

**Pivotal / VMware KuBo (Kubernetes on Bosh)**

CI/CD Pipeline with Jenkins

Prepared by:

Technical Product Management team

VMware Cloud Native Business Unit

Version: 1.1

**Document releases**

|  |  |
| --- | --- |
| Version | Description |
| 1.0 | Date: 09/07/2017 - Initial Version |

Special thanks to Mike Lloyd and Merlin Glynn for developing the code.

Table of Contents

Table of Contents 3

1. Overview 4

1.1 Acronyms 4

1.2 Useful Links 4

1.3 KuBo Deployment 5

1.4 Workflow 5

2. Lab Configuration 7

2.1 Github 7

2.1.1 Github Repository 7

2.1.2 Webhooks – Jenkins (Github Plugin) 7

2.2 Harbor 8

2.3 Jenkins 10

2.3.1 Install Jenkins Server 10

2.3.2 Configure Jenkins Server 11

2.3.3 Install Docker on Jenkins Server 14

2.3.4 Install Kubectl on Jenkins Server 15

2.3.5 Create Jenkins job 17

2.4 Developer Environment 19

2.4.1 Clone Github Repository 20

2.4.2 Import kubectl config 20

2.4.3 Deploying spring-music application on K8s cluster 20

2.4.4 Triggering CI/CD Pipeline 23

3. Conclusion 26

# Overview

This document provides information about integrating KuBo with a CI/CD pipeline software (Jenkins here) in a lab environment.

Objective is to provide enough content so reader can set up quickly the whole environment for learning and testing purposes.

CI/CD pipeline allows to automate the process of building the code, testing it and deploying the artifact to production environment.

In this guide, we are going to see how this whole automation can be triggered from checking code to a source code management software (Github) to productizing the application on a Kubernetes cluster deployed by Kubo.

This guide assumes a Kubernetes cluster has already been deployed. Please refer to the guide “Kubo – Lab Install Guide” if needed.

## Acronyms

|  |  |
| --- | --- |
| Acronym | Definition |
| KuBo | Kubernetes on Bosh |
| K8s | Kubernetes |

## Useful Links

These links provide detailed information about KuBo:

* Kubo – Lab Install Guide:  
  <https://github.com/guillierf/KuBo-Docs/tree/master/Lab_Install_Guide>
* Kubo Deployment:

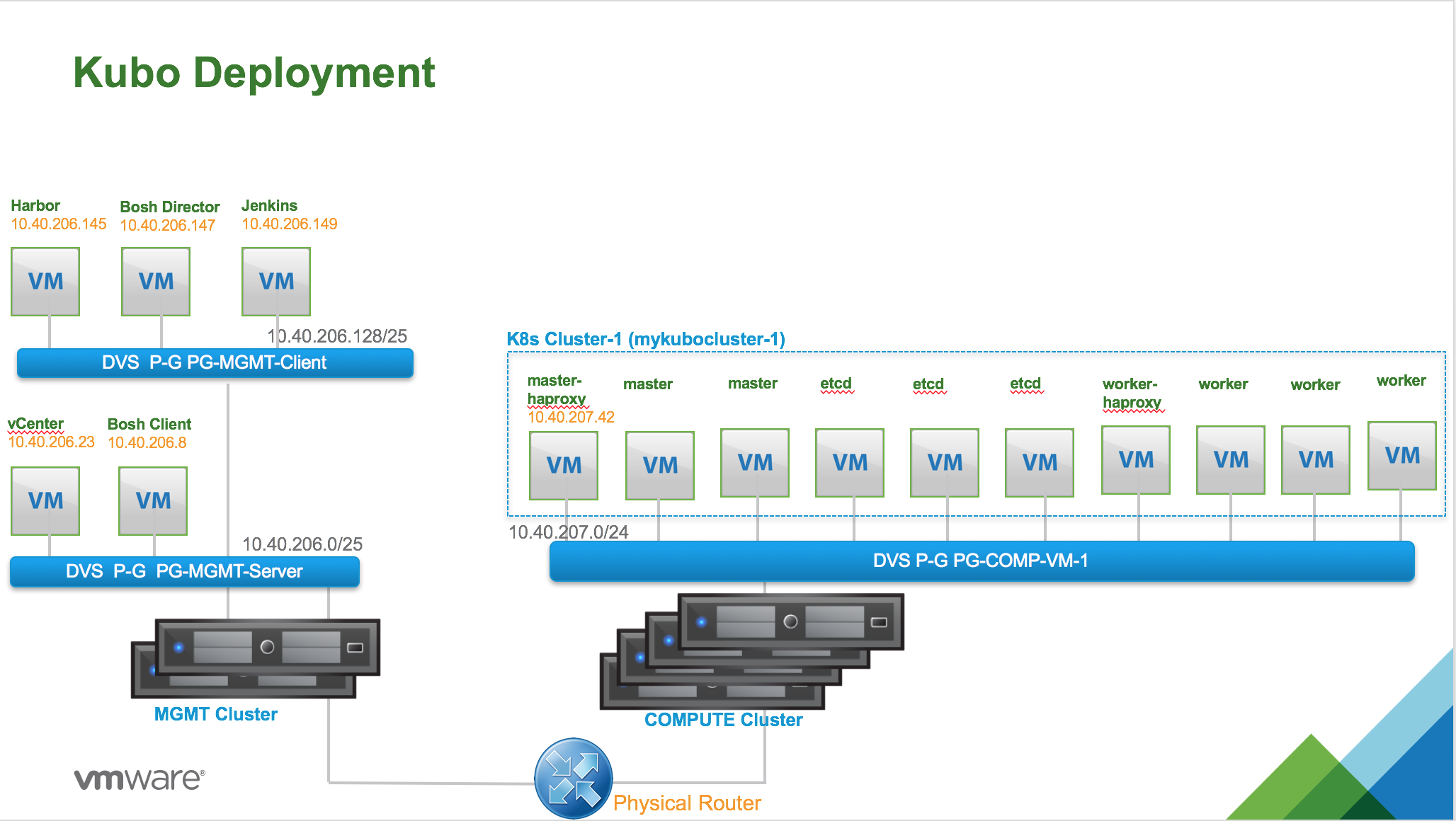
<https://github.com/cloudfoundry-incubator/kubo-deployment>

* Introduction to Bosh:

<https://github.com/virtmerlin/doc-bosh-intro/blob/master/Readme.md>

* Jenkins:  
  <https://jenkins.io/>
* Harbor:  
  <https://vmware.github.io/harbor/>

## KuBo Deployment



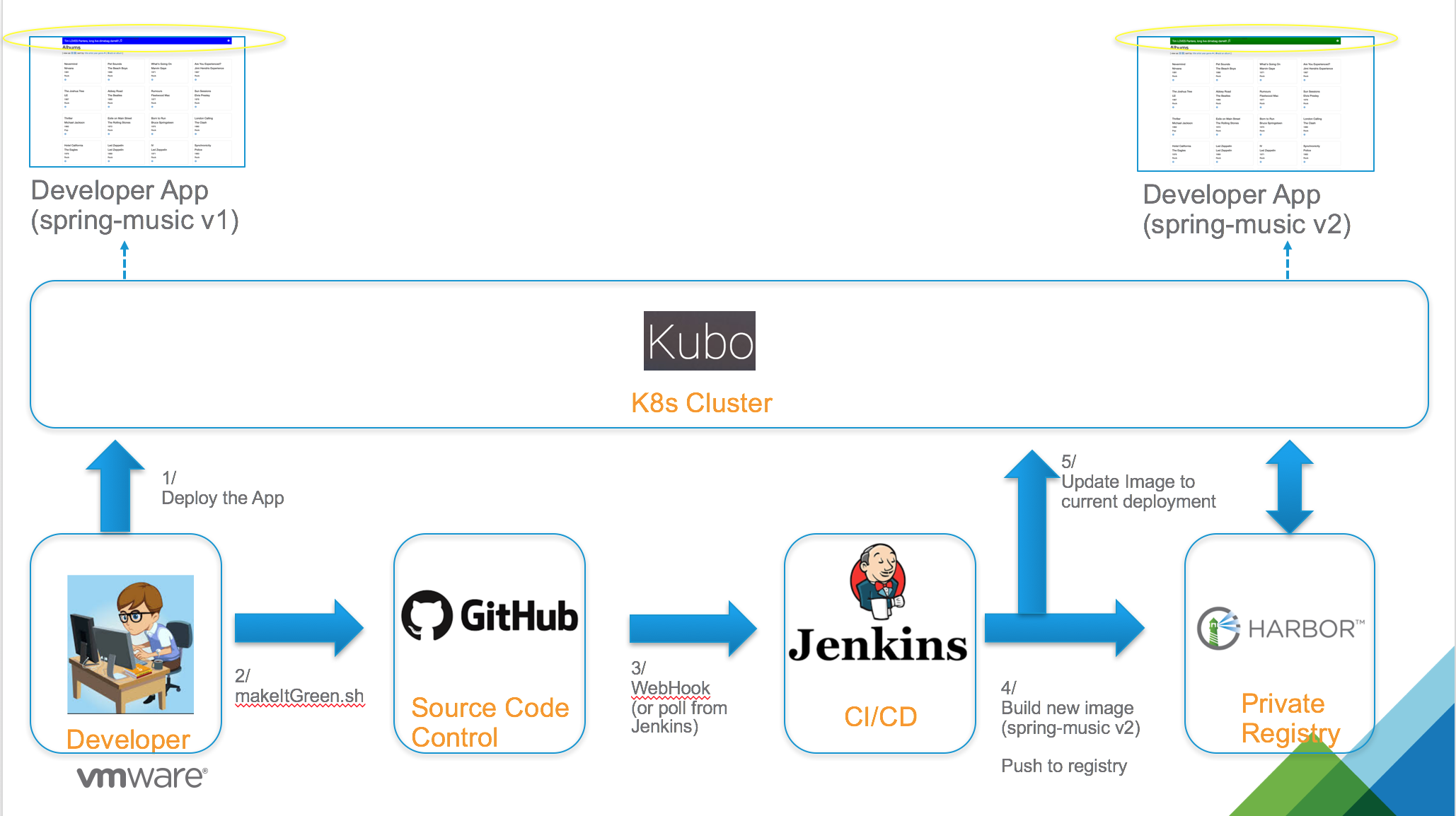
For the purpose of this document, we assume K8s cluster-1 have been deployed by KuBo.

Harbor (open source private registry from VMware) is configured with secure access mode and is fully operational.

Again, please refer to the guide “Kubo – Lab Intall Guide” to get all the necessary steps to instantiate K8s cluster using Kubo and Harbor.

## Workflow

The whole process involving KuBo and CI/CD Pipeline with Jenkins is depicted below:



Detailed sequencing steps are the following:

1. John, our developer, has written a containerized application spring-music v1 (based on Java). The docker image has been pushed to Harbor (not shown on the diagram above) and finally the app is deployed on K8s cluster (using the deployment object from K8s – with multiple replicas).  
     
   As you can see, the current app has a blue banner and John was asked by the marketing team to change the color from blue to green.
2. John has created a script named ‘makeItGreen.sh’ that perform the following operations:  
    a/ modifiy code so banner color is now green  
    b/ update the code to Github, the select Source Code Control from his company
3. Github is configured with a WebHook that instantly triggers a Jenkins job (if Jenkins is accessible from outside). Otherwise, Jenkins can be configured to periodically poll a Github repository and start a job when code has been checked in the repository.
4. Jenkins start a job that perform the following actions:  
    a/ build a new Docker image based on the new source code (spring-music v2 here)  
    b/ push the Docker image to Harbor
5. Jenkins then notifies K8s cluster to use the new image for the current deployment.  
   Kubernetes will perform a rolling upgrade of the app by initiating a blue/green deployment (will terminate half of the pod and recreate new pods with the new image. Once done, will repeat the operation for the other half).

At the end of the cycle, the app banner is now green!

# Lab Configuration

## Github

We assume reader has already an account on Github and is familiar with the product.

