Advanced Application Development with Red Hat OpenShift – Homework Assignment
Student: Kostiantyn Iarovi
September 2018

RED HAT® OPENSHIFT Container Platform

MitziCom POC

Table of Contents

1. Overview

This document is a step-by-step guide to configure and demonstrate Proof of Concept of using Red Hat OpenShift as a PaaS solution for end-to-end CI/CD pipeline for existing Java-based microservices. Demo applications are 2 backend and 1 frontend app. Components of this solution include source GitHub repository with applications code in separate folders, Nexus as artifact repository, SonarQube as code analysis tool and Jenkins as a CI/CD tool. Deployment to production are using Blue/Green strategy.

Our frontend Java application is supposed to display a world map, whereas 2 backend Java applications will be providing geo data with location of National Parks and Major League Baseball (MLB) Parks. Frontend app will try to dynamically discover available backends by searching for available routes in OpenShift with type=”parksmap-backend”.

2. Environment details

OpenShift AdvDev Homework environment has been requested through OPENTLC portal.
URL: https://master.na39.openshift.opentlc.com/

All needed templates, bash scripts have been committed to the following repo:
https://github.com/kyarovoy/openshift_advdev_homework

Projects, tools and credentials

Project	**Description**	**URLs**	**Credentials**
ki-nexus	Nexus Artifact repository	http://nexus3-ki-nexus.apps.na39.openshift.opentlc.com/	user: admin pass: admin123
ki-jenkins	Jenkins for CI/CD pipelines	https://jenkins-ki-jenkins.apps.na39.openshift.opentlc.com/	OPENTLC
ki-sonarqube	SonarQube code analysis tool	http://sonarqube-ki-sonarqube.apps.na39.openshift.opentlc.com/	user: admin pass: admin
ki-parks-dev	Project for Development & Test environment		-
ki-parks-prod	Project for Production environment	http://parksmap-ki-parks-prod.apps.na39.openshift.opentlc.com/	-

3. Infrastructure components

3.1. Nexus

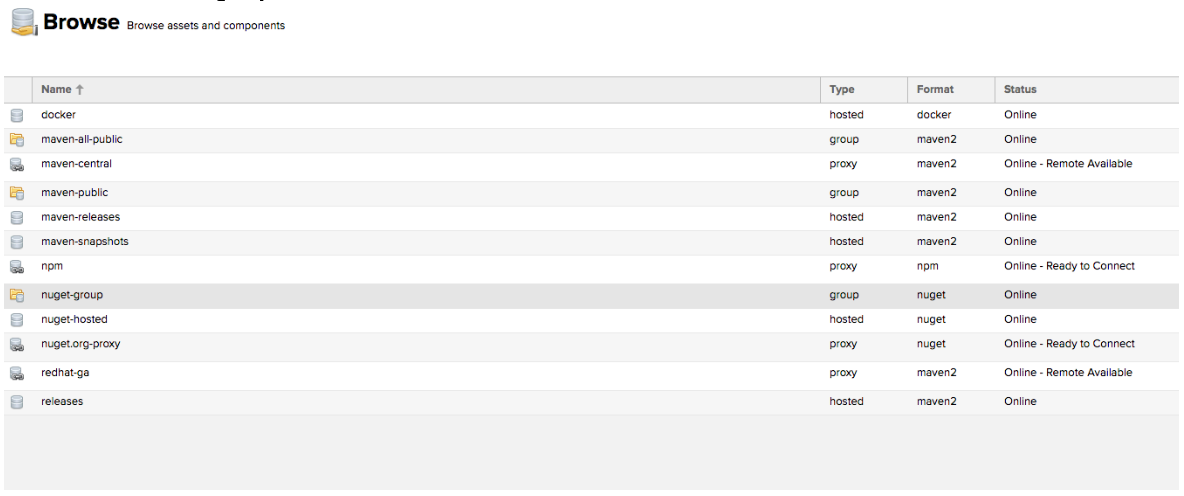
You can use pre-created OpenShift template to deploy Nexus with persistent storage (persistent volume claim is mounted at /nexus-data) and pre-configured liveness/readiness probes. Template also includes post deployment hook that runs a shell script, which pre-configures repositories needed by demo applications. This template is parameterized and accepts the following parameters:

```
parameters:
- name: NEXUS_SERVICE_NAME
  value: nexus3
  displayName: Nexus Service Name
  description: The name of Nexus Service.
- name: NEXUS_MEMORY_MIN
  value: 1Gi
  displayName: Mininum Nexus Memory
  description: Minimum amount of memory for nexus container.
- name: NEXUS_MEMORY_MAX
  value: 2Gi
  displayName: Maximum Nexus Memory
  description: Maximum amount of memory for nexus container.
- name: NEXUS_STORAGE_SIZE
  value: 4Gi
  displayName: Nexus storage size
  description: Volume space available for data.
```

Command line to deploy this component using oc tool:

```
oc -n $GUID-nexus new-app -f ${TMPL_DIR}/nexus.yaml
```

Screenshot of deployed Nexus UI:



Browse Browse assets and components

Name ↑	Type	Format	Status
docker	hosted	docker	Online
maven-all-public	group	maven2	Online
maven-central	proxy	maven2	Online - Remote Available
maven-public	group	maven2	Online
maven-releases	hosted	maven2	Online
maven-snapshots	hosted	maven2	Online
npm	proxy	npm	Online - Ready to Connect
nuget-group	group	nuget	Online
nuget-hosted	hosted	nuget	Online
nuget.org-proxy	proxy	nuget	Online - Ready to Connect
redhat-ga	proxy	maven2	Online - Remote Available
releases	hosted	maven2	Online

