

Deploy Spring Boot microservices on kubernetes?

📅 Jun 10, 2020 · 16 min read · [SPRING BOOT](#)

· Last Modified : Aug 26, 2021 · Share on: [Twitter](#) [Facebook](#) [LinkedIn](#) [Email](#) · Author : [Rahul Wagh](#)

TL;DR: This is going to be long post and the objective of this blog post is to deploy [Spring Boot Microservice](#) into the [kubernetes cluster](#).

The main objective here is to build everything from scratch and what I mean by that - creating a Spring Boot Application RestMicroservice, Building a Docker image and Finally deploying docker image into the kubernetes cluster. Here are the steps which we will go through -

Kubernetes Part -1 : Steps to Deploy Spring Boot application in Kubernetes running on Virtual Machine

1. [Create a Spring Boot Application using the Spring Boot Initializr](#)
2. [Import the spring boot project into IDE](#)
3. [Implement your microservice](#)
4. [Test the rest webservice](#)
5. [Create Dockerfile for spring boot application](#)
6. [Create Docker registry at "https://hub.docker.com/"](#)
7. ["docker build" && "docker push image"](#)
8. [Start kubernetes cluster](#)
9. [Deploy Spring Boot microservices on kubernetes](#)
10. [Clean up the Deployment and Service](#)

Categories

[TERRAFORM](#) 63[DOCKER](#) 28[KUBERNETES](#) 26[AWS](#) 13[ANSIBLE](#) 11[HELM-CHART](#) 11[BLOGGING](#) 6[SSL](#) 6[SPRING-BOOT](#) 5[QUARKUS](#) 4[GITHUB](#) 3[KUBESPRAY](#) 3[PROMETHEUS-GRAFANA](#) 3[VAGRANT](#) 3[ALL CATEGORIES](#)

Series

Kubernetes Part -2 : Steps to Deploy Spring Boot application in Kubernetes running on Google Cloud

1. [Create a project on Google Cloud](#)
2. [Create Kubernetes Cluster on Google Cloud](#)
3. [Push spring boot docker image to google container registry\(gcr.io\)](#)
4. [Deploy the spring boot microservice inside kubernetes cluster running on Google Cloud](#)

In the [Part-1](#) of this blog we will first build and deploy Spring Boot microservice application into the Kubernetes cluster running into [virtual machine](#) using [Vagrant](#) and in the [Part-2](#) of this blog we will use [Google Cloud](#) to deploy the same Spring Boot microservice application.

Here are some more advance topics-

If you are interested in taking the same application to Jenkins CI/CD

1. [Setting up CI/CD Jenkins pipeline for kubernetes using Spring Boot Application on Virtual machine](#)
2. [Setting up kubernetes jenkins pipeline on AWS using Spring boot Application](#)

Part 1 - Deploy spring boot microservice on local kubernetes cluster using Kubespray

