**Deploy an application using Kubernetes cluster on IBM Cloud**

You can deploy and manage a Kubernetes cluster in IBM Cloud Kubernetes Service. Learn how to automate the deployment, operation, scaling of containerized apps in a cluster.

You configure a custom Kubernetes cluster to deploy and test a Guestbook app (Written in Go language) in IBM Cloud Kubernetes Service.

Make sure you’ve completed all the prerequisites given in “IBM Cloud Pre-requisites.docx” before executing this assignment.

**Delete an existing cluster - GUI**

1. You can delete the cluster that you created during the workshop and create a new one, to make sure you see only the Kubernetes resources that are created as part of this assignment
2. Login to IBM Cloud with your credentials
3. In the Dashboard, click on Kubernetes Clusters
4. In the right most corner of your cluster, click on the 3 dots and select Delete
5. Will take 10-15 minutes to delete the cluster
6. Refresh your browser to check the latest status

**Create a cluster - GUI**

1. Login to IBM Cloud with your credentials
2. Go to Catalog
3. Click on Containers from left hand side menu and click on Kubernetes Service
4. Click on Create
5. Select a **Free** cluster from “Select a plan”
6. Leave the Cluster name, Tags and Resource Group to the default populated values
7. Select **North America** for Geography
8. Select **Dallas** for Metro
9. Click on Create cluster
10. It will take 15 to 20 minutes to create your cluster
11. Refresh your browser to check the latest status
12. Make sure your cluster status is Normal (green colour dot) in your Cloud Dashboard and Cluster home page

**Cluster Configuration**

1. Open Command Prompt as an Administrator
2. Login to IBM Cloud

*ibmcloud login -a* [*https://cloud.ibm.com*](https://cloud.ibm.com)

1. Enter your IBM Cloud user id (email id) and password
2. Login to your IBM Cloud account from browser - [*https://cloud.ibm.com*](https://cloud.ibm.com)
3. From the Dashboard, click on Kubernetes Clusters
4. Click on your cluster, for e.g., **mycluster**, to go to its home page
5. Click on Access tab
6. Target the Kubernetes Service region in which you want to work. Copy from the browser and paste on your command prompt.

*ibmcloud ks region-set us-south*

1. Get the command to set the environment variable and download the Kubernetes configuration files. Copy from the browser and paste on your command prompt.

*ibmcloud ks cluster-config mycluster*

1. You will get an output similar to this

*SET KUBECONFIG=/Users/$USER/.bluemix/plugins/container-service/clusters/mycluster/kube-config-hou02-mycluster.yml*

1. Copy the output from step 10 and paste it on your command prompt
2. Make sure *KUBECONFIG* environment variable is set properly by running

*echo % KUBECONFIG%*

1. This will make your kubectl client point to your Kubernetes cluster. Once your client is configured, you are ready to deploy your first application - guestbook.

# **Deploy your application – version 1**

In this assignment, you will deploy an application called guestbook that has already been built and uploaded to DockerHub under the name ibmcom/guestbook:v1.

1. Start by running guestbook

*$ kubectl run –-generator=run-pod/v1 guestbook --image=ibmcom/guestbook:v1*

This action will take a bit of time

To check the status of the running application, you can use *$kubectl get pods*

You should see output similar to the following:

NAME READY STATUS RESTARTS AGE

guestbook-59bd679fdc-bxdg7 0/1 ContainerCreating 0 1m

Eventually, the status should show up as Running

*$ kubectl get pods*

NAME READY STATUS RESTARTS AGE

guestbook-59bd679fdc-bxdg7 1/1 Running 0 1m

The end result of the run command is not just the pod containing our application containers, but a Deployment resource that manages the lifecycle of those pods.

1. Expose the deployment as a Service, so it can be accessed through the IP of the worker node. The guestbook application listens on port 3000.