### Github Repository

Github repository used for this lab is:

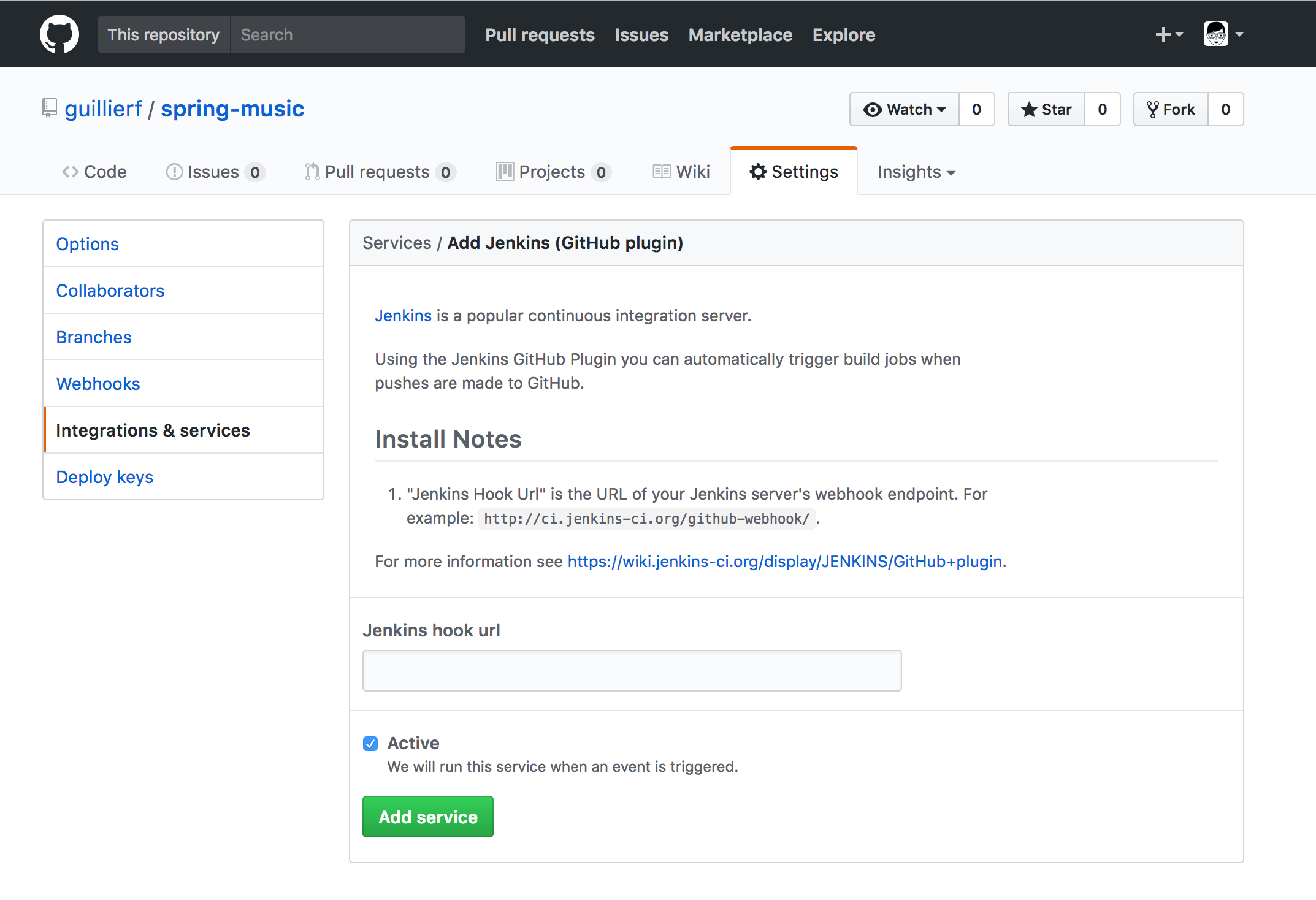
<https://github.com/guillierf/spring-music>

You will need to clone this repository in order to replay this lab.

### Webhooks – Jenkins (Github Plugin)

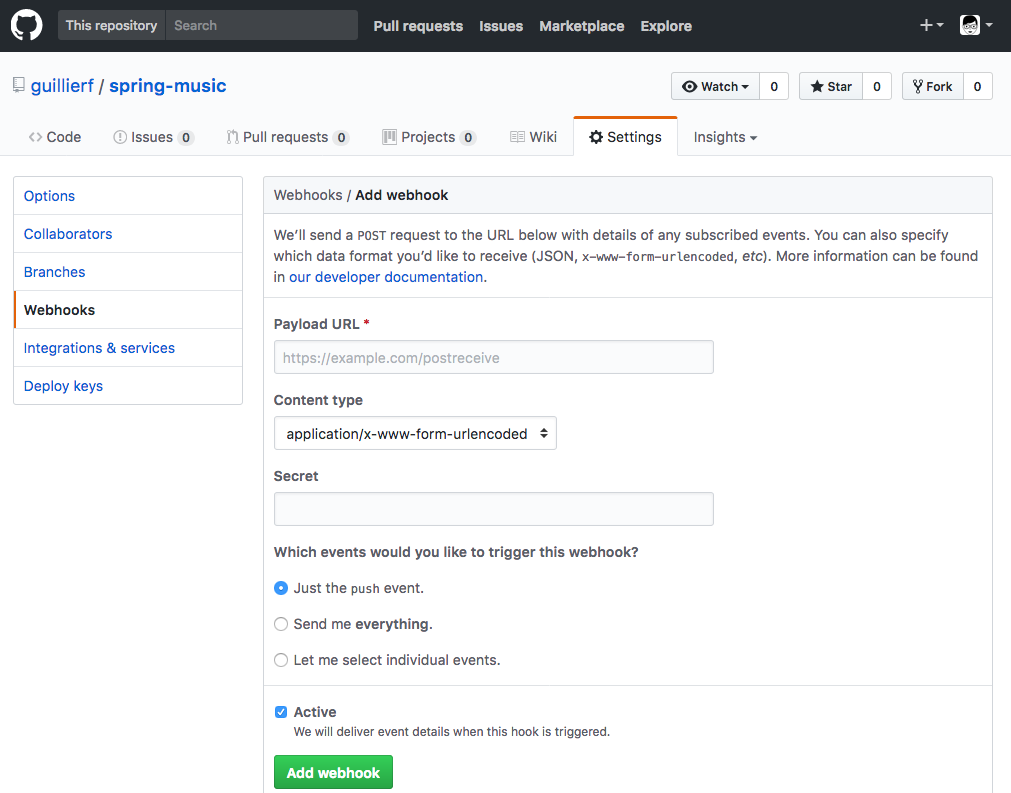
If your Jenkins is accessible from the outside, you can configure Webhooks.

Go on Settings and click on Integration & services:



Specify the Jenkins hook URL and click on Add service.

Click on Webhooks and click on Add webhooks:



Webhooks is a nice mechanism to inform Jenkins about real time code check-in. However, it is not a mandatory configuration.

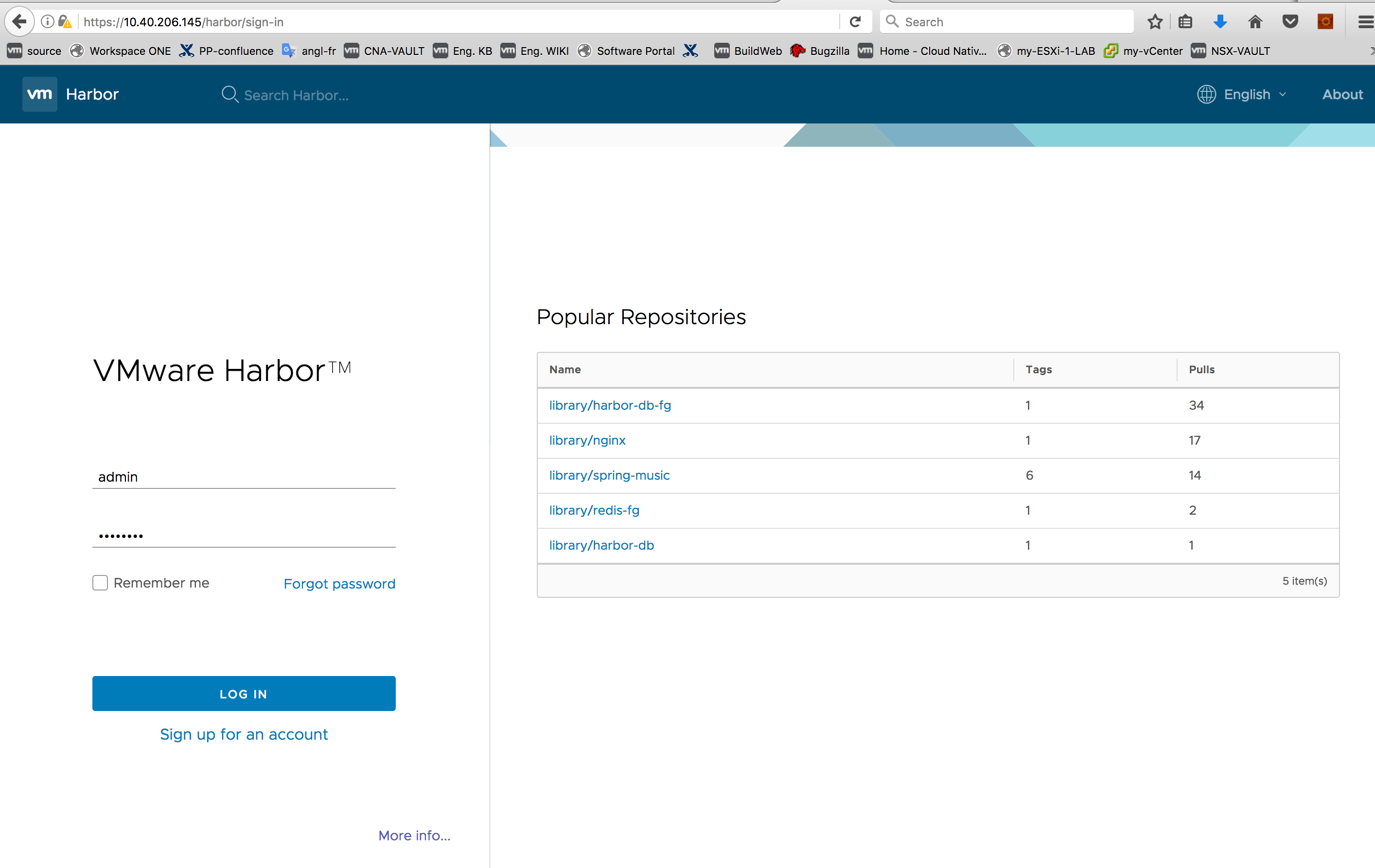
Jenkins provides periodic polling of Github repository to detect any code change. This configuration can be used instead or in complement of Github Webhooks.

