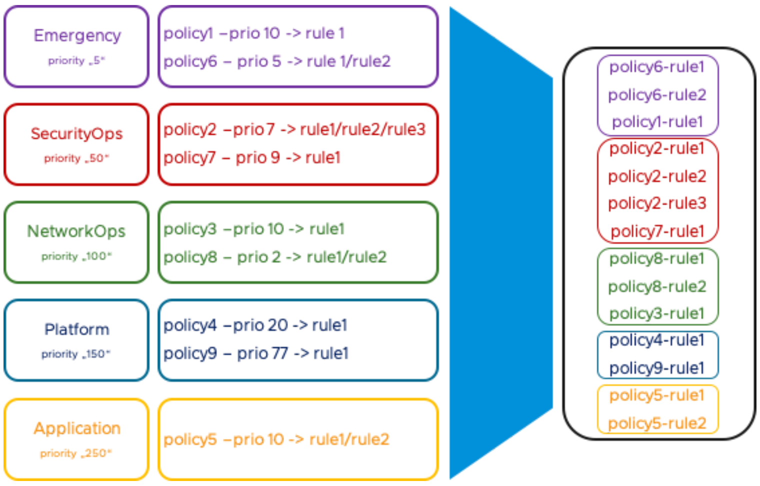
Kubernetes (Antrea-)NetworkPolicy

Antrea NetworkPolicy Exercise - Each trainee has its own cluster

Our Goal

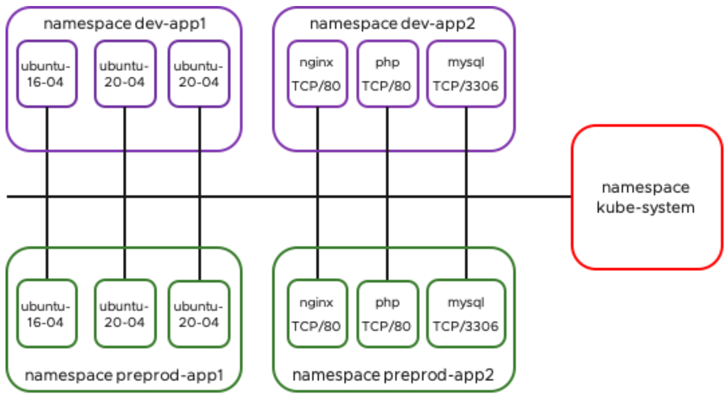


How the order of priorities work



Our Setup

```
In app1 are some Ubuntu Servers for Testing: dev-app1 / preprod-app1

1x Ubuntu Server 16.04
2x Ubuntu Server 20.04
In app2 is a simple 3 Tier-App (WEB-APP-DB): dev-app2 / preprod-app2 (3tier-app)

1x nginx TCP/80 (NodePort)
1x php TCP/80 (ClusterIP)
1x mysql TCP/3306 (ClusterIP)
```



Step 1: Rollout the pods (dev-app1/dev-app2)

```
cd
mkdir -p manifests
cd manifests
mkdir 10-antrea
cd 10-antrea
nano 01-pods-dev-app1-app2.yaml
```

```
apiVersion: v1
kind: Namespace
metadata:
  name: dev-app1
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ubuntu-16-04
  labels:
    app: ubuntu-16-04
  namespace: dev-app1
spec:
  replicas: 1
  selector:
    matchLabels:
      app: ubuntu-16-04
  template:
    metadata:
      labels:
        app: ubuntu-16-04
    spec:
      containers:
      - name: ubuntu-16-04
        image: ubuntu:16.04
        imagePullPolicy: IfNotPresent
        command: [ "/bin/bash", "-c" ]
        args:
          - apt-get update;
            apt-get install iputils-ping -y;
            apt-get install net-tools;
            apt-get install curl -y;
            sleep infinity;
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ubuntu-20-04
  labels:
    app: ubuntu-20-04
  namespace: dev-app1
spec:
  replicas: 2
  selector:
    matchLabels:
      app: ubuntu-20-04
  template:
    metadata:
      labels:
        app: ubuntu-20-04
    spec:
      containers:
      - name: ubuntu-20-04
        image: ubuntu:20.04
        imagePullPolicy: IfNotPresent
        command: [ "/bin/bash", "-c" ]
        args:
          - apt-get update;
            apt-get install tcpdump -y;
            apt-get install telnet -y;
            apt-get install iputils-ping -y;
            apt-get install nmap -y;
            apt-get install net-tools;
            apt-get install netdiscover -y;
            apt-get install mysql-client -y;
            apt-get install curl -y;
            apt-get install dsniff -y;
            sleep infinity;
---
apiVersion: v1
kind: Namespace
metadata:
  name: dev-app2
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: default-conf
  namespace: dev-app2
data:
  default.conf: |
    server {
    listen 80 default_server;

    location / {
      proxy_pass http://app-service;
      proxy_http_version 1.1;
    }

    error_page   500 502 503 504  /50x.html;
    location = /50x.html {
        root   /usr/share/nginx/html;
    }

    }
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  namespace: dev-app2
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
        service: web
        kind: dev
        type: internal
    spec:
      containers:
      - name: nginx
        image: nginx
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 80
        volumeMounts:
        - mountPath: /etc/nginx/conf.d # mount nginx-conf volumn to /etc/nginx
          readOnly: true
          name: default-conf
        - mountPath: /var/log/nginx
          name: log
      volumes:
```

```yaml
      - name: default-conf
        configMap:
          name: default-conf # place ConfigMap `nginx-conf` on /etc/nginx
          items:
            - key: default.conf
              path: default.conf
      - name: log
        emptyDir: {}
---
apiVersion: v1
kind: Service
metadata:
  name: nginx
  namespace: dev-app2
spec:
  type: NodePort
  ports:
  - port: 80
    targetPort: 80
  selector:
    app: nginx
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: appserver
  labels:
    app: app
  namespace: dev-app2
spec:
  replicas: 1
  selector:
    matchLabels:
      app: app
  template:
    metadata:
      labels:
        app: app
        kind: dev
        type: internal
    spec:
      containers:
      - name: php-apache
        image: derstich/miserver:006
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: app-service
  labels:
    app: app
  namespace: dev-app2
spec:
  ports:
  - port: 80
    protocol: TCP
  selector:
    app: app
---
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: mysql
  namespace: dev-app2
spec:
  selector:
    matchLabels:
      app: mysql8
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: mysql8
        service: db
        kind: dev
        type: internal
    spec:
      containers:
      - image: mysql:5.6
        name: mysql
        imagePullPolicy: IfNotPresent
        env:
        - name: MYSQL_ROOT_PASSWORD
          value: .sweetpwd.
        - name: MYSQL_DATABASE
          value: my_db
        - name: MYSQL_USER
          value: db_user
        - name: MYSQL_PASSWORD
          value: .mypwd
        args: ["--default-authentication-plugin=mysql_native_password"]
        ports:
        - containerPort: 3306
          name: mysql8
---
apiVersion: v1
kind: Service
metadata:
  name: mysql8-service
  labels:
    app: mysql8
  namespace: dev-app2
spec:
  type: ClusterIP
  ports:
  - port: 3306
    protocol: TCP
  selector:
    app: mysql8
```

```
kubectl apply -f .
kubectl -n dev-app1 get all
kubectl -n dev-app2 get all
```

Schritt 2: rollout preprod-app1/preprod-app2

```
nano 02-deployment-preprod-app1-app2.yaml
```

```
apiVersion: v1
kind: Namespace
metadata:
  name: preprod-app1
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ubuntu-16-04
  labels:
    app: ubuntu-16-04
  namespace: preprod-app1
spec:
  replicas: 1
  selector:
    matchLabels:
      app: ubuntu-16-04
  template:
    metadata:
      labels:
        app: ubuntu-16-04
    spec:
      containers:
      - name: ubuntu-16-04
        image: ubuntu:16.04
        imagePullPolicy: IfNotPresent
        command: [ "/bin/bash", "-c" ]
        args:
          - apt-get update;
            apt-get install iputils-ping -y;
            apt-get install net-tools;
            apt-get install curl -y;
            sleep infinity;
```

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ubuntu-20-04
  labels:
    app: ubuntu-20-04
  namespace: preprod-app1
spec:
  replicas: 2
  selector:
    matchLabels:
      app: ubuntu-20-04
  template:
    metadata:
      labels:
        app: ubuntu-20-04
    spec:
      containers:
      - name: ubuntu-20-04
        image: ubuntu:20.04
        imagePullPolicy: IfNotPresent
        command: [ "/bin/bash", "-c" ]
        args:
          - apt-get update;
            apt-get install tcpdump -y;
            apt-get install telnet -y;
            apt-get install iputils-ping -y;
            apt-get install nmap -y;
            apt-get install net-tools;
            apt-get install netdiscover -y;
            apt-get install mysql-client -y;
            apt-get install curl -y;
            apt-get install dsniff -y;
            sleep infinity;
---
apiVersion: v1
kind: Namespace
metadata:
  name: preprod-app2
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: default-conf
  namespace: preprod-app2
data:
  default.conf: |
    server {
    listen 80 default_server;

    location / {
      proxy_pass http://app-service;
      proxy_http_version 1.1;
    }

    error_page   500 502 503 504  /50x.html;
    location = /50x.html {
        root   /usr/share/nginx/html;
    }

    }
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  namespace: preprod-app2
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
        service: web
        kind: dev
        type: internal
    spec:
      containers:
      - name: nginx
        image: nginx
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 80
        volumeMounts:
        - mountPath: /etc/nginx/conf.d # mount nginx-conf volumn to /etc/nginx
          readOnly: true
          name: default-conf
        - mountPath: /var/log/nginx
          name: log
      volumes:
      - name: default-conf
        configMap:
          name: default-conf # place ConfigMap `nginx-conf` on /etc/nginx
          items:
            - key: default.conf
              path: default.conf
      - name: log
        emptyDir: {}
---
apiVersion: v1
kind: Service
metadata:
  name: nginx
  namespace: preprod-app2
spec:
  type: NodePort
  ports:
  - port: 80
    targetPort: 80
  selector:
    app: nginx
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: appserver
  labels:
    app: app
  namespace: preprod-app2
spec:
  replicas: 1
  selector:
    matchLabels:
      app: app
  template:
    metadata:
      labels:
        app: app
        kind: dev
        type: internal
    spec:
      containers:
      - name: php-apache
        image: derstich/miserver:005
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: app-service
  labels:
    app: app
  namespace: preprod-app2
spec:
  ports:
  - port: 80
    protocol: TCP
  selector:
    app: app
---
```

```
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: mysql
  namespace: preprod-app2
spec:
  selector:
    matchLabels:
      app: mysql8
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: mysql8
        service: db
        kind: dev
        type: internal
    spec:
      containers:
      - image: mysql:5.6
        name: mysql
        imagePullPolicy: IfNotPresent
        env:
        - name: MYSQL_ROOT_PASSWORD
          value: .sweetpwd.
        - name: MYSQL_DATABASE
          value: my_db
        - name: MYSQL_USER
          value: db_user
        - name: MYSQL_PASSWORD
          value: .mypwd
        args: ["--default-authentication-plugin=mysql_native_password"]
        ports:
        - containerPort: 3306
          name: mysql8
---
apiVersion: v1
kind: Service
metadata:
  name: mysql8-service
  labels:
    app: mysql8
  namespace: preprod-app2
spec:
  type: ClusterIP
  ports:
  - port: 3306
    protocol: TCP
  selector:
    app: mysql8
```

```
kubectl apply -f .
```

Schritt 3: Daten auslesen

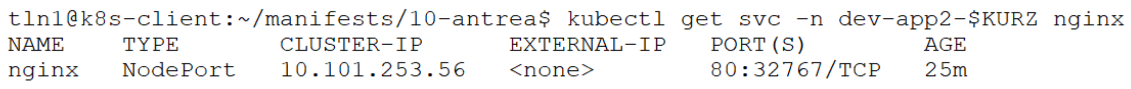
```

## dev-app1
kubectl -n dev-app1 get pods -o=custom-
columns=NAMESPACE:.metadata.namespace,NAME:.metadata.name,STATUS:.status.phase,IP:.status.podIP,NODE:.spec.nodeName

## dev-app2
kubectl -n dev-app2 get pods -o=custom-
columns=NAMESPACE:.metadata.namespace,NAME:.metadata.name,STATUS:.status.phase,IP:.status.podIP,NODE:.spec.nodeName

## preprod-app1
kubectl -n preprod-app1 get pods -o=custom-
columns=NAMESPACE:.metadata.namespace,NAME:.metadata.name,STATUS:.status.phase,IP:.status.podIP,NODE:.spec.nodeName

## preprod-app2
kubectl -n preprod-app2 get pods -o=custom-
columns=NAMESPACE:.metadata.namespace,NAME:.metadata.name,STATUS:.status.phase,IP:.status.podIP,NODE:.spec.nodeName
```

```
## BITTE die Infos zwischen speichern oder Screenshot machen
```

Schritt 4: Zugriffe auf dev-app2/prepod-app2 klären

```
## nodeip rausbekommen
kubectl get nodes -o wide
```

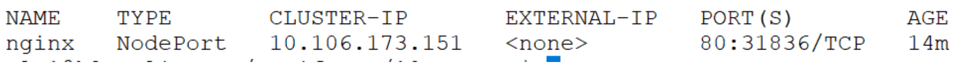
```
kubectl get svc -n dev-app2 nginx
```



```
tln1@k8s-client:~/manifests/10-antrea$ kubectl get svc -n dev-app2-$KURZ nginx
NAME    TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
nginx   NodePort   10.101.253.56   <none>        80:32767/TCP   25m
```

```
curl -i http://10.135.0.5:32767
## oder im Browser mit Public - IP
```

```
kubectl get svc -n preprod-app2 nginx
```



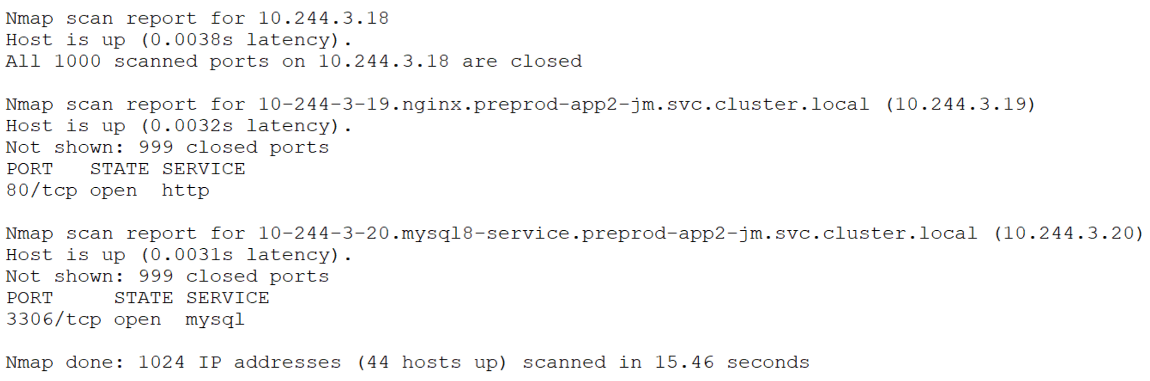
```
NAME    TYPE       CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
nginx   NodePort   10.106.173.151   <none>        80:31836/TCP   14m
```

```
curl -i http://10.135.0.5:31836
```

Schritt 5: Zugriff ohne antrea policy testen

```
kubectl exec -it -n dev-app1 deployment/ubuntu-20-04 -- /bin/bash
```

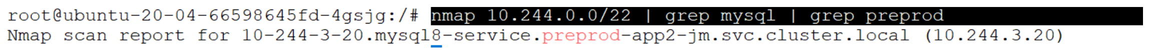
```
## scannen des netzes
## !!! Achtung Netz kann anders sein !!!
nmap 10.244.0.0/22
```



```
Nmap scan report for 10.244.3.18
Host is up (0.0038s latency).
All 1000 scanned ports on 10.244.3.18 are closed

Nmap scan report for 10-244-3-19.nginx.preprod-app2-jm.svc.cluster.local (10.244.3.19)
Host is up (0.0032s latency).
Not shown: 999 closed ports
PORT   STATE SERVICE
80/tcp open  http

Nmap scan report for 10-244-3-20.mysql8-service.preprod-app2-jm.svc.cluster.local (10.244.3.20)
Host is up (0.0031s latency).
Not shown: 999 closed ports
PORT     STATE SERVICE
3306/tcp open  mysql

Nmap done: 1024 IP addresses (44 hosts up) scanned in 15.46 seconds
```

- Namen werden aufgelöst (rückwärtig)
- alle ports sind einsehbar
- Verbindung funktioniert nach überall

```
## mysql preprod herausfinden
## !!! Achtung Netz ändern
nmap 10.244.0.0/22 | grep mysql | grep preprod
```

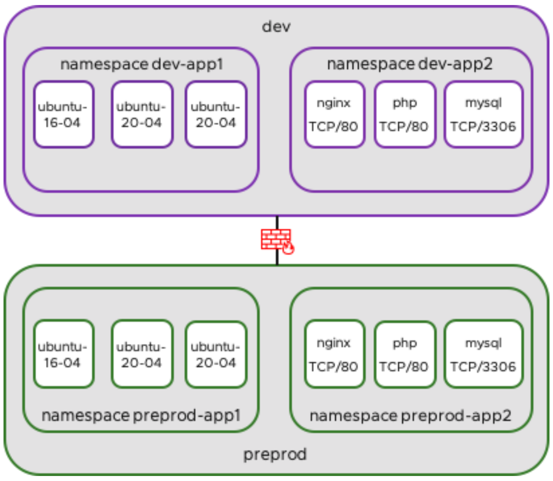


```
root@ubuntu-20-04-66598645fd-4gsjg:/# nmap 10.244.0.0/22 | grep mysql | grep preprod
Nmap scan report for 10-244-3-20.mysql8-service.preprod-app2-jm.svc.cluster.local (10.244.3.20)
```

```
## Oh, wir haben das Passwort herausgefunden (Social Engineering ;o))
.sweetpwd.
```

```
mysql -h 10-244-3-20.mysql8-service.preprod-app2.svc.cluster.local -p
```

Schritt 6: Isolieren von dev und preprod



```
## Namspaces labeln
kubectl label ns dev-app1 env=dev ns=dev-app1
kubectl label ns dev-app2 env=dev ns=dev-app2
kubectl label ns preprod-app1 env=preprod ns=preprod-app1
kubectl label ns preprod-app2 env=preprod ns=preprod-app2

kubectl describe ns dev-app1
```

```
## now create the policy
## nano 10-deny-dev-to-preprod.yaml
apiVersion: crd.antrea.io/v1beta1
kind: ClusterNetworkPolicy
metadata:
  name: deny-dev-to-preprod
spec:
    priority: 100
    tier: SecurityOps
    appliedTo:
      - namespaceSelector:
          matchLabels:
            env: preprod
    ingress:
      - action: Drop
        from:
          - namespaceSelector:
              matchLabels:
                env: dev
```

```
## Test ob ping von preprod nach dev funktioniert
## Hier ein POD-IP raussuchen
kubectl -n dev-app1 get pods -o wide
kubectl -n preprod-app1 exec deployments/ubuntu-20-04 -- ping 10.244.3.15

## Test ob ping von dev nach preprod funktioniert - der sollte nicht funktionieren
## Hier eine POD-IP rausschen
kubectl -n preprod-app1 get pods -o wide
kubectl -n dev-app1 exec deployments/ubuntu-20-04 -- ping 10.244.2.25
```

```
## ClusterNetworkPolicy anwenden
kubectl apply -f .
```

```
## Jetzt nochmal die Pings testen von oben
## ---> Ping ist immer noch möglich --> da keine Firewall - Regel
kubectl -n preprod-app1 exec deployments/ubuntu-20-04 -- ping 10.244.3.15

## in die andere Richtung geht es aber nicht !!
kubectl -n dev-app1 exec deployments/ubuntu-20-04 -- ping 10.244.2.25
```

```
## ok jetzt in die andere richtung
## nano 15-deny-preprod-to-dev.yaml
apiVersion: crd.antrea.io/v1beta1
kind: ClusterNetworkPolicy
metadata:
  name: deny-preprod-to-dev
spec:
    priority: 101
    tier: SecurityOps
    appliedTo:
      - namespaceSelector:
          matchLabels:
            env: dev
    ingress:
      - action: Drop
        from:
          - namespaceSelector:
              matchLabels:
                env: preprod
```

```
kubectl apply -f .
kubectl get clusternetworkpolicies
```

```
## Only output
NAME                     TIER          PRIORITY   DESIRED NODES   CURRENT NODES   AGE
deny-dev-to-preprod-jm   SecurityOps   100        2               2               16m
deny-preprod-to-dev      SecurityOps   101        2               2               3m15s
```

```
## und jetzt geht pingen in die andere Richtung auch nicht mehr
kubectl -n preprod-app1 exec deployments/ubuntu-20-04 -- ping 10.244.3.15
```

Schritt 7: Isolate Pods (allow only traffic within the namespaces)

- Aktuell ist das ping vom preprod-app1 zum preprod-app2 namespace noch möglich
- Das wollen wir einschränken
- Ausserdem von dev-app1 zu dev-app2