*$ kubectl expose deployment guestbook --type="NodePort" --port=3000*

service "guestbook" exposed

1. To find the port used on the worker node, examine your new service

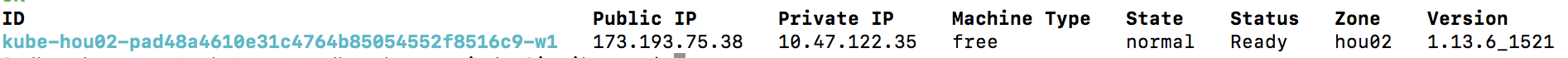
*$ kubectl get service guestbook*

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

guestbook NodePort 10.10.10.253 <none> 3000:30438/TCP 1m

You can see that the <nodeport> is 30438. You can see in the output, the port mapping from 3000 inside the pod exposed to the cluster on port 30438. This port is automatically chosen and could be different for you.

1. guestbook is now running on your cluster, and exposed to the internet. You need to find out where it is accessible. The worker nodes running in IBM Cloud Kubernetes Service get external IP addresses.
2. *$ ibmcloud ks workers mycluster*



You can see that Public IP is 173.193.75.38.

1. Now that you have both the address and the port, you can access the application in the web browser at http://<public-IP>:<nodeport>. In this case, it is <http://173.193.75.38:30438>
2. Output :

****

**Congratulations, you've now deployed an application to Kubernetes !**

1. If you want to remove the deployment, do the following:

* To remove the deployment, use $ kubectl delete deployment guestbook.
* To remove the service, use $ kubectl delete service guestbook.

# **Deploy your application – version 2**

In this section, you'll learn how to update the number of instances a deployment has, and how to safely roll out an update of your application on Kubernetes.

# **Step 1. Scale your app with replicas**

A replica is a copy of a pod that contains a running service. By having multiple replicas of a pod, you can ensure your deployment has the available resources to handle increasing load on your application.

1. kubectl provides a scale subcommand to change the size of an existing deployment. Let's increase our capacity from a single running instance of guestbook up to 10 instances.
2. *$ kubectl scale --replicas=10 deployment guestbook*

deployment "guestbook" scaled

Kubernetes will now try to match the desired state of 10 replicas by starting 9 new pods with the same configuration as the first.

1. To see your changes being rolled out, run

*$ kubectl rollout status deployment guestbook*

Waiting for rollout to finish: 1 of 10 updated replicas are available...

Waiting for rollout to finish: 2 of 10 updated replicas are available...

Waiting for rollout to finish: 3 of 10 updated replicas are available...

Waiting for rollout to finish: 4 of 10 updated replicas are available...

Waiting for rollout to finish: 5 of 10 updated replicas are available...

Waiting for rollout to finish: 6 of 10 updated replicas are available...

Waiting for rollout to finish: 7 of 10 updated replicas are available...

Waiting for rollout to finish: 8 of 10 updated replicas are available...

Waiting for rollout to finish: 9 of 10 updated replicas are available...

deployment "guestbook" successfully rolled out

The rollout might occur so quickly that the above messages might not display

1. Once the rollout has finished, ensure your pods are running

*$ kubectl get pods*

NAME READY STATUS RESTARTS AGE

guestbook-562211614-1tqm7 1/1 Running 0 1d

guestbook-562211614-1zqn4 1/1 Running 0 2m

guestbook-562211614-5htdz 1/1 Running 0 2m

guestbook-562211614-6h04h 1/1 Running 0 2m

guestbook-562211614-ds9hb 1/1 Running 0 2m

guestbook-562211614-nb5qp 1/1 Running 0 2m

guestbook-562211614-vtfp2 1/1 Running 0 2m

guestbook-562211614-vz5qw 1/1 Running 0 2m

guestbook-562211614-zksw3 1/1 Running 0 2m

guestbook-562211614-zsp0j 1/1 Running 0 2m

You should see output listing 10 replicas of your deployment

# **Step 2. Update and roll back apps**

Kubernetes allows you to do rolling upgrade of your application to a new container image. This allows you to easily update the running image and also allows you to easily undo a rollout if a problem is discovered during or after deployment.

