CVT 1:

* Create New Virtual Machine
* If operating systerm file like ubuntu (.iso) is not installed , installed it and give its path
* Create Username
* Specify disk size
* Customize hardware : change RAM ( like 2048 MB : 2GB )

CVT 2:

* Either Open Docker Desktop terminal or Run CMD as administrator ( Docker Desktop Should be running in background )
* Run Following 6 Commands :

**1. docker --version**

* **Explanation**: This command shows the installed version of Docker on your system.

**Output**: It would display something like:  
  
 nginx  
CopyEdit  
Docker version 20.10.7, build f0df350

* **Why it's used**: To check which version of Docker is installed on your machine.

### **2. docker pull ubuntu**

* **Explanation**: This command downloads the latest ubuntu image from Docker Hub to your local machine.

**Output**:  
  
 vbnet  
CopyEdit  
Using default tag: latest

latest: Pulling from library/ubuntu

2726e237d1a3: Download complete

Digest: sha256:1e622c5f073b4f6bfad6632f2616c7f59ef256e96fe78bf6a595d1dc4376ac02

Status: Downloaded newer image for ubuntu:latest

docker.io/library/ubuntu:latest

* **Why it's used**: To download the ubuntu image and make it available for container creation.

### **3. docker images**

* **Explanation**: This command lists all the Docker images available on your system.

**Output**:  
  
 nginx  
CopyEdit  
REPOSITORY TAG IMAGE ID CREATED SIZE

ubuntu latest 1e622c5f073b 2 weeks ago 117MB

* **Why it's used**: To verify that the ubuntu image has been successfully pulled and is available locally.

### 

### 

### 

### 

(“**docker create --name container\_name ubuntu” to create container with your name )**

### 

### 

### **4. docker run -it ubuntu**

* **Explanation**: This command creates and runs a container based on the ubuntu image in interactive mode.

**Output**:  
  
 ruby  
CopyEdit  
root@47001fc18143:/# ls

bin dev home lib64 mnt proc run srv tmp var

boot etc lib media opt root sbin sys usr

root@47001fc18143:/# exit

* **Why it's used**: To start an interactive session inside an Ubuntu container and run commands like ls (list files).

### **5. docker ps -a**

* **Explanation**: This command lists all containers, both running and stopped.

**Output**:  
  
 java  
CopyEdit  
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

47001fc18143 ubuntu "/bin/bash" About a minute ago Exited (0) 38 seconds ago dreamy\_chaplygin

* **Why it's used**: To check the status of all containers on your system, including those that are stopped.

### **6. docker rm 47001fc18143**

* **Explanation**: This command removes a stopped container from the system. The container ID 47001fc18143 refers to the previously run Ubuntu container.
* **Output**: The container is removed, and you'll see no further output after the command finishes.
* **Why it's used**: To clean up by removing the stopped container, freeing up resources.

### **7. docker ps -a**

* **Explanation**: You run this command again to confirm that the container has been removed successfully.

**Output**:  
  
 nginx  
CopyEdit  
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

* **Why it's used**: To verify that the container was removed and no longer exists in the list.

CVT 3:

#### **1. Create Project Folder**

* **Create a folder on your system where you'll store the Dockerfile and other necessary files. For example: C:\Users\student\Downloads\docker-file.**
* **Open the Command Prompt in the folder where you want to create the Docker project.**

#### **2. Create Dockerfile**

* **In the docker-file folder, create a file named Dockerfile (without any extension). You can use Notepad or Command Prompt to create this file.**
* **Inside the Dockerfile, write the following instructions:**

**dockerfile**

**CopyEdit**

**# Use the official Nginx image from Docker Hub as a base image**

**FROM nginx:latest**

**# Set the working directory inside the container**

**WORKDIR /usr/share/nginx/html**

**# Copy the local 'index.html' file into the container**

**COPY index.html .**

**# Expose port 80 to make the container accessible from outside**

**EXPOSE 80**

* **Note: Ensure the file is named exactly as Dockerfile with no extension (like .txt).**

#### **3. Create index.html File**

* **In the same folder, create an index.html file. For example, you can write a simple HTML file with this content:**

**html**

**CopyEdit**

**<!DOCTYPE html>**

**<html lang="en">**

**<head>**

**<meta charset="UTF-8">**

**<meta name="viewport" content="width=device-width, initial-scale=1.0">**

**<title>Welcome to Nginx</title>**

**</head>**

**<body>**

**<h1>Hello from Docker and Nginx!</h1>**

**</body>**

**</html>**

#### **4. Build Docker Image**

* **Open Command Prompt and navigate to the folder where your Dockerfile and index.html are located.**
* **Run the following command to build your Docker image:**