## Harbor

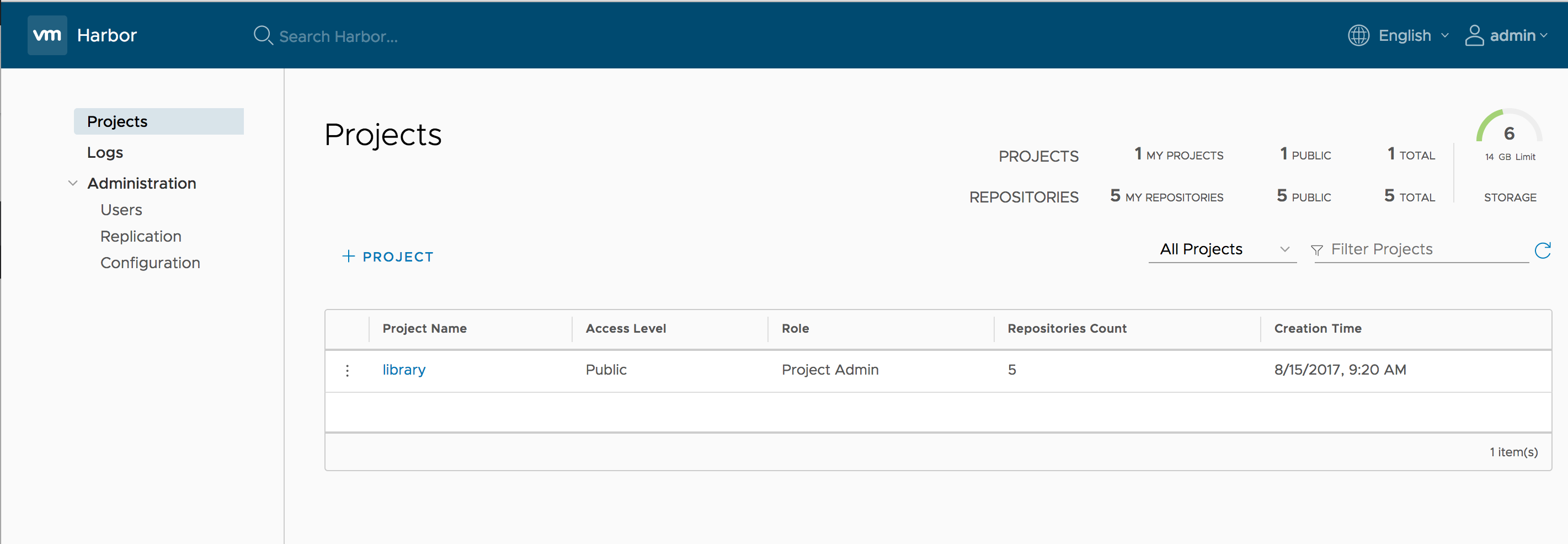
We assume here that Harbor is already installed and operational. Please refer to the guide “Kubo – Lab Install Guide” for information about installing and configuring Harbor.

To access Harbor web UI, use the following URL:

https://<Harbor IP>

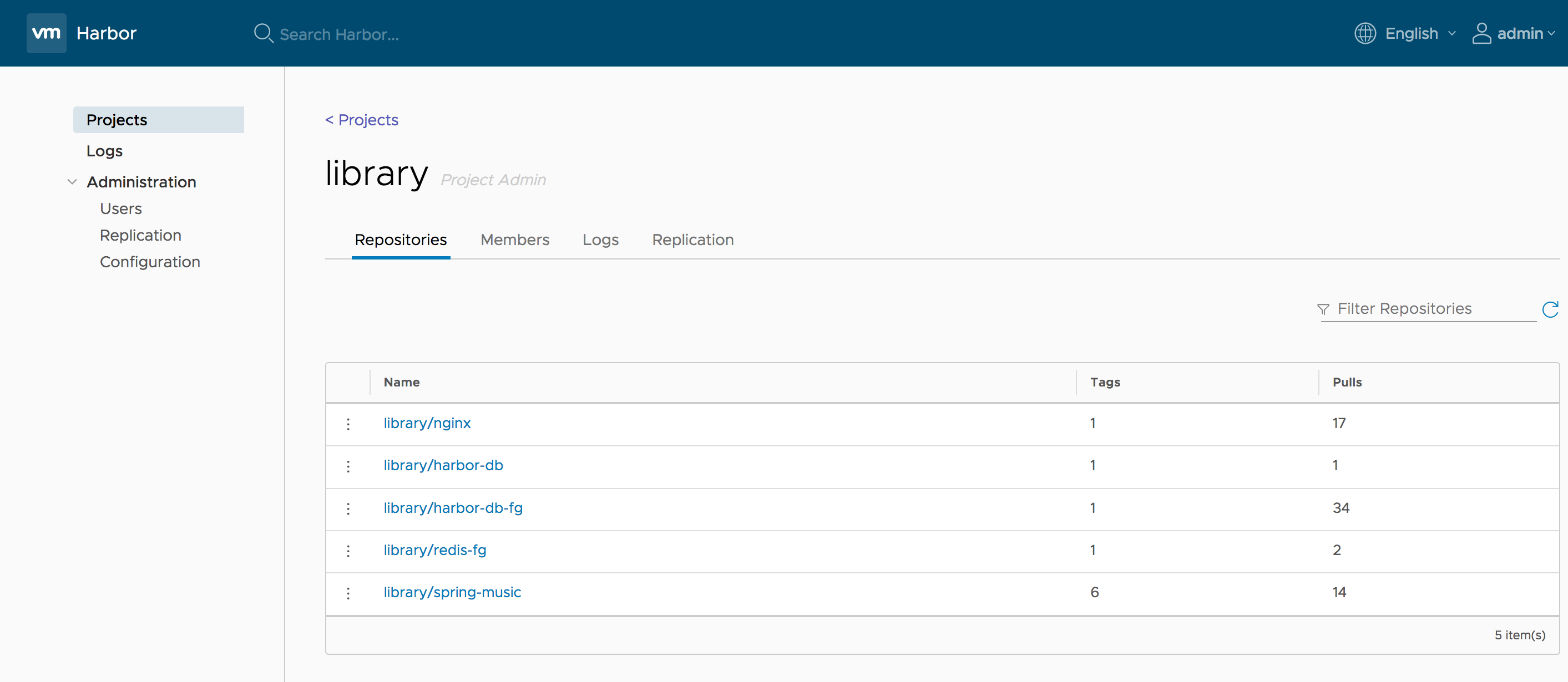


Log into Arbor using admin/<password>



You should be able to see a default project named ‘library’

Clicking on the ‘library’ link should display all images in the repository:



spring-music (v1) has already been uploaded to the registry.

## Jenkins

### Install Jenkins Server

Jenkins can be installed in ultiple ways. For this document, we have chosen to install Jenkins as a service on a Ubuntu (16.04) VM.

Follow this procedure to install Jenkins:

root@jenkins-server:~# **wget -q -O -** [**https://pkg.jenkins.io/debian/jenkins.io.key**](https://pkg.jenkins.io/debian/jenkins.io.key) **|  apt-key add -**  
OK

root@jenkins-server:~# **sh -c 'echo deb** [**http://pkg.jenkins.io/debian-stable**](http://pkg.jenkins.io/debian-stable) **binary/ > /etc/apt/sources.list.d/jenkins.list'**

root@jenkins-server:~# **apt-get update**  
Hit:1 <http://us.archive.ubuntu.com/ubuntu> xenial InRelease  
Hit:2 <http://us.archive.ubuntu.com/ubuntu> xenial-updates InRelease        
Hit:3 <http://us.archive.ubuntu.com/ubuntu> xenial-backports InRelease      
Hit:4 <http://security.ubuntu.com/ubuntu> xenial-security InRelease         
Ign:5 <http://pkg.jenkins.io/debian-stable> binary/ InRelease  
Get:6 <http://pkg.jenkins.io/debian-stable> binary/ Release [2,042 B]  
Get:7 <http://pkg.jenkins.io/debian-stable> binary/ Release.gpg [181 B]  
Get:8 <http://pkg.jenkins.io/debian-stable> binary/ Packages [12.4 kB]  
Fetched 14.6 kB in 1s (10.6 kB/s)  
Reading package lists... Done

root@jenkins-server:~# **apt-get install jenkins**  
Reading package lists... Done  
Building dependency tree         
Reading state information... Done  
The following additional packages will be installed:  
  ca-certificates-java daemon default-jre-headless fontconfig-config fonts-dejavu-core java-common libavahi-client3  
  libavahi-common-data libavahi-common3 libcups2 libfontconfig1 libjpeg-turbo8 libjpeg8 liblcms2-2 libnspr4 libnss3 libnss3-nssdb  
  libpcsclite1 libxi6 libxrender1 libxtst6 openjdk-8-jre-headless x11-common  
Suggested packages:  
  default-jre cups-common liblcms2-utils pcscd openjdk-8-jre-jamvm libnss-mdns fonts-dejavu-extra fonts-ipafont-gothic  
  fonts-ipafont-mincho fonts-wqy-microhei fonts-wqy-zenhei fonts-indic  
The following NEW packages will be installed:  
  ca-certificates-java daemon default-jre-headless fontconfig-config fonts-dejavu-core java-common jenkins libavahi-client3  
  libavahi-common-data libavahi-common3 libcups2 libfontconfig1 libjpeg-turbo8 libjpeg8 liblcms2-2 libnspr4 libnss3 libnss3-nssdb  
  libpcsclite1 libxi6 libxrender1 libxtst6 openjdk-8-jre-headless x11-common  
0 upgraded, 24 newly installed, 0 to remove and 0 not upgraded.  
Need to get 100 MB of archives.  
After this operation, 181 MB of additional disk space will be used.  
Do you want to continue? [Y/n] **Y**  
Get:2 <http://us.archive.ubuntu.com/ubuntu> xenial/main amd64 libjpeg-turbo8 amd64 1.4.2-0ubuntu3 [111 kB]  
Get:3 <http://us.archive.ubuntu.com/ubuntu> xenial/main amd64 liblcms2-2 amd64 2.6-3ubuntu2 [137 kB]  
Get:4 <http://us.archive.ubuntu.com/ubuntu> xenial/main amd64 x11-common all 1:7.7+13ubuntu3 [22.4 kB]  
<SNIP>  
Setting up default-jre-headless (2:1.8-56ubuntu2) ...  
Setting up jenkins (2.60.3) ...  
Processing triggers for libc-bin (2.23-0ubuntu9) ...  
Processing triggers for systemd (229-4ubuntu19) ...  
Processing triggers for ureadahead (0.100.0-19) ...  
Processing triggers for ca-certificates (20160104ubuntu1) ...  
Updating certificates in /etc/ssl/certs...  
0 added, 0 removed; done.  
Running hooks in /etc/ca-certificates/update.d...  
  
done.  
done.

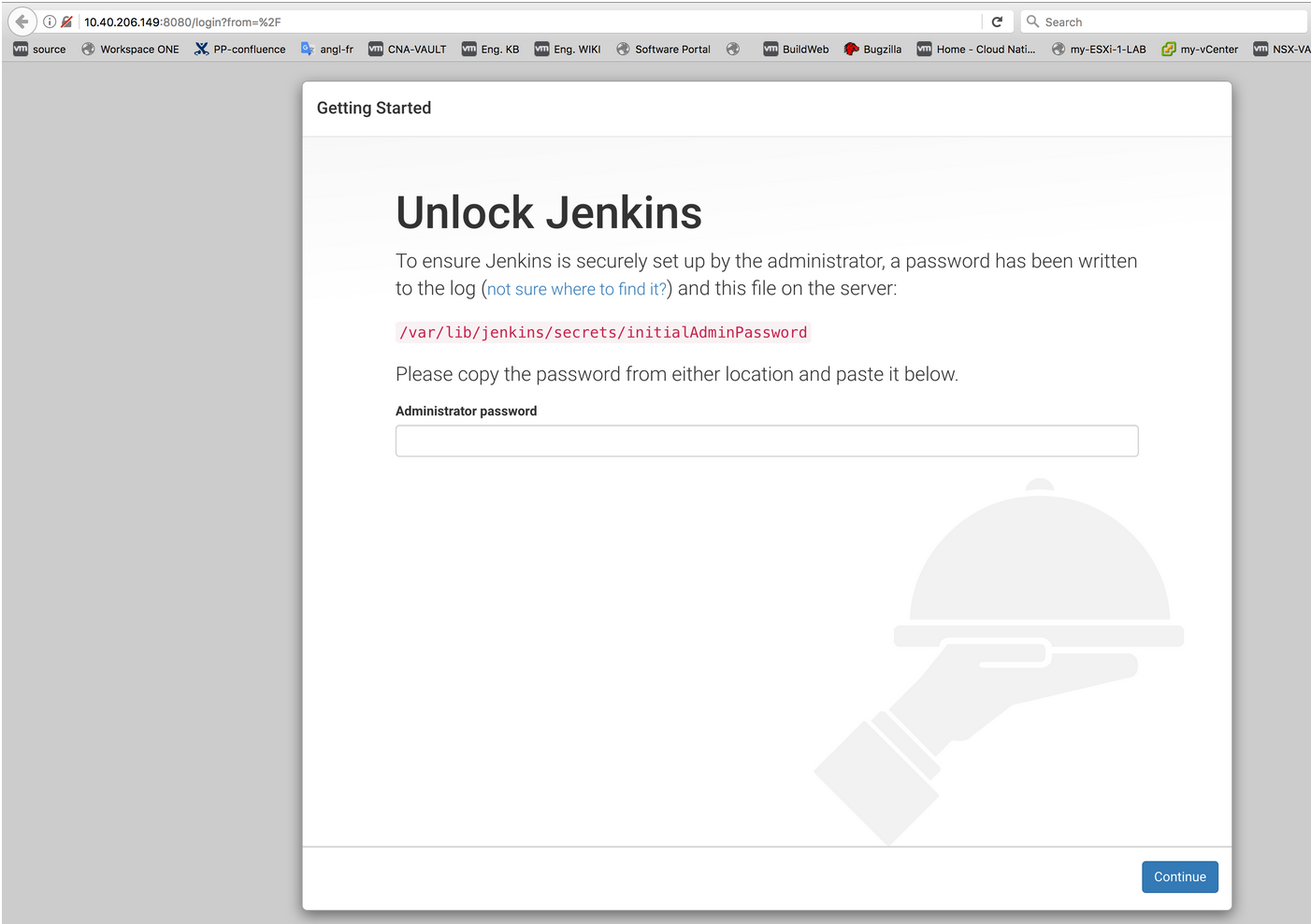
Install of Jenkins is completed!

