Containerization of a Web Application Using Docker and Container Orchestration using Kubernetes

Md Zahirul Islam, Kantish Roy Chowdhury, and Mansur Uddin Khan
High Integrity System (MSc.)
Frankfurt University of Applied Sciences

Abstract

Due to the portability and ease of reproducible, the containerized application development became trend among software organizations. As the number of containers in a system increases the deployment and managing the containers became difficult. Therefore, container orchestration tools like Kubernetes offers automatic deployment, scaling, and management of containerized applications. In this paper, we have explained containerized application deployment into Kubernetes cluster with Jenkins.

1 Introduction

In past few years, containers have added new dimensions to the way the software build, ship, and maintain. A containerized process is highly portable, and reproducible which enable us to move, and scale container application more easier than before. Docker is the leading container platform to build, ship, and run application. It provides loosely isolated environment called container to package and run application [5].

Managing the life-cycle of containers, especially in a large and dynamic environment, requires container orchestration tool. The most popular orchestration tool is Kubernetes. With kubernetes, we can provision, and deploy containers, scale up, and down number of containers based on demand, load balancing of the service discovery among containers, and many other things [6].

Our goal is to deploy the containerized web application to kubernetes cluster in public cloud. To bridge the gaps between development, and operational activities (build, test and deployment) of our application we have implemented Continuous Integration and Continuous Deployment (CI/CD) process which is the backbone of modern DevOps practice.

The scope of this paper would work as an installation guide that explains step by step instructions of deployment process of web API. It covers the deployment of container from local to public cloud Kubernetes cluster of both manual and automated approach.

This paper discusses following contributions in different sections: Section 2, describes system architecture, tools, and methodologies used in this project. We have explained the containerization process of our application in section 3. Section 4 explains the container orchestration process. In this section, we have shown the container deployment and scaling of Kubernetes to local and cloud environment. Jenkins installation and pipeline configurations are included in section 5. We have discussed the deployment results in section 6. Finally, section 7 concludes this report.

2 System Overview

Our application is a web REST API service to fetch dummy data from an in-memory database of application. We have deployed containerized application in kubernetes cluster of Google Cloud Platform (GCP) using Jenkins pipeline. The deployment covers both local, and cloud Kubernetes cluster. Finally, we have automated whole deployment procedures with Jenkins pipeline.

2.1 System Architecture

The backbone of the system architecture is Jenkins which handles container deployment to google cloud kubernetes cluster. We have used github as source control repository, and dockerhub as a container repository. Container creation, and deployment are handled by Jenkins pipeline. Pipeline execution is triggered as soon as github source is updated by developer. Jenkins uses plugins to build application and interact with dockerhub, github, and google kubernetes engine. Kubernetes cluster has a loadbalancer service which route traffic to available pods. Our high level system architecture is illustrated in figure 1.

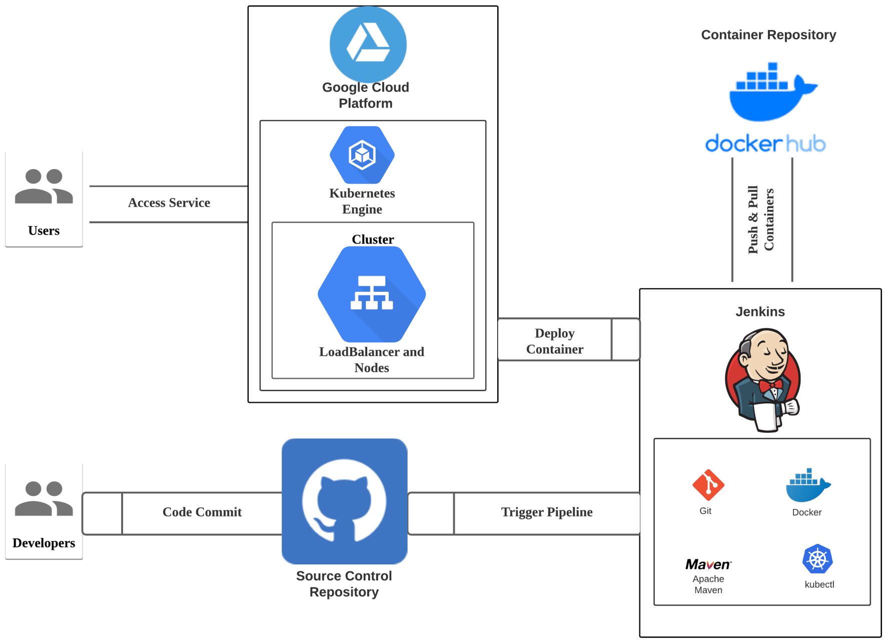


Figure 1: System Architecture

2.2 Tools and Methodologies

In this project, we have used number of tools including some DevOps tools, and the techniques which are listed below:

```
Programming Language: Java
Database: H2
Framework: Spring Boot
Application Build Tool: Apache Maven
DevOps Tools: Docker, Jenkins, Kubernetes
Cloud Platform: Google Cloud Platform
Source Control Repository: GitHub
Container Repository: DockerHub
API Test: Postman
```