3.2. SonarQube

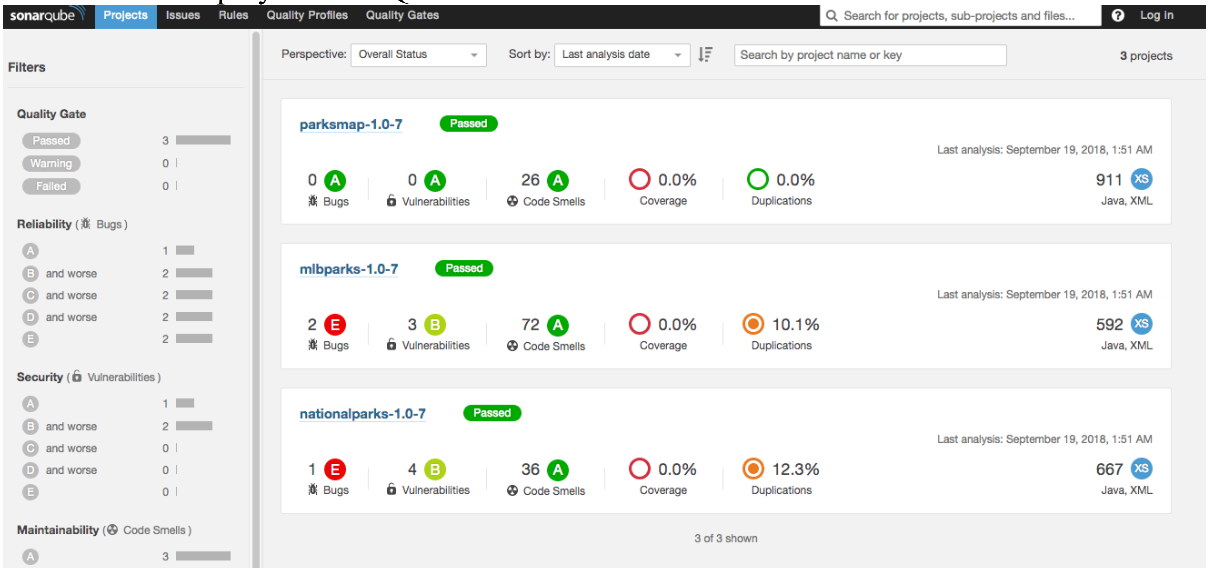
You can use pre-created OpenShift template to deploy SonarQube code analysis tool as well as it's dependency – PostgreSQL database, all with persistent storage (persistent volume claim is mounted at /var/lib/pgsql/data for PostgreSQL) and pre-configured liveness/readiness probes. This template is parameterized and accepts the following parameters:

```
parameters:
- name: SONARQUBE_SERVICE_NAME
  value: sonarqube
  displayName: SonarQube Service Name
  description: The name of the OpenShift Service exposed for the SonarQube
container (also used as the name of docker image).
- name: SONARQUBE_VERSION
  value: 6.7.4
  displayName: SonarQube Version
  description: the version (tag) of SonarQube docker image
- name: SONARQUBE_CPU_MIN
  value: 500m
  displayName: Minimum SonarQube CPU
  description: Minimum amount of CPU for sonarqube container.
- name: SONARQUBE_CPU_MAX
  value: 1500m
  displayName: Maximum SonarQube CPU
  description: Maximum amount of CPU for sonarqube container.
- name: SONARQUBE_MEMORY_MIN
  value: 2Gi
  displayName: Minimum SonarQube Memory
  description: Minimum amount of memory for sonarqube container.
- name: SONARQUBE_MEMORY_MAX
  value: 3Gi
  displayName: Maximum SonarQube Memory
  description: Maximum amount of memory for sonarqube container.
- name: SONARQUBE_STORAGE_SIZE
  value: 4Gi
  displayName: SonarQube storage size
  description: Volume space available for data.
- name: SONARQUBE_POSTGRESQL_STORAGE_SIZE
  value: 4Gi
  displayName: PostgreSQL for SonarQube storage size
  description: Volume space available for data.
- name: SONARQUBE_POSTGRESQL_VERSION
  value: "9.6"
  displayName: PostgreSQL for SonarQube version
  description: Version of PostgreSQL server used for SonarQube
- name: SONARQUBE_POSTGRESQL_USER
  value: sonar
  displayName: PostgreSQL user for SonarQube
  description: User used by SonarQube to connect to PostgreSQL server
- name: SONARQUBE_POSTGRESQL_PASSWORD
  value: sonar
  displayName: PostgreSQL password for SonarQube
  description: Password used to authenticate SonarQube while connecting to
PostgreSQL server
- name: SONARQUBE_POSTGRESQL_DATABASE
  value: sonar
  displayName: PostgreSQL database name for SonarQube
  description: PostgreSQL database name to store configuration for
SonarQube
```

Command line to deploy this component using oc tool:

```
oc -n $GUID-sonarqube new-app -f ${TMPL_DIR}/sonarqube.yaml
```

Screenshot of deployed SonarQube UI:



Filters

Quality Gate
Passed 3
Warning 0
Failed 0

Reliability (Bugs)
A 1
B and worse 2
C and worse 2
D and worse 2
E 2

Security (Vulnerabilities)
A 1
B and worse 2
C and worse 0
D and worse 0
E 0

Maintainability (Code Smells)
A 3

Perspective: Overall Status   Sort by: Last analysis date   Search by project name or key   3 projects

parksmap-1.0-7 Passed
Last analysis: September 19, 2018, 1:51 AM
0 A Bugs | 0 A Vulnerabilities | 26 A Code Smells | 0.0% Coverage | 0.0% Duplications | 911 XS Java, XML

mlbparks-1.0-7 Passed
Last analysis: September 19, 2018, 1:51 AM
2 E Bugs | 3 B Vulnerabilities | 72 A Code Smells | 0.0% Coverage | 10.1% Duplications | 592 XS Java, XML

nationalparks-1.0-7 Passed
Last analysis: September 19, 2018, 1:51 AM
1 E Bugs | 4 B Vulnerabilities | 36 A Code Smells | 0.0% Coverage | 12.3% Duplications | 667 XS Java, XML

3 of 3 shown

3.3. Jenkins

You can use pre-created OpenShift template to deploy Jenkins CI/CD tool with persistent storage (persistent volume claim is mounted at /var/lib/jenkins) and pre-configured liveness/readiness probes. This template is parameterized and accepts the following parameters:

```
parameters:
- description: The name of the OpenShift Service exposed for the Jenkins
container.
  displayName: Jenkins Service Name
  name: JENKINS_SERVICE_NAME
  value: jenkins
- description: The name of the service used for master/slave communication.
  displayName: Jenkins JNLP Service Name
  name: JNLP_SERVICE_NAME
  value: jenkins-jnlp
- description: Whether to enable OAuth OpenShift integration. If false, the
static
    account 'admin' will be initialized with the password 'password'.
  displayName: Enable OAuth in Jenkins
  name: ENABLE_OAUTH
  value: "true"
- name: CPU_MIN
  value: "1"
  displayName: Minimum Jenkins CPU
  description: Minimum amount of CPU for jenkins container.
- name: CPU_MAX
  value: "2"
  displayName: Maximum Jenkins CPU
  description: Maximum amount of CPU for jenkins container.
- description: Maximum amount of memory the container can use.
  displayName: Memory Limit
  name: MEMORY_LIMIT
  value: 1Gi
- description: Volume space available for data, e.g. 512Mi, 2Gi.
  displayName: Volume Capacity
  name: VOLUME_CAPACITY
  required: true
  value: 1Gi
- description: The OpenShift Namespace where the Jenkins ImageStream
resides.
  displayName: Jenkins ImageStream Namespace
  name: NAMESPACE
  value: openshift
- description: Name of the ImageStreamTag to be used for the Jenkins image.
  displayName: Jenkins ImageStreamTag
  name: JENKINS_IMAGE_STREAM_TAG
  value: jenkins:2
```

Command line to deploy Jenkins using oc tool:

```
oc -n $GUID-jenkins new-app -f ${TMPL_DIR}/jenkins.yaml -p
MEMORY_LIMIT=2Gi -p VOLUME_CAPACITY=4Gi
```

Once Jenkins is deployed we also need a custom Jenkins slave that supports not only Build-specific dependencies (e.g. maven etc.), but also has pre-installed tool for manipulating Docker images called Skopeo. This can be useful when deployments are using Docker and you need to copy images between different private Docker registries.

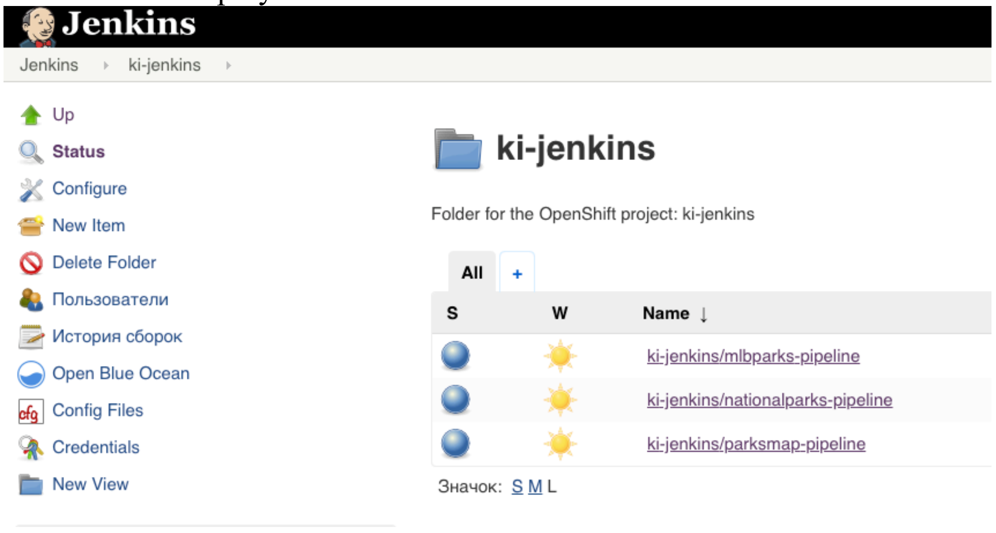
You can build custom Jenkins slave image using pre-created Dockerfile:
```
cat ${TMPL_DIR}/jenkins-slave-maven.Dockerfile | oc -n $GUID-
jenkins new-build --name=jenkins-slave-maven -D -
```

