**Lab Guide: Docker Installation and Basic Usage with Node.js Application Deployment**

This lab guide will walk you through installing Docker on Ubuntu, creating a Docker image using a Dockerfile, and running a containerized Node.js application.

**Prerequisites:**

* A machine with Ubuntu installed.
* SSH access to the Ubuntu machine.

**Step 1: Set Up the Environment**

1. **Access the Machine**  
   Use SSH to log into the Ubuntu server:

bash

Copy code

ssh username@your-server-ip

sudo -i

1. **Update the System**  
   Ensure your system is updated:

bash

Copy code

apt update && apt upgrade -y

1. **Install Necessary Tools**  
   Install tree for directory visualization:

bash

Copy code

apt install tree -y

**Step 2: Install Docker**

1. **Follow Official Docker Installation Steps**  
   Use the Docker documentation for a clean installation.  
   Link: [Install Docker on Ubuntu](https://docs.docker.com/engine/install/ubuntu/#install-using-the-repository)

Example commands:

bash

Copy code

apt update

apt install -y apt-transport-https ca-certificates curl software-properties-common

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

apt update

apt install -y docker-ce docker-ce-cli containerd.io

1. **Verify Docker Installation**  
   Check Docker version:

bash

Copy code

docker -v

**Step 3: Clone the Node.js Application Repository**

1. **Clone the Repository**  
   Clone the repository that contains the Node.js application and Dockerfile:

bash

Copy code

git clone https://github.com/ramannkhanna2/nodejsapp\_jenkins-docker-kubernetes.git

1. **Navigate to the Repository**  
   Change to the application directory:

bash

Copy code

cd nodejsapp\_jenkins-docker-kubernetes

1. **Remove Unnecessary Files**  
   Clean up irrelevant files:

bash

Copy code

rm -rf Jenkinsfile Jenkinsfile2 deploy.tf openshiftdeploymentfile.yaml Readme.md

1. **Visualize the Directory Structure**  
   Use tree to confirm the file structure:

bash

Copy code

tree

**Step 4: Create and Build a Docker Image**

1. **Inspect the Dockerfile**  
   View the Dockerfile to understand the application build process:

bash

Copy code

cat Dockerfile

1. **Build the Docker Image**  
   Build the image using the Dockerfile:

bash

Copy code

docker build -t socgenimage .

1. **Verify the Image**  
   Check that the image has been created:

bash

Copy code

docker images

**Step 5: Run and Manage the Container**

1. **Run a Container**  
   Start a container using the image:

bash

Copy code

docker run -dt --name c1 socgenimage

1. **Verify the Running Container**  
   Check the running containers:

bash

Copy code

docker ps

1. **Access the Container**  
   Execute an interactive shell inside the container:

bash

Copy code

docker exec -it c1 /bin/bash

**Step 6: Explore Docker Commands**

1. **View Image History**  
   Inspect the layers of the Docker image:

bash

Copy code

docker image history socgenimage

1. **Check Disk Space**  
   Verify disk usage on the system:

bash

Copy code

df -h

1. **Stop and Remove the Container (Optional)**  
   Stop and delete the container if needed:

bash

Copy code

docker stop c1

docker rm c1

1. **Remove the Docker Image (Optional)**  
   Delete the image to free up space:

bash

Copy code

docker rmi socgenimage

**Lab Completion**

You have successfully installed Docker, built a Docker image, and deployed a Node.js application in a container. This guide provided hands-on experience with basic Docker commands and containerization concepts.

**Bonus: Push Docker Image to Azure Container Registry**

* Set up Azure CLI and authenticate.
* Create an Azure Container Registry.
* Tag the Docker image and push it to ACR.

**Lab Guide: Pushing a Custom Docker Image to Azure Container Registry and Running a Container**

This guide explains how to push a custom Docker image to Azure Container Registry (ACR), deploy it, and test its functionality.

**Prerequisites**

* An existing custom Docker image built on your local system (socgenimage in this case).
* Azure Container Registry (e.g., ramanimagereg.azurecr.io) is already created.
* Docker and Azure CLI installed.

**Step 1: Verify Docker Image**

1. **List Local Docker Images**  
   Ensure your custom image exists:

bash

Copy code

docker images

**Step 2: Authenticate with Azure Container Registry**

1. **Log In to Docker**  
   Authenticate with Docker CLI:

bash

Copy code

docker login

1. **Log In to Azure Container Registry**  
   Authenticate using the registry URL:

bash

Copy code

docker login ramanimagereg.azurecr.io

Provide your Azure credentials when prompted.

