



# Vertiefung OpenShift

Tobias Derksen & Norbert Schneider

---



# Agenda

- Wrap-Up Grundlagen
- CI/CD Automation
- Hochverfügbarkeit
- Security
- Best Practices

# Wrap Up Grundlagen

---

# Was haben wir gelernt?

- Cluster Aufbau
- WebConsole & GUI
- Persistent Storage Framework
- OpenShift 4

Fragen?

# CI/CD Automation

---

# Continuous Integration

- Build application after each push
- Execute automated tests after each build
- Execute static analysis after each build
- Report any errors or warnings
- Store created artifacts

# Continuous Delivery

- Create deployable image after each build
- Store image in registry
- Deploy image to development stage after each image build
- Optional: Perform image security scan
- Optional: Run smoke tests after deployment
- Optional: Run automated integration tests
- Optional: Promote image to next stage after successful tests

# What OpenShift provides

- Image Builds including build triggers
- Jenkins Integration
- Jenkins Pipeline Integration (Deprecated)
- Jenkins Plugins for customized integrations



✓ Build #3  
5 minutes ago  
[View Log](#)





# Die neuen OpenShift Pipelines

- Pipeline Operator
- Pipelines werden über Objekte abgebildet
- Objekte werden lose gekoppelt
- Wiederverwendbare Einzelteile

<https://github.com/openshift/pipelines-tutorial>

# Security

---

# Übersicht

- Role based access control (RBAC)
- Security Context Constraints (SCC)
- ~~PodSecurityPolicy (PSP)~~

# Rollen & Rechte

- Cluster Rollen
- Projekt Rollen
- Rechte bestehen aus Verb + Objekttype (Beispiel: get projects)
- Rechte eines Accounts = Summe aller erlaubten Aktionen
- Serviceaccounts

## Cluster Rollen:

- cluster-admin
- cluster-reader
- self-provisioner

## Projekt Rollen:

- admin
- edit
- view

# Security Context Constraints (SCC)

- Kontrolliert die Rechte eines Pods
- Ohne SCC werden erweiterte Rechte vom Scheduler zurückgewiesen
- Erlaubt Pods:
  - Zugriff auf Host Dateisystem
  - Zugriff auf Host Netzwerk
  - Starten als spezifischer User, bzw Root
  - Setzen von SELinux context
  - Erweiterte Möglichkeiten mit Gruppen
  - Erlauben bestimmter Linux Capabilities

## Was man **NIEMALS** tun sollte ...

- Rechte an den default Service Account geben
- SCC an den default Service Account geben
- "privileged" SCC vergeben
- *Container als root laufen lassen weil man zu faul ist es richtig zu machen*

*oc adm policy add-scc-to-user privileged -z default*

# Best Practices - Security

- SELinux nicht deaktivieren
- Cluster Nodes nur intern (über Bastion) erreichbar
- non-root Container
- Container Scanning nach Sicherheitslücken
- Blocken von offenen Registries (Docker Hub, Quay.io)
- EgressIP für Firewalls / Network Policies
- Traffic Encryption (Service Mesh)
- Regelmäßige Updates im Cluster
- **Regelmäßige Updates der Base Images**

# OpenShift Ready Applications



# The cluster is your friend ... but it needs your help

- Cluster ensures a certain state  
*Tell the cluster the state you desire; how it gets there is not your problem.*
- Cluster needs to know what resources you need
- Cluster needs information about the state of the application to ensure it has the desired state

# Health Checks

- **Liveness Probe**

*Checks whether the container is alive*

If fail, container is restarted

- HTTP GET
- Shell command
- Open TCP ports

- **Readiness Probe**

*Checks whether the container is able to accept traffic*

If fail, container will not get any traffic from service layer

# Resource Allocation

- Resources are valuable and expensive
- Know what you need ... and tell the cluster
- Requests are guaranteed ... limits are not
- *If there is no node with enough resources, the pod will not start at all*

```
apiVersion: v1
kind: "Pod"
metadata:
  name: "test"
  labels:
    app: test
spec:
  containers:
    - image: mysql
      resources:
        requests:
          cpu: 500m
          memory: 1Gi
        limits:
          cpu: 2
          memory: 2Gi
```

# Failing is a totally valid option

- Expect that any pod is killed by kubernetes at any time
- Allow your container to fail ... as early as possible

Reasons why a pod is killed:

- Manual interaction (Admin, Developer, etc)
- Node failure or maintenance
- Network issues
- Pod / Container out-of-memory
- **Node out-of-memory**

## Noteworthy points

- Make your application **timezone aware**  
*by default container run in UTC*
- Avoid file system writes; keep it to “/tmp”
- Log messages to stdout and stderr
- Reduce dependencies; especially hard dependencies

# Deployments

---

# Basics

- Deployment
- DeploymentConfig
- Pod Name auto-generated
- Pod IP changes regularly
- Pods can get rescheduled somewhere else

# Rollout Strategy

Strategy	<b>Rolling</b>	<b>Recreate</b>
Effect	Create new pods and wait until they are ready, then kill old pods	Kill all old pods, wait until they are terminated and then create new pods
Useful for	Services, APIs, Stateless Deployments	Databases, Deployments with exclusive resources



# How do I configure an application?

- ConfigMaps
- Secret
- Environment
- AutoDiscovery (e.g. with Spring Boot kubernetes)

# ConfigMap usage

- Key-Value store
- Reference into environment
- Mount as files
- Overwrite files in container

```
apiVersion: v1
kind: "Pod"
metadata:
  name: "test"
spec:
  containers:
    - image: mysql
      env:
        - name: MYSQL_USER
          valueFrom:
            configMapKeyRef:
              name: test-db-credentials
              key: username
      volumeMounts:
        - name: config
          mountPath: /app/config
  volumes:
    - name: config
      configMap:
        name: test-db-config
```

# StatefulSets

---

# Basics

- Ensures deterministic Pod Names
- Ensures deterministic Pod creation order
- Pod Name is stable
- Pod IP can still change
- Pods get rescheduled last

# Update Strategy

Strategy	<b>RollingUpdate</b>	<b>OnDelete</b>
Effect	Terminate one pod, start a new one and wait until ready, then go over to the next pod	Don't do something, wait until a pod gets deleted, then make a new one
Useful for	Databases, clustered applications	Databases, In-Memory Cache cluster

# Troubleshooting

---

# Debug Applications

- Container Logs
- Centralized Logging Collector (EFK)
- Events
- Execute commands in container
- Start debug container