3 Containerization

To create and deploy software faster and more efficiently, containerization became a major trend in software development [1]. We have used Docker to create container for our web application. For our local environment, we have installed Docker for Desktop, for detail installation instructions please refer to reference [2]. Docker builds images automatically by executing instructions written in Dockerfile which is stored in our source control repository [13]. For a container repository, we have used Dockerhub. With the following commands we create, run and push Docker container image to Docherhub.

```
- Build image: docker build -f Dockerfile -t library-cloud-api .
- Run image: docker run -t -d library-cloud-api
- Tag image: docker tag library-cloud-api zahirulislam/library-cloud-api:1.0
- Login Dockerhub: docker login -u zahirulislam
- Push to Dockerhub: docker push zahirulislam/library-cloud-api
```

4 Container Orchestration with Kubernetes

The containerized application is highly portable, and reproducible which is easy to move and scale across clouds [7]. We have used Kubernetes as an orchestrator tool to manage, scale, and maintain our containerized application. Kubernetes comes along with Docker engine of *Docker for Desktop tool* where we just needed to enable few settings to use Kubernetes of Docker engine and explained in section 4.2.1.

4.1 Kubernetes Architecture

Following are the key components of a kubernetes cluster (see figure 2):

- Cluster: A Kubernetes cluster is a set of node machines for running containerized applications which has two parts: control plane and the compute machines, or nodes.
- Nodes: Physical or vitural machine
- pods: Each node runs pods which are groups of co-located containers that share some resources.

To run a kubernetes cluster, we require a container registry where container images are stored (in our case Dockerhub), underlying infrastructure where kubernetes can run, and a persistent storage to manage the application data attached to a cluster (we did not used it).

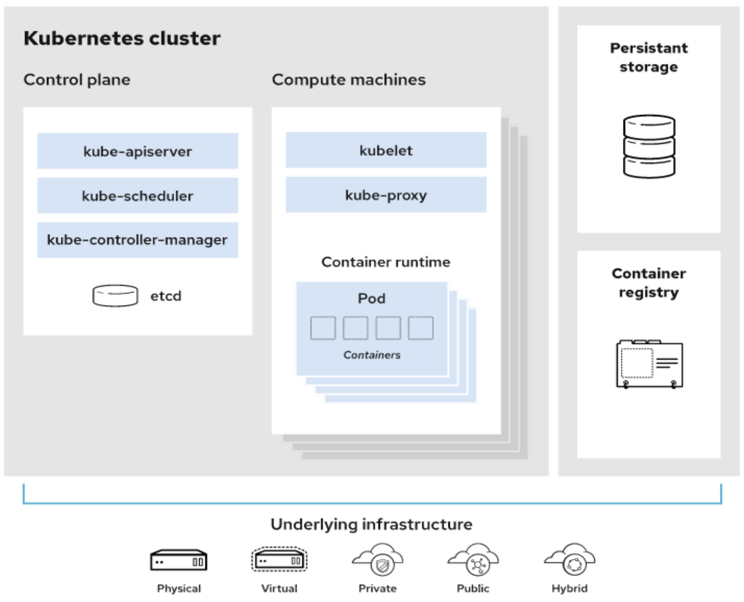


Figure 2: Kubernetes Architecture [8]

4.2 Deployment

Workloads are scheduled as deployment which are scalable group of pods maintained by kubernetes automatically. At first, we have deployed our containers in local kubernetes cluster, then we have deployed in Kubernetes engine of GCP, and finally, we have automated whole deployment using Jenkins.

4.2.1 Local Cluster

For local kubernetes cluster depoloyment, we need to check and enable following features include respective commands, and instructions given below:

```
- Enable Hypervisor: Enable-WindowsOptionalFeature -Online
                     -FeatureName Microsoft-Hyper-V -All
- Enable Kubernetes: From setting in Docker for desktop tool
```

Kubernetes objects for local environment are described in manifest file called *deployment_local.yml* file [10]. It has following two objects:

- Deployments: It describes a scalable group of three identical pods.
- NodePort: It is a service to route traffic from port 8080 on your host to port 8080 inside the pods.