You can configure Jenkins to recognize previously built slave image as a Slave using the following command:
```
oc -n $GUID-jenkins new-app -f ${TMPL_DIR}/jenkins-
configmap.yaml --param GUID=${GUID}
```

Since Jenkins will be used to run 3 pipelines which will build, test and deploy corresponding microservices first in Dev env and then in production env, we need to create Jenkins Pipeline jobs using OpenShift using the following oc command:

```
oc -n $GUID-jenkins new-build ${REPO} --name="mlbparks-
pipeline" --strategy=pipeline --context-dir="MLBParks"
oc -n $GUID-jenkins set env bc/mlbparks-pipeline
CLUSTER=${CLUSTER} GUID=${GUID}

oc -n $GUID-jenkins new-build ${REPO} --name="nationalparks-
pipeline" --strategy=pipeline --context-dir="Nationalparks"
oc -n $GUID-jenkins set env bc/nationalparks-pipeline
CLUSTER=${CLUSTER} GUID=${GUID}

oc -n $GUID-jenkins new-build ${REPO} --name="parksmap-
pipeline" --strategy=pipeline --context-dir="ParksMap"
oc -n $GUID-jenkins set env bc/parksmap-pipeline
CLUSTER=${CLUSTER} GUID=${GUID}
```

Screenshot of deployed Jenkins UI:



Folder for the OpenShift project: ki-jenkins

S	W	Name ↓
●	☀	ki-jenkins/mlbparks-pipeline
●	☀	ki-jenkins/nationalparks-pipeline
●	☀	ki-jenkins/parksmap-pipeline

4. OpenShift application configuration

We have separate projects for Development and Production environment.

Development project will be used to deploy application after changes have happened in source code repository. It will be also used to build code and run code analysis and integration tests. If all tests will pass we are going to publish Artifacts in Nexus as an artifact repository, build and tag application Docker image and push it to our private Nexus Docker registry.

Development project can be created using the following command:

```
oc new-project ${GUID}-parks-dev  --display-name="${GUID}
AdvDev Homework Parks Development"
```

We can allow Jenkins from a separate Jenkins infrastructure project to manipulate objects in Development project using the following command:

```
oc -n ${GUID}-parks-dev policy add-role-to-user edit
system:serviceaccount:${GUID}-jenkins:jenkins
```

If all tests have passed in Development project, Production project will pull app Docker build image from Development project, tag it with production tag and deploy it in Production project. We need to allow that inter-project interaction using the following command:

```
oc -n ${GUID}-parks-dev policy add-role-to-group system:image-
puller system:serviceaccounts:${GUID}-parks-prod
```

Production project will be used to deploy application using Blue/Green approach if all tests passed in Development project. Using blue-green deployment, you serve one application at a time and switch from one application to the other.

Production project can be created using the following command:

```
oc new-project ${GUID}-parks-prod --display-name="${GUID}
AdvDev Homework Parks Production"
```

We can allow Jenkins from a separate Jenkins infrastructure project to manipulate objects in Production project using the following command:

```
oc -n ${GUID}-parks-prod policy add-role-to-user edit
system:serviceaccount:${GUID}-jenkins:jenkins
```

Our applications store data in MongoDB, so we will install MongoDB with persistent storage in both projects. Production MongoDB will be configured as a stateful set with 3 replicas for HA purposes, each set member will communicate with each other using headless service, whereas clients will connect to Replica Set using regular service.
You can use pre-configured templates for MongoDB deployments:

```
oc -n ${GUID}-parks-dev new-app mongodb-persistent --
name=mongodb -p MONGODB_USER=mongodb -p
MONGODB_PASSWORD=mongodb -p MONGODB_DATABASE=parks

oc -n ${GUID}-parks-prod new-app -f ${TMPL_DIR}/mongodb.yaml -
p MONGO_CONFIGMAP_NAME=parksdb-conf
```

We also keep in mind that our backend applications will need to access MongoDB using certain credentials and OpenShift provides a great way to decouple App image and it's configuration artifacts by means of using ConfigMaps. As an example, we will define configmap for MongoDB from command line which will be both used by backend apps and production MongoDB template:

```
oc -n ${GUID}-parks-dev create configmap parksdb-conf \
       --from-literal=DB_HOST=mongodb \
       --from-literal=DB_PORT=27017 \
       --from-literal=DB_USERNAME=mongodb \
       --from-literal=DB_PASSWORD=mongodb \
       --from-literal=DB_NAME=parks

oc -n ${GUID}-parks-prod create configmap parksdb-conf \
       --from-literal=DB_REPLICASET=rs0 \
       --from-literal=DB_HOST=mongodb \
       --from-literal=DB_PORT=27017 \
       --from-literal=DB_USERNAME=mongodb \
       --from-literal=DB_PASSWORD=mongodb \
       --from-literal=DB_NAME=parks
```

OpenShift templates can set their ENV variables based on ConfigMap key/value pairs, e.g. in MongoDB template this is done using the following code:

```
…
            env:
              - name: MONGODB_DATABASE
                valueFrom:
                  configMapKeyRef:
                    name: ${MONGO_CONFIGMAP_NAME}
                    key: DB_NAME
              - name: MONGODB_USER
                valueFrom:
                  configMapKeyRef:
                    name: ${MONGO_CONFIGMAP_NAME}
                    key: DB_USERNAME
              - name: MONGODB_PASSWORD
                valueFrom:
                  configMapKeyRef:
                    name: ${MONGO_CONFIGMAP_NAME}
                    key: DB_PASSWORD
              - name: MONGODB_ADMIN_PASSWORD
                valueFrom:
                  configMapKeyRef:
                    name: ${MONGO_CONFIGMAP_NAME}
                    key: DB_PASSWORD
              - name: MONGODB_REPLICA_NAME
                valueFrom:
                    configMapKeyRef:
                      name: ${MONGO_CONFIGMAP_NAME}
                      key: DB_REPLICASET
                - name: MONGODB_KEYFILE_VALUE
                  value: "12345678901234567890"
                - name: MONGODB_SERVICE_NAME
                  value: "mongodb-internal"
…
parameters:
- name: MONGO_CONFIGMAP_NAME
  value: parksdb-conf
  displayName: ConfigMap name with MongoDB configuration
  description: ConfigMap name with MongoDB configuration
```

Once all application dependencies have been satisfied we will proceed with configuring apps in OpenShift. Our goal in this phase is to pre-configure OpenShift application-related entities (Deployment configs, Build Configs etc.) to be ready for further invocation by Jenkins pipelines. We will need to configure 3 apps in 2 projects (Dev, Prod), in production Project we are going to configure 2 entities of each app (1 – for green deployment and 2nd for blue deployment). You can follow below provided steps to configure applications, instructions are provided just for 1 application, please change names accordingly and repeat for all apps in all projects (unless explicitly noted otherwise).

