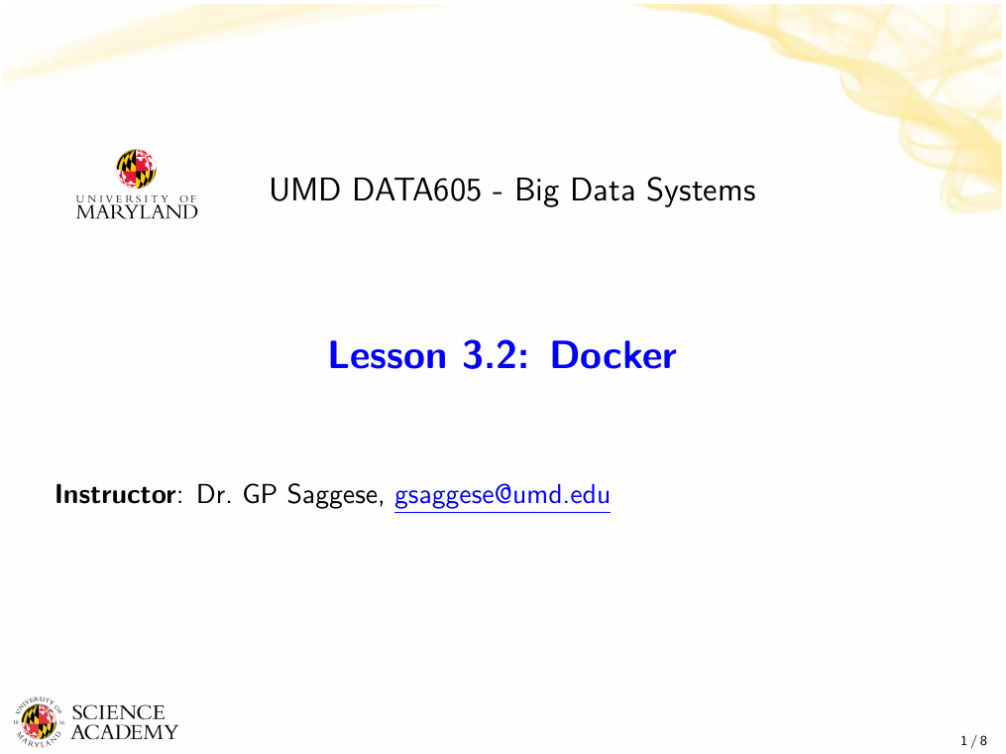
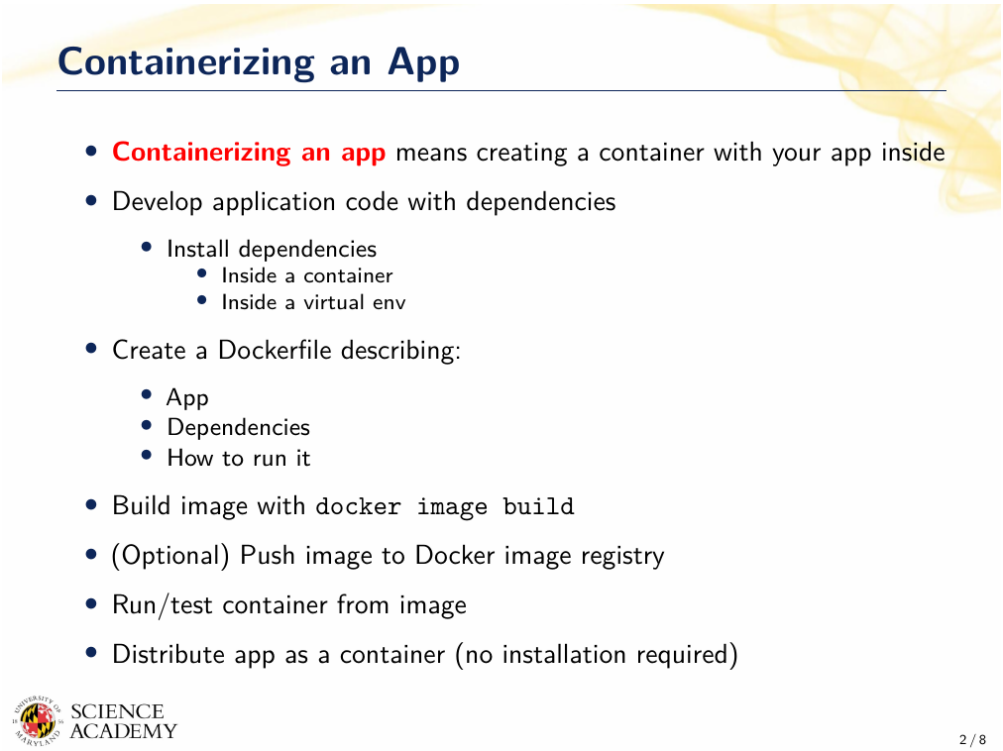
Lesson 3.2: Docker



