CI/CD mit Openshift

Thomas Herzog B.Sc / Phillip Wurm B.Sc

January 21, 2018

Contents

1 Problemstellung

Dieser Abschnitt behandelt die Problembeschreibung des Projekts *CI/CD mit Openshift*. Nachdem *Cloud*-Lösungen wie Microsoft Azure, AWS oder Openshift immer mehr an Bedeutung gewinnen, soll mit diesem Projekt ein Prototyp implementiert werden, der eine CI/CD Umgebung in einer *PaaS (Platform as a Service)* Lösung wie Openshift realisiert.

Die zu implementierende CI/CD Umgebung soll einen *Build*-Server (Jenkins) und einen *Repository*-Manager (Nexus) beinhalten. Nexus soll als *Mirror* und Artefakt-*Repository* verwendet werden. Einerseits sollen die abhängigen Artefakte im Nexus zwischengespeichert werden, damit die Artefakte innerhalb des *Clusters* geladen werden und nicht aus dem Internet. Andererseits soll Nexus alle freigegebenen Artefakte in einem *Repository* verwalten.

Mit einer Jenkins *Pipeline*[1], die in Jenkins ausgeführt wird, soll eine Beispielanwendung in einem eigenen Docker *Container* gebaut, in Nexus hochgeladen und anschließend in Openshift eingespielt werden. Die Jenkins Pipeline soll als Openshift *Build*-Konfiguration angelegt werden. Als Basis für den Docker *Container*, in dem die Anwendung gebaut wird, soll ein Docker Image spezifiziert werden, das die *Build*-Umgebung (Gradle) definiert sowie eine *Build*-Konfiguration für das Bauen des Docker Images in Openshift.

Für die Beispielanwendung soll ein *Template* erstellt werden, dass die Infrastruktur für diese Anwendung in Openshift spezifiziert. Die in diesem *Template* zu spezifizierende *Build*-Konfiguration soll von der Jenkins *Pipeline* ausgelöst werden, die wiederum das *Deployment* der Anwendung auslösen soll, womit die neue Version in Openshift eingespielt werden soll.

2 Projektaufbau

Dieser Abschnitt behandelt den Aufbau des Projekts *CI/CD mit Openshift*. Die Quelltexte des Projekts wurden in mehreren Github *Repositories* organisiert, die folgend aufgelistet sind:

1. ***buildserver***[2] ist das *Repository*, das die Openshift *Templates* und Skripten für das Aufsetzen des *Build*-Servers beinhaltet.
2. ***service-jenkins***[3] ist das *Repository*, das die Ressourcen für das Bauen des Jenkins Docker Image und der Jenkins *Slave* Docker Images beinhaltet.
3. ***service-app***[4] ist das *Repository*, das die Quelltexte und die *Build*-Definition der Beispielanwendung beinhaltet.
4. ***appserver***[5] ist das *Repository*, das die Openshift *Templates* und Skripten für das Aufsetzen des Applikation Servers beinhaltet.

[1] `https://jenkins.io/doc/book/pipeline/`
[2] `https://github.com/OpenshiftCICD/buildserver`
[3] `https://github.com/OpenshiftCICD/service-jenkins`
[4] `https://github.com/OpenshiftCICD/service-app.git`
[5] `https://github.com/OpenshiftCICD/appserver`

Die Aufteilung in mehrere *Repositories* wurde eingeführt, da die verschiedenen *Repositories* Quelltexte und Ressourcen für verschiedene Anwendungszwecke beinhalten, die nicht zwangsweise zusammenhängend sind. Z.B. beinhaltet das *Repository service-jenkins* die Ressourcen zum Bauen eines Jenkins Docker Image über einem S2I *Build*, wobei das resultierende Jenkins Docker Image auch anderweitig verwendet werden kann. Daher sind die Ressourcen des *Repositories service-jenkins* nicht exklusiver Teil des *Build*-Servers und daher auch nicht im *Repsoitory buildserver* enthalten.

Es werden Github *Hooks* verwendet, um Openshift *Builds* auszulösen, wobei bei der Aufteilung auf mehrere *Repositories* Openshift *Builds* nur dann ausgelöst werden, wenn die Ressourcen, die auch im *Build* verwendet werden, geändert wurden. Bei der Verwendung von nur einem *Repository* würden alle *Builds* ausgelöst werden, was zu vielen unnötigen *Builds* führen würde.

3 Entwickler Setup

Dieser Abschnitt beschreibt das Aufsetzen einer lokalen Entwicklungsumgebung für die Entwicklung mit Openshift. Es wird davon ausgegangen, dass auf einem Linux System gearbeitet wird.

