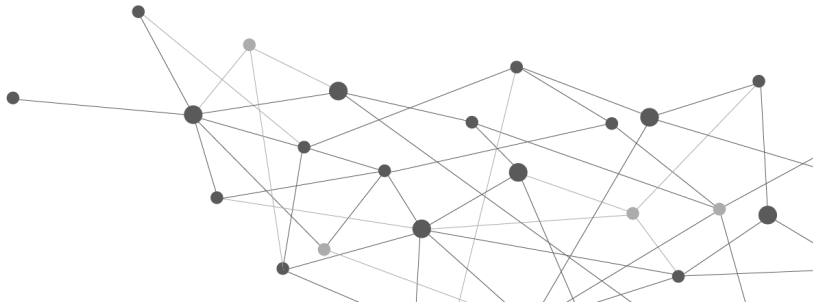
cronn

wir entwickeln software_

Hands-on kubernetes workshop

Brühl, 17.03.2022



About me_

- Software Engineer @cronn
  - 110 people - Bonn, Hamburg, Białystok

github.com/chrisingenhaag

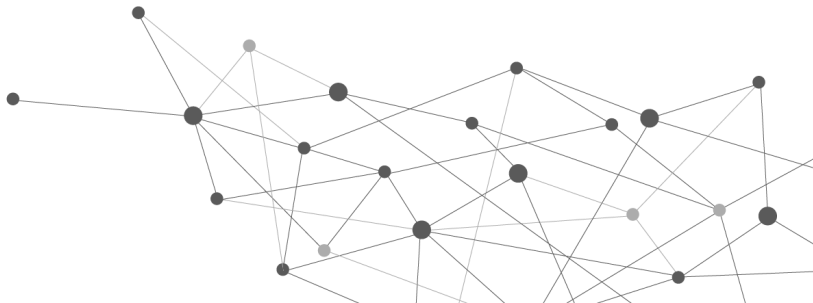
@chris_ingenhaag

christian.ingenhaag@cronn.de

Agenda_

1. intro
2. kubernetes fast start
3. running a pod / basic deployment
4. manage traffic (service, ingress)
5. configurations
6. advanced deployments
7. persistence (optional)

preparation_



Preparation_

We are working on a cronn kubernetes server, managed by Rancher

- Rancher as multi-cluster management platform
  - https://rancher.cronn.de
- Kubernetes cluster
  - assigned to Wildcard domain *.rancher-k8s.cronn.de
  - 5 nodes as a kubernetes cluster

Let´s configure access for you_

- git clone https://github.com/chrisingenhaag/k8s-javaland
- Github as authentication provider - **Your username is?**
- Everybody gets a separate "namespace" for today

Preparation_

Kubernetes cli tools_

- Have kubectl installed - https://kubernetes.io/docs/tasks/tools/install-kubectl/
- Get Kubeconfig file from rancher and place it in ~/.kube/config
- Bash completion is a plus!

```
$ kubectl config set-context cronn-hetzner --namespace=[yournamespace]
```

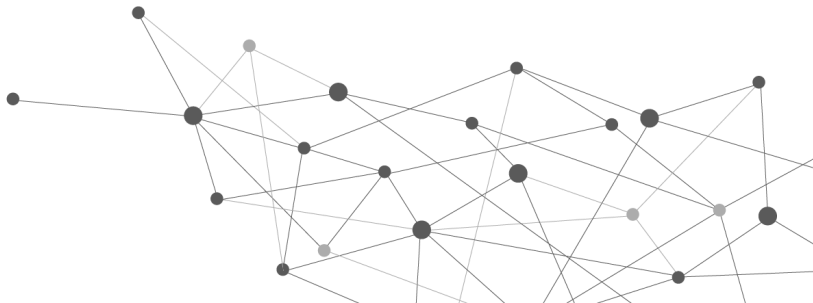
Docker_

- Have docker installed - https://docs.docker.com/install/
- Docker-Hub account for publishing application (optional)

Task_

- Check if your tooling work and your able to work with kubectl

kubernetes faststart_



Kubernetes fast start_

Kubernetes fast start_

Overview_

- Open source container orchestration
- First github commit in 2014
- Now in 2022 hosted by CNCF
  - First "graduated" project within CNCF
  - 100k commits, 3k contributors, ~500 releases
- Everything is an API
  - Definition via kubernetes manifests in YAML format
  - Create, change or delete objects and kubernetes does the rest for you
  - Basic components: Pod, Container, Service, Volume, Namespace, Ingress
  - Controllers: Deployment, StatefulSet, DaemonSet

Kubernetes fast start_

Overview_

- Open source container orchestration
- First github commit in 2014
- Now in 2022 hosted by CNCF
  - First "graduated" project within CNCF
  - 100k commits, 3k contributors, ~500 releases
- Everything is an API
  - Definition via kubernetes manifests in YAML format
  - Create, change or delete objects and kubernetes does the rest for you
  - Basic components: Pod, Container, Service, Volume, Namespace, Ingress
  - Controllers: Deployment, StatefulSet, DaemonSet

"Bring Dev and Ops closer together, but with clear responsibilities!"