In the previous section, you used an image with a v1 tag. For upgrade, you will use the image with the v2 tag.

To update and roll back:

1. Using kubectl, you can now update your deployment to use the v2 image. kubectl allows you to change details about existing resources with the *set* subcommand. You can use it to change the image being used.

*$ kubectl set image deployment/guestbook guestbook=ibmcom/guestbook:v2*

Note that a pod could have multiple containers, each with its own name. Each image can be changed individually or all at once, by referring to the name. In the case of our guestbook Deployment, the container name is also guestbook.

1. To see your changes being rolled out, run

*$ kubectl rollout status deployment/guestbook*

Waiting for rollout to finish: 2 out of 10 new replicas have been updated...

Waiting for rollout to finish: 3 out of 10 new replicas have been updated...

Waiting for rollout to finish: 3 out of 10 new replicas have been updated...

Waiting for rollout to finish: 3 out of 10 new replicas have been updated...

Waiting for rollout to finish: 4 out of 10 new replicas have been updated...

Waiting for rollout to finish: 4 out of 10 new replicas have been updated...

Waiting for rollout to finish: 4 out of 10 new replicas have been updated...

Waiting for rollout to finish: 4 out of 10 new replicas have been updated...

Waiting for rollout to finish: 4 out of 10 new replicas have been updated...

Waiting for rollout to finish: 5 out of 10 new replicas have been updated...

Waiting for rollout to finish: 5 out of 10 new replicas have been updated...

Waiting for rollout to finish: 5 out of 10 new replicas have been updated...

Waiting for rollout to finish: 6 out of 10 new replicas have been updated...

Waiting for rollout to finish: 6 out of 10 new replicas have been updated...

Waiting for rollout to finish: 6 out of 10 new replicas have been updated...

Waiting for rollout to finish: 7 out of 10 new replicas have been updated...

Waiting for rollout to finish: 7 out of 10 new replicas have been updated...

Waiting for rollout to finish: 7 out of 10 new replicas have been updated...

Waiting for rollout to finish: 7 out of 10 new replicas have been updated...

Waiting for rollout to finish: 8 out of 10 new replicas have been updated...

Waiting for rollout to finish: 8 out of 10 new replicas have been updated...

Waiting for rollout to finish: 8 out of 10 new replicas have been updated...

Waiting for rollout to finish: 8 out of 10 new replicas have been updated...

Waiting for rollout to finish: 9 out of 10 new replicas have been updated...

Waiting for rollout to finish: 9 out of 10 new replicas have been updated...

Waiting for rollout to finish: 9 out of 10 new replicas have been updated...

Waiting for rollout to finish: 1 old replicas are pending termination...

Waiting for rollout to finish: 1 old replicas are pending termination...

Waiting for rollout to finish: 1 old replicas are pending termination...

Waiting for rollout to finish: 9 of 10 updated replicas are available...

Waiting for rollout to finish: 9 of 10 updated replicas are available...

Waiting for rollout to finish: 9 of 10 updated replicas are available...

