* **What is a Container?**
  + Containers are isolated environments that package an application and its dependencies (e.g., libraries, frameworks) into a single unit. This ensures consistency across different computing environments, solving issues like “it works on my machine but not elsewhere.”
  + Example analogy: Think of a container as a shipping container that holds everything needed for an app, ready to be shipped and run anywhere without changes.
  + Benefits include efficiency (using fewer resources than VMs), isolation (preventing conflicts between apps), and portability (runs on any system with Docker installed).
* **Docker vs VM: A Comparison**
  + VMs run a full operating system on top of the host hardware, including a kernel, making them resource-intensive and slower to start. Containers, however, share the host OS kernel, making them lighter and faster.
  + Table: Comparison of Docker Containers and VMs

| **Aspect** | **Docker Containers** | **Virtual Machines (VMs)** |
| --- | --- | --- |
| Resource Usage | Lightweight, shares host OS kernel | Heavy, includes full OS |
| Startup Time | Fast (seconds) | Slow (minutes) |
| Isolation | Application-level isolation | OS-level isolation |
| Use Case | Development, microservices, CI/CD | Legacy apps, testing multiple OSes |

* + Research suggests containers are ideal for modern, scalable applications, while VMs are better for scenarios requiring full OS isolation, such as running different operating systems.
* **Why Use Containers?**
  + They provide consistency, portability, and efficiency, solving issues like "it works on my machine but not elsewhere."
  + Containers are critical for cloud-native applications and microservices architecture, enabling rapid deployment and scaling.

**Lab: Docker Installation (30 min)**

* Installation is the first practical step. Students should choose their OS and follow the official guides:
  + Mac: [Install Docker Desktop for Mac](https://docs.docker.com/docker-for-mac/install/)
  + Windows: [Install Docker Desktop for Windows](https://docs.docker.com/docker-for-windows/install/)
  + Linux (Ubuntu): [Install Docker on Ubuntu](https://docs.docker.com/install/linux/docker-ce/ubuntu/)
* After installation, verify with docker run hello-world. This command pulls a test image and runs it, confirming Docker works. On Linux, you may need to prefix commands with sudo or add your user to the docker group for permissions.

**Day 2: Building and Running a Simple Python App in a Container - Detailed Steps**

**Theory: Docker Components and Commands (30 min)**

* **Docker Components: Images, Containers, Dockerfile**
  + **Image:** A read-only template with the application code and dependencies, like a blueprint. Example: python:3.9-slim is a base image for Python apps.
  + **Container:** A running instance of an image, like a live version of the blueprint. You can have multiple containers from one image, each with its own state.
  + **Dockerfile:** A script with instructions to build an image, specifying steps like installing dependencies or setting ports. It’s like a recipe for creating the image.
* **Basic Docker Commands:**
  + docker build: Builds an image from a Dockerfile.
  + docker run: Creates and starts a container from an image.
  + docker ps: Lists all running containers.
  + docker stop: Stops a running container.
  + docker rm: Removes a container.

**Lab: Create and Run a Python App in a Container (1.5 hours)**

* **Create a Simple Python Application:**
  + Create a project directory (e.g., my\_python\_app) and add:
  + app.py: A simple Python script:

python

CollapseWrapCopy

print("Hello, World from a container!")

* **Write a Dockerfile:**
  + Create a Dockerfile in the same directory:

dockerfile

CollapseWrapCopy

FROM python:3.9-slim

WORKDIR /app

COPY . .

CMD ["python", "app.py"]

* + Explanation:
    - FROM python:3.9-slim: Uses a lightweight Python 3.9 image as the base.
    - WORKDIR /app: Sets the working directory inside the container.
    - COPY . .: Copies all project files into the container.
    - CMD ["python", "app.py"]: Runs the Python script on container start.
* **Build the Docker Image:**
  + Open a terminal or command prompt in the project directory.
  + Run: docker build -t my\_python\_app .
  + This builds the image and tags it as my\_python\_app.
* **Run the Container:**
  + Run: docker run my\_python\_app
  + Verify the output: Hello, World from a container!
* **Experiment with Docker Commands:**
  + List running containers: docker ps
  + Stop the container (if running): docker stop <container\_id>
  + Remove the container: docker rm <container\_id>