```
## So sehen unsere Namespace - Labels aus
kubectl describe namespace dev-app1
```

```
## Ausgabe, z.B.
Name:         dev-app1-jm
Labels:       env=dev-jm
              ns=dev-app1-jm
```

```
## nano 20-allow-ns-dev-app1-dev-app1.yaml
## Traffic innerhalb des Namespaces erlaubt

apiVersion: crd.antrea.io/v1beta1
kind: ClusterNetworkPolicy
metadata:
  name: 20-allow-ns-dev-app1-dev-app1
spec:
    priority: 100
    tier: application
    appliedTo:
      - namespaceSelector:
          matchLabels:
            ns: dev-app1
    ingress:
      - action: Allow
        from:
          - namespaceSelector:
              matchLabels:
                ns: dev-app1
```

```
kubectl apply -f .
```

```
## nano 25-drop-any-ns-dev-app1.yaml
## allen anderen Traffic zum namespace app2 hin verbieten aus anderen namespaces
apiVersion: crd.antrea.io/v1beta1
kind: ClusterNetworkPolicy
metadata:
  name: 25-drop-any-ns-dev-app1
spec:
    priority: 110
    tier: application
    appliedTo:
      - namespaceSelector:
          matchLabels:
            ns: dev-app1
    ingress:
      - action: Drop
        from:
          - namespaceSelector: {}
```

```
kubectl apply -f .
```

```
## nano 30-allow-ns-preprod-app1-preprod-app1.yaml
## Same for preprod-app1
## Allow all traffic within namespace
apiVersion: crd.antrea.io/v1beta1
kind: ClusterNetworkPolicy
metadata:
  name: 30-allow-ns-preprod-app1-preprod-app1
spec:
    priority: 120
    tier: application
    appliedTo:
      - namespaceSelector:
          matchLabels:
            ns: preprod-app1
    ingress:
      - action: Allow
        from:
          - namespaceSelector:
              matchLabels:
                ns: preprod-app1
```

```
kubectl apply -f .
```

```
## disallow all traffic from other namespaces to prepr
## nano 35-drop-any-ns-preprod-app1.yaml
apiVersion: crd.antrea.io/v1beta1
kind: ClusterNetworkPolicy
metadata:
  name: 35-drop-any-ns-preprod-app1
spec:
    priority: 130
    tier: application
    appliedTo:
      - namespaceSelector:
          matchLabels:
            ns: preprod-app1
    ingress:
      - action: Drop
        from:
          - namespaceSelector: {}
```

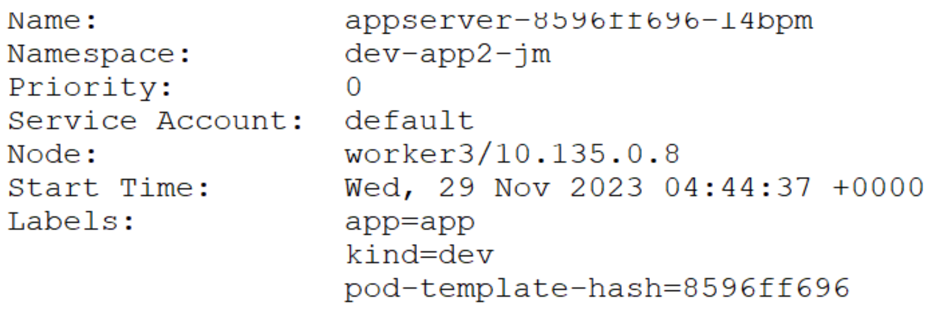
```
kubectl apply -f .
```

```
## TESTEN
## Pod ausfinding machen, und ip vom 2. Pod finden in preprod-app1
kubectl -n preprod-app1 get pods -o wide
kubectl -n preprod-app1 exec -it ubuntu-16-04-b7d656f5b-f55rs -- ping 192.168.1.12
## ping aus anderem Namespace sollte nicht gehen
kubectl -n preprod-app2 get pods -o wide
kubectl -n preprod-app2 exec -it appserver-98bc7fd55-bjv94 -- ping 192.168.1.12
```

Schritt 8: Isolate traffic within app2 - namespaces (3-Tier-app)

```
## For dev-app2 we want
web->app (80)
app->db (3306)
drop everything else
```

```
kubectl -n dev-app2 describe pods | head -n 20
kubectl -n preprod-app2 describe pods | head -n 20
```



```
Name:             appserver-8596ff696-l4bpm
Namespace:        dev-app2-jm
Priority:         0
Service Account:  default
Node:             worker3/10.135.0.8
Start Time:       Wed, 29 Nov 2023 04:44:37 +0000
Labels:           app=app
                  kind=dev
                  pod-template-hash=8596ff696
```

- we are using the label app=xxx

```
## nano 40-allow-web-app.yaml
apiVersion: crd.antrea.io/v1beta1
kind: ClusterNetworkPolicy
metadata:
  name: 40-allow-web-app
spec:
    priority: 10
    tier: application
    appliedTo:
      - podSelector:
          matchLabels:
            app: app
    ingress:
      - action: Allow
        from:
          - podSelector:
              matchLabels:
                app: nginx
        ports:
          - protocol: TCP
            port: 80
```

```
kubectl apply -f  .
```

```
## nano 45-allow-app-db.yaml
apiVersion: crd.antrea.io/v1beta1
kind: ClusterNetworkPolicy
metadata:
  name: 45-allow-app-db
spec:
    priority: 20
    tier: application
    appliedTo:
      - podSelector:
          matchLabels:
            app: mysql8
    ingress:
      - action: Allow
        from:
          - podSelector:
              matchLabels:
                app: app
        ports:
          - protocol: TCP
            port: 3306
```

```
kubectl apply -f .
```

```
## nano 50-deny-any-to-app2.yaml
## Deny everything else
apiVersion: crd.antrea.io/v1beta1
kind: ClusterNetworkPolicy
metadata:
  name: 50-deny-any-to-app2
spec:
    priority: 30
    tier: application
    appliedTo:
      - namespaceSelector:
          matchLabels:
                  ns: dev-app2
      - namespaceSelector:
          matchLabels:
                  ns: preprod-app2
    ingress:
      - action: Drop
        from:
          - namespaceSelector: {}
```

```
kubectl apply -f .
```

```
## TESTEN -> das sollte gehen // VOM web->app
kubectl -n dev-app2 get pods -l app=app -o wide
kubectl -n dev-app2 exec -it nginx-655cc89789-cjfmh -- curl -i http://192.168.1.9

## TESTEN -> geht nicht // VOM app->web
kubectl -n dev-app2 get pods -l app=nginx -o wide
kubectl -n dev-app2 exec -it appserver-8596ff696-jd9k4 -- wget -O - 192.168.2.8
```

Schritt 9: Usage of the Emergency Tier - e.g. Attack

- We have problems with Ubuntu 16.04. an we want to isolate it.

```
kubectl get tiers
```

```
## nano 80-emergency.yaml
apiVersion: crd.antrea.io/v1beta1
kind: ClusterNetworkPolicy
metadata:
  name: 80-emergency
spec:
    priority: 50
    tier: emergency
    appliedTo:
      - podSelector:
          matchLabels:
                  app: ubuntu-16-04
    ingress:
      - action: Drop
        from:
          - namespaceSelector: {}
```

```
kubectl apply -f .
```

- Because Emergency has the highest priority, the policy in application (allow any in ns-app1) has no Impact anymore.

```
## TESTEN
## GET IP
kubectl -n dev-app1 get pods -l app=ubuntu-16-04 -o wide
## Use that IP for testing, e.g. 192.168.1.8
kubectl -n dev-app1 exec -it ubuntu-20-04-66598645fd-dfx7f -- ping 192.168.1.8
```

Reference:

- https://www.vrealize.it/2020/09/28/securing-you-k8s-network-with-antrea-clusternetworkpolicy/

Antrea - Enabling logging

Steps

```
## Activate Logging in Policy Ingress or Egress

## nano 80-emergency.yaml
apiVersion: crd.antrea.io/v1beta1
kind: ClusterNetworkPolicy
metadata:
  name: 80-emergency-ubuntu16
spec:
    priority: 50
    tier: emergency
    appliedTo:
      - podSelector:
          matchLabels:
                  app: ubuntu-16-04
    ingress:
      - action: Drop
```

```
        enableLogging: true
        from:
          - namespaceSelector: {}
```

```
kubectl apply -f .
```

```
## On which node is it running ?
kubectl -n dev-app1 -l app=ubuntu-16 -o wide
## Ausgabe: worker1
```

```
## Connect to worker1 per ssh
tail /var/log/antrea/networkpolicy/np.log
```

Kubernetes calico (CNI-Plugin)

Find corresponding networks

Walkthrough

```
## Step 1: create pod
kubectl run nginx-master --image=nginx
## Find out on which node it runs
kubectl get pods -o wide
## create a debug container
kubectl debug -it nginx-master --image=busybox
```

```
## now within debug pod found out interface
 ip a | grep @
3: eth0@if22: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
```

```
## Log in to worker node  where pod runs and check interfaces
kubectl debug -it node/worker1 --image=busybox
```

```
## on worker node
## show matched line starting with 22 and then another 4 lines
ip a | grep -A 5 ^22
## e.g.
##
ip a | grep -A 5 ^22
22: cali42c2aab93f3@if3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether ee:ee:ee:ee:ee:ee brd ff:ff:ff:ff:ff:ff link-netns cni-5adf994b-3a7e-c344-5d82-ef1f7a293d88
    inet6 fe80::ecee:eeff:feee:eeee/64 scope link
       valid_lft forever preferred_lft forever
```

```
## Now you are able to determine the firewall rules
## you will find fw and tw rules (fw - from workload and tw - to workload)
iptables -L -v | grep  cali42c2aab93f3
```

```
## ... That is what you see as an example
Chain cali-tw-cali42c2aab93f3 (1 references)
 pkts bytes target     prot opt in     out     source               destination
   10  1384 ACCEPT     all  --  any    any     anywhere             anywhere             /* cali:WKA8EzdUNM0rVty1 */ ctstate
RELATED,ESTABLISHED
    0     0 DROP       all  --  any    any     anywhere             anywhere             /* cali:wr_OqGXKIN_LWnX0 */ ctstate
INVALID
    0     0 MARK       all  --  any    any     anywhere             anywhere             /* cali:kOUMqNj8np60A3Bi */ MARK and
0xfffeffff
```

Calico Logging Firewall Rules

General

- NetworkPolicy of Kubernetes does not provide possibility to track

Solutions

- Use NetworkPolicy from calico (to apply it with kubectl - the calico api server needs to be installed) / or use calicoctl
- Enable Tracing
- Use: https://kubernetes.io/blog/2019/04/19/introducing-kube-iptables-tailer/

Solution 1: NetworkPolicy calico

- https://github.com/projectcalico/calico/issues/4344

Logs

```
## Normally you should see it with (on the right kubernetes node)
cat /var/log/syslog | grep calico-packet

## This is how a syslog entry looks like
Here is a example (default) Log:
Apr  3 10:12:30 aks-workerpool1-13987120-vmss000000 kernel: [10821.860593] calico-packet: IN=calic440f455693 OUT=eth0
MAC=ee:ee:ee:ee:ee:ee:f2:f8:09:3d:97:03:08:00 SRC=10.244.2.7 DST=8.8.8.8 LEN=84 TOS=0x00 PREC=0x00 TTL=63 ID=33536 DF PROTO=ICMP
TYPE=8 CODE=0 ID=32113 SEQ=43
```

Walkthrough

```
cd
mkdir -p manifests
cd manifests
mkdir pol2
cd pol2
vi 01-pod.yaml
```

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

```
vi 02-pol.yaml
```

```
apiVersion: projectcalico.org/v3
kind: NetworkPolicy
metadata:
  name: log
spec:
  selector: app == 'web'
  types:
  - Ingress
  - Egress
  ingress:
  - action: Log
  egress:
  - action: Log
  - action: Deny
```

```
kubectl apply -f .
## find the node, where it runs on
kubectl get pods -o wide
```

```
## login to that node with ssh (kubernetes node)
## e.g. ssh user@node
## switch to root: sudo su -
tail -f /var/log/syslog | grep calico-packet
## or
journalctl -f | grep calico-packet
```

```
## now open a debug pod
kubectl debug -it static-web --image=busybox
## in pod ping - this will not work, because we cannot retrieve dns
ping www.google.de
```

```
## watch output from other node in the meanwhile
```

Reference

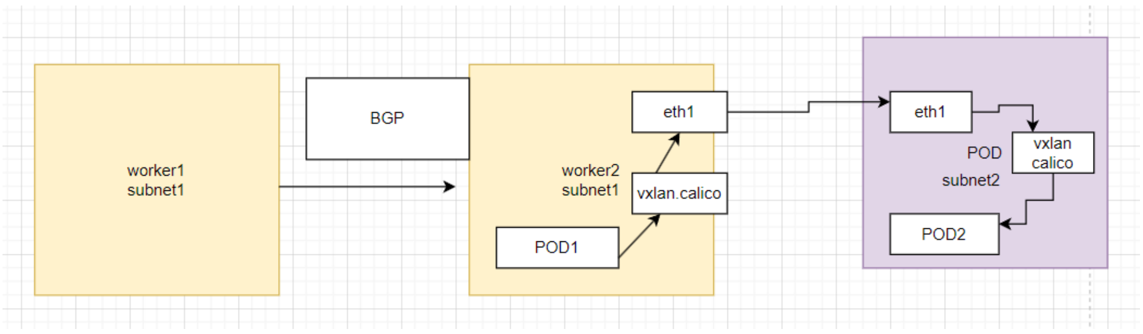
- Eventually set a prefix for logging:
- https://docs.tigera.io/calico-cloud/visibility/iptables

Calico Default Routing Mode BGP & vxlancrossnet

What does it do ?

- BGP is used, when other node is on same subnet
- vxlan is used, when worker node to reach is in other subnet

Grafics

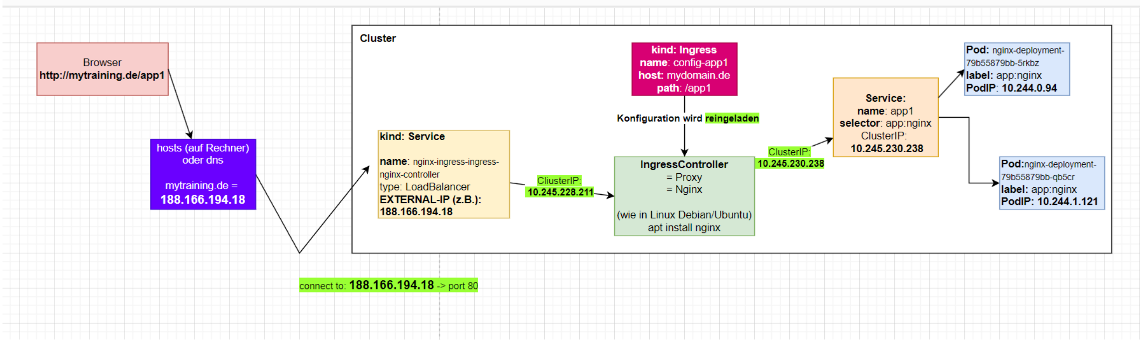


How to find out, if this node is used

```
kubectl -n calico-system get ippool -o yaml | grep vxlan
```

Kubernetes - Ingress

Vom Browser über den Ingress bis zum Pod - Schaubild



Kubernetes - Wartung / Debugging

Netzwerkverbindung zu pod testen

Situation

```
Managed Cluster und ich kann nicht auf einzelne Nodes per ssh zugreifen
```

Behelf: Eigenen Pod starten mit busybox

```
## laengere Version
kubectl run podtest --rm -ti --image busybox -- /bin/sh
```

```
## kuerzere Version
kubectl run podtest --rm -ti --image busybox
```

Example test connection

```
## wget befehl zum Kopieren
wget -O - http://10.244.0.99
```

```
## -O -> Output (grosses O (buchstabe))
kubectl run podtest --rm -ti --image busybox -- /bin/sh
/ # wget -O - http://10.244.0.99
/ # exit
```

Arbeiten mit tcpdump in pods / ingress controller

Prerequisites: Project abi is up and running

Debug traffic to pod

```
## IP des pod apple-app rausfiltern
kubectl get pods -o wide

kubectl debug apple-app -it --image nicolaka/netshoot
## Show processes of other container first
kubectl debug apple-app -it --image nicolaka/netshoot --target=apple-app
```

in pod

```
ps aux
tcpdump -n port 5678
```

in 2. Session (kubectl)

```
kubectl run -it --rm podtester --image=busybox
```

```
wget -O - <ip-des-apple-pods>:5678
```

Debug traffic to ingress controller

mit netshoot connecten

Variante 1: Direkt

```
kubectl -n ingress debug nginx-ingress-ingress-nginx-controller-7bc7c7776d-jpj5h -it --image nicolaka/netshoot
```

```
## in der shell
tcpdump -n port 80
## write to file in pcap format
Older versions of tcpdump truncate packets to 68 or 96 bytes. If this is the case, use -s to capture full-sized packets:
tcpdump -i <interface> -s 65535 -w <file>
```

Variante 2: Im Hintergrund laufen lassen und connecten

```
kubectl -n ingress debug nginx-ingress-ingress-nginx-controller-7bc7c7776d-jpj5h --image nicolaka/netshoot -- sleep infinite
kubectl -n ingress exec -it nginx-ingress-ingress-nginx-controller-7bc7c7776d-jpj5h -c debugger-gwvsr -- zsh
```

```
## in der shell
tcpdump -n port 80
## write to file in pcap format
Older versions of tcpdump truncate packets to 68 or 96 bytes. If this is the case, use -s to capture full-sized packets:
tcpdump -i <interface> -s 65535 -w <file>
```

Testen

```
## Im browser url aufrufen
## z.B.
http://jochen.lab1.t3isp.de
```

Kubernetes Cheatsheet/Spickzettel

Das Tool kubectl (Devs/Ops) - Spickzettel

Allgemein

```
## Zeige Information über das Cluster
kubectl cluster-info

## Welche api-resources gibt es ?
kubectl api-resources

## Hilfe zu object und eigenschaften bekommen
kubectl explain pod
kubectl explain pod.metadata
kubectl explain pod.metadata.name
```

Arbeiten mit manifesten

```
kubectl apply -f nginx-replicaset.yml
## Wie ist aktuell die hinterlegte config im system
kubectl get -o yaml -f nginx-replicaset.yml

## Änderung in nginx-replicaset.yml z.B. replicas: 4
## dry-run - was wird geändert
kubectl diff -f nginx-replicaset.yml

## anwenden
kubectl apply -f nginx-replicaset.yml

## Alle Objekte aus manifest löschen
kubectl delete -f nginx-replicaset.yml
```

Ausgabeformate

```
## Ausgabe kann in verschiedenen Formaten erfolgen
kubectl get pods -o wide # weitere informationen
## im json format
kubectl get pods -o json

## gilt natürluch auch für andere kommandos
kubectl get deploy -o json
kubectl get deploy -o yaml

## get a specific value from the complete json - tree
kubectl get node k8s-nue-jo-ff1p1 -o=jsonpath='{.metadata.labels}'
```

Zu den Pods

```
## Start einen pod // BESSER: direkt manifest verwenden
## kubectl run podname image=imagename
kubectl run nginx image=nginx

## Pods anzeigen
kubectl get pods
kubectl get pod
## Format weitere Information
kubectl get pod -o wide
## Zeige labels der Pods
kubectl get pods --show-labels

## Zeige pods mit einem bestimmten label
kubectl get pods -l app=nginx

## Status eines Pods anzeigen
kubectl describe pod nginx

## Pod löschen
kubectl delete pod nginx

## Kommando in pod ausführen
kubectl exec -it nginx -- bash
## direkt in den 1. Pod des Deployments wechseln
kubectl exec -it deployment/name-des-deployments -- bash
```

Logs ausgeben

```
kubectl logs podname
## -n = namespace
## | less -> seitenweise Ausgabe
kubectl -n ingress logs nginx-ingress-ingress-nginx-controller-7bc7c7776d-jpj5h | less
```

Arbeiten mit namespaces

```
## Welche namespaces auf dem System
kubectl get ns
kubectl get namespaces
## Standardmäßig wird immer der default namespace verwendet
## wenn man kommandos aufruft
kubectl get deployments

## Möchte ich z.B. deployment vom kube-system (installation) aufrufen,
## kann ich den namespace angeben
```

```
kubectl get deployments --namespace=kube-system
kubectl get deployments -n kube-system

## wir wollen unseren default namespace ändern
kubectl config set-context --current --namespace <dein-namespace>
```

Referenz

- https://kubernetes.io/de/docs/reference/kubectl/cheatsheet/

Kubernetes Praxis (zum Verständnis von Netzwerk)

kubectl example with run

Example (that does work)

```
## Show the pods that are running
kubectl get pods

## Synopsis (most simplistic example
## kubectl run NAME --image=IMAGE_EG_FROM_DOCKER
## example
kubectl run nginx --image=nginx:1.23

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

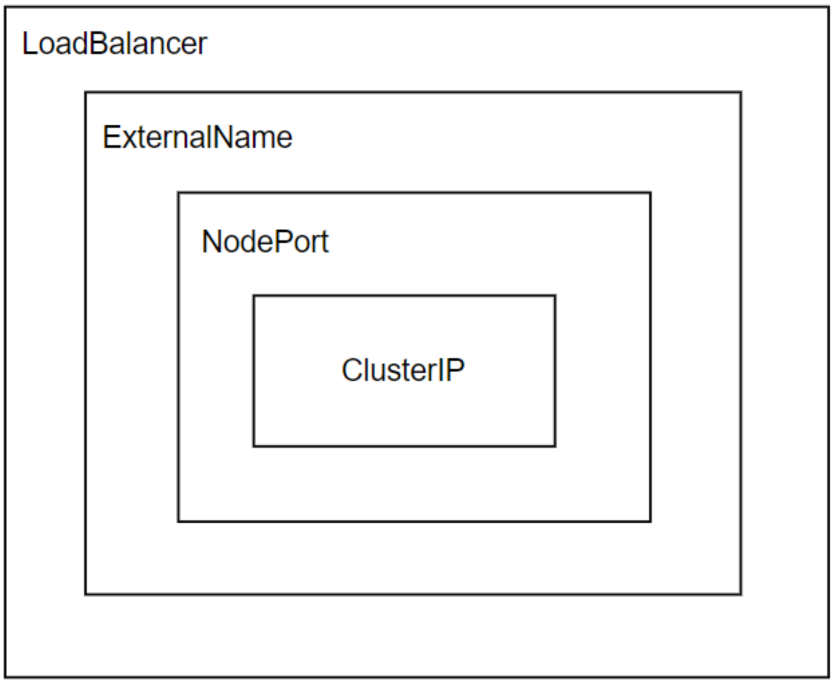
Example (that does not work)

```
kubectl run foo2 --image=foo2
## ImageErrPull - Image konnte nicht geladen werden
kubectl get pods
## Weitere status - info
kubectl describe pods foo2
```

Ref:

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

Service Typen / Ebenen - Schaubild



kubectl/manifest/service

Schritt 1: Deployment

```
cd
mkdir -p manifests
cd manifests
mkdir 04-service
cd 04-service
##vi 01-deploy.yml
```

```
## 01-deploy.yml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-nginx
spec:
  selector:
    matchLabels:
      app: my-nginx
  replicas: 3
  template:
    metadata:
      labels:
        app: my-nginx
    spec:
      containers:
      - name: my-nginx
        image: nginx
        ports:
        - containerPort: 80
```

```
kubectl apply -f .
```

Schritt 2:

```
## 02-svc.yml
apiVersion: v1
kind: Service
metadata:
  name: my-nginx
  labels:
    svc: nginx
spec:
  ports:
  - port: 80
    protocol: TCP
  selector:
    app: my-nginx
```

```
kubectl apply -f .
```

Ref.