3.1 Docker Setup

Dieser Abschnitt behandelt, dass Einrichten von Docker für die Verwendung von Openshift. Es muss eine aktuelle Version von Docker installiert sein.

```
# 1. Define insecure registry, which is used by openshift
#    depending on your linux distribution
INSECURE_REGISTRY='--insecure-registry 172.30.0.0/16'

# 2. Reload the docker service
sudo systemctl daemon-reload

# 3.Restart the docker service
sudo systemctl restart docker
```

Die ungesicherte Docker *Registry* wird von Openshift dazu verwendet, um in Openshift Docker Images zu verwalten. Openshift lädt sich die verwendeten externen Docker Images nur einmal in die lokale Docker *Registry* und verwendet dann ausschließlich diese Images. Die in Openshift gebauten Docker Images werden ebenfalls in der lokalen Docker *Registry* verwaltet.

3.2 Firewall Setup

Dieser Abschnitt behandelt das Einrichten der Firewall für die Verwendung von Openshift. Ohne die folgenden Firewall Einstellungen kann in Openshift nicht auf das Internet zugegriffen werden.

```
# 1. Check docker bridge subnet
docker network inspect
        -f "{{range .IPAM.Config }}{{ .Subnet }}{{end}}" bridge

# 2. Create new firewall zone
firewall-cmd --permanent --new-zone dockerc

# 3. Add docker bridge network ass source
#    The network address, is the one we got at #1
firewall-cmd --permanent --zone dockerc --add-source 172.17.0.0/16

# 4. Add all ports docker and openshift needs
firewall-cmd --permanent --zone dockerc --add-port 8443/tcp
firewall-cmd --permanent --zone dockerc --add-port 53/udp
firewall-cmd --permanent --zone dockerc --add-port 8053/udp

# 5. Reload the firewall rules
firewall-cmd --reload
```

3.3 Openshift Setup

Dieser Abschnitt beschreibt das Einrichten des lokalen Oepnshift *Clusters*. Es werden folgende Ressourcen benötigt, die aus dem Internet heruntergeladen werden können.

1. ***Openshift Client Tools***[6] ist das Linux Paket, mit dem der lokale *Cluster* erstellt werden kann.

2. ***openshift-client-wrapper***[7] ist ein Github *Repository* das ein Shell-Skript zur Verfügung stellt, welches das Arbeiten mit oc erleichtert.

Das *oc* Binary sowie das Skript *oc-cluster-wrapper* müssen in den *PATH* mitaufgenommen werden. Das Skript *oc-cluster-wrapper* verwendet das *oc* Binary, das mit *oc* über den *PATH* angesprochen werden kann.

[6] https://developers.redhat.com/products/openshift/download/
[7] https://github.com/openshift-evangelists/oc-cluster-wrapper/releases/tag/0.9.3

Die folgenden *Shell*-Kommandos, die von *oc-cluster-wrapper* bereitgestellt werden, zeigen wie der Lebenszyklus des *Clusters* gesteuert werden kann. Es wird *oc-cluster-wrapper* anstatt *oc* selbst verwendet, da bei `oc cluster down` der *Cluster* vollständig gelöscht wird.

```
# Create or start persistent profile for local cluster named 'ci'
oc-cluster-wrapper up ci

# Stop the current running cluster (assume profile is ci)
oc-cluster-wrapper down [ci]

# Delete profile ci and all related cluster data
oc-cluster-wrapper destroy ci
```

Wenn beim erneuten Starten eines bestehenden *Cluster* Profils folgende Fehlermeldung auftritt, dann liegt es daran, dass die Konfiguration nicht mehr gültig, da der *Cluster* gestartet wurde, wenn sich der Rechner in einem anderen Netz befindet als zuvor.

```
# Command built by oc-cluster-wrapper
oc cluster up --version v3.5.5.31
--image registry.access.redhat.com/openshift3/ose
--public-hostname 127.0.0.1
--routing-suffix apps.127.0.0.1.nip.io
--host-data-dir /home/het/.oc/profiles/ci/data
--host-config-dir /home/het/.oc/profiles/ci/config
--host-pv-dir /home/het/.oc/profiles/ci/pv -
-use-existing-config -e TZ=CET
-- Checking OpenShift client ... OK

...

# Resulting error, because the cluster gets startet,
# when computer is in a different net.
Finding server IP ...
Using 10.29.18.80 as the server IP
-- Starting OpenShift container ... FAIL
Error: Docker run error rc=2
Details:
Image: registry.access.redhat.com/openshift3/ose:v3.5.5.31
Entrypoint: [/bin/bash]
Command: [-c for name in 10.29.18.80 het.linux.gepardec.com; kma
do ls /var/lib/origin/openshift.local.config/node-$name &> /dev/null
&& echo $name && break; done]
```

Kopieren Sie das *Shell*-Kommando ***oc cluster up ...*** und entfernen Sie das Argument ***–use-existing-config***. Ohne das Argument ***–use-existing-config*** wird eine neue Konfiguration erstellt.