Kubernetes fast start_

Overview_

- Open source container orchestration
- First github commit in 2014
- Now in 2022 hosted by CNCF
  - First "graduated" project within CNCF
  - 100k commits, 3k contributors, ~500 releases
- Everything is an API
  - Definition via kubernetes manifests in YAML format
  - Create, change or delete objects and kubernetes does the rest for you
  - Basic components: Pod, Container, Service, Volume, Namespace, Ingress
  - Controllers: Deployment, StatefulSet, DaemonSet

"Bring Dev and Ops closer together, but with clear responsibilities!"

https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.21/

https://rancher.com/docs/rancher/v2.6/en/overview/architecture/#rancher

Kubernetes fast start_

Cluster overview_

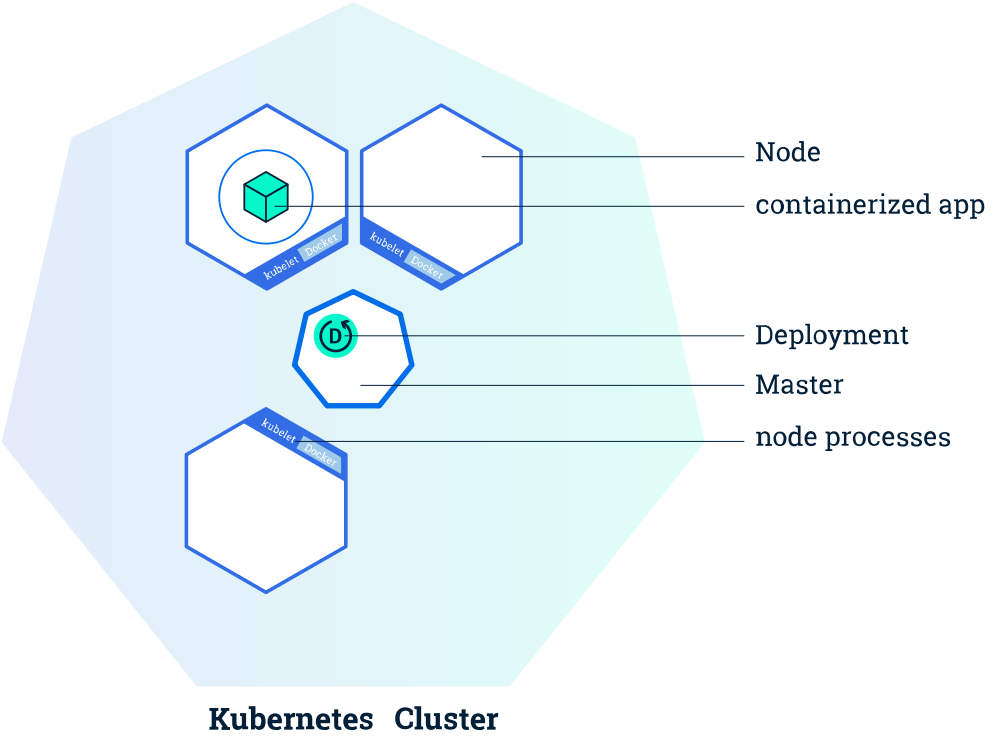


image source: kubernetes.io

Kubernetes fast start_

Node overview_

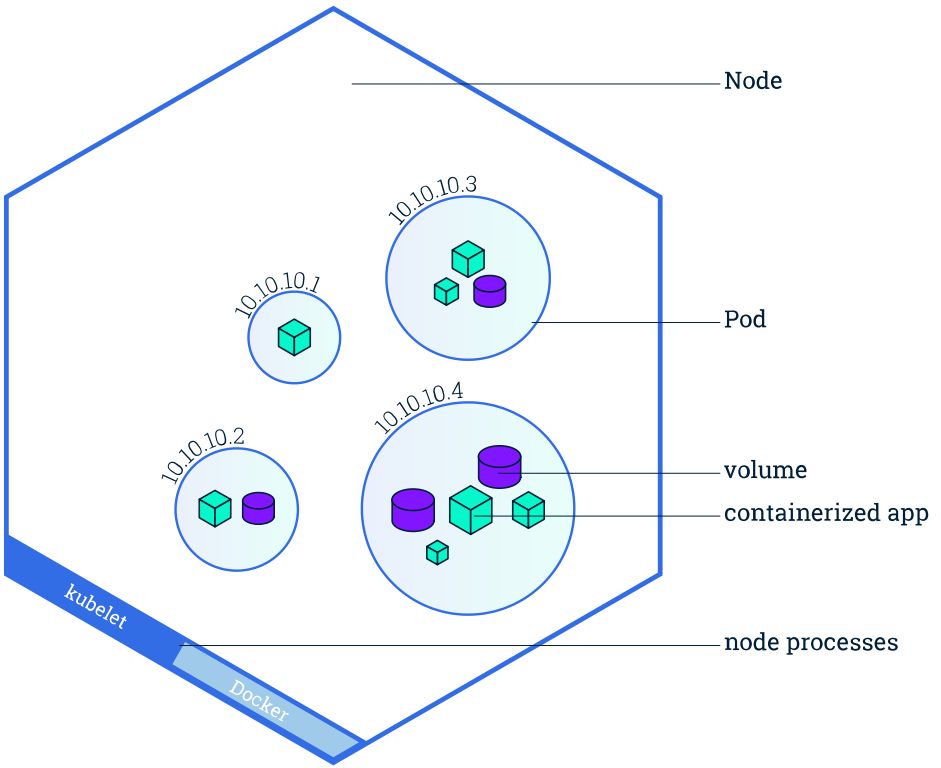
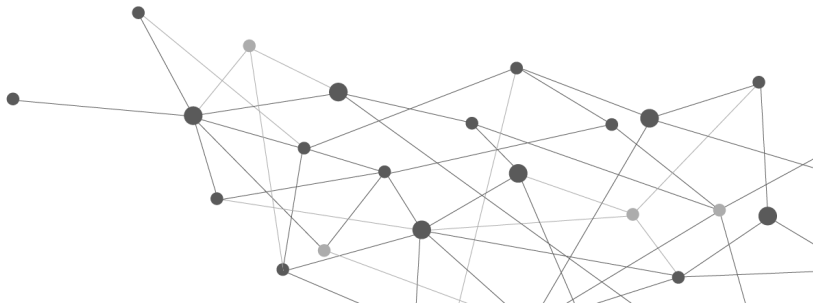