With the following commands, we create kubernetes deployments and services in local kubernetes cluster where name of both deployment, and service objects is *library-cloud-api*.

```
- kubectl create -f deployment_local.yml
```

We can scale up and down number of pods either with manual or automatic approach. For example, if we want to increase current number of pods from three to five, then we can execute following scripts. This is manual approach.

```
kubectl scale deployments/library-cloud-api --replicas=5
```

We can instruct kubernetes engine to scale up and scale down automatically to number of pods based on CPU usage with following script:

```
kubectl autoscale deployment library-cloud-api \
--cpu-percent=50 --min=1 --max=10
```

The above script will increase and decrease number of pods between 1 to 10 based on CPU utilization where we mentioned threshold utilization is 50%.

We have checked and discussed the deployment results in section 6.1.

4.2.2 Google Kubernetes Cluster

In section 3, we have explained how to push container image in Dockerhub. We took same image from Dockerhub and deploy it in Google Kubernetes Engine. We can deploy our containers in Kubernetes cluster by following instructions given below, and executing related commands in GCP. This is a manual approach, however, we will automate whole deployment process with Jenkins which is explained in section 5.

At first, we have created a project with name *my-k8s-project* and our project id is *my-k8s-project-300415* (id will be going to be used in script). We will use cloud shell and Google command line tool *gcloud* to execute our commands.

```
- Pull Image from Dockerhub : docker pull zahirulislam
                              /library-cloud-api:1.0

- Tag image : docker tag zahirulislam/library-cloud-api:1.0 \
              gcr.io/my-k8s-project-300415/library-cloud-api:1.0
```

```
- Authenticate gcloud Tool : gcloud auth configure-docker

- Push Image to GCR : docker push gcr.io/my-k8s-project-300415/
                      library-cloud-api:1.0

- Create deployment.yaml Files with nano Command. Contents of the
  file is same as the deployment_cloud_manually.yaml file in our
  github repository. It contains both service and deployment object.

- Create a Cluster  : gcloud container clusters create \
                      library-cloud-api-k8-cluster \
                      --num-nodes 1 --enable-basic-auth \
                      --issue-client-certificate --zone \
                      europe-west1

- Check Nodes  : kubectl get nodes

- Create Deployment  : kubectl apply -f deployment.yaml

- Check pods  : kubectl get pods

- Check Services  : kubectl get services
```

We have checked and discussed the outcome of this deployment in section 6.2.

5 CI/CD with Jenkins

Jenkins is a popular open source tool written in Java that allows to deliver software continuously by integrating building, testing and deployment technologies. We will include step-by-step instructions to configure Jenkins.

5.1 Installation

Firstly, we have created the virtual machine (VM) in GCP to configure Jenkins server with following steps:

```
- Create a VM instance (debian is default OS) in GCP
- Select both http and https firewall rules from option list
- Select region "europe-west1" (we have selected same region
       as for kubernetes cluster but any region can be selected)
```

Jenkins requires following dependencies: JDK (Jenkins need java to run), Git, Maven, wget (to download jenkins package), docker, kubectl

5.1.1 JDK

Jenkins needs Java to run, so we have installed JDK. Execute following commands to install *openjdk-8*.

```
    - sudo apt-get update
    - sudo apt-get install software-properties-common
    - sudo apt-add-repository
      'deb http://security.debian.org/debian-security stretch/updates main'
    - sudo apt-get update
    - sudo apt-get install openjdk-8-jdk
```

5.1.2 Git and Maven

We need to install Git for source control and Maven for building application in Jenkins server with the following commands.

```
    - sudo apt-get install git
    - sudo apt install maven
```

5.1.3 Docker

We have used Docker to create container of our web application, so Jenkins server needs Docker to be installed by following commands [4].

```
    - sudo apt-get update
    - sudo apt-get install \
         apt-transport-https \
         ca-certificates \
         curl \
         gnupg-agent \
         software-properties-common
    - curl -fsSL https://download.docker.com/linux/debian/gpg
        | sudo apt-key add -
    - sudo apt-key fingerprint 0EBFCD88
    - sudo add-apt-repository
         "deb [arch=amd64] https://download.docker.com/linux/debian
        $(lsb_release -cs) stable"
    - sudo apt-get update
    - sudo apt-get install docker-ce docker-ce-cli containerd.io
```

We also need to add users to docker group, and restart Jenkins server with following command

```
    - sudo usermod -aG docker $USER
    - sudo service jenkins restart
```