TERRAFORM 61

DOCKER 28

KUBERNETES 27

ANSIBLE 11

HELM-CHART 11

AWS 2

JENKINS 2

LINUX-COMMANDS 1

Tags

KUBERNETES 18

HELM-CHART 10

BLOGGING 4

QUARKUS 4

DOCKER 3

GITHUB 3

SSL 3

KUBESPRAY 2

SPRING-BOOT 2

ANSIBLE 1

HADOOP 1

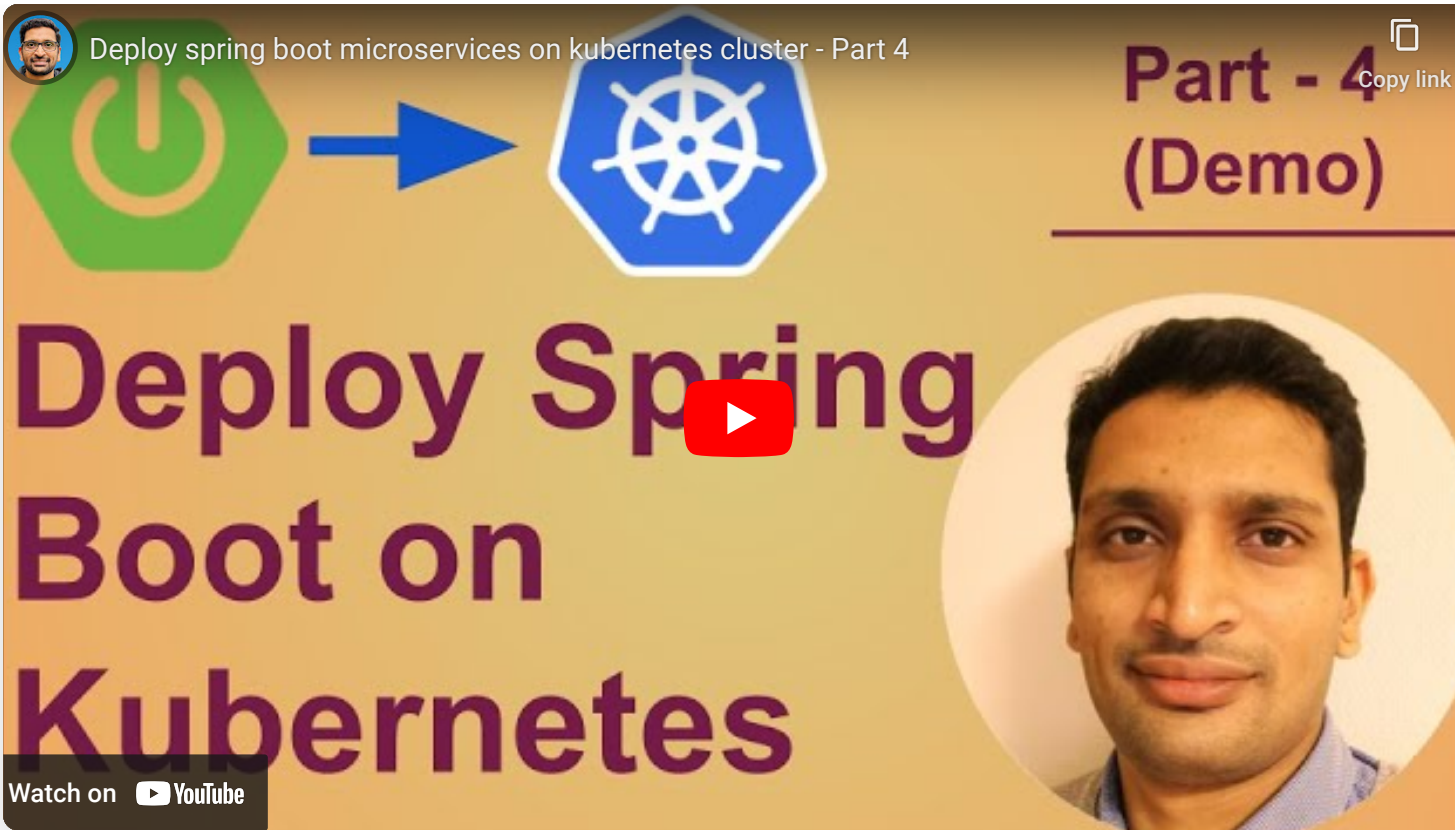
INDEX 1

NGINX 1

TERRAFORM 1

Recent Posts

- [Ansible Handlers Explained Real-World Use Cases & Examples](#)
- [Securing Sensitive Data in Terraform](#)
- [Boost Your AWS Security with Terraform : A Step-by-Step Guide](#)
- [How to Load Input Data from a File in Terraform?](#)



As we know our final goal is to "deploy spring boot micro-services on kubernetes" but before that, we need to build our spring boot application.

1. Create a Spring Boot Application using the Spring Boot Initializr

Let's head over to **spring initializr**(<https://start.spring.io/>) for generating skeleton for our spring boot application.
(Note: This complete setup has been tested with [Spring Boot Version v2.5.4](#))

Fill in the following details for spring boot application -

- Group - com.jhooq
- Artifacts - Jhooq-k8s

- Can Terraform be used to provision on-premises infrastructure?
- Fixing the Terraform Error creating IAM Role. MalformedPolicyDocument Has prohibited field Resource
- In terraform how to handle null value with default value?
- Terraform use module output variables as inputs for another module?