image source: kubernetes.io

Sample Java application_



Sample Java application_

Our plan_

- Demo application
  - Simple spring-boot application
  - Including a test controller `/infos`
  - Including actuator lib for health-checks
  - Sourcecode available in the git repo
- Docker Image
  - image available at `ingenhaag/demo-app:latest`
- Kubernetes
  - Run our application

Sample Java application_

Simple dockerizing a Java application_

```
FROM openjdk:11-jre-slim

COPY build/lib/demo-app-1.0.jar /app/demo-app.jar

ENTRYPOINT ["java","-jar","/app/demo-app.jar"]
```

Sample Java application_

Advanced dockerizing a Java application including security considerations_

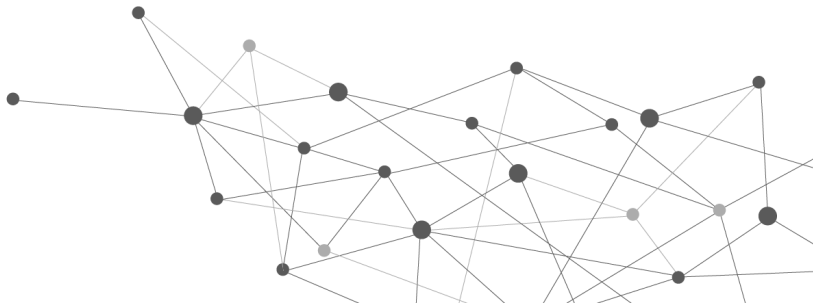
```
FROM openjdk:11-jre-slim

RUN useradd -s /bin/false --no-log-init -r -u 1001 appuser
RUN mkdir -p /app && \
    chown -R appuser:appuser /app

COPY --chown=appuser:appuser build/dependency/BOOT-INF/lib /app/lib
COPY --chown=appuser:appuser build/dependency/META-INF /app/META-INF
COPY --chown=appuser:appuser build/dependency/BOOT-INF/classes /app

USER 1001

EXPOSE 8080
ENV JAVA_OPTS="-XX:MaxRAMPercentage=50 -XshowSettings:vm"

ENTRYPOINT ["/bin/bash", "-c","/usr/local/openjdk-11/bin/java $JAVA_OPTS -cp app:app/lib/* de.cronn.k8s.der
```

Sample Java application_

Build, test and publish_

```
cd applications/demo-app
# build your application as jar
./gradlew clean assemble
# build your docker image
docker build . -t <your-docker-hub-name>/demo-app:latest
# test your built docker image
docker run -d --rm -p 8080:8080 <your-docker-hub-name>/demo-app:latest
# test your container
curl -i http://localhost:8080/infos
# push the image to docker hub
docker push <your-docker-hub-name>/demo-app:latest
```

running a pod / basic deployment_



Run your first container_

Basic deployment for our demo-app container_

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: demo-app-deployment
  namespace: christian
  labels:
    app: demo-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: demo-app
  template:
    metadata:
      labels:
        app: demo-app
    spec:
      containers:
      - name: demo-app
        image: ingenhaag/demo-app:latest
        ports:
        - containerPort: 8080
        securityContext:
          runAsNonRoot: true
          runAsUser: 1001
```

Run your first container_