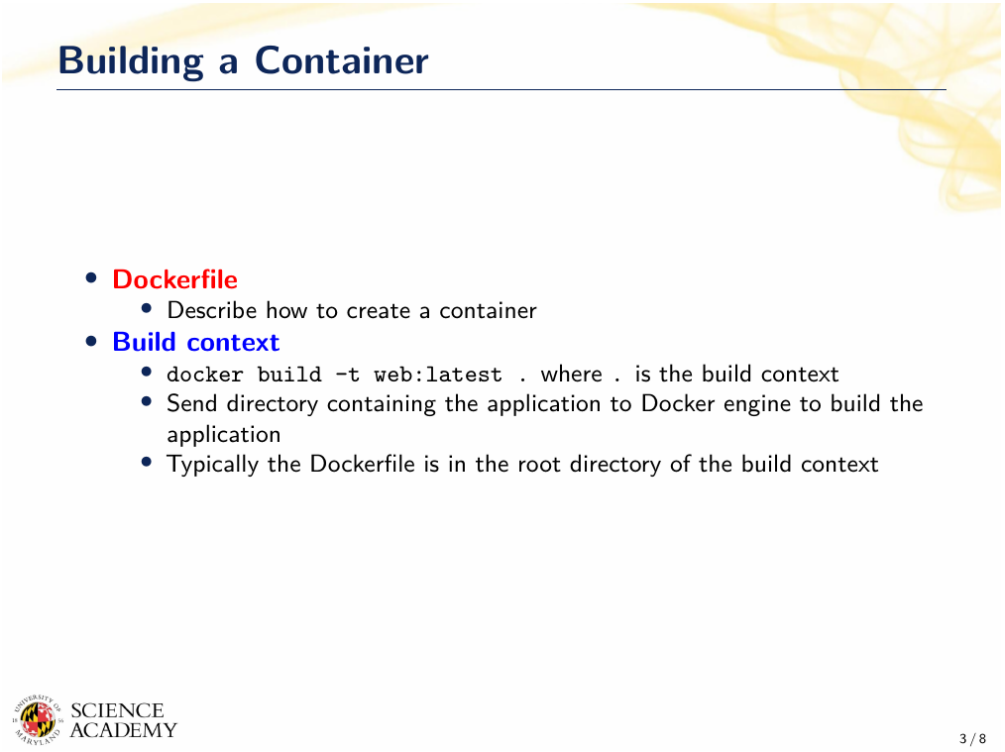
2 / 8: Containerizing an App



- **Containerizing an app** involves packaging your application and all its dependencies into a single, lightweight container. This makes it easy to run the app consistently across different environments.
- **Develop application code with dependencies**:
  - When developing your app, you need to ensure all necessary libraries and tools are included.
  - *Install dependencies* can be done in two main ways:
    - **Inside a container**: This ensures that the app and its dependencies are bundled together, making it portable.
    - **Inside a virtual environment**: This is another method to manage dependencies, but it is more common in traditional development setups.
- **Create a Dockerfile describing**:
  - The **App**: Specify the application code and any necessary configurations.
  - **Dependencies**: List all the libraries and tools your app needs to run.
  - **How to run it**: Provide instructions on how to start the application within the container.
- **Build image with `docker image build`**: This command compiles the Dockerfile into a Docker image, which is a snapshot of your app and its environment.
- **(Optional) Push image to Docker image registry**: By uploading your image to a registry, you make it accessible to others or to different environments, facilitating easy sharing and deployment.

- **Run/test container from image**: Once the image is built, you can run it to test if the app works as expected in the containerized environment.
- **Distribute app as a container (no installation required)**: Containers simplify app distribution because they encapsulate everything needed to run the app, eliminating the need for users to install dependencies separately.

3 / 8: Building a Container



- **Building a Container**
  - **Dockerfile**
    - The Dockerfile is a script that contains a series of instructions on how to build a Docker container. Think of it as a recipe that tells Docker what base image to use, what software to install, and what commands to run. This file is crucial because it defines the environment and configuration of your containerized application. By writing a Dockerfile, you ensure that your application can run consistently across different environments.
  - **Build context**
    - The build context is the set of files that Docker needs to build the container. When you run the command `docker build -t web:latest .`, the `.` specifies the current directory as the build context. This means Docker will use all the files in this directory to build the container. It's important to ensure that the Dockerfile is located in the root directory of the build context so Docker can find it easily. This setup allows Docker to package your application and its dependencies into a container efficiently.

4 / 8: Dockerfile: Example

Dockerfile: Example