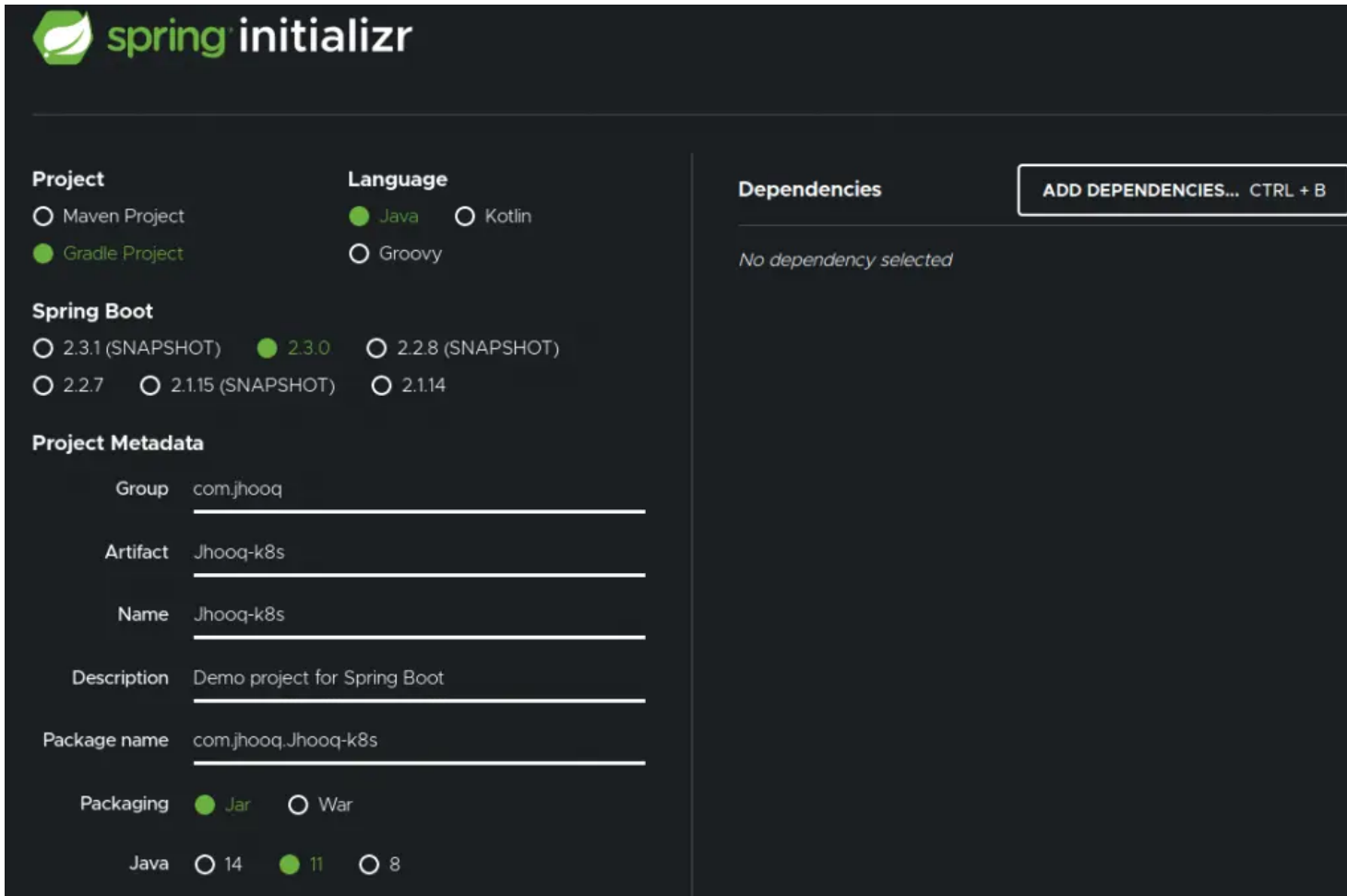
Rahul Wagh



Its all about Open Source and DevOps, here I talk about Kubernetes, Docker, Java, Spring boot and practices.

[READ MORE](#)

- Project - Gradle Project
- Packaging - Jar
- Java - 11

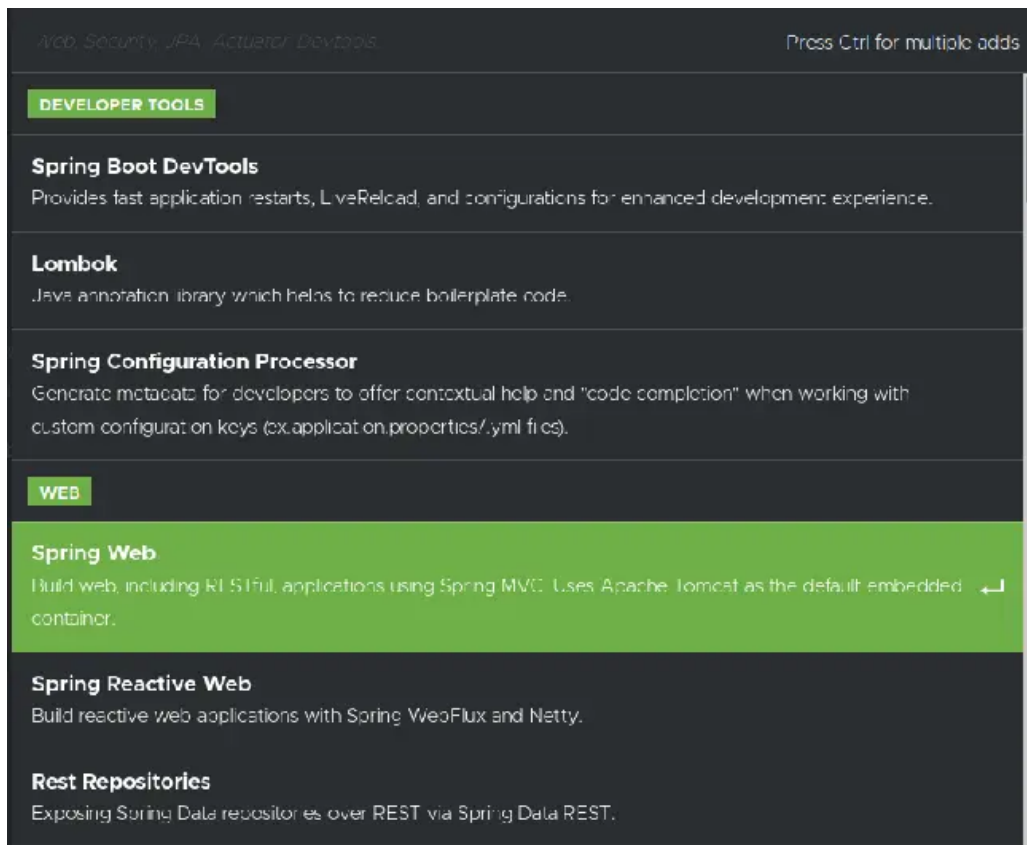


The image shows the Spring Initializr web form for creating a new project. The form is divided into several sections:

- Project:** Radio buttons for Maven Project and Gradle Project (selected).
- Language:** Radio buttons for Java (selected), Kotlin, and Groovy.
- Spring Boot:** Radio buttons for versions 2.3.1 (SNAPSHOT), 2.3.0 (selected), 2.2.8 (SNAPSHOT), 2.2.7, 2.1.15 (SNAPSHOT), and 2.1.14.
- Project Metadata:**
 - Group:** com.jhooq
 - Artifact:** Jhooq-k8s
 - Name:** Jhooq-k8s
 - Description:** Demo project for Spring Boot
 - Package name:** com.jhooq.Jhooq-k8s
 - Packaging:** Radio buttons for Jar (selected) and War.
 - Java:** Radio buttons for 14, 11 (selected), and 8.
- Dependencies:** A section with a button "ADD DEPENDENCIES... CTRL + B" and the text "No dependency selected".