- https://kubernetes.io/docs/concepts/services-networking/connect-applications-service/

Ingress Controller auf Digitalocean (doks) mit helm installieren

Basics

- Das Verfahren funktioniert auch so auf anderen Plattformen, wenn helm verwendet wird und noch kein IngressController vorhanden
- Ist kein IngressController vorhanden, werden die Ingress-Objekte zwar angelegt, es funktioniert aber nicht.

Prerequisites

- kubectl muss eingerichtet sein

Walkthrough (Setup Ingress Controller)

```
helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
helm repo update
helm show values ingress-nginx/ingress-nginx

## It will be setup with type loadbalancer - so waiting to retrieve an ip from the external loadbalancer
## This will take a little.
helm install nginx-ingress ingress-nginx/ingress-nginx --namespace ingress --create-namespace --set
controller.publishService.enabled=true
```

```
## See when the external ip comes available
kubectl -n ingress get all
kubectl --namespace ingress get services -o wide -w nginx-ingress-ingress-nginx-controller

## Output
NAME                                        TYPE           CLUSTER-IP     EXTERNAL-IP      PORT(S)                      AGE
SELECTOR
nginx-ingress-ingress-nginx-controller   LoadBalancer   10.245.78.34   157.245.20.222   80:31588/TCP,443:30704/TCP   4m39s
app.kubernetes.io/component=controller,app.kubernetes.io/instance=nginx-ingress,app.kubernetes.io/name=ingress-nginx

## Now setup wildcard - domain for training purpose
## inwx.com
*.lab1.t3isp.de A 157.245.20.222
```

Beispiel mit Hostnamen

Walkthrough

Step 1: pods and services

```
cd
mkdir -p manifests
cd manifests
mkdir abi
cd abi
```

```
## 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-<dein-name>"
---

kind: Service
apiVersion: v1
metadata:
  name: apple-service
spec:
  selector:
    app: apple
  ports:
    - protocol: TCP
      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-<dein-name>"

---

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
```

Step 2: Ingress

```
## Ingress
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: example-ingress
  annotations:
    ingress.kubernetes.io/rewrite-target: /
    # with the ingress controller from helm, you need to set an annotation
    # otherwice it does not know, which controller to use
    # old version... use ingressClassName instead
    # kubernetes.io/ingress.class: nginx
spec:
  ingressClassName: nginx
  rules:
  - host: "<euername>.lab<nr>.t3isp.de"
    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

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: example-ingress
  annotations:
    ingress.kubernetes.io/rewrite-target: /
    # with the ingress controller from helm, you need to set an annotation
    # old version useClassName instead
    # otherwice it does not know, which controller to use
    # kubernetes.io/ingress.class: nginx
spec:
  ingressClassName: nginx
  rules:
  - host: "app12.lab.t3isp.de"
    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
```

Kubernetes Load Balancer

Kubernetes Load Balancer

Attention

- On digitalocean, we will probably run into problems, that it is not working properly

General

- Supports bgp and arp
- Divided into controller, speaker

Installation Ways

- helm
- manifests

Step 1: install metallb

```
## Just to show some basics
## Page from metallb says that digitalocean is not really supported well
## So we will not install the speaker .

helm repo add metallb https://metallb.github.io/metallb
```

```
## Eventually disabling speaker
## vi values.yml
```

```
helm install metallb metallb/metallb --namespace=metallb-system --create-namespace
```

Step 2: addresspool und Propagation-type (config)

```
cd
mkdir -p manifests
cd manifests
mkdir lb
cd lb
nano 01-addresspool.yml
```

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: first-pool
  namespace: metallb-system
spec:
  addresses:
  # we will use our external ip here
  - 134.209.231.154-134.209.231.154
  # both notations are possible
  - 157.230.113.124/32
```

```
kubectl apply -f .
```

```
nano 02-advertisement.yml
```

```
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: example
  namespace: metallb-system
```

```
kubectl apply -f .
```

Schritt 4: Test do i get an external ip

```
nano 03-deploy.yml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-nginx
spec:
  selector:
    matchLabels:
      run: web-nginx
  replicas: 3
  template:
    metadata:
      labels:
        run: web-nginx
    spec:
      containers:
      - name: cont-nginx
        image: nginx
        ports:
        - containerPort: 80
```

```
nano 04-service.yml
```

```
## 02-svc.yml
apiVersion: v1
kind: Service
metadata:
  name: svc-nginx
  labels:
    svc: nginx
spec:
  type: LoadBalancer
  ports:
  - port: 80
    protocol: TCP
  selector:
    run: web-nginx
```

```
kubectl apply -f .
kubectl get pods
kubectl get svc
```

```
kubectl delete -f 03-deploy.yml 04-service.yml

### Kubernetes Load Balancer new version for IpAdresses - object


### Installatiion

 * Refs: https://metallb.universe.tf/installation/

### Step 1: Installation:
```

kubectl apply -f https://raw.githubusercontent.com/metallb/metallb/v0.13.9/config/manifests/metallb-native.yaml

```

### Step 2: Konfiguration
```

mkdir -p manifests cd manifests mkdir metallb vi 01-pool.yaml

```
```

apiVersion: metallb.io/v1beta1 kind: IPAddressPool metadata: name: first-pool namespace: metallb-system spec: addresses:

- 192.168.1.240-192.168.1.250

```
```

vi 02-l2.yaml

```
```

now we need to propagate

apiVersion: metallb.io/v1beta1 kind: L2Advertisement metadata: name: example namespace: metallb-system

```
### References

  * https://microk8s.io/docs/addon-metallb
  * https://metallb.universe.tf/
  * Calico Issues: https://metallb.universe.tf/configuration/calico/


## Kubernetes - Netzwerk (CNI's) / Mesh

### Calico/Cilium - nginx example NetworkPolicy


### Schritt 1: Deployment und Service erstellen
```

KURZ=jm kubectl create ns policy-demo-$KURZ

```
```

cd mkdir -p manifests cd manifests mkdir -p np cd np

```
```

nano 01-deployment.yml

apiVersion: apps/v1 kind: Deployment metadata: name: nginx-deployment spec: selector: matchLabels: app: nginx replicas: 1 template: metadata: labels: app: nginx spec: containers: - name: nginx image: nginx:1.23 ports: - containerPort: 80

```
```

kubectl -n policy-demo-$KURZ apply -f .

```
```

nano 02-service.yaml

apiVersion: v1 kind: Service metadata: name: nginx spec: type: ClusterIP # Default Wert ports:

- port: 80 protocol: TCP selector: app: nginx

```
```

kubectl -n policy-demo-$KURZ apply -f .

```
### Schritt 2: Zugriff testen ohne Regeln
```

lassen einen 2. pod laufen mit dem auf den nginx zugreifen

kubectl run --namespace=policy-demo-$KURZ access --rm -ti --image busybox

```
```

innerhalb der shell

wget -q nginx -O -

```
```

Optional: Pod anzeigen in 2. ssh-session zu jump-host

kubectl -n policy-demo-$KURZ get pods --show-labels

```
### Schritt 3: Policy festlegen, dass kein Zugriff erlaubt ist.
```

nano 03-default-deny.yaml

Schritt 2: Policy festlegen, dass kein Ingress-Traffic erlaubt

in diesem namespace: policy-demo-$KURZ

kind: NetworkPolicy apiVersion: networking.k8s.io/v1 metadata: name: default-deny spec: podSelector: matchLabels: {}

```
```

kubectl -n policy-demo-$KURZ apply -f .

```
### Schritt 3.5: Verbindung mit deny all Regeln testen
```

kubectl run --namespace=policy-demo-$KURZ access --rm -ti --image busybox

```
```

innerhalb der shell

wget -q nginx -O -

```
### Schritt 4: Zugriff erlauben von pods mit dem Label run=access (alle mit run gestarteten pods mit namen access haben dieses
label per default)
```

nano 04-access-nginx.yaml

apiVersion: networking.k8s.io/v1 kind: NetworkPolicy metadata: name: access-nginx spec: podSelector: matchLabels: app: nginx ingress: - from: - podSelector: matchLabels: run: access

```
```

kubectl -n policy-demo-$KURZ apply -f .

```
### Schritt 5: Testen (zugriff sollte funktionieren)
```

lassen einen 2. pod laufen mit dem auf den nginx zugreifen

pod hat durch run -> access automatisch das label run:access zugewiesen

kubectl run --namespace=policy-demo-$KURZ access --rm -ti --image busybox

```
```

innerhalb der shell

wget -q nginx -O -

```
### Schritt 6: Pod mit label run=no-access - da sollte es nicht gehen
```

kubectl run --namespace=policy-demo-$KURZ no-access --rm -ti --image busybox

```
```

in der shell

wget -q nginx -O -

```
### Schritt 7: Aufräumen
```

kubectl delete ns policy-demo-$KURZ

```
### Ref:

  * https://projectcalico.docs.tigera.io/security/tutorials/kubernetes-policy-basic

### Beispiele Ingress Egress NetworkPolicy


### Links
```

```
  * https://github.com/ahmetb/kubernetes-network-policy-recipes
  * https://k8s-examples.container-solutions.com/examples/NetworkPolicy/NetworkPolicy.html

### Example with http (Cilium !!)
```

apiVersion: "cilium.io/v2" kind: CiliumNetworkPolicy description: "L7 policy to restrict access to specific HTTP call" metadata: name: "rule1" spec: endpointSelector: matchLabels: type: l7-test ingress:

- fromEndpoints:
  - matchLabels: org: client-pod toPorts:
  - ports:
    - port: "8080" protocol: TCP rules: http:
      - method: "GET" path: "/discount"

```

### Downside egress

  * No valid api for anything other than IP's and/or Ports
  * If you want more, you have to use CNI-Plugin specific, e.g.

#### Example egress with ip's
```

Allow traffic of all pods having the label role:app

egress only to a specific ip and port

apiVersion: networking.k8s.io/v1 kind: NetworkPolicy metadata: name: test-network-policy namespace: default spec: podSelector: matchLabels: role: app policyTypes:

- Egress egress:
- to:
  - ipBlock: cidr: 10.10.0.0/16 ports:
  - protocol: TCP port: 5432

```

### Example Advanced Egress (cni-plugin specific)

#### Cilium
```

apiVersion: v1 kind: Pod metadata: name: nginx-static-web labels: webserver: nginx spec: containers:

- name: web image: nginx

```

```

apiVersion: cilium.io/v2 kind: CiliumNetworkPolicy metadata: name: "fqdn-pprof"

namespace: msp

spec: endpointSelector: matchLabels: webserver: nginx egress:

- toFQDNs:
  - matchPattern: '*.google.com'
- toPorts:
  - ports:
    - port: "53" protocol: ANY rules: dns:
      - matchPattern: '*'

```

```

kubectl apply -f .

```

#### Calico

  * Only Calico enterprise
    * Calico Enterprise extends Calico’s policy model so that domain names (FQDN / DNS) can be used to allow access from a pod or set of pods (via label selector) to external resources outside of your cluster.
    * https://projectcalico.docs.tigera.io/security/calico-enterprise/egress-access-controls

#### Using isitio as mesh (e.g. with cilium/calico )

##### Installation of sidecar in calico

  * https://projectcalico.docs.tigera.io/getting-started/kubernetes/hardway/istio-integration
```

```
##### Example
```

apiVersion: networking.k8s.io/v1 kind: NetworkPolicy metadata: name: test-network-policy namespace: default spec: podSelector: matchLabels: role: app policyTypes:

- Egress egress:
- to:
  - ipBlock: cidr: 10.10.0.0/16 ports:
  - protocol: TCP port: 5432

```
### Mesh / istio


### Schaubild

![istio Schaubild](https://istio.io/latest/docs/examples/virtual-machines/vm-bookinfo.svg)

### Istio
```

Visualization

with kiali (included in istio)

https://istio.io/latest/docs/tasks/observability/kiali/kiali-graph.png

Example

https://istio.io/latest/docs/examples/bookinfo/

The sidecars are injected in all pods within the namespace by labeling the namespace like so: kubectl label namespace default istio-injection=enabled

Gateway (like Ingress in vanilla Kubernetes)

kubectl label namespace default istio-injection=enabled

```
### istio tls

 * https://istio.io/latest/docs/ops/configuration/traffic-management/tls-configuration/


### istio - the next generation without sidecar

  * https://istio.io/latest/blog/2022/introducing-ambient-mesh/

### DNS - Resolution - Services
```

kubectl run podtest --rm -ti --image busybox -- /bin/sh If you don't see a command prompt, try pressing enter. / # wget -O - http://apple-service.jochen Connecting to apple-service.jochen (10.245.39.214:80) writing to stdout apple-tln1

-
  ```
                 100%
  |*****************************************************************************************************************|    11
  0:00:00 ETA
  ```

written to stdout / # wget -O - http://apple-service.jochen.svc.cluster.local Connecting to apple-service.jochen.svc.cluster.local (10.245.39.214:80) writing to stdout apple-tln1

-
  ```
                 100%
  |*****************************************************************************************************************|    11
  0:00:00 ETA
  ```

written to stdout / # wget -O - http://apple-service Connecting to apple-service (10.245.39.214:80) writing to stdout apple-tln1

-
  ```
                 100%
  |*****************************************************************************************************************|    11
  0:00:00 ETA
  ```

written to stdout

```
## Calico NetworkPolicy

### Protecting Services
```

```
### Example
```

apiVersion: projectcalico.org/v3 kind: GlobalNetworkPolicy metadata: name: allow-cluster-ips spec: selector: k8s-role == 'node' types:

- Ingress applyOnForward: true preDNAT: true ingress:

Allow 50.60.0.0/16 to access Cluster IP A

- action: Allow source: nets:
  - 50.60.0.0/16 destination: nets:
  - 10.20.30.40/32 Cluster IP A

Allow 70.80.90.0/24 to access Cluster IP B

- action: Allow source: nets:
  - 70.80.90.0/24 destination: nets:
  - 10.20.30.41/32 Cluster IP B

```
### Referenz

  * https://docs.tigera.io/calico/latest/network-policy/services/services-cluster-ips

### Exercise calico Network Policy


### Step 1: Set global policy
```

apiVersion: crd.projectcalico.org/v1 kind: GlobalNetworkPolicy metadata: name: default-deny spec: namespaceSelector: kubernetes.io/metadata.name != "kube-system" types:

- Ingress
- Egress egress:

allow all namespaces to communicate to DNS pods

- action: Allow protocol: UDP destination: selector: 'k8s-app == "kube-dns"' ports:
  - 53
- action: Allow protocol: TCP destination: selector: 'k8s-app == "kube-dns"' ports:
  - 53

```

```

kubectl apply -f .

```
### Step 2: nginx ausrollen aus manifests/04-service und testen
```

cd mkdir -p manifests cd manifests mkdir 04-service cd 04-service

```

```

nano deploy.yml

```

```

apiVersion: apps/v1 kind: Deployment metadata: name: web-nginx spec: selector: matchLabels: web: my-nginx replicas: 2 template: metadata: labels: web: my-nginx spec: containers: - name: cont-nginx image: nginx ports: - containerPort: 80

```

```

nano service.yml

```

```

apiVersion: v1 kind: Service metadata: name: svc-nginx labels: run: svc-my-nginx spec: type: ClusterIP ports:

- port: 80 protocol: TCP selector: web: my-nginx

```

```

kubectl apply -f .

