**RED HAT SERVICES**

**MitziCom Openshift POC**



Table of Contents

1.0 Document Summary 2

2.0 Environment 3

2.1 Git 3

2.2 POC resources 4

2.3 POC urls 4

3.0 POC High Level Deployment Diagram 5

4.0 CI/CD Infrastructure Setup. 6

4.1 Nexus setup 6

The following repositories shown above are now available. 7

4.2 Gogs setup 8

4.3 SonarQube setup 11

4.4 Jenkins setup 12

5.0 OpenShift Setup 15

6.0 Development Pipeline 26

7.0 Deployment Pipeline 34

**Document Control**

## 

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Reason for update** |
| V1.0 | 11/06/2018 | Joe Kenny | Document Creation |

# Document Summary

The purpose of the POC is to determine the feasibility of using OpenShift as a target for an existing Java-based microservices workload.

It demonstrates a fully integrated CI/CD pipeline using Gogs source code control system, a Nexus artifact repository, SonarQube code analysis tool and is deployed to production in a blue-green strategy orchestrated by Jenkins.

The application used in the POC is an application with three microservices—two back-end services and one front-end service calling the back-end services.

# 2.0 Environment

The OPENTLC shared environment was used to host this POC.

<https://master.na37.openshift.opentlc.com/>

Name: OTLC-LAB-jkenny-redhat.com-PROD\_SHARED\_DEVELOPER\_ENV-632f

Description: OPENTLC OpenShift 3.7 Shared Access

Management Engine GUID: 8c9e176a-6892-11e8-8f36-001a4a16015a

Lifecycle: Retirement Date 06/16/18

**Note**: Due to resource constraints, in order to avoid the following memory quota limit:

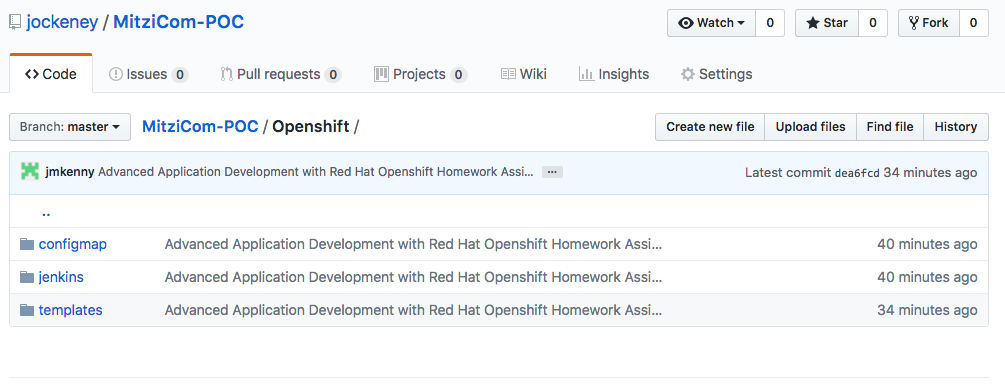
*Error creating deployer pod: pods ~" is forbidden: exceeded quota: clusterquota-jkenny-redhat.com-20a0, requested: requests.memory=256Mi, used: requests.memory=6Gi, limited: requests.memory=6Gi*

the Development project resources can not be active at the same time as the Production project resources. Additional resources have been requested so this may not be an issue when viewing the POC.

## 2.1 Git

The Openshift templates, Jenkins pipeline scripts and configmaps used throughout the POC are publicly available on Github:

<https://github.com/jockeney/MitziCom-POC>



## 2.2 POC resources

Here is a list of the Openshift Projects created to manage the POC:

|  |  |  |
| --- | --- | --- |
| **Project Name** | **Display Name** | **Comment** |
| mitzicom-gogs | MitziCom Gogs GitHub Repository. | Gogs is our open source Git repository for the POC. |
| mitzicom-nexus | MitziCom Nexus Repository. | Nexus manages software "artifacts" required for developing the POC. |
| mitzicom-sonarqube | MitziCom SonarQube code quality tool. | This tool provides code quality analysis. |
| mitzicom-jenkins | MitziCom Jeknins CI/CD | The development/production pipelines are managed by Jenkins. |
| mitzicom-tasks-dev | MitziCom Dev Runtime Environment | The development runtime environment for the POC’s 3 microservices. |
| mitzicom-tasks-prod | MitziCom Prod Runtime Environment | The production runtime environment for the POC’s 3 microservices. |

## 2.3 POC urls

|  |  |
| --- | --- |
| **Project Name** | **Credentials** |
| gogs-jmk.apps.na37.openshift.opentlc.com/poc | poc/admin |
| nexus.apps.na37.openshift.opentlc.com | admin/admin123 |
| sonarqube-mitzicom-sonarqube.apps.na37.openshift.opentlc.com/projects | admin/admin |
| jenkins-mitzicom-jenkins.apps.na37.openshift.opentlc.com |  |

# 3.0 POC High Level Deployment Diagram

# 4.0 CI/CD Infrastructure Setup.

## 4.1 Nexus setup

Setup up Nexus 3 (Repository Manager) with persistent storage from a template.

*$ oc new-project mitzicom-nexus --display-name "MitziCom Nexus Repository"*

The git repo contains an OpenShift template for deploying Sonatype Nexus 3 and pre-configuring Red Hat and JBoss maven repositories on Nexus via post deploy hooks. You can modify the post hook in the templates and add other Nexus repositories if required:

post:

execNewPod:

containerName: ${SERVICE\_NAME}

command:

- "/bin/bash"

- "-c"

- "add scripts here"

**Import Templates**

In order to add this template to the OpenShift project run the following commands:

*$ oc create -f <https://raw.githubusercontent.com/jockeney/MitziCom-POC/master/Openshift/templates/mitzicom_nexus_template.yaml> -n mitzicom-nexus*

**Deploy Nexus 3**

Deploy Sonatype Nexus 3 using the provided templates:

*$ oc new-app mitzicom-nexus3-persistent --param=HOSTNAME=nexus3-jmk.apps.na37.openshift.opentlc.com --param=MAX\_MEMORY=1.5Gi*

The template comes with additional parameters, use the –-param to add:

parameters

NEXUS\_VERSION=2.14.3

HOSTNAME

SERVICE\_NAME

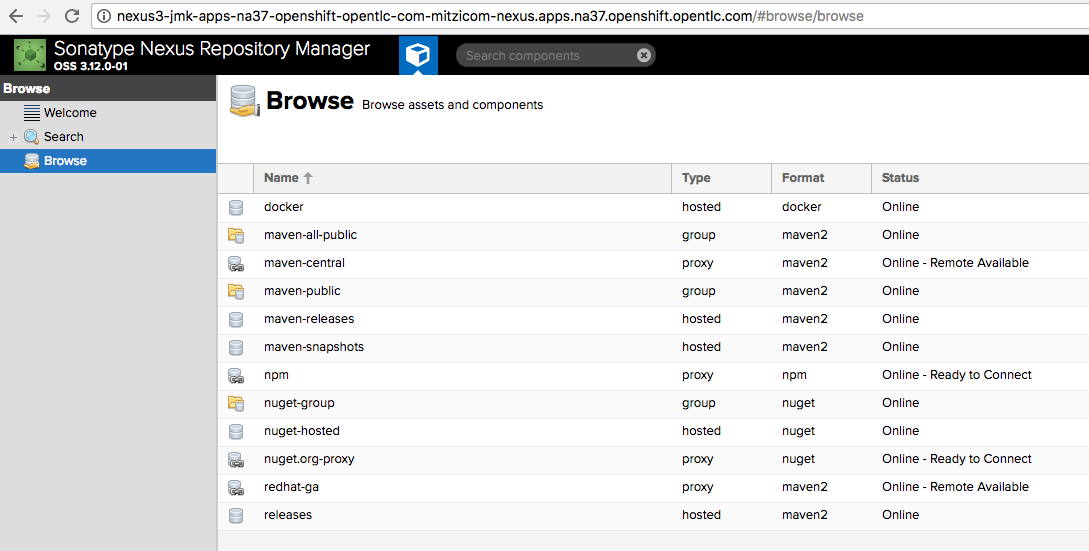
VOLUME\_CAPACITY (default 4Gi)

MAX\_MEMORY (default 2Gi)

The template also comes preconfigured with a persistent volume claim (PVC) nexus-pvc mounted at /nexus-data and liveness and readiness probes for Nexus.

