Lecture 05

SOFTWARE CONFIGURATION MANAGEMENT

Docker Introduction

1. Virtualization

Virtualization involves running a new system within an existing system. For instance, if the main operating system on your hardware is macOS, but you need to run Linux or Windows, you have several options. One option is to completely remove macOS, format the machine, and install the desired operating system.

However, this process involves reinstalling everything and losing access to the original system. Another option is to use swappable hard disks, allowing you to physically switch between operating systems. However, virtualization often provides a more convenient solution, allowing you to run multiple operating systems concurrently on the same hardware.

Sometimes, virtualization may not be enabled by default and needs to be activated in the BIOS or system settings.

Virtualization remains a key feature in cloud computing, although its implementation may differ significantly. Remember, virtualization is essential in any operating system environment.

A virtualization machine allows running various base operating systems of choice concurrently without issues. Deleting a guest operating system removes all its associated traces from the existing setup.

2. Containerization

Containerization, on the other hand, differs from virtualization in many aspects. It involves sharing resources and information in a manner different from traditional virtual machines.

Creating from the Linux world, containerization offers a lightweight alternative to virtualization.

It shares resources efficiently, extracting only the necessary components into separate environments.

Docker is a popular tool used within containerized environments.

In both virtualization and containerization, users can install various operating systems and software, including Windows, Linux, and even macOS, depending on their needs and configurations.

3. Docker

Docker, the concept of images plays a central role. Before running anything in Docker, you must either create a Docker image or download one.

Docker design for the Linux operating system.

Windows doesn't have containerization capabilities. Alternative: Docker Desktop

Use Windows terminal for the commands instead of PowerShell or Command-Line.

Check whether docker is properly installed with **docker –v**

Questions:

1. What is Virtualization?
2. What are Containers and Docker? What are the differences between each?
3. Find 5 other Docker commands excluding pull, run, stop, start, and exec.

Answers

1. Virtualization:

Virtualization is a technology that allows multiple operating systems or applications to run on the same physical hardware simultaneously. It creates virtual instances, often referred to as virtual machines (VMs), which act as independent entities with their own operating systems, applications, and resources, even though they share the underlying hardware.

2. Containers and Docker:

Containers: Containers are lightweight, portable, and isolated environments that package applications and their dependencies. They provide

consistency across different environments, making it easier to develop, deploy, and scale applications.

Docker: Docker is a platform for developing, shipping, and running applications in containers. It provides tools and a runtime for creating and managing containers. Docker simplifies the containerization process, making it accessible for developers and enabling the seamless deployment of applications across various environments.

Differences:

- Containers are a broader concept, while Docker is a specific containerization platform.
- Docker uses a client-server architecture, includes a registry for sharing container images (Docker Hub), and provides a command-line interface (CLI) for managing containers.

3. Other Docker Commands:

- **build**: Builds a Docker image from a Dockerfile.
  docker build -t imagename: tag

- **images:** Lists the local Docker images.
  docker images

- **inspect:** Displays detailed information about a container, image, or network.
  docker inspect container_or_image_id

- **logs:** Retrieves the logs from a running container.
  docker logs container_id

- **rm:** Removes one or more stopped containers.
  docker rm container_id

---

4. Docker images

Docker images, which are essentially blueprints, are based on Linux operating systems.

Once the docker image is run we call it a Docker container.

You need a Docker image containing the necessary components and configurations to initiate a Docker container.

Docker images can be found on **Docker Hub**, which is the primary repository for Docker images. On Docker Hub, users can explore various available images or even create and upload their own. One notable Docker image is **Alpine.**

Others : hello world, WordPress Debian and

toys

5. Alpine

This is based on a lightweight Linux variant called Alpine Linux. Alpine Linux is known for its minimalism and efficiency, making it one of the smallest Linux distributions available.

It includes only essential components and utility tools, making it lightweight compared to other distributions like **Red Hat or Ubuntu**.