```

```

kubectl run -it --rm access --image=busybox

In der Bbusybox

wget -O - http://svc-nginx

```
### Step 3: Traffic erlauben egress von busybox
```

cd cd manifests mkdir cnp cd cnp

vi 02-egress-allow-busybox.yml

apiVersion: crd.projectcalico.org/v1 kind: NetworkPolicy metadata: name: allow-busybox-egress spec: selector: run == 'access' types:

- Egress egress:
- action: Allow

kubectl apply -f .

kubectl run -it --rm access --image=busybox

sollte gehen

wget -O - http://www.google.de

sollte nicht funktionieren

wget -O - http://my-nginx

```
### Step 4: Traffic erlauben für nginx
```

03-allow-ingress-my-nginx.yml

apiVersion: crd.projectcalico.org/v1 kind: NetworkPolicy metadata: name: allow-nginx-ingress spec: selector: run == 'my-nginx' types:

- Ingress ingress:
- action: Allow source: selector: run == 'access'

kubectl apply -f .

kubectl run -it --rm access --image=busybox

In der Bbusybox

wget -O - http://my-nginx

```
## Kubernetes calico (CNI-Plugin)

### Welcher Routing-Mode wird im aktuellen Cluster verwendet
```

kubectl -n calico-system describe ds calico-node | grep -A 35 calico-node

or specific

kubectl -n calico-system describe ds calico-node | egrep -i -e vxlan -e cluster_type

Environment: DATASTORE_TYPE: kubernetes WAIT_FOR_DATASTORE: true CLUSTER_TYPE: k8s,operator,bgp CALICO_DISABLE_FILE_LOGGING: false FELIX_DEFAULTENDPOINTTOHOSTACTION: ACCEPT FELIX_HEALTHENABLED: true FELIX_HEALTHPORT: 9099 NODENAME: (v1:spec.nodeName) NAMESPACE: (v1:metadata.namespace) FELIX_TYPHAK8SNAMESPACE: calico-system FELIX_TYPHAK8SSERVICENAME: calico-typha FELIX_TYPHACAFILE: /etc/pki/tls/certs/tigera-ca-bundle.crt FELIX_TYPHACERTFILE: /node-certs/tls.crt FELIX_TYPHAKEYFILE: /node-certs/tls.key FIPS_MODE_ENABLED: false FELIX_TYPHACN: typha-server CALICO_MANAGE_CNI: true CALICO_IPV4POOL_CIDR: 192.168.0.0/16 CALICO_IPV4POOL_VXLAN: CrossSubnet CALICO_IPV4POOL_BLOCK_SIZE: 26 CALICO_IPV4POOL_NODE_SELECTOR: all() CALICO_IPV4POOL_DISABLE_BGP_EXPORT: false CALICO_NETWORKING_BACKEND: bird IP: autodetect IP_AUTODETECTION_METHOD: first-found IP6: none FELIX_IPV6SUPPORT: false KUBERNETES_SERVICE_HOST: 10.96.0.1 KUBERNETES_SERVICE_PORT: 443 Mounts:

```

### Wird eBPF verwendet ?


  * Hint: By default this should not be activated

### Version microk8s
```

kubectl -n kube-system logs calico-node-78s8q | grep -i bpfenabled

```

### Version installed on your own in cluster, e.g. kubeadm
```

Is in different namespace in this case

kubectl -n calico-system logs calico-node-78s8q | grep -i bpfenabled

```

### Install calicoctl in pod


### General

#### It was like that ....

  * calicoctl used to do validation locally in calicoctl for your manifests in the projectcalico/v3 api-version
  * This version was not available in kube-api-server

#### Now ....

  * Validation takes place on server side.
  * For this to work the kube-api-server needs to be configured with calico
  * Now the preferred method is to use kubectl (without dependencies to calicoctl) but not for.....
    * calicoctl node
    * calicoctl ipam
    * calicoctl convert
    * calicoctl version

#### Reference:

  * https://docs.tigera.io/calico/latest/operations/calicoctl/configure/kdd





### calicoctl Installation walkthrough (running in pod)

#### Find out version
```

welche version von calico setzen wir aktuell auf dem server ein

kubectl -n kube-system get ds calico-node -o=jsonpath='{.spec.template.spec.containers[0].image}'

docker.io/calico/node:v3.23.5

```

#### Pod erstellen für calicoctl auf Basis von
```

cd mkdir -p manifests cd manifests mkdir calicoctl cd calicoctl vi calicoctl.yaml

```

```

https://raw.githubusercontent.com/projectcalico/calico/v3.25.1/manifests/calicoctl.yaml

Calico Version master

https://projectcalico.docs.tigera.io/releases#master

This manifest includes the following component versions:

calico/ctl:v3.25.1

apiVersion: v1 kind: ServiceAccount metadata: name: calicoctl namespace: kube-system

---

apiVersion: v1 kind: Pod metadata: name: calicoctl namespace: kube-system spec: nodeSelector: kubernetes.io/os: linux hostNetwork: true serviceAccountName: calicoctl containers:

- name: calicoctl image: calico/ctl:v3.23.5 command:
  - /calicoctl args:
  - version
  - --poll=1m env:
  - name: DATASTORE_TYPE value: kubernetes

---

kind: ClusterRole apiVersion: rbac.authorization.k8s.io/v1 metadata: name: calicoctl rules:

- apiGroups: [""] resources:
  - namespaces
  - nodes verbs:
  - get
  - list
  - update
- apiGroups: [""] resources:
  - nodes/status verbs:
  - update
- apiGroups: [""] resources:
  - pods
  - serviceaccounts verbs:
  - get
  - list
- apiGroups: [""] resources:
  - pods/status verbs:
  - update
- apiGroups: ["crd.projectcalico.org"] resources:
  - bgppeers
  - bgpconfigurations
  - clusterinformations
  - felixconfigurations
  - globalnetworkpolicies
  - globalnetworksets
  - ippools
  - ipreservations
  - kubecontrollersconfigurations
  - networkpolicies
  - networksets
  - hostendpoints
  - ipamblocks
  - blockaffinities
  - ipamhandles
  - ipamconfigs verbs:
  - create
  - get
  - list
  - update
  - delete
- apiGroups: ["networking.k8s.io"] resources:
  - networkpolicies verbs:
  - get
  - list

---

apiVersion: rbac.authorization.k8s.io/v1 kind: ClusterRoleBinding metadata: name: calicoctl roleRef: apiGroup: rbac.authorization.k8s.io kind: ClusterRole name: calicoctl subjects:

- kind: ServiceAccount name: calicoctl namespace: kube-system

```
### calicoctl verwenden
```

this will always work, no matter what version

kubectl -n kube-system exec calicoctl -- /calicoctl version

this will only work without flags, if we have the same version

on both sides

```
### Wann calicoctl (Stand 2024/01 calico 3.27)


### Für Informationen über die Nodes (z.B. BGP) - direkt auf Node ausführen

 * calicoctl get nodes

### Um Zusatzinformationen abzufragen, die nur in calicoctl zur Verfügung stehen
```

namespace in command needs to be written at then end

calicoctl get wep -n namespace-der-application

get version

calicoctl version

show cidr / the ippool

calicoctl ipam show calicoctl ipam check

```
### Calico - only on one of nodes (e.g. controlplane - need to login with ssh)
```

.kube/config does not need to be configured

calicoctl node status calicoctl ipam status

```
### Install calico-api-server to use kubectl instead of calicoctl


### prepare kube-api-server for to be use for calico calls.

  * Possible from calico 3.20+ (GA)
  * https://docs.tigera.io/calico/latest/operations/install-apiserver


#### Step 1: Apply manifests for api server
```

cd mkdir -p manifests cd manifests

calico api server

mkdir cas cd cas vi cas.yaml

```
```

taken from https://raw.githubusercontent.com/projectcalico/calico/v3.25.1/manifests/apiserver.yaml

but adjusted images version to corresponding installation

kubectl -n kube-system get ds calico-node -o=jsonpath='{.spec.template.spec.containers[0].image}'

This is a tech-preview manifest which installs the Calico API server. Note that this manifest is liable to change

or be removed in future releases without further warning.

Namespace and namespace-scoped resources.

apiVersion: v1 kind: Namespace metadata: labels: name: calico-apiserver name: calico-apiserver spec:

Policy to ensure the API server isn't cut off. Can be modified, but ensure

that the main API server is always able to reach the Calico API server.

kind: NetworkPolicy apiVersion: networking.k8s.io/v1 metadata: name: allow-apiserver namespace: calico-apiserver spec: podSelector: matchLabels: apiserver: "true" ingress:

- ports:
  - protocol: TCP port: 5443

---

apiVersion: v1 kind: Service metadata: name: calico-api namespace: calico-apiserver spec: ports:

- name: apiserver port: 443 protocol: TCP targetPort: 5443 selector: apiserver: "true" type: ClusterIP

---

apiVersion: apps/v1 kind: Deployment metadata: labels: apiserver: "true" k8s-app: calico-apiserver name: calico-apiserver namespace: calico-apiserver spec: replicas: 1 selector: matchLabels: apiserver: "true" strategy: type: Recreate template: metadata: labels: apiserver: "true" k8s-app: calico-apiserver name: calico-apiserver namespace: calico-apiserver spec: containers: - args: - --secure-port=5443 # - -v=5 # not working in v3.23.5 not available as flag there env: - name: DATASTORE_TYPE value: kubernetes image: calico/apiserver:v3.23.5 livenessProbe: httpGet: path: /version port: 5443 scheme: HTTPS initialDelaySeconds: 90 periodSeconds: 10 name: calico-apiserver readinessProbe: exec: command: - /code/filecheck failureThreshold: 5 initialDelaySeconds: 5 periodSeconds: 10 securityContext: privileged: false runAsUser: 0 volumeMounts: - mountPath: /code/apiserver.local.config/certificates name: calico-apiserver-certs dnsPolicy: ClusterFirst nodeSelector: kubernetes.io/os: linux restartPolicy: Always serviceAccount: calico-apiserver serviceAccountName: calico-apiserver tolerations: - effect: NoSchedule key: node-role.kubernetes.io/master - effect: NoSchedule key: node-role.kubernetes.io/control-plane volumes: - name: calico-apiserver-certs secret: secretName: calico-apiserver-certs

---

apiVersion: v1 kind: ServiceAccount metadata: name: calico-apiserver namespace: calico-apiserver

---

Cluster-scoped resources below here.

apiVersion: apiregistration.k8s.io/v1 kind: APIService metadata: name: v3.projectcalico.org spec: group: projectcalico.org groupPriorityMinimum: 1500 service: name: calico-api namespace: calico-apiserver port: 443 version: v3 versionPriority: 200

---

apiVersion: rbac.authorization.k8s.io/v1 kind: ClusterRole metadata: name: calico-crds rules:

- apiGroups:
  - extensions
  - networking.k8s.io
  - "" resources:
  - networkpolicies
  - nodes
  - namespaces
  - pods
  - serviceaccounts verbs:
  - get
  - list
  - watch
- apiGroups:
  - crd.projectcalico.org resources:
  - globalnetworkpolicies
  - networkpolicies
  - clusterinformations
  - hostendpoints
  - globalnetworksets
  - networksets
  - bgpconfigurations
  - bgppeers
  - felixconfigurations
  - kubecontrollersconfigurations
  - ippools
  - ipreservations
  - ipamblocks
  - blockaffinities
  - caliconodestatuses
  - ipamconfigs verbs:
  - get
  - list
  - watch
  - create
  - update
  - delete
- apiGroups:
  - policy resourceNames:
  - calico-apiserver resources:
  - podsecuritypolicies verbs:
  - use

---

apiVersion: rbac.authorization.k8s.io/v1 kind: ClusterRole metadata: name: calico-extension-apiserver-auth-access rules:

- apiGroups:
  - "" resourceNames:
  - extension-apiserver-authentication resources:
  - configmaps verbs:

- list
    - watch
    - get
- apiGroups:
    - rbac.authorization.k8s.io resources:
    - clusterroles
    - clusterrolebindings
    - roles
    - rolebindings verbs:
    - get
    - list
    - watch

---

apiVersion: rbac.authorization.k8s.io/v1 kind: ClusterRole metadata: name: calico-webhook-reader rules:

- apiGroups:
    - admissionregistration.k8s.io resources:
    - mutatingwebhookconfigurations
    - validatingwebhookconfigurations verbs:
    - get
    - list
    - watch

---

apiVersion: rbac.authorization.k8s.io/v1 kind: ClusterRoleBinding metadata: name: calico-apiserver-access-crds roleRef: apiGroup: rbac.authorization.k8s.io kind: ClusterRole name: calico-crds subjects:

- kind: ServiceAccount name: calico-apiserver namespace: calico-apiserver

---

apiVersion: rbac.authorization.k8s.io/v1 kind: ClusterRoleBinding metadata: name: calico-apiserver-delegate-auth roleRef: apiGroup: rbac.authorization.k8s.io kind: ClusterRole name: system:auth-delegator subjects:

- kind: ServiceAccount name: calico-apiserver namespace: calico-apiserver

---

apiVersion: rbac.authorization.k8s.io/v1 kind: ClusterRoleBinding metadata: name: calico-apiserver-webhook-reader roleRef: apiGroup: rbac.authorization.k8s.io kind: ClusterRole name: calico-webhook-reader subjects:

- kind: ServiceAccount name: calico-apiserver namespace: calico-apiserver

---

apiVersion: rbac.authorization.k8s.io/v1 kind: ClusterRoleBinding metadata: name: calico-extension-apiserver-auth-access roleRef: apiGroup: rbac.authorization.k8s.io kind: ClusterRole name: calico-extension-apiserver-auth-access subjects:

- kind: ServiceAccount name: calico-apiserver namespace: calico-apiserver

```

#### Step 2: create certificates
```

openssl req -x509 -nodes -newkey rsa:4096 -keyout apiserver.key -out apiserver.crt -days 365 -subj "/" -addext "subjectAltName = DNS:calico-api.calico-apiserver.svc" kubectl create secret -n calico-apiserver generic calico-apiserver-certs --from-file=apiserver.key --from-file=apiserver.crt

configure server with ca-bundle

kubectl patch apiservice v3.projectcalico.org -p
"{"spec": {"caBundle": "$(kubectl get secret -n calico-apiserver calico-apiserver-certs -o go-template='{{ index .data "apiserver.crt" }}')"}}"

```

### Step 3: check if it is working
```

pod should run

kubectl -n calico-apiserver get pods

if not delete it

e.g.

kubectl -n calico-apiserver delete po calico-apiserver-6f64fdcc5c-kz45t

it will get recreated because of deployment

```

```

kubectl api-resources | grep '\sprojectcalico.org'

only available in v3

kubectl get clusterinfo

```
### Calico Default Routing Mode BGP & vxlancrossnet


### What does it do ?

  * BGP is used, when other node is on same subnet
  * vxlan is used, when worker node to reach is in other subnet

### Grafics

![image](https://github.com/jmetzger/training-kubernetes-networking/assets/1933318/a2766737-e1e5-4ee0-8e03-9216a0379d97)

### How to find out, if this node is used
```

kubectl -n calico-system get ippool -o yaml | grep vxlan

```
### Internals - Pod to Pod - Communication on Worker3 (node))


![image](https://github.com/jmetzger/training-kubernetes-networking/assets/1933318/ba9d497d-36ed-467f-9965-faad76a201cd)

### Internals - Inter-Pod - Communication (worker 3 -> worker 1


![image](https://github.com/jmetzger/training-kubernetes-networking/assets/1933318/305e0dac-5d13-4f6c-88b0-3b06b88eba7c)

## Kubernetes Praxis API-Objekte

### Bauen einer Applikation mit Resource Objekten


![Bauen einer Webanwendung](images/WebApp.drawio.png)

### Pod manifest


### Walkthrough
```

cd mkdir -p manifests cd manifests mkdir -p web cd web

```
```

vi nginx-static.yml

apiVersion: v1 kind: Pod metadata: name: nginx-static-web labels: webserver: nginx spec: containers:

- name: web image: nginx

```
```

kubectl apply -f nginx-static.yml kubectl describe pod nginx-static-web

show config

kubectl get pod/nginx-static-web -o yaml kubectl get pod/nginx-static-web -o wide

```
### Replicasets
```

cd mkdir -p manifests cd manifests mkdir 02-rs cd 02-rs

vi rs.yml

```
```

apiVersion: apps/v1 kind: ReplicaSet metadata: name: nginx-replica-set spec: replicas: 2 selector: matchLabels: tier: frontend template: metadata: name: template-nginx-replica-set labels: tier: frontend spec: containers: - name: nginx image: nginx:1.21 ports: - containerPort: 80

```
```

kubectl apply -f rs.yml

```
### kubectl/manifest/deployments
```

cd mkdir -p manifests cd manifests mkdir 03-deploy cd 03-deploy nano deploy.yml

```
```

vi deploy.yml

apiVersion: apps/v1 kind: Deployment metadata: name: nginx-deployment spec: selector: matchLabels: app: nginx replicas: 8 # tells deployment to run 8 pods matching the template template: metadata: labels: app: nginx spec: containers: - name: nginx image: nginx:1.21 ports: - containerPort: 80

```
```

kubectl apply -f deploy.yml

```
### Services - Aufbau

![Services Aufbau](/images/kubernetes-services.drawio.svg)

### Hintergrund Ingress



### Ref. / Dokumentation

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

### Documentation for default ingress nginx

  * https://kubernetes.github.io/ingress-nginx/user-guide/nginx-configuration/configmap/

### Beispiel Ingress

### Prerequisits
```

Ingress Controller muss aktiviert sein

microk8s enable ingress

```
### Walkthrough

#### Schritt 1:
```

cd mkdir -p manifests cd manifests mkdir abi cd abi

```
```

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: - protocol: TCP 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

```
#### Schritt 2:
```

Ingress

apiVersion: extensions/v1beta1 kind: Ingress metadata: name: example-ingress annotations: ingress.kubernetes.io/rewrite-target: / spec: ingressClassName: nginx 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: ingressClassName: nginx 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

```
### Achtung: Ingress mit Helm - annotations

### Permanente Weiterleitung mit Ingress


### Example
```

redirect.yml

apiVersion: v1 kind: Namespace metadata: name: my-namespace

---

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" creationTimestamp: null name: destination-home namespace: my-namespace spec: rules:

- host: web.training.local http: paths:
  - backend: service: name: http-svc port: number: 80 path: /source pathType: ImplementationSpecific

```

```

Achtung: host-eintrag auf Rechner machen, von dem aus man zugreift

/etc/hosts 45.23.12.12 web.training.local

```
```

curl -I http://web.training.local/source HTTP/1.1 308 Permanent Redirect

```
### Umbauen zu google ;o)
```

This annotation allows to return a permanent redirect instead of sending data to the upstream. For example nginx.ingress.kubernetes.io/permanent-redirect: https://www.google.com would redirect everything to Google.

```
### Refs:

  * https://github.com/kubernetes/ingress-nginx/blob/main/docs/user-guide/nginx-configuration/annotations.md#permanent-redirect
  *

### ConfigMap Example


### Schritt 1: configmap vorbereiten
```

cd mkdir -p manifests cd manifests mkdir configmaptests cd configmaptests nano 01-configmap.yml

```
```

01-configmap.yml

kind: ConfigMap apiVersion: v1 metadata: name: example-configmap data:

als Wertepaare

database: mongodb database_uri: mongodb://localhost:27017

```
```

kubectl apply -f 01-configmap.yml kubectl get cm kubectl get cm -o yaml

```
### Schrit 2: Beispiel als Datei
```

nano 02-pod.yml

```
```

kind: Pod apiVersion: v1 metadata: name: pod-mit-configmap

spec:

Add the ConfigMap as a volume to the Pod

volumes: # `name` here must match the name # specified in the volume mount - name: example-configmap-volume # Populate the volume with config map data configMap: # `name` here must match the name # specified in the ConfigMap's YAML name: example-configmap

containers: - name: container-configmap image: nginx:latest # Mount the volume that contains the configuration data # into your container filesystem volumeMounts: # `name` here must match the name # from the volumes section of this pod - name: example-configmap-volume mountPath: /etc/config

```
```

kubectl apply -f 02-pod.yml

```
```

##Jetzt schauen wir uns den Container/Pod mal an kubectl exec pod-mit-configmap -- ls -la /etc/config kubectl exec -it pod-mit-configmap -- bash

ls -la /etc/config

```
### Schritt 3: Beispiel. ConfigMap als env-variablen
```

nano 03-pod-mit-env.yml

```
```

03-pod-mit-env.yml

kind: Pod apiVersion: v1 metadata: name: pod-env-var spec: containers: - name: env-var-configmap image: nginx:latest envFrom: - configMapRef: name: example-configmap

kubectl apply -f 03-pod-mit-env.yml

und wir schauen uns das an

##Jetzt schauen wir uns den Container/Pod mal an kubectl exec pod-env-var -- env kubectl exec -it pod-env-var -- bash

env

```


### Reference:

 * https://matthewpalmer.net/kubernetes-app-developer/articles/ultimate-configmap-guide-kubernetes.html

### Configmap MariaDB - Example


### Schritt 1: configmap
```

cd mkdir -p manifests cd manifests mkdir cftest cd cftest nano 01-configmap.yml

01-configmap.yml

kind: ConfigMap apiVersion: v1 metadata: name: mariadb-configmap data:

als Wertepaare

MARIADB_ROOT_PASSWORD: 11abc432

kubectl apply -f . kubectl get cm kubectl get cm mariadb-configmap -o yaml

```


### Schritt 2: Deployment
```

nano 02-deploy.yml

##deploy.yml apiVersion: apps/v1 kind: Deployment metadata: name: mariadb-deployment spec: selector: matchLabels: app: mariadb replicas: 1 template: metadata: labels: app: mariadb spec: containers: - name: mariadb-cont image: mariadb:latest envFrom: - configMapRef: name: mariadb-configmap

kubectl apply -f .

```

### Important Sidenode

  * If configmap changes, deployment does not know
  * So kubectl apply -f deploy.yml will not have any effect
  * to fix, use stakater/reloader: https://github.com/stakater/Reloader


### Configmap MariaDB my.cnf


### configmap zu fuss
```

vi mariadb-config2.yml

kind: ConfigMap apiVersion: v1 metadata: name: example-configmap data:

als Wertepaare

database: mongodb my.cnf: | [mysqld] slow_query_log = 1 innodb_buffer_pool_size = 1G

```
```

kubectl apply -f .

```
```

##deploy.yml apiVersion: apps/v1 kind: Deployment metadata: name: mariadb-deployment spec: selector: matchLabels: app: mariadb replicas: 1 template: metadata: labels: app: mariadb spec: containers: - name: mariadb-cont image: mariadb:latest envFrom: - configMapRef: name: mariadb-configmap

```
    volumeMounts:
      - name: example-configmap-volume
        mountPath: /etc/my

  volumes:
  - name: example-configmap-volume
    configMap:
      name: example-configmap
```

```
```

kubectl apply -f .