Access the nexus application via route ['nexus3-jmk-apps-na37-openshift-opentlc-com-mitzicom-nexus.apps.na37.openshift.opentlc.com](nexus3-jmk-apps-na37-openshift-opentlc-com-mitzicom-nexus.apps.na37.openshift.opentlc.com)'

Internal service: <http://nexus3.mitzicom-nexus.svc.cluster.local:8081/repository/>



## The following repositories shown above are now available.4.2 Gogs setup

Setup up Gogs (Source code Repository Manager) with persistent storage from a template.

Gogs is the Go Git service see <https://gogs.io/> This open source GitHub clone can be deployed in a local infrastructure. It requires a PostgreSQL or MySQL database with persistent storage as well as a persistent volume to store its own data.

$ *oc new-project mitzicom-gogs --display-name "MitziCom Gogs GitHub Repository"*

**Import Templates**

oc create -f <https://raw.githubusercontent.com/jockeney/MitziCom-POC/master/Openshift/templates/mitzicom_gogs_persistent_template.yaml>-n mitzicom-gogs

This persistent template comes preconfigured with two persistent volume claims (PVC’s), gogs-pvc mounted at /data and postgresql pvc mounted at /var/lib/pgsql/data. These persistent volumes come available with default size required of 1Gi and the volume size can be specified with the template variables: GOGS\_VOLUME\_CAPACITY and DB\_VOLUME\_CAPACITY.

The template also comes preconfigured with a ConfigMap. Gogs writes it's configuration to a file on the local container and this configuration file needs to be saved in persistent storage, so a ConfigMap is used to provide this. This ConfigMap is mounted as a volume at /opt/gogs/custom/conf and has the following definition :

- kind: ConfigMap

apiVersion: v1

metadata:

name: gogs-config

labels:

app: ${APPLICATION\_NAME}

data:

app.ini: |

RUN\_MODE = prod

RUN\_USER = gogs

[database]

DB\_TYPE = postgres

HOST = ${APPLICATION\_NAME}-postgresql:5432

NAME = ${DATABASE\_NAME}

USER = ${DATABASE\_USER}

PASSWD = ${DATABASE\_PASSWORD}

[repository]

ROOT = /opt/gogs/data/repositories

[server]

ROOT\_URL=http://${HOSTNAME}

SSH\_DOMAIN=${HOSTNAME}

[security]

INSTALL\_LOCK = ${INSTALL\_LOCK}

[service]

ENABLE\_CAPTCHA = false

[webhook]

SKIP\_TLS\_VERIFY = ${SKIP\_TLS\_VERIFY}

In addition the template also provide some customization via paramters, the following are available:

parameters

APPLICATION\_NAME (default gogs)

HOSTNAME

GOGS\_VOLUME\_CAPACITY (default 4Gi)

DB\_VOLUME\_CAPACITY (default 1Gi)

DATABASE\_USER (default gogs)  
Database Password (default gogs)

DATABASE\_NAME (default gogs)

DATABASE\_ADMIN\_PASSWORD

GOGS\_VERSION (default 0.9.97)

INSTALL\_LOCK (default true, meaning the installation (/install) page will be disabled. Set to false if you want to run the installation wizard via web')

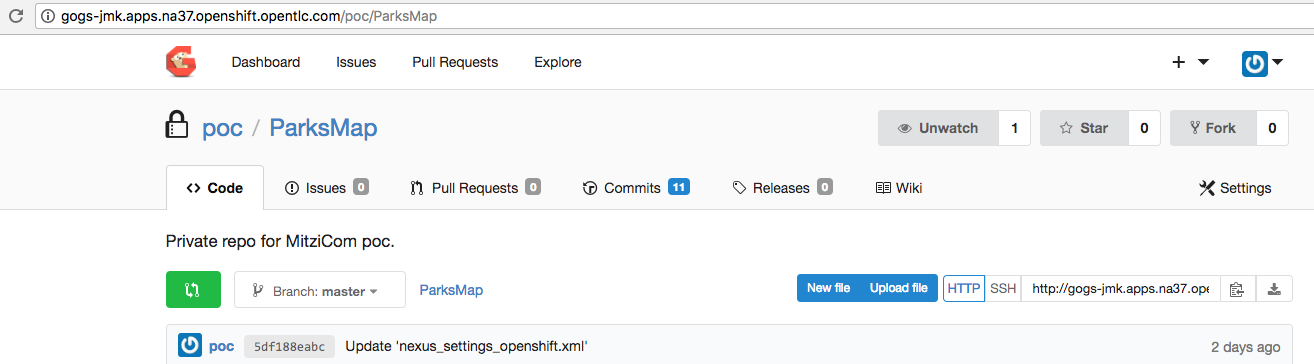
SKIP\_TLS\_VERIFY (default false, skip TLS verification on webhooks)

Finally the template also come preconfigured with liveness and readiness probes for Gogs.

**Deploy Gogs**

*$ oc new-app mitzicom-gogs-persistent --param=HOSTNAME=gogs-jmk.apps.na37.openshift.opentlc.com*

Access the application via route http://gogs-jmk.apps.na37.openshift.opentlc.com



Register a new user, noting the first registered user becomes the administrator for Gogs.

**Install Source Code into Gogs**

We want the ParksMap source code for the 3 microservices available in a private Gogs repository, so the following needs to performed:

New Repository - ParksMap (private)

*$ git clone https://github.com/wkulhanek/ParksMap.git*

*$ cd ParksMap*

*$ edit nexus\_settings.xml (note: Set up nexus\_settings.xml for local builds, making sure that <url> points the installed Nexus URL)*

*$ git add .*

*$ git commit -m "first commit"*

*$ git remote add gogsPoc http://poc:admin@gogsjmk.apps.na37.openshift.opentlc.com/poc/ParksMap.git*

*$ git push -u gogsPoc master*

note: Make sure to replace <gogs\_user> and <gogs\_password> with your credentials

## 4.3 SonarQube setup

Setup up SonarQube (code analysis) with persistent storage from a template.

*$ oc new-project mitzicom-sonarqube --display-name "MitziCom SonarQube code quality tool"*

**Import Templates**

In order to add this templates to OpenShift project run the following commands:

oc create -f <https://raw.githubusercontent.com/jockeney/MitziCom-POC/master/Openshift/templates/mitzicom_sonarqube_persistent_template.yaml> -n mitzicom-sonarqube

The template comes preconfigured with a persistent volume claim (PVC) postgresql-sonarqube-data mounted at /var/lib/pgsql/data and liveness and readiness probes for SonarQube.

**Deploy SonarQube**

Deploy SonarQube using the provided templates:

*$ oc new-app mitzicom-sonarqube-persistent --param=SONARQUBE\_VERSION=6.7*

The template comes with additional parameters, use the –-param to add:

parameters

SONARQUBE\_VERSION (default 6.7)

POSTGRESQL\_PASSWORD

POSTGRESQL\_VOLUME\_CAPACITY (default 1Gi)

SONAR\_VOLUME\_CAPACITY (default 1Gi)

Access the application once deployed at <http://sonarqube-mitzicom-sonarqube.apps.na37.openshift.opentlc.com> using the credentials admin/admin.