Alpine Linux also comes in different versions, denoted by tags. Specifying the version when pulling an image is crucial to ensure compatibility with your system.

When you use the **"latest"** tag to pull Docker images, you're essentially fetching the most recent version available on Docker Hub.

However, this approach may not always be suitable, especially in production environments, because newer versions may introduce changes or updates that could potentially break your application or environment.

6. Download Docker Image

To download an image from Docker Hub, you use the **docker pull** command followed by the **image name**.

Docker Hub is one of the main repositories for Docker images, which are sometimes referred to as **container registries**

For example, if you want to download the Alpine image, you would use the command :

docker pull Alpine.

However, it's important to note that this command fetches the image with the "**latest**" tag by default.

The "latest" tag isn't always advisable, as it may not ensure stability or compatibility with your system.

A better practice is to specify a specific version or tag when pulling images.

For instance, to pull Alpine version 3.8, you would execute:

docker pull Alpine:3.8

This ensures that you know exactly which version of the image you're using.

Docker images are designed to be compact and efficient. Even though they contain various layers, including Linux libraries, they are often much smaller than traditional operating system installations. This is because Docker images leverage existing system resources and only include what's necessary to run specific applications within containers.

7. virtualization and containerization

Virtualization and containerization (not "conization") are distinct concepts. While they share some similarities, they differ in how they use resources and separate environments.

In containerization, some resources are shared between the host and guest environments, which helps keep the image sizes relatively small compared to virtual machine (VM) images.

Container images, like the Alpine image, are often lightweight and optimized for specific tasks.

Container images can include not only the base operating system but also **pre-configured software packages**.

For example, the **PostgreSQL** image comes with **PostgreSQL pre-installed**, allowing users to run a PostgreSQL instance directly from the image.

8. PostgreSQL

This is also an image like Alpine. Container images can include not only the base operating system but also pre-configured software packages.

Users can find images with other pre-installed software, such as Python or MySQL.

9. Easiness of containerization

The flexibility of containerization allows developers to work with multiple versions of software simultaneously.

For instance, you can have both MySQL 5.7 and MySQL 8.0.35 images pulled from Docker Hub and run them concurrently on the same machine.

In containerization, you can specify which ports each container should use, allowing multiple versions of the same software to coexist without conflict. This flexibility simplifies development and testing workflows, as developers can easily switch between different software versions as needed.

Furthermore, removing a specific version of a containerized software is straightforward. You can simply remove the container and image associated with it, leaving no trace of its presence on the host machine.

10. base image

Every Docker image should have a base image, which serves as the foundation upon which the image is built. For instance, the PostgreSQL (post) image is based on Debian. However, users have the flexibility to choose other base images according to their preferences and requirements.

In the case of the PostgreSQL image, there are two versions available: one based on Ubuntu/Debian and the other based on Alpine Linux. Users can select the base image that best fits their needs. If they prefer a lightweight and more basic image, they can opt for the Alpine version.

This flexibility allows users to tailor their Docker images to specific use cases and requirements. Depending on the desired features, performance,

and resource utilization, users can choose the appropriate base image and build their Docker images accordingly.

11. Installing the docker image

Executing the Docker Pull Command: You've copied the command to your terminal to download the Docker image from Docker Hub to your local machine. Once executed, it retrieves the specified image with the version or tag mentioned.

Verifying Downloaded Images: After the download completes, you can verify the downloaded images by using the

- **docker images** [commad]

This command lists all the images present on your local machine, whether you pulled them from Docker Hub or created them yourself.

Each image entry includes details like the repository, tag version, unique image ID, creation time, and size.

After downloading the Docker image to your local machine, you need to run it.

```
C:\Users\hp>docker images
REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
alpine       3.18.6    d3782b16ccc9   3 weeks ago   7.34MB
```

- **docker run** [commad]

command is used for this purpose. You specify the image you want to run, either by the **repository name and tag or by the image ID** obtained from the docker images command.

```
C:\Users\hp>docker run d3782b16ccc9
```

After running the Docker image, you can check if the container is running using the

- **docker ps**

command. This command lists all the running containers along with their container ID, image used, command executed, creation time, status, and other details.