```
cd material/01_basic_deployments
# create demo-app deployment
kubectl apply -f deployment-demo-app.yaml
# OR kubectl apply -f deployment-demo-app.yaml --namespace christian
# see deployment status
kubectl get all
# see pod details
kubectl describe pod demo-app-deployment-XXXXXXXX-YYY
# port forward to pod
kubectl port-forward demo-app-deployment-XXXXXXXX-YYY 8080
# test
curl -i http://localhost:8080/infos
```

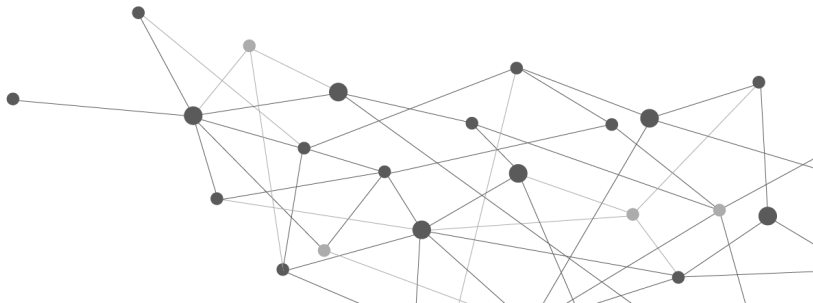
Keep in mind: Pod/Container itself is not accessible from outside cluster

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

Task_

- Go through the steps above

manage traffic_



Manage traffic_

Keep in mind, that pods ..._

Manage traffic_

Keep in mind, that pods ..._

- may crash at any time.
- can be rescheduled to another node at any time.
- can run scaled with n replicas in parallel.
- get random ip addresses.

Manage traffic_

Keep in mind, that pods ..._

- may crash at any time.
- can be rescheduled to another node at any time.
- can run scaled with n replicas in parallel.
- get random ip addresses.

Conclusion_

- Pods are **volatile**
- No consistent reachability possible

Manage traffic_

Services in general_

- Kubernetes services provide a consistent addressability for pods
- Services are available in different types:
  - NodePort
  - ClusterIP
  - LoadBalancer
  - ExternalName
- Additionally, **"headless services"** provide a special, advanced way to address a group of pods without a strict coupling

Manage traffic_

Pods and services_

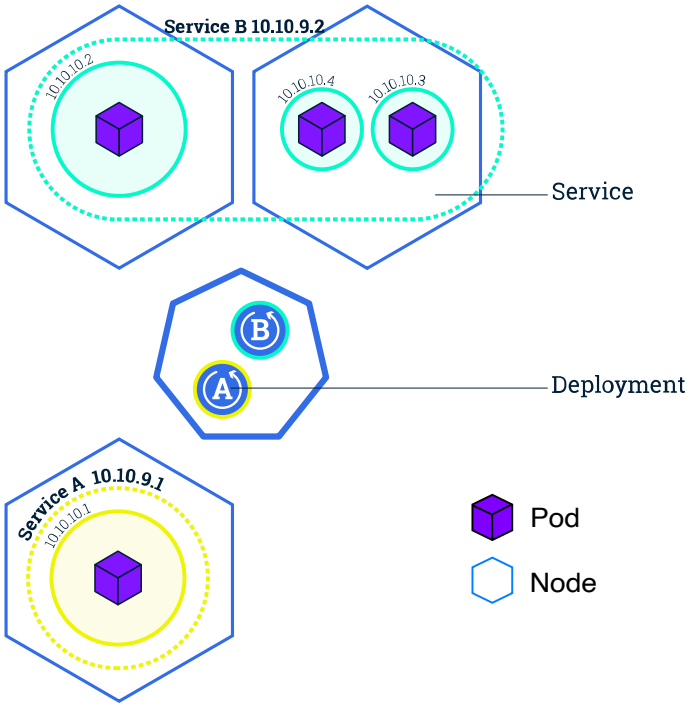


image source: kubernetes.io

Manage traffic_

Service type ClusterIP_

- **Default** service type
- Creates additional, virtual and internal ip for service which load balances traffic to selected pods
- Only cluster-internal availability

```
kind: Service
apiVersion: v1
metadata:
  name: demo-app-service-clusterip
spec:
  type: ClusterIP
  selector:
    app: demo-app
  ports:
  - name: http
    protocol: TCP
    port: 8080
    targetPort: 8080
```

Manage traffic_

Service type ClusterIP_

- Every services get´s own dns entry
- Easy, consistent service-to-service-communication

```
# setup busybox container for testing inside cluster
cd material/01_basic_deployments
kubectl apply -f busybox.yaml
# set up clusterip service
cd ../02_manage_traffic
kubectl apply -f service-clusterip.yaml
# Login to your pod
kubectl exec -it busybox-deployment-XXXXXXXXX-YYYY /bin/sh
# Test working dns resolution
# pattern: servicename[.namespace[.svc[.cluster.local]]]
# example: demo-app-service-clusterip.christian.svc.cluster.local
nslookup demo-app-service-clusterip.christian.svc.cluster.local
# Test http get for your service
wget -qO- demo-app-service-clusterip.christian.svc.cluster.local:8080/infos
```

Manage traffic_

Service type ClusterIP_

- Every services get´s own dns entry
- Easy, consistent service-to-service-communication

```
# setup busybox container for testing inside cluster
cd material/01_basic_deployments
kubectl apply -f busybox.yaml
# set up clusterip service
cd ../02_manage_traffic
kubectl apply -f service-clusterip.yaml
# Login to your pod
kubectl exec -it busybox-deployment-XXXXXXXXX-YYYY /bin/sh
# Test working dns resolution
# pattern: servicename[.namespace[.svc[.cluster.local]]]
# example: demo-app-service-clusterip.christian.svc.cluster.local
nslookup demo-app-service-clusterip.christian.svc.cluster.local
# Test http get for your service
wget -qO- demo-app-service-clusterip.christian.svc.cluster.local:8080/infos
```

Task_

- Create a ClusterIP service for your demo-app-deployment
- Create a deployment with image busybox:1.28
- Create and login to busybox Deployment and check your ClusterIP service

Manage traffic_

Service type NodePort_

- Opens external, random port on all nodes
- Configurable by cluster admin, in our case 30000-32767