**Step 3: Tag and Push the Docker Image**

1. **Tag the Image**  
   Tag the custom Docker image with the ACR URL and version:

bash

Copy code

docker tag socgenimage:latest ramanimagereg.azurecr.io/socgenimage:version1

1. **Push the Image to ACR**  
   Push the tagged image to ACR:

bash

Copy code

docker push ramanimagereg.azurecr.io/socgenimage:version1

1. **Verify the Push**  
   Confirm the image is successfully pushed by checking ACR through the Azure Portal or CLI:

bash

Copy code

az acr repository list --name ramanimagereg --output table

**Step 4: Clean Up Local Docker Environment (Optional)**

1. **Remove Local Containers and Images**  
   Stop and remove the running container:

bash

Copy code

docker rm -f c1

Remove all local Docker images:

bash

Copy code

docker rmi -f $(docker images -q)

1. **Verify the Clean-Up**  
   Ensure no images or containers are running:

bash

Copy code

docker images

docker ps

**Step 5: Deploy the Image from ACR**

1. **Run the Container**  
   Use the pushed image to create a container:

bash

Copy code

docker run -dt --name socgencont -P ramanimagereg.azurecr.io/socgenimage:version1

1. **Verify Running Container**  
   List running containers:

bash

Copy code

docker ps

**Step 6: Test the Application**

1. **Check Mapped Ports**  
   Identify the exposed port from the docker ps output (e.g., 32768).
2. **Test Using Curl**  
   Use curl to test the application on the exposed port:

bash

Copy code

curl localhost:<mapped-port>

Example:

bash

Copy code

curl localhost:32768

**Step 7: Access the Container (Optional)**

1. **Execute a Bash Shell in the Container**  
   Access the running container for debugging or additional configurations:

bash

Copy code

docker exec -it socgencont /bin/bash

**Lab Completion**

You have successfully pushed a custom Docker image to Azure Container Registry, deployed it as a container, and verified its functionality. This guide provided experience with Docker image management, ACR usage, and container testing.

**Lab Guide: Deploying a Node.js Application on Kubernetes with a LoadBalancer Service**

This lab guide outlines the process of deploying a Node.js application on a Kubernetes cluster using a deployment manifest (deploy.yml) and exposing it via a LoadBalancer service.

**Prerequisites**

* A Kubernetes cluster is already set up and accessible (e.g., AKS, Minikube, EKS).
* kubectl is installed and configured to interact with the cluster.
* Docker image (ramanimagereg.azurecr.io/socgenimage:version1) is already available in Azure Container Registry.

**Step 1: Verify Kubernetes Cluster**

1. **List Pods in the Current Namespace**

bash

Copy code

kubectl get pods

1. **List Pods in All Namespaces**

bash

Copy code

kubectl get pods -A

1. **Check Node Status**

bash

Copy code

kubectl get nodes

1. **Confirm Kubernetes API Resources**

bash

Copy code

kubectl api-resources

**Step 2: Create the Deployment and Service Manifest**

1. **Write the Deployment and Service YAML** Open your favorite editor (e.g., vim or nano) and create a file called deploy.yml:

bash

Copy code

vi deploy.yml

Add the following YAML configuration:

yaml

Copy code

apiVersion: apps/v1

kind: Deployment

metadata:

name: nodeapp-deployment

labels:

app: nodeapp

spec:

replicas: 5

selector:

matchLabels:

app: nodeapp

template:

metadata:

labels:

app: nodeapp

spec:

containers:

- name: nodeserver

image: ramanimagereg.azurecr.io/socgenimage:version1

ports:

- containerPort: 3000

---

apiVersion: v1

kind: Service

metadata:

name: nodeapp-service

spec:

selector:

app: nodeapp

type: LoadBalancer

ports:

- protocol: TCP

port: 3000

targetPort: 3000

1. **Save the File** Exit the editor after saving (:wq in vim).

**Step 3: Apply the Deployment and Service Configuration**

1. **Create Resources Using the Manifest** Apply the deploy.yml configuration file:

bash

Copy code

kubectl create -f deploy.yml

1. **Verify Deployment and Service** List all resources created:

bash

Copy code

kubectl get all

**Step 4: Check the Deployment and Pods**

1. **List Pods**

bash

Copy code

kubectl get pods

1. **Detailed Pod Information** View details of pods with node assignment and IPs:

bash

Copy code

kubectl get pods -o wide

**Step 5: Access the Application**

1. **Retrieve LoadBalancer IP** Once the LoadBalancer service is provisioned, check the external IP:

bash

Copy code

kubectl get service nodeapp-service

Example output:

scss

Copy code

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

nodeapp-service LoadBalancer 10.96.0.1 20.75.101.2 3000:30000/TCP 2m

1. **Test the Application** Use the external IP to test the application:

bash

Copy code

curl http://<EXTERNAL-IP>:3000

Example:

bash

Copy code

curl http://20.75.101.2:3000

**Step 6: Debugging and Verification (Optional)**

1. **Inspect Deployment Details**

bash

Copy code

kubectl describe deployment nodeapp-deployment

1. **Access a Running Pod** Execute a shell in one of the pods for debugging:

bash

Copy code

kubectl exec -it <pod-name> -- /bin/bash

1. **Verify Logs** Check logs of a specific pod:

bash

Copy code

kubectl logs <pod-name>

**Step 7: Clean Up Resources**

1. **Delete the Deployment and Service** Remove the resources created:

bash

Copy code

kubectl delete -f deploy.yml

1. **Verify Deletion** Confirm all resources are deleted:

bash

Copy code

kubectl get all

**Summary**

This lab demonstrated:

1. Creating a Kubernetes deployment for a Node.js application.
2. Exposing the application using a LoadBalancer service.
3. Testing the application via the LoadBalancer's external IP.

This practical exercise reinforces deployment strategies and Kubernetes service management.

4o