### Configure Jenkins Server

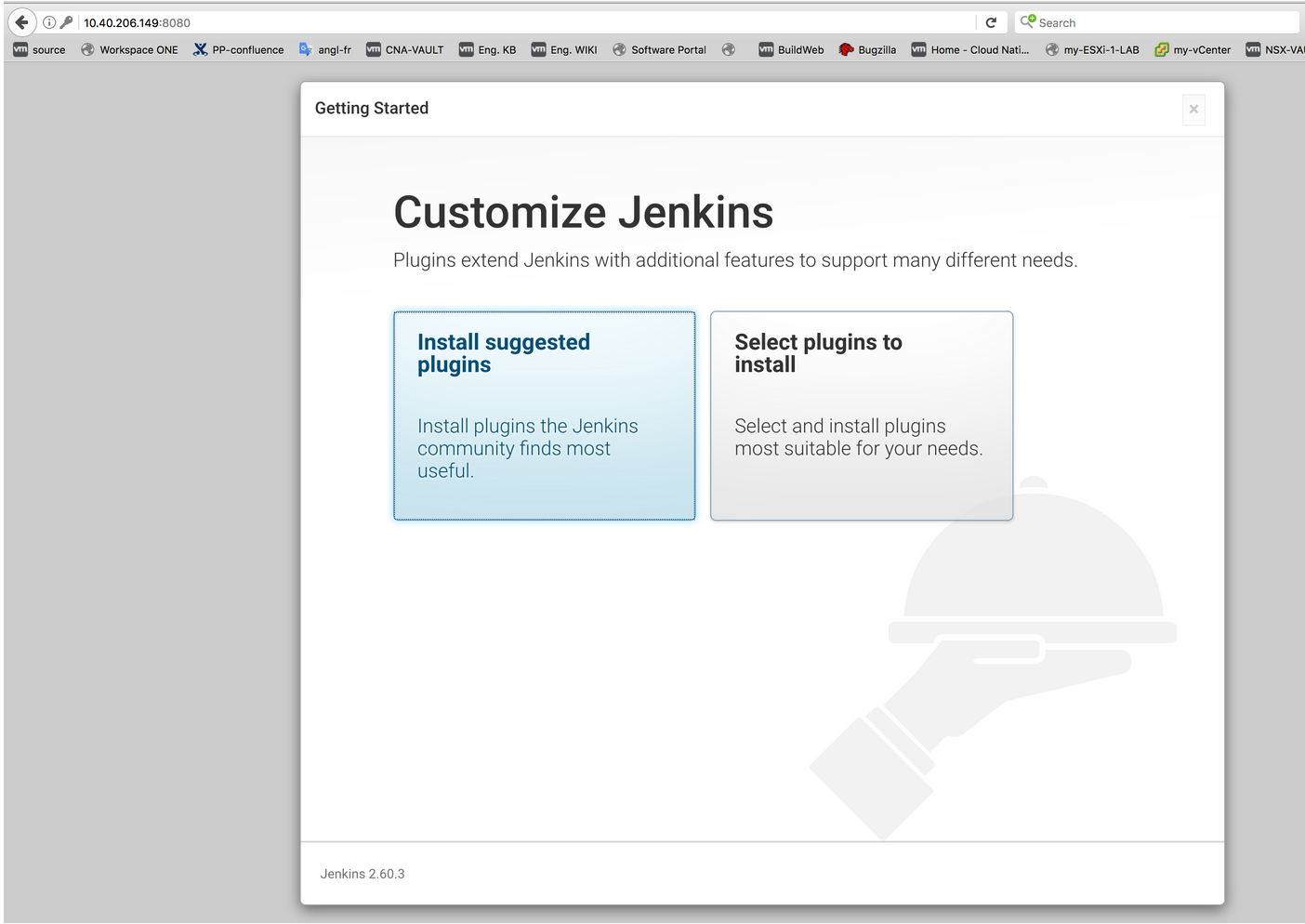
open a web browser and use this URL:

http://<Jenkins IP>:8080

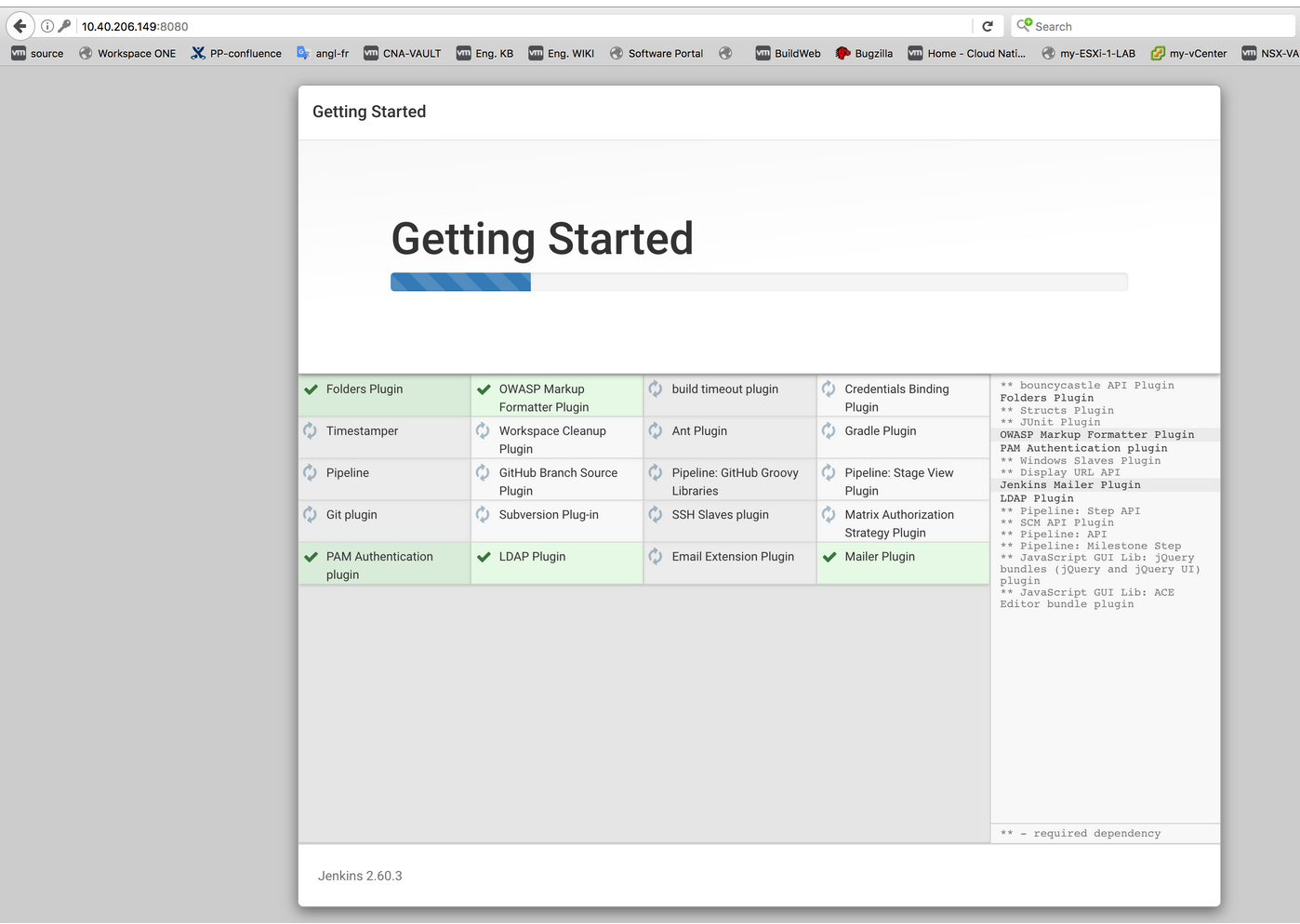
You will see:



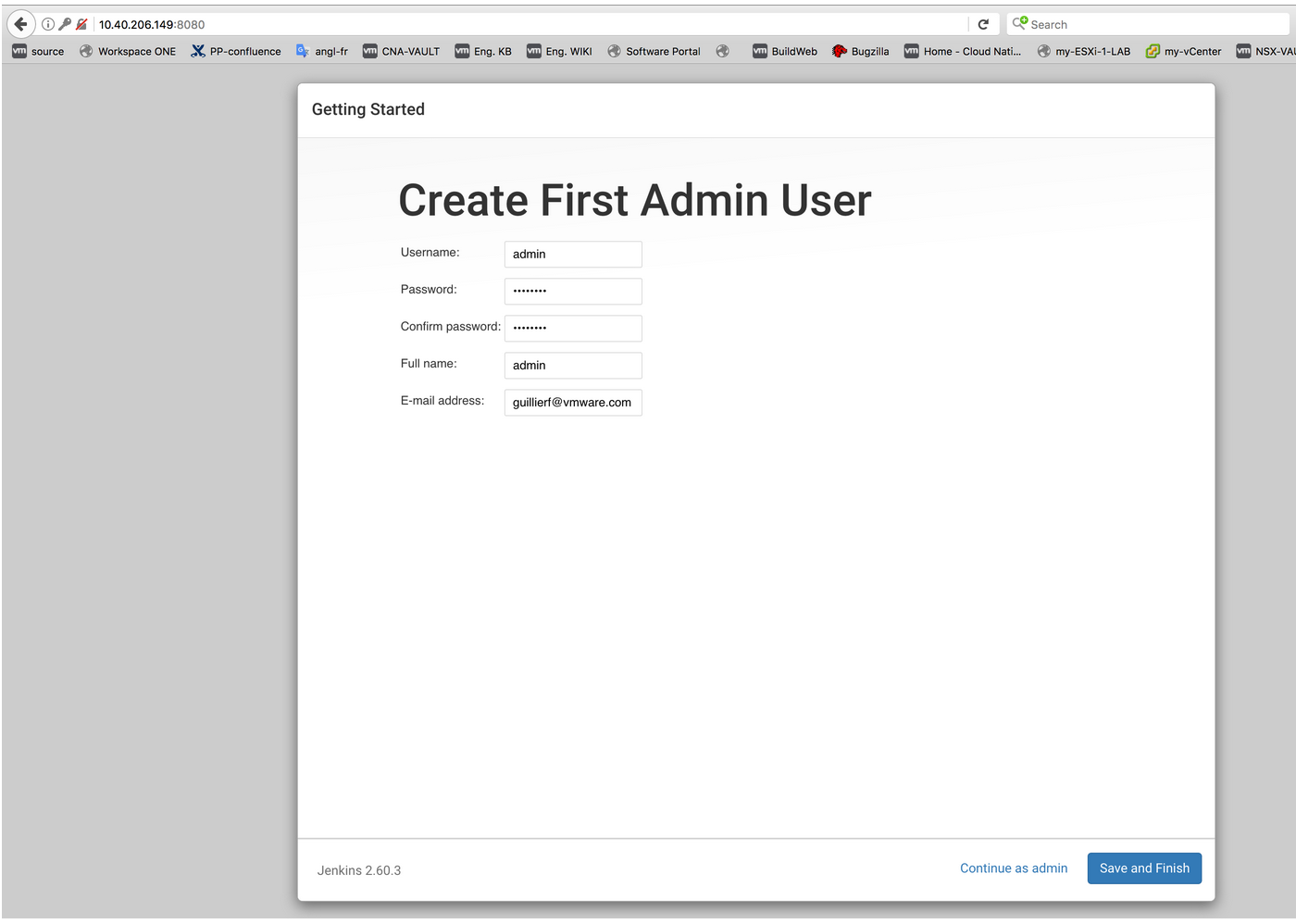
Copy/paste the requested password and click on continue:



Click on Install suggested plugins.

