Advanced Application Development with Red Hat OpenShift

By Marc Dätwyler

Avaloq Evolution AG

marc.daetwyler@avaloq.com

Table of Contents

Overview....2
Infrastructure Setup....2
    Nexus Setup....2
    SonarQube Setup....2
    Jenkins Setup....3
        Jenkins Slave....3
        Jenkins build pipelines....3
Application Setup....4
    Development....4
    Production....5
Pipelines....5
    MLB Parks Pipeline....5
    Nationalparks & Parksmap....6
Challenges & Lessons Learned....6

Overview

This document is a step-by-step guide on how to setup a complex cluster of infrastructure components and applications in a typical CI/CD environment based on Red Hat OpenShift.

In our assignment we are supposed to help MitziCom, a company that deals wit geographical information systems. In our case we have an application architecture with two backend applications (one for handling National Park data and one for Mayor League Baseball Parks information) and one frontend application, that is able to connect to both backends dynamically based on an OpenShift route.

Infrastructure Setup

The infrastructure consists of the following components:

- Jenkins as the core component for defining and executing our CI/CD pipelines
- Nexus 3 to cache the maven dependencies (to make our Java maven builds faster) and to store the docker images we build.
- SonarQube for static code analysis
- All the source code is hosted on GIT hub (https://github.com/argaen77/open_shift_hw)

Nexus Setup

- Nexus is setup from a public docker image: **docker.io/sonatype/nexus3:latest**
- We use a memory request of 1 GB and a limit of 2 GB as well as a request for 1 CPU core
- A persistent volume claim is mounted on path **/nexus-data**
- We need to initialize Nexus and setup the repositories after start up. This is done with a call to curl: curl -s https://raw.githubusercontent.com/wkulhanek/ocp_advanced_development_resources/master/nexus/setup_nexus3.sh | bash -s admin admin123 http://nexus-service:8081"
- The readiness probe check that the maven repository can be accessed via http-get: **/repository/maven-public/**
- Two services are created: **nexus3-registry** and **maven-registry**
- Two routes are created: **nexus3** and **nexus3-registry** (for docker images)

SonarQube Setup

- SonarQube needs a PostgreSQL database, that is setup in a separate deployment config.
  - PostgreSQL is setup with the following docker image: r**egistry.access.redhat.com/rhscl/postgresql-96-rhel7:latest**
  - User name, password and database name for PostgreSQL can be set via template parameters

    - The readiness probe is done by calling PostgreSQL directly via the psql command line client: psql -h 127.0.0.1 -U $POSTGRESQL_USER -q -d $POSTGRESQL_DATABASE -c 'SELECT 1'
    - The liveliness probe calls the pg_isready program
    - A persistent volume claim is mounted at **/var/lib/pgsql/data**
- SonarQube is setup from docker image: **wkulhanek/sonarqube:6.7.5**
- Resource limits and requests can be set as parameters in the template
- The readiness probe checks the **/about** service at port 9000
- A service is created: **sonarqube** at port 9000
- A route is exposed: **sonarqube** at port 9000

Jenkins Setup

- Jenkins is setup from an image stream: **jenkins:2**
- Jenkins needs to be setup with enough resources: We allow 1 – 2 GB or memory and 1 -2 CPU cores.
- Both readiness and liveliness probes check the /**login** service at port **8080**
- A persistent volume claim is mounted at **/var/lib/jenkins**
- A ServiceAccount is added to allow OAuth authentication.
- Two services are added: **jenkis-jnlp** at port **5000** and **jenkins**
- A route is exposed on the jenkins service (**jenkins**).

Jenkins Slave

Jenkins will run the CI/CD pipelines in a dedicated OpenShift pod. For Java we could create a pod based on the **docker.io/openshift/jenkins-slave-maven-centos7:v3.9** maven image, but this image does not contain skopeo, a program that we need to copy images. Therefore we define our own image and use it in Jenkins.

- In the Dockerfile we simply inherit from the maven image and add skopeo:
    - RUN yum -y install skopeo
- Then we create a config map for Jenkins in which we create a Jenkins slave pod configuration based on our new image.
- The image is stored in the OpenShift image registry and used form there by Jenkins:

  <image>docker-registry.default.svc:5000/${GUID}-jenkins/jenkins-slave-maven</image>

Jenkins build pipelines

We setup an OpenShift build config for each of the three Jenkins pipelines that we will execute later on.

Application Setup

Development

First we setup a (non-replicated) MongoDB

- MongoDB is setup via a template for a persistent deployment
- With a config map we setup the host, user name, password and database name that are later on used by the application.