3.4 Ultrahook Setup

Dieser Abschnitt behandelt das Einrichten von Ultrahook, das ein externer Service und eine lokale Applikation ist, mit der auf *Localhost* auf *Webhook* reagiert werden kann. Ohne Ultrahook können die *Hooks* nicht lokal getestet werden. Die folgenden zwei Punkte beschreiben das Einrichten von Ultrahook.

1. Registrieren eines *Webhook Namespace* auf `http://www.ultrahook.com/register`.
2. Starten der lokalen Anwendung mit folgenden *Shell*-Kommandos
   `ultrahook -k <API_KEY> github <OPENSHIFT_HOOK_URL>`

Jetzt kann auf einer lokalen Maschine auf z.B. Github *Hooks* reagiert werden, wobei der *Webhook* die von Ultrahook auf der Konsole ausgegebene *Url* verwenden muss.

4 *Build Server*

Dieser Abschnitt behandelt die Infrastruktur des *Build*-Server in Openshift. Der *Build*-Server wird innerhalb eines Openshift Projekts organisiert.

4.1 *Templates*

Dieser Abschnitt behandelt die Openshift *Templates*, welche die Integration der einzelnen Services innerhalb der *Build*-Server Infrastruktur in Openshift spezifizieren. Die Openshift *Templates* beinhalten alle Definitionen wie z.B. *Build*-Konfigurationen und *Deployment*-Konfigurationen, die Aspekte der *Core Conepts*[8] von Kubernetes und Openshift sind.

Die folgende Auflistung beschreibt die implementierten *Templates*:

1. Im *Template* ***jenkins-slaves.yml*** werden die für Jenkins zur Verfügung gestellten *Slave-Container* spezifiziert.
2. Im *Template* ***jenkins.yml*** wird der Jenkins Service spezifiziert.
3. Im *Template* ***nexus.yml*** wird der Nexus3 Service spezifiziert.
4. Im *Template* ***pipeline.yml*** wird eine Openshift *Build*-Konfiguration Pipeline spezifiziert.

[8] `https://docs.openshift.com/container-platform/3.5/architecture/core_concepts/index.html`

4.2 Skripte

Dieser Abschnitt behandelt die Skripten, die für das Verwalten des *Clusters* verwendet werden. Mit der Applikation *oc* kann mit einem lokalen oder entfernten *Cluster* interagiert werden. Damit der *Build Server* einfach erstellt und gelöscht werden kann, sind für die einzelnen Services und für den *Build Server* selbst Skripte erstellt worden, die alle nötigen Funktionalitäten beinhalten.

Auf dem Level der Skripten wird eine Datei namens *.openshift-env* und *.openshift-secret-env* erwartet. Die Datei *.openshift-env* definiert Umgebungsvariablen wie z.B. Service und *Secret* Namen, die über mehrere Skripten verwendet werden. Die Datei *.openshift-secret-env* definiert Umgebungsvariablen, welche die Passwörter für die *Secrets* definieren. Die folgende Auflistung beschreibt die implementierten *Skripte*:

1. Im *Skript* ***openshift-jenkins.sh*** sind alle Jenkins Service und Jenkins *Slave* spezifischen Funktionen implementiert.

2. Im *Skript* ***openshift-nexus.sh*** sind alle Jenkins Service und Jenkins *Slave* spezifischen Funktionen implementiert.

3. Im *Skript* ***openshift-buildserver.sh*** sind alle Funktionalitäten für das Verwalten des *Build Servers* implementiert.

4. Im *Skript* ***openshift-secrets.sh*** sind alle Funktionalitäten für das Verwalten von *Secrets* implementiert. Siehe Abschnitt 6 für eine genauere Beschreibung der verwendeten *Secrets*.

4.3 *Update* Szenarien

Dieser Abschnitt behandelt die *Update*-Szenarien für den *Build Server*. Es gibt folgende drei Szenarien für das Updaten des *Build Servers*:

1. Bei ***Änderung der Quelltexte*** wie *S2I Builds*[9] oder *Dockerfiles*, muss der Docker *Container* neu gebaut und der Service neu eingespielt werden.

2. Bei ***Änderung der Konfigurationen***, muss der Service gegebenenfalls neu gebaut und eingespielt werden.

3. Bei ***Änderung der Docker Images***, die Services beinhalten oder Basisimages darstellen, muss der Service neu gebaut und eingespielt werden.

Diese drei *Update*-Szenarien können einfach in Openshift abgebildet werden, da Openshift alle nötigen *Trigger*-Mechanismen zur Verfügung stellt, damit diese Änderung eingespielt werden können.