## 4.4 Jenkins setup

Setup up Jenkins with persistent storage from a template.

*$ oc new-project mitzicom-jenkins --display-name "MitziCom Jeknins CI/CD"*

**Import Templates**

In order to add this template to OpenShift project run the following commands:

oc create -f <https://raw.githubusercontent.com/jockeney/MitziCom-POC/master/Openshift/templates/mitzicom_jenkins_persistent_template.yaml> -n mitzicom-jenkins

**Deploy Jenkins**

Deploy Jenkins using the provided templates:

*$ oc new-app mitzicom-jenkins-persistent --param=MEMORY\_LIMIT=2Gi --param=VOLUME\_CAPACITY=4Gi --param=ENABLE\_OAUTH=true -n mitzicom-jenkins*

The template comes with additional parameters, use the –-param to add:

parameters

JENKINS\_SERVICE\_NAME (default jenkins)

JNLP\_SERVICE\_NAME (default jenkins-jnlp)

ENABLE\_OAUTH (default true)

MEMORY\_LIMIT (default 512Mi)

VOLUME\_CAPACITY (default 1Gi)

JENKINS\_IMAGE\_STREAM\_TAG (default jenkins:2)

The template also comes preconfigured with a persistent volume claim (PVC) jenkins mounted at /var/lib/jenkins and liveness and readiness probes for Jenkins.

**Setup Jenkins**

The stock Jenkins Maven slave pod does not have Skopeo installed. Skopeo is a tool that allows you to move your built container images into another registry. To build this custom slave image perform the following:

// Build a new slave Image and tag it in the Openshift registry

*$ wget https://raw.githubusercontent.com/jockeney/MitziCom-POC/master/Openshift/scripts/Dockerfile*

This downloads a Dockerfile which is using docker.io/openshift/jenkins-slave-maven-centos7:v3.9 as the base image and installs Skopeo on top of it.

On your local VM add the Openshift Container Registry to /etc/containers/registries.conf

*$ sudo systemctl restart docker*

*$ mkdir jenkins-slave-appdev*

*$ sudo docker login -u jkenny-redhat.com -p $(oc whoami -t) docker-registry-default.apps.na37.openshift.opentlc.com (test connection)*

*// Build and tag Custom Slave Image - note -t <repository\_name>/<image\_name>:<tag>*

*$ sudo docker build . -t docker-registry-default.apps.na37.openshift.opentlc.com/mitzicom-jenkins/jenkins-slave-maven-appdev:v3.9*

*$ sudo docker images*

*$ sudo docker push docker-registry-default.apps.na37.openshift.opentlc.com/mitzicom-jenkins/jenkins-slave-maven-appdev:v3.9*

Now the Custom Slave Image with Skopeo is built, tagged and available in the Openshift Container registry, we can now use this Slave image in Jenkins.

// SLAVE POD REGISTER IN JENKINS -

Manage Jenkins -> Configure System -> Cloud -> Add Pod Template -> Kubernetes Pod Template

Docker image:docker-registry.default.svc:5000/mitzicom-jenkins/jenkins-slave-maven-appdev:v3.9

Labels: 'maven-custom-slavepod'

// SLAVE POD TEST IN JENKINS

create a Pipeline, Test skopeo with node('maven-dev')

node('maven-custom-slavepod') {

stage('Test skopeo') {

sh('skopeo --version')

sh('oc whoami')

}

}

when you run the pipleline you should see this message in the console:

*// Pulled Successfully pulled image "docker-registry.default.svc:5000/mitzicom-jenkins/jenkins-slave-maven-appdev:v3.9"*

The final test of CI/CD Infrastructure involves testing your Local VM builds to verify all the build tools are working as expected:

*// Test the Local Build environment in the VM*

$ *oc project mitzicom-gogs*

$ *cd ParksMap*

*// Test MLBParks backend application*

$ *cd mlbparks/*

$ *mvn -s ../nexus\_settings.xml clean package -DskipTests=true*

This will create a war file target/mlbparks.jar that can be used in a binary build for OpenShift. The S2I Builder image to use is jboss-eap70-openshift:1.7.

*// Nationalparks backend application*

$ *mvn -s ../nexus\_settings.xml clean package -Dmaven.test.skip=true*

This will create a jar file target/nationalparks.jar that can be used in a binary build for OpenShift. The S2I Builder image to use is redhat-openjdk18-openshift:1.2.

*// Parksmap application (This Spring Boot application is a frontend web gateway to backend services that provide geolocation data on services)*

$ *cd parksmap*

$ *mvn -s ../nexus\_settings.xml clean package spring-boot:repackage -DskipTests -Dcom.redhat.xpaas.repo.redhatga*

*see https://github.com/wkulhanek/ParksMap/tree/master/mlbparks*

This will create a jar file target/parksmap.jar that can be used in a binary build for OpenShift. The S2I Builder image to use is redhat-openjdk18-openshift:1.2.

As part of this verification you need to confirm the output of the build to verify that the local Maven dependencies come from Nexus and not the public Internet repository. Post the build step, check Nexus maven-all-public to confirm dependencies have been added.

# 5.0 OpenShift Setup

The next part of the POC invloves creating two Openshift projects for development and production, these projects will be the runtime for the output of a successful pipeline build. Both projects will have MongoDB persistence, with the addition of a stateful set for production with three replicas.

Both projects will have include the necessary build and deployment configurations, including ConfigMaps for each microservice. Both projects will also grant the necessary permissions to Jenkins to allow it to manipulate objects in these environments.

**Development setup**

*oc new-project mitzicom-tasks-dev --display-name "MitziCom Dev Runtime Environment"*

// Setup a MongoDB database in Dev (used empheral template with)

• MongoDB Connection Username: mitzicomuser

• MongoDB Connection Password: redhat

• MongoDB Database Name: mitzicomdb

// Set up the permissions for Jenkins to be able to manipulate objects in the mitzicom-tasks-dev project

*$ oc policy add-role-to-user edit system:serviceaccount:mitzicom-jenkins:jenkins -n mitzicom-tasks-dev*

**NOTE**: Two backend services providing geospatial data about Nationalparks and Major League Baseball park. The backend services are exposed as routes with label "type=parksmap-backend".

**Configure the MLBParks backend application**

The following needs to be performed:

* Create a binary build using the jboss-eap70-openshift:1.6image stream.
* Create a new deployment configuration pointing to mlbparks:0.0-0.
* Turn off automatic building and deployment.
* Expose the deployment configuration as a service (on port 8080) and the service as a route.
* Create a ConfigMap
* Attach the ConfigMap to the deployment configuration

*$ oc new-build --binary=true --name="mlbparks" jboss-eap70-openshift:1.6 -n mitzicom-tasks-dev*

*$ oc new-app mitzicom-tasks-dev/mlbparks:0.0-0 --name=mlbparks --allow-missing-imagestream-tags=true -n mitzicom-tasks-dev*

*$ oc set triggers dc/mlbparks --remove-all -n mitzicom-tasks-dev*

*$ oc expose dc mlbparks --port 8080 -n mitzicom-tasks-dev*

*$ oc expose svc mlbparks -n mitzicom-tasks-dev -l type=parksmap-backend*

The ParksMap microservices require configuration data e.g. Database name. These configuration artifacts should be externalized form the application. ConfigMaps allow you to decouple configuration artifacts from image content to keep containerized applications portable.

