Lab 8 - Kubernetes

Week 8 Tasks:

1. Create a Kubernetes cluster using minikube
    a. 1a.png Screenshot of kubectl get nodes showing minikube is running properly
2. Kubernetes pods
    a. 2a.png Screenshot showing:
        i. nginx pod in running status
        ii. port-forwarding of the pod using port 80
    b. 2b.png Nginx home page access through localhost
3. Kubernetes Deployments and Kubernetes services
    a. 3a.png Screenshot showing deployment running and replicas as 2/2. Also, show the service
    b. 3b.png port-forwarding of the service using port 80 and accessing the web page using localhost
4. Scaling Kubernetes
    a. 4a.png showing 10 pods (either running/pending state)

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 but only in PDF or WORD format (and not in ZIP format)

TASK-1 (Create a Kubernetes cluster using minikube)

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* 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 minikube Kubernetes cluster, follow the following steps:

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. Test your kubectl is configured and correctly running, by running: `kubectl get node`

You should see a similar output:

```
NAME       STATUS   ROLES    AGE   VERSION
minikube   Ready    master   65m   v1.18.3
```

TASK-2 (Create a Kubernetes pod)

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 examine the contents of the YAML and what each property/field does.

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 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"
        ii. 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.
4. Delete the pods by either:
    a. Deleting every pod in the default namespace, by running:
        i. `kubectl delete pods --all`
    b. Deleting the specific pod, by :
        i. Finding the pod name, using:
            1. `kubectl get pods`
        ii. Deleting the specific pod by running
            1. `kubectl delete pod/<pod_name>`

TASK-3 (Create a Kubernetes Deployment and create a Load Balancing Service)

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.

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.

This task involves creating a deployment and then a service for it.

1. Download the deploy.yaml file which defines the deployment. Go through the deployment YAML file to understand what each field is and how it affects the deployment. **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
2. Just like a pod, run the deployment using the following command:

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

3. This deployment creates 2 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
   ```

4. Once you see all the pods up and running, your deployment is complete. The newly created replica set should show 2/2.
5. To create a service for this deployment, we can either use:
   a. `kubectl expose deploy mynginx --port=80 --target-port=80`
   b. Or use a YAML specification which **we will do now**.
6. Delete the previous deployment and service by running:

   ```
   kubectl  delete deploy mynginx
   ```

   ```
   kubectl delete service mynginx
   ```

7. Download the deploy_service.yaml file.
8. Create the deployment **and its service** by running:

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

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

```
kubectl port-forward service/nginx 80:80
```

10. Access the webserver on http://localhost:80
11. Keep the deployment and service running for the next task

TASK-4 (Scaling deployments)

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

```
kubectl scale deploy mynginx --replicas=10
```

2. 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)
3. Delete the deployment and the service:
    a. `kubectl delete service nginx`
    b. `kubectl delete deploy mynginx`

TASK-5 (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
```