**CIS:470 Report BY Rahul Gamara HW 4**

**Docker Report – Flask Application**

**1) Why Docker is Important in Modern Software Development**

Docker is a tool that helps developers create small, portable "containers" that hold everything an app needs to run — like the code, settings, and libraries. This way, the app works the same no matter where it's run: on a laptop, server, or cloud platform.

Think of it like packing all your clothes and gear in a suitcase. No matter where you go, you're fully prepared — you don’t have to worry about forgetting socks or toothpaste!

**Example 1:** When a developer writes an app and sends it to a teammate or uploads it to a server, Docker makes sure the app works exactly the same everywhere. No more "it works on my machine" problems.

**Example 2:** Companies use Docker to launch apps on the internet (like websites or services). Because Docker keeps everything inside a container, it’s easier and faster to update or fix the app without breaking other things.

**2) How Docker containers are different from virtual machines, provide 2 examples:**

Both Docker and Virtual Machines help create separate environments for apps, but they work differently.

**Difference 1 – Size and Speed:**  
Docker is much lighter and faster. It uses the existing system’s core (called the operating system), so it can start and stop quickly. VMs are heavier — they run their own full operating system inside, which takes more time and space.

**Example:** Starting a Docker container is like flipping on a light switch. Starting a VM is like starting a whole new house generator.

**Difference 2 – Efficiency:**  
Docker allows many containers to run at once using the same resources, while VMs use more computer power and memory.

**Example:** You can run 10 Docker apps using the same space that maybe 2 or 3 VMs would need.

**3) examples for use cases for docker-compose, and make the files tree for each use case**

**Docker Compose** is a tool that helps manage multiple containers (parts of an app) at the same time. It’s like a team manager organizing different players on a field.

**Use Case 1 – Web App + Database**

A simple app that shows data (like a website) and a separate database that stores the data.

**Files Tree:**

csharp

project/

├── docker-compose.yml

├── app/

│ └── app.py

├── db/

│ └── init.sql

**Explanation:** One container runs the app, another container runs the database. Compose helps them talk to each other easily.

**Use Case 2 – API + Redis Cache + Worker**

A system where:

* One container handles incoming tasks (API),
* One stores fast-access data (Redis),
* One processes background jobs (Worker).

**Files Tree:**

arduino

project/

├── docker-compose.yml

├── api/

│ └── app.py

├── redis/

│ └── (uses official Redis image)

├── worker/

│ └── worker.py

**Explanation:** Compose starts all 3 containers together and makes sure they work as a team.