The ConfigMap object in OpenShift provides mechanisms to provide configuration data to the application container, it can be used to store key-value properties, configuration files, JSON blobs and alike. You can create a ConfigMap by pointing at a file containing the application configuration The configuration files for the microservices is available on github. First download these files locally:

*$ wget <https://raw.githubusercontent.com/jockeney/MitziCom-POC/master/Openshift/configmap/mlbparks.ini>*

*$ oc create configmap config --from-file=mlbparks.ini -n mitzicom-tasks-dev*

Use envFrom to define all of the ConfigMap’s data as Pod environment variables. The key from the ConfigMap becomes the environment variable name in the Pod.

*$ oc describe configmap config*

*Name: config*

*Namespace: mitzicom-tasks-dev*

*Labels: <none>*

*Annotations:<none>*

*Data*

*====*

*mlbparks.ini:*

*----*

*DB\_HOST=mitzicomdb*

*DB\_PORT=27017*

*DB\_USERNAME=mitzicomdb*

*DB\_PASSWORD=redhat*

*DB\_NAME=parks*

*APPNAME=MLB Parks*

*Events: <none>*

Edit the deployment configuration to wire in the configMap’s data as environment variables:

$ *oc edit dc mlbparks -o json*

*"env": [*

*{*

*"name": "DB\_HOST",*

*"valueFrom": {*

*"configMapKeyRef": {*

*"name": "config",*

*"key": "mlbparks.ini"*

*}*

*}*

*},*

*{*

*"name": "DB\_PORT",*

*"valueFrom": {*

*"configMapKeyRef": {*

*"name": "config",*

*"key": "mlbparks.ini"*

*}*

*}*

*},*

*{*

*"name": "DB\_USERNAME",*

*"valueFrom": {*

*"configMapKeyRef": {*

*"name": "config",*

*"key": "mlbparks.ini"*

*}*

*}*

*},*

*{*

*"name": "DB\_PASSWORD",*

*"valueFrom": {*

*"configMapKeyRef": {*

*"name": "config",*

*"key": "mlbparks.ini"*

*}*

*}*

*},*

*{*

*"name": "DB\_NAME",*

*"valueFrom": {*

*"configMapKeyRef": {*

*"name": "config",*

*"key": "mlbparks.ini"*

*}*

*}*

*},*

*{*

*"name": "APPNAME",*

*"valueFrom": {*

*"configMapKeyRef": {*

*"name": "config",*

*"key": "mlbparks.ini"*

*}*

*}*

*}*

*]*

**Configure the Nationalparks backend application**

The following needs to be performed:

* Create a binary build using the redhat-openjdk18-openshift:1.2image stream.
* Create a new deployment configuration pointing to nationalparks:0.0-0.
* Turn off automatic building and deployment.
* Expose the deployment configuration as a service (on port 8080) and the service as a route.
* Create a ConfigMap
* Attach the ConfigMap to the deployment configuration

// Set up the Dev Nationalparks backend application

*$ oc new-build --binary=true --name="nationalparks" redhat-openjdk18-openshift:1.2 -n mitzicom-tasks-dev*

*$ oc new-app mitzicom-tasks-dev/nationalparks:0.0-0 --name=nationalparks --allow-missing-imagestream-tags=true -n mitzicom-tasks-dev*

*$ oc set triggers dc/nationalparks --remove-all -n mitzicom-tasks-dev*

*$ oc expose dc nationalparks --port 8080 -n mitzicom-tasks-dev*

*$ oc expose svc nationalparks -n mitzicom-tasks-dev -l type=parksmap-backend*

*$ wget <https://raw.githubusercontent.com/jockeney/MitziCom-POC/master/Openshift/configmap/nationalParks.ini>*

*$ oc create configmap nationalparksconfig --from-file=nationalParks.ini -n mitzicom-tasks-dev*

Again edit the deployment configuration to wire in the configMap’s data as environement variables:

*$ oc edit dc nationalparks -o json*

"env": [

{

"name": "DB\_HOST",

"valueFrom": {

"configMapKeyRef": {

"name": "nationalparksconfig",

"key": "nationalParks.ini"

}

}

},

{

"name": "DB\_PORT",

"valueFrom": {

"configMapKeyRef": {

"name": "nationalparksconfig",

"key": "nationalParks.ini"

}

}

},

{

"name": "DB\_USERNAME",

"valueFrom": {

"configMapKeyRef": {

"name": "nationalparksconfig",

"key": "nationalParks.ini"

}

}

},

{

"name": "DB\_PASSWORD",

"valueFrom": {

"configMapKeyRef": {

"name": "nationalparksconfig",

"key": "nationalParks.ini"

}

}

},

{

"name": "DB\_NAME",

"valueFrom": {

"configMapKeyRef": {

"name": "nationalparksconfig",

"key": "nationalParks.ini"

}

}

},

{

"name": "APPNAME",

"valueFrom": {

"configMapKeyRef": {

"name": "nationalparksconfig",

"key": "nationalParks.ini"

}

}

}

]

**Configure the Parksmap** **frontend application**

The following needs to be performed:

* Create a binary build using the redhat-openjdk18-openshift:1.2image stream.
* Create a new deployment configuration pointing to *parksmap*:0.0-0.
* Turn off automatic building and deployment.
* Expose the deployment configuration as a service (on port 8080) and the service as a route.

// Set up the Dev Parksmap frontend application (This is a Spring Boot application)

*$ oc new-build --binary=true --name="parksmap" redhat-openjdk18-openshift:1.2 -n mitzicom-tasks-dev*

*$ oc new-app mitzicom-tasks-dev/parksmap:0.0-0 --name=parksmap --allow-missing-imagestream-tags=true -n mitzicom-tasks-dev*

*$ oc set triggers dc/parksmap --remove-all -n mitzicom-tasks-dev*

*$ oc expose dc parksmap --port 8080 -n mitzicom-tasks-dev*

*$ oc expose svc parksmap -n mitzicom-tasks-dev*

**Production setup**

*$ oc new-project mitzicom-tasks-prod --display-name "MitziCom Prod Runtime Environment"*

The production project requires a stateful set with 3 replicas. We set up a StatefulSet for MongoDB that consists of three copies of the MongoDB database that replicate the data amongst themselves. This requires three pods and three persistent volume claims. It also requires a headless service for the pods of the set to communicate, as well as a regular service for clients to connect to the database. Install the following template to create it:

//Setup a StatefulSet for MongoDB in Prod

*$ oc create -f <https://raw.githubusercontent.com/jockeney/MitziCom-POC/master/Openshift/templates/mitzicom_statefulset_template.yaml>(note 3.7 apiVersion: apps/v1beta1)*

The StatefulSet requires a headless service

$ oc process -f <https://raw.githubusercontent.com/jockeney/MitziCom-POC/master/Openshift/templates/mitzicom_headlessservice_template.yaml>| oc create -f -

// Allow the production project to pull images from development project

*$ oc policy add-role-to-group system:image-puller system:serviceaccounts:mitzicom-tasks-prod -n mitzicom-tasks-dev*

// Set up the permissions for Jenkins to be able to manipulate objects in the production project

*$ oc policy add-role-to-user edit system:serviceaccount:mitzicom-jenkins:jenkins -n mitzicom-tasks-prod*

As before we now setup the 3 microservices, but this time we configure it to use a Blue/Green deployment. Using blue-green deployment, you serve one application at a time and switch from one application to the other.