1. We first create a binary build using JBOSS image stream jboss-eap70-openshift:1.7 (for MLBParks) and OpenJDK1.8 image stream redhat-openjdk18-openshift:1.2 (for National Parks and FrontEnd app). This step doesn't need to be done in Production project since prod deployment will be done not based on binary builds:

```
oc -n ${GUID}-parks-dev --name=mlbparks new-build --
binary=true jboss-eap70-openshift:1.7
```

2. Create a new deployment config linked to a previously created imagestream. This deployment config will be tagged with TYPE=parksmap-backend to allow Frontend App to find it dynamically:

Development project command:
```
oc -n ${GUID}-parks-dev new-app ${GUID}-parks-
dev/mlbparks:0.0-0 --allow-missing-imagestream-tags=true --
name=mlbparks -l type=parksmap-backend
```

Production project command:
```
oc -n ${GUID}-parks-prod new-app ${GUID}-parks-
dev/mlbparks:0.0 --allow-missing-images=true --allow-missing-
imagestream-tags=true --name=mlbparks-blue -l type=parksmap-
backend
```

Please keep in mind that in production project we need to tag Blue and Green app deployments differently (e.g. "parksmap-backend" for Blue deployment and "parksmap-backend-reserve" for Green deployment).

3. For demo purposes we'll prefer to disable automatic building and deployment (e.g. to prevent any automatic rebuilds when config has been changed):

```
oc -n ${GUID}-parks-dev set triggers dc/mlbparks --remove-all
```

4. Expose deployment config as a service on port 8080:

```
oc -n ${GUID}-parks-dev expose dc/mlbparks --port 8080
```

5. Create readiness and liveness probes. :

```
oc -n ${GUID}-parks-dev set probe dc/mlbparks --readiness --
initial-delay-seconds 30 --failure-threshold 3 --get-
url=http://:8080/ws/healthz/
oc -n ${GUID}-parks-dev set probe dc/mlbparks --liveness --
initial-delay-seconds 30 --failure-threshold 3 --get-
url=http://:8080/ws/healthz/
```

6. Create additional app-specific ConfigMap with just 1 parameter (APPNAME) and attach it to deployment configuration. This is needed to allow Grading pipeline to identify type of active running applications and determine whether Apps in production are switching correctly. We also need to attach previously created MongoDB ConfigMap to backend apps (FrontEnd app doesn't need that). Change names accordingly and repeat for all apps in all projects:

```
oc -n ${GUID}-parks-dev create configmap mlbparks-conf --from-
literal=APPNAME="MLB Parks (Dev)"
oc -n ${GUID}-parks-dev set env dc/mlbparks --
from=configmap/parksdb-conf
oc -n ${GUID}-parks-dev set env dc/mlbparks --
from=configmap/mlbparks-conf
```

7. Configure post deployment hook to populate MongoDB database with needed initial app data once deployment has finished. Change names accordingly and repeat for all apps in all projects:

```
oc -n ${GUID}-parks-dev set deployment-hook dc/mlbparks –post
 -- curl -s http://mlbparks:8080/ws/data/load/
```

8. Expose services (this step will create routes to services in OpenShift):

For development project – exposing only frontend service:

```
oc -n ${GUID}-parks-dev expose svc/parksmap
```

For production project – exposing all three services:

```
oc -n ${GUID}-parks-prod expose svc/parksmap-green --name
parksmap
oc -n ${GUID}-parks-prod expose svc/mlbparks-green --name
mlbparks
oc -n ${GUID}-parks-prod expose svc/nationalparks-green --name
nationalparks
```

5. CI/CD Pipeline: Development stages

Jenkins Pipeline is defined as a Jenkinsfile written in Groovy, commonly stored in a repo for versioning purposes and normally consists of stages defining distinct pipeline processes.

MLBParks Pipeline URL:
https://github.com/kyarovoy/openshift_advdev_homework/blob/master/MLBParks/Jenkinsfile
NationalParks Pipeline URL:
https://github.com/kyarovoy/openshift_advdev_homework/blob/master/Nationalparks/Jenkinsfile
ParksMap Pipeline URL:
https://github.com/kyarovoy/openshift_advdev_homework/blob/master/ParksMap/Jenkinsfile

Our pipelines will have the following development stages:

1. Builds Java source code using Nexus artifact repository as Maven proxy cache. Resuling JAR/WAR will be stored in target directory ready for subsequent stages:

```
  def mvnCmd = "mvn -s ./nexus_settings.xml"

echo "Copy modified nexus_settings.xml to current directory"
sh "sed 's/GUID/${GUID}/' ../nexus_settings.xml >
./nexus_settings.xml"


// Build app
stage("Build") {
    sh "${mvnCmd} clean package -DskipTests"
}
```

…

```
  Nexus settings file contents:

<?xml version="1.0"?>
<settings>
  <mirrors>
    <mirror>
      <id>Nexus</id>
      <name>Nexus Public Mirror</name>
      <url>http://nexus3.GUID-
nexus.svc.cluster.local:8081/repository/maven-all-public/</url>
      <mirrorOf>*</mirrorOf>
    </mirror>
  </mirrors>
  <servers>
    <server>
      <id>nexus</id>
      <username>admin</username>
      <password>admin123</password>
    </server>
  </servers>
</settings>
```

2. Runs unit tests and code coverage tests in parallel. Code coverage tests results will be stored in SonarQube with projectName tagged as "applicationName-applicationVersion-JenkinsBuildNumber". Application name and version will be extracted from app pom.xml file:

```
def appName = getArtifactIdFromPom("pom.xml")
def appVer = getVersionFromPom("pom.xml")
def devTag  = "${appVer}-${BUILD_NUMBER}"

stage('Run Tests') {
  parallel (
    "Run Unit tests": {
      sh "${mvnCmd} test"
    },
    "Run code coverage tests": {
      def SONAR_URL = "http://sonarqube.${GUID}-sonarqube.svc:9000"
      sh "${mvnCmd} sonar:sonar -Dsonar.host.url=${SONAR_URL} -
Dsonar.projectName=${appName}-${devTag}"
    }
  )
}
```

Results are presented in the build Log:

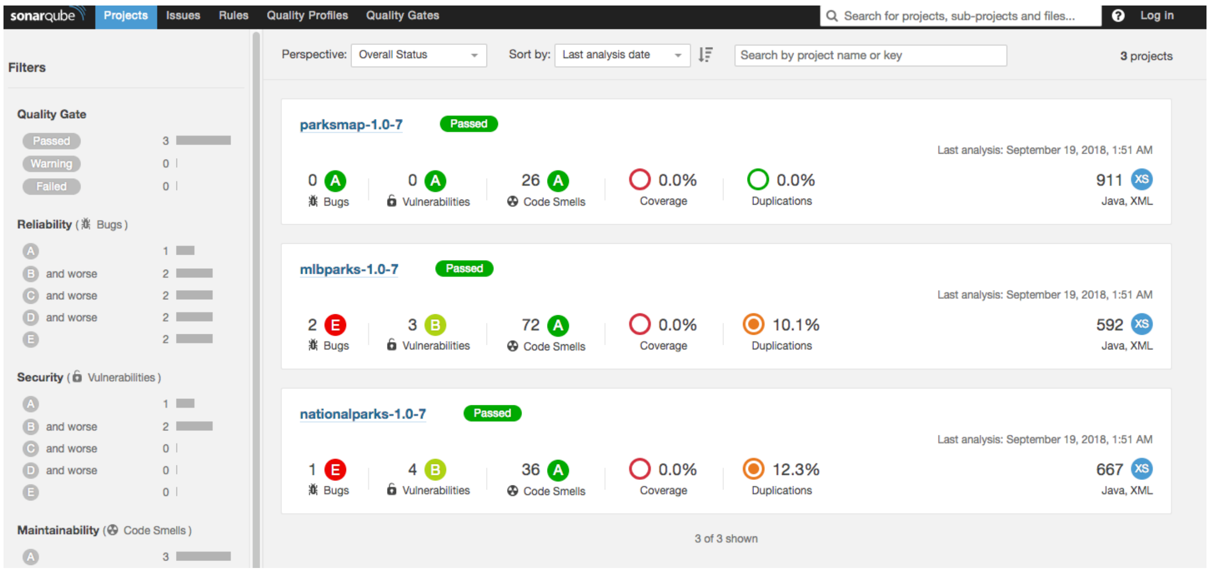
```
[Pipeline] [Run Unit tests] { (Branch: Run Unit tests)
[Pipeline] [Run code coverage tests] { (Branch: Run code coverage tests)
[Pipeline] [Run Unit tests] sh
[Run Unit tests] [MLBParks] Running shell script
[Pipeline] [Run code coverage tests] sh
[Run Unit tests] + mvn -s ./nexus_settings.xml test
[Run Unit tests] Picked up JAVA_TOOL_OPTIONS: -
XX:+UnlockExperimentalVMOptions -XX:+UseCGroupMemoryLimitForHeap -
Dsun.zip.disableMemoryMapping=true
[Run code coverage tests] [MLBParks] Running shell script
[Run code coverage tests] + mvn -s ./nexus_settings.xml sonar:sonar -
Dsonar.host.url=http://sonarqube.ki-sonarqube.svc:9000 -
Dsonar.projectName=mlbparks-1.0-7
…
[Run Unit tests] [INFO] No tests to run.
[Run Unit tests] [INFO] ---------------------------------------------------
---------------------
[Run Unit tests] [INFO] BUILD SUCCESS
[Run Unit tests] [INFO] ---------------------------------------------------
---------------------
[Run Unit tests] [INFO] Total time: 16.793 s
[Run Unit tests] [INFO] Finished at: 2018-09-19T05:50:49+00:00
[Run code coverage tests] [INFO] User cache: /home/jenkins/.sonar/cache
[Run code coverage tests] [INFO] SonarQube version: 6.7.4
[Run code coverage tests] [INFO] Default locale: "en_US", source code
encoding: "UTF-8"
[Run code coverage tests] [INFO] Publish mode
[Run code coverage tests] [INFO] Load global settings
[Run code coverage tests] [INFO] Load global settings (done) | time=310ms
[Run code coverage tests] [INFO] Server id: AWXumfC75F_CnW_pJtEB
[Run code coverage tests] [INFO] User cache: /home/jenkins/.sonar/cache
[Run code coverage tests] [INFO] Load plugins index
[Run code coverage tests] [INFO] Load plugins index (done) | time=66ms
```

```
[Run code coverage tests] [INFO] Download sonar-flex-plugin-2.3.jar
[Run code coverage tests] [INFO] Download sonar-csharp-plugin-
6.5.0.3766.jar
[Run code coverage tests] [INFO] Download sonar-javascript-plugin-
3.2.0.5506.jar
[Run code coverage tests] [INFO] Download sonar-java-plugin-
4.15.0.12310.jar
[Run code coverage tests] [INFO] Download sonar-php-plugin-2.11.0.2485.jar
[Run code coverage tests] [INFO] Download sonar-python-plugin-
1.8.0.1496.jar
[Run code coverage tests] [INFO] Download sonar-scm-git-plugin-
1.3.0.869.jar
[Run code coverage tests] [INFO] Download sonar-scm-svn-plugin-
1.6.0.860.jar
[Run code coverage tests] [INFO] Download sonar-typescript-plugin-
1.1.0.1079.jar
[Run code coverage tests] [INFO] Download sonar-xml-plugin-1.4.3.1027.jar
[Run code coverage tests] [INFO] Process project properties
[Run code coverage tests] [INFO] Load project repositories
[Run code coverage tests] [INFO] Load project repositories (done) |
time=294ms
[Run code coverage tests] [INFO] Load quality profiles
[Run code coverage tests] [INFO] Load quality profiles (done) | time=30ms
[Run code coverage tests] [INFO] Load active rules
[Run code coverage tests] [INFO] Load active rules (done) | time=1081ms
[Run code coverage tests] [INFO] Load metrics repository
[Run code coverage tests] [INFO] Load metrics repository (done) | time=98ms
[Run code coverage tests] [INFO] Project key:
com.openshift.evg.roadshow:mlbparks
[Run code coverage tests] [INFO] -------------  Scan mlbparks-1.0-7
[Run code coverage tests] [INFO] Load server rules
[Run code coverage tests] [INFO] Load server rules (done) | time=33ms
[Run code coverage tests] [INFO] Base dir: /home/jenkins/workspace/ki-
jenkins/ki-jenkins-mlbparks-pipeline/MLBParks
[Run code coverage tests] [INFO] Working dir: /home/jenkins/workspace/ki-
jenkins/ki-jenkins-mlbparks-pipeline/MLBParks/target/sonar
[Run code coverage tests] [INFO] Source paths: src/main/webapp, pom.xml,
src/main/java
[Run code coverage tests] [INFO] Source encoding: UTF-8, default locale:
en_US
[Run code coverage tests] [INFO] Index files
[Run code coverage tests] [INFO] 13 files indexed
[Run code coverage tests] [INFO] Quality profile for java: Sonar way
[Run code coverage tests] [INFO] Quality profile for xml: Sonar way
[Run code coverage tests] [INFO] Sensor JavaSquidSensor [java]
[Run code coverage tests] [INFO] Configured Java source version
(sonar.java.source): 8
[Run code coverage tests] [INFO] JavaClasspath initialization
[Run code coverage tests] [INFO] JavaClasspath initialization (done) |
time=96ms
[Run code coverage tests] [INFO] JavaTestClasspath initialization
[Run code coverage tests] [INFO] JavaTestClasspath initialization (done) |
time=1ms
[Run code coverage tests] [INFO] Java Main Files AST scan
[Run code coverage tests] [INFO] 10 source files to be analyzed
[Run code coverage tests] [INFO] 0/10 files analyzed, current file:
/home/jenkins/workspace/ki-jenkins/ki-jenkins-mlbparks-
pipeline/MLBParks/src/main/java/com/openshift/evg/roadshow/db/MongoDBConnec
tion.java
[Run code coverage tests] [INFO] 7/10 files analyzed, current file:
/home/jenkins/workspace/ki-jenkins/ki-jenkins-mlbparks-
```

```
pipeline/MLBParks/src/main/java/com/openshift/evg/roadshow/rest/MLBParksApp
lication.java
[Run code coverage tests] [INFO] Java Main Files AST scan (done) |
time=25599ms
[Run code coverage tests] [INFO] Java Test Files AST scan
[Run code coverage tests] [INFO] 0 source files to be analyzed
[Run code coverage tests] [INFO] 10/10 source files have been analyzed
[Run code coverage tests] [INFO] Java Test Files AST scan (done) |
time=97ms
[Run code coverage tests] [INFO] Sensor JavaSquidSensor [java] (done) |
time=28600ms
[Run code coverage tests] [INFO] 0/0 source files have been analyzed
[Run code coverage tests] [INFO] Sensor SurefireSensor [java]
[Run code coverage tests] [INFO] parsing [/home/jenkins/workspace/ki-
jenkins/ki-jenkins-mlbparks-pipeline/MLBParks/target/surefire-reports]
[Run code coverage tests] [INFO] Sensor SurefireSensor [java] (done) |
time=4ms
[Run code coverage tests] [INFO] Sensor JaCoCoSensor [java]
[Run code coverage tests] [INFO] Sensor JaCoCoSensor [java] (done) |
time=96ms
[Run code coverage tests] [INFO] Sensor SonarJavaXmlFileSensor [java]
[Run code coverage tests] [INFO] 3 source files to be analyzed
[Run code coverage tests] [INFO] 0/3 files analyzed, current file:
/home/jenkins/workspace/ki-jenkins/ki-jenkins-mlbparks-
pipeline/MLBParks/pom.xml
[Run code coverage tests] [INFO] Sensor SonarJavaXmlFileSensor [java]
(done) | time=12800ms
[Run code coverage tests] [INFO] 3/3 source files have been analyzed
[Run code coverage tests] [INFO] Sensor XML Sensor [xml]
[Run code coverage tests] [INFO] Sensor XML Sensor [xml] (done) |
time=1402ms
[Run code coverage tests] [INFO] Sensor Analyzer for "php.ini" files [php]
[Run code coverage tests] [INFO] Sensor Analyzer for "php.ini" files [php]
(done) | time=2ms
[Run code coverage tests] [INFO] Sensor Zero Coverage Sensor
[Run code coverage tests] [INFO] Sensor Zero Coverage Sensor (done) |
time=496ms
[Run code coverage tests] [INFO] Sensor CPD Block Indexer
[Run code coverage tests] [INFO] Sensor CPD Block Indexer (done) |
time=1100ms
[Run code coverage tests] [INFO] 2 files had no CPD blocks
[Run code coverage tests] [INFO] Calculating CPD for 8 files
[Run code coverage tests] [INFO] CPD calculation finished
[Run code coverage tests] [INFO] Analysis report generated in 1399ms, dir
size=85 KB
[Run code coverage tests] [INFO] Analysis reports compressed in 402ms, zip
size=41 KB
[Run code coverage tests] [INFO] Analysis report uploaded in 199ms
[Run code coverage tests] [INFO] ANALYSIS SUCCESSFUL, you can browse
http://sonarqube.ki-
sonarqube.svc:9000/dashboard/index/com.openshift.evg.roadshow:mlbparks
[Run code coverage tests] [INFO] Note that you will be able to access the
updated dashboard once the server has processed the submitted analysis
report
[Run code coverage tests] [INFO] More about the report processing at
http://sonarqube.ki-sonarqube.svc:9000/api/ce/task?id=AWXwYr3Q5F_CnW_pJt6B
[Run code coverage tests] [INFO] Task total time: 1:02.188 s
[Run code coverage tests] [INFO] ------------------------------------------
------------------------------
[Run code coverage tests] [INFO] BUILD SUCCESS
```

```
[Run code coverage tests] [INFO] ------------------------------------------
------------------------------
[Run code coverage tests] [INFO] Total time: 01:36 min
[Run code coverage tests] [INFO] Finished at: 2018-09-19T05:52:08+00:00
[Run code coverage tests] [INFO] Final Memory: 28M/263M
[Run code coverage tests] [INFO] ------------------------------------------
------------------------------
```