5.1.4 kubectl

It is a command line tool to control kubernetes cluster. We have installed *kubectl* with following commands:

```
    sudo apt-get update && sudo apt-get install \
         -y apt-transport-https gnupg2 curl
    curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg
         | sudo apt-key add -
    echo "deb https://apt.kubernetes.io/ kubernetes-xenial main"
         | sudo tee -a /etc/apt/sources.list.d/kubernetes.list
    sudo apt-get update
    sudo apt-get install -y kubectl
```

5.1.5 Jenkins

With the following commands [3], we have installed Jenkins in server. We have used *wget* to download Jenkins from internet. Prior to install Jenkins, we have installed wget.

```
    sudo apt-get install wget
    wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key
        | sudo apt-key add -
    sudo sh -c 'echo deb https://pkg.jenkins.io/debian-stable binary/ > \
        /etc/apt/sources.list.d/jenkins.list'
    sudo apt-get update
    sudo apt-get install jenkins
```

5.2 Activation

Browse the Jenkins by clicking the external IP of Jenkins VM from google cloud. But the site might not accessible as port 8080 might not be already opened. So, we must create a firewall rule to allow all traffics on port 8080. To do that, follow the steps mentioned below.

```
- Go to firewall and create firewall rules
- Provide any name of the rule
- Select target as "All instances in the network"
- Provide IP range 0.0.0.0/0
- Select tcp port 8080
```

For the first time, Jenkins needs to be activated by providing it's secret stored in server. Secret is stored in file *initialAdminPassword* in directory */var/lib/jenkins/secrets*. Follow below steps to fetch the secret.

```
cd /var/lib/jenkins/secrets
nano initialAdminPassword
copy the secret
```

After submitting the secrets, select install Jenkins option, then it will start installation. After installation, it will let us create another user, and password which we can later usable for the login Jenkins from browser.

5.3 Configure Pipeline

We want our Jenkins pipeline to handle the following task: pull source code from github as soon as git repository is updated, build the application, build docker image and deploy image to google Kubernetes cluster.

5.3.1 GitHub Webhook

With webhook, Jenkins pipeline would be triggered whenever github repository gets updated.

```
- Go to GitHub -> Setting -> Webhook
- Provide payload url "http://34.76.14.187:8080/github-webhook/"
- Select content type "application/json"
- Select "Let me select individual option" and "Push Request" option
```

5.3.2 Plugins

Install following plugins

```
- Docker Pipeline
- Kubernetes
- Kubernetes Continuous Deploy
- Google kubernetes engine
```

5.3.3 Credentials

Go to *Manage Credentials* in Jenkins, and create credentials for the following service.

```
- DockerHub
- Google Cloud Kubernetes Engine
```

We also need to ensure that the service account in GCP has access to the Google Kubernetes Engine. We can ensure that with following steps:

```
- Go to IAM & Admin in Google Cloud
- Assign the role "Kubernetes Engine Admin"
```

5.3.4 Pipeline

To create a pipeline from Jenkins user interface, click *New Item* and then follow below steps:

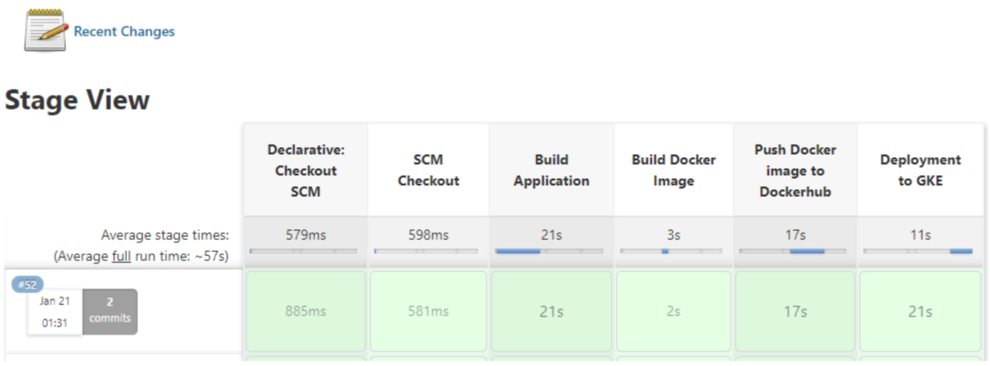
```
- Select pipeline
- Select github project (and give project url)
- Select build triggers +it["github hook trigger for GITScm polling"]
- In pipeline select "Pipeline script from scm"
- Provide git repository url and branch information
```

As a pipleline syntax, we have used declarative pipeline, and pipeline scripts are written in Jenkinsfile. The Jenkinsfile has been stored in source control repository [14] which serves the single source of truth for our pipeline. We have defined following five stages in our Jenkinsfile:

```
- SCM Checkout: Checkout source code from github specified branch.
- Build Application: Build the application with Maven and create
  executable Jar file
- Build Docker Image: With the help of Dockerfile [13]  &
  Docker Pipeline plugin, it create Docker image
```

- Push Docker image to Dockerhub: Using Docker Pipeline plugin, it pushes the image to Dockerhub (container repository)
- Deployment to GKE: Finally deploy to Google Kubernetes cluster. It uses a Kubernetes manifest file stored in source control repository [12] to create deployments and services.