**Configure the MlBParks Blue** **backend application**

The following needs to be performed:

* Create a binary build using the jboss-eap70-openshift:1.6image stream.
* Create a new deployment configuration pointing to *mlbparks*:0.0-0.
* Turn off automatic building and deployment.
* Expose the deployment configuration as a service (on port 8080) and the service as a route.
* Create a ConfigMap
* Attach the ConfigMap to the deployment configuration

*$ oc new-app mitzicom-tasks-dev/mlbparks:0.0 --name=mlbparks-blue --allow-missing-imagestream-tags=true -n mitzicom-tasks-prod*

*$ oc set triggers dc/mlbparks-blue --remove-all -n mitzicom-tasks-prod*

*$ oc expose dc mlbparks-blue --port 8080 -n mitzicom-tasks-prod*

*$ oc create configmap mlbparks-blue-config --from-file=mlbparks.ini -n mitzicom-tasks-prod*

*$ oc edit dc mlbparks-blue -o json (note key= mlbparks-blue-config for configmap)*

# Expose Blue service as route to make blue application active

*$ oc expose svc/mlbparks-blue --name mlbparks-blue -n mitzicom-tasks-prod -l type=parksmap-backend*

**Configure the MLBParks Green** **backend application**

The following needs to be performed:

* Create a binary build using the jboss-eap70-openshift:1.6image stream.
* Create a new deployment configuration pointing to *mlbparks*:0.0-0.
* Turn off automatic building and deployment.
* Expose the deployment configuration as a service (on port 8080) and the service as a route.
* Create a ConfigMap
* Attach the ConfigMap to the deployment configuration

*$ oc new-app mitzicom-tasks-dev/mlbparks:0.0 --name=mlbparks-green --allow-missing-imagestream-tags=true -n mitzicom-tasks-prod*

*$ oc set triggers dc/mlbparks-green --remove-all -n mitzicom-tasks-prod*

*$ oc expose dc mlbparks-green --port 8080 -n mitzicom-tasks-prod*

*$ oc create configmap mlbparks-green-config --from-file=mlbparks.ini -n mitzicom-tasks-prod*

*$ oc edit dc mlbparks-green -o json (note key= mlbparks-green-config for configmap)*

**Configure the NationalParks Blue** **backend application**

The following needs to be performed:

* Create a binary build using the redhat-openjdk18-openshift:1.2image stream.
* Create a new deployment configuration pointing to *nationalparks*:0.0-0.
* Turn off automatic building and deployment.
* Expose the deployment configuration as a service (on port 8080) and the service as a route.
* Create a ConfigMap
* Attach the ConfigMap to the deployment configuration

*$ oc new-app mitzicom-tasks-dev/nationalparks:0.0 --name=nationalparks-blue --allow-missing-imagestream-tags=true -n mitzicom-tasks-prod*

*$ oc set triggers dc/nationalparks-blue --remove-all -n mitzicom-tasks-prod*

*$ oc expose dc nationalparks-blue --port 8080 -n mitzicom-tasks-prod*

*$ oc create configmap nationalparks-blue-config --from-file=nationalParks.ini -n mitzicom-tasks-prod*

*$ oc edit dc nationalparks-blue -o json (note key= nationalparks-blue-config for configmap)*

# Expose Blue service as route to make blue application active

*$ oc expose svc/nationalparks-blue --name nationalparks-blue -n mitzicom-tasks-prod -l type=parksmap-backend*

**Configure the Nationalparks Green** **backend application**

The following needs to be performed:

* Create a binary build using the redhat-openjdk18-openshift:1.2image stream.
* Create a new deployment configuration pointing to *nationalparks*:0.0-0.
* Turn off automatic building and deployment.
* Expose the deployment configuration as a service (on port 8080) and the service as a route.
* Create a ConfigMap
* Attach the ConfigMap to the deployment configuration

*$ oc new-app mitzicom-tasks-dev/nationalparks:0.0 --name=nationalparks-green --allow-missing-imagestream-tags=true -n mitzicom-tasks-prod*

*$ oc set triggers dc/nationalparks-green --remove-all -n mitzicom-tasks-prod*

*$ oc expose dc nationalparks-green --port 8080 -n mitzicom-tasks-prod*

*$ oc create configmap nationalparks-green-config --from-file=nationalParks.ini -n mitzicom-tasks-prod*

*$ oc edit dc nationalparks-green -o json (note key= nationalparks-green-config for configmap)*

**Configure the parksMap Blue** **frontend application**

The following needs to be performed:

* Create a binary build using the redhat-openjdk18-openshift:1.2image stream.
* Create a new deployment configuration pointing to *parksmap*:0.0-0.
* Turn off automatic building and deployment.
* Expose the deployment configuration as a service (on port 8080) and the service as a route.

*$ oc new-app mitzicom-tasks-dev/parksmap:0.0 --name=parksmap-blue --allow-missing-imagestream-tags=true -n mitzicom-tasks-prod*

*$ oc set triggers dc/parksmap-blue --remove-all -n mitzicom-tasks-prod*

*$ oc expose dc parksmap-blue --port 8080 -n mitzicom-tasks-prod*

*# Expose Blue service as route to make blue application active*

*$ oc expose svc/parksmap-blue --name parksmap-blue -n mitzicom-tasks-prod*

**Configure the parksMap Green** **frontend application**

*$ oc new-app mitzicom-tasks-dev/parksmap:0.0 --name=parksmap-green --allow-missing-imagestream-tags=true -n mitzicom-tasks-prod*

*$ oc set triggers dc/parksmap-green --remove-all -n mitzicom-tasks-prod*

*$ oc expose dc parksmap-green --port 8080 -n mitzicom-tasks-prod*

# 6.0 Development Pipeline

The development pipleline performs the following stages:

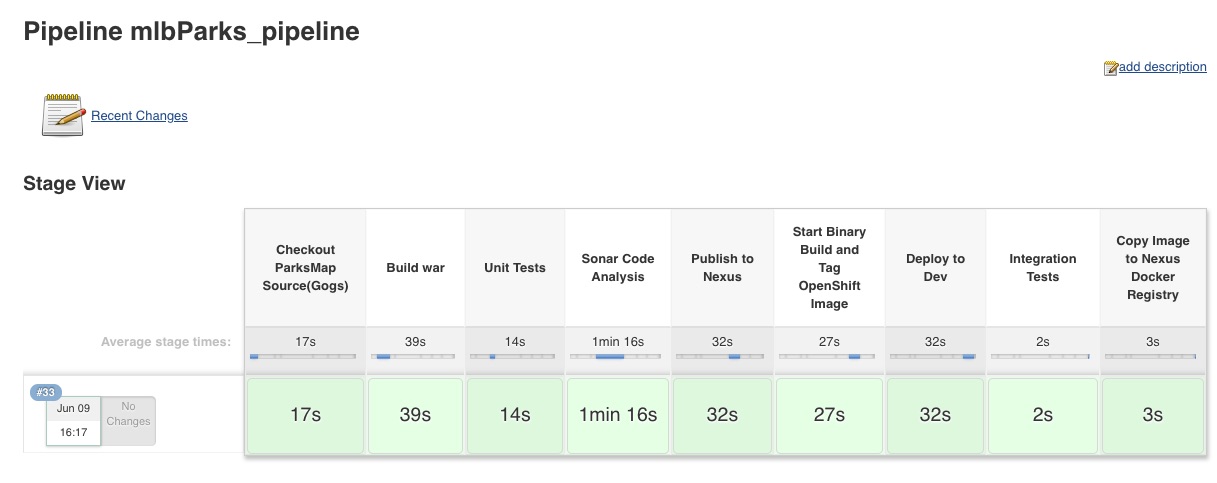
* Builds the source code, using the artifact repository as a Maven proxy cache.
* Executes the Unit tests
* Executes coverage tests
* Tags the image with the version and build number
* Upload the generated artifact to the Nexus artifact repository
* Runs an integration test if applicable i.e a back-end service
* Upload the tested container image to the Nexus Docker registry