```
kind: Service
apiVersion: v1
metadata:
  name: demo-service-nodeport
spec:
  type: NodePort
  selector:
    app: demo-app
  ports:
  - name: http
    protocol: TCP
    port: 8080
    targetPort: 8080
```

Manage traffic_

Service type LoadBalancer_

- Assigns external ip from pool of ips (not node ip)
- Traffic to this ip will be routed to pods behind service
- No out of the box feature in private clusters, you´ll need something like metallb
- Common in/Integrated with aws, gke or aks clusters

```
kind: Service
apiVersion: v1
metadata:
  name: demo-app-service-loadbalancer
spec:
  type: LoadBalancer
  selector:
    app: demo-app
  ports:
  - name: http
    protocol: TCP
    port: 8080
    targetPort: 8080
```

Manage traffic_

Service type ExternalName_

- Is able to point to a cluster-external dns name
- Used to simplyfy address across environments as servicename can be the same everywhere

Examples:

- external database
- some external api

```
kind: Service
apiVersion: v1
metadata:
  name: external-db-service
spec:
  type: ExternalName
  externalName: my.database.example.com
```

Manage traffic_

Headless service_

- "Freedom to do discovery their own way"
- Service doesn´t get own ip
- Kubernetes internal dns returns cname records for each pod behind service

```
kind: Service
apiVersion: v1
metadata:
  name: demo-app-service-headless
spec:
  type: ClusterIP
  clusterIP: None #!!!
  selector:
    app: demo-app
  ports:
  - name: http
    protocol: TCP
    port: 8080
    targetPort: 8080
```

Manage traffic_

Multiple ports_

- Of course it is possible to define multiple ports per service
- Pods may offer multiple ports as well
- For Example: spring-boot management port

```
kind: Service
apiVersion: v1
metadata:
  name: demo-app-service-clusterip
spec:
  type: ClusterIP
  selector:
    app: nginx
  ports:
  - name: http
    protocol: TCP
    port: 8080
    targetPort: 8080
  - name: http-management
    protocol: TCP
    port: 8081
    targetPort: 8081
```

Manage traffic_

Ingress-Controller_

Services, Services and Services but ...

But how are we able to offer our website or service via a domain?

Manage traffic_

Ingress-Controller_

Services, Services and Services but ...

But how are we able to offer our website or service via a domain?

some ingress-controllers implementations

- nginx-ingress - https://kubernetes.github.io/ingress-nginx/
- traefik - https://traefik.io/
- and many more

Manage traffic_

Ingress_

- Let's create a basic ingress for our nginx-deployment

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: demo-app-ingress
  namespace: christian
spec:
  rules:
    - host: christian.rancher-k8s.cronn.de
      http:
        paths:
          - pathType: Prefix
            path: /
            backend:
              service:
                name: demo-app-service-clusterip
                port:
                  number: 8080
```

- We can now expose one port of the service to the outside world
- Even if the service has multiple ports (e.g. internal management port)

Manage traffic_

Ingress_

- Keep in mind rules and paths objects are arrays[]! so ...

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: demo-app-ingress-multiple-backends
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
    - host: christian.rancher-k8s.cronn.de
      http:
        paths:
          - path: /app
            pathType: Prefix
            backend:
              service:
                name: serviceA
                port:
                  number: 8080
          - path: /api
            pathType: Prefix
            backend:
              service:
                name: serviceB
                port:
```

see https://kubernetes.github.io/ingress-nginx/user-guide/nginx-configuration/annotations/

Manage traffic_

Ingress_

Ingress offers a set of functionality for abstraction from your serivce:

- Authentication (Basic, OAuth, Client certs)
- Rewriting paths
- TLS Termination
- Session affinity by cookie
- Metrics

Manage traffic_

Ingress_

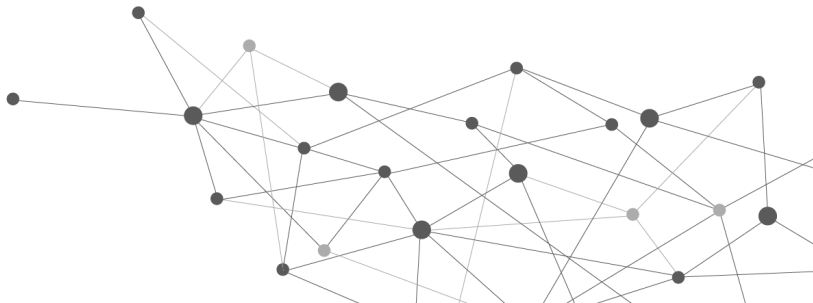
Ingress offers a set of functionality for abstraction from your serivce:

- Authentication (Basic, OAuth, Client certs)
- Rewriting paths
- TLS Termination
- Session affinity by cookie
- Metrics

Task_

- Create an ingress for your demo-app-deployment pointing the the clusterip-service
- The ingress should listen on host yourname.rancher-k8s.cronn.de
- The ingress should listen on subpath /foo/bar/ which rewrites to / on demo-app side
- Look in material/03_ingress for input

configurations_



Configuration_

Fine, now you have a working Deployment.

But how to configure your application?

Configuration_

Fine, now you have a working Deployment.

But how to configure your application?

- Bake in config in Docker image?
- Docker Image for each environment?
- Docker Image for each customer?

Configuration_

Fine, now you have a working Deployment.

But how to configure your application?

- Bake in config in Docker image?
- Docker Image for each environment?
- Docker Image for each customer?

No, For this we´re able to use:

- Configmaps
- Secrets

Configuration_

ConfigMaps_

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: configmap-envvalues
data:
  CRONN_JAVALAND: '2022'
  CRONN_WORKSHOP: 'Kubernetes'
```