[9] `https://github.com/openshift/source-to-image`

Openshift erlaubt es bei *Deployment*-Konfigurationen und *Build*-Konfigurationen *Trigger*[10] zu definieren. Folgende *Trigger*-Typen werden bereitgestellt:

1. ***Github*** ist ein *Trigger*-Typ, der auf Github *Webhooks* regiert.
2. ***ImageChange*** ist ein *Trigger*-Typ, der auf Änderungen des angegebenen Docker Images regiert.
3. ***ConfigChange*** ist ein *Trigger*-Typ, der auf Änderungen der Konfiguration eines Openshift Artefaktes regiert.

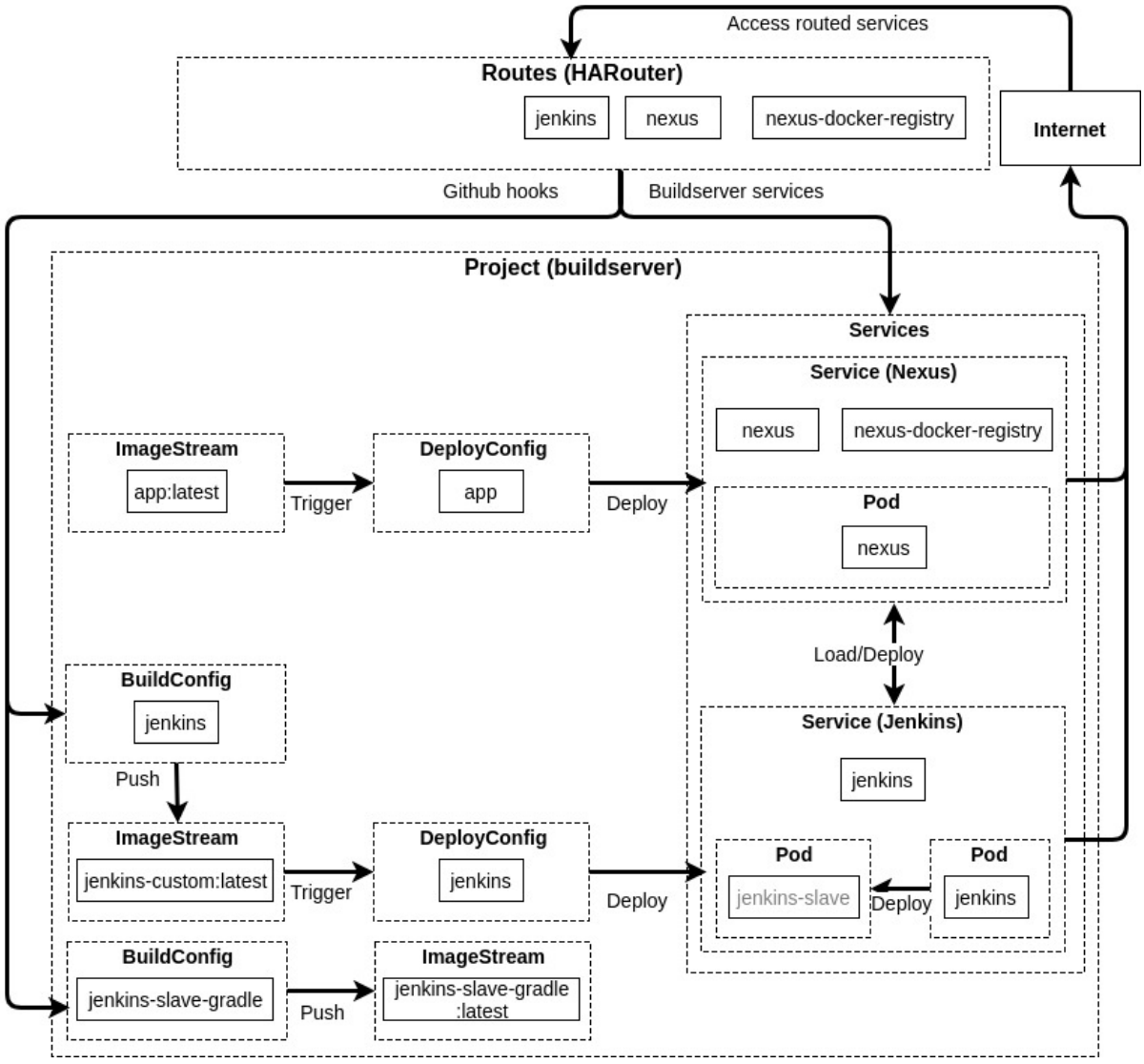

Figure 1: *Build Server* Architektur

[10] `https://docs.openshift.com/container-platform/3.5/dev_guide/builds/triggering_builds.html`

Die Abbildung 4 zeigt die Architektur der *Build*-Server Infrastruktur, sowie den Ablauf der Prozesse, die von den Triggern ausgelöst werden. Die folgenden Abschnitte beschreiben die verwendeten *Trigger*-Typen näher.