The Jenkins console is available at https://jenkins-mitzicom-jenkins.apps.na37.openshift.opentlc.com/

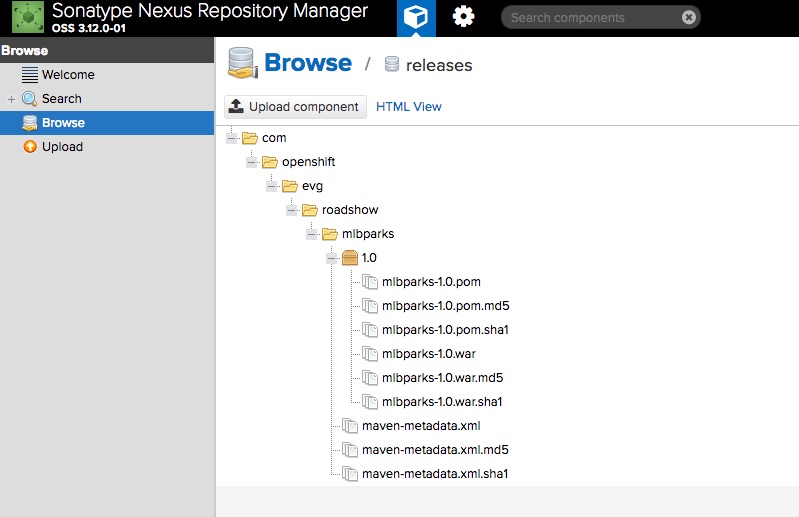
**mlbparks pipeline**

The Jenkins file is available in Gogs under the mlbparks folder in ParksMap source code project and is also available here: <https://raw.githubusercontent.com/jockeney/MitziCom-POC/master/Openshift/jenkins/mlparks_pipeline_script>

The pipeline successfully ran from start to finish:



The Nexus release repository was successfully updated:

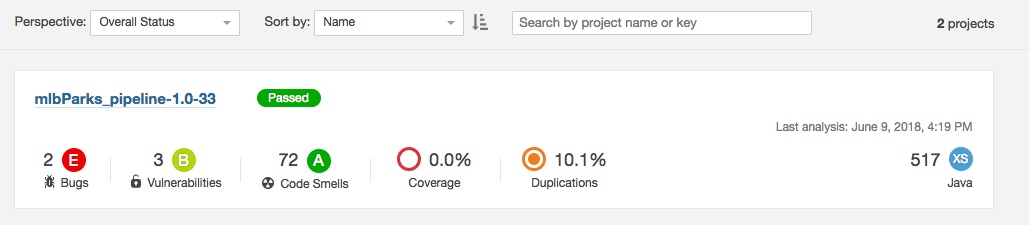


The Unit tests / Coverage tests / Integration test results are available in the Jenkins build log here:

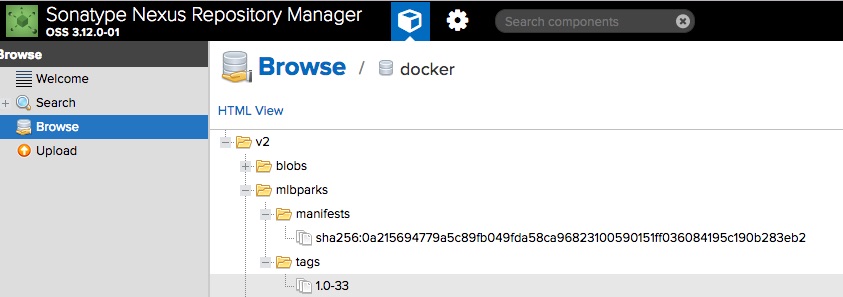
<https://jenkins-mitzicom-jenkins.apps.na37.openshift.opentlc.com/job/mlbParks_pipeline/33/consoleFull>

<https://raw.githubusercontent.com/jockeney/MitziCom-POC/master/Openshift/pipeline_results/mlbparks/pipeline_build_log>

SonarQube was able to apply code analysis on the build artifact:



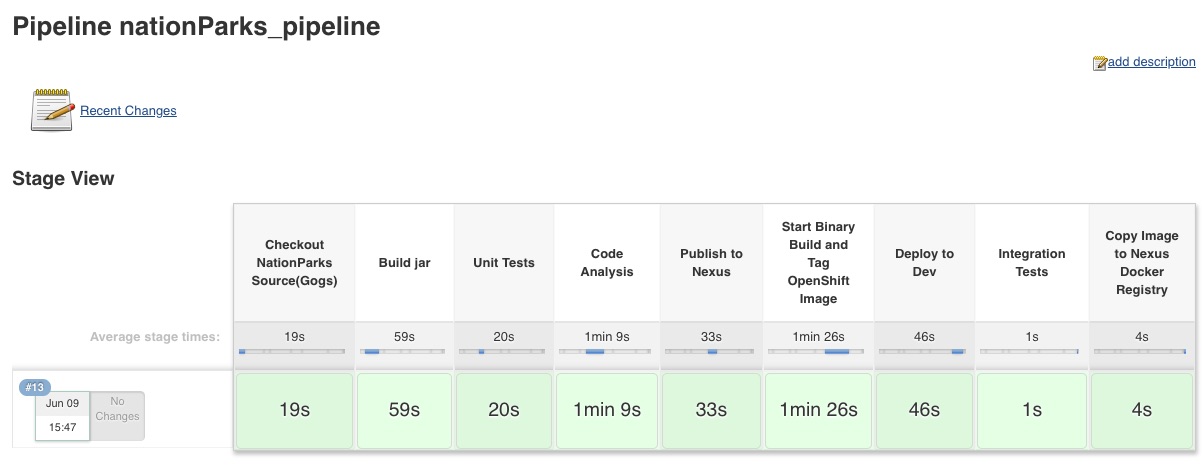
The tested container was successfully uploaded to the Nexus Docker repository:



**nationalparks pipeline**

The Jenkins file is available in Gogs under the nationalparks folder in ParksMap source code and is also available here: <https://raw.githubusercontent.com/jockeney/MitziCom-POC/master/Openshift/jenkins/nationalparks_pipeline_script>

The pipeline successfully ran from start to finish:

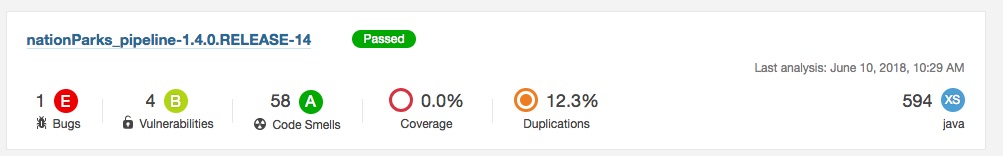


The Unit tests / Coverage tests / Integration test results are available in the Jenkins build log here:

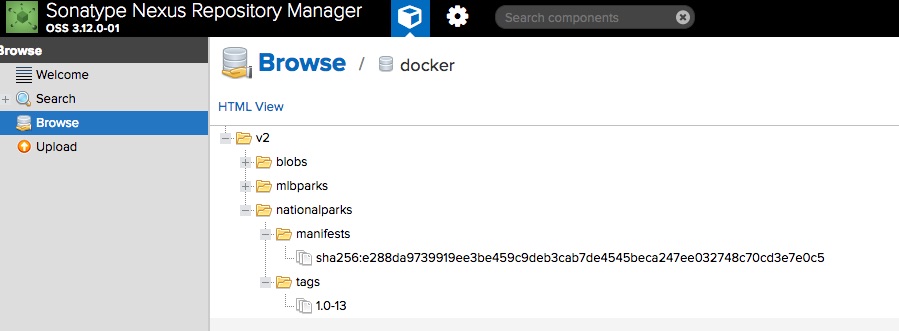
<https://jenkins-mitzicom-jenkins.apps.na37.openshift.opentlc.com/job/nationParks_pipeline/13/consoleFull>

