CI/CD mit Openshift

Thomas Herzog B.Sc / Phillip Wurm B.Sc

January 21, 2018

Abstract

Dieses Dokument beinhaltet die Dokumentation der *CI/CD*-Umgebung, die in Oepnshift aufgesetzt wurde.

Contents

1 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.

1.1 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 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 ausschließlich diese Images.

1.2 Firewall Setup

Dieser Abschnitt behandelt das Einrichten der Firewall für die Verwendung von Openshift.

```
# 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
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
```

Ohne diese Firewall Einstellungen kann kein Docker Container, der in Openshift läuft, DNS Namen auflösen, oder auf das Internet zugreifen.

1.3 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***
   https://developers.redhat.com/products/openshift/download/
   Es wird die Version *3.5.5.31.24* benötigt, die unter *älter Versionen* gefunden werden kann.
   Es wird ein aktiver JBoss *Developer Account* vorausgesetzt.

2. ***openshift-client-wrapper***
   https://github.com/openshift-evangelists/oc-cluster-wrapper/releases/tag/0.9.3:
   Bei *oc-cluster-wrapper* handlet es sich um ein Shell-Skript, 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.

1.3.1 Kommandos

Dieser Abschnitt beschreibt die Kommandos, die für das Kontrollieren des *Cluster*.

