**Kubernetes Assignment Part-1**

**Deploy a Kubernetes Cluster for 3 nodes**

<https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/>

**Steps:**

**Step 1:** Install Docker on all nodes:

sudo apt update

sudo apt install -y docker.io

sudo systemctl enable --now docker

**Step 2:** Install kubeadm, kubelet, and kubectl on all nodes:

* sudo apt update
* sudo apt install -y apt-transport-https ca-certificates 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 /etc/apt/sources.list.d/kubernetes.list

* sudo apt update
* sudo apt install -y kubelet kubeadm kubectl
* sudo systemctl enable --now kubelet

**Step 3:** Initialize the master node:

* sudo kubeadm init

Note the kubeadm join command at the end of the output. You will use this command to join the worker nodes to the cluster.

**Step 4:** Set up the kubeconfig for the current user on the master node:

* mkdir -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

sudo chown $(id -u):$(id -g) $HOME/.kube/config

**Step 5:** Master Node SG open port

6443 - API Server

10250 - Kubelet

TCP 6783 and UDP 6783/6784 – CNI

**Step 6:** Worker Node SG open port

10250 - kubelet

TCP 6783 and UDP 6783/6784 -CNI

**Step 7:** Join the worker nodes (Node 2 and Node 3) to the cluster:

Run the **\*kubeadm join\*** command obtained from Step 3 on both worker nodes.

* sudo kubeadm join 192.168.1.100:6443 --token abcdef.1234567890abcdef --discovery-token-ca-cert-hash sha256:1234567890abcdef1234567890abcdef1234567890abcdef1234567890abcdef

**Step 8:** Verify the cluster status on the master node:

* kubectl get nodes

**NAME        STATUS     ROLES                  AGE    VERSION**

**master      NotReady   control-plane,master   33m    v1.20.0**

**worker-01   NotReady   <none>                 2m9s   v1.20.0**

**worker-02   NotReady   <none>                 114s   v1.20.0**

**Step 9:** Install a Pod network (We'll use Calico in this example):

* kubectl apply -f https://docs.projectcalico.org/v3.15/manifests/calico.yaml

You should see all nodes in the "Ready" state.

**NAME        STATUS     ROLES                  AGE    VERSION**

**master      Ready   control-plane,master   33m    v1.20.0**

**worker-01   Ready   <none>                 2m9s   v1.20.0**

**worker-02   Ready   <none>                 114s   v1.20.0**

**Create a nginx deployment of 3 replicas**

**Step 1:** To create an Nginx deployment with three replicas in a Kubernetes cluster, you can use a YAML manifest file to describe the deployment.

apiVersion: apps/v1

kind: Deployment

metadata:

  name: nginx-deployment

spec:

  replicas: 3

  selector:

    matchLabels:

      app: nginx

  template:

    metadata:

      labels:

        app: nginx

    spec:

      containers:

      - name: nginx

        image: nginx:latest

        ports:

        - containerPort: 80

**Step 2:** Save the above content in a file

for example, **nginx-deployment.yaml**, and then apply it to your Kubernetes cluster using the following command:

* kubectl apply -f nginx-deployment.yaml

This YAML file defines a deployment named nginx-deployment with three replicas. It uses the official Nginx Docker image, exposes port 80, and labels the pods with app: nginx. Adjust the image field to use a specific version or variant of the Nginx image if needed.

After applying the deployment, you can check the status using:

* kubectl get deployments
* kubectl get pods

This will show you the deployment status and the status of the pods. The pods should eventually reach the "Running" state.

**Step 2:** To expose the Nginx deployment externally

you can create a service. For example, to expose it on NodePort, you can create a **Service**:

apiVersion: v1

kind: Service

metadata:

  name: nginx-service

spec:

  selector:

    app: nginx

  type: NodePort

  ports:

    - protocol: TCP

      port: 80

      targetPort: 80

Save this in a file, for example, **nginx-service.yaml**, and apply it:

* kubectl apply -f nginx-service.yaml

Now, you can access Nginx using the Node's IP and the NodePort assigned by Kubernetes.