Screenshot of SonarQube UI with code analysis results for all 3 apps:


sonarqube
Projects
Issues
Rules
Quality Profiles
Quality Gates
Search for projects, sub-projects and files...
Log in
Perspective: Overall Status
Sort by: Last analysis date
Search by project name or key

3 projects

Filters

Quality Gate: Passed 3, Warning 0, Failed 0

Reliability (Bugs): A 1, B and worse 2, C and worse 2, D and worse 2, E 2

Security (Vulnerabilities): A 1, B and worse 2, C and worse 0, D and worse 0, E 0

Maintainability (Code Smells): A 3

Project	Quality Gate	Bugs	Vulnerabilities	Code Smells	Coverage	Duplications	Size	Last analysis
parksmap-1.0-7	Passed	0 A	0 A	26 A	0.0%	0.0%	911 XS Java, XML	September 19, 2018, 1:51 AM
mlbparks-1.0-7	Passed	2 E	3 B	72 A	0.0%	10.1%	592 XS Java, XML	September 19, 2018, 1:51 AM
nationalparks-1.0-7	Passed	1 E	4 B	36 A	0.0%	12.3%	667 XS Java, XML	September 19, 2018, 1:51 AM

3 of 3 shown

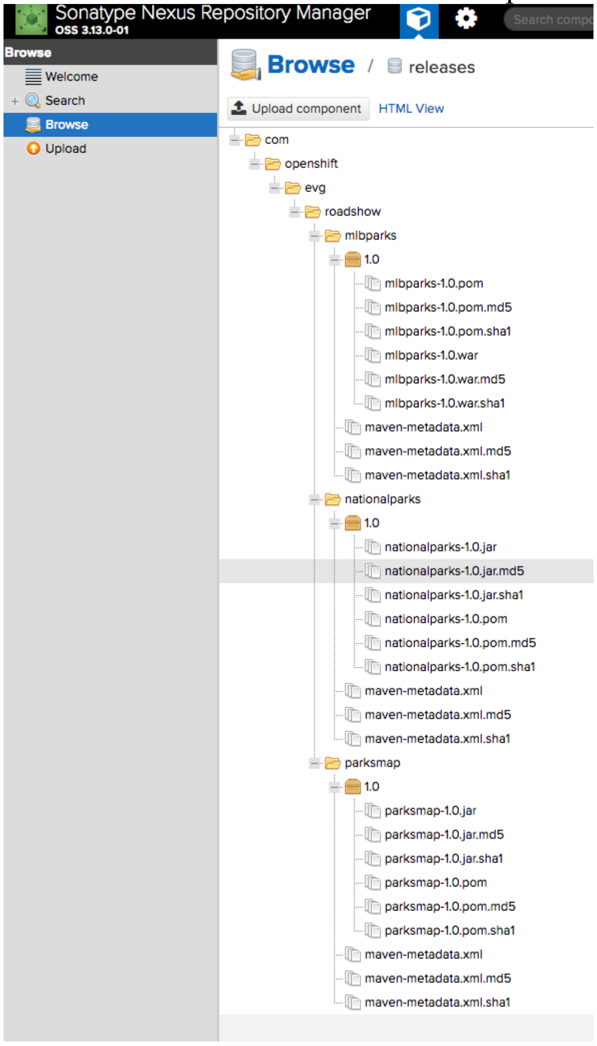
3. Builds OpenShift container image and tags it with Development tag (appName-appVersion-JenkinsBuildNumber). This stage will start previously pre-configured OpenShift binary build and send JAR/WAR file (which has been build in stage #1) as a binary to be included in resulting container image. In this stage we will show that instead of running OC shell command to send command to OpenShift we can use existing methods provided by Jenkins OpenShift Plugin (e.g. openshiftTag in this case):

```
      def devTag  = "${appVer}-${BUILD_NUMBER}"
      def devProj = "${GUID}-parks-dev"

        stage('Build image') {
          echo "Building OpenShift container image ${appName}:${devTag}"
          sh "oc -n ${devProj} start-build ${appName} --from-
file=./target/${appName}.war --follow"
          // tag image with version and build Number (devTag)
          openshiftTag  alias: 'false',
                        srcStream: appName,
                        destStream: appName,
                        srcTag: 'latest',
                        destTag: devTag,
                        namespace: devProj,
                        destinationNamespace: devProj,
                        verbose: 'false'
      }
```

4. Deploy built artifacts to Nexus:

```
      stage('Publish artifact to Nexus') {
          def NEXUS_URL = "http://nexus3-${GUID}-
nexus.apps.${CLUSTER}/repository/releases"
          sh "${mvnCmd} deploy -DskipTests=true -
DaltDeploymentRepository=nexus::default::${NEXUS_URL}"
      }
```