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

# Stop the local cluster of profile 'ci'
oc-cluster-wrapper down ci

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

Wenn beim Starten folgende Fehlermeldung auftritt, dann liegt es daran, dass die Konfiguration nicht mehr gültig, da Sie sich z.B. in einem anderen Netz befinden.

```
# 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.

1.4 Ultrahook Setup

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

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.

2 *Build Server*

Dieser Abschnitt behandelt die Infrastruktur des *Build*-Server Projekts in Openshift. Die Ressourcen für den *Build Server* werden im *Repository buildserver*[1] verwaltet.

2.1 *Templates*

Dieser Abschnitt behandelt die Openshift *Templates*, welche die Services für die *Build*-Server Infrastruktur definiert. Die Openshift *Templates* beinhalten alle Definitionen wie z.B. *BuildConfigurations* und *Deployments*, die Aspekte des *Core Conepts*[2] von Kubernetes und Openshift sind.

Diese Auflistung beschreibt die implementierten *Templates*:

1. Im *Template* ***jenkins-slaves.yml*** werden die für Jenkins zur Verfügung gestellten *Slave-Container* verwaltet.
2. Im *Template* ***jenkins.yml*** wird der Jenkins Service verwaltet.
3. Im *Template* ***nexus.yml*** wird der Nexus3 Service verwaltet.
4. Im *Template* ***pipeline.yml*** wird für das Anlegen einer Openshift *Pipeline* verwendet.

2.2 Skripte

Dieser Abschnitt behandelt die Skripten, die für das Verwalten des *Clusters* verwendet werden. Mit der Applikation *oc* kann mit dem *Cluster* interagiert werden, wie z.B. PRojekte erstellen/löschen, oder Applikation in Projekten anlegen/löschen. Damit der *Build Server* einfach erstellt oder gelöscht werden kann, sind für die Services und für den *Build Server* Skripte erstellt worden, die alle nötigen Kommandos beinhalten.

Auf dem Level der Skripten wird eine Datei namens *.openshift-env* erwartet, die Umgebungsvariablen definiert, die von Skripten verwendet wird.

Diese 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.

[1] `https://github.com/OpenshiftCICD/buildserver`

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

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 5 für eine genauere Beschreibung der verwendeten *Secrets*.

2.3 Architektur

Dieser Abschnitt behandelt den Aufbau des *Build Servers*. Die Abbildung 5 zeigt, die Architektur des *Build Server* Projekts in Openshift.

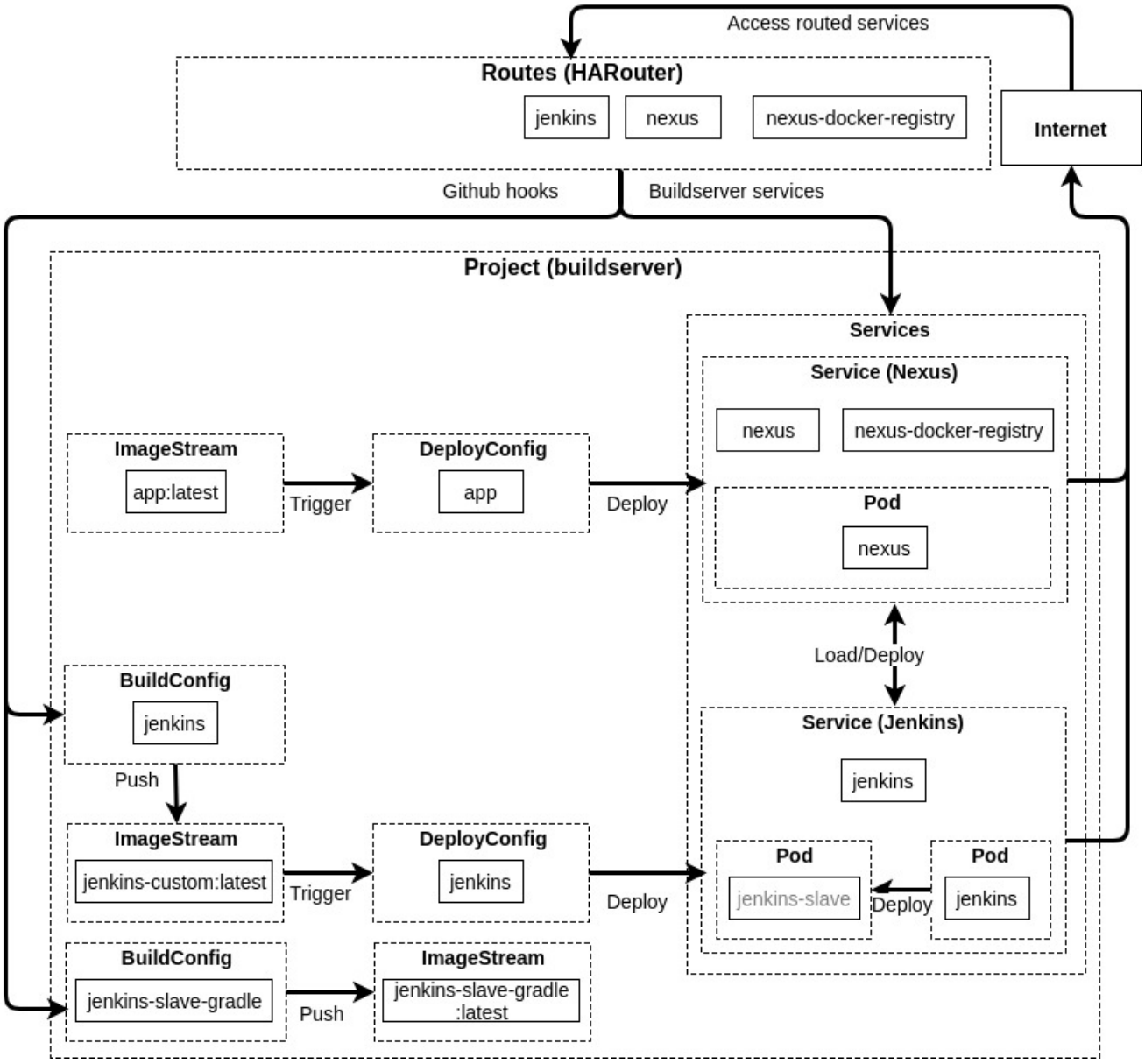

Figure 1: *Build Server* Architektur

2.4 *Upgrade / Downgrade*

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

1. Bei ***Änderung der Quelltexte*** wie *S2I Builds*[3] oder *Dockerfiles*, muss der Docker *Container* neu gebaut und der Service aktualisiert werden.
2. Bei ***Änderung der Konfigurationen***, muss der Service gegebenenfalls neu gebaut und aktualisiert werden.
3. Bei ***Änderung der Docker Images***, die Services beinhalten oder Basisimages darstellen, muss der Service aktualisiert werden.

Openshift erlaubt es bei *DeploymentConfigs* und *BuildConfigs Trigger*[4] zu definieren, die bei Ereignissen wie *Github Commit*, *ImageChange* oder *ConfigChange* ausgelöst werden. Die folgenden Abschnitte beschreiben die verwendeten *Trigger*.

2.4.1 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* auf den *master Branch*, das Image neu gebaut, da ein *Push* auf den *master Branch* als ein Release angesehen wird.