The docker ps command output includes details:

- container ID
- image name
- command executed
- creation time
- status
- image used to create the container
- command executed inside the container
- whether the container is currently running or not

If you don't see any running containers with docker ps

- **docker ps -a**

you can use the -a flag to display all containers, including those that are not currently running. This helps you track containers that were previously executed but have since stopped running.

- **docker ps -a:** List all containers, including those that have exited, and find the Container ID.
- **docker start <container_id>:** Restart the stopped container using its Container ID.
- **docker attach <container_id>:** Attach to the shell or command-line interface of the container to interact with it.

We can use this 2 to enter the container after run

```
docker run -td 0ce902b62a96
```

Last number = image

Once attached, your terminal essentially becomes the primary input/output interface for that process.

docker exec -it <container_id> /bin/s

- Perform your actions within the container.
- When finished, exit the container's shell or command-line interface as you would normally.
- Run container:

  ```
  docker run -td 0ce902b62a96
  ```

  Last number = image

- Create a new container with a custom name: docker run -–name Minuri –td 30987739

Docker Container Management

- used to start a Docker container and attach your current terminal's input/output to it in interactive mode.
  docker start -i <container_id>

- Naming the Container:
  (Runs a container with a custom name (<container_name>) and initiates an interactive shell session within it using /bin/sh.)
  docker run --name <container_name> -it <image_name> /bin/sh

Create a new container and enter it. At this time we want to add a new container name that is not used in the previous time.

- Stopping a Running Container:
  docker stop <container_id>

- Executing Interactive Terminal in Running Container:
  docker exec -it <container_id> /bin/sh

the sh-terminal name we used in the Alpine image now we are not machines, we are in the container. Enter Existing container.

- Use apk add <package_name> to install packages within an Alpine-based container,
  apk add git.

- Exiting the Container Shell:
  Use **Ctrl + D or the exit** command to exit from the container's shell back to the host system.

- Stopping a Running Container:
  Use docker **stop <container_id>** to stop a running container.

- Removing Containers:
  Use docker **rm <container_id>** to remove a specific container instance.

  To remove multiple containers at once, list their IDs with spaces and use docker rm <container_id1> <container_id2> ...

- -td
  This is combination of -t and -d

  Detached Mode (-d): Starts the container in the background and allows you to continue using the terminal for other tasks without being attached to the container's console.

  Pseudo-TTY (-t): Allocates a pseudo-terminal, which enables interaction with the container's command-line interface.

  Ex : docker run -td alpine

  This command starts a new container using the Alpine Linux image.

  The -td flags ensure that the container runs in detached mode with a pseudo-TTY allocated.

  After executing this command, you'll get back the container ID, indicating that the container is running in the background.

Activity

finding five other Docker commands?

- **docker ps:** This command lists the running containers.
- **docker build:** Used to build a Docker image from a Dockerfile.
- **docker stop:** Stops one or more running containers.
- **docker-compose up:** Starts services defined in a docker-compose.yml file.
- **docker logs:** Displays logs generated by a running container.

<u>Starting a MongoDB Server with Docker:</u>

Use the command docker run followed by the **MongoDB image** name to start the server.

Docker Environment Variables:

Environment variables can be passed to Docker containers using the **-e** flag.

Example:

```
run -e password=minuri -d d3782b16ccc9
```

Last number: Image Name

Use **docker run --help** to see all available customization options.

Various options include limiting CPU usage, setting resource limits, and configuring devices.

Docker Run Flags:

-td: Starts a container in detached mode with a pseudo-TTY allocated.

-**d:** Runs containers in the background and prints container ID.

-e: Sets environment variables inside the container