4.3.1 Github *Trigger*

Dieser Abschnitt behandelt den verwendeten *Github Trigger*, der dazu verwendet wird, um die Docker Images bei Quelltextänderungen neu zu bauen und den Service zu aktualisieren.

Beim Jenkins Docker Image und den *Slave* Images, wird bei einem *Push*, das Image neu gebaut.

```
triggers:
- type: "GitHub"
  github:
  secret: "<SECRET_NAME>" # Obfuscates the webhook url, not really a secret
```

Wenn ein Github *Trigger*-Typ definiert wurde, wird von Openshift eine *Webhook Url* erstellt, die bei Github registriert werden muss.

4.3.2 *ImageChange Trigger*

Dieser Abschnitt behandelt den verwendeten *ImageChange Trigger*, der dazu verwendet wird, um bei Änderungen der Docker Images, die über *ImageStream* repräsentiert werden, die Docker Images der Service neu zu bauen.

Alle verwendeten Services definieren einen *ImageChange Trigger*. Änderungen an den Docker Images, die über *ImageStreamTags* referenziert werden, werden nur bei Docker *Registries* in Version 2 unterstützt, da Docker *Registries* in Version 1 es nicht erlauben Images eindeutig zu identifizieren.

```
triggers:
- type: "ImageChange"
  imageChange:
  automatic: true
  containerNames:
    - "<CONTAINER_NAME_USING_IMAGE>"
  from:
    kind: "ImageStreamTag"
    name: "<IMAGE_STREAM_NAME:IMAGE_STREAM_TAG_NAME>"
```

4.3.3 *ConfigChange Trigger*

Dieser Abschnitt behandelt die verwendeten *ConfigChange Trigger*, die auf Änderungen der Konfiguration des Openshift Artefakts reagieren.

Alle definierten Konfigurationen verwenden den *ConfigChange Trigger*, der bei Änderungen einer Konfiguration z.B. einen neuen *Build* oder ein neues *Deployment* auslöst.

```
triggers:
  - type: "ConfigChange"
```

5 *App Server*

Dieser Abschnitt behandelt das Openshift Projekt *App*-Server, das die gebauten Services beinhaltet. Wenn der Jenkins Service, der im *Build*-Server Projekt enthalten ist am *Cluster* die nötigen Berechtigungen hat, kann die Jenkins Pipeline mit anderen Openshift Projekten interagieren. Da dies in der It&Tel *Cloud* nicht möglich war, wird zu Demozwecken, der App Service im *Build*-Server Projekt eingespielt.