Deploy Spring Boot microservices on kubernetes

Now click on the "Add Dependencies" sections to add "**Spring Web**" dependencies which include RESTful, application using Spring MVC and uses Apache Tomcat as the default embedded cotainer.



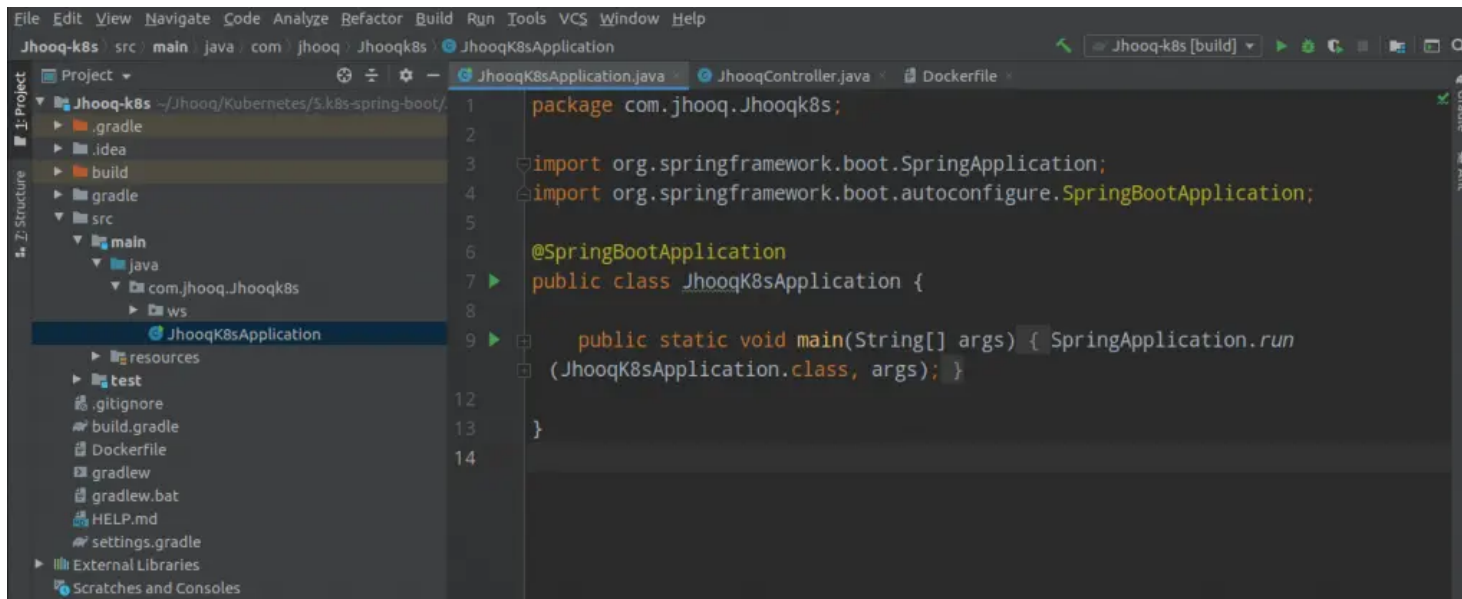
Spring initializr for building spring boot application

After you ready to go and click on "**Generate CTRL+G**" and save the zip file containing spring boot skeleton project.

2. Import the spring boot project into IDE

The next step would be to extract the zip file which you have downloaded in the - [Step 1](#)

Open you favorite IDE to import the project.(In my case i am using [IntelliJ Idea](#)). It should look something like this



IntelliJ import spring boot project

3. Implement your microservice

For the simplicity we will be implementing a very simple **"Hello World"** microservice and accessing it with either **web browser** or **curl** from the terminal.

Here is my RestController class called **JhooqController**

```
JAVA
1 package com.jhooq.JhooqK8s.ws;
2
3 import org.springframework.web.bind.annotation.GetMapping;
4 import org.springframework.web.bind.annotation.RestController;
5
6 @RestController
7 public class JhooqController {
8
```

```
9     @GetMapping("/hello")
10    public String hello() {
11        return "Hello - Jhooq-k8s";
12    }
13 }
```

Next step would be to build your Spring boot project using gradle

Run the following gradle build command -

```
1 $ ./gradlew build
2
3 > Task :test
4 2020-05-16 13:03:51.176 INFO 6235 --- [extShutdownHook] o.s.s.concurrent.ThreadPoolTaskExecutor
5
6 BUILD SUCCESSFUL in 6s
7 5 actionable tasks: 5 executed
```

Now you have build your Spring Boot project and its time to run the Spring boot application.

Switch to the "**build/libs**" directory and you should see your jar file over there .i.e. - **Jhooq-k8s-0.0.1-SNAPSHOT.jar**

```
1 $ cd build/libs/
```

```
1 $ ls -lart
2 total 17228
3 drwxrwxr-x 2 rahul rahul 4096 touko 16 13:03 .
```

```
4 -rw-rw-r-- 1 rahul rahul 17633090 touko 16 13:03 Jhooq-k8s-0.0.1-SNAPSHOT.jar
5 drwxrwxr-x 9 rahul rahul 4096 touko 16 13:03 ..
```

Now run the spring boot application

```
1 $ java -jar Jhooq-k8s-0.0.1-SNAPSHOT.jar
```

BASH

You should following successful start message

```
1 Started JhooqK8sApplication in 2.099 seconds (JVM running for 2.517)
```

BASH

4. Test the rest webserive

Once your spring boot application has started, now you can test the rest end point

```
1 $ curl http://localhost:8080/hello
```

BASH

```
1 Hello - Jhooq-k8s
```

BASH

5. Create Dockerfile for spring boot application

After testing your spring boot rest endpoint, now you need to create docker container of your Spring boot application.

