**Deployment of 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 that is used to deploy and test a Guestbook app (Written in Go) in IBM Cloud Kubernetes Service.

Before attending this Mini Code Lab, Please make sure the you’ve finished the following prerequisites.

you need to install the required CLIs to create and manage your Kubernetes clusters in IBM Cloud Container Service and to deploy containerized apps to your cluster.

* Account on [IBM Cloud](https://console.bluemix.net/registration/)
* As a prerequisite for the IBM Cloud Kubernetes Service plug-in, install the [**IBM Cloud CLI**](https://console.bluemix.net/docs/cli/reference/ibmcloud/download_cli.html#install_use)
* Login to your IBM Cloud Account and create a Kubernetes cluster (Using GUI) Steps Below.

1. Login to IBM Cloud with your credentials.
2. Go to Catalog
3. On Left hand side, Select container and click on Kubernetes Service
4. Click on Create
5. Under the cluster type. Select Free cluster and give a Unique name to your Cluster and select Create Cluster. (This will take 20 -25 mins to spin up your cluster, which will be used to deploy your application later in the lab below).

* Install the [kubernetes CLI](https://kubernetes.io/docs/tasks/tools/install-kubectl/)
* Install [Cloud container service plug-in](https://console.bluemix.net/docs/containers/cs_cli_install.html#cs_cli_install) , Run Step 3 in the terminal.

Below are the steps to login to IBM Cloud by using CLI/Terminal.

1. Log in to the IBM Cloud CLI: $bx login.
2. Enter your IBM Cloud credentials when prompted.

**Note:** If you have a federated ID, use $bx login --sso to log in to the IBM Cloud CLI. Enter your user name, and use the provided URL in your CLI output to retrieve your one-time passcode. You know you have a federated ID when the login fails without the --sso and succeeds with the --sso option.

**Set up and deploy your first application**

If you have not created the cluster with the above steps, Please use the below command to setup the cluster from CLI.

1. Provision a cluster:

$ bx cs cluster-create --name <name-of-cluster>

Once the cluster is provisioned, the kubernetes client CLI kubectl needs to be configured to talk to the provisioned cluster.

1. Run $ bx cs cluster-config <name-of-cluster>, and set the KUBECONFIG environment variable based on the output of the command. This will make your kubectl client point to your new Kubernetes cluster.

(If you're running in a Windows PowerShell environment, the *SET* and/or *EXPORT* equivalent is $env:KUBECONFIG="<value of KUBECONFIG filename>"; after setting, confirm the value is in the shell environment with ls env:KUBECONFIG)

Once your client is configured, you are ready to deploy your first application, guestbook.

# **Deploy your application**

In this part of the lab we 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 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:

$ kubectl get pods

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. Once the status reads Running, we need to expose that deployment as a service so we can access it through the IP of the worker nodes. The guestbook application listens on port 3000. Run:

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

service "guestbook" exposed

1. To find the port used on that 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> 3000:31208/TCP 1m

We can see that our <nodeport> is 31208. We can see in the output the port mapping from 3000 inside the pod exposed to the cluster on port 31208. This port in the 31000 range is automatically chosen, and could be different for you.

1. guestbook is now running on your cluster, and exposed to the internet. We need to find out where it is accessible. The worker nodes running in the container service get external IP addresses. Run $ bx cs workers <name-of-cluster>, and note the public IP listed on the <public-IP> line.

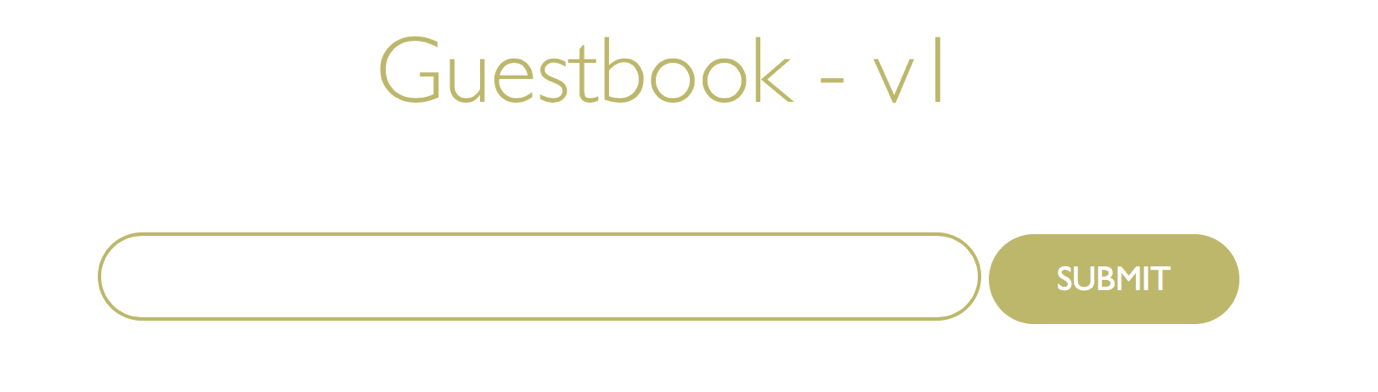


We can see that our <public-IP> is 173.193.99.136.