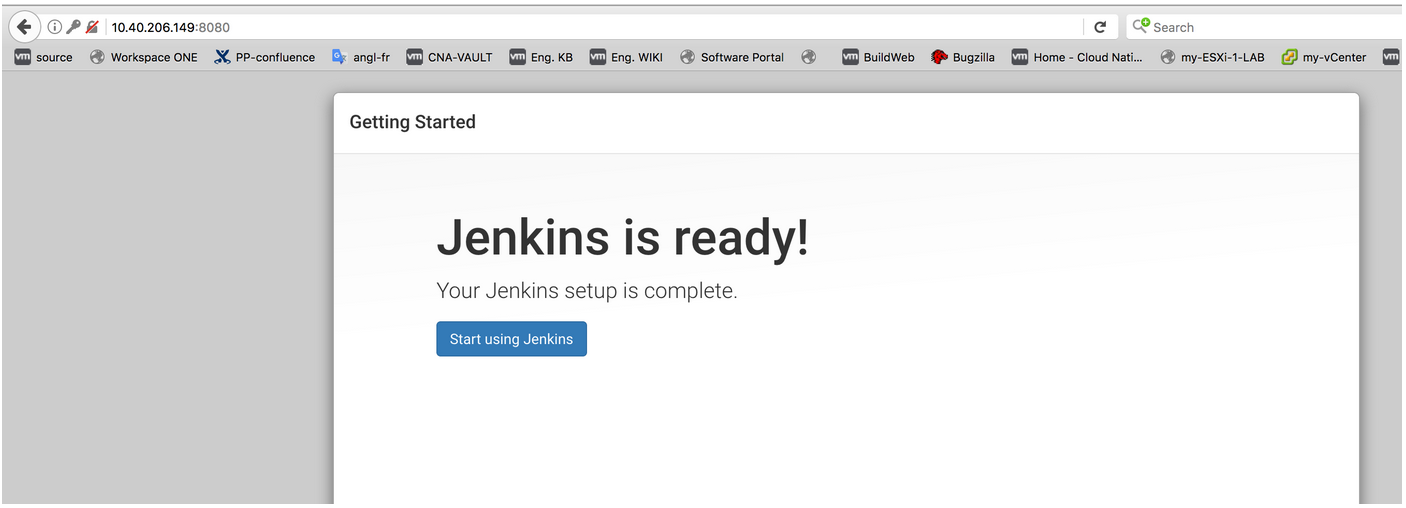


Once done, the following window will appear:



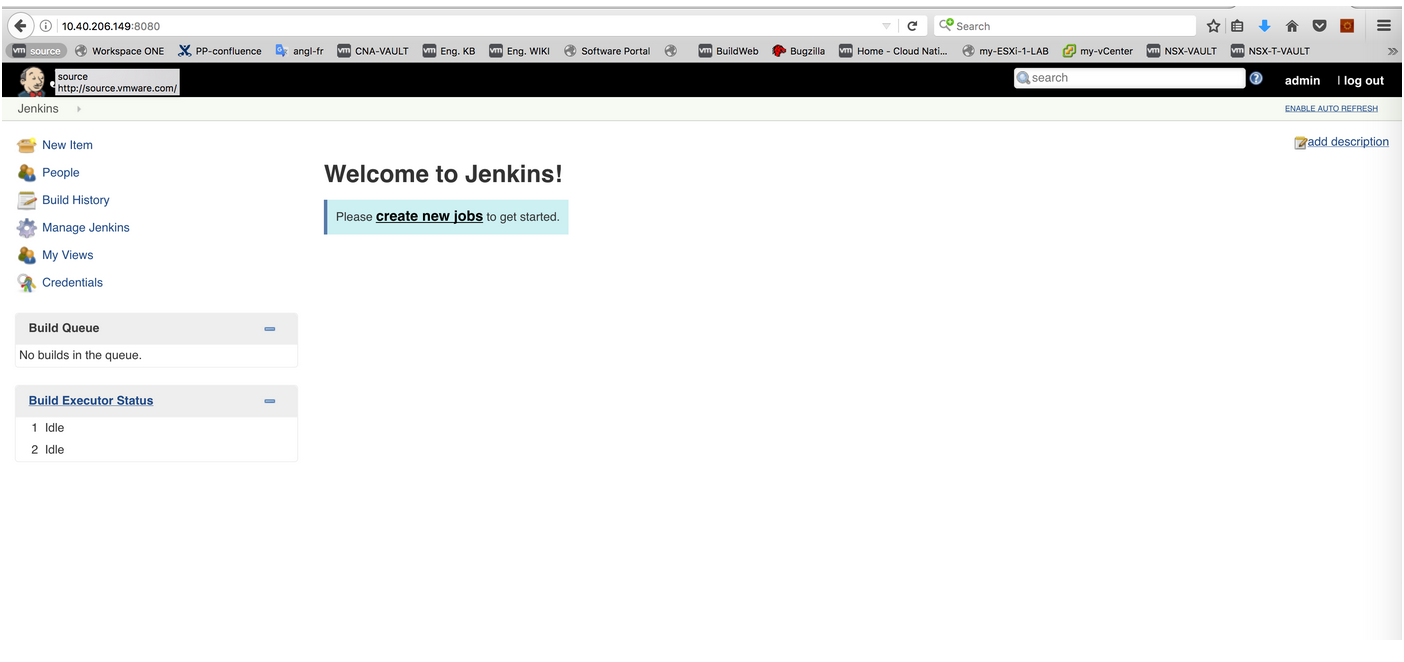
Fill the required fields and click on Save and Finish.

The following window will then appear:



Click on Start Using Jenkins.

The following window will then appear:



We will create a new job later.

To simplify lab configuration, the Jenkins server will run all the jobs (we won’t be adding Jenkins nodes for this purpose).

### Install Docker on Jenkins Server

root@jenkins-server:~# **curl -sSL https://get.docker.com/ | sh**

Executing docker install script, commit: c34e43c

+ sh -c apt-get update

<SNIP>

Processing triggers for ureadahead (0.100.0-19) ...

+ sh -c docker version

Client:

Version: 17.07.0-ce

API version: 1.31

Go version: go1.8.3

Git commit: 8784753

Built: Tue Aug 29 17:42:53 2017

OS/Arch: linux/amd64

Server:

Version: 17.07.0-ce

API version: 1.31 (minimum version 1.12)

Go version: go1.8.3

Git commit: 8784753

Built: Tue Aug 29 17:41:43 2017

OS/Arch: linux/amd64

Experimental: false

If you would like to use Docker as a non-root user, you should now consider

adding your user to the "docker" group with something like:

sudo usermod -aG docker your-user

Remember that you will have to log out and back in for this to take effect!

WARNING: Adding a user to the "docker" group will grant the ability to run

containers which can be used to obtain root privileges on the

docker host.

Refer to https://docs.docker.com/engine/security/security/#docker-daemon-attack-surface

for more information.

Check:

root@jenkins-server:~# docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

Importnant Node:

when jenkins launches a build job, it will run it using the unix user 'jenkins'.

So make sure to add user ‘jenkins’ to the ‘docker group’:

$ **sudo usermod -aG docker jenkins**

then reboot completely the Jenkins server (sudo usermod -aG docker jenkins - seems to require a full reboot of the VM - otherwise jenkins job will complain with message like:

docker: Got permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Post http://%2Fvar%2Frun%2Fdocker.sock/v1.26/containers/create: dial unix /var/run/docker.sock: connect: permission denied.

$ **init 6**

### Install Kubectl on Jenkins Server

root@jenkins-server:~# **curl -LO https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd64/kubectl**

% Total % Received % Xferd Average Speed Time Time Time Current

Dload Upload Total Spent Left Speed

100 68.9M 100 68.9M 0 0 23.2M 0 0:00:02 0:00:02 --:--:-- 23.2M

root@jenkins-server:~# **chmod +x ./kubectl**

root@jenkins-server:~# **mv ./kubectl /usr/local/bin/kubectl**

Check:

root@jenkins-server:~# **kubectl version**

Client Version: version.Info{Major:"1", Minor:"7", GitVersion:"v1.7.4", GitCommit:"793658f2d7ca7f064d2bdf606519f9fe1229c381", GitTreeState:"clean", BuildDate:"2017-08-17T08:48:23Z", GoVersion:"go1.8.3", Compiler:"gc", Platform:"linux/amd64"}

Retrieve kubectl config file from Bosh Client to Jenkins Server (this is the K8s config file that allows Bosh Client to initiate communications with K8s cluster using kubectl):

root@jenkins-server:~# **cd /**

root@jenkins-server:~# **mkdir .kube**

root@jenkins-server:~# **cd .kube**

root@jenkins-server:~/.kube# **scp root@10.40.207.38:~/.kube/config .**

The authenticity of host '10.40.207.38 (10.40.207.38)' can't be established.

ECDSA key fingerprint is SHA256:nvhvcXMuVsRDt0xDcN7o/htwLfnpm2vjJY9+sj5W5iY.

