

Spring boot app

Sunday, January 2, 2022 9:59 PM

Deploy Spring boot app on Kubernetes and also on GCP.

Deploy spring boot microservice on local kubernetes cluster using Kubespray

Create a spring boot application.

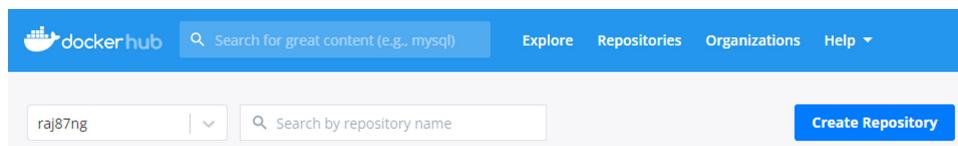
Create a Docker file:

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

Create a 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"



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

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

A screenshot of the 'Create Repository' dialog on Docker Hub. At the top, it says 'Using 1 of 1 private repositories. [Get more](#)'. Below that, there's a section for 'Create Repository' with two input fields: 'Name' (containing 'raj87ng') and 'Tag' (containing 'raj-k8s-springboot'). A note below says 'Test project to deploy in K8 cluster'. To the right, there's a 'Pro tip' section with CLI commands: 'docker tag local-image:tagname new-repo:tagname' and 'docker push new-repo:tagname'. It also says 'Make sure to change tagname with your desired image repository tag.' Under 'Visibility', there are two options: 'Public' (selected) and 'Private'. The 'Public' option includes a note 'Appears in Docker Hub search results'. At the bottom, there are 'Cancel' and 'Create' buttons.

raj87ng > Repositories > raj-k8s-springboot >

Using 1 of 1 private repositories. [Get more](#)

General Tags Builds Collaborators Webhooks Settings

Advanced Image Management
View all your images and tags in this repository, clean up unused content, recover untagged images. Available with Pro, Team and Business subscriptions. [View preview](#)

raj87ng / raj-k8s-springboot
Test project to deploy in K8 cluster [Edit](#)

Last pushed: never

Docker commands
To push a new tag to this repository,
`docker push raj87ng/raj-k8s-springboot:tagname`

[Public View](#)

Tags and Scans
This repository is empty. When it's not empty, you'll see a list of the most recent tags here.

VULNERABILITY SCANNING - DISABLED [Enable](#)

Automated Builds
Manually pushing images to Hub? Connect your account to GitHub or Bitbucket to automatically build and tag new images whenever your code is updated, so you can focus your time on creating.
Available with Pro, Team and Business subscriptions.

[Upgrade to Pro](#) [Learn more](#)

raj87ng | [Create Repository](#)

Search by repository name

raj87ng / raj-k8s-springboot	Not Scanned	0 stars	0 downloads	Public
raj87ng / testingapp	Not Scanned	0 stars	0 downloads	Private

“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/>".
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 “raj-k8s-springboot”

\$ docker build -t raj-k8s-springboot .

BASH

Step 2 - Since our registry name on dockerhub is “raj87ng/raj-k8s-springboot” so we need to tag the build image one more time with docker registry name

\$ docker tag raj-k8s-springboot raj87ng/kubernetes:raj-k8s-springboot

BASH

Step 3 - Push the docker image to docker hub

\$ docker push raj87ng/kubernetes:raj-k8s-springboot

BASH

The push refers to repository [docker.io/raj87ng/kubernetes]14170fe294f1: Pushed

ceaf9e1ebef5: Layer already exists

9b9b7f3d56a0: Layer already exists

f1b5933fe4b5: Layer already exists

raj-k8s-springboot: digest: sha256:b50ccfc5f80308795051b75fe69cc76f30dc37996b2c6065d2c978125621aefa size: 1159

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.