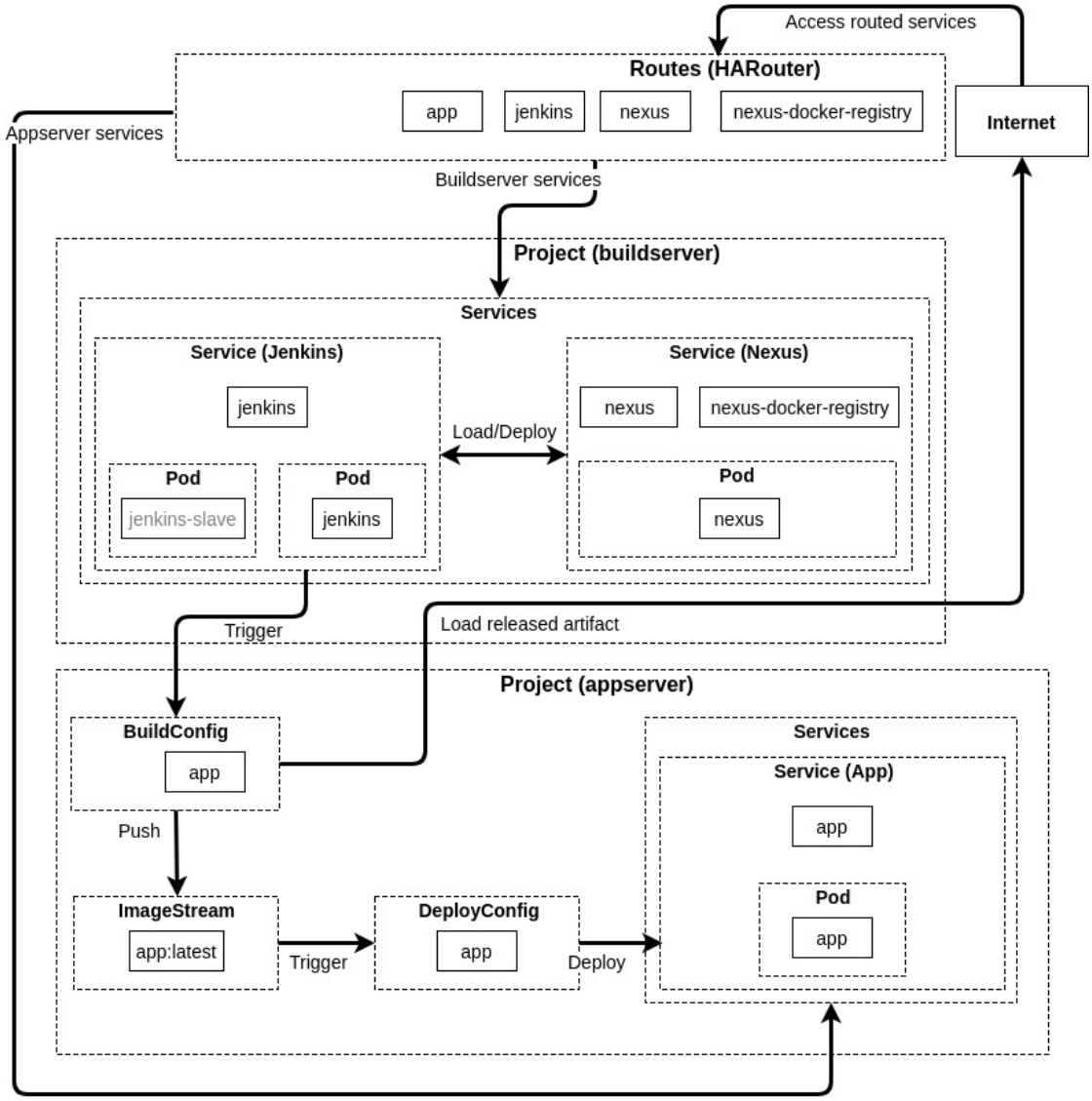

Figure 2: *App* Server Architektur

5.1 *Update*-Szenarien

Dieser Abschnitt behandelt die *Update*-Szenarien der Beispielanwendung, die in Jenkins über eine Jenkins Pipeline gebaut wird. Für diese Anwendung gibt es nur ein *Update*-Szenario, nämlich wenn ein Jenkins Pipeline *Build* eine neue Version freigibt, die neu eingespielt werden muss.

Die Jenkins Pipeline löst eine *Build*-Konfiguration aus, die das neue Docker Image baut, wobei im Anschluss ein *ImageChagne* Trigger ausgelöst wird, der den Service mit dem neuen Docker Image neu einspielt.

6 *Secrets*

Dieser Abschnitt behandelt die verwendeten *Secrets*.

Das *Secret-Objekt* oder kurz *Secret* bietet eine Möglichkt zum Speichern vertraulicher Informationen wie Passwörter, Konfigurationsdateien, dockercfg-Dateien, Anmeldeinformationen für Source-Repositorys und viele mehr. *Secrets* entkoppeln sensible Inhalte von den Pods und können mithilfe eines Volumen-Plug-Ins in Containern bereitgestellt werden.

```
apiVersion: "v1"
kind: "Secret"
metadata:
  name: "test-secret"
  namespace: "my-namespace"
data:
  username: "dmFsdWUtMQ0K"
  password: "dmFsdWUtMg0KDQo="
stringData:
  hostname: "myapp.mydomain.com"
```

Listing 1: Secret Definition

6.1 Eigenschaften von Secrets

Secrets können unabhängig von ihrer Definition referenziert werden. Sie werden in temporären *File-storage facilities* (tmpfs) gespeichert und werden niemals auf einem Knoten abgelegt. Secrets können auch innerhalb eines Namensraums geteilt werden.

6.2 Projekt Secrets

In diesem Projekt werden in den diversen Builds verschiedene Secrets verwendet. Zu diesen Secrets zählen unter Anderem ein SSH-Key für GitHub und die jeweiligen Benutzernamen und Passwörter für zweichen OpenShift, Jenkins und Nexus.

7 Jenkins *Build*

7.1 Stage Prepare

In diesem Build-Schritt werden alle Vorbereitungen für den eigentlichen Build getroffen. Es werden Variablen für *Stash* und *Unstash* angelegt, um das Projektverzeichnis zwischen den Steps/Pods zu verschieben. Weiters wird das aktuelle Repository verifiziert und eine Prüfsumme abgefragt.

```
stage('Prepare') {
  println "Preparing the build..."
  STASH_GIT_REPO="git-repo"
  STASH_BUILD="build-result"

  println "Stashing git repo..."
  dir('../workspace@script'){
    GIT_REF = sh returnStdout: true, script: 'git rev-parse --verify HEAD'
    stash name: STASH_GIT_REPO, includes: '**/*'
  }
  println "Stashed  git repo: 'git-repo'"
  println "Prepared  the build"
}
```