Execution of Jenkins pipeline will display the status, and time what it took to complete each stage defined in Jenkinsfile [14], see the figure (3.



Recent Changes

Stage View

	Declarative: Checkout SCM	SCM Checkout	Build Application	Build Docker Image	Push Docker image to Dockerhub	Deployment to GKE
Average stage times: (Average full run time: ~57s)	579ms	598ms	21s	3s	17s	11s
#52 Jan 21 01:31 2 commits	885ms	581ms	21s	2s	17s	21s

Figure 3: Jenkins Pipeline Execution

Outcomes of pipeline execution has been discussed in section 6.3.

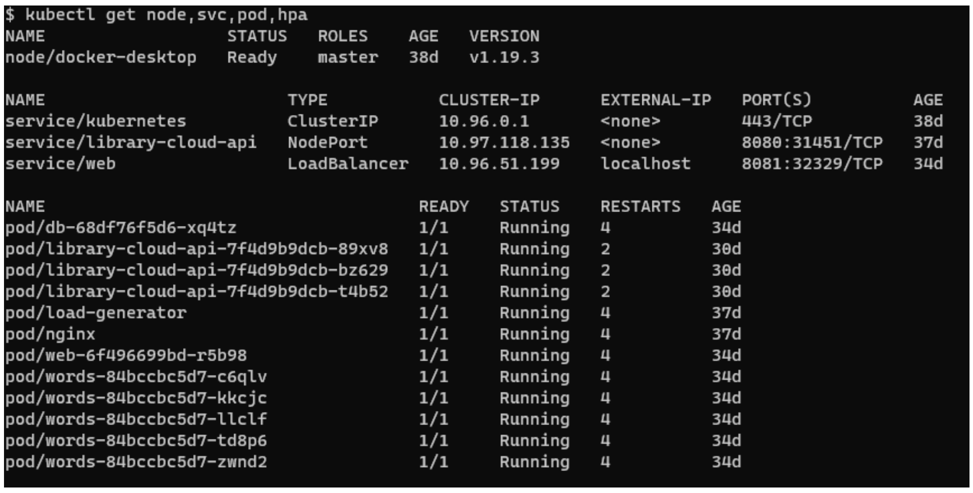
6 Results and Discussion

Although our containerized application has been deployed in cloud kubernetes cluster with Jenkins, but prior to use Jenkins we also have tested our deployments in local and cloud kubernetes clusters manually. In this section, we will discuss the outcomes of deployments in both local, and cloud kubernetes cluster.

6.1 Local Kubernetes Cluster

Deployment in local environment has been explained in section 4.2.1. With the following commands, we can check whether our deployments were successful.

- `kubectl get node,svc,pod`



```
$ kubectl get node,svc,pod,hpa
NAME                  STATUS   ROLES    AGE   VERSION
node/docker-desktop   Ready    master   38d   v1.19.3

NAME                          TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
service/kubernetes            ClusterIP      10.96.0.1       <none>        443/TCP          38d
service/library-cloud-api     NodePort       10.97.118.135   <none>        8080:31451/TCP   37d
service/web                   LoadBalancer   10.96.51.199    localhost     8081:32329/TCP   34d

NAME                                      READY   STATUS    RESTARTS   AGE
pod/db-68df76f5d6-xq4tz                   1/1     Running   4          34d
pod/library-cloud-api-7f4d9b9dcb-89xv8    1/1     Running   2          30d
pod/library-cloud-api-7f4d9b9dcb-bz629    1/1     Running   2          30d
pod/library-cloud-api-7f4d9b9dcb-t4b52    1/1     Running   2          30d
pod/load-generator                        1/1     Running   4          37d
pod/nginx                                 1/1     Running   4          37d
pod/web-6f496699bd-r5b98                  1/1     Running   4          34d
pod/words-84bccbc5d7-c6qlv                1/1     Running   4          34d
pod/words-84bccbc5d7-kkcjc                1/1     Running   4          34d
pod/words-84bccbc5d7-llclf                1/1     Running   4          34d
pod/words-84bccbc5d7-td8p6                1/1     Running   4          34d
pod/words-84bccbc5d7-zwnd2                1/1     Running   4          34d
```

Figure 4: Local cluster

The above command will list all nodes, pods, and services running in local kubernetes cluster like in figure 4.