**bash**

**CopyEdit**

**docker build -t my-nginx-image .**

**This will create a Docker image named my-nginx-image based on the Dockerfile in the current directory.**

#### **5. Run Docker Container**

* **After the image is successfully built, you can run a container from that image. Use the following command to start the container and map port 8080 on your machine to port 80 inside the container:**

**bash**

**CopyEdit**

**docker run -d -p 8080:80 my-nginx-image**

**This will run the container in detached mode (-d), and the application will be accessible via http://localhost:8080.**

#### **6. Access the Application**

* **Open a browser and go to http://localhost:8080 to see your index.html being served by Nginx inside the Docker container.**

#### **7. Verify and Manage Docker Containers**

* **To check if your container is running, use:**

**bash**

**CopyEdit**

**docker ps**

* **To stop the running container:**

**bash**

**CopyEdit**

**docker stop <container\_id>**

* **To remove the container:**

**bash**

**CopyEdit**

**docker rm <container\_id>**

* **To remove the Docker image:**

**bash**

**CopyEdit**

**docker rmi my-nginx-image**

CVT 5:

### **Kubernetes Commands Documentation**

**Step 1: Go to KLLrcoda**

* **Open your terminal or KLLrcoda and ensure you're logged into your Kubernetes environment.**

#### **1. kubectl run harsh --image=ubuntu**

**Explanation:  
 This command creates a pod named harsh and runs a container using the Ubuntu image. The container will run Ubuntu's default shell, and you can interact with it as if you were logged into an Ubuntu machine.**

**Output:**

**bash**

**CopyEdit**

**pod/harsh created**

**Why it's used:  
 To quickly create a pod running an Ubuntu container. This allows you to use the Ubuntu environment for testing or running commands.**

#### **2. kubectl get pods**

**Explanation:  
 This command lists all the pods running in your current Kubernetes cluster in the default namespace.**

**Output:**

**bash**

**CopyEdit**

**NAME READY STATUS RESTARTS AGE**

**nginx-pod 1/1 Running 0 5m**

**harsh 1/1 Running 0 10m**

**Why it's used:  
 To get an overview of the pods running in the cluster. You can see the pod names, their status (e.g., Running, Pending), and the number of restarts.**

#### **3. kubectl get pod harsh -o wide**

**Explanation:  
 This command gives more detailed information about a specific pod (harsh) in the cluster. The -o wide option adds additional details like the pod's IP address and node name.**

**Output:**

**bash**

**CopyEdit**

**NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES**

**harsh 1/1 Running 0 10m 172.17.0.4 worker-node <none> <none>**

**Why it's used:  
 To get more detailed information about the pod, such as its IP address and the node where it's running. This is useful for troubleshooting and understanding pod placement.**

#### **4. kubectl describe pod harsh**

**Explanation:  
 This command provides a detailed description of the pod (harsh), including container details, events, and status.**

**Output:**

**bash**

**CopyEdit**

**Name: harsh**

**Namespace: default**

**Priority: 0**

**Node: worker-node/172.17.0.2**

**Start Time: Thu, 25 Apr 2025 10:45:00 +0000**

**Labels: <none>**

**Annotations: <none>**

**Status: Running**

**IP: 172.17.0.4**

**Containers:**

**ubuntu-container:**

**Container ID: docker://abcd1234**

**Image: ubuntu:latest**

**Ports: 80/TCP**

**State: Running**

**Started: Thu, 25 Apr 2025 10:45:00 +0000**

**Events:**

**Type Reason Age From Message**

**---- ------ ---- ---- -------**

**Normal Scheduled 5m default-scheduler Successfully assigned default/harsh to worker-node**

**Normal Pulling 5m kubelet Pulling image "ubuntu:latest"**

**Normal Pulled 5m kubelet Successfully pulled image "ubuntu:latest"**

**Why it's used:  
 To get detailed information about the pod, including the containers running inside it, its status, and any recent events (like pulling images or starting the pod).**

#### **5. kubectl logs harsh**

**Explanation:  
 This command displays the logs from the container inside the pod (harsh). It’s helpful for debugging the container’s output.**

**Output:**

**bash**

**CopyEdit**

**root@harsh:/# ls**

**bin dev home lib64 mnt proc run srv tmp var**

**boot etc lib media opt root sbin sys usr**

**Why it's used:  
 To view the logs from the container and check its output (in this case, it’s the list of directories inside the Ubuntu container). This is useful for troubleshooting or monitoring container activity.**