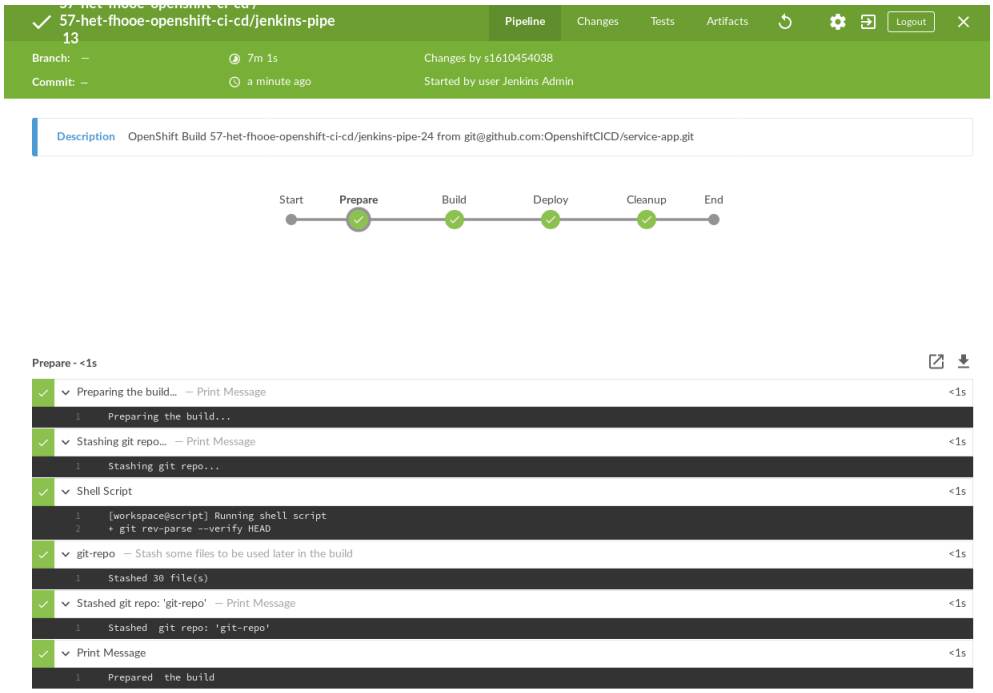

Figure 3: Prepare Build Server

7.2 Stage Build

Der eigentliche *Build* findet in einem Pod, d.h. in einem *Build-Slave*, der extra dafür gestartet wird, statt. In diesem Fall wird ein *Gradle-Build-Slave* gestartet, die *Sourcen* werden mittels *./gradle* gebaut und danach wird der *Build-Slave* zerstört. Zusätzlich werden noch Umgebungsvariablen an Gradle übergeben, welche in den Build-Targets genutzt werden, um z.B. Abhängigkeiten aus einem lokalen Nexus-Repository zu laden.

```
stage('Build') {
  NEXUS_USER="${env.NEXUS_USER}"
  NEXUS_PASSWORD="${env.NEXUS_PASSWORD}"
  NEXUS_MIRROR_URL="${env.MAVEN_MIRROR_URL}"
  MAVEN_REPOSITORY_URL="${env.MAVEN_REPOSITORY_URL}"

  podTemplate(name: 'jenkins-slave-gradle',
        cloud: 'openshift', containers: [
    containerTemplate(name: 'jnlp',
            image: 'ci/jenkins-slave-gradle', resourceRequestCpu: '500m',
            resourceLimitCpu: '4000m', resourceRequestMemory: '1024Mi',
            resourceLimitMemory: '4096Mi', slaveConnectTimeout: 180)
  ]) {
    node('jenkins-slave-gradle'){
      container('jnlp'){
        println "Unstashing '${STASH_GIT_REPO}'..."
        unstash STASH_GIT_REPO
        dir('\\complete') {
          echo sh(returnStdout: true, script: "gradle
          -PnexusUsername=$NEXUS_USER -PnexusPassword=$NEXUS_PASSWORD
          -PmirrorUrl=$NEXUS_MIRROR_URL
          -PrepositoryUrl=$MAVEN_REPOSITORY_URL build")
        }
        println "Built  with gradle"

        println "Stashing the workspace..."
        stash name: STASH_BUILD, includes: '**/*'
        println "Stashed  the workspace"
      }
    }
  }
}
```