To do that you need create the **Dockerfile**. You can create **Dockerfile** at any location. In this example we have created the Dockerfile at root level.

Docker file location in Spring boot project

Now you need write some docker command for creating Docker container

```
1 FROM openjdk:8-jdk-alpine
2 ARG JAR_FILE=build/libs/*.jar
3 COPY ${JAR_FILE} app.jar
4 ENTRYPOINT ["java", "-jar", "/app.jar"]
```

BASH

(**Note** - You need to be careful about the path to pick the **Jhooq-k8s-0.0.1-SNAPSHOT.jar** file. In this example our jar file location is **build/libs/**, So the docker file command should be ****_ARG JARFILE=build/libs/.jar****)

6. Create Docker registry at "<https://hub.docker.com/>"

Before we build our docker image, the first thing we need to do is create a docker registry at "<https://hub.docker.com/>". So that we will push our spring boot docker image into the registry. Later on, we will pull the same docker image during kubernetes deployment.

Goto "<https://hub.docker.com/>" and click on "Create Repository"

Docker hub create registry for spring boot container

Now it will prompt you for the docker registry name, in this example I am going to put my registry name as - **jhooq-k8s-springboot**.

Based on your privacy settings either you can keep the docker registry either **Public** or **Private**.

Docker hub create registry for spring boot container

Once you click on the save button you can see verify your docker registry onto the dashboard of your ["https://hub.docker.com/"](https://hub.docker.com/)

Created docker hub repository for spring boot kubernetes deployment

7. "docker build" && "docker push image"

Now you have all the per-requisites done for building docker image and pushing it to the ["https://hub.docker.com/"](https://hub.docker.com/).

But the first thing - Let's build our docker image and assign a suitable tag name to the docker image

Step 1 - Build docker image and tag it with the name **"jhooq-k8s-springboot"**

```
1 $ docker build -t jhooq-k8s-springboot .
```

BASH

Step 2 - Since our registry name on dockerhub is **"rahulwagh17/jhooq-k8s-springboot"** so we need to tag the build image one more time with docker registry name

```
1 $ docker tag jhooq-k8s-springboot rahulwagh17/kubernetes:jhooq-k8s-springboot
```

BASH

Step 3 - Push the docker image to docker hub

```
1 $ docker push rahulwagh17/kubernetes:jhooq-k8s-springboot
```

BASH

```
1 The push refers to repository [docker.io/rahulwagh17/kubernetes]
2 14170fe294f1: Pushed
3 ceaf9elebef5: Layer already exists
4 9b9b7f3d56a0: Layer already exists
5 f1b5933fe4b5: Layer already exists
6 jhooq-k8s-springboot: digest: sha256:b50ccfc5f80308795051b75fe69cc76f30dc37996b2c6065d2c978125621
```

BASH

Great now our spring boot docker image is onto the docker hub. Let's move onto our next step where we will be starting our kubernetes cluster to deploy "docker container" which we just pushed onto docker hub.

8. Start kubernetes cluster

If you do not have kubernetes cluster running than please **stop** here and I would highly recommend going through the guide on - [Kubespary – 12 Steps for Installing a Production Ready Kubernetes Cluster](#).

(Note: - I have tested the kubernetes cluster setup with [latest available version of kubernetes v1.22](#))

Alright so if you have reached so far then you have pretty much set up your kubernetes cluster to deploy spring boot application on kubernetes.

If you haven't run the kuberspray ansible then you could re-run it.

```
1 vagrant@amaster:~/kuberspray$ ansible-playbook -i inventory/mycluster/hosts.yml --become --become=
```

BASH

The ansible playbook will take some time to finish, once your ansible playbook run has finished successful then you should login to your kubernetes master node

```
1 $vagrant ssh kmaster
```

BASH

9. Deploy Spring Boot microservices on kubernetes

Now you have logged into your kubernetes master node, we need to create deployment using **kubectl**.

Step 1 - Use the following command to create **kubectl deployment** with the name **demo**

```
1 vagrant@kmaster:~$ kubectl create deployment demo --image=rahulwagh17/kubernetes:jhooq-k8s-spring
```

BASH

Check the deployment using following command

```
1 $ kubectl get deployments
2 NAME      READY    UP-TO-DATE    AVAILABLE    AGE
3 demo      1/1      1             1            5h20m
```

BASH

Step 2 - Expose the deployment to outside world

```
1 vagrant@kmaster:~$ kubectl expose deployment demo --type=LoadBalancer --name=demo-service --exter
```

BASH

```
1 service/demo-service exposed
```

BASH

Step 3 - Check the services

After exposing the service verify it with the following command

```
1 $ kubectl get service
```

BASH

It should return something similar

```
1 NAME          TYPE          CLUSTER-IP    EXTERNAL-IP    PORT(S)          AGE
2 demo-service  LoadBalancer 10.233.5.144   1.1.1.1        8080:31901/TCP   2s
3 kubernetes    ClusterIP      10.233.0.1    <none>         443/TCP          28h
```

BASH

Step 4 - Curl or test the rest endpoint

Since we setup the kubernetes cluster on laptop, so to access the spring boot rest point outside of the cluster we need to use vagrant box(kmaster's) ip address .i.e. - 100.0.0.2

So from your laptop which is outside the cluster you can use the following curl command

```
1 $ curl http://100.0.0.2:31901/hello
```

BASH

```
1 Hello - Jhooq-k8s
```

BASH

And if you are interested in accessing it from browser then please refer to the following screenshot

kubernetes expose service external ip

But in case if you have deployed the cluster on [AWS\(Amazon web service\)](#) or [Google Cloud](#) then use the EXTERNAL-IP which you got after running the command `$ kubectl get service` .i.e.

```
1 $ curl http://1.1.1.1:31901/hello
```

BASH

Great you have deployed your Spring Boot microservices on kubernetes.