From figure 4, we see our service *"service/library-cloud-api"* has been deployed with cluster IP *10.97.118.135* and no external-ip which is obvious case for local deployment. We further see that service type is NodePort and its port is 8080 which means application container is listening on port 8080. Finally, we can browse application (see figure 5) with following url: *localhost:31451/api/books*

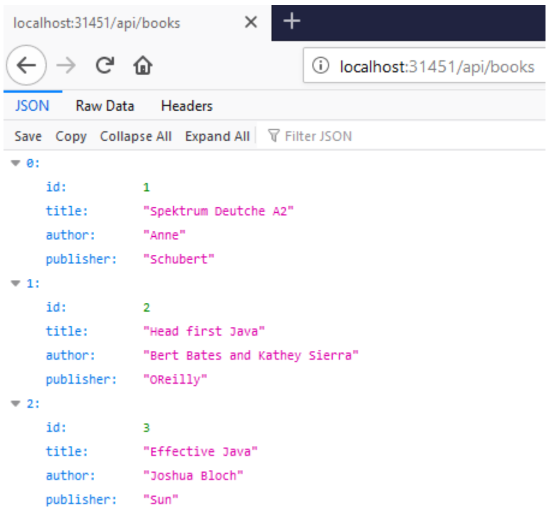
6.2 Cloud Environment

In section 4.2.2, we have explained the deployment procedures in Google kubernetes cluster. We can check our deployments with following commands:

```
kubectl get node,pod,service
```

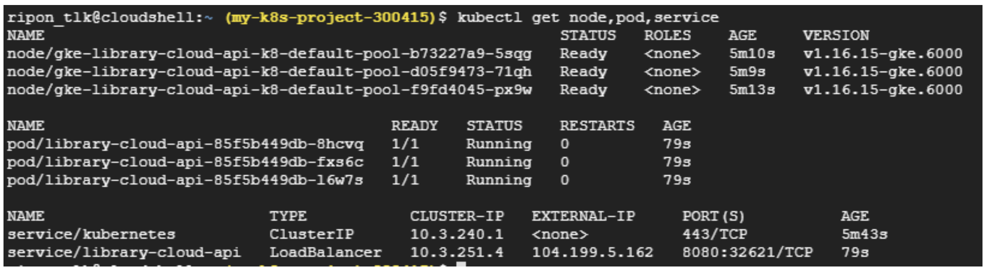
The above command will list nodes, pods and services which are created in kubernetes cluster (see the figure 6).

In above figure 6, we can see our loadbalancer service is created with external-ip is 104.199.5.162 and port 8080. So, we can browse our application (see figure 7) with following url: *http://104.199.5.162:8080/api/books*



```
▼ 0:
    id:         1
    title:      "Spektrum Deutche A2"
    author:     "Anne"
    publisher:  "Schubert"
▼ 1:
    id:         2
    title:      "Head first Java"
    author:     "Bert Bates and Kathey Sierra"
    publisher:  "OReilly"
▼ 2:
    id:         3
    title:      "Effective Java"
    author:     "Joshua Bloch"
    publisher:  "Sun"
```