Are you sure you want to continue connecting (yes/no)? yes

Warning: Permanently added '10.40.207.38' (ECDSA) to the list of known hosts.

root@10.40.207.38's password:

config 100% 5359 5.2KB/s 00:00

Check:

root@jenkins-server:~/.kube# **kubectl get nodes**

NAME STATUS AGE VERSION

10.40.207.74 Ready 13d v1.6.6

10.40.207.75 Ready 13d v1.6.6

10.40.207.76 Ready 13d v1.6.6

10.40.207.78 Ready 7d v1.6.6

10.40.207.79 Ready 7d v1.6.6

Important Note:

When Jenkins starts a job, it will use the unix user 'jenkins'. So make sure the user ‘jenkins’ can launch successfully kubectl commands:

root@jenkins-server:~/.kube# **cp ~/.kube/config ~jenkins/.kube/config**

root@jenkins-server:~/.kube# **chmod 666 /var/lib/jenkins/.kube/config**

Check:

root@jenkins-server:~/.kube# **su - jenkins**

jenkins@jenkins-server:~$ **kubectl get nodes**

NAME STATUS AGE VERSION

10.40.207.74 Ready 13d v1.6.6

10.40.207.75 Ready 13d v1.6.6

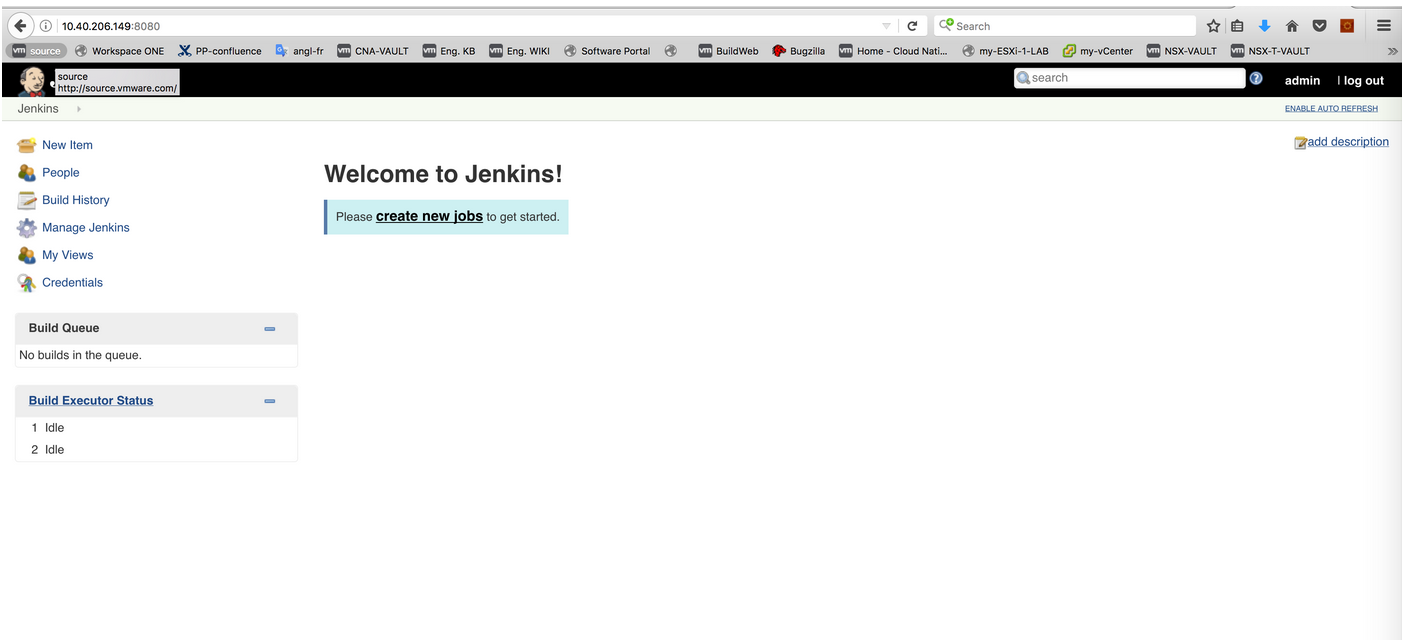
10.40.207.76 Ready 13d v1.6.6

10.40.207.78 NotReady 7d v1.6.6

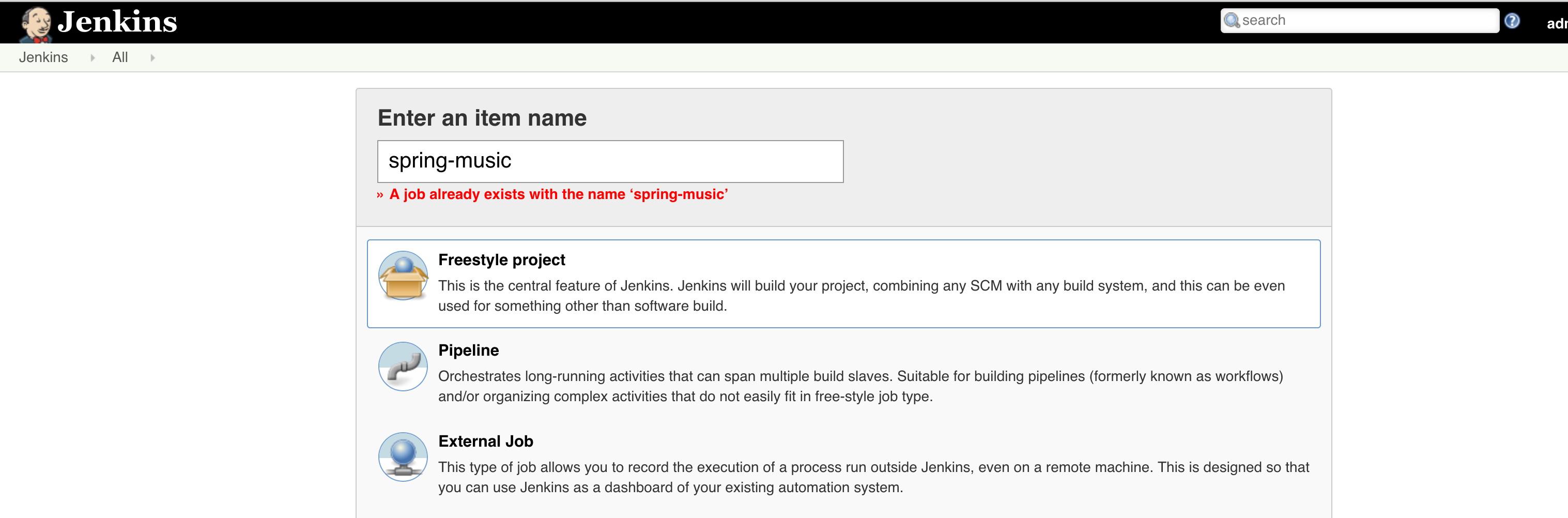
10.40.207.79 NotReady 7d v1.6.6

### Create Jenkins job

On Jenkins web UI, click on create new jobs:

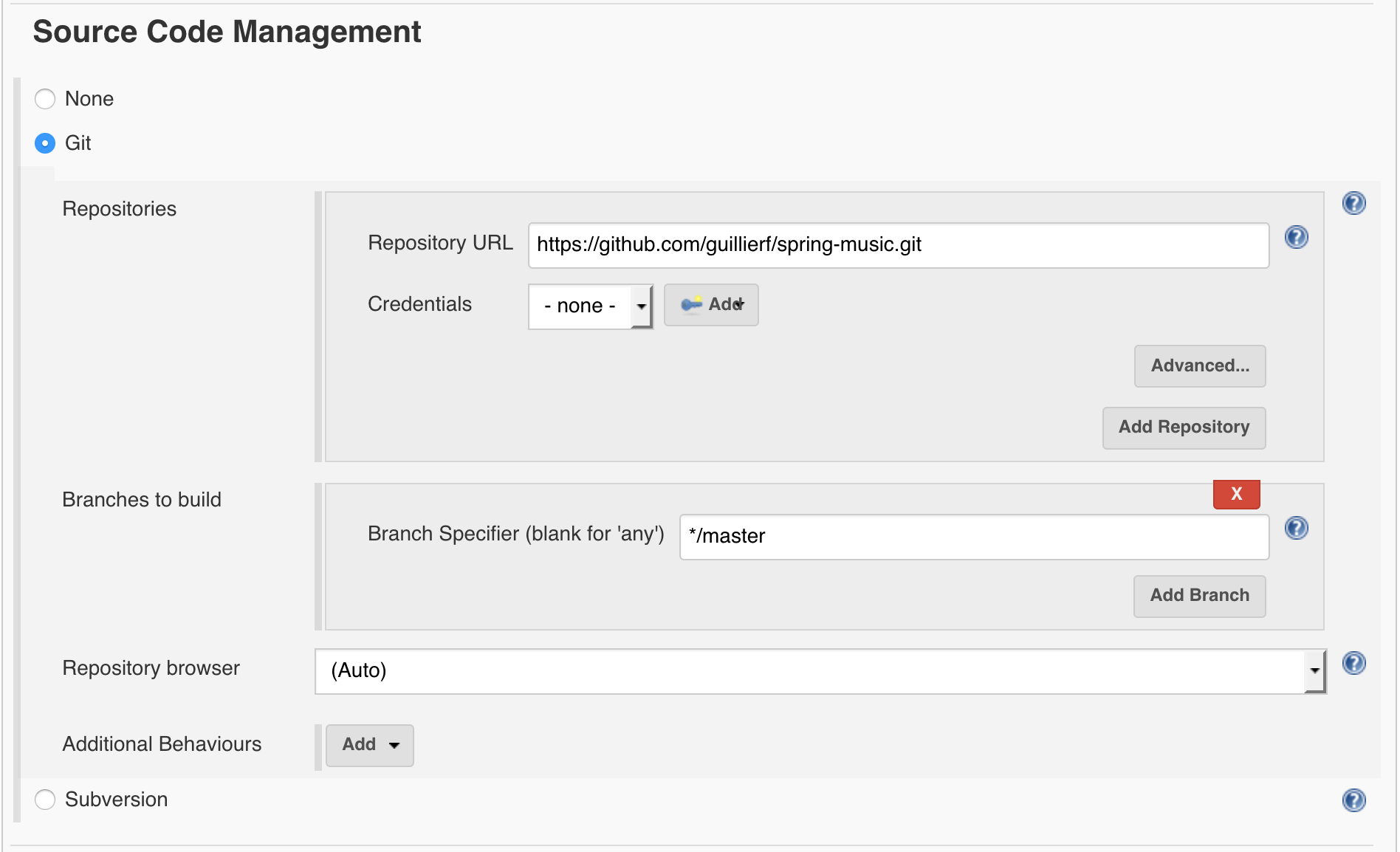


Enter a name and select Freestyle project:



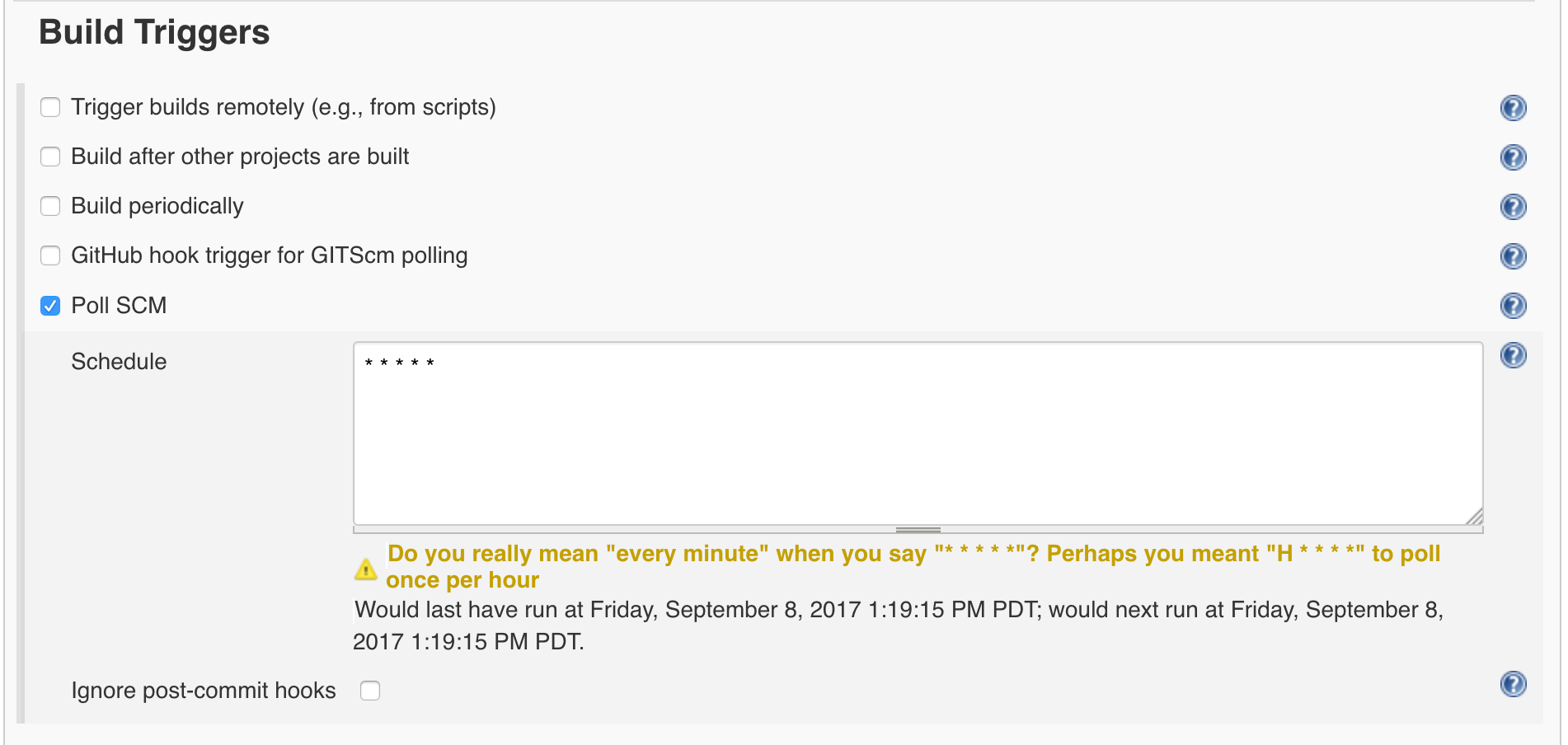
Click on OK

Configure Source Code Management section as following:



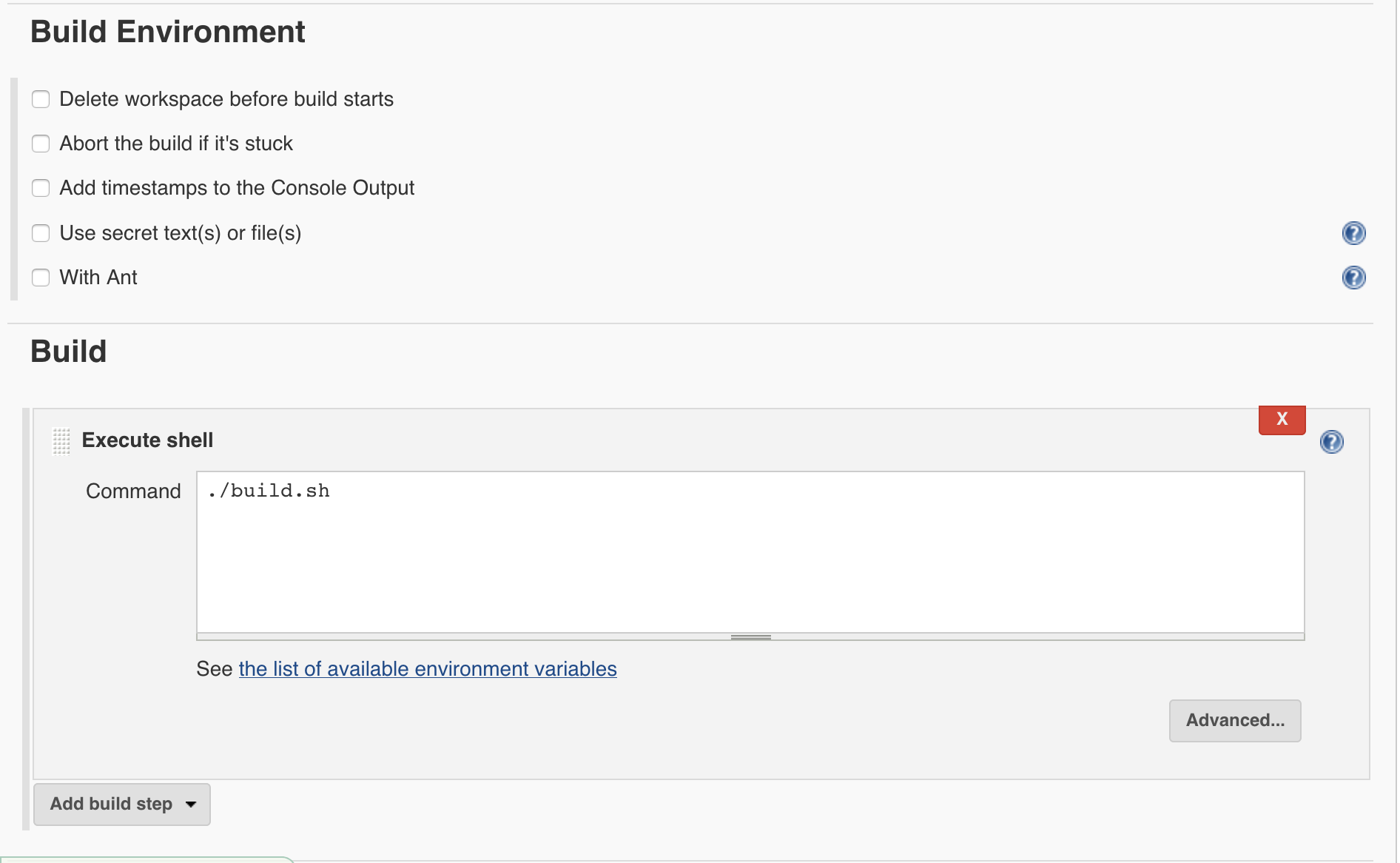
Remember to replace the Github repository URL with one that belongs to you (and that is the clone of the repository https://github.com/guillierf/spring-music.git).

Configure Build Triggers section a following:



Click on Poll SCM and enter ‘\* \* \* \* \*’ which means poll the Github repository every minute.

Configure Build Environment and Build section as following:



the shell script build.sh is located in /var/lib/jenkins/workspace/spring-music/:

build.sh

#!/bin/bash

now=$(date +"%s")

docker login 10.40.206.145 -u admin -p 'VMware1!'

docker build -t 10.40.206.145/library/spring-music:"$now" .

docker push 10.40.206.145/library/spring-music:"$now"

kubectl set image deployments/spring-music spring-music="10.40.206.145/library/spring-music:$now" --record

Actions performed by the script:

* Log into Harbor private registry (docker login)
* Build a new Docker image based on the updated content of the Github repository (docker build)
* Push the new Docker image to Harbor private registry
* Invoke K8s cluster to use the new image for the deployment named ‘spring-music’

## Developer Environment

Let’s pretend now we are John, the developer.

The first action we need to perform is to deploy our app onto K8s cluster. Then based on the marketing request, we will change the code to modify the banner color from blue to green.

From a developer environment standpoint, we need to clone the repository <https://github.com/guillierf/spring-music> and import kubectl config in order to be able to connect to K8s cluster.

All commands below are initiated from John laptop:

### Clone Github Repository

Use the following commands to clone the Github repository:

$ **git clone https://github.com/guillierf/spring-music.git**

$ **cd spring-music**

$ **rm –rf .git**

$ **git init**

$ **git add .**

$ **git commit –m “first commit”**

$ **git remote add origin https://github.com/<your username><your repository>.git**

$ **git push -u origin master**

### Import kubectl config

Import kubectl config from Bosh client:

$ **scp root@10.40.207.38:~/.kube/config .**

root@10.40.207.38's password:

config 100% 5359 5.2KB/s 00:00

Check:

# **kubectl get nodes**

NAME STATUS AGE VERSION

10.40.207.74 Ready 1m v1.6.6

10.40.207.78 Ready 1m v1.6.6

10.40.207.79 Ready 1m v1.6.6

### Deploying spring-music application on K8s cluster

Following files should be available from the Github repository cloning:

spring-music.yml

spring-music.yml

apiVersion: apps/v1beta1

kind: Deployment

metadata:

name: "spring-music"

spec:

replicas: 6

minReadySeconds: 30

revisionHistoryLimit: 2

strategy:

rollingUpdate:

maxSurge: "25%"

template:

metadata:

labels:

app: "spring-music"

spec:

containers:

- name: "spring-music"

image: "10.40.206.145/library/spring-music"

imagePullPolicy: "Always"

ports:

- containerPort: 8080

Note: make sure to change the image field with the appropriate parameter from your lab.

spring-music-service.yml

spring-music-service.yml

apiVersion: "v1"

kind: "Service"

metadata:

name: "spring-music"

namespace: "default"

spec:

type: "NodePort"

selector:

app: "spring-music"

ports:

- name: "http"

port: 8080

nodePort: 32000

protocol: "TCP"

Use the following commands to deploy spring-music app on K8s cluster:

**kubectl create –f spring-music.yml**

**kubectl create –f spring-music-service.yml**

Check:

# **kubectl get pod**

NAME READY STATUS RESTARTS AGE

spring-music-2358449792-31bbs 1/1 Running 0 1d

spring-music-2358449792-6cbw4 1/1 Running 0 1d

spring-music-2358449792-733wq 1/1 Running 0 1d

spring-music-2358449792-7kz76 1/1 Running 0 1d

spring-music-2358449792-9qkt1 1/1 Running 0 1d

spring-music-2358449792-zb4lz 1/1 Running 0 1d

# **kubectl get svc**

NAME CLUSTER-IP EXTERNAL-IP PORT(S) AGE

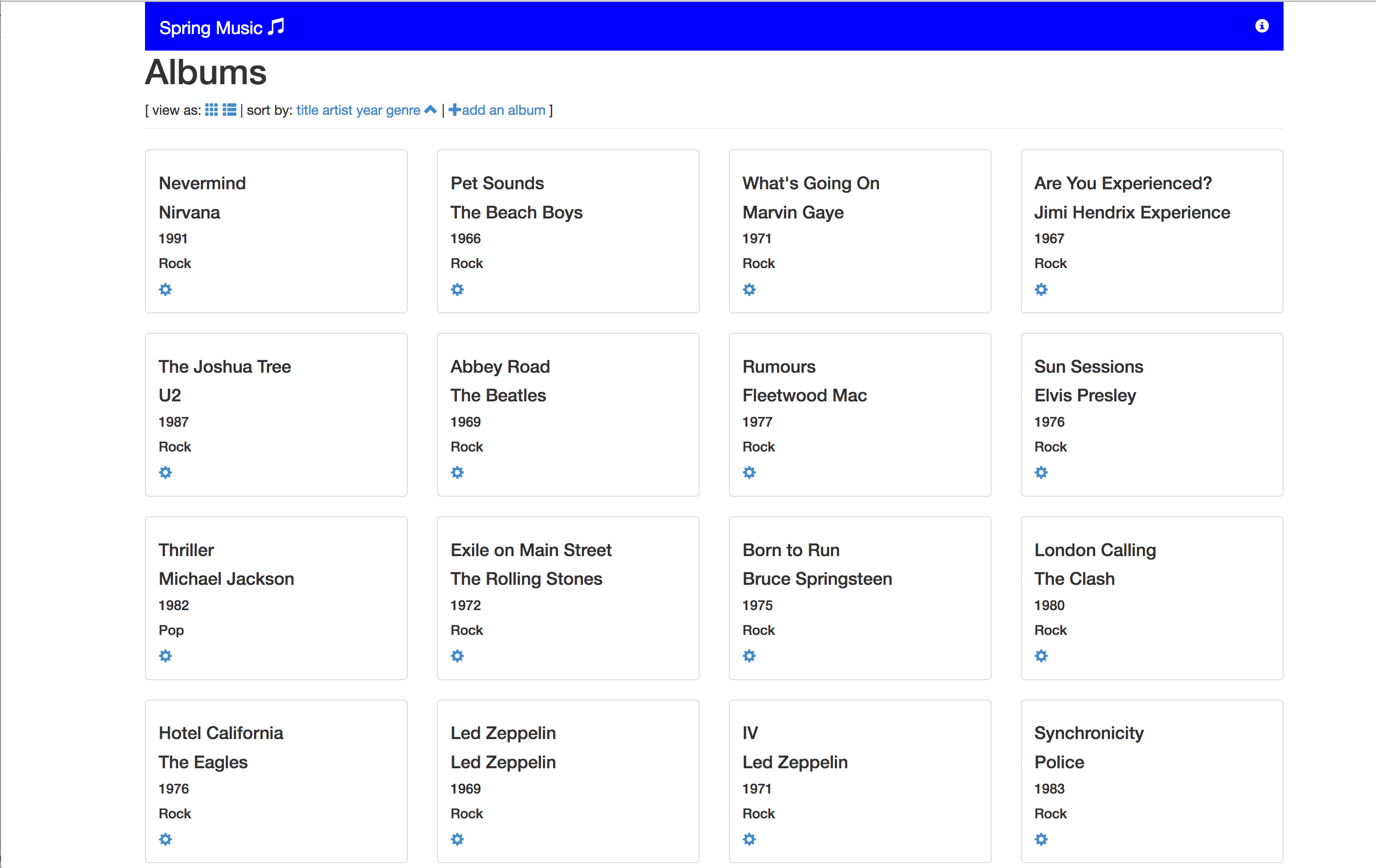
kubernetes 10.100.200.1 <none> 443/TCP 24d

spring-music 10.100.200.57 <nodes> 8080:32000/TCP 22d

To access the spring-music app, use the following URL:

http://<Any Worker Node IP>:32000

You should be able to see:



### Triggering CI/CD Pipeline

To trigger the job on Jenkins, we need to change the source code of the spring-music app (banner color from blue to green) and push the changes to Github.