#### **6. kubectl exec -it harsh -- /bin/bash**

**Explanation:  
 This command opens an interactive terminal session (/bin/bash) inside the running container of the harsh pod, allowing you to execute commands directly within the container.**

**Output:**

**bash**

**CopyEdit**

**root@harsh:/# ls**

**bin dev home lib64 mnt proc run srv tmp var**

**boot etc lib media opt root sbin sys usr**

**Why it's used:  
 To get an interactive shell inside the container. This allows you to inspect the container’s file system, run commands, or troubleshoot issues directly from within the container.**

#### **7. kubectl get pods --all-namespaces**

**Explanation:  
 This command lists all the pods across all namespaces in the Kubernetes cluster, not just the default namespace.**

**Output:**

**bash**

**CopyEdit**

**NAMESPACE NAME READY STATUS RESTARTS AGE**

**default nginx-pod 1/1 Running 0 5m**

**kube-system coredns-... 1/1 Running 0 20m**

**kube-system etcd-... 1/1 Running 0 20m**

**Why it's used:  
 To view all the pods in the entire cluster, including those in system namespaces like kube-system, which are important for the cluster’s internal operations (e.g., DNS, etcd, etc.).**

CVT 6

### **Kubernetes Pod Troubleshooting Documentation**

#### **Objective:**

**Create a pod in Kubernetes, find its IP address, and troubleshoot it using logs.**

### **Steps:**

#### **1. Create a Pod Named harsh:**

**kubectl run harsh --image=ubuntu --restart=Never**

* **Why it's used: This command is used to quickly create a pod running an Ubuntu container. It allows you to use the Ubuntu environment for testing or running commands.**

#### **2. Check the Status of All Pods:**

**kubectl get pods**

* **Why it's used: Lists all pods in your Kubernetes cluster in the default namespace.**

**Example Output:**

**NAME READY STATUS RESTARTS AGE**

**nginx-pod 1/1 Running 0 5m**

**harsh 1/1 Running 0 10m**

#### **3. Get Detailed Information About the harsh Pod:**

**kubectl get pod harsh -o wide**

* **Why it's used: Provides more details like the IP address, node, and other metadata.**

**Example Output:**

**NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES**

**harsh 1/1 Running 0 10m 172.17.0.4 worker-node <none> <none>**

#### **4. Create a Pod that Fails Intentionally (for Troubleshooting):**

**kubectl run test-pod2 --image=ubuntu --restart=Never --command -- /bin/bash -c "exit 1"**

* **Explanation:**
  + **--restart=Never: Ensures the pod does not restart after completion.**
  + **The command exits with code 1 to simulate a failure.**

**Expected Output:**

**pod/test-pod2 created**

#### **5. Check Pod Status Again:**

**kubectl get pods**

**Example Output:**

**NAME READY STATUS RESTARTS AGE**

**test-pod2 0/1 Error 0 15s**

* **Explanation: The test-pod2 is in an Error state, meaning it encountered an issue.**

#### **6. Describe the Failed Pod:**

**kubectl describe pod test-pod2**

**Example Output (Excerpt):**

**State: Terminated**

**Reason: Error**

**Exit Code: 1**

**Started: ...**

**Finished: ...**

* **Explanation:**
  + **Exit Code 1 shows the container exited with an error.**
  + **Useful to inspect container state, events, and reason for failure.**

#### **7. Check Pod Logs:**

**kubectl logs test-pod2**

* **Why it's used: Helps in identifying what happened just before the pod exited.**
* **Note: For this case, logs may be empty since the container exited immediately.**

#### **8. Delete the Pod (Optional):**

**kubectl delete pod test-pod2**

**Output:**

**pod "test-pod2" deleted**

#### **9. Recreate the Pod (Optional):**

**kubectl run test-pod2 --image=ubuntu --restart=Never**

* **Why it's used: To rerun the pod for further testing.**

#### **10. Verify Pod Status Again:**

**kubectl get pods**

**Example Output:**

**NAME READY STATUS RESTARTS AGE**

**test-pod2 0/1 Completed 0 7s**

### **Conclusion:**

* **You successfully created and managed a pod in Kubernetes.**
* **You retrieved its IP address using kubectl get pod -o wide.**
* **You simulated a failure and used describe and logs commands to troubleshoot.**
* **This end-to-end guide demonstrates how to debug pod issues using built-in Kubernetes tools.**