<https://raw.githubusercontent.com/jockeney/MitziCom-POC/master/Openshift/pipeline_results/nationalparks/pipeline_build_log>

SonarQube was able to apply code analysis on the build artifact:



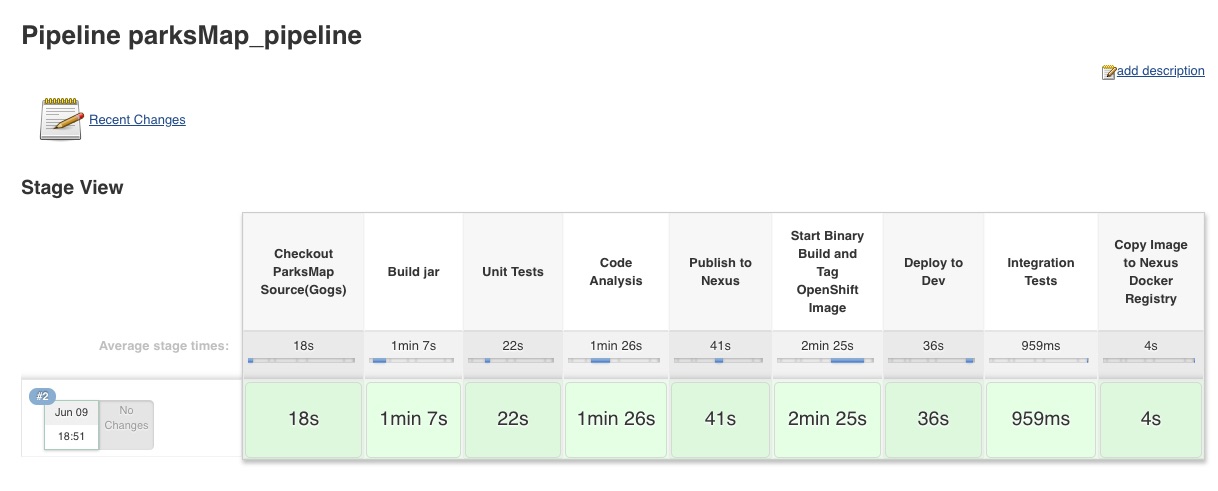
The tested container was successfully uploaded to the Nexus Docker repository:



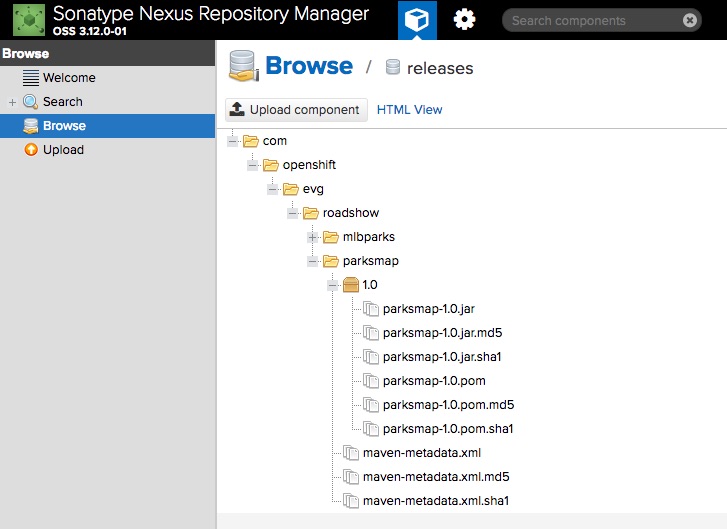
**parksmap pipeline**

The Jenkins file is available in Gogs under the parksmap folder in the ParksMap source code project and is also available here: <https://raw.githubusercontent.com/jockeney/MitziCom-POC/master/Openshift/jenkins/parksmap_pipeline_script>

The pipeline successfully ran from start to finish:



The Nexus release repository was successfully updated:

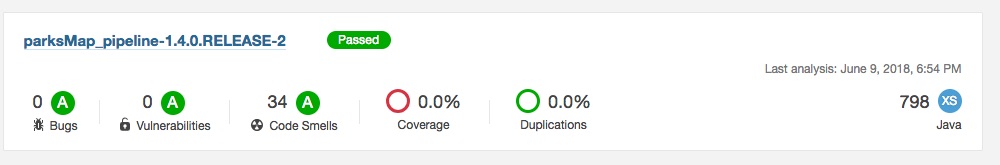


The Unit tests / Coverage tests / Integration test results are available in the Jenkins build log here:

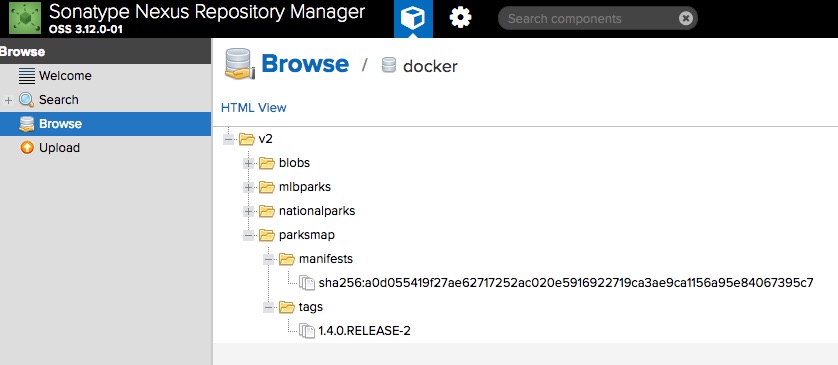
<https://jenkins-mitzicom-jenkins.apps.na37.openshift.opentlc.com/job/parksMap_pipeline/2/consoleFull>

<https://raw.githubusercontent.com/jockeney/MitziCom-POC/master/Openshift/pipeline_results/parksmap/pipeline_build_log>

SonarQube was able to apply code analysis on the build artifact:



The tested container was successfully uploaded to the Nexus Docker repository:



# 7.0 Deployment Pipeline

The Deployment Pipeline included the following additions:

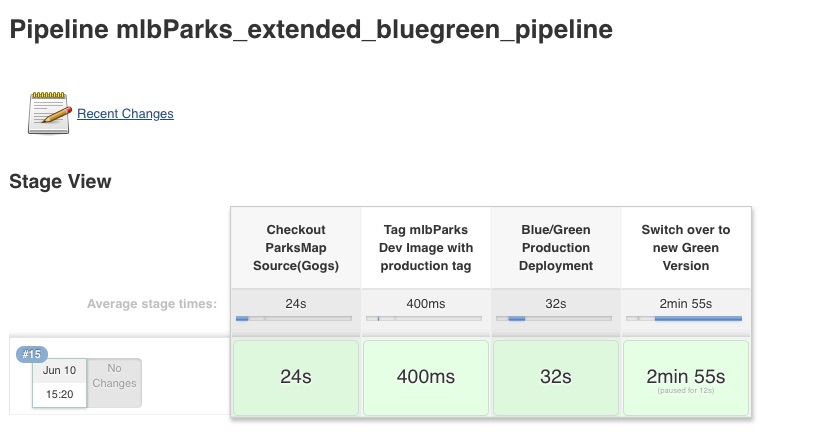
* Tags the image as ‘version’ for production deployment
* Deploys only the newly built microservice using a blue-green strategy
* Waits for approval to execute the final go-live switch
* Jenkins pipeline switches the route to the newly built microservice

**Note**: The Shared OpenTLC Openshift environment did not have sufficient memory resources to run all the CI/CD tooling and the development and production environments simultaneously. Therefore to work around this constraint and still demonstrate a blue/green deployment, I created an extended pipeline just for Production that only worked with the Development resources switched off. As these extended pipelines built upon the original pipeline scripts they expected all the development pipeline stages to have completed successfully.