```

## Kubernetes multus (Meta-CNI - Plugin)

### Multus Überblick


### Problem, Warum multus ?

  * Aktuell kann seitens kubernetes nur ein Interface verwaltet werden, weil der CNI-Call nur 1x ausgeführt wird. (eigentlich 2x
wenn man localhost mit einbezieht)

### Prerequisites

  * a CNI, that manages the network needs to be installed before hand, like Calico, Cilium

### Graphics

![Multus](https://github.com/k8snetworkplumbingwg/multus-cni/raw/master/docs/images/multus-pod-image.svg)

### General

  * Multus is a meta-plugin, which makes it possible to attach additional networks to your pod (multi - homing)

### macvlan plugin

### Example macvlan

  * https://github.com/k8snetworkplumbingwg/multus-cni/blob/master/examples/macvlan-pod.yml
```

This net-attach-def defines macvlan-conf with

+ ips capabilities to specify ip in pod annotation and

+ mac capabilities to specify mac address in pod annotation

default gateway is defined as well

apiVersion: "k8s.cni.cncf.io/v1" kind: NetworkAttachmentDefinition metadata: name: macvlan-conf spec: config: '{ "cniVersion": "0.3.1", "plugins": [{ "type": "macvlan", "capabilities": { "ips": true }, "master": "eth0", "mode": "bridge", "ipam": { "type": "static", "routes": [{ "dst": "0.0.0.0/0", "gw": "10.1.1.1" }] } }, { "capabilities": { "mac": true }, "type": "tuning" }] }'

Define a pod with macvlan-conf, defined above, with ip address and mac, and

"gateway" overrides default gateway to use macvlan-conf's one.

without "gateway" in k8s.v1.cni.cncf.io/networks, default route will be cluster

network interface, eth0, even tough macvlan-conf has default gateway config.

apiVersion: v1 kind: Pod metadata: name: samplepod annotations: k8s.v1.cni.cncf.io/networks: '[{ "name": "macvlan-conf", "ips": ["10.1.1.101/24"], "mac": "c2:b0:57:49:47:f1", "gateway": ["10.1.1.1"] }]' spec: containers:

- name: samplepod command: ["/bin/bash", "-c", "trap : TERM INT; sleep infinity & wait"] image: dougbtv/centos-network ports:
  - containerPort: 80

```

### sr-iov mit multus


### Voraussetzung: Multus:

### Konzept SR-IOV

  * Direkte Hardwareanbindung der Netzwerkkarte
  * Offload wird auf Netzwerkkarte gemacht (nicht im Kernel)
  * bessere Performance

### Generell

  * Erweiterung des PCI-Express Standarads
  * Eine Netzwerkkarte wird mehrmals angeboten und Kommunikation erfolgt direkt und nicht über den Umweg Kernel

### Vorbereitung

 * https://github.com/k8snetworkplumbingwg/sriov-network-device-plugin
 * https://github.com/k8snetworkplumbingwg/sriov-network-device-plugin/tree/db98d96cc0d6ad3fff917ba238bd1cc5cc3f7e82#config-
parameters

### Einbindung

  * https://github.com/k8snetworkplumbingwg/sriov-network-device-plugin#example-deployments
  * https://github.com/k8snetworkplumbingwg/multus-cni/blob/master/examples/sriov-pod.yml







## Kubernetes coil (egress - gateway)

### coil


### Opt-In egress-gateway (NAT-Service)
```

apiVersion: coil.cybozu.com/v2 kind: Egress metadata: namespace: internet-egress name: nat spec: replicas: 2 destinations:

- 0.0.0.0/0
- ::/0

```

  * Not all Pods become the client of Egress. To become a client, Pods need to have special annotations like this:
```

apiVersion: v1 kind: Pod metadata: namespace: default name: nat-client annotations: egress.coil.cybozu.com/internet-egress: nat spec:

```

### Reference

  * Refs: https://blog.kintone.io/entry/coilv2
  * https://github.com/cybozu-go/coil


## Kubernetes antrea (CNI-Plugin)

### Unterschiede Dokus vmware (antrea mit nsx-t) und OpenSource Antrea


  * OpenSource - Version has less features than closed version

### Antrea (OpenSource) - Version

  * https://antrea.io/docs/v1.13.2/
```

```
### vmware - spread across tanzu (AFAIK)

  * https://docs.vmware.com/en/VMware-Tanzu-Kubernetes-Grid/2.4/tkg-deploy-mc/mgmt-reqs-network-antrea-tiering.html

### Overview Kubernetes Antrea CNI-Plugin


### Overview

![Overview](https://antrea.io/docs/v1.3.0/docs/assets/arch.svg.png)

### Basics

  * Created by vmware
  * Uses Open VShift (virtuell Switches)
  * Kernel-Modul openswitch.ko takes care of traffic (performant)

### Components

#### antrea-controller (+api)

  * Watches kube-api-server for changes on
    * pod
    * namespaces
    * NetworkPolicy
  * Implementation of Controller - API-Server
  * Reachable over kube-api-server by implementation https://kubernetes.io/docs/concepts/extend-kubernetes/api-
extension/apiserver-aggregation/
  * Currently only 1 replica is supported
  * computes NetworkPolicies and distributes them to the Antrea agents

##### antrea controller api - part (how authentication works)

  * The Controller API server delegates authentication and authorization to the Kubernetes API
  * the Antrea Agent uses a Kubernetes ServiceAccount token to authenticate to the Controller,
  * the Controller API server validates the token and whether the ServiceAccount is authorized for the API request with the
Kubernetes API.

#### antrea-agent

  * Runs on every pod, deployed by Daemonset
  * has an endpoint running gRPC which the controller connects to
  * Agents connect to controller api by ClusterIP - wit a service Account
  * Authentication is done through the kubernetes api server

### antctl

  * cli for some debugging
  * controller-mode on controller (accessing from within controller pod)
  * agent-mode on agent (accessing from within agent-pod)
  * external also possible - uses kubeconfig to connect
    * Connection is done through kube-api-server

#### Important antctl commands
```

on kube-system

kubectl -n kube-system get üpods

antctl get featuregates

```
### Reference

  * https://antrea.io/docs/v1.3.0/docs/design/architecture/

### Antctl


### Install (externally as tool (not in pod)): uses .kube/config (Done by trainer)
```

as root

cd /usr/local/sbin
curl -Lo ./antctl "https://github.com/antrea-io/antrea/releases/download/v1.13.2/antctl-$(uname)-x86_64" chmod +x ./antctl

```
```

run as unprivileged user having a .kube/config in homedir

antctl version

```
### Shows feature-gates for controller and agent (using antctl client externally)

   * Shows both (for controller and for agent), when you do it externally as client-tool from outside pod
```

antctl get featuregates

```
![image](https://github.com/jmetzger/training-kubernetes-networking/assets/1933318/285069b5-5d6f-4b40-b828-24f2b8879e16)

### Use antctl from within agent
```

kubectl -n kube-system exec -it ANTREA-AGENT_POD_NAME -n kube-system -c antrea-agent -- bash

```
  * or
```

kubectl -n kube-system exec -it daemonset/antrea-agent -n kube-system -c antrea-agent -- bash

```
```

antctl help antctl log-level antctl get featuregates

```
### Antrea view bridge and config


### Finding the bridge

  *  ovs-vsctl - utility for querying and configuring ovs-vswitchd
```

How to see the bridge

kubectl -n kube-system exec -it antrea-agent-79bx2 -c antrea-agent -- ovs-vsctl show

or: always shows the first pod it finds

kubectl -n kube-system exec -it daemonset/antrea-agent -c antrea-agent -- ovs-vsctl show

```
![image](https://github.com/jmetzger/training-kubernetes-networking/assets/1933318/d369db27-630a-4b9f-9d9b-834075514737)

### Show the configuraton settings of antrea (configmap)
```

kubectl -n kube-system get cm antrea-config -o yaml

```
### Antrea NetworkPolicy Exercise - 1 Cluster in Group


### Our Goal

![image](https://github.com/jmetzger/training-kubernetes-networking/assets/1933318/14cc372e-e3df-4075-9330-d9ca50eab959)

### How the order of priorities work

![image](https://github.com/jmetzger/training-kubernetes-networking/assets/1933318/8bf05e88-a7cd-4906-9271-eee2acc014a7)

### Our Setup
```

In app1 are some Ubuntu Servers for Testing: dev-app1 / preprod-app1

1x Ubuntu Server 16.04 2x Ubuntu Server 20.04 In app2 is a simple 3 Tier-App (WEB-APP-DB): dev-app2 / preprod-app2 (3tier-app)

1x nginx TCP/80 (NodePort) 1x php TCP/80 (ClusterIP) 1x mysql TCP/3306 (ClusterIP)

```
![image](https://github.com/jmetzger/training-kubernetes-networking/assets/1933318/067f275e-152c-42b6-82fd-7bebc6921cbb)

### Step 1: Rollout the pods (dev-app1)

  * Important - you need to adjust the namespaces as follows:
    * dev-app1-<name-kurz> -> z.B. dev-app1-jjm (Deine Initialien)
```

cd mkdir -p manifests cd manifests mkdir 10-antrea cd 10-antrea

nano 01-deployment-dev-app1.yaml

apiVersion: v1 kind: Namespace metadata: name: dev-app1-

apiVersion: apps/v1 kind: Deployment metadata: name: ubuntu-16-04 labels: app: ubuntu-16-04 namespace: dev-app1- spec: replicas: 1 selector: matchLabels: app: ubuntu-16-04 template: metadata: labels: app: ubuntu-16-04 spec: containers: - name: ubuntu-16-04 image: ubuntu:16.04 imagePullPolicy: IfNotPresent command: ["/bin/bash", "-c"] args: - apt-get update; apt-get install iputils-ping -y; apt-get install net-tools; apt-get install curl -y; sleep infinity;

apiVersion: apps/v1 kind: Deployment metadata: name: ubuntu-20-04 labels: app: ubuntu-20-04 namespace: dev-app1- spec: replicas: 2 selector: matchLabels: app: ubuntu-20-04 template: metadata: labels: app: ubuntu-20-04 spec: containers: - name: ubuntu-20-04 image: ubuntu:20.04 imagePullPolicy: IfNotPresent command: ["/bin/bash", "-c"] args: - apt-get update; apt-get install tcpdump -y; apt-get install telnet -y; apt-get install iputils-ping -y; apt-get install nmap -y; apt-get install net-tools; apt-get install netdiscover -y; apt-get install mysql-client -y; apt-get install curl -y; apt-get install dsniff -y; sleep infinity;

check if we have replaced all the kurz entries

cat 01-deployment-dev-app1.yaml | grep kurz

kubectl apply -f .

kubectl -n dev-app1- get pods

z.B. kubectl -n dev-app1-jjm get pods

```
### Step 2: Rollout the pods (dev-app2)
```

nano 02-deployment-dev-app2.yaml

apiVersion: v1 kind: Namespace metadata: name: dev-app2-

apiVersion: v1 kind: ConfigMap metadata: name: default-conf namespace: dev-app2- data: default.conf: | server { listen 80 default_server;

```
location / {
  proxy_pass http://app-service;
  proxy_http_version 1.1;
}

error_page   500 502 503 504  /50x.html;
location = /50x.html {
    root   /usr/share/nginx/html;
}

}
```

---

apiVersion: apps/v1 kind: Deployment metadata: name: nginx namespace: dev-app2- spec: replicas: 1 selector: matchLabels: app: nginx template: metadata: labels: app: nginx service: web kind: dev type: internal spec: containers: - name: nginx image: nginx imagePullPolicy: IfNotPresent ports: - containerPort: 80 volumeMounts: - mountPath: /etc/nginx/conf.d # mount nginx-conf volumn to /etc/nginx readOnly: true name: default-conf - mountPath: /var/log/nginx name: log volumes: - name: default-conf configMap: name: default-conf # place ConfigMap `nginx-conf` on /etc/nginx items: - key: default.conf path: default.conf - name: log emptyDir: {}

apiVersion: v1 kind: Service metadata: name: nginx namespace: dev-app2- spec: type: NodePort ports: - port: 80 targetPort: 80 selector: app: nginx

apiVersion: apps/v1 kind: Deployment metadata: name: appserver labels: app: app namespace: dev-app2- spec: replicas: 1 selector: matchLabels: app: app template: metadata: labels: app: app kind: dev type:

internal spec: containers: - name: php-apache image: derstich/miserver:006 imagePullPolicy: IfNotPresent ports: - containerPort: 80

apiVersion: v1 kind: Service metadata: name: app-service labels: app: app namespace: dev-app2- spec: ports: - port: 80 protocol: TCP selector: app: app

apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2 kind: Deployment metadata: name: mysql namespace: dev-app2- spec: selector: matchLabels: app: mysql8 strategy: type: Recreate template: metadata: labels: app: mysql8 service: db kind: dev type: internal spec: containers: - image: mysql:5.6 name: mysql imagePullPolicy: IfNotPresent env: - name: MYSQL_ROOT_PASSWORD value: .sweetpwd. - name: MYSQL_DATABASE value: my_db - name: MYSQL_USER value: db_user - name: MYSQL_PASSWORD value: .mypwd args: ["--default-authentication-plugin=mysql_native_password"] ports: - containerPort: 3306 name: mysql8

apiVersion: v1 kind: Service metadata: name: mysql8-service labels: app: mysql8 namespace: dev-app2- spec: type: ClusterIP ports:

- port: 3306 protocol: TCP selector: app: mysql8

```
```

kubectl apply -f . kubectl -n dev-app2- get all

```
### Schritt 3: rollout preprod-app1
```

nano 03-deployment-preprod-app1.yaml

apiVersion: v1 kind: Namespace metadata: name: preprod-app1-

apiVersion: apps/v1 kind: Deployment metadata: name: ubuntu-16-04 labels: app: ubuntu-16-04 namespace: preprod-app1- spec: replicas: 1 selector: matchLabels: app: ubuntu-16-04 template: metadata: labels: app: ubuntu-16-04 spec: containers: - name: ubuntu-16-04 image: ubuntu:16.04 imagePullPolicy: IfNotPresent command: ["/bin/bash", "-c"] args: - apt-get update; apt-get install iputils-ping -y; apt-get install net-tools; apt-get install curl -y; sleep infinity;

apiVersion: apps/v1 kind: Deployment metadata: name: ubuntu-20-04 labels: app: ubuntu-20-04 namespace: preprod-app1- spec: replicas: 2 selector: matchLabels: app: ubuntu-20-04 template: metadata: labels: app: ubuntu-20-04 spec: containers: - name: ubuntu-20-04 image: ubuntu:20.04 imagePullPolicy: IfNotPresent command: ["/bin/bash", "-c"] args: - apt-get update; apt-get install tcpdump -y; apt-get install telnet -y; apt-get install iputils-ping -y; apt-get install nmap -y; apt-get install net-tools; apt-get install netdiscover -y; apt-get install mysql-client -y; apt-get install curl -y; apt-get install dsniff -y; sleep infinity;

```
```

kubectl apply -f .

```
### Schritt 4: Deploy preprod-app2
```

nano 04-deployment-preprod-app2.yaml

apiVersion: v1 kind: Namespace metadata: name: preprod-app2-

apiVersion: v1 kind: ConfigMap metadata: name: default-conf namespace: preprod-app2- data: default.conf: | server { listen 80 default_server;

```
location / {
  proxy_pass http://app-service;
  proxy_http_version 1.1;
}

error_page   500 502 503 504  /50x.html;
location = /50x.html {
    root   /usr/share/nginx/html;
}

}
```

---

apiVersion: apps/v1 kind: Deployment metadata: name: nginx namespace: preprod-app2- spec: replicas: 1 selector: matchLabels: app: nginx template: metadata: labels: app: nginx service: web kind: dev type:

internal spec: containers: - name: nginx image: nginx imagePullPolicy: IfNotPresent ports: - containerPort: 80 volumeMounts: - mountPath: /etc/nginx/conf.d # mount nginx-conf volumn to /etc/nginx readOnly: true name: default-conf - mountPath: /var/log/nginx name: log volumes: - name: default-conf configMap: name: default-conf # place ConfigMap `nginx-conf` on /etc/nginx items: - key: default.conf path: default.conf - name: log emptyDir: {}

apiVersion: v1 kind: Service metadata: name: nginx namespace: preprod-app2- spec: type: NodePort ports: - port: 80 targetPort: 80 selector: app: nginx

apiVersion: apps/v1 kind: Deployment metadata: name: appserver labels: app: app namespace: preprod-app2- spec: replicas: 1 selector: matchLabels: app: app template: metadata: labels: app: app kind: dev type: internal spec: containers: - name: php-apache image: derstich/miserver:005 imagePullPolicy: IfNotPresent ports: - containerPort: 80

apiVersion: v1 kind: Service metadata: name: app-service labels: app: app namespace: preprod-app2- spec: ports: - port: 80 protocol: TCP selector: app: app

apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2 kind: Deployment metadata: name: mysql namespace: preprod-app2- spec: selector: matchLabels: app: mysql8 strategy: type: Recreate template: metadata: labels: app: mysql8 service: db kind: dev type: internal spec: containers: - image: mysql:5.6 name: mysql imagePullPolicy: IfNotPresent env: - name: MYSQL_ROOT_PASSWORD value: .sweetpwd. - name: MYSQL_DATABASE value: my_db - name: MYSQL_USER value: db_user - name: MYSQL_PASSWORD value: .mypwd args: ["--default-authentication-plugin=mysql_native_password"] ports: - containerPort: 3306 name: mysql8

apiVersion: v1 kind: Service metadata: name: mysql8-service labels: app: mysql8 namespace: preprod-app2- spec: type: ClusterIP ports:

- port: 3306 protocol: TCP selector: app: mysql8

```
```

kubectl apply -f .

```
### Schritt 5: Daten auslesen
```

Das bitte anpassen

KURZ=jm

```
```

dev-app1

kubectl -n dev-app1-$KURZ get pods -o=custom-columns=NAMESPACE:.metadata.namespace,NAME:.metadata.name,STATUS:.status.phase,IP:.status.podIP,NODE:.spec.nodeName

dev-app2

kubectl -n dev-app2-$KURZ get pods -o=custom-columns=NAMESPACE:.metadata.namespace,NAME:.metadata.name,STATUS:.status.phase,IP:.status.podIP,NODE:.spec.nodeName

preprod-app1

kubectl -n preprod-app1-$KURZ get pods -o=custom-columns=NAMESPACE:.metadata.namespace,NAME:.metadata.name,STATUS:.status.phase,IP:.status.podIP,NODE:.spec.nodeName

preprod-app2

kubectl -n preprod-app2-$KURZ get pods -o=custom-columns=NAMESPACE:.metadata.namespace,NAME:.metadata.name,STATUS:.status.phase,IP:.status.podIP,NODE:.spec.nodeName

```
```

BITTE die Infos zwischen speichern oder Screenshot machen

```
### Schritt 6: Zugriff auf dev-app2 klären
```

Das ändern

KURZ=jm

```

```

kubectl get svc -n dev-app2-$KURZ nginx

```
![image](https://github.com/jmetzger/training-kubernetes-networking/assets/1933318/00a9d952-732a-4e12-98d5-766734e96ba7)
```

curl -i http://10.135.0.5:32767

oder im Browser mit Public - IP

```
### Schritt 7: Zugriff auf preprod-app klären
```

Das ändern

KURZ=jm

```

```

kubectl get svc -n preprod-app2-$KURZ nginx

```
![image](https://github.com/jmetzger/training-kubernetes-networking/assets/1933318/0f53b7a4-0fe2-4294-b3eb-f5ac968da47c)
```

curl -i http://10.135.0.5:31836

```
### Schritt 8: Zugriff ohne antrea policy testen
```

KURZ=jm kubectl exec -it -n dev-app1-$KURZ deployment/ubuntu-20-04 -- /bin/bash

```

```

scannen des netzes

nmap 10.244.0.0/22

```
![image](https://github.com/jmetzger/training-kubernetes-networking/assets/1933318/33ee8c86-d3d9-46c6-9b67-93b3318a2739)

 * Namen werden aufgelöst (rückwärtig)
 * alle ports sind einsehbar
 * Verbindung funktioniert nach überall
```

mysql preprod herausfinden

nmap 10.244.0.0/22 | grep mysql | grep preprod

```
![image](https://github.com/jmetzger/training-kubernetes-networking/assets/1933318/852f4781-fb45-45f6-9c8b-22474be6e092)
```

Oh, wir haben das Passwort herausgefunden (Social Engineering ;o))

.sweetpwd.

