Assignment 2 - Kubernetes

Week 2 tasks and deliverables:

1. Setting up Kubernetes
2. Deploying Kubernetes pods
   - 2.1. *2a.jpg* Screenshot of kubectl commands and port forwarding
   - 2.2. *2b.jpg* Nginx home page access through localhost
3. Seal healing, updates, and rollback with Kubernetes deployment
   - 3.1. *3a.jpg* Screenshot showing deleted pod and seal healing of pods in a deployment
   - 3.2. *3b.jpg* Screenshot showing updates and revision history of a deployment
   - 3.3. *3c.jpg* Screenshot showing rollback of deployment
4. Scaling Kubernetes services
   - 4.1. *4a.jpg* Screenshot showing:
     - 4.1.1. deployment and service creation
     - 4.1.2. port forwarding
     - 4.1.3. Scaling deployment
   - 4.2. *4b.jpg* Nginx home page access through localhost

Introduction to Kubernetes

https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

Kubernetes makes it easy to deploy and operate applications in a microservice architecture. It does so by creating an abstraction layer on top of a group of hosts so that development teams can deploy their applications and let Kubernetes manage:

- Controlling resource consumption by application or team
- Evenly spreading application load across a hosting infrastructure
- Automatically load balancing requests across the different instances of an application
- Monitoring resource consumption and resource limits to automatically stop applications from consuming too many resources and restarting the applications again

- Moving an application instance from one host to another if there is a shortage of resources in a host, or if the host dies
- Automatically leveraging additional resources made available when a new host is added to the cluster
- Easily performing canary deployments and rollbacks

Important Kubernetes Terminologies:

1. Kubernetes Namespace
    a. Kubernetes Documentation: Namespaces
2. Pod
    a. Pods – Kubernetes Documentation
    b. What is a Pod.
3. Replica Set
    a. ReplicaSet
4. Deployment
    a. Deployments
5. Service
    a. Kubernetes Service
6. LoadBalancer Service
    a. Kubernetes LoadBalancer Service
7. NodePort Service (Not needed for this lab, but must know)
    a. Kubernetes NodePort Service
8. Ingress Controller (Not needed for this lab, but must know)
    a. Ingress
9. Horizontal Pod Autoscaler (Not needed for this lab, but must know)
    a. Kubernetes Horizontal Pod Autoscaler

Quick Points:

1. Ensure Docker is installed, up and running before using minikube.
2. Pods may take some time (a few minutes in a bad network) initially to start up, this is normal.
3. All resources mentioned in all the tasks must be created only in the **default Kubernetes namespace**, modifying other namespaces such as kube-system may cause Kubernetes to stop working.

4. If you are unable to show the results to your respective lab faculty, you can submit the screenshots to Edmodo in a zip format. Make sure all files in the zip follow the naming convention <srn>_2a.jpg or so on.

TASK-1 (Setting up Kubernetes)

The first task is to set up a Kubernetes cluster and a command-line tool called “kubectl” to interact with this cluster. For this lab, we will be using *minikube* (only on Linux environment) which is a VM/Docker-based tool that helps to create a Kubernetes cluster quickly to either test applications/learn Kubernetes.

Kubernetes being a container-orchestration service needs an engine to create/destroy containers and uses Docker for this purpose.

A real Kubernetes cluster runs pods natively on the host machine using the docker engine. So all container processes are run natively on the host machine. Some examples of popular Kubernetes clusters are AWS Elastic Kubernetes Service(EKS), Google Kubernetes Engine(GKE), and kubeadm which is used to create clusters manually.

Minikube is slightly different from the real Kubernetes cluster, it creates a virtual machine/docker container and runs the pods inside the VM/container. This is done so that we can get a quick Kubernetes cluster up and running without bothering about the details of setting up one or to avoid cloud services. Details about how this impacts deployments/services will be clearly explained as we go on in the lab.