The following script will perform these actions (it should be part of the Github repository clone):

makeItGreen.sh

#!/usr/bin/env bash

git pull

cp templates/green-app.css src/main/resources/static/css/app.css

git add src/main/resources/static/css/app.css

git commit -m 'making banner green'

git push -u origin master

sleep 3

echo " \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

echo " press any button to continue"

echo " \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

read

watch -n1 kubectl get all -o wide

Launch the script:

$ **makeItGreen.sh**

Already up-to-date.

[master b6d6093] making banner green

1 file changed, 7 insertions(+), 7 deletions(-)

Counting objects: 7, done.

Delta compression using up to 2 threads.

Compressing objects: 100% (6/6), done.

Writing objects: 100% (7/7), 578 bytes | 0 bytes/s, done.

Total 7 (delta 3), reused 1 (delta 0)

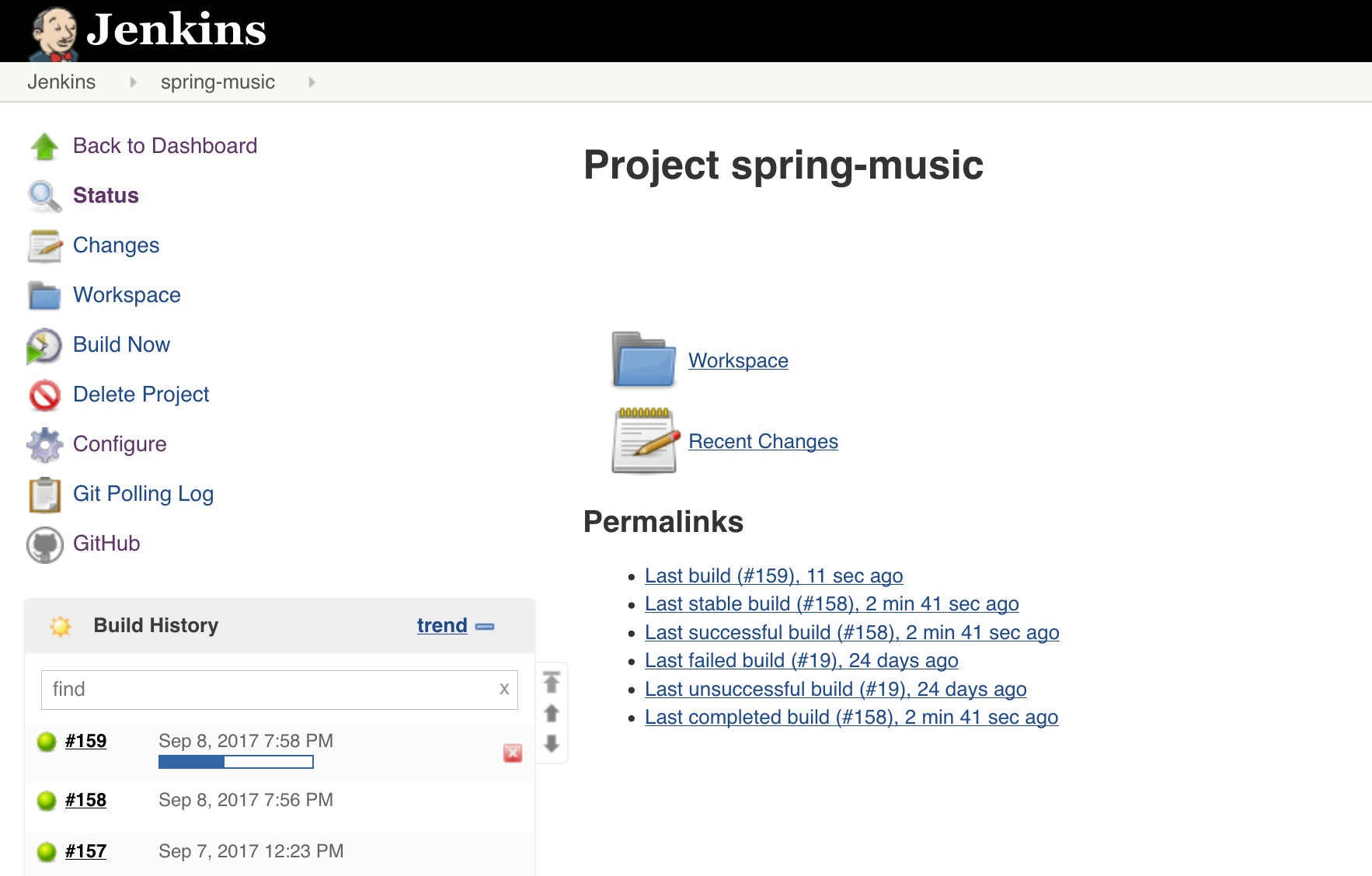
remote: Resolving deltas: 100% (3/3), completed with 3 local objects.

To https://github.com/guillierf/spring-music.git

e2bb40c..b6d6093 master -> master

Branch master set up to track remote branch master from origin.

On Jenkins, you should be able to see a job triggering:



Once the Jenkins job is completed, an upgrade of the Docker image will be initiated for the current Kubernetes deployment. Kubernetes will start a blue/green type of upgrade by terminating half of the POD and creating new PODs with the new image. Then it will complete the process with the other half PODs.

During the upgrade process:

# **kubectl get pod**

NAME READY STATUS RESTARTS AGE

spring-music-217978518-3xpxt 1/1 Running 0 3m

spring-music-217978518-fghhn 1/1 Terminating 0 3m

spring-music-217978518-gqvbv 1/1 Terminating 0 1m

spring-music-217978518-mz3b1 1/1 Terminating 0 1m

spring-music-217978518-rr7lx 1/1 Running 0 3m

spring-music-3337426570-41nd0 0/1 ContainerCreating 0 5s

spring-music-3337426570-c5tkd 0/1 ContainerCreating 0 5s

spring-music-3337426570-fdl95 1/1 Running 0 48s

spring-music-3337426570-l804q 1/1 Running 0 48s

spring-music-3337426570-vf8hr 1/1 Running 0 48s

spring-music-3337426570-x275q 1/1 Running 0 5s

# **kubectl get deployment**

NAME DESIRED CURRENT UP-TO-DATE AVAILABLE AGE

spring-music 6 8 6 5 22d

A new Replica Set is created. It contains the new Docker image for the spring-music deployment:

# **kubectl get rs**

NAME DESIRED CURRENT READY AGE

spring-music-2358449792 0 0 0 1d

spring-music-3337426570 6 6 6 58s

At the end of the upgrade process, you should be able to see all the new PODs in running state:

# **kubectl get pod**

NAME READY STATUS RESTARTS AGE

spring-music-3337426570-41nd0 1/1 Running 0 2m

spring-music-3337426570-c5tkd 1/1 Running 0 2m

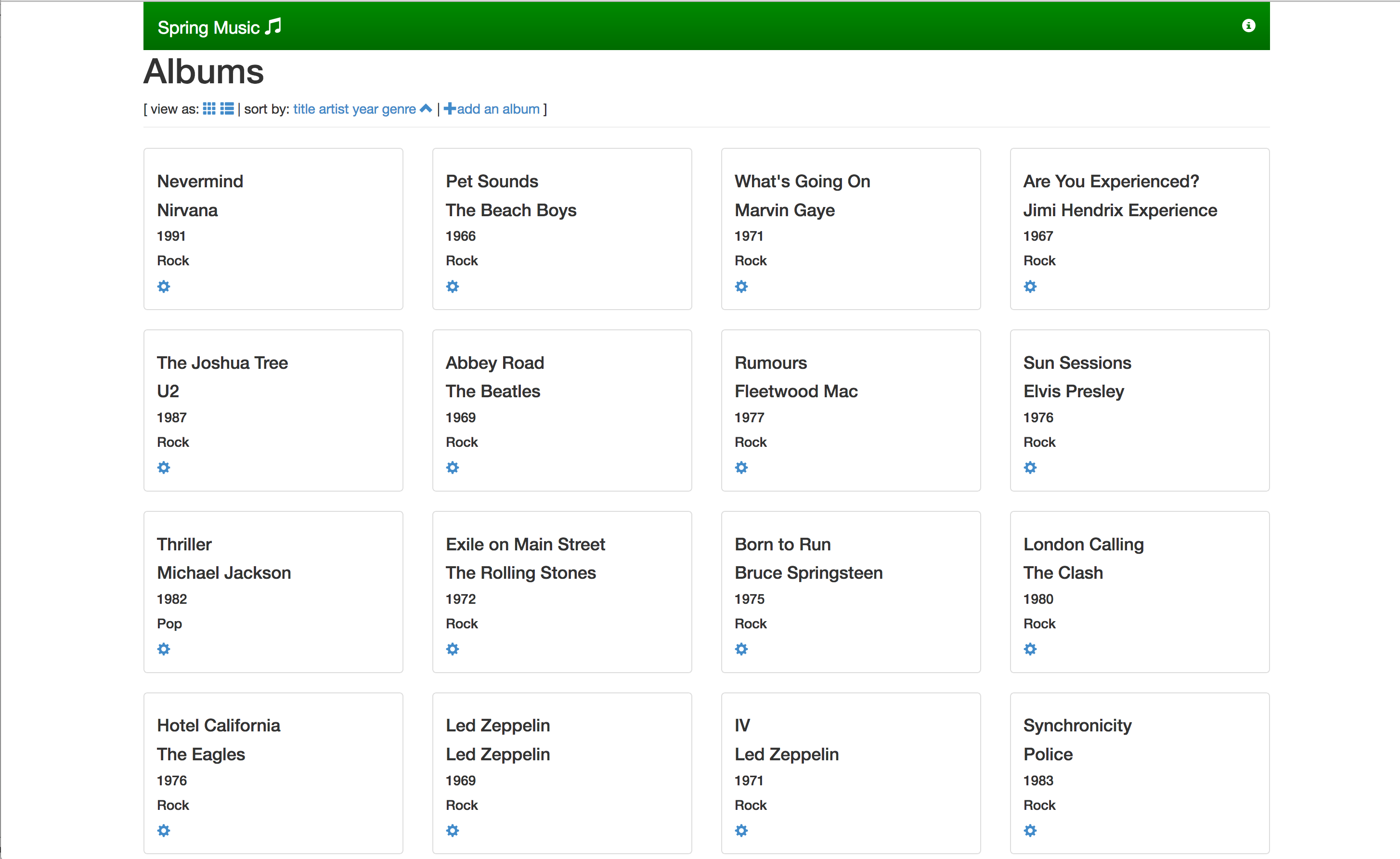
spring-music-3337426570-fdl95 1/1 Running 0 2m

spring-music-3337426570-l804q 1/1 Running 0 2m

spring-music-3337426570-vf8hr 1/1 Running 0 2m

spring-music-3337426570-x275q 1/1 Running 0 2m

The app should appear with a green banner now:



# Conclusion

This guide has shown the integration between Kubo and a CI/CD pipeline software like Jenkins.

By checking the latest source code on Github, Jenkins automatically launches a job that build the new Docker image, pushes it to private registry Harbor and then invokes Kubernetes cluster to use it for an existing deployment.

All the build and deployment process have been automated, meaning developers have now this agile and rapid mechanism to see their code running quickly on a production environment.