Integration as environment variables

- Environment variables become part of your configuration in docker/kubernetes based applications

Configuration_

ConfigMaps_

Use all keys of the configmap as environment variables

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: demo-app-deployment
  labels:
    app: demo-app
spec:
  #...
  template:
    #...
    spec:
      containers:
      - name: demo-app
        #...
        envFrom:
        - configMapRef:
            name: configmap-envvalues
```

Configuration_

ConfigMaps_

Use some of the configmap keys as environment variables

```
apiVersion: v1
kind: Deployment
spec:
  #...
  template:
    #...
    spec:
      containers:
      - name: demo-app
        #...
        env:
        - name: CRONN_JAVALAND
          valueFrom:
            configMapKeyRef:
              name: configmap-envvalues
              key: CRONN_JAVALAND
```

Configuration_

ConfigMaps_

Complete files in ConfigMaps

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: configmap-files
data:
  application.properties: |-
    enemies=aliens
    lives=3
    enemies.cheat=true
  someother.config: |-
    color.good=purple
    color.bad=yellow
```

- YAML is a b!#&% - see https://yaml.org/refcard.html
  - | or |+ or |- or >+ or >-

Configuration_

ConfigMaps_

Use ConfigMap as volumes

```
apiVersion: v1
kind: Deployment
spec:
  #...
  template:
    #...
    spec:
      containers:
      - name: demo-app
        #...
        volumeMounts:
        - name: config-volume
          mountPath: /opt/config
      volumes:
      - name: config-volume
        configMap:
          name: configmap-files
```

Configuration_

ConfigMaps - Task_

Task for env vars_

- Create a config map with simple values
- Extend your deployment to use all values as environment variables
- Apply changes, login to your container and check env vars

Task for filesystem_

- Create a config map with text file content
- Extend your deployment to mount all content values in your Pods filesystem
- Apply changes, login to your container and check env vars

Hint: look in material/04_configuration for input

Configuration_

Secrets_

- More or less the same as configmaps
- But not everybody will be able to see secrets
- Credentials should be stored in secrets

Configuration_

Secrets_

- More or less the same as configmaps
- But not everybody will be able to see secrets
- Credentials should be stored in secrets

Base64 encoded values_

```
apiVersion: v1
kind: Secret
metadata:
  name: example-secret
type: Opaque
data:
  username: Y2hyaXN0aWFu
  password: MWYyZDFlMmU2N2Rm
```

```
$ echo -n "christian" | base64
Y2hyaXN0aWFu
```

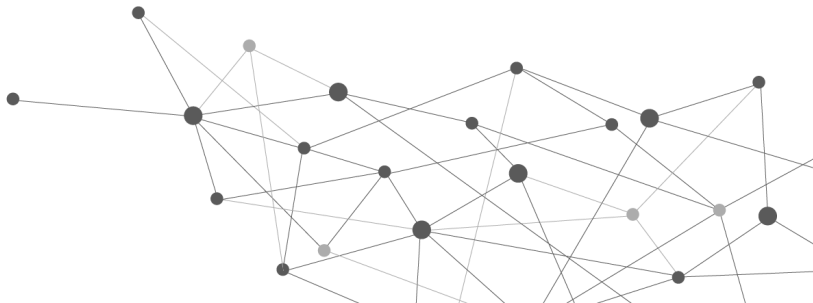
Configuration_

Secrets_

StringData values_

```
apiVersion: v1
kind: Secret
metadata:
  name: example-secret-stringdata
type: Opaque
stringData:
  username: plainusername
  password: plainpassword
```

advanced deployments_



Advanced Deployments_

Health checks_

- Liveness Probes - Indicates that the pod is running and reachable
- Readiness Probes - Indicates that kubernetes can pass traffic to Pod

```
# excerpt deployment
spec:
  template:
    spec:
      containers:
      - image: someimage:latest
        readinessProbe: #or livenessProbe
          httpGet:
            path: /
            port: 8080
            scheme: HTTP
          initialDelaySeconds: 30
          periodSeconds: 15
          timeoutSeconds: 1
          failureThreshold: 3
          successThreshold: 1
```

Advanced Deployments_

Resources_

- Each Worker-Node is able to run 110 Pods per default
- But would you run 110 Java Applications in 4 CPU machine?

Advanced Deployments_

Resources_

- Each Worker-Node is able to run 110 Pods per default
- But would you run 110 Java Applications in 4 CPU machine?