Screenshot of Nexus maven artifacts repo contents:



Sonatype Nexus Repository Manager
OSS 3.13.0-01

Browse
- Welcome
- Search
- Browse
- Upload

Browse / releases

Upload component   HTML View

```
com
  openshift
    evg
      roadshow
        mlbparks
          1.0
            mlbparks-1.0.pom
            mlbparks-1.0.pom.md5
            mlbparks-1.0.pom.sha1
            mlbparks-1.0.war
            mlbparks-1.0.war.md5
            mlbparks-1.0.war.sha1
          maven-metadata.xml
          maven-metadata.xml.md5
          maven-metadata.xml.sha1
        nationalparks
          1.0
            nationalparks-1.0.jar
            nationalparks-1.0.jar.md5
            nationalparks-1.0.jar.sha1
            nationalparks-1.0.pom
            nationalparks-1.0.pom.md5
            nationalparks-1.0.pom.sha1
          maven-metadata.xml
          maven-metadata.xml.md5
          maven-metadata.xml.sha1
        parksmap
          1.0
            parksmap-1.0.jar
            parksmap-1.0.jar.md5
            parksmap-1.0.jar.sha1
            parksmap-1.0.pom
            parksmap-1.0.pom.md5
            parksmap-1.0.pom.sha1
          maven-metadata.xml
          maven-metadata.xml.md5
          maven-metadata.xml.sha1
```

5. Deploy applications in Development project and wait until they deploy succesfully. In this stage we will re-use previously pre-configured OpenShift application deployment configuration, patch it’s image setting to point to previously built app container image and initiate manual deployment using openshiftDeploy plugin function. Alternatively we can also use “oc rollout dc/name” shell command:

```
      stage('Deploy to Dev') {
          echo "Deploying image to DevProj"
          sh "oc -n ${devProj} set image dc/${appName} ${appName}=docker-
registry.default.svc:5000/${devProj}/${appName}:${devTag}"
          openshiftDeploy namespace: devProj, depCfg: appName,  verbose:
'false', waitTime: '', waitUnit: 'sec'
          openshiftVerifyDeployment namespace: devProj, depCfg: appName,
replicaCount: '1', verbose: 'false', verifyReplicaCount: 'false', waitTime:
'', waitUnit: 'sec'
```

```
          openshiftVerifyService namespace: devProj, svcName: appName,
verbose: 'false'
      }
```

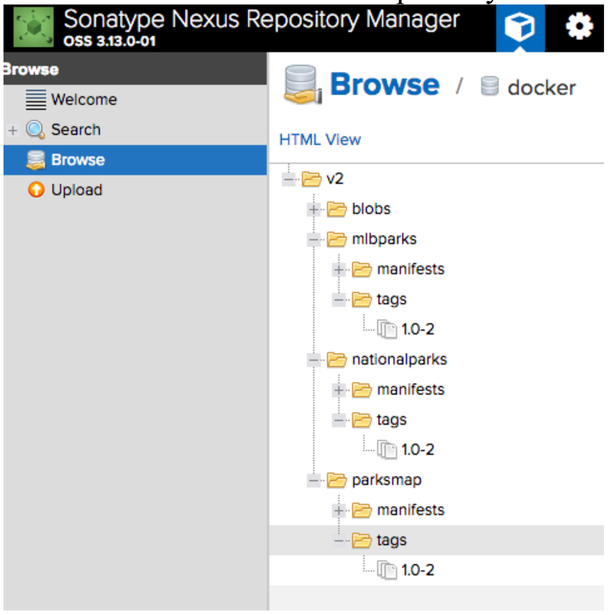
6. Runs demo integration test by sending a GET CURL request to backend application and expecting to get HTTP 200 return code. "/ws/data/all" request is expected to return data from MongoDB (which is inserted there by app post-deployment hook, configured on earlier stages). This stage is only present in Backend-related application pipelines:

```
      stage('Integration Tests') {
          sh "curl -s -o /dev/null -w '%{http_code}\n'
http://${appName}.${devProj}.svc:8080/ws/data/all | fgrep 200"
      }
```

7. If integration tests pass - copies app container image from OpenShift docker registry to our own Nexus3 docker registry using Skopeo tool:

```
      stage('Copy Image to Nexus Docker Registry') {
          sh """skopeo copy \
                --src-tls-verify=false \
                --dest-tls-verify=false \
                --src-creds openshift:\$(oc whoami -t) \
                --dest-creds admin:admin123 \
                docker://docker-
registry.default.svc.cluster.local:5000/${devProj}/${appName}:${devTag} \
                docker://nexus3-registry.${GUID}-
nexus.svc.cluster.local:5000/${appName}:${devTag}
             """
      }
```

Screenshot of Nexus docker repository contents:



```
v2
  blobs
  mlbparks
    manifests
    tags
      1.0-2
  nationalparks
    manifests
    tags
      1.0-2
  parksmap
    manifests
    tags
      1.0-2
```

6. CI/CD Pipeline: deployment stages

Deployment pipeline is actually a set of stages in app pipeline:

8. Tags app container image as “appVersion” which identifies app as ready for production deployments:

```
def prodProj = "${GUID}-parks-prod"
def prodTag = "${appVer}"

stage('Tag Image for production') {
    // Tag image with prod tag
    openshiftTag  alias: 'false',
                  srcStream: appName,
                  destStream: appName,
                  srcTag: devTag,
                  destTag: prodTag,
                  namespace: devProj,
                  destinationNamespace: prodProj,
                  verbose: 'false'
}
```

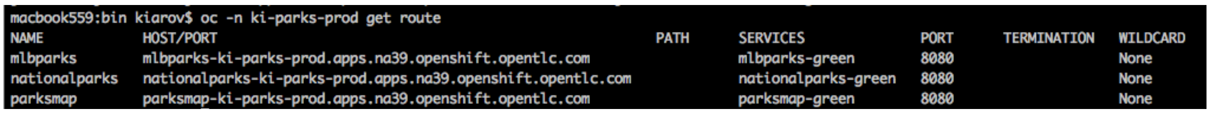
9. Performs blue/green deployment only of a newly built application in production project and wait until they deploy successfully. This stage first tries to understand which type of app is currently deployed (blue/green) to know which deployment config needs to be deployed. In this stage we will re-use previously pre-configured OpenShift application deployment configuration, patch its image setting to point to previously built and tested app container image and initiate manual deployment using openshiftDeploy plugin function. Alternatively, we can also use “oc rollout dc/name” shell command:

```
      stage('Blue/Green deployment to prod') {
          curApp = sh(returnStdout: true, script: "oc -n ${prodProj} get
route ${appName} -o jsonpath='{ .spec.to.name }'").trim()
          if (curApp == "${appName}-green") {
            newApp = "${appName}-blue"
          }
          else {
            newApp = "${appName}-green"
          }

          echo "Active app: ${curApp}"
          echo "New app: ${newApp}"

          // Update prod DC with new image
          sh "oc -n ${prodProj} set image dc/${newApp} ${newApp}=docker-
registry.default.svc:5000/${prodProj}/${appName}:${prodTag}"

          // Deploy non-active app
          openshiftDeploy namespace: prodProj, depCfg: newApp, verbose:
'false', waitTime: '', waitUnit: 'sec'
          openshiftVerifyDeployment namespace: prodProj, depCfg: newApp,
replicaCount: '1', verbose: 'false', verifyReplicaCount: 'true', waitTime:
'', waitUnit: 'sec'
```

```
            openshiftVerifyService namespace: prodProj, svcName: newApp,
verbose: 'false'
        }
```