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 - Kubespray – [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](#))

How to use kubespray – [12 Steps for Installing a Production Ready Kubernetes Cluster](#)

```
kubectl create deployment demo --image=rahulwagh17/kubernetes:jhooq-k8s-springboot
```

Check the deployment using following command

```
$ kubectl get deployments
```

```
NAME READY UP-TO-DATE AVAILABLE AGE
```

```
demo 1/1 115h20m
```

Step 2 - Expose the deployment to outside world

```
vagrant@kmaster:~$ kubectl expose deployment demo --type=LoadBalancer --name=demo-service --external-ip=1.1.1.1 --port=8080
```

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

Step 3 - Check the services

After exposing the service verify it with the following command

```
$ kubectl get service
```

It should return something similar

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

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

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

Hello - Jhooq-k8s

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

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.
```

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

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

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

```
$ kubectl delete deployments demo
```

Deploy Spring boot service on Google Cloud Platform (GCP)

Create a project on Google Cloud

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

The screenshot shows the Google Cloud Platform dashboard. At the top, there's a blue header bar with the text "Google Cloud Platform" and "My First Project". Below the header, there are two main navigation tabs: "Home" (indicated by a house icon) and "DASHBOARD" (indicated by a bar chart icon). The "DASHBOARD" tab is currently selected.

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

This screenshot shows the "Select a project" screen. At the top right, there's a "NEW PROJECT" button with a gear icon. Below it is a search bar with the placeholder "Search projects and folders". Underneath the search bar, there are two tabs: "RECENT" (which is selected) and "ALL". A table follows, displaying one project entry:

Name	ID
✓ My First Project <small>?</small>	quantum-boulder-278914

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

From <<https://jhooq.com/deploy-spring-boot-microservices-on-kubernetes/#part-2>>

This screenshot shows the "New Project" creation dialog. At the top left, it says "New Project". On the right, there's a search bar. The main area contains a warning message: "⚠ You have 22 projects remaining in your quota. Request an increase or delete projects. [Learn more](#)". Below this is a "MANAGE QUOTAS" link. The next section is for "Project name *", which has "jhooq-springboot-gc" entered. To the right of the input field is a question mark icon. Below the project name, it says "Project ID: jhooq-springboot-gc. It cannot be changed later." with an "EDIT" link. The next section is for "Location *", with "No organization" selected. To the right is a "BROWSE" button. Below the location input is a note: "Parent organization or folder". At the bottom of the dialog are two buttons: "CREATE" (in a blue box) and "CANCEL".

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

From <<https://jhooq.com/deploy-spring-boot-microservices-on-kubernetes/#part-2>>

Select a project

 NEW PROJECT

 Search projects and folders

RECENT ALL

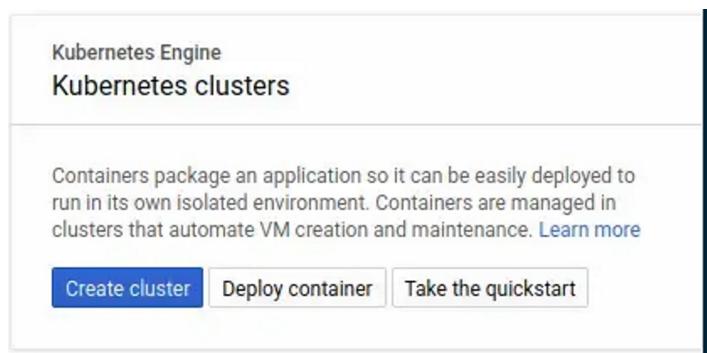
Name	ID
jhooc-springboot-gc 	jhooc-springboot-gc

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"



Kubernetes Engine

Kubernetes clusters

Containers package an application so it can be easily deployed to run in its own isolated environment. Containers are managed in clusters that automate VM creation and maintenance. [Learn more](#)

[Create cluster](#) [Deploy container](#) [Take the quickstart](#)

Cluster basics

The new cluster will be created with the name, version, and in the location you specify here. After the cluster is created, name and location can't be changed.

 To experiment with an affordable cluster, try [My first cluster](#) in the Cluster set-up guides

Name 

Location type

- Zonal
 Regional

Zone  

Specify default node locations [?](#)

Current default: us-central1-c

Master version

Choose Release Channel to get automatic GKE upgrades as new versions are ready. Choose a static version to upgrade manually in the future. [Learn more](#).

Release channel

Static version

Static version

1.14.10-gke.36 (default) [▼](#)

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

A Kubernetes cluster is a managed group of VM instances for running containerized applications. [Learn more](#)

Filter by label or name

<input type="checkbox"/> Name ^	Location	Cluster size	Total cores	Total memory	Notifications	Labels
<input checked="" type="checkbox"/> jhooq-springboot-demo	us-central1-c	3	3 vCPUs	11.25 GB		Connect

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