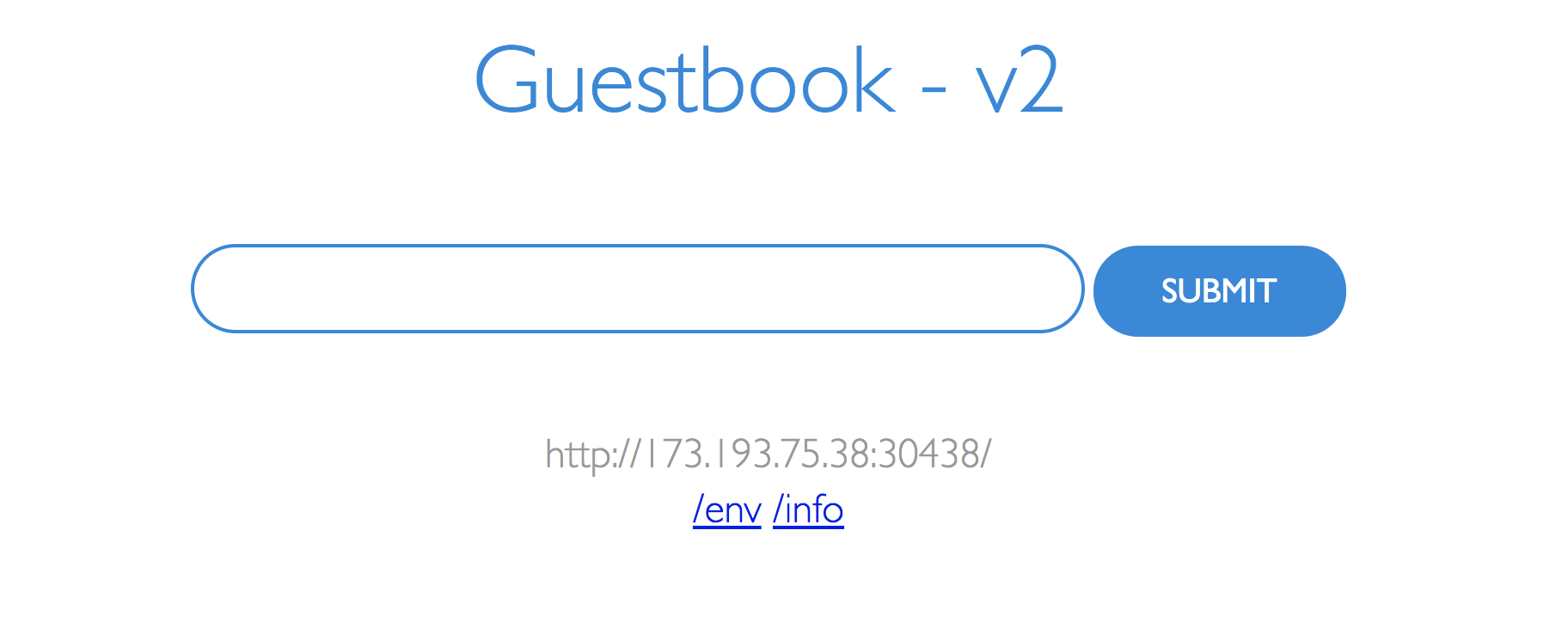
deployment "guestbook" successfully rolled out

The rollout might occur so quickly that the following messages might not display

1. Test the application as before, by accessing http://<Public IP>:<NodePort> in the browser to confirm your new code is active

Remember, to get the "Node Port", use *$ kubectl get service guestbook* and to get "Public IP", use *$ ibmcloud ks workers mycluster*

1. To verify that you're running "v2" of guestbook, look at the title of the page, it should now be “Guestbook - v2”. If the browser is still showing v1, close the tab / window, clear the cache and try again. Alternatively, open the application in another browser.
2. Output:



**Congratulations, you have deployed the second version of the app !**

1. If you want to undo your latest rollout, use:

*$ kubectl rollout undo deployment guestbook*

deployment "guestbook"

Use *kubectl rollout status deployment/guestbook* to see the status.

1. When doing a rollout, you see references to old replicas and new replicas. The old replicas are the original 10 pods deployed when we scaled the application. The new replicas come from the newly created pods with the different image. All of these pods are owned by the Deployment. The deployment manages these two sets of pods with a resource called a ReplicaSet. We can see the guestbook ReplicaSet with:

*$ kubectl get replicasets -l run=guestbook*

NAME DESIRED CURRENT READY AGE

guestbook-5f5548d4f 10 10 10 21m

guestbook-768cc55c78 0 0 0 3h