Limit resources_

- CPU - (compressable) - 1, 0.5, 1000m, 250m
- Memory - (uncompressable) - 1Gi, 1024Mi, 256Mi

Advanced Deployments_

Resources_

- Each Worker-Node is able to run 110 Pods per default
- But would you run 110 Java Applications in 4 CPU machine?

Limit resources_

- CPU - (compressable) - 1, 0.5, 1000m, 250m
- Memory - (uncompressable) - 1Gi, 1024Mi, 256Mi

```
# excerpt deployment
spec:
  template:
    spec:
      containers:
      - image: someimage:latest
        resources:
          limits:
            cpu: 1000m
            memory: 512Mi
          requests:
            cpu: 250m
            memory: 256Mi
```

Advanced Deployments_

Task_

- Extend your Deployment to use spring-actuator health-endpoint for readiness check
- Extend your Deployment to limit container resources
  - 100m / 1000m cpu
  - 256Mi / 512Mi memory

Some Questions / Discussion_

- When is your application healthy regarding e.g. Database availability?
- What startup time do you expect with 100m cpu?
- What happens if your pod wants to use more than requested resources?

Advanced Deployments_

Some notes on java in k8s environments_

CPU resources_

- Java needs more cpu at startup time
- Make use of requests/limits

Memory resources_

- Heap size, Java from 11 respects container cgroup limits out of the box
- Tweaking Heap size in respect of resource limits
  - `-XX:MaxRAMPercentage=50`
- Normally java is not designed to give back memory to os (default)
- Which garbage collector with cpu resource limits 1000m?
  - Multithreaded garbage collection with just one cpu?
  - May be use something like `-XX:+UseSerialGC`
- Fix memory request == limit?

Advanced Deployments_

Deployment Strategies_

Types_

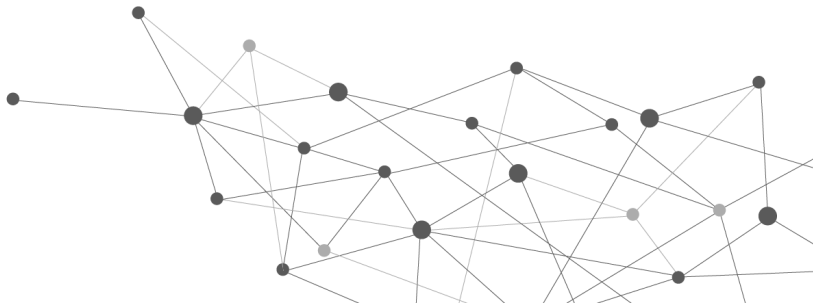
- Rolling - Start new, then stop old
- Recreate - Stop old, then start new

```
spec:
  strategy:
    type: Rolling
    rollingParams:
      intervalSeconds: 1
      maxSurge: 25%
      maxUnavailable: 25%
      timeoutSeconds: 600
      updatePeriodSeconds: 1
```

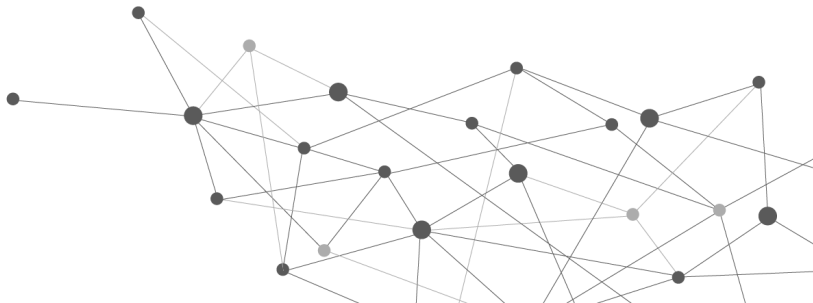
```
spec:
  strategy:
    type: Recreate
```

Thank you!_

cronn.de_



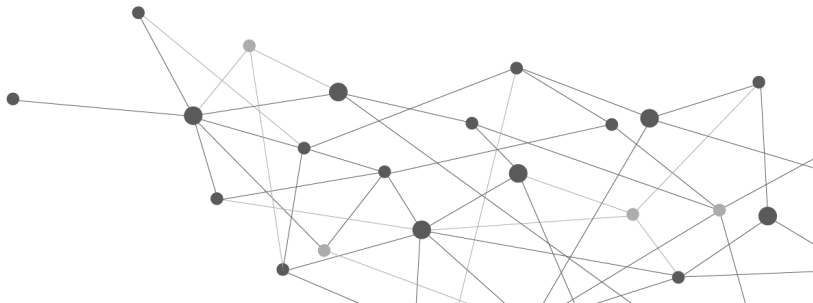
Backup Slides_



Backup Topics_

- Persistence
- Horizontal Pod Autoscaling
- Secure ingress (letsencrypt)
- Security
  - Pod security policies
  - Network policies

persistence_



Persistence_

Introduction_

- Everything a Pod stores to its local filesystem is lost after restart
- Kubernetes provides a volume concept for using persistent storage
- Remember: Pods are volatile, can be stopped and rescheduled at any time

We differentiate between_

- Static volume creation and assignment
  - Some Administrator creates volumes and you´re able to use them
  - **Persistent Volumes**
- Dynamic volume creation and assignment
  - Cluster creates volumes dynamically if no volume matches claim
  - E.g. request for Persistent Volume
  - **Persistent Volume Claims**

Persistence_

Access Modes_

- ReadWriteOnce – the volume can be mounted as read-write by a single node
- ReadOnlyMany – the volume can be mounted read-only by many nodes
- ReadWriteMany – the volume can be mounted as read-write by many nodes

Some available Types_

- Emptydir
- Longhorn (our default)
- NFS
- CephFS
- AWSElasticBlockStore
- Azure Disk
- GCEPersistentDisk

Persistence_

PersistentVolume Example_

Created by an administrator.