```

```

mysql -h 10-244-3-20.mysql8-service.preprod-app2-jm.svc.cluster.local -p

```
### Schritt 9: Isolate dev and preprod

![image](https://github.com/jmetzger/training-kubernetes-networking/assets/1933318/52d514ce-806b-48ed-bb90-8de5897537ef)
```

entsprechend anpassen

KURZ=jm

Namspaces labeln

kubectl label ns dev-app1-$KURZ env=dev-$KURZ ns=dev-app1-$KURZ kubectl label ns dev-app2-$KURZ env=dev-$KURZ ns=dev-app2-$KURZ kubectl label ns preprod-app1-$KURZ env=preprod-$KURZ ns=preprod-app1-$KURZ kubectl label ns preprod-app2-$KURZ env=preprod-$KURZ ns=preprod-app2-$KURZ

kubectl describe ns dev-app1-$KURZ

now create the policy

nano 10-deny-dev-to-preprod.yaml

apiVersion: crd.antrea.io/v1beta1 kind: ClusterNetworkPolicy metadata: name: deny-dev-to-preprod- spec: priority: 100 tier: SecurityOps appliedTo: - namespaceSelector: matchLabels: env: preprod- ingress: - action: Drop from: - namespaceSelector: matchLabels: env: dev-

KURZ=jm

Test ob ping von preprod nach dev funktioniert

Hier ein POD-IP raussuchen

kubectl -n dev-app1-$KURZ get pods -o wide kubectl -n preprod-app1-$KURZ exec deployments/ubuntu-20-04 -- ping 10.244.3.15

Test ob ping von dev nach preprod funktioniert - der sollte nicht funktionieren

Hier eine POD-IP rausschen

kubectl -n preprod-app1-$KURZ get pods -o wide kubectl -n dev-app1-$KURZ exec deployments/ubuntu-20-04 -- ping 10.244.2.25

ClusterNetworkPolicy anwenden

kubectl apply -f .

Jetzt nochmal die Pings testen von oben

---> Ping ist immer noch möglich --> da keine Firewall - Regel

kubectl -n preprod-app1-$KURZ exec deployments/ubuntu-20-04 -- ping 10.244.3.15

in die andere Richtung geht es aber nicht !!

kubectl -n dev-app1-$KURZ exec deployments/ubuntu-20-04 -- ping 10.244.2.25

ok jetzt in die andere richtung

nano 15-deny-preprod-to-dev.yaml

apiVersion: crd.antrea.io/v1beta1 kind: ClusterNetworkPolicy metadata: name: deny-preprod-to-dev- spec: priority: 101 tier: SecurityOps appliedTo: - namespaceSelector: matchLabels: env: dev- ingress: - action: Drop from: - namespaceSelector: matchLabels: env: preprod-

kubectl apply -f . kubectl get clusternetworkpolicies

Only output

NAME TIER PRIORITY DESIRED NODES CURRENT NODES AGE deny-dev-to-preprod-jm SecurityOps 100 2 2 16m deny-preprod-to-dev SecurityOps 101 2 2 3m15s

und jetzt geht pingen in die andere Richtung auch nicht mehr

kubectl -n preprod-app1-$KURZ exec deployments/ubuntu-20-04 -- ping 10.244.3.15

```
### Schritt 11: Isolate Pods (only within the namespaces)

  * Aktuell ist das ping vom preprod-app1-<kurz-name> zum preprod-app2-<kurz-name> namespace noch möglich
  * Das wollen wir einschränken
  * Ausserdem von dev-app1-<name-kurz> zu dev-app2-<name>
```

bei dir anpassen

KURZ=jm

```
```

So sehen unsere Namespace - Labels aus

kubectl describe namespace dev-app1-$KURZ

```
```

Ausgabe, z.B.

Name: dev-app1-jm Labels: env=dev-jm ns=dev-app1-jm

```
```

nano 20-allow-ns-dev-app1-dev-app1.yaml

Traffic innerhalb des Namespaces erlaubt

apiVersion: crd.antrea.io/v1beta1 kind: ClusterNetworkPolicy metadata: name: 20-allow-ns-dev-app1-dev-app1- spec: priority: 100 tier: application appliedTo: - namespaceSelector: matchLabels: ns: dev-app1- ingress: - action: Allow from: - namespaceSelector: matchLabels: ns: dev-app1-

```
```

kubectl apply -f .

```
```

nano 25-drop-any-ns-dev-app2.yaml

allen anderen Traffic zum namespace app2 hin verbieten aus anderen namespaces

apiVersion: crd.antrea.io/v1beta1 kind: ClusterNetworkPolicy metadata: name: 25-drop-any-ns-dev-app2- spec: priority: 110 tier: application appliedTo: - namespaceSelector: matchLabels: ns: dev-app2- ingress: - action: Drop from: - namespaceSelector: {}

```
```

kubectl apply -f .

```
```

nano 30-allow-ns-preprod-app1-preprod-app1.yaml

Same for preprod-app1

Allow all traffic within namespace

apiVersion: crd.antrea.io/v1beta1 kind: ClusterNetworkPolicy metadata: name: 30-allow-ns-preprod-app1-preprod-app1- spec: priority: 120 tier: application appliedTo: - namespaceSelector: matchLabels: ns: preprod-app1- ingress: - action: Allow from: - namespaceSelector: matchLabels: ns: preprod-app1-

```
```

kubectl apply -f .

```
```

disallow all traffic from other namespaces to prepr

nano 35-drop-any-ns-preprod-app2.yaml

apiVersion: crd.antrea.io/v1beta1 kind: ClusterNetworkPolicy metadata: name: 21-drop-any-ns-preprod-app2 spec: priority: 130 tier: application appliedTo: - namespaceSelector: matchLabels: ns: preprod-app2- ingress: - action: Drop from: - namespaceSelector: {}

kubectl apply -f .

```
### Schritt 12: Isolate traffic within app2 - namespaces (3-Tier-app) (Das kann leider nur er Trainer machen ;o() - wg der Labels
```

For dev-app2- we want

web->app (80) app->db (3306) drop everything else

KURZ=jm;

kubectl -n dev-app2-$KURZ describe pods | head -n 20 kubectl -n preprod-app2-$KURZ describe pods | head -n 20

```
![image](https://github.com/jmetzger/training-kubernetes-networking/assets/1933318/bfd0b89b-aa47-4493-8952-3c2aff5f7f1c)

  * we are using the label app=xxx
```

nano 40-allow-web-app.yaml

apiVersion: crd.antrea.io/v1beta1 kind: ClusterNetworkPolicy metadata: name: 40-allow-web-app- spec: priority: 10 tier: application appliedTo: - podSelector: matchLabels: app: app ingress: - action: Allow from: - podSelector: matchLabels: app: nginx ports: - protocol: TCP port: 80

kubectl apply -f .

nano 45-allow-app-db.yaml

apiVersion: crd.antrea.io/v1beta1 kind: ClusterNetworkPolicy metadata: name: 02-allow-app-db- spec: priority: 20 tier: application appliedTo: - podSelector: matchLabels: app: mysql8 ingress: - action: Allow from: - podSelector: matchLabels: app: app ports: - protocol: TCP port: 3306

kubectl apply -f .

nano 50-deny-any-to-app2.yaml

Deny everything else

apiVersion: crd.antrea.io/v1beta1 kind: ClusterNetworkPolicy metadata: name: 03-deny-any-to-app2- spec: priority: 30 tier: application appliedTo: - namespaceSelector: matchLabels: ns: dev-app2- - namespaceSelector: matchLabels: ns: preprod-app2- ingress: - action: Drop from: - namespaceSelector: {}

kubectl apply -f .

```
### Schritt 13: Usage of the Emergency Tier - e.g. Attack (only Trainer)

  * We have problems with Ubuntu 16.04. an we want to isolate it.
```

kubectl get tiers

nano 80-emergency.yaml

apiVersion: crd.antrea.io/v1beta1 kind: ClusterNetworkPolicy metadata: name: 50-deny-any-pod-ubuntu16- spec: priority: 50 tier: emergency appliedTo: - podSelector: matchLabels: app: ubuntu-16-04 ingress: - action: Drop from: - namespaceSelector: {}

kubectl apply -f .

```
  * Because Emergency has the highest priority, the policy in application (allow any in ns-app1) has no Impact anymore.


### Reference:

  * https://www.vrealize.it/2020/09/28/securing-you-k8s-network-with-antrea-clusternetworkpolicy/


## Kubernetes - Wartung / Debugging

### kubectl drain/uncordon
```

Achtung, bitte keine pods verwenden, dies können "ge"-drained (ausgetrocknet) werden

kubectl drain z.B.

Daemonsets ignorieren, da diese nicht gelöscht werden

kubectl drain n17 --ignore-daemonsets

Alle pods von replicasets werden jetzt auf andere nodes verschoben

Ich kann jetzt wartungsarbeiten durchführen

Wenn fertig bin:

kubectl uncordon n17

Achtung: deployments werden nicht neu ausgerollt, dass muss ich anstossen.

z.B.

kubectl rollout restart deploy/webserver

```
### Alte manifeste konvertieren mit convert plugin


### What is about?

  * Plugins needs to be installed seperately on Client (or where you have your manifests)

### Walkthrough
```

curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl-convert"

Validate the 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

sudo install -o root -g root -m 0755 kubectl-convert /usr/local/bin/kubectl-convert

Does it work

kubectl convert --help

Works like so

Convert to the newest version

kubectl convert -f pod.yaml

```
### Reference

  * https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/#install-kubectl-convert-plugin

### Curl from pod api-server
```

```
https://nieldw.medium.com/curling-the-kubernetes-api-server-d7675cfc398c

## Kubernetes Deployment Scenarios

### Deployment green/blue,canary,rolling update


### Canary Deployment
```

A small group of the user base will see the new application (e.g. 1000 out of 100.000), all the others will still see the old version

From: a canary was used to test if the air was good in the mine (like a test balloon)

```
### Blue / Green Deployment
```

The current version is the Blue one The new version is the Green one

New Version (GREEN) will be tested and if it works
the traffic will be switch completey to the new version (GREEN)

Old version can either be deleted or will function as fallback

```
### A/B Deployment/Testing
```

2 Different versions are online, e.g. to test a new design / new feature You can configure the weight (how much traffic to one or the other) by the number of pods

```
#### Example Calculation
```

e.g. Deployment1: 10 pods Deployment2: 5 pods

Both have a common label, The service will access them through this label

```
### Service Blue/Green


### Step 1: Deployment + Service
```

vi blue.yml

apiVersion: apps/v1 kind: Deployment metadata: name: nginx-version-blue spec: selector: matchLabels: version: blue replicas: 10 # tells deployment to run 2 pods matching the template template: metadata: labels: app: nginx version: blue spec: containers: - name: nginx image: nginx:1.21 ports: - containerPort: 80

```

```

vi green.yml

apiVersion: apps/v1 kind: Deployment metadata: name: nginx-version-green spec: selector: matchLabels: version: green replicas: 1 # tells deployment to run 2 pods matching the template template: metadata: labels: app: nginx version: green spec: containers: - name: nginx image: nginx:1.22 ports: - containerPort: 80

```

```

svc.yml

apiVersion: v1 kind: Service metadata: name: svc-nginx spec: ports:

- port: 80 protocol: TCP selector: app: nginx

```
### Step 2: Ingress
```

apiVersion: networking.k8s.io/v1 kind: Ingress metadata: name: ingress-config annotations: ingress.kubernetes.io/rewrite-target: / # with the ingress controller from helm, you need to set an annotation # old version useClassName instead # otherwice it does not know, which controller to use # kubernetes.io/ingress.class: nginx spec: ingressClassName: nginx rules:

- host: "app.lab1.t3isp.de" http: paths: - path: / pathType: Prefix backend: service: name: svc-nginx port: number: 80

```

```

kubectl apply -f .

```

```

```
### Praxis-Übung A/B Deployment



### Walkthrough
```

cd cd manifests mkdir ab cd ab

```
```

vi 01-cm-version1.yml

apiVersion: v1 kind: ConfigMap metadata: name: nginx-version-1 data: index.html: |

Welcome to Version 1

Hi! This is a configmap Index file Version 1

```
```

vi 02-deployment-v1.yml

apiVersion: apps/v1 kind: Deployment metadata: name: nginx-deploy-v1 spec: selector: matchLabels: version: v1 replicas: 2 template: metadata: labels: app: nginx version: v1 spec: containers: - name: nginx image: nginx:latest ports: - containerPort: 80 volumeMounts: - name: nginx-index-file mountPath: /usr/share/nginx/html/ volumes: - name: nginx-index-file configMap: name: nginx-version-1

```
```

vi 03-cm-version2.yml

apiVersion: v1 kind: ConfigMap metadata: name: nginx-version-2 data: index.html: |

Welcome to Version 2

Hi! This is a configmap Index file Version 2

```
```

vi 04-deployment-v2.yml

apiVersion: apps/v1 kind: Deployment metadata: name: nginx-deploy-v2 spec: selector: matchLabels: version: v2 replicas: 2 template: metadata: labels: app: nginx version: v2 spec: containers: - name: nginx image: nginx:latest ports: - containerPort: 80 volumeMounts: - name: nginx-index-file mountPath: /usr/share/nginx/html/ volumes: - name: nginx-index-file configMap: name: nginx-version-2

```
```

vi 05-svc.yml

apiVersion: v1 kind: Service metadata: name: my-nginx labels: svc: nginx spec: type: NodePort ports:

- port: 80 protocol: TCP selector: app: nginx

```
```

kubectl apply -f .

get external ip

kubectl get nodes -o wide

get port

kubectl get svc my-nginx -o wide

test it with curl apply it multiple time (at least ten times)

curl :

```
## Helm (Kubernetes Paketmanager)

### Helm Grundlagen


### Wo ?
```

artifacts helm

```
 * https://artifacthub.io/

### Komponenten
```

Chart - beeinhaltet Beschreibung und Komponenten tar.gz - Format oder Verzeichnis

Wenn wir ein Chart ausführen wird eine Release erstellen (parallel: image -> container, analog: chart -> release)

```
### Installation
```

Beispiel ubuntu

snap install --classic helm

Cluster muss vorhanden, aber nicht notwendig wo helm installiert

Voraussetzung auf dem Client-Rechner (helm ist nichts als anderes als ein Client-Programm)

Ein lauffähiges kubectl auf dem lokalen System (welches sich mit dem Cluster verbinden kann). -> saubere -> .kube/config

Test

kubectl cluster-info

```


### Helm Warum ?
```

Ein Paket für alle Komponenten Einfaches Installieren, Updaten und deinstallieren Feststehende Struktur

```
### Helm Example


### Prerequisites

  * kubectl needs to be installed and configured to access cluster
  * Good: helm works as unprivileged user as well - Good for our setup
  * install helm on ubuntu (client) as root: snap install --classic helm
    * this installs helm3
  * Please only use: helm3. No server-side components needed (in cluster)
    * Get away from examples using helm2 (hint: helm init) - uses tiller

### Simple Walkthrough (Example 0)
```

Repo hinzufpgen

helm repo add bitnami https://charts.bitnami.com/bitnami

gecachte Informationen aktualieren

helm repo update

helm search repo bitnami

helm install release-name bitnami/mysql

helm install my-mysql bitnami/mysql

Chart runterziehen ohne installieren

helm pull bitnami/mysql

Release anzeigen zu lassen

helm list

Status einer Release / Achtung, heisst nicht unbedingt nicht, dass pod läuft

helm status my-mysql

weitere release installieren

helm install neuer-release-name bitnami/mysql

```
### Under the hood
```

Helm speichert Informationen über die Releases in den Secrets

kubectl get secrets | grep helm

```
### Example 1: - To get know the structure
```

helm repo add bitnami https://charts.bitnami.com/bitnami helm search repo bitnami helm repo update helm pull bitnami/mysql tar xzvf mysql-9.0.0.tgz

```
### Example 2: We will setup mysql without persistent storage (not helpful in production ;o()
```

helm repo add bitnami https://charts.bitnami.com/bitnami helm search repo bitnami helm repo update

helm install my-mysql bitnami/mysql

```
### Example 2 - continue - fehlerbehebung
```

helm uninstall my-mysql

Install with persistentStorage disabled - Setting a specific value

helm install my-mysql --set primary.persistence.enabled=false bitnami/mysql

just as notice

helm uninstall my-mysql

```
### Example 2b: using a values file
```

mkdir helm-mysql

cd helm-mysql

vi values.yml

primary: persistence: enabled: false

```

```

helm uninstall my-mysql helm install my-mysql bitnami/mysql -f values.yml

```
### Example 3: Install wordpress
```

helm repo add bitnami https://charts.bitnami.com/bitnami helm install my-wordpress
--set wordpressUsername=admin
--set wordpressPassword=password
--set mariadb.auth.rootPassword=secretpassword
bitnami/wordpress

```
### Example 4: Install Wordpress with values and auth
```

mkdir helm-mysql

cd helm-mysql

vi values.yml

persistence: enabled: false

wordpressUsername: admin wordpressPassword: password mariadb: primary: persistence: enabled: false

auth: rootPassword: secretpassword

```
```

helm uninstall my-wordpress helm install my-wordpress bitnami/wordpress -f values

```
### Referenced

  * https://github.com/bitnami/charts/tree/master/bitnami/mysql/#installing-the-chart
  * https://helm.sh/docs/intro/quickstart/

## Kubernetes - RBAC

### Nutzer einrichten microk8s ab kubernetes 1.25


### Enable RBAC in microk8s
```

This is important, if not enable every user on the system is allowed to do everything

do this on one of the nodes

microk8s enable rbac

```
### Schritt 1: Nutzer-Account auf Server anlegen und secret anlegen / in Client
```

cd mkdir -p manifests/rbac cd manifests/rbac

```
####  Mini-Schritt 1: Definition für Nutzer
```

vi 01-service-account.yml

apiVersion: v1 kind: ServiceAccount metadata: name: training namespace: default

```
```

kubectl apply -f .

```
#### Mini-Schritt 1.5: Secret erstellen

  * From Kubernetes 1.25 tokens are not created automatically when creating a service account (sa)
  * You have to create them manually with annotation attached
  * https://kubernetes.io/docs/reference/access-authn-authz/service-accounts-admin/#create-token
```

vi 02-secret.yml

apiVersion: v1 kind: Secret type: kubernetes.io/service-account-token metadata: name: trainingtoken annotations: kubernetes.io/service-account.name: training

```
```

kubectl apply -f .

```
#### Mini-Schritt 2: ClusterRolle festlegen - Dies gilt für alle namespaces, muss aber noch zugewiesen werden
```

Bevor sie zugewiesen ist, funktioniert sie nicht - da sie keinem Nutzer zugewiesen ist

vi 03-pods-clusterrole.yml

apiVersion: rbac.authorization.k8s.io/v1 kind: ClusterRole metadata: name: pods-clusterrole rules:

- apiGroups: [""] # "" indicates the core API group resources: ["pods"] verbs: ["get", "watch", "list"]

```
```

kubectl apply -f 03-pods-clusterrole.yml

```
#### Mini-Schritt 3: Die ClusterRolle den entsprechenden Nutzern über RoleBinding zu ordnen
```

vi 04-rb-training-ns-default-pods.yml

apiVersion: rbac.authorization.k8s.io/v1 kind: RoleBinding metadata: name: rolebinding-ns-default-pods namespace: default roleRef: apiGroup: rbac.authorization.k8s.io kind: ClusterRole name: pods-clusterrole subjects:

- kind: ServiceAccount name: training namespace: default

```
```

kubectl apply -f .

```
#### Mini-Schritt 4: Testen (klappt der Zugang)
```

kubectl auth can-i get pods -n default --as system:serviceaccount:default:training

yes

kubectl auth can-i get deployment -n default --as system:serviceaccount:default:training

no

```
### Schritt 2: Context anlegen / Credentials auslesen und in kubeconfig hinterlegen (bis Version 1.25.)

#### Mini-Schritt 1: kubeconfig setzen
```

kubectl config set-context training-ctx --cluster microk8s-cluster --user training

extract name of the token from here

TOKEN= `kubectl get secret trainingtoken -o jsonpath='{.data.token}' | base64 --decode` echo $TOKEN kubectl config set-credentials training --token=$TOKEN kubectl config use-context training-ctx

Hier reichen die Rechte nicht aus

kubectl get deploy

Error from server (Forbidden): pods is forbidden: User "system:serviceaccount:kube-system:training" cannot list # resource "pods" in API group "" in the namespace "default"

```
#### Mini-Schritt 2:
```

kubectl config use-context training-ctx kubectl get pods

```
#### Mini-Schritt 3: Zurück zum alten Default-Context
```

kubectl config get-contexts

```
```

CURRENT NAME CLUSTER AUTHINFO NAMESPACE microk8s microk8s-cluster admin2

-
  ```
      training-ctx   microk8s-cluster   training2
  ```

```
```

kubectl config use-context microk8s

```
### Refs:
```

```
  * https://docs.oracle.com/en-us/iaas/Content/ContEng/Tasks/contengaddingserviceaccttoken.htm
  * https://microk8s.io/docs/multi-user
  * https://faun.pub/kubernetes-rbac-use-one-role-in-multiple-namespaces-d1d08bb08286

### Ref: Create Service Account Token

  * https://kubernetes.io/docs/reference/access-authn-authz/service-accounts-admin/#create-token

### Tipps&Tricks zu Deploymnent - Rollout


### Warum
```

Rückgängig machen von deploys, Deploys neu unstossen. (Das sind die wichtigsten Fähigkeiten

```

### Beispiele
```

Deployment nochmal durchführen

z.B. nach kubectl uncordon n12.training.local

kubectl rollout restart deploy nginx-deployment

Rollout rückgängig machen

kubectl rollout undo deploy nginx-deployment

```

## Kubernetes QoS

### Quality of Service - evict pods


### Die Class wird auf Basis der Limits und Requests der Container vergeben
```

Request: Definiert wieviel ein Container mindestens braucht (CPU,memory) Limit: Definiert, was ein Container maximal braucht.

in spec.containers.resources kubectl explain pod.spec.containers.resources

```

### Art der Typen:

  * Guaranteed
  * Burstable
  * BestEffort

### Guaranteed
```

Type: Guaranteed: https://kubernetes.io/docs/tasks/configure-pod-container/quality-service-pod/#create-a-pod-that-gets-assigned-a-qos-class-of-guaranteed

set when limit equals request (request: das braucht er, limit: das braucht er maximal)

Garantied ist die höchste Stufe und diese werden bei fehlenden Ressourcen als letztes "evicted"

apiVersion: v1

kind: Pod metadata: name: qos-demo namespace: qos-example spec: containers:

- name: qos-demo-ctr image: nginx resources: limits: memory: "200Mi" cpu: "700m" requests: memory: "200Mi" cpu: "700m"

```




## Kustomize

### Kustomize Overlay Beispiel


### Konzept Overlay

  * Base + Overlay = Gepatchtes manifest
  * Sachen patchen.
  * Die werden drübergelegt.