7.3 Stage Deploy

Der Trigger *OpenShiftBuild* führt das Äquivalent zum Aufruf des Befehls oc start-build aus, bei dem die Build-Protokolle in Echtzeit an die Ausgabe des Jenkins-Plug-ins ausgegeben werden können. Zusätzlich zur Bestätigung, ob der Build erfolgreich war oder nicht, kann dieser Build-Schritt optional prüfen, ob die Deployment-Configs sogenannte Image-Change-Trigger für das von der Build-Config erzeugte Image haben. Wenn solche Deployment-Configs gefunden werden, werden diese analysiert, um festzustellen, ob sie durch eine Imageänderung ausgelöst wurden. Dabei wird das vom aktuell ausgeführten Replication-Controler verwendete Image mit dem Image verglichen, das von seinem unmittelbaren Vorgänger verwendet wurde.

```
stage('Deploy') {
  // Set app version on app build config
  echo sh(returnStdout: true,
              script: "oc env buildconfigs/spring-boot VERSION=1.0.0")

  // Trigger the build config with the new version
  openshiftBuild(buildConfig: 'spring-boot',
                      showBuildLogs: "true",
                      checkForTriggeredDeployments: "true")

   // Verify successful deployment
   openshiftVerifyDeployment(deploymentConfig: 'spring-boot',
                                          replicaCount: "1",
                                          verifyReplicaCount: "true",
                                           waitTime: "30000")
}
```

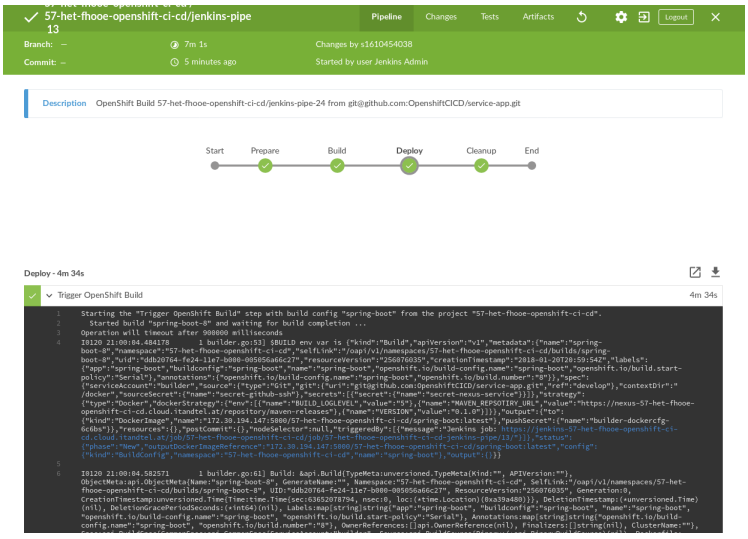

Figure 4: Deploy Build

8 Diskussion

Dieser Abschnitt behandelt die Diskussion, des implementierten Prototypen. Die implementierte CI/CD Umgebung in Openshift war relativ einfach zu realisieren, da Jenkins und Jenkins Pipeline von Openshift nativ unterstützt werden. Da Openshift eine relativ alte Version von Jenkins und seinen Plugins zur Verfügung stellt, wurde ein eigenes Jenkins Image über einen S2I *Build* definiert, das eine aktuelle Version von Jenkins und seinen Plugins zur Verfügung stellt. Das war einfach zu relaisieren und in Openshift zu integrieren, da beim Anlegen einer Jenkins Pipeline nach einem bestehenden Service mit Namen *jenkins* gesucht wird, um in dieser Jenkins Instanz den Pipeline *Build* anzulegen.

Für Jenkins und Nexus werden auch Openshift *Templates* bereitgestellt, die aber angepasst wurden, um unseren Wünschen zu entsprechen. Hier verhält es sich wie bei *Dockerfiles*, bei denen man auch über kurz oder lang seine eigenen *Dockerfiles* implementiert anstatt die zur Verfügung gestellten zu verwenden.

Wenn man die Grundkonzepte, die hinter Kubernetes und Openshift stehen, verstanden hat, ist es relativ leicht mit Opensift zu arbeiten und Strukturen wie eine CI/CD Umgebung zu realisieren. Vor allem die Richtlinien wie *Dockerfiles* implementiert werden sollen sind essentiell. Wenn man sich z.B. mit `oc rsh` in einem Docker *Container* via ssh verbindet, so ist man immer ein Benutzer mit einer zufälligen *uid* in der Gruppe *0*, daher müssen alle Rechte so gesetzt werden, dass ein Benutzer in dieser Gruppe auch die nötigen Rechte hat.

Openshift verlangt auch dass immer explizit ein Benutzer in der *Dockerfile* angegeben wird, mit dem der Prozess laufen soll. Standardmäßig ist es auch nicht erlaubt einen Docker Container mit *Root*-Rechten zu starten. In den beiden vorherig beschriebenen Fällen wird von Openshift der Docker Container mit einer zufälligen *uid* gestartet, was dazu führen kann, dass der Docker *Container* nicht starten kann.