10. Clean up the Deployment and Service

Once you are done with the deployment and exposing the deployment as a service. You can run the following command to clean the kubernetes cluster

```
1 $ kubectl delete service demo-service
```

BASH

```
1 $ kubectl delete deployments demo
```

BASH

Part 2 - Deploy spring boot microservice on Google Cloud Platform (**GCP**)



In the [Part-1](#) we setup local kubernetes cluster using kubespray and then deployed spring boot application on it.

Now in the Part-2 we are going to deploy the same Spring boot application on [Google Cloud Platform \(GCP\)](#)

Prerequisite

- Register to [Google Cloud Platform \(GCP\)](#) (Google offers free trial for 1 year or 300\$ credit to try out)

1. Create a project on Google Cloud

After the login goto **"My First Project"** and click on it.

Google cloud goto my first project

Now we need to click on the **New Project** to create a new project

Google cloud goto new project

Enter the name of the project. In our case, we are going to put the cluster name as **"jhooq-springboot-gc"**

Create new project on google cloud platform

After click that you should be able to create the project and it should be visible under your project section

Project name inside google cloud platform

2. Create Kubernetes Cluster on Google Cloud

Now after creating the project, we need to setup/create kubernetes cluster

Setting up or creating a new cluster inside the Google Cloud Platform(GCP) is pretty simple and it can be done with few clicks and some information.

If you haven't selected your project then please do select from "My First Project " menu. After than navigate to **"Kubernetes Engine"** menu in the left hand side and then click on **"Cluster"**

Create cluster on google cloud platform

Cluster information during cluster creating on google cloud

It should take around couple of minutes or more to setup the cluster. But Once its ready you should be able to see your cluster information as show below

google cloud kubernetes cluster is ready

3. Push spring boot docker image to google container registry(gcr.io)

In [Step 1](#) and [Step 2](#) we pretty much setup the kubernetes cluster on the google cloud but we need docker image which we are going to deploy on the kubernetes cluster.

And for that, we need to push the docker image to [Google Container Registry\(gcr.io\)](#). GCR is very much same as [docker-hub](#) but in the Google cloud world we use google container registry instead.

Before we push the docker image to the Google Container register, we need to understand the [Google Cloud SDK: Command Line Interface](#).

Because whenever you develop a spring boot or any other microservice you develop it on your local system or laptop but then you need to push or bring your application into the cloud environment from your local development machine(laptop).

So for that, we need to set up the [Google Cloud SDK: Command Line Interface](#). While writing this article I was working on Ubuntu 20.04 so these steps can be followed for Ubuntu. But for another operating system such as Windows or Mac OS you can follow the guide from google(Its pretty easy) - <https://cloud.google.com/sdk/docs/downloads-versioned-archives>

Step 1 - Download the google SDK from the <https://cloud.google.com/sdk/docs/downloads-versioned-archives>

Step 2 - Now you need to goto directory - **google-cloud-sdk-294.0.0-linux-x86_64/google-cloud-sdk** run the **install.sh**

```
1 ./install.sh
```

BASH

```
1 To help improve the quality of this product, we collect anonymized usage data
2 and anonymized stacktraces when crashes are encountered; additional information
3 is available at <https://cloud.google.com/sdk/usage-statistics>. This data is
4 handled in accordance with our privacy policy
5 <https://policies.google.com/privacy>. You may choose to opt in this
6 collection now (by choosing 'Y' at the below prompt), or at any time in the
7 future by running the following command:
8
9 gcloud config set disable_usage_reporting false
```

BASH

```
1 Do you want to help improve the Google Cloud SDK (y/N)? y
```

BASH

```
1 Do you want to continue (Y/n)? y
```

BASH

```
1 To install or remove components at your current SDK version [294.0.0], run:
2 $ gcloud components install COMPONENT_ID
3 $ gcloud components remove COMPONENT_ID
4
5 To update your SDK installation to the latest version [294.0.0], run:
6 $ gcloud components update
```

BASH

```
7
8
9 Modify profile to update your $PATH and enable shell command
10 completion?
```

```
1 Do you want to continue (Y/n)? Y
```

BASH

Step 3 - Enter the path to RC file to update, leave it blank if you do not want to change the path

```
1 The Google Cloud SDK installer will now prompt you to update an rc
2 file to bring the Google Cloud CLIs into your environment.
3
4 Enter a path to an rc file to update, or leave blank to use
5 [/home/rahul/.bashrc]:
6 Backing up [/home/rahul/.bashrc] to [/home/rahul/.bashrc.backup].
7 [/home/rahul/.bashrc] has been updated.
8
9 ==> Start a new shell for the changes to take effect.
```

BASH

Step 4 - Run the following command to setup the source

```
1 $ source ~/.bashrc
```

BASH

Step 5 - Initialize the google cloud on your local development computer

```
1 $ gcloud init
```

BASH

(If you are setting it up for the first time then it will redirect you to google account login page for authentication.)