Figure 5: Access to Service

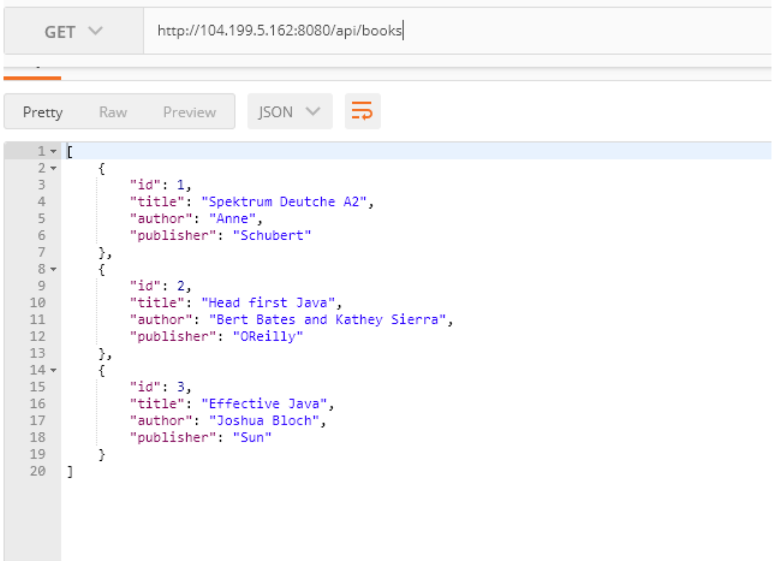


```
ripon_tlk@cloudshell:~ (my-k8s-project-300415)$ kubectl get node,pod,service
NAME                                                    STATUS   ROLES    AGE     VERSION
node/gke-library-cloud-api-k8-default-pool-b73227a9-5sqg   Ready    <none>   5m10s   v1.16.15-gke.6000
node/gke-library-cloud-api-k8-default-pool-d05f9473-71qh   Ready    <none>   5m9s    v1.16.15-gke.6000
node/gke-library-cloud-api-k8-default-pool-f9fd4045-px9w   Ready    <none>   5m13s   v1.16.15-gke.6000

NAME                                         READY   STATUS    RESTARTS   AGE
pod/library-cloud-api-85f5b449db-8hcvq   1/1     Running   0          79s
pod/library-cloud-api-85f5b449db-fxs6c   1/1     Running   0          79s
pod/library-cloud-api-85f5b449db-l6w7s   1/1     Running   0          79s

NAME                            TYPE           CLUSTER-IP    EXTERNAL-IP     PORT(S)          AGE
service/kubernetes              ClusterIP      10.3.240.1    <none>          443/TCP          5m43s
service/library-cloud-api       LoadBalancer   10.3.251.4    104.199.5.162   8080:32621/TCP   79s
```

Figure 6: Kubernetes Cluster in google cloud



```
http://104.199.5.162:8080/api/books
```

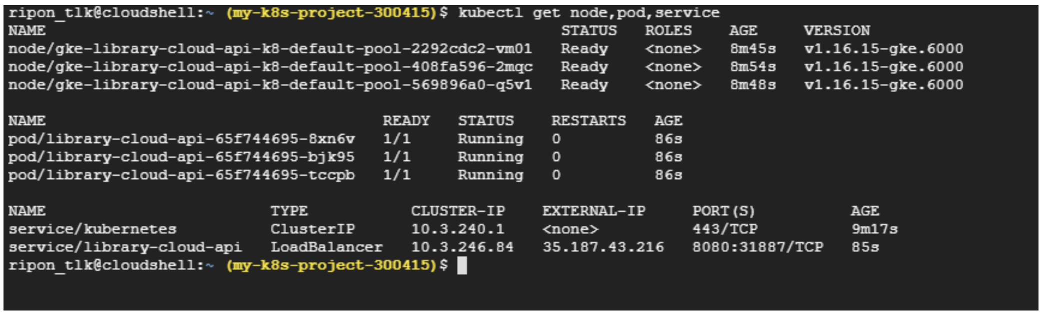
```
[
    {
        "id": 1,
        "title": "Spektrum Deutche A2",
        "author": "Anne",
        "publisher": "Schubert"
    },
    {
        "id": 2,
        "title": "Head first Java",
        "author": "Bert Bates and Kathey Sierra",
        "publisher": "OReilly"
    },
    {
        "id": 3,
        "title": "Effective Java",
        "author": "Joshua Bloch",
        "publisher": "Sun"
    }
]
```

Figure 7: Access to the service

6.3 Jenkins

We have discussed how to configure Jenkins and create Jenkins pipeline in section 5. In this section we will check our deployments done by Jenkins.

Figure 8 shows the deployment status. It displays nodes, pods, and services which are created by Jenkins pipeline execution.