```
triggers:
  - type: "GitHub"
    github:
      secret: "<SECRET_NAME>"
```

Wenn ein Github *Trigger* definiert wurde, wird von Openshift eine *Hook Url* erstellt, die bei Github registriert werden kann.

2.4.2 *ImageChange Trigger*

Dieser Abschnitt behandelt den verwendeten *ImageChange Trigger*, der dazu verwendet wird, um bei Änderungen der Docker Images die Service zu aktualisieren.

Alle verwendeten Services definieren einen *ImageChange Trigger*, der bei einer Änderung des Docker Images, welches über *ImageStreams* verwaltet werden, ausgelöst wird. 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.

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

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

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

2.4.3 *ConfigChange Trigger*

Dieser Abschnitt behandelt die verwendeten *COnfigChange Trigger*, die auf Änderungen der Konfiguration regieren.

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

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

3 *Build Server*

Dieser Abschnitt behandelt die Infrastruktur des *Build*-Server Projekts in Openshift. Die Ressourcen für den *Build Server* werden im *Repository buildserver*[5] verwaltet.

3.1 *Templates*

Dieser Abschnitt behandelt die Openshift *Templates*, welche die Services für die *Build*-Server Infrastruktur definiert. Die Openshift *Templates* beinhalten alle Definitionen wie z.B. *BuildConfigurations* und *Deployments*, die Aspekte des *Core Conepts*[6] von Kubernetes und Openshift sind.