In the next step, it will ask you to *Pick configuration to use*: (Select or enter 1 because we already created the cluster and now we need re-initialize that configuration here again.)

```
1 Pick configuration to use:
2 [1] Re-initialize this configuration [default] with new settings
3 [2] Create a new configuration
4 Please enter your numeric choice: 1
```

BASH

In the next step it will ask to *Choose the account to perform the operation*

```
1 Choose the account you would like to use to perform operations for
2 this configuration:
3 [1] rahul.wagh17@gmail.com
4 [2] Log in with a new account
5 Please enter your numeric choice: 1
```

BASH

Now we need to select our project which we have created on google cloud web interface. So we will go with - **jhooq-springboot-gc**.

```
1 You are logged in as: [rahul.wagh17@gmail.com].
2
3 Pick cloud project to use:
4 [1] cryptic-tower-275912
5 [2] jhooq-gc-springboot-k8s
6 [3] jhooq-springboot-gc
7 [4] quantum-boulder-278914
8 [5] Create a new project
```

BASH

```
9 Please enter numeric choice or text value (must exactly match list
10 item): 3
```

In the next step it will ask for **default Compute Region and Zone** ?

Go for default and press Y

```
1 Your current project has been set to: [jhooq-springboot-gc].
2
3 Do you want to configure a default Compute Region and Zone? (Y/n)? y
```

BASH

It will show you a long list of the compute zone so you can select the nearest one and input the zone number

```
1 Too many options [74]. Enter "list" at prompt to print choices fully.
2 Please enter numeric choice or text value (must exactly match list
3 item): 14
```

BASH

Alright, now we are ready to push our docker image to Google cloud registry.

Step 1 - In the [Step 5](#) we created a docker file, so switch to that directory. After that we need to **tag** the docker image with the project name .i.e. **_jhooq-springboot-k8s-demo_**

```
1 $ docker build -t gcr.io/jhooq-springboot-k8s-demo/jhooq-springboot:v1 .
```

BASH

```
1 Sending build context to Docker daemon 28.31MBB
2 Step 1/4 : FROM openjdk:8-jdk-alpine
3 ---> a3562aa0b991
```

BASH

```
4 Step 2/4 : ARG JAR_FILE=build/libs/*.jar
5 ---> Using cache
6 ---> ab3e5f96d4dc
7 Step 3/4 : COPY ${JAR_FILE} app.jar
8 ---> Using cache
9 ---> ede5735c296c
10 Step 4/4 : ENTRYPOINT ["java","-jar","/app.jar"]
11 ---> Using cache
12 ---> adaf5b0a60a2
13 Successfully built adaf5b0a60a2
14 Successfully tagged gcr.io/jhooq-springboot-k8s-demo/jhooq-springboot:v1
```

Step 2 - Push the docker image to the google container registry

```
1 $ docker push gcr.io/jhooq-springboot-k8s-demo/jhooq-springboot:v1
2 ```bash
3
4 ```bash
5 The push refers to repository [gcr.io/jhooq-springboot-k8s-demo/jhooq-springboot]
6 14170fe294f1: Mounted from jhooq-gc-springboot-k8s/jhooq-springboot
7 ceaf9elebef5: Layer already exists
8 9b9b7f3d56a0: Layer already exists
9 f1b5933fe4b5: Layer already exists
10 v1: digest: sha256:b50ccfc5f80308795051b75fe69cc76f30dc37996b2c6065d2c978125621aeefa size: 1159
```

Now we have pushed our spring boot docker image to the Google Container registry. You can verify the image by logging into the Google Cloud selecting project *jhooq-springboot-k8s-demo* and navigating to the option *Container Registry*

spring boot docker image inside google container registry

4. Deploy the spring boot microservice inside kubernetes cluster running on Google Cloud

Alright so till now you setup the kubernetes cluster then you pushed the spring-boot docker image to Google container registry and now its time to play with some kubernetes command.

Now we are going to use the Google Cloud shell to deploy docker image inside Kubernetes cluster.

Step 1- Connect to cloudshell. Click on the connect button for the cloudshell

Google cloud connect to cloud shell

Step 2- Now we need to do kubectl deployment, use the following command for the creating the deployment

```
1 $ kubectl create deployment jhooq-springboot --image=gcr.io/jhooq-springboot-k8s-demo/jhooq-springboot BASH
```

```
1 deployment.apps/jhooq-springboot created BASH
```

Step 3- expose deployment on external IP

```
1 $ kubectl expose deployment jhooq-springboot --type=LoadBalancer --port 80 --target-port 8080 BASH
```

```
1 service/jhooq-springboot exposed BASH
```

Now you need to find the external IP of service, so that you can access the spring boot microservice outside of the network

```
1 $ kubectl get service
```

BASH

```
1 NAME                TYPE          CLUSTER-IP    EXTERNAL-IP    PORT(S)          AGE
2 jhooq-springboot     LoadBalancer  10.64.8.231   34.72.142.92   80:32537/TCP     11m
3 kubernetes           ClusterIP     10.64.0.1     <none>         443/TCP          3h51m
```

BASH

Well if you reached till step then i would say you did hell of job with kubernetes cluster setup on the Google Cloud Plateform.