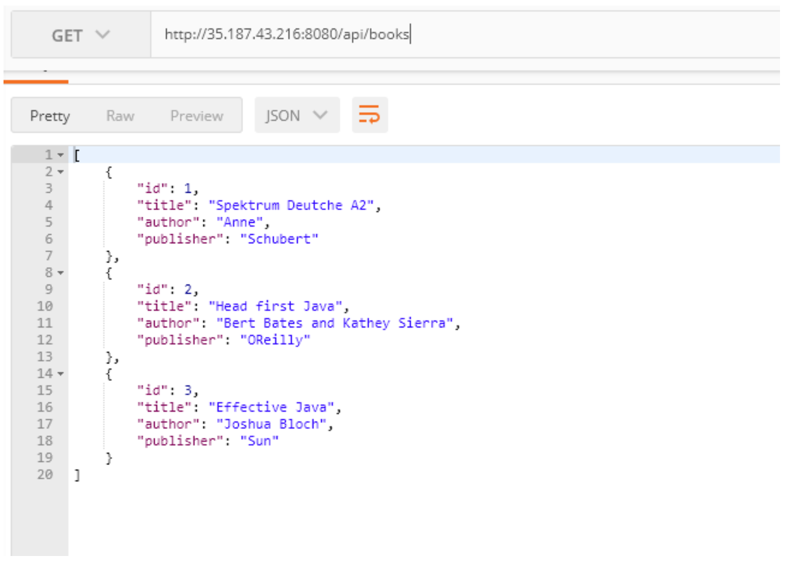
```
ripon_tlk@cloudshell:~ (my-k8s-project-300415)$ kubectl get node,pod,service
NAME                                                          STATUS   ROLES    AGE    VERSION
node/gke-library-cloud-api-k8-default-pool-2292cdc2-vm01     Ready    <none>   8m45s  v1.16.15-gke.6000
node/gke-library-cloud-api-k8-default-pool-408fa596-2mqc     Ready    <none>   8m54s  v1.16.15-gke.6000
node/gke-library-cloud-api-k8-default-pool-569896a0-q5v1     Ready    <none>   8m48s  v1.16.15-gke.6000

NAME                                       READY   STATUS    RESTARTS   AGE
pod/library-cloud-api-65f744695-8xn6v     1/1     Running   0          86s
pod/library-cloud-api-65f744695-bjk95     1/1     Running   0          86s
pod/library-cloud-api-65f744695-tccpb     1/1     Running   0          86s

NAME                          TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)          AGE
service/kubernetes            ClusterIP      10.3.240.1     <none>          443/TCP          9m17s
service/library-cloud-api     LoadBalancer   10.3.246.84    35.187.43.216   8080:31887/TCP   85s
ripon_tlk@cloudshell:~ (my-k8s-project-300415)$
```

Figure 8: Deployment by Jenkins

Now our service is exposed and accessible publicly, (see figure 9).



```
http://35.187.43.216:8080/api/books
```

```
[
    {
        "id": 1,
        "title": "Spektrum Deutche A2",
        "author": "Anne",
        "publisher": "Schubert"
    },
    {
        "id": 2,
        "title": "Head first Java",
        "author": "Bert Bates and Kathey Sierra",
        "publisher": "OReilly"
    },
    {
        "id": 3,
        "title": "Effective Java",
        "author": "Joshua Bloch",
        "publisher": "Sun"
    }
]
```

Figure 9: Access to the services (deployed by Jenkins)

7 Conclusion

This paper has discussed the container deployment to Kubernetes cluster and explained our approaches to do so. It includes step-by-step instructions to configure and install necessary tools and services. Finally, we have covered the procedures to test deployments, and discussed the deployment outcomes.

In future, we wish to explore more cloud platforms (which includes private and hybrid cloud) and their services. Also we want to work with containerized micro-service deployment and its service discovery.

References

[1] ”What is containerization?”, by IBM Cloud Kubernetes Service (https://www.ibm.com/cloud/learn/containerization)

[2] ”Docker Installation in different operating system”, by Docker official documents page (https://docs.docker.com/get-docker/)

[3] ”Jenkins installation in Debian”, by Jenkins users handbook (https://www.jenkins.io/doc/book/installing/linux/debianubuntu)

[4] ”Install Docker Engine on Debian”, by Docker official documents page (https://docs.docker.com/engine/install/debian/)

[5] ”Docker overview”, by Docker official documents page (https://docs.docker.com/get-started/overview/)

[6] ”What Is Container Orchestration?”, by Isaac Eldridge on Jul. 17th, 2018 (https://blog.newrelic.com/engineering/container-orchestration-explained/)

[7] ”Container Orchestration with Kubernetes”, by Docker official documents page (https://docs.docker.com/get-started/orchestration/)

[8] ”Introduction to Kubernetes architecture”, by Redhat (https://www.redhat.com/en/topics/containers/kubernetes-architecture)

[9] ”Source control repository in Github” (https://github.com/zahirulislam04/library-cloud-api)

[10] ”Kubernetes manifest file for local deployment” (https://github.com/zahirulislam04/library-cloud-api/blob/main/deployment$_{l}ocal.yml$)

[11] ”Kubernetes manifest file for deployment in Cloud manualy” (https://github.com/zahirulislam04/library-cloud-api/blob/main/deployment$_{c}loud_{m}anually.yaml$)

[12] ”Kubernetes manifest file for deployment by Jenkins” (https://github.com/zahirulislam04/library-cloud-api/blob/main/deployment.yaml)

[13] ”Dockerfile in source control repository” (https://github.com/zahirulislam04/library-cloud-api/blob/main/Dockerfile)

[14] ”Jenkinsfile for Jenkins pipeline stored in source control repository” (https://github.com/zahirulislam04/library-cloud-api/blob/main/Jenkinsfile)