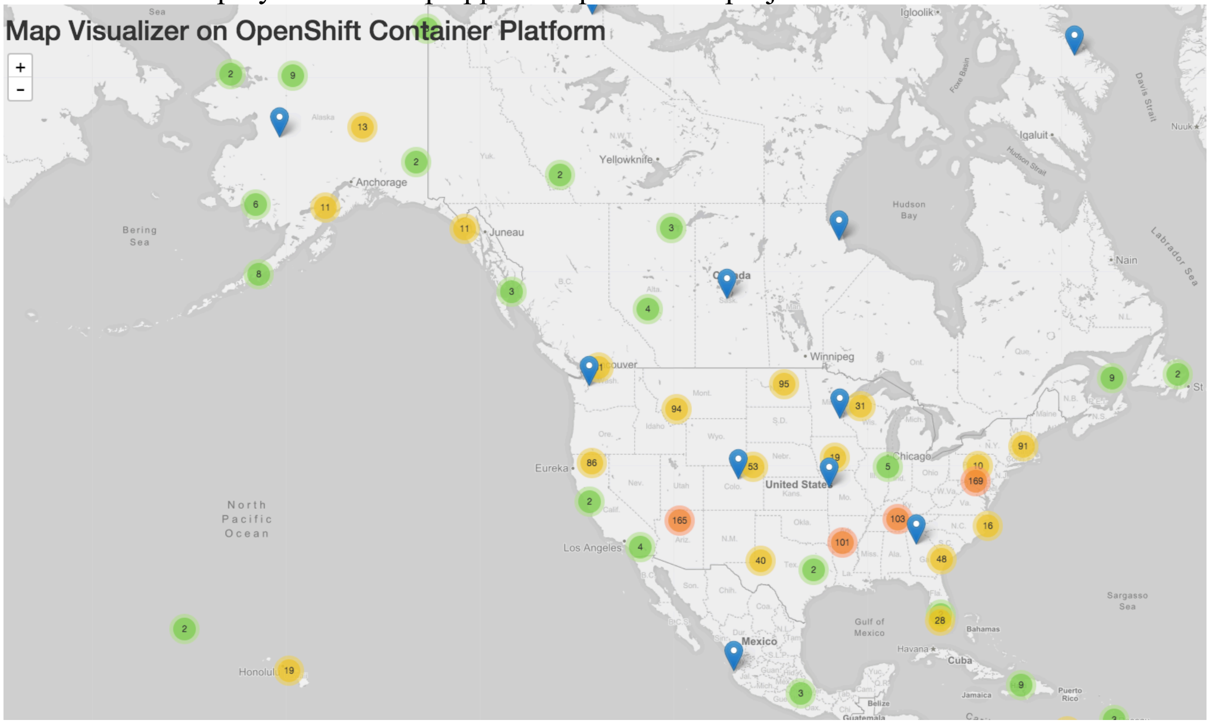
10. Switches the route from an old service to a new service by using “oc patch route” command:

```
        stage('Switch apps in prod') {
            sh "oc -n ${prodProj} patch route ${appName} -p
'{\"spec\":{\"to\":{\"name\":\"${newApp}\"}}}'"
        }
```

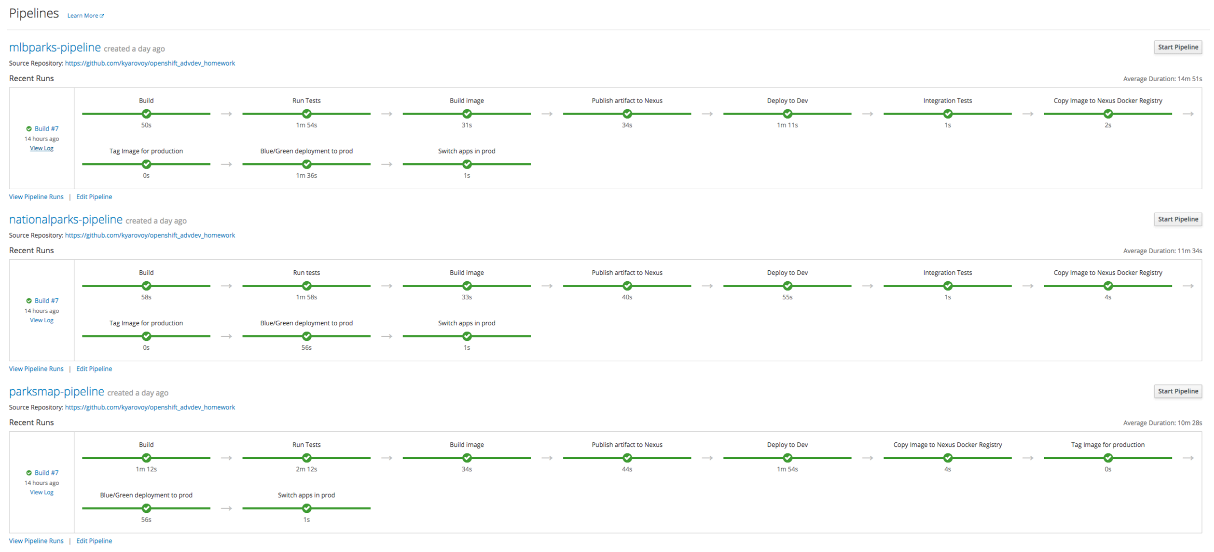


```
macbook559:bin kiarov$ oc -n ki-parks-prod get route
NAME            HOST/PORT                                              PATH   SERVICES              PORT   TERMINATION   WILDCARD
mlbparks        mlbparks-ki-parks-prod.apps.na39.openshift.opentlc.com        mlbparks-green        8080                 None
nationalparks   nationalparks-ki-parks-prod.apps.na39.openshift.opentlc.com   nationalparks-green   8080                 None
parksmap        parksmap-ki-parks-prod.apps.na39.openshift.opentlc.com        parksmap-green        8080                 None
```

Screenshot of deployed ParksMap app UI in production project:



Screenshot of successful Jenkins pipelines' execution when running Homework Grading automation:



Pipelines Learn More

mlbparks-pipeline created a day ago

Source Repository: https://github.com/kyarovoy/openshift_advdev_homework

Recent Runs

Average Duration: 14m 51s

Build #7
14 hours ago
View Log

Build 50s → Run Tests 1m 54s → Build image 31s → Publish artifact to Nexus 34s → Deploy to Dev 1m 11s → Integration Tests 1s → Copy Image to Nexus Docker Registry 2s → Tag Image for production 0s → Blue/Green deployment to prod 1m 36s → Switch apps in prod 1s

View Pipeline Runs | Edit Pipeline

nationalparks-pipeline created a day ago

Source Repository: https://github.com/kyarovoy/openshift_advdev_homework

Recent Runs

Average Duration: 11m 34s

Build #7
14 hours ago
View Log

Build 58s → Run tests 1m 58s → Build image 33s → Publish artifact to Nexus 40s → Deploy to Dev 55s → Integration Tests 1s → Copy Image to Nexus Docker Registry 4s → Tag Image for production 0s → Blue/Green deployment to prod 56s → Switch apps in prod 1s

View Pipeline Runs | Edit Pipeline

parksmap-pipeline created a day ago

Source Repository: https://github.com/kyarovoy/openshift_advdev_homework

Recent Runs

Average Duration: 10m 28s

Build #7
14 hours ago
View Log

Build 1m 12s → Run Tests 2m 12s → Build image 34s → Publish artifact to Nexus 44s → Deploy to Dev 1m 54s → Copy Image to Nexus Docker Registry 4s → Tag Image for production 0s → Blue/Green deployment to prod 56s → Switch apps in prod 1s

View Pipeline Runs | Edit Pipeline