Diese Auflistung beschreibt die implementierten *Templates*:

1. Im *Template* ***jenkins-slaves.yml*** werden die für Jenkins zur Verfügung gestellten *Slave-Container* verwaltet.
2. Im *Template* ***jenkins.yml*** wird der Jenkins Service verwaltet.
3. Im *Template* ***nexus.yml*** wird der Nexus3 Service verwaltet.
4. Im *Template* ***pipeline.yml*** wird für das Anlegen einer Openshift *Pipeline* verwendet.

[5] `https://github.com/OpenshiftCICD/buildserver`

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

3.2 Skripte

Dieser Abschnitt behandelt die Skripten, die für das Verwalten des *Clusters* verwendet werden. Mit der Applikation *oc* kann mit dem *Cluster* interagiert werden, wie z.B. PRojekte erstellen/löschen, oder Applikation in Projekten anlegen/löschen. Damit der *Build Server* einfach erstellt oder gelöscht werden kann, sind für die Services und für den *Build Server* Skripte erstellt worden, die alle nötigen Kommandos beinhalten.

Auf dem Level der Skripten wird eine Datei namens *.openshift-env* erwartet, die Umgebungsvariablen definiert, die von Skripten verwendet wird.

Diese 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 5 für eine genauere Beschreibung der verwendeten *Secrets*.

3.3 Architektur

Dieser Abschnitt behandelt den Aufbau des *Build Servers*. Die Abbildung 5 zeigt, die Architektur des *Build Server* Projekts in Openshift.

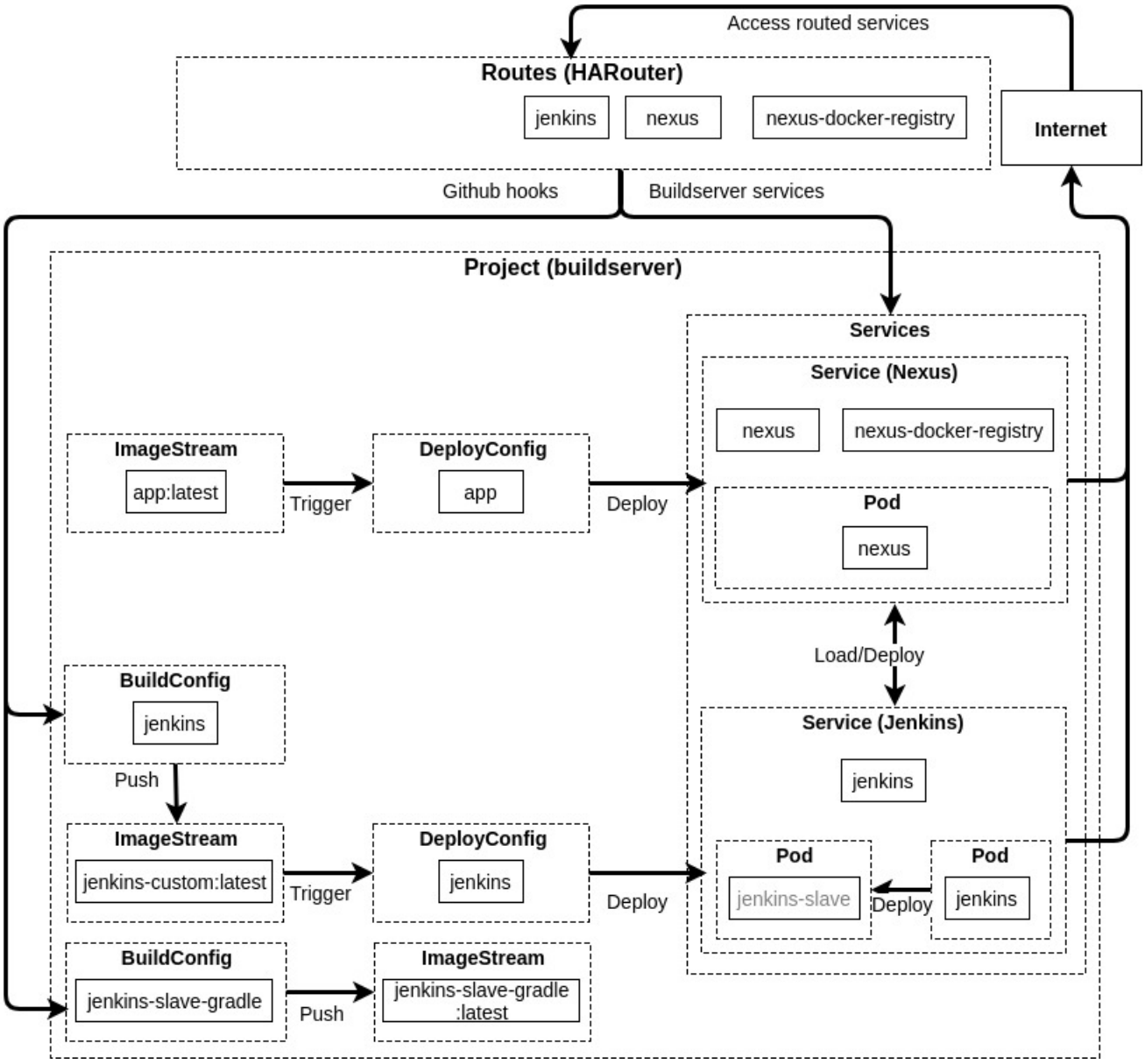