### Example 1: Walkthrough
```

Step 1:

Create the structure

kustomize-example1

L base

| - kustomization.yml

L overlays

##. L dev

- kustomization.yml

##. L prod ##. - kustomization.yml cd; mkdir -p manifests/kustomize-example1/base; mkdir -p manifests/kustomize-example1/overlays/prod; cd manifests/kustomize-example1

Step 2: base dir with files

now create the base kustomization file

vi base/kustomization.yml

resources:

- service.yml

Step 3: Create the service - file

vi base/service.yml

kind: Service apiVersion: v1 metadata: name: service-app spec: type: ClusterIP selector: app: simple-app ports:

- name: http port: 80

See how it looks like

kubectl kustomize ./base

Step 4: create the customization file accordingly

##vi overlays/prod/kustomization.yaml bases:

- ../../base patches:
- service-ports.yaml

Step 5: create overlay (patch files)

vi overlays/prod/service-ports.yaml

kind: Service apiVersion: v1 metadata: #Name der zu patchenden Ressource name: service-app spec:

Changed to Nodeport

type: NodePort ports: #Die Porteinstellungen werden überschrieben

- name: https port: 443

Step 6:

kubectl kustomize overlays/prod

or apply it directly

kubectl apply -k overlays/prod/

```
```

Step 7:

mkdir -p overlays/dev

vi overlays/dev/kustomization

bases:

- ../../base

```
```

Step 8:

statt mit der base zu arbeiten

kubectl kustomize overlays/dev

```
### Example 2: Advanced Patching with patchesJson6902 (You need to have done example 1 firstly)
```

Schritt 1:

Replace overlays/prod/kustomization.yml with the following syntax

bases:

- ../../base patchesJson6902:
- target: version: v1 kind: Service name: service-app path: service-patch.yaml

```
```

Schritt 2:

vi overlays/prod/service-patch.yaml

- op: remove path: /spec/ports value:
  - name: http port: 80
- op: add
  path: /spec/ports value:
  - name: https port: 443

```
```

Schritt 3:

kubectl kustomize overlays/prod

```
### Special Use Case: Change the metadata.name
```

Same as Example 2, but patch-file is a bit different

vi overlays/prod/service-patch.yaml

- op: remove
  path: /spec/ports value:

  - name: http
    port: 80

- op: add
  path: /spec/ports
  value:

  - name: https
    port: 443
- op: replace
  path: /metadata/name value: svc-app-test

```
```

kubectl kustomize overlays/prod

```

### Ref:

  * https://blog.ordix.de/kubernetes-anwendungen-mit-kustomize



### Helm mit kustomize verheiraten

## Kubernetes - Tipps & Tricks

### Kubernetes Debuggen ClusterIP/PodIP


### Situation

  * Kein Zugriff auf die Nodes, zum Testen von Verbindungen zu Pods und Services über die PodIP/ClusterIP

### Lösung
```

Wir starten eine Busybox und fragen per wget und port ab

busytester ist der name

long version

kubectl run -it --rm --image=busybox busytester

wget

exit

quick and dirty

kubectl run -it --rm --image=busybox busytester -- wget

```

### Debugging pods


### How ?

  1. Which pod is in charge
  1. Problems when starting: kubectl describe po mypod
  1. Problems while running: kubectl logs mypod

### Taints und Tolerations


### Taints
```

Taints schliessen auf einer Node alle Pods aus, die nicht bestimmte taints haben:

Möglichkeiten:

o Sie werden nicht gescheduled - NoSchedule o Sie werden nicht executed - NoExecute o Sie werden möglichst nicht gescheduled. - PreferNoSchedule

```

### Tolerations
```

Tolerations werden auf Pod-Ebene vergeben: tolerations:

Ein Pod kann (wenn es auf einem Node taints gibt), nur gescheduled bzw. ausgeführt werden, wenn er die Labels hat, die auch als Taints auf dem Node vergeben sind.

```

### Walkthrough
```

```
#### Step 1: Cordon the other nodes - scheduling will not be possible there
```

Cordon nodes n11 and n111

You will see a taint here

kubectl cordon n11 kubectl cordon n111 kubectl describe n111 | grep -i taint

```



### Step 2: Set taint on first node
```

kubectl taint nodes n1 gpu=true:NoSchedule

```

### Step 3
```

cd mkdir -p manifests cd manifests mkdir tainttest cd tainttest nano 01-no-tolerations.yml

```
```

##vi 01-no-tolerations.yml apiVersion: v1 kind: Pod metadata: name: nginx-test-no-tol labels: env: test-env spec: containers:

- name: nginx image: nginx:1.21

```
```

kubectl apply -f . kubectl get po nginx-test-no-tol kubectl get describe nginx-test-no-tol

```

### Step 4:
```

vi 02-nginx-test-wrong-tol.yml

apiVersion: v1 kind: Pod metadata: name: nginx-test-wrong-tol labels: env: test-env spec: containers:

- name: nginx image: nginx:latest tolerations:
- key: "cpu" operator: "Equal" value: "true" effect: "NoSchedule"

```
```

kubectl apply -f . kubectl get po nginx-test-wrong-tol kubectl describe po nginx-test-wrong-tol

```

### Step 5:
```

vi 03-good-tolerations.yml

apiVersion: v1 kind: Pod metadata: name: nginx-test-good-tol labels: env: test-env spec: containers:

- name: nginx image: nginx:latest tolerations:
- key: "gpu" operator: "Equal" value: "true" effect: "NoSchedule"

```
```

kubectl apply -f . kubectl get po nginx-test-good-tol kubectl describe po nginx-test-good-tol

```

#### Taints rausnehmen
```

kubectl taint nodes n1 gpu:true:NoSchedule-

```

#### uncordon other nodes
```

kubectl uncordon n11 kubectl uncordon n111

```

### References

  * [Doku Kubernetes Taints and Tolerations](https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/)
  * https://blog.kubecost.com/blog/kubernetes-taints/

### Autoscaling Pods/Deployments

### Example: newest version with autoscaling/v2 used to be hpa/v1
```

---

apiVersion: apps/v1 kind: Deployment metadata: name: hello spec: replicas: 3 selector: matchLabels: app: hello template: metadata: labels: app: hello spec: containers: - name: hello image: k8s.gcr.io/hpa-example resources: requests: cpu: 100m

kind: Service apiVersion: v1 metadata: name: hello spec: selector: app: hello ports: - port: 80 targetPort: 80

apiVersion: autoscaling/v2 kind: HorizontalPodAutoscaler metadata: name: hello spec: scaleTargetRef: apiVersion: apps/v1 kind: Deployment name: hello minReplicas: 2 maxReplicas: 20 metrics:

- type: Resource resource: name: cpu target: type: Utilization averageUtilization: 80

```

  * https://docs.digitalocean.com/tutorials/cluster-autoscaling-ca-hpa/

### Reference

  * https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale-walkthrough/#autoscaling-on-more-specific-metrics
  * https://medium.com/expedia-group-tech/autoscaling-in-kubernetes-why-doesnt-the-horizontal-pod-autoscaler-work-for-me-
5f0094694054

### pod aus deployment bei config - Änderung neu ausrollen

  * https://github.com/stakater/Reloader

## Kubernetes Advanced

### Curl api-server kubernetes aus pod heraus


https://nieldw.medium.com/curling-the-kubernetes-api-server-d7675cfc398c

## Kubernetes - Documentation

### Documentation zu microk8s plugins/addons

  * https://microk8s.io/docs/addons

### Shared Volumes - Welche gibt es ?

  * https://kubernetes.io/docs/concepts/storage/volumes/

## Kubernetes - Hardening

### Kubernetes Tipps Hardening


### PSA (Pod Security Admission)
```

Policies defined by namespace. e.g. not allowed to run container as root.

Will complain/deny when creating such a pod with that container type

```

### Möglichkeiten in Pods und Containern
```

für die Pods

kubectl explain pod.spec.securityContext kubectl explain pod.spec.containers.securityContext

```

### Example (seccomp / security context)
```

A. seccomp - profile https://github.com/docker/docker/blob/master/profiles/seccomp/default.json

```

```

apiVersion: v1 kind: Pod metadata: name: audit-pod labels: app: audit-pod spec: securityContext: seccompProfile: type: Localhost localhostProfile: profiles/audit.json

containers:

- name: test-container image: hashicorp/http-echo:0.2.3 args:
  - "-text=just made some syscalls!" securityContext: allowPrivilegeEscalation: false

```
### SecurityContext (auf Pod Ebene)
```

kubectl explain pod.spec.containers.securityContext

```


### NetworkPolicy
```

Firewall Kubernetes

```
### Kubernetes Security Admission Controller Example



### Seit: 1.2.22 Pod Security Admission

  * 1.2.22 - ALpha - D.h. ist noch nicht aktiviert und muss als Feature Gate aktiviert (Kind)
  * 1.2.23 - Beta -> d.h. aktiviert

### Vorgefertigte Regelwerke

  * privileges - keinerlei Einschränkungen
  * baseline - einige Einschränkungen
  * restricted - sehr streng

### Praktisches Beispiel für Version ab 1.2.23 - Problemstellung
```

mkdir -p manifests cd manifests mkdir psa cd psa nano 01-ns.yml

```
```

Schritt 1: Namespace anlegen

vi 01-ns.yml

apiVersion: v1 kind: Namespace metadata: name: test-ns1 labels: pod-security.kubernetes.io/enforce: baseline pod-security.kubernetes.io/audit: restricted pod-security.kubernetes.io/warn: restricted

```
```

kubectl apply -f 01-ns.yml

```
```

Schritt 2: Testen mit nginx - pod

vi 02-nginx.yml

apiVersion: v1 kind: Pod metadata: name: nginx namespace: test-ns1 spec: containers: - image: nginx name: nginx ports: - containerPort: 80

```
```

a lot of warnings will come up

kubectl apply -f 02-nginx.yml

```
```

Schritt 3:

Anpassen der Sicherheitseinstellung (Phase1) im Container

vi 02-nginx.yml

apiVersion: v1 kind: Pod metadata: name: nginx namespace: test-ns1 spec: containers: - image: nginx name: nginx ports: - containerPort: 80 securityContext:
seccompProfile:
type: RuntimeDefault

```
```

kubectl delete -f 02-nginx.yml kubectl apply -f 02-nginx.yml kubectl -n test-ns1 get pods

```
```

Schritt 4:

Weitere Anpassung runAsNotRoot

vi 02-nginx.yml

apiVersion: v1 kind: Pod metadata: name: nginx namespace: test-ns spec: containers: - image: nginx name: nginx ports: - containerPort: 80 securityContext: seccompProfile: type: RuntimeDefault runAsNonRoot: true

```
```

pod kann erstellt werden, wird aber nicht gestartet

kubectl delete -f 02-nginx.yml kubectl apply -f 02-nginx.yml kubectl -n test-ns1 get pods kubectl -n test-ns1 describe pods nginx

```
### Praktisches Beispiel für Version ab 1.2.23 -Lösung - Container als NICHT-Root laufen lassen

  * Wir müssen ein image, dass auch als NICHT-Root laufen kann
  * .. oder selbst eines bauen (;o))
  o bei nginx ist das bitnami/nginx
```

vi 03-nginx-bitnami.yml

apiVersion: v1 kind: Pod metadata: name: bitnami-nginx namespace: test-ns1 spec: containers: - image: bitnami/nginx name: bitnami-nginx ports: - containerPort: 80 securityContext: seccompProfile: type: RuntimeDefault runAsNonRoot: true

```
```

und er läuft als nicht root

kubectl apply -f 03_pod-bitnami.yml kubectl -n test-ns1 get pods

```
### Was muss ich bei der Netzwerk-Sicherheit beachten ?


### Bereich 1: Kubernetes (Cluster)
```

1. Welche Ports sollten wirklich geöffnet sein ?

für Kubernetes

2. Wer muss den von wo den Kube-Api-Server zugreifen

- den Traffic einschränken

```
### Bereich 2: Nodes
```

Alle nicht benötigten fremden Ports sollten geschlossen sein Wenn offen, nur über vordefinierte Zugangswege (und auch nur bestimmte Nutzer)

```
### Pods (Container / Image)
```

Ingress (NetworkPolicy) - engmaschig stricken

1. Wer soll von wo auf welche Pod zugreifen können

2. Welche Pod auf welchen anderen Pod (Service)

ä Egress

Welche Pods dürfen wohin nach draussen

```
### Einschränking der Fähigkeien eines Pods
```

kein PrivilegeEscalation nur notwendige Capabilities unter einem nicht-root Benutzer laufen lassen ...

Patching

```
## pods -> neuestes images bei security vulnerablities
## nodes -> auch neues patches (apt upgrade)
## kubernetes cluster -> auf dem neuesten Stand
  # -> wie ist der Prozess ClusterUpdate, update der manifeste zu neuen API-Versionen
```

RBAC

```
## Nutzer (kubectl, systemnutzer -> pods)

## 1. Zugriff von den pods


## 2. Zugriff über helm / kubectl
## Wer darf was ? Was muss der Nutzer können
```

Compliance

```
PSP's / PSA
PodSecurityPolicy was deprecated in Kubernetes v1.21, and removed from Kubernetes in v1.25

PSA - Pode Security Admission
```

Kubernetes Interna / Misc.

OCI,Container,Images Standards

Schritt 1:

```
cd
mkdir bautest
cd bautest
```

Schritt 2:

```
## nano docker-compose.yml
version: "3.8"

services:
  myubuntu:
    build: ./myubuntu
    restart: always
```

Schritt 3:

```
mkdir myubuntu
cd myubuntu
```

```
nano hello.sh
```

```
##!/bin/bash
let i=0

while true
do
  let i=i+1
  echo $i:hello-docker
  sleep 5
done
```

```
## nano Dockerfile
FROM ubuntu:latest
RUN apt-get update; apt-get install -y inetutils-ping
COPY hello.sh .
RUN chmod u+x hello.sh
CMD ["/hello.sh"]
```

Schritt 4:

```
cd ../
## wichtig, im docker-compose - Ordner seiend
##pwd
##~/bautest
docker-compose up -d
```

```
## wird image gebaut und container gestartet

## Bei Veränderung vom Dockerfile, muss man den Parameter --build mitangeben
docker-compose up -d --build
```

Geolocation Kubernetes Cluster

- https://learnk8s.io/bite-sized/connecting-multiple-kubernetes-clusters

statische IP für Pod in calico

- https://docs.tigera.io/calico/latest/networking/ipam/use-specific-ip

yaml linting

- https://www.kubeval.com/installation/

ssl terminierung über proxy nginx

mit ssl

- https://jackiechen.blog/2019/01/24/nginx-sample-config-of-http-and-ldaps-reverse-proxy/

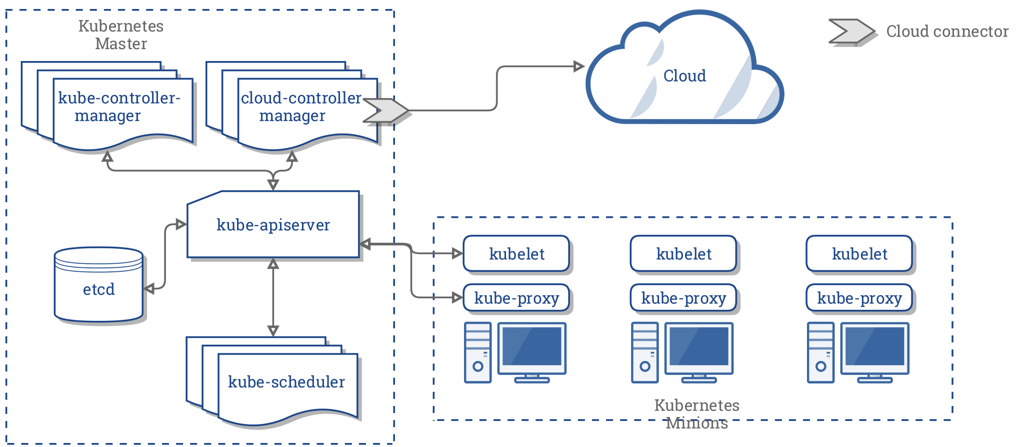
Ohne ssl

- https://kubernetes.github.io/ingress-nginx/user-guide/exposing-tcp-udp-services/

LoadBalancer / Cluster Controller Manager

Keypart: Cluster Controller Manager (CCM)

- was decoupled from Kube Controller Manager
  - to make it easier for cloud providers to implement their specific environment/workings (e.g. LoadBalancer)
- To do this a skeleton was provided.



Control Loops in the CCM

- Der CCM erbt seine Funktionen von Komponenten des Kubernetes, die von einem Cloud Provider abhängig sind.
- Die meisten Funktionen des CCM stammen aus dem KCM. Wie im vorherigen Abschnitt erwähnt, führt das CCM die folgenden Steuerschleifen durch:

```
Node Controller
Route Controller
Service Controller
```

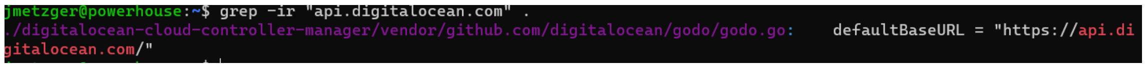
Service Controller

```
Der Service Controller ist verantwortlich für das Abhören von Ereignissen zum Erstellen, Aktualisieren und Löschen von Diensten.
Basierend auf dem aktuellen Stand der Services in Kubernetes konfiguriert es Cloud Load Balancer (wie ELB, Google LB oder Oracle
Cloud Infrastructure LB), um den Zustand der Services in Kubernetes abzubilden. Darüber hinaus wird sichergestellt, dass die
Service Backends für Cloud Loadbalancer auf dem neuesten Stand sind.
```

Load Balancer Implementation in DigitalOcean (DO)

- https://github.com/digitalocean/digitalocean-cloud-controller-manager/tree/master
- https://github.com/digitalocean/digitalocean-cloud-controller-manager/blob/master/cloud-controller-manager/do/loadbalancers.go

api - domain is hardcoded in cloud controller manager for digitalocean



```
jmetzger@powerhouse:~$ grep -ir "api.digitalocean.com" .
./digitalocean-cloud-controller-manager/vendor/github.com/digitalocean/godo/godo.go:    defaultBaseURL = "https://api.di
gitalocean.com/"
```

References:

- Good explanation
- Zugrundeliegende Konzepte

Kubernetes - Ingress

Ingress controller in microk8s aktivieren

Aktivieren

```
microk8s enable ingress
```

Referenz

- https://microk8s.io/docs/addon-ingress

ingress mit ssl absichern

Kubernetes Documentation

Well-Known Annotations

- https://kubernetes.io/docs/reference/labels-annotations-taints/

Kubernetes - Überblick

Installation - Welche Komponenten from scratch

Step 1: Server 1 (manuell installiert -> microk8s)

```
## Installation Ubuntu - Server

## cloud-init script
## s.u. BASIS (keine Voraussetzung - nur zum Einrichten des Nutzers 11trainingdo per ssh)

## Server 1 - manuell
## Ubuntu 20.04 LTS - Grundinstallation

## minimal Netzwerk - öffentlichen IP
## nichts besonderes eingerichtet - Standard Digitalocean

## Standard vo Installation microk8s
lo               UNKNOWN        127.0.0.1/8 ::1/128
## public ip / interne
eth0             UP             164.92.255.234/20 10.19.0.6/16 fe80::c:66ff:fec4:cbce/64
## private ip
eth1             UP             10.135.0.3/16 fe80::8081:aaff:feaa:780/64

snap install microk8s --classic
## namensaufloesung fuer pods
microk8s enable dns
```

```
## Funktioniert microk8s
microk8s status
```

Steps 2: Server 2+3 (automatische Installation -> microk8s)

```
## Was macht das ?
## 1. Basisnutzer (11trainingdo) - keine Voraussetzung für microk8s
## 2. Installation von microk8s
##.>>>>>>> microk8s installiert <<<<<<<<
## - snap install --classic microk8s
## >>>>>>> Zuordnung zur Gruppe microk8s - notwendig für bestimmte plugins (z.B. helm)
## usermod -a -G microk8s root
## >>>>>>> Setzen des .kube - Verzeichnisses auf den Nutzer microk8s -> nicht zwingend erforderlich
## chown -r -R microk8s ~/.kube
## >>>>>>> REQUIRED .. DNS aktivieren, wichtig für Namensauflösungen innerhalb der PODS
## >>>>>>> sonst funktioniert das nicht !!!
## microk8s enable dns
## >>>>>>> kubectl alias gesetzt, damit man nicht immer microk8s kubectl eingeben muss
## - echo "alias kubectl='microk8s kubectl'" >> /root/.bashrc
## cloud-init script
## s.u. MITMICROK8S (keine Voraussetzung - nur zum Einrichten des Nutzers 11trainingdo per ssh)
##cloud-config
users:
  - name: 11trainingdo
    shell: /bin/bash

runcmd:
  - sed -i "s/PasswordAuthentication no/PasswordAuthentication yes/g" /etc/ssh/sshd_config
  - echo " " >> /etc/ssh/sshd_config
  - echo "AllowUsers 11trainingdo" >> /etc/ssh/sshd_config
  - echo "AllowUsers root" >> /etc/ssh/sshd_config
  - systemctl reload sshd
  - sed -i '/11trainingdo/c
11trainingdo:$6$HeLUJW3a$4xSfDFQjKWfAoGkZF3LFAxM4hgl3d6ATbr2kEu9zMOFwLxkYMO.AJF526mZONwdmsm9sg0tCBKl.SYbhS52u70:17476:0:99999:7:::'
 /etc/shadow
  - echo "11trainingdo ALL=(ALL) ALL" > /etc/sudoers.d/11trainingdo
  - chmod 0440 /etc/sudoers.d/11trainingdo