Now you can access your rest end point with the external IP *.i.e. <http://34.72.142.92:80/hello>*

google cloud kubernetes external ip

Conclusion

Let's conclude on what we have done through out this article on [Deploy Spring Boot microservices](#) on [kubernetes](#)

Local kubernetes cluster setup and microservice deployment

1. Bootstrapped Spring Boot application from **spring initializr**(<https://start.spring.io/>)
2. Implemented "Hello World" microservice
3. Build the Docker Image of Spring boot Application
4. Pushed the Docker image to <https://hub.docker.com/>

5. Setup [kubernetes](#) cluster using [kubespray](#)
6. Used `kubectl create deployment demo -image=rahulwagh17/kubernetes:jhooq-k8s-springboot` to create the deployment
7. Expose deployment on external-ip **`kubectl expose deployment demo -type=LoadBalancer -name=demo-service -external-ip=1.1.1.1 -port=8080`**
8. Finally tested the rest endpoint which is deployed as Spring Boot microservice inside kubernetes cluster.

Google Cloud Platform(GCP) - kubernetes setup and microservice deployment

1. Setting up google cloud project
2. Creating a Kubernetes cluster with 3 nodes
3. Building and tagging docker image of spring boot application
4. Pushing the spring boot docker image to [Google Container Registry\(GCR\)](#)
5. Creating kubectl deployment for spring boot
6. Exposing the deployment on external IP address

Let us know about the issues during your exercise. I will be happy to troubleshoot your problems.

Learn more On Kubernetes -

1. [Setup kubernetes on Ubuntu](#)
2. [Setup Kubernetes on CentOS](#)
3. [Setup HA Kubernetes Cluster with Kubespray](#)
4. [Setup HA Kubernetes with Minikube](#)
5. [Setup Kubernetes Dashboard for local kubernetes cluster](#)
6. [Setup Kubernetes Dashboard On GCP\(Google Cloud Platform\)](#)
7. [How to use Persistent Volume and Persistent Volume Claims in Kubernetes](#)
8. [Deploy Spring Boot Microservice on local Kubernetes cluster](#)
9. [Deploy Spring Boot Microservice on Cloud Platform\(GCP\)](#)
10. [Setting up Ingress controller NGINX along with HAproxy inside Kubernetes cluster](#)
11. [CI/CD Kubernetes | Setting up CI/CD Jenkins pipeline for kubernetes](#)

12. [kubectl export YAML | Get YAML for deployed kubernetes resources\(service, deployment, PV, PVC....\)](#)
13. [How to setup kubernetes jenkins pipeline on AWS?](#)
14. [Implementing Kubernetes liveness, Readiness and Startup probes with Spring Boot Microservice Application?](#)
15. [How to fix kubernetes pods getting recreated?](#)
16. [How to delete all kubernetes PODS?](#)
17. [How to use Kubernetes secrets?](#)
18. [Share kubernetes secrets between namespaces?](#)
19. [How to Delete PV\(Persistent Volume\) and PVC\(Persistent Volume Claim\) stuck in terminating state?](#)
20. [Delete Kubernetes POD stuck in terminating state?](#)

Posts in this series

- [Kubernetes Cheat Sheet for day to day DevOps operations?](#)
- [Delete Kubernetes POD stuck in terminating state?](#)
- [How to Delete PV\(Persistent Volume\) and PVC\(Persistent Volume Claim\) stuck in terminating state?](#)
- [Share kubernetes secrets between namespaces?](#)
- [How to use Kubernetes secrets?](#)
- [How to delete all kubernetes PODS?](#)
- [kubernetes pods getting recreated?](#)
- [Implementing Kubernetes liveness, Readiness and Startup probes with Spring Boot Microservice Application?](#)
- [kubectl export yaml OR How to generate YAML for deployed kubernetes resources](#)
- [Kubernetes Updates](#)
- [CI/CD Kubernetes | Setting up CI/CD Jenkins pipeline for kubernetes](#)
- [Kubernetes cluster setup with Jenkins](#)
- [How to use Persistent Volume and Persistent Claims | Kubernetes](#)
- [How to fix ProvisioningFailed persistentvolume controller no volume plugin matched](#)
- [Fixing – Cannot bind to requested volume: storageClassName does not match](#)

- [Fixing – pod has unbound immediate persistentvolumeclaims or cannot bind to requested volume incompatible accessmode](#)
- [How to fix kubernetes dashboard forbidden 403 error – message services https kubernetes-dashboard is forbidden User](#)
- [How to fix Kubernetes – error execution phase preflight \[preflight\]](#)
- [Deploy Spring Boot microservices on kubernetes?](#)
- [How to fix – ansible_memtotal_mb minimal_master_memory_mb](#)
- [How to use kubespray – 12 Steps for Installing a Production Ready Kubernetes Cluster](#)
- [How to setup kubernetes on CentOS 8 and CentOS 7](#)
- [How to fix – How to fix - ERROR Swap running with swap on is not supported. Please disable swap](#)
- [14 Steps to Install kubernetes on Ubuntu 20.04\(bento/ubuntu-20.04\), 18.04\(hashicorp/bionic64\)](#)
- [Kubernetes Dashboard | Kubernetes Admin GUI | Kubernetes Desktop Client](#)
- [Install Kubernetes with Minikube](#)



Copyright 2024 COPYRIGHT © 2019–2020, JHOOQ; ALL RIGHTS RESERVED.. All Rights Reserved