1. Now that you have both the address and the port, you can now access the application in the web browser at <public-IP>:<nodeport>. In the example case this is 173.193.99.136:31208.

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

**Output :**

****

If you want to stop here and remove the deployment do the following:

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

# **Optional :**

# **Scale and Update Deployments**

In this lab, 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.

For this lab, you need a running deployment of the guestbook application from the previous lab. If you deleted it, recreate it using:

$ kubectl run guestbook --image=ibmcom/guestbook:v1

# **1. Scale apps 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 make reality 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, you can run: kubectl rollout status deployment guestbook.

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

$ 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

1. Once the rollout has finished, ensure your pods are running by using: kubectl get pods.

You should see output listing 10 replicas of your deployment:

$ 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

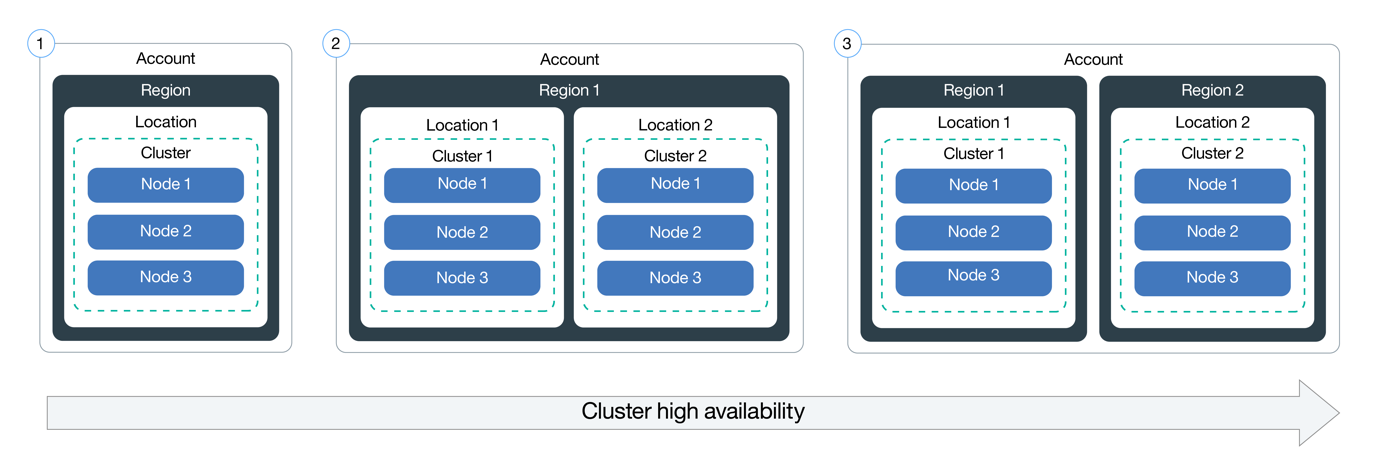
guestbook-562211614-vtfp2 1/1 Running 0 2m

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

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

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

**Tip:** Another way to improve availability is to [add clusters and regions](https://console.bluemix.net/docs/containers/cs_planning.html#cs_planning_cluster_config) to your deployment, as shown in the following diagram:

[](https://github.com/IBMDeveloperUK/kube101/blob/master/workshop/images/cluster_ha_roadmap.png)

# **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 lab, we used an image with a v1 tag. For our upgrade we'll 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. We 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. Multiple containers can be updated at the same time. ([More information](https://kubernetes.io/docs/user-guide/kubectl/kubectl_set_image/).)

1. Run kubectl rollout status deployment/guestbook to check the status of the rollout. The rollout might occur so quickly that the following messages might not display:

$ 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

1. Test the application as before, by accessing <public-IP>:<nodeport> in the browser to confirm your new code is active.

Remember, to get the "nodeport" and "public-ip" use:

$ kubectl describe service guestbook and $ bx cs workers <name-of-cluster>

To verify that you're running "v2" of guestbook, look at the title of the page, it should now be Guestbook - v2

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

$ kubectl rollout undo deployment guestbook

deployment "guestbook"

You can then 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 ReplicaSets with:

$ kubectl get replicasets -l run=guestbook

NAME DESIRED CURRENT READY AGE

guestbook-5f5548d4f 10 10 10 21m

guestbook-768cc55c78 0 0 0 3h

Before we continue, let's delete the application so we can learn about a different way to achieve the same results:

To remove the deployment, use kubectl delete deployment guestbook.

To remove the service, use kubectl delete service guestbook.

**Congratulations! You deployed the second version of the app.**

**Output for the v2 Application should be:**