```
./install.sh
```

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

```
gcloud config set disable_usage_reporting false
```

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

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

```
To install or remove components at your current SDK version [294.0.0], run:
```

```
$ gcloud components install COMPONENT_ID  
$ gcloud components remove COMPONENT_ID
```

```
To update your SDK installation to the latest version [294.0.0], run:
```

```
$ gcloud components update
```

```
Modify profile to update your $PATH and enable shell command  
completion?
```

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

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

```
BASH
The Google Cloud SDK installer will now prompt you to update an rc
file to bring the Google Cloud CLIs into your environment.
```

```
Enter a path to an rc file to update, or leave blank to use
[/home/rahul/.bashrc]:
Backing up [/home/rahul/.bashrc] to [/home/rahul/.bashrc.backup].
[/home/rahul/.bashrc] has been updated.
```

```
==> Start a new shell for the changes to take effect.
```

Step 4 - Run the following command to setup the source

```
BASH
$ source ~/.bashrc
```

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

```
BASH
$ gcloud init
```

(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.)

BASH

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

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

BASH

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

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

```
You are logged in as: [rahul.wagh17@gmail.com].
```

```
Pick cloud project to use:  
[1] cryptic-tower-275912  
[2] jhooq-gc-springboot-k8s  
[3] jhooq-springboot-gc  
[4] quantum-boulder-278914  
[5] Create a new project  
Please enter numeric choice or text value (must exactly match list  
item): 3
```

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

Go for default and press Y

BASH

```
Your current project has been set to: [jhooq-springboot-gc].
```

```
Do you want to configure a default Compute Region and Zone? (Y/n)? y
```

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

BASH

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

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-sprinboot-k8s-demo****

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

```
Sending build context to Docker daemon 28.31MBB
```

```
Step 1/4 : FROM openjdk:8-jdk-alpine
```

```
--> a3562aa0b991
```

```
Step 2/4 : ARG JAR_FILE=build/libs/*.jar
```

```
--> Using cache
```

```
--> ab3e5f96d4dc
```

```
Step 3/4 : COPY ${JAR_FILE} app.jar
```

```
--> Using cache
```

```
--> ede5735c296c
```

```
Step 4/4 : ENTRYPOINT ["java","-jar","/app.jar"]
```

```
--> Using cache
```

```
--> adaf5b0a60a2
```

```
Successfully built adaf5b0a60a2
```

```
Successfully tagged gcr.io/jhooq-springboot-k8s-demo/jhooq-springboot:v1
```

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

```
BASH
$ docker push gcr.io/jhooq-springboot-k8s-demo/jhooq-springboot:v1
````bash
The push refers to repository [gcr.io/jhooq-springboot-k8s-demo/jhooq-springboot]
14170fe294f1: Mounted from jhooq-gc-springboot-k8s/jhooq-springboot
ceaf9e1ebef5: Layer already exists
9b9b7f3d56a0: Layer already exists
f1b5933fe4b5: Layer already exists
v1: digest: sha256:b50ccfc5f80308795051b75fe69cc76f30dc37996b2c6065d2c978125621aefa 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*

Container Registry	Repositories	REFRESH				
Images	jhooq-springboot-gc					
Settings	<input type="text" value="Filter"/> <span>All hostnames ▾</span>	<table><thead><tr><th>Name ^</th><th>Hostname</th></tr></thead><tbody><tr><td> jhooq-springboot</td><td>gcr.io</td></tr></tbody></table>	Name ^	Hostname	jhooq-springboot	gcr.io
Name ^	Hostname					
jhooq-springboot	gcr.io					

## 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

Name ^	Location	Cluster size	Total cores	Total memory	Notifications	Labels
<input checked="" type="checkbox"/> jhooq-springboot-demo	us-central1-c	3	3 vCPUs	11.25 GB		<button>Connect</button> <span>edit</span> <span>trash</span>

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

```
BASH
$ kubectl create deployment jhooq-springboot --image=gcr.io/jhooq-springboot-k8s-demo/jhooq-springboot:v1
```

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

**Step 3-** expose deployment on external IP

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

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

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

```
$ kubectl get service
```

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

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>

Ref : <https://jhoog.com/>