Figure 2: *Build Server* Architektur

4 *App Server*

Dieser Abschnitt behandelt das Openshift Projekt *Appserver*, das die gebauten Services beinhaltet. Die Jenkins Pipeline löst nach dem Bauen und Veröffentlichen des Artefakts wird im Openshift Projekt *Appserver* eine *Build*-Konfiguration ausgelöst, die das Docker Image mit dem freigegebenen Artefakts baut. Wenn der *Build* fertiggestellt ist, dann wird ein *Trigger* bei der *Deploy*-Konfiguration ausgelöst, der aus dem gebauten Docker Image den neuen Service einspielt.

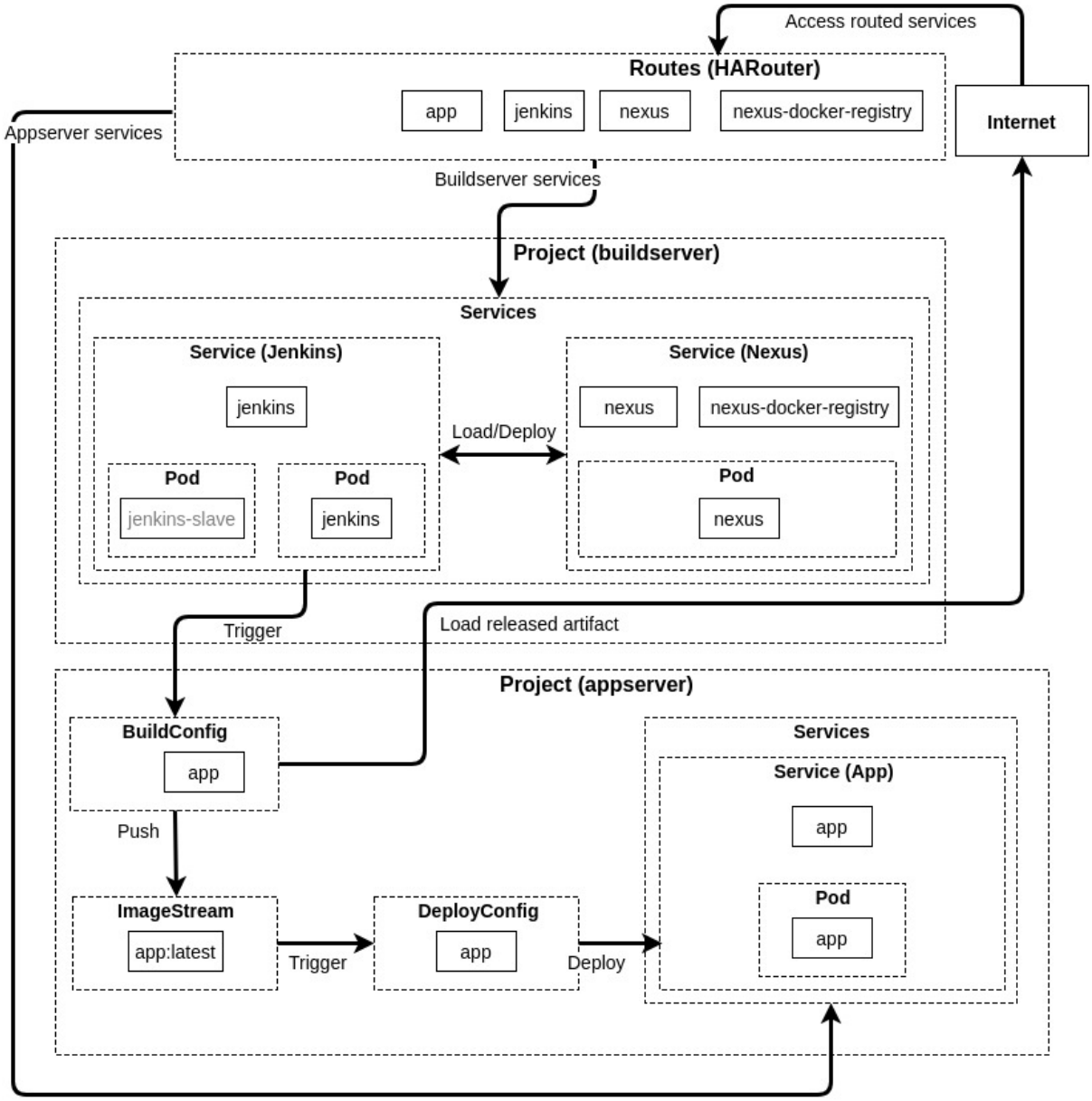

Figure 3: *Build Server* Architektur

Die Abbildung 3 illustriert das Zusammenspiel der beteiligten Openshift Artefakte und

Services aus mehreren Projekten. In der It&Tel *Cloud*, ist es nicht möglich gewesen einen *Service Account* die *EDIT* Rolle zu zuweisen und einen *API-Token* zu bekommen. Dadurch ist es Jenkins nicht möglich auf Artefakte des Projektes *Appserver* zu zugreifen und dort *Builds* zu starten.

Daher wird der Service im *Buildserver* Projekt eingespielt, denn in diesem Projekt hat Jenkins die *EDIT* Rolle. Der Prozess des Service *Releases* ändert sich dadurch aber nicht.

5 *Secrets*

Dieser Abschnitt behandelt die verwendeten *Secrets*.

Das *Secret-Objekt* oder kurz *Secrets* 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

5.1 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.

5.2 Projekt Secrets

6 Jenkins *Build*

6.1 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"
}
```