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0003
spec:
  capacity:
    storage: 100Mi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Delete
  storageClassName: longhorn
```

A **storage class** helps an administrator to describe the storage he offers. Examples could be `ssd` or `high-redundant-but-slow`.

Persistence_

PersistentVolumeClaim Example_

Created by yourself alongside with your application.

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: demo-app-pvc
spec:
  accessModes:
    - ReadWriteOnce
  volumeMode: Filesystem
  resources:
    requests:
      storage: 100Mi
  storageClassName: longhorn
```

Persistence_

Use Claims as Volumes_

```
apiVersion: v1
kind: Deployment
spec:
  # ...
  template:
    # ...
    spec:
      securityContext:
        fsGroup: 1001
      containers:
      - name: demo-app
        # ...
        volumeMounts:
        - mountPath: "/opt/data"
          name: data
      volumes:
      - name: data
        persistentVolumeClaim:
          claimName: demo-app-pvc
```

Persistence_

PersistentVolumeClaim_

```
cd 09_backup_material/persistence
# deploy failing pvc-deployment
kubectl apply -f deployment-w-pvc.yaml
# see if it´s working
kubectl describe pod demo-app-deployment-xxxxxxxxxx-yyyyy
# delete deployment and pvc
kubectl delete -f deployment-w-pvc.yaml
```

Let´s see what happened_

- 1 Volume for 3 Replicas? ReadWriteMany needed for scaling
- Volumes get deleted if PersistentVolumeClaim gets deleted!
- Common, but (bad) confusing practice to use together with Deployments
  - Even official MySQL Kubernetes charts use deployments
- Wasn´t there the thing about StatefulSets for stateful applications?

Persistence_

StatefulSets_

Regarding volumes:

- You can define a property volumeClaimTemplates
- Each replica gets it´s own PVC
- Volumes don´t get deleted if you delete your StatefulSet
- Much easier to manage Pods with their PVC (especially when scaling)

StatefulSets in General

- Provide a persistent pod identifier
- Ordered deployments
- Ordered updates
- Require a headless service

Persistence_

StatefulSet Example_

```
kind: StatefulSet
# ...
spec:
  # ...
  template:
    # ...
    spec:
      containers:
      - name: demo-app
        image: ingenhaag/demo-app:latest
        volumeMounts:
        - name: data
          mountPath: /opt/data
  volumeClaimTemplates:
  - metadata:
      name: data
    spec:
      accessModes: [ "ReadWriteOnce" ]
      storageClassName: "longhorn"
      resources:
        requests:
          storage: 100Mi
```

Persistence_

StatefulSets with Volumes_

Task_

- Create a StatefulSet as a copy of former demo-app-deployment
- Delete and recrate Statefulset and check volume retention and reusage
- Scale up to 3 replicas and scale down back to 1 replica

HorizontalPodAutoscaler_

Cpu load based autoscaling_

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: deployment-hpa
spec:
  minReplicas: 2
  maxReplicas: 5
  targetCPUUtilizationPercentage: 80
  scaleTargetRef:
    apiVersion: v1
    kind: Deployment
    name: deplyoment-name
```

Watched load percentage refers to **requested** CPU!

Task_

- Extend your java deployment with a hpa scaling up to 3 replicas

Secure Ingress_

SSL Certficates with Letsencrypt_

- Requires certmanager installed on cluster
- Listens on ingress Objects with specific annotations
- Needs some tls informations in your ingress object

Secure Ingress_

SSL Certficates with Letsencrypt_

- Requires certmanager installed on cluster
- Listens on ingress Objects with specific annotations
- Needs some tls informations in your ingress object

```
# ...
  annotations:
    kubernetes.io/tls-acme: "true"
    cert-manager.io/cluster-issuer: letsencrypt-prod # CRD created with certmanager
  # ...
  tls:
  - hosts:
    - christian.rancher-k8s.cronn.de # has to match with ingress hostname above
    secretName: demo-app-tls # will be created by certmanager
```

Task_

- Apply config to your java application ingress

see https://cert-manager.io/v1.1-docs/ for details

Security_

Pod Security Policies_

- Rule #1 - Don´t run as root VS Docker hub images
- Fine grained options
  - ReadOnlyFilesystem
  - Allows volume types
  - Linux capabilities - Link

Network Policies_

- Ability to restrict cluster internal communication
- "Microservices should not be able to communicate with each other in an unrestricted way"
- Leveraged with services meshes like istio

Example - see Kubernetes docs