**Module-9: Kubernetes Assignment – 2**

Create a service of type NodePort for nginx deployment

**Step 1:** Create a Deployment for Nginx:

# nginx-deployment.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

  name: nginx-deployment

spec:

  replicas: 3

  selector:

    matchLabels:

      app: nginx

  template:

    metadata:

      labels:

        app: nginx

    spec:

      containers:

      - name: nginx

        image: nginx:latest

        ports:

        - containerPort: 80

Save this YAML configuration in a file, for example, **nginx-deployment.yaml**, and apply it using the following command:

* kubectl apply -f nginx-deployment.yaml

**Step 2:** Create a Service for Nginx:

# nginx-service.yaml

apiVersion: v1

kind: Service

metadata:

  name: nginx-service

spec:

  selector:

    app: nginx

  ports:

    - protocol: TCP

      port: 80

      targetPort: 80

  type: NodePort

Save this YAML configuration in a file, for example, **nginx-service.yaml**, and apply it using the following command:

* kubectl apply -f nginx-service.yaml

This service will expose the Nginx deployment on a specific port on all nodes in your cluster. The port will be in the range 30000-32767 by default. You can access the service on any node's IP address at the chosen NodePort.

Make sure that your cluster has the necessary firewall rules or security groups configured to allow traffic on the chosen NodePort.

**Step 3:** Check the nodeport service on a browser to verify

Remember that exposing services using NodePort may not be suitable for production environments. In production, you might want to consider using a LoadBalancer service type or an Ingress controller depending on your infrastructure and requirements.

**Module-9: Kubernetes Assignment - 3**

**Use the previous deployment**

**Change the replicas to 5 for the deployment**

**Step 1:** change the replicas

Certainly! To change the replicas to 5 for the Nginx deployment, you can update the **replicas** field in the **nginx-deployment.yaml** file. Here's the modified file:

# nginx-deployment.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

  name: nginx-deployment

spec:

  replicas: 5  # Updated replicas to 5

  selector:

    matchLabels:

      app: nginx

  template:

    metadata:

      labels:

        app: nginx

    spec:

      containers:

      - name: nginx

        image: nginx:latest

        ports:

        - containerPort: 80

Save the changes to the file and apply the updated deployment using the following command:

kubectl apply -f nginx-deployment.yaml

**Module-9: Kubernetes Assignment – 4**

Use the previous deployment

**Step 1:** Deploy an nginx deployment of 3 replicas

deploy an Nginx deployment with 3 replicas, you can use the following YAML configuration for the deployment:

# nginx-deployment.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

  name: nginx-deployment

spec:

  replicas: 3

  selector:

    matchLabels:

      app: nginx

  template:

    metadata:

      labels:

        app: nginx

    spec:

      containers:

      - name: nginx

        image: nginx:latest

        ports:

        - containerPort: 80

Save this configuration in a file, for example, **nginx-deployment.yaml**, and apply it using the following command:

* kubectl apply -f nginx-deployment.yaml

This will deploy an Nginx deployment with 3 replicas. You can verify the deployment by checking the status with the following command:

* kubectl get deployment nginx-deployment

Make sure to replace nginx-deployment with the actual name you provided in the YAML file if it's different

Create an nginx service of type clusterip

**Step 2:** To create an Nginx service of type **ClusterIP** in Kubernetes

you can use the following YAML configuration:

# nginx-service.yaml

apiVersion: v1

kind: Service

metadata:

  name: nginx-service

spec:

  selector:

    app: nginx

  ports:

    - protocol: TCP

      port: 80

      targetPort: 80

  type: ClusterIP

Save this configuration in a file, for example, **nginx-service.yaml**, and apply it using the following command:

* kubectl apply -f nginx-service.yaml

This will create a ClusterIP service named **nginx-service** that is associated with the pods labeled with **app: nginx**. The service will be assigned a cluster-internal IP address.

You can use this service to access the Nginx deployment within the Kubernetes cluster using the service name (nginx-service). Note that this type of service is not externally accessible from outside the cluster. If you need external access, you might want to consider using NodePort or LoadBalancer service types depending on your requirements and environment.

**Create an ingress service /apache to apache service /nginx to nginx service**

To create Ingress resources that route traffic to different services based on the path, you can define Ingress rules for each path. Below are examples for creating Ingress resources for paths **/apache** and **/nginx** that route traffic to Apache and Nginx services, respectively.