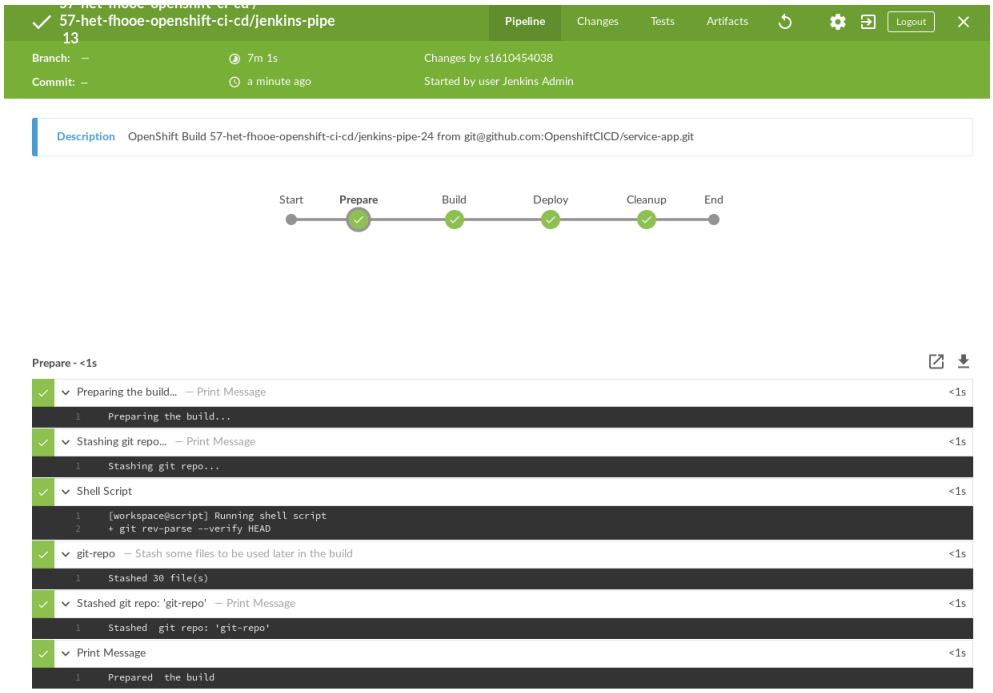

Figure 4: Prepare Build Server

6.2 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"
      }
    }
  }
}
```

6.3 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') {
  openshiftBuild(buildConfig: 'spring-boot',
                 env: [[ name: 'VERSION', value: '0.1.0' ]],
                 showBuildLogs: "true")
}
```

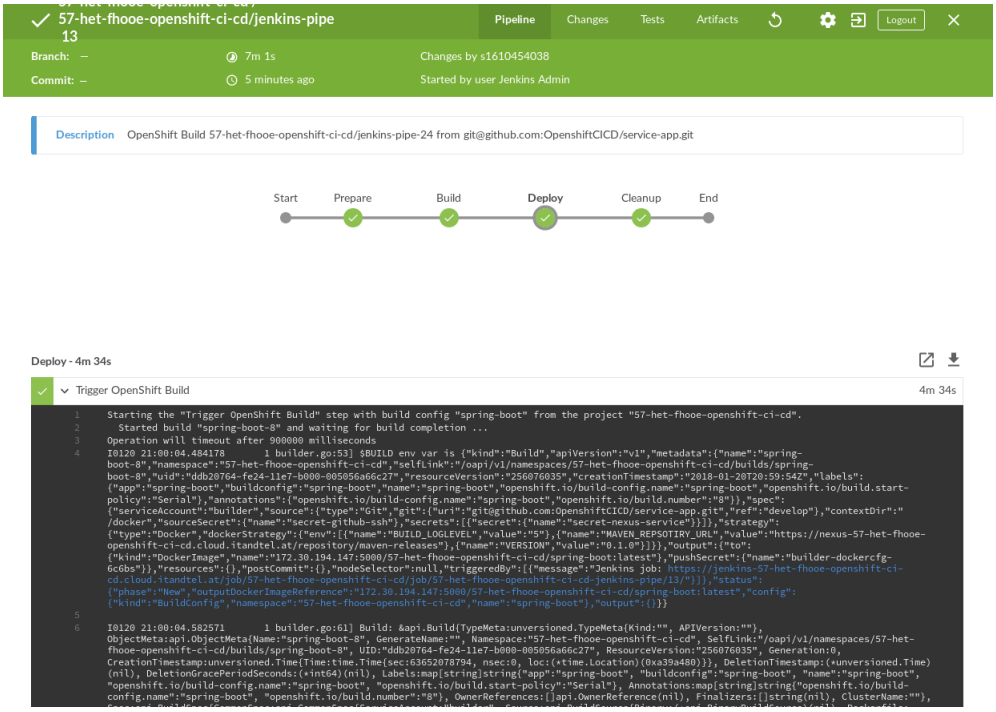

Figure 5: Deploy Build