**mlbparks pipeline extended**

The Jenkins file is available here: <https://raw.githubusercontent.com/jockeney/MitziCom-POC/master/Openshift/jenkins/mlparks_extended_bluegreen_pipeline_script>

The pipeline successfully ran from start to finish:

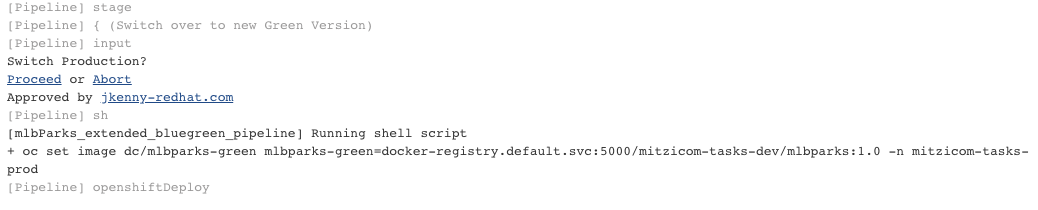


The Jenkins build log is available here:

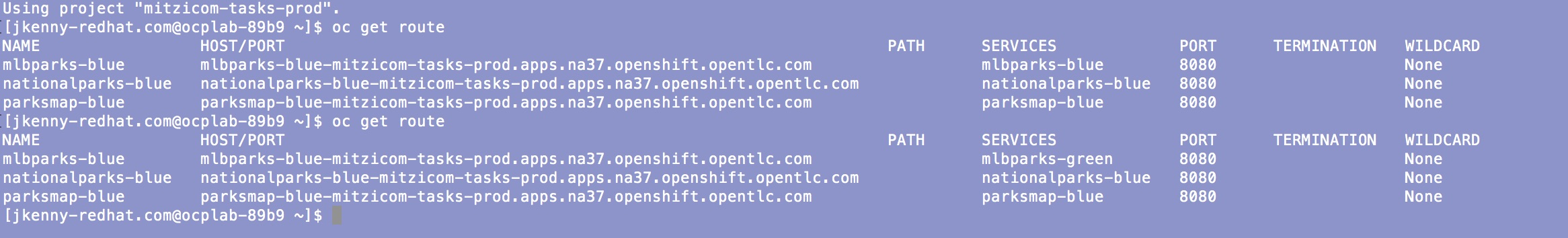
<https://jenkins-mitzicom-jenkins.apps.na37.openshift.opentlc.com/job/mlbParks_extended_bluegreen_pipeline/15/console>

<https://raw.githubusercontent.com/jockeney/MitziCom-POC/master/Openshift/pipeline_results/mlbparks/blue_green/pipeline_build_log>

The pipeline Waits for approval to execute the final go-live switch



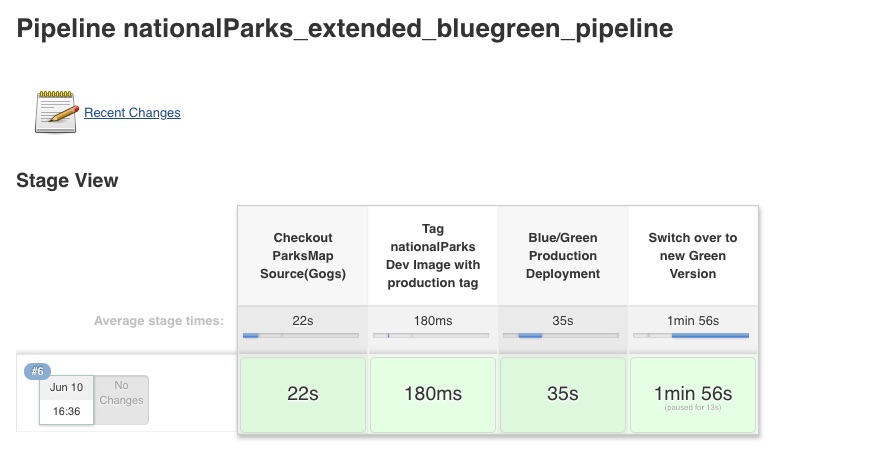
Jenkins pipeline switches the route to the newly built microservice



**nationalparks pipeline extended**

The Jenkins file is available here: <https://raw.githubusercontent.com/jockeney/MitziCom-POC/master/Openshift/jenkins/nationalParks_extended_bluegreen_pipeline_script>

The pipeline successfully ran from start to finish:

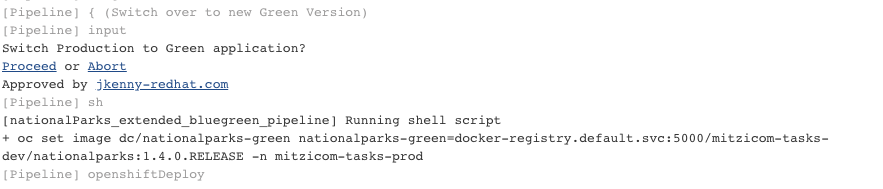


The Jenkins build log is available here:

<https://jenkins-mitzicom-jenkins.apps.na37.openshift.opentlc.com/job/nationalParks_extended_bluegreen_pipeline/6/console>

<https://raw.githubusercontent.com/jockeney/MitziCom-POC/master/Openshift/pipeline_results/mlbparks/blue_green/pipeline_build_log>

The pipeline Waits for approval to execute the final go-live switch



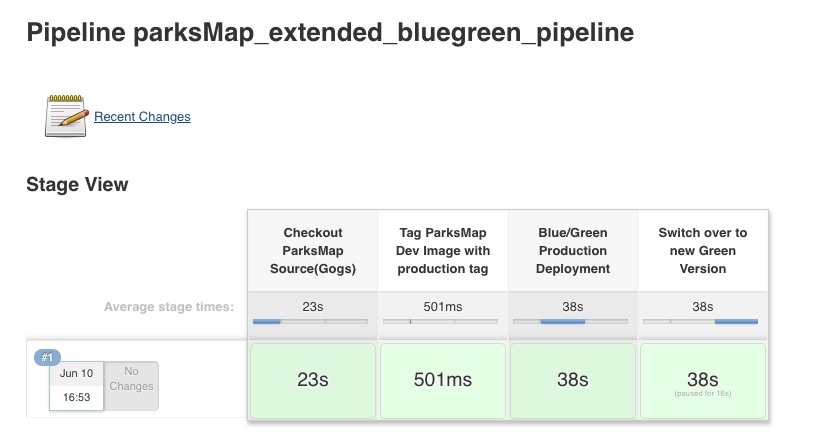
Jenkins pipeline switches the route to the newly built microservice



**parksmap pipeline extended**

The Jenkins file is available here: <https://raw.githubusercontent.com/jockeney/MitziCom-POC/master/Openshift/jenkins/parksMap_extended_bluegreen_pipeline_script>

The pipeline successfully ran from start to finish:



The Jenkins build log is available here:

<https://jenkins-mitzicom-jenkins.apps.na37.openshift.opentlc.com/job/parksMap_extended_bluegreen_pipeline/1/console>

<https://raw.githubusercontent.com/jockeney/MitziCom-POC/master/Openshift/pipeline_results/parksmap/blue_green/pipeline_build_log>

The pipeline Waits for approval to execute the final go-live switch



Jenkins pipeline switches the route to the newly built microservice