  - echo "Installing microk8s"
  - snap install --classic microk8s
  - usermod -a -G microk8s root
  - chown -f -R microk8s ~/.kube
  - microk8s enable dns
  - echo "alias kubectl='microk8s kubectl'" >> /root/.bashrc
```

```
## Prüfen ob microk8s - wird automatisch nach Installation gestartet
## kann eine Weile dauern
microk8s status
```

Step 3: Client - Maschine (wir sollten nicht auf control-plane oder cluster - node arbeiten

```
Weiteren Server hochgezogen.
Vanilla + BASIS

## Installation Ubuntu - Server

## cloud-init script
## s.u. BASIS (keine Voraussetzung - nur zum Einrichten des Nutzers 11trainingdo per ssh)

## Server 1 - manuell
## Ubuntu 20.04 LTS - Grundinstallation

## minimal Netzwerk - öffentlichen IP
## nichts besonderes eingerichtet - Standard Digitalocean

## Standard vo Installation microk8s
lo               UNKNOWN        127.0.0.1/8 ::1/128
## public ip / interne
eth0             UP             164.92.255.232/20 10.19.0.6/16 fe80::c:66ff:fec4:cbce/64
## private ip
eth1             UP             10.135.0.5/16 fe80::8081:aaff:feaa:780/64
```

```
##### Installation von kubectl aus dem snap
## NICHT .. keine microk8s - keine control-plane / worker-node
## NUR Client zum Arbeiten
snap install kubectl --classic

##### .kube/config
## Damit ein Zugriff auf die kube-server-api möglich
## d.h. REST-API Interface, um das Cluster verwalten.
## Hier haben uns für den ersten Control-Node entschieden
## Alternativ wäre round-robin per dns möglich

## Mini-Schritt 1:
## Auf dem Server 1: kubeconfig ausspielen
microk8s config > /root/kube-config
## auf das Zielsystem gebracht (client 1)
scp /root/kubeconfig 11trainingdo@10.135.0.5:/home/11trainingdo

## Mini-Schritt 2:
## Auf dem Client 1 (diese Maschine) kubeconfig an die richtige Stelle bringen
## Standardmäßig der Client nach eine Konfigurationsdatei sucht in ~/.kube/config
sudo su -
cd
mkdir .kube
cd .kube
```

```
mv /home/11trainingdo/kube-config config

## Verbindungstest gemacht
## Damit feststellen ob das funktioniert.
kubectl cluster-info
```

Schritt 4: Auf allen Servern IP's hinterlegen und richtigen Hostnamen überprüfen

```
## Auf jedem Server
hostnamectl
## evtl. hostname setzen
## z.B. - auf jedem Server eindeutig
hostnamectl set-hostname n1.training.local

## Gleiche hosts auf allen server einrichten.
## Wichtig, um Traffic zu minimieren verwenden, die interne (private) IP

/etc/hosts
10.135.0.3 n1.training.local n1
10.135.0.4 n2.training.local n2
10.135.0.5 n3.training.local n3
```

Schritt 5: Cluster aufbauen

```
## Mini-Schritt 1:
## Server 1: connection - string (token)
microk8s add-node
## Zeigt Liste und wir nehmen den Eintrag mit der lokalen / öffentlichen ip
## Dieser Token kann nur 1x verwendet werden und wir auf dem ANDEREN node ausgeführt
## microk8s join 10.135.0.3:25000/e9cdaa11b5d6d24461c8643cdf107837/bcad1949221a

## Mini-Schritt 2:
## Dauert eine Weile, bis das durch ist.
## Server 2: Den Node hinzufügen durch den JOIN - Befehl
microk8s join 10.135.0.3:25000/e9cdaa11b5d6d24461c8643cdf107837/bcad1949221a

## Mini-Schritt 3:
## Server 1: token besorgen für node 3
microk8s add-node

## Mini-Schritt 4:
## Server 3: Den Node hinzufügen durch den JOIN-Befehl
microk8s join 10.135.0.3:25000/09c96e57ec12af45b2752fb45450530c/bcad1949221a

## Mini-Schritt 5: Überprüfen ob HA-Cluster läuft
Server 1: (es kann auf jedem der 3 Server überprüft werden, auf einem reicht
microk8s status | grep high-availability
high-availability: yes
```

Ergänzend nicht notwendige Scripte

```
## cloud-init script
## s.u. BASIS (keine Voraussetzung - nur zum Einrichten des Nutzers 11trainingdo per ssh)

## Digitalocean - unter user_data reingepastet beim Einrichten

##cloud-config
users:
  - name: 11trainingdo
    shell: /bin/bash

runcmd:
  - sed -i "s/PasswordAuthentication no/PasswordAuthentication yes/g" /etc/ssh/sshd_config
  - echo " " >> /etc/ssh/sshd_config
  - echo "AllowUsers 11trainingdo" >> /etc/ssh/sshd_config
  - echo "AllowUsers root" >> /etc/ssh/sshd_config
  - systemctl reload sshd
  - sed -i '/11trainingdo/c
11trainingdo:$6$HeLUJW3a$4xSfDFQjKWfAoGkZF3LFAxM4hgl3d6ATbr2kEu9zMOFwLxkYMO.AJF526mZONwdmsm9sg0tCBKl.SYbhS52u70:17476:0:99999:7:::'
 /etc/shadow
  - echo "11trainingdo ALL=(ALL) ALL" > /etc/sudoers.d/11trainingdo
  - chmod 0440 /etc/sudoers.d/11trainingdo
```

Kubernetes - microk8s (Installation und Management)

kubectl unter windows - Remote-Verbindung zu Kuberenets (microk8s) einrichten

Walkthrough (Installation)

```
## Step 1
chocolatry installiert.
(powershell als Administrator ausführen)
## https://docs.chocolatey.org/en-us/choco/setup
Set-ExecutionPolicy Bypass -Scope Process -Force; [System.Net.ServicePointManager]::SecurityProtocol =
[System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-Object
System.Net.WebClient).DownloadString('https://community.chocolatey.org/install.ps1'))

## Step 2
choco install kubernetes-cli

## Step 3
testen:
kubectl version --client

## Step 4:
## powershell als normaler benutzer öffnen
```

Walkthrough (autocompletion)

```
in powershell (normaler Benutzer)
kubectl completion powershell | Out-String | Invoke-Expression
```

kubectl - config - Struktur vorbereiten

```
## in powershell im heimatordner des Benutzers .kube - ordnern anlegen
## C:\Users\<dein-name>\
mkdir .kube
cd .kube
```

IP von Cluster-Node bekommen

```
## auf virtualbox - maschine per ssh einloggen
## öffentliche ip herausfinden - z.B. enp0s8 bei HostOnly - Adapter
ip -br a
```

config für kubectl aus Cluster-Node auslesen (microk8s)

```
## auf virtualbox - maschine per ssh einloggen / zum root wechseln
## abfragen
microk8s config

## Alle Zeilen ins clipboard kopieren
## und mit notepad++ in die Datei \Users\<dein-name>\.kube\config
## schreiben

## Wichtig: Zeile cluster -> clusters / server
## Hier ip von letztem Schritt eintragen:
## z.B.
Server: https://192.168.56.106/......
```

Testen

```
## in powershell
## kann ich eine Verbindung zum Cluster aufbauen ?
kubectl cluster-info
```

- https://kubernetes.io/docs/tasks/tools/install-kubectl-windows/

Arbeiten mit der Registry

Installation Kubernetes Dashboard

Reference:

- https://blog.tippybits.com/installing-kubernetes-in-virtualbox-3d49f666b4d6

Kubernetes - RBAC

Nutzer einrichten - kubernetes bis 1.24

Enable RBAC in microk8s

```
## This is important, if not enable every user on the system is allowed to do everything
microk8s enable rbac
```

Schritt 1: Nutzer-Account auf Server anlegen / in Client

```
cd
mkdir -p manifests/rbac
cd manifests/rbac
```

Mini-Schritt 1: Definition für Nutzer

```
## vi service-account.yml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: training
  namespace: default
```

```
kubectl apply -f service-account.yml
```

Mini-Schritt 2: ClusterRolle festlegen - Dies gilt für alle namespaces, muss aber noch zugewiesen werden

```
### Bevor sie zugewiesen ist, funktioniert sie nicht - da sie keinem Nutzer zugewiesen ist

## vi pods-clusterrole.yml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: pods-clusterrole
rules:
- apiGroups: [""] # "" indicates the core API group
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
```

```
kubectl apply -f pods-clusterrole.yml
```

Mini-Schritt 3: Die ClusterRolle den entsprechenden Nutzern über RoleBinding zu ordnen

```
## vi rb-training-ns-default-pods.yml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: rolebinding-ns-default-pods
  namespace: default
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: pods-clusterrole
subjects:
- kind: ServiceAccount
  name: training
  namespace: default
```

```
kubectl apply -f rb-training-ns-default-pods.yml
```

Mini-Schritt 4: Testen (klappt der Zugang)

```
kubectl auth can-i get pods -n default --as system:serviceaccount:default:training
```

Schritt 2: Context anlegen / Credentials auslesen und in kubeconfig hinterlegen (bis Version 1.25.)

Mini-Schritt 1: kubeconfig setzen

```
kubectl config set-context training-ctx --cluster microk8s-cluster --user training

## extract name of the token from here

TOKEN=`kubectl get secret trainingtoken -o jsonpath='{.data.token}' | base64 --decode`
echo $TOKEN
kubectl config set-credentials training --token=$TOKEN
kubectl config use-context training-ctx

## Hier reichen die Rechte nicht aus
kubectl get deploy
```

```
## Error from server (Forbidden): pods is forbidden: User "system:serviceaccount:kube-system:training" cannot list # resource
"pods" in API group "" in the namespace "default"
```

Mini-Schritt 2:

```
kubectl config use-context training-ctx
kubectl get pods
```

Refs:

- https://docs.oracle.com/en-us/iaas/Content/ContEng/Tasks/contengaddingserviceaccttoken.htm
- https://microk8s.io/docs/multi-user
- https://faun.pub/kubernetes-rbac-use-one-role-in-multiple-namespaces-d1d08bb08286

Ref: Create Service Account Token

- https://kubernetes.io/docs/reference/access-authn-authz/service-accounts-admin/#create-token

kubectl

Tipps&Tricks zu Deploymnent - Rollout

Warum

```
Rückgängig machen von deploys, Deploys neu unstossen.
(Das sind die wichtigsten Fähigkeiten
```

Beispiele

```
## Deployment nochmal durchführen
## z.B. nach kubectl uncordon n12.training.local
kubectl rollout restart deploy nginx-deployment

## Rollout rückgängig machen
kubectl rollout undo deploy nginx-deployment
```

Kubernetes - Monitoring (microk8s und vanilla)

metrics-server aktivieren (microk8s und vanilla)

Warum ? Was macht er ?

```
Der Metrics-Server sammelt Informationen von den einzelnen Nodes und Pods
Er bietet mit

kubectl top pods
kubectl top nodes

ein einfaches Interface, um einen ersten Eindruck über die Auslastung zu bekommen.
```

Walktrough

```
## Auf einem der Nodes im Cluster (HA-Cluster)
microk8s enable metrics-server

## Es dauert jetzt einen Moment bis dieser aktiv ist auch nach der Installation
## Auf dem Client
kubectl top nodes
kubectl top pods
```

Kubernetes

- https://kubernetes-sigs.github.io/metrics-server/
- kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml

Kubernetes - Backups

Kubernetes - Tipps & Tricks

Assigning Pods to Nodes

Walkthrough

```
## leave n3 as is
kubectl label nodes n7 rechenzentrum=rz1
kubectl label nodes n17 rechenzentrum=rz2
kubectl label nodes n27 rechenzentrum=rz2
kubectl get nodes --show-labels
```

```
## nginx-deployment
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 9 # tells deployment to run 2 pods matching the template
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80
      nodeSelector:
        rechenzentrum: rz2

## Let's rewrite that to deployment
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    env: test
spec:
  containers:
  - name: nginx
    image: nginx
    imagePullPolicy: IfNotPresent
  nodeSelector:
    rechenzentrum=rz2
```

Ref:

- https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/

Kubernetes - Documentation

LDAP-Anbindung

- https://github.com/apprenda-kismatic/kubernetes-ldap

Helpful to learn - Kubernetes

- https://kubernetes.io/docs/tasks/

Environment to learn

- https://killercoda.com/killer-shell-cks

Environment to learn II

- https://killercoda.com/

Youtube Channel

- https://www.youtube.com/watch?v=01qcYSck1c4

Kubernetes - Shared Volumes

Shared Volumes with nfs

Create new server and install nfs-server

```
## on Ubuntu 20.04LTS
apt install nfs-kernel-server
systemctl status nfs-server

vi /etc/exports
## adjust ip's of kubernetes master and nodes
## kmaster
```

```
/var/nfs/ 192.168.56.101(rw,sync,no_root_squash,no_subtree_check)
## knode1
/var/nfs/ 192.168.56.103(rw,sync,no_root_squash,no_subtree_check)
## knode 2
/var/nfs/ 192.168.56.105(rw,sync,no_root_squash,no_subtree_check)

exportfs -av
```

On all nodes (needed for production)

```
##
apt install nfs-common
```

On all nodes (only for testing)

```
#### Please do this on all servers (if you have access by ssh)
### find out, if connection to nfs works !

## for testing
mkdir /mnt/nfs
## 10.135.0.18 is our nfs-server
mount -t nfs 10.135.0.18:/var/nfs /mnt/nfs
ls -la /mnt/nfs
umount /mnt/nfs
```

Persistent Storage-Step 1: Setup PersistentVolume in cluster

```
cd
cd manifests
mkdir -p nfs
cd nfs
nano 01-pv.yml
```

```
apiVersion: v1
kind: PersistentVolume
metadata:
  # any PV name
  name: pv-nfs-tln<nr>
  labels:
    volume: nfs-data-volume-tln<nr>
spec:
  capacity:
    # storage size
    storage: 1Gi
  accessModes:
    # ReadWriteMany(RW from multi nodes), ReadWriteOnce(RW from a node), ReadOnlyMany(R from multi nodes)
    - ReadWriteMany
  persistentVolumeReclaimPolicy:
    # retain even if pods terminate
    Retain
  nfs:
    # NFS server's definition
    path: /var/nfs/tln<nr>/nginx
    server: 10.135.0.18
    readOnly: false
  storageClassName: ""
```

```
kubectl apply -f 01-pv.yml
kubectl get pv
```

Persistent Storage-Step 2: Create Persistent Volume Claim

```
nano 02-pvc.yml
```

```
## vi 02-pvc.yml
## now we want to claim space
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pv-nfs-claim-tln<nr>
spec:
  storageClassName: ""
  volumeName: pv-nfs-tln<nr>
  accessModes:
  - ReadWriteMany
  resources:
     requests:
       storage: 1Gi
```

```
kubectl apply -f 02-pvc.yml
kubectl get pvc
```

Persistent Storage-Step 3: Deployment

```
## deployment including mount
## vi 03-deploy.yml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 4 # tells deployment to run 4 pods matching the template
  template:
    metadata:
      labels:
        app: nginx
    spec:

      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80

        volumeMounts:
          - name: nfsvol
            mountPath: "/usr/share/nginx/html"

      volumes:
      - name: nfsvol
        persistentVolumeClaim:
          claimName: pv-nfs-claim-tln<tln>
```

```
kubectl apply -f 03-deploy.yml
```

Persistent Storage Step 4: service

```
## now testing it with a service
## cat 04-service.yml
apiVersion: v1
kind: Service
metadata:
  name: service-nginx
  labels:
    run: svc-my-nginx
spec:
  type: NodePort
  ports:
  - port: 80
    protocol: TCP
  selector:
    app: nginx
```

```
kubectl apply -f 04-service.yml
```

Persistent Storage Step 5: write data and test

```
## connect to the container and add index.html - data
kubectl exec -it deploy/nginx-deployment -- bash
## in container
echo "hello dear friend" > /usr/share/nginx/html/index.html
exit

## now try to connect
kubectl get svc
```

```
## connect with ip and port
kubectl run -it --rm curly --image=curlimages/curl -- /bin/sh
## curl http://<cluster-ip>
## exit

## now destroy deployment
kubectl delete -f 03-deploy.yml

## Try again - no connection
kubectl run -it --rm curly --image=curlimages/curl -- /bin/sh
## curl http://<cluster-ip>
## exit
```

Persistent Storage Step 6: retest after redeployment

```
## now start deployment again
kubectl apply -f 03-deploy.yml

## and try connection again
kubectl run -it --rm curly --image=curlimages/curl -- /bin/sh
## curl http://<cluster-ip>
## exit
```

Kubernetes - Hardening

Kubernetes Tipps Hardening

PSA (Pod Security Admission)

```
Policies defined by namespace.
e.g. not allowed to run container as root.

Will complain/deny when creating such a pod with that container type
```

Möglichkeiten in Pods und Containern

```
## für die Pods
kubectl explain pod.spec.securityContext
kubectl explain pod.spec.containers.securityContext
```

Example (seccomp / security context)

```
A. seccomp - profile
https://github.com/docker/docker/blob/master/profiles/seccomp/default.json
```

```
apiVersion: v1
kind: Pod
metadata:
  name: audit-pod
  labels:
    app: audit-pod
spec:
  securityContext:
    seccompProfile:
      type: Localhost
      localhostProfile: profiles/audit.json

  containers:

  - name: test-container
    image: hashicorp/http-echo:0.2.3
    args:
    - "-text=just made some syscalls!"
    securityContext:
      allowPrivilegeEscalation: false
```

SecurityContext (auf Pod Ebene)

```
kubectl explain pod.spec.containers.securityContext
```

NetworkPolicy

```
## Firewall Kubernetes
```

Kubernetes Probes (Liveness and Readiness)

Übung Liveness-Probe

Übung 1: Liveness (command)

```
What does it do ?

* At the beginning pod is ready (first 30 seconds)
* Check will be done after 5 seconds of pod being startet
* Check will be done periodically every 5 minutes and will check
  * for /tmp/healthy
  * if file is there will return: 0
  * if file is not there will return: 1
* After 30 seconds container will be killed
* After 35 seconds container will be restarted
```

```
## cd
## mkdir -p manifests/probes
## cd manifests/probes
## vi 01-pod-liveness-command.yml

apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-exec
spec:
  containers:
  - name: liveness
    image: busybox
    args:
    - /bin/sh
    - -c
    - touch /tmp/healthy; sleep 30; rm -f /tmp/healthy; sleep 600
    livenessProbe:
      exec:
        command:
        - cat
        - /tmp/healthy
      initialDelaySeconds: 5
      periodSeconds: 5
```

```
## apply and test
kubectl apply -f 01-pod-liveness-command.yml
kubectl describe -l test=liveness pods
sleep 30
kubectl describe -l test=liveness pods
sleep 5
kubectl describe -l test=liveness pods
```

```
## cleanup
kubectl delete -f 01-pod-liveness-command.yml
```

Übung 2: Liveness Probe (HTTP)

```
## Step 0: Understanding Prerequisite:
This is how this image works:
## after 10 seconds it returns code 500
http.HandleFunc("/healthz", func(w http.ResponseWriter, r *http.Request) {
    duration := time.Now().Sub(started)
    if duration.Seconds() > 10 {
        w.WriteHeader(500)
        w.Write([]byte(fmt.Sprintf("error: %v", duration.Seconds())))
    } else {
        w.WriteHeader(200)
        w.Write([]byte("ok"))
    }
})
```

```
## Step 1: Pod  - manifest
## vi 02-pod-liveness-http.yml
## status-code >=200 and < 400 o.k.
## else failure
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-http
spec:
  containers:
  - name: liveness
    image: k8s.gcr.io/liveness
    args:
    - /server
    livenessProbe:
      httpGet:
        path: /healthz
        port: 8080
        httpHeaders:
        - name: Custom-Header
          value: Awesome
      initialDelaySeconds: 3
      periodSeconds: 3
```

```
## Step 2: apply and test
kubectl apply -f 02-pod-liveness-http.yml
## after 10 seconds port should have been started
sleep 10
kubectl describe pod liveness-http
```

Reference:

- https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/

Funktionsweise Readiness-Probe vs. Liveness-Probe

Why / Howto /

- Readiness checks, if container is ready and if it's not READY
  - SENDS NO TRAFFIC to the container

Difference to LiveNess

- They are configured exactly the same, but use another keyword
  - readinessProbe instead of livenessProbe

Example

```
readinessProbe:
  exec:
    command:
    - cat
    - /tmp/healthy
  initialDelaySeconds: 5
  periodSeconds: 5
```

Reference

- https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/#define-readiness-probes