**Step 1:** Create Apache Deployment and ClusterIP Service:

# apache-deployment.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

  name: apache-deployment

spec:

  replicas: 3

  selector:

    matchLabels:

      app: apache

  template:

    metadata:

      labels:

        app: apache

    spec:

      containers:

      - name: apache

        image: httpd:latest

        ports:

        - containerPort: 80

---

# apache-service.yaml

apiVersion: v1

kind: Service

metadata:

  name: apache-service

spec:

  selector:

    app: apache

  ports:

    - protocol: TCP

      port: 80

      targetPort: 80

  type: ClusterIP

Apply these using:

* kubectl apply -f apache-deployment.yaml
* kubectl apply -f apache-service.yaml

**Step 2:** Create Nginx Deployment and ClusterIP Service:

# nginx-deployment.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

  name: nginx-deployment

spec:

  replicas: 3

  selector:

    matchLabels:

      app: nginx

  template:

    metadata:

      labels:

        app: nginx

    spec:

      containers:

      - name: nginx

        image: nginx:latest

        ports:

        - containerPort: 80

---

# nginx-service.yaml

apiVersion: v1

kind: Service

metadata:

  name: nginx-service

spec:

  selector:

    app: nginx

  ports:

    - protocol: TCP

      port: 80

      targetPort: 80

  type: ClusterIP

**Apply these using:**

* kubectl apply -f nginx-deployment.yaml
* kubectl apply -f nginx-service.yaml

**Step 3:** Create Ingress Resource:

Make sure to replace your-ingress-host with the actual host you want to use for your Ingress.

# ingress.yaml

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

  name: my-ingress

spec:

  rules:

  - host: your-ingress-host  # Update this to your Ingress host

    http:

      paths:

      - path: /apache

        pathType: Prefix

        backend:

          service:

            name: apache-service

            port:

              number: 80

      - path: /nginx

        pathType: Prefix

        backend:

          service:

            name: nginx-service

            port:

              number: 80

Apply the Ingress resource using:

* kubectl apply -f ingress.yaml

Now, when you access your-ingress-host/apache, it will route traffic to the Apache service, and when you access your-ingress-host/nginx, it will route traffic to the Nginx service. Ensure that you have a proper Ingress controller set up in your cluster to handle these Ingress resources.

**Kubernetes – Assignment**

**Problem Statement:**

You work for the xyz organization. Your organization uses Kubernetes for container orchestration. Your organization has recently created pods from which data was being lost. Now they require volume mounts which preserve data and save a password called “xyzIsthebest” and this has to be put on a particular node of your choice.

**You have been asked to:**

1. Create a persistent volume

2. create a persistent volume claim

3. create a secret password “xyzIsthebest”

4. Taint one of the nodes of the cluster

**Step 1:** Create a persistent volume

Define a Persistent Volume that represents the storage resource. Create a YAML file, for example, **pv.yaml**:

apiVersion: v1

kind: PersistentVolume

metadata:

  name: my-pv

spec:

  capacity:

    storage: 1Gi

  accessModes:

    - ReadWriteOnce

  hostPath:

    path: /path/on/your/node

Apply the PV configuration to your cluster:

* kubectl apply -f pv.yaml

**Step 2:** Create a persistent volume claim

Define a Persistent Volume Claim that requests storage from the PV. Create a YAML file, for example, **pvc.yaml**

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

  name: my-pvc

spec:

  accessModes:

    - ReadWriteOnce

  resources:

    requests:

      storage: 1Gi

Apply the PVC configuration to your cluster:

$ kubectl apply -f pvc.yaml

**Step 3:** create a secret password “xyzIsthebest”

Create a Kubernetes Secret to store the password. Create a YAML file, for example, **secret.yaml**:

apiVersion: v1

kind: Secret

metadata:

  name: my-secret

type: Opaque

data:

  password: xyzlsthbest  # Base64 encoded password

Apply the Secret configuration to your cluster:

* kubectl apply -f secret.yaml

You can use the following command to base64 encode your password:

* echo -n 'xyzIsthebest' | base64

**Step 4:** Taint one of the nodes of the cluster

Mount PVC and Secret in Pod:

Update your Pod configuration to mount the PVC for data and the Secret for the password. Here's a simplified example:

apiVersion: v1

kind: Pod

metadata:

  name: my-pod

spec:

  containers:

    - name: my-container

      image: your-image

      volumeMounts:

        - name: my-data

          mountPath: /path/in/container

        - name: my-secret

          mountPath: /path/to/password

          readOnly: true

  volumes:

    - name: my-data

      persistentVolumeClaim:

        claimName: my-pvc

    - name: my-secret

      secret:

        secretName: my-secret

Save and Apply the updated Pod configuration to your cluster:

* kubectl apply -f pod.yaml

Replace placeholders like /path/on/your/node, /path/in/container, and your-image with your actual paths and container image.

This setup ensures that your data is persisted using a Persistent Volume and the password is securely stored using a Kubernetes Secret.