Then we setup the three development configurations:

MLB Parks build and deployment configuration

- Build by S2I based on a **jboss-eap70-openshift:1.7** docker image
- Deployment configuration that is ready for the pipelined build (--allow-missing-imagestream-tags=true)
- A service is created on port **8080**
- Readiness and liveliness probes are added based on the health service of the J2EE application (--get-url=http://:8080/ws/healthz/)
- Config maps are added to link the deployment to the MongoDB (user, password etc.)
- A deployment hook is set to populate the database (curl -s http://mlbparks:8080/ws/data/load/)

Notional Parks build and deployment configuration:

- Build by S2I based on a **redhat-openjdk18-openshift:1.2** docker image
- Uses the same MongoDB config map as the MLB Parks
- Deployment configuration that is ready for the pipelined build (--allow-missing-imagestream-tags=true)
- A service is created on port **8080**
- Readiness and liveliness probes are added based on the health service of the SpringBoot application (--get-url=http://:8080/ws/healthz/)
- Config maps are added to link the deployment to the MongoDB (user, password etc.)
- A deployment hook is set to populate the database (curl -s http://nationalparks:8080/ws/data/load/)

ParksMap build and deployment configuration

- Build by S2I based on a **redhat-openjdk18-openshift:1.2** docker image
- Deployment configuration that is ready for the pipelined build (--allow-missing-imagestream-tags=true)
- A service is created on port **8080**

- Readiness and liveliness probes are added based on the health service of the SpringBoot application (--get-url=http://:8080/ws/healthz/)
- Expose a route (${GUID-parks-dev)

Production

The production build and deployment configurations are in many aspects identical to the development environment. I will only explain the differences to the development configurations here.

- The MongoDB database is created as a replicated database with three replicas running at the same time.
  - A statefull set is created with two services: An internal service (**mongodb-internal**), that allows the replicated instances to communicate and an external service (**mongodb**) for the applications to connect to.
  - Configuration comes form the **parksdb-conf** configuration map that contains information like the database name, user, password and db replica set name.
  - The script (setup_prod.sh) needs to wait for the replicated database to become available. This has to be done by adding a while loop to the shell script that checks for all pods to be available. **oc get pod -n ${GUID}-parks-prod|grep '\-2'|grep -v deploy|grep "1/1"**
- For the applications we add two configurations for each application:
  - One for “green” and one for “blue”
  - These configurations are identical except for the label tagging them as “green” or “blue”
  - There is only one route to the applications (**parksmap**, **mlbparks** and **nationalparks**) that initially all point to “green” deployments.

Pipelines

The pipelines are build in the Jenkinsfile source in each of the application locations inside the GIT repository. The pipelines can be monitored in both Jenkins and OpenShift.

MLB Parks Pipeline

The pipeline uses the **maven-custom** node/pod and the j**enkins-slave-maven** docker image, that allows us to use skopeo. The following description is in the order of the steps that are executed.

1. Get source code from SCM (https://github.com/argaen77/open_shift_hw)
2. Setup the maven settings.xml to point to our infrastructure nexus repository
3. Run the maven build: sh "mvn clean package -DskipTests"
4. Run the unit tests via maven (sh "mvn test") and the static code analysis via SonarQube (also triggered via maven target). This is done in parallel.

5. Build the docker image via source to image. After the build has succeed the image is tagged via the **openshiftTag** Jenkins pipeline command.
6. The application build by maven (J2EE) is copied to the Nexus repository, by calling the **deploy** maven target (sh “mvn deploy -DskipTests=true -DaltDeploymentRepository=nexus::default::${NEXUS_URL}")
7. The image is deployed to the development project. First the image is set on the deployment configuration and then the **rollout** is triggered. We wait for the rollout to finish.

   sh "oc -n 6a6f-parks-dev rollout latest dc/mlbparks && oc rollout status dc/mlbparks -w “
8. The integration test stage here is simulated by calling one of the services of the application. This is done via curl. We check for the HTTP response code to be 200 with a fgrep call on the shell:

   sh "curl -s -o /dev/null -w '%{http_code}\n' http://mlbparks.6a6f-parks-dev.svc:8080/ws/data/all | fgrep 200"
9. Now we take the docker image from the OpenShift docker registry and copy it to the Nexus repository server. This is done with **skopeo**. We call skopeo as a shell command on our Jenkins pod.
10. We tag the image for productive use via the Jenkins **openshiftTag** command.
11. “Green/Blue” deployment. First we check if “green” or “blue” is active. This can be found out by checking the route. Then we do a deployment to the non-active deployment configuration via the **openshiftDeploy** Jenkins pipeline command. After this has finished we check both deployment and service with the corresponding Jenkins commands (**openshiftVerifyDeployment**, **openshiftVerifyService**)
12. In the last step we switch the **route** from the active deployment (“green”/”blue”) to the newly deployed deployment. sh oc -n 6a6f-parks-prod patch route mlbparks -p '{\"spec\":{\"to\":{\"name\":\"mlbparks\"}}}'"

Nationalparks & Parksmap

These pipelines look almost exactly the same. The only difference is the build step. There are spring boot applications and not targeted towards an application server as the MLB Parks application.

sh "$mvn clean package spring-boot:repackage -DskipTests -Dcom.redhat.xpaas.repo.redhatga"

Challenges & Lessons Learned

This exercise gives a good overview on what can be done with the combination of Jenkins and OpenShift in term of CI/CD. There was a lot of research needed to find the right commands or configuration templates but this is something that you would expect of such a type of work.

What I did not like at all was the lack of processing resources and the missing stability on the AppDev Homework environment. You could never be sure if your scripts and pipelines would run or if they would suddenly stop due to a deployment that could not be finished etc.

I would go so far to say that it is impossible to finish the homework on the https://master.na39.openshift.opentlc.com if your are doing this exercise d77uring normal office hours or on evenings.