To set up the Kubernetes cluster, follow the following steps:

For Windows

1. Install Docker/Ensure docker is already installed on the host machine. Make sure docker is up and running.

2. Under Docker Desktop -> Settings -> Kubernetes -> Enable the "Enable Kubernetes" checkbox

3. Click on Apply and Restart.
4. To test your kubectl is configured and correctly running, run: `kubectl get node`

   You should see a similar output:

```
shubh@DESKTOP-NSQ2KG6 MINGW64 ~
$ kubectl get node
NAME             STATUS   ROLES                  AGE   VERSION
docker-desktop   Ready    control-plane,master   92s   v1.22.5
```

For Linux (Minikube)

1. Install Docker/Ensure docker is already installed on the host machine. Make sure docker is up and running
2. Install Kubectl, the CLI tool used to interact with the Kubernetes cluster by following **Steps 1 to 4** under “**Install kubectl binary with curl on Linux**” in the following link:

https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/ **or**
https://kubernetes.io/docs/tasks/tools/install-kubectl/

3. Download and install minikube by running the following 2 commands (each is a complete line of command:

```
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
```

```
sudo install minikube-linux-amd64 /usr/local/bin/minikube
```

Ref Link - https://minikube.sigs.k8s.io/docs/start/#binary-download

4. Startup the minikube cluster by running: `minikube start`
   a. This might take a while initially as it needs to pull the docker image and set up kubectl
5. To test your kubectl is configured and correctly running, run: `kubectl get node`

   You should see a similar output:

```
shubham@shubham-VirtualBox:~$ kubectl get node
NAME       STATUS   ROLES                  AGE     VERSION
minikube   Ready    control-plane,master   5m38s   v1.23.1
```

All commands below are the same across all platforms: Windows / Linux / MacOS / WSL2

TASK-2 (Deploying Kubernetes pods)

All resources such as pods, deployments, services etc are created using YAML files. For this task and the following tasks, we will see how we can use Kubernetes to orchestrate Nginx web servers. Pods are the smallest deployable units of computing that you can create and manage in Kubernetes. Open the **pod.yaml** file and fill in the values based on the table below.

Ref links to understand resource limits used in the YAML file and their meaning :
https://jamesdefabia.github.io/docs/user-guide/compute-resources/ or
https://medium.com/@betz.mark/understanding-resource-limits-in-kubernetes-cpu-time-9eff74d3161b

Important:

YAML files are indentation sensitive. Do NOT change the indentation of the templates provided. All indentations are in increments of 2 spaces.

YAML files require all field values to be in lowercase, can include numbers

Key	Value
kind	Pod
metadata: label	nginx-<srn>
metadata: label: name	nginx-<srn>
spec: containers: name	nginx-<srn>
spec: containers: image	nginx:latest
spec: containers: resources: limits: memory	"128Mi"
spec: containers: resources: limits: cpu	"500m"
spec: ports: containerPort	80

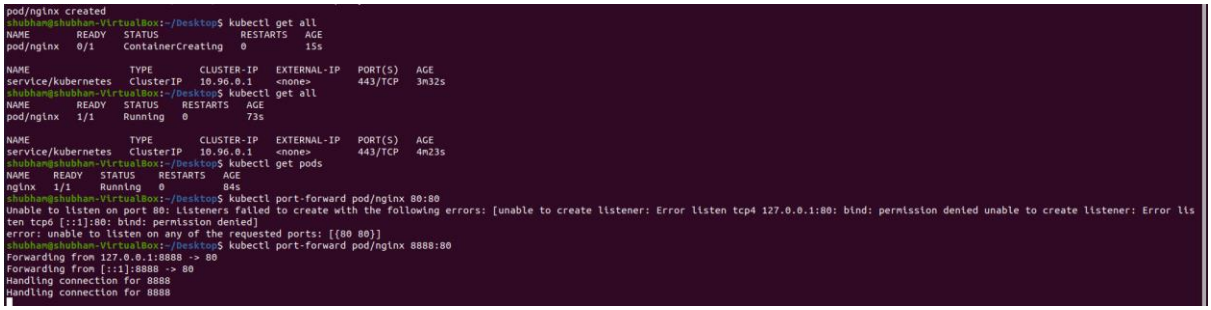
To create a pod from the yaml file, follow the following steps:

1. To run the pod, run:
   a. `kubectl create -f pod.yaml`
2. To find out whether the pod is running successfully, you can run:
   a. `kubectl get all` (This command fetches all the resources in the default namespace)
   b. `kubectl get pods`(This command shows only the pods in the default namespace)
3. The pod we are running is a web server, and we must be able to connect to the server.
   a. Since we have only a single pod, we want to just access THAT pod, and not a deployment (as we will see in the next task).
   b. Run. `kubectl port-forward pod/nginx-<srn> 80:80`
      i. What this command does is, say "I have a pod called Nginx, expose it such that if any HTTP request is coming for port 80 of the host machine, redirect it to me(Nginx pod), I (pod) will handle that request"

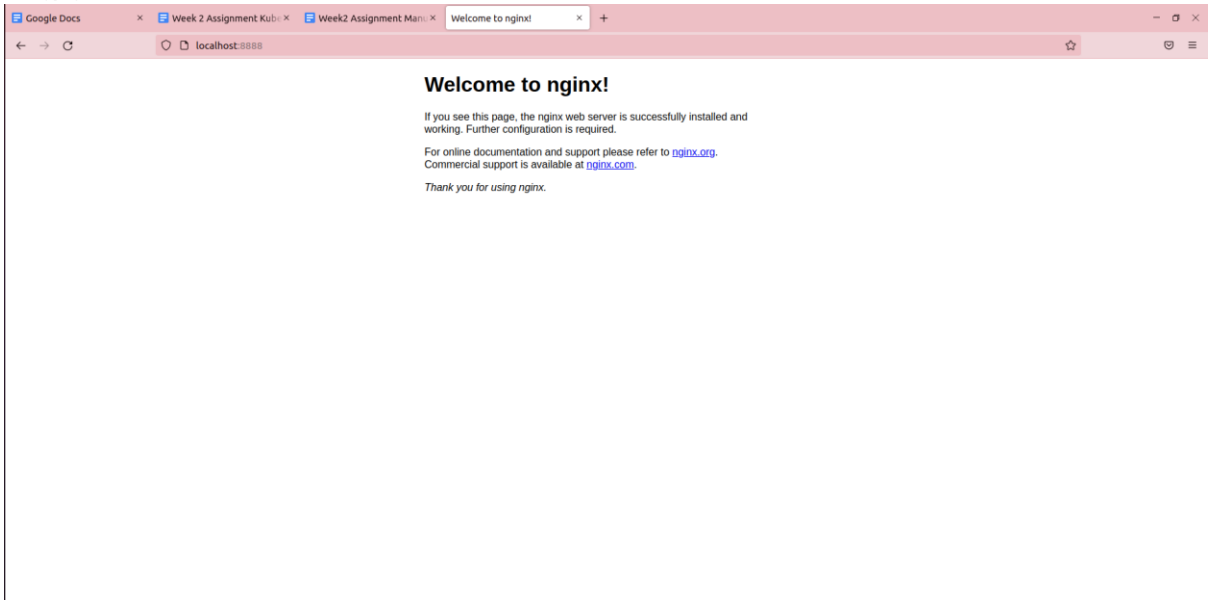
Note: You may see "Unable to Create Listener" errors if port 80 is already being used on the host system. To resolve this, you can use a different port. In the above command, the following is the format: <port_on_host>:<default_container_port>. The default container port here is the default Nginx port i.e 80. Therefore to expose another port say 8080, use 8080:80 . This is important to note every time you try to port-forward.

c. Access your webserver through http://localhost:80 (Or the port you changed to , if port 80 was not available)
d. Press Ctrl+C to stop the port-forwarding.

2a.jpg



2b.jpg



4. Delete the pods by either:
   a. Deleting every pod in the default namespace, by running:
      i. `kubectl delete pods --all`

   OR

   b. Deleting the specific pod, by :
      i. Finding the pod name, using:
         `kubectl get pods`
      ii. Deleting the specific pod by running
         `kubectl delete pod/<pod_name>`

TASK-3 (Seal healing, updates and rollback with Kubernetes deployment)

A pod is only a single instance(the smallest part) in a Kubernetes “deployment”. A Kubernetes deployment refers to a collection of pods called replica sets, and configuration parameters for the replica sets. Deployment is used to tell Kubernetes how to create or

modify instances of the pods that hold a containerized application. Deployments can scale the number of replica pods, enable the rollout of updated code in a controlled manner, or roll back to an earlier deployment version if necessary.

https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

To create a deployment on Kubernetes:

1. Download the **deploy.yaml** file which defines the deployment and complete the file using the table below. <srn> must be replaced by your respective srn.

Key	Value
kind	Deployment
metadata: name	mynginx-<srn>
spec: replicas	3
spec: template: metadata: labels: app	mynginx-<srn>
spec: template: spec: containers: name	mynginx-<srn>
spec: template: spec: containers: image	nginx:1.14.2
spec: template: spec: containers: resources: limits: memory	"128Mi"
spec: template: spec: containers: resources: limits: cpu	"500m"
spec: template: spec: containers: ports: containerPort	80

2. Just like a pod, run the deployment using the following command:

   ```
   kubectl create -f deploy.yaml
   ```

3. Get the deployment name using:

   ```
   kubectl get deployments
   ```

4. See the deployment rollout status:

   ```
   kubectl rollout status deployment/mynginx-<srn>
   ```

5. This deployment creates 3 pods of NGINX web server and a replica set to manage these pods. To view all the aspects of the deployment(pods, replica sets etc), run:

   ```
   kubectl get all
   ```

6. Note down the name of any one of the pods. This will be helpful in later steps.

7. Once you see all the pods up and running, your deployment is complete. The newly created replica set should show 3/3.

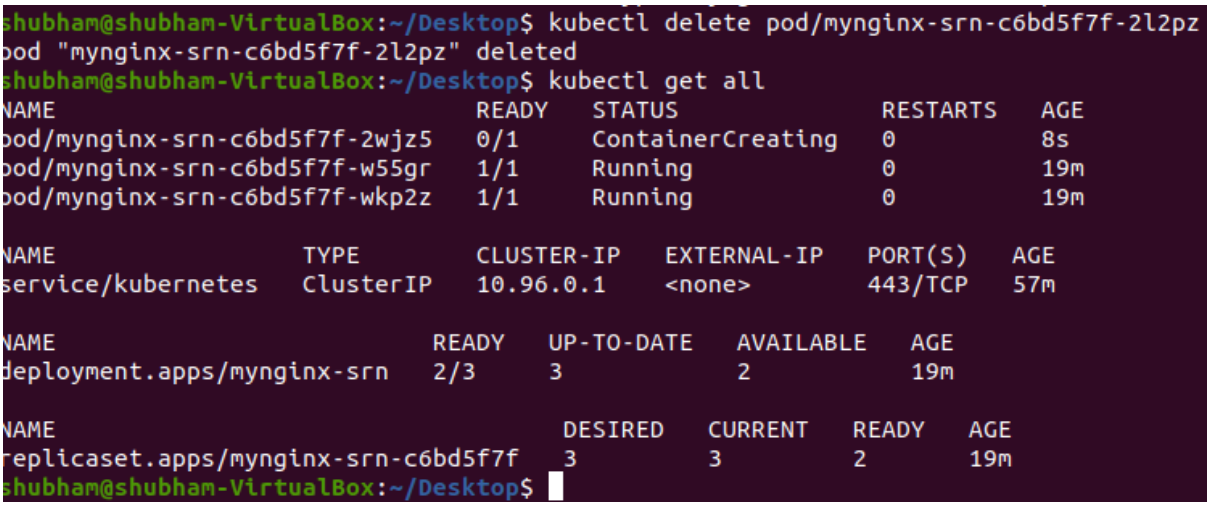
To observe Self Healing

8. Delete a pod using the name noted previously. The command is:

   ```
   kubectl delete pod/<pod-name>
   ```

9. The deleted pod will be replaced by another. Observe using the following command:
   `kubectl get all`

   3a.jpg



To update the deployment

10. We will be updating the image version used by the deployment. To do so, run the command:

    ```
    kubectl set image --record=true deployment/mynginx-<srn> mynginx-<srn>=nginx:1.16.1 --record=true
    ```

11. To see the changes made to the deployment:

    ```
    kubectl rollout history deployment/mynginx-<srn>
    ```

    3b.jpg Note: output for rollout deployment will vary based on when you execute the command

```
shubham@shubham-VirtualBox:~/Desktop$ kubectl set image deployment/mynginx-srn mynginx-srn=nginx:1.16.1 --record=true
Flag --record has been deprecated, --record will be removed in the future
deployment.apps/mynginx-srn image updated
shubham@shubham-VirtualBox:~/Desktop$ kubectl rollout status deployment/mynginx-srn
deployment "mynginx-srn" successfully rolled out
shubham@shubham-VirtualBox:~/Desktop$ kubectl rollout history deployment/mynginx-srn
deployment.apps/mynginx-srn
REVISION  CHANGE-CAUSE
1         <none>
2         kubectl set image deployment/mynginx-srn mynginx-srn=nginx:1.16.1 --record=true

shubham@shubham-VirtualBox:~/Desktop$
```

To rollback the deployment

12. To undo any changes made to the deployment. Run the following command:

    ```
    kubectl rollout undo deployment/mynginx-<srn>
    ```

    OR

    Note down the <revision-number> from the rollout history. Revert by specifying the <revision-number>

    ```
    kubectl rollout undo deployment/mynginx-<srn> --to-revision=<revision-number>
    ```

13. [OPTIONAL] Observe the rollout using:

    ```
     kubectl rollout status deployment/mynginx-<srn>
    ```

    3c.jpg Note: output for rollout deployment will vary based on when you execute the command

```
shubham@shubham-VirtualBox:~/Desktop$ kubectl rollout undo deployment/mynginx-srn
deployment.apps/mynginx-srn rolled back
shubham@shubham-VirtualBox:~/Desktop$ kubectl rollout status deployment/mynginx-srn
Waiting for deployment "mynginx-srn" rollout to finish: 1 out of 3 new replicas have been updated...
Waiting for deployment "mynginx-srn" rollout to finish: 1 out of 3 new replicas have been updated...
Waiting for deployment "mynginx-srn" rollout to finish: 2 out of 3 new replicas have been updated...
Waiting for deployment "mynginx-srn" rollout to finish: 2 out of 3 new replicas have been updated...
Waiting for deployment "mynginx-srn" rollout to finish: 2 out of 3 new replicas have been updated...
Waiting for deployment "mynginx-srn" rollout to finish: 1 old replicas are pending termination...
Waiting for deployment "mynginx-srn" rollout to finish: 1 old replicas are pending termination...
deployment "mynginx-srn" successfully rolled out
shubham@shubham-VirtualBox:~/Desktop$
```

14. Delete the deployment using:

    ```
    kubectl delete deploy mynginx-<srn>
    ```

TASK-4 (Scaling Kubernetes services)

A Service in Kubernetes is an abstraction that defines a logical set of Pods and a policy by which to access them. What this means is, it defines an abstract layer on top of a logical set of pods, essentially a “micro-service”-layer. A microservice can be complex and also be horizontally scaled, as a user/caller of the micro-service, I am not bothered by which specific pod services my request but I only care of the “service”, this is essentially what kubernetes service does. There are many types of Kubernetes services such as LoadBalancer, NodePort, Ingress etc.

Replica sets are what control the pods in a deployment, they are what allow for creating rolling updates, config changes etc but more importantly scaling. Replica sets scale the pods in the deployment without affecting the micro-service availability.

1. Modify the **deploy_service.yaml** file according to the following tables:

   For the Deployment

Key	Value
kind	Deployment
metadata: name	mynginx-srn
spec: replicas	2
spec: selector: matchLabels: app	mynginx-<srn>
spec: template: metadata: labels: app	mynginx-<srn>
spec: template: spec: containers: name	mynginx-<srn>
spec: template: spec: containers: image	nginx:latest
spec: template: spec: containers: resources: limits: memory	"128Mi"
spec: template: spec: containers: resources: limits: cpu	"500m"
spec: template: spec: containers: ports: containerPort	80

   For the Service

Key	Value
kind	Deployment

metadata: name	nginx-<srn>
spec: selector: app	mynginx-<srn>
spec: selector: ports: protocol	TCP
spec: selector: ports: port	80
spec: selector: ports: targetPort	80
spec: selector: type	LoadBalancer

2. Create the deployment **and its service** by running:

   ```
   kubectl create -f deploy_service.yaml
   ```

3. Once the pods are up and running, expose the service using:

   ```
   kubectl port-forward service/nginx-<srn> 80:80
   ```

Note: You may see "Unable to Create Listener" errors if port 80 is already being used on the host system. To resolve this, you can use a different port. In the above command, the following is the format: <port_on_host>:<default_container_port>. The default container port here is the default Nginx port i.e 80. Therefore to expose another port say 8080, use 8080:80 . This is important to note every time you try to port-forward.

4. To scale the deployment done before, run the following command:

   ```
   kubectl scale deploy mynginx-<srn> --replicas=10
   ```

5. The above command scales the pods to 10 pods. (You may see pods in a pending state, this is because we are using minikube which has resource restrictions, on a real cluster having sufficient resources we would see the scaling successfully done)

   4a.jpg

```
shubham@shubham-VirtualBox:~/Desktop$ kubectl create -f deploy_service.yaml
deployment.apps/mynginx-srn created
service/nginx-srn created
shubham@shubham-VirtualBox:~/Desktop$ kubectl port-forward service/nginx-<srn> 8050:80
bash: srn: No such file or directory
shubham@shubham-VirtualBox:~/Desktop$ kubectl port-forward service/nginx-srn 8050:80
Forwarding from 127.0.0.1:8050 -> 80
Forwarding from [::1]:8050 -> 80
Handling connection for 8050
^Cshubham@shubham-VirtualBox:~/Desktop$
shubham@shubham-VirtualBox:~/Desktop$ kubectl scale deploy mynginx-srn --replicas=10
deployment.apps/mynginx-srn scaled
shubham@shubham-VirtualBox:~/Desktop$ kubectl get all
NAME                                READY   STATUS              RESTARTS   AGE
pod/mynginx-srn-7cb84b96fb-4fqfn    0/1     ContainerCreating   0          7s
pod/mynginx-srn-7cb84b96fb-6zghp    0/1     Pending             0          6s
pod/mynginx-srn-7cb84b96fb-9mdfs    1/1     Running             0          2m15s
pod/mynginx-srn-7cb84b96fb-cx75q    0/1     Pending             0          7s
pod/mynginx-srn-7cb84b96fb-n2mkl    0/1     ContainerCreating   0          7s
pod/mynginx-srn-7cb84b96fb-sphsd    0/1     Pending             0          6s
pod/mynginx-srn-7cb84b96fb-sqnpd    0/1     Pending             0          6s
pod/mynginx-srn-7cb84b96fb-sv2p2    0/1     Pending             0          6s
pod/mynginx-srn-7cb84b96fb-tvwd6    1/1     Running             0          2m15s
pod/mynginx-srn-7cb84b96fb-v8gmj    0/1     Pending             0          7s

NAME                 TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
service/kubernetes   ClusterIP      10.96.0.1        <none>        443/TCP        159m
service/nginx-srn    LoadBalancer   10.105.216.232   <pending>     80:31197/TCP   2m15s

NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/mynginx-srn    2/10    10           2           2m15s

NAME                                      DESIRED   CURRENT   READY   AGE
replicaset.apps/mynginx-srn-7cb84b96fb    10        10        2       2m15s
shubham@shubham-VirtualBox:~/Desktop$
```

4b.jpg

6. Delete the deployment and the service:
    a. `kubectl delete service nginx-<srn>`
    b. `kubectl delete deploy mynginx-<srn>`

CLEANUP - (Shutdown minikube)

This task ensures all resources are stopped:

1. Ensure all resources are stopped and not seen when running the Kubernetes commands.
2. Stop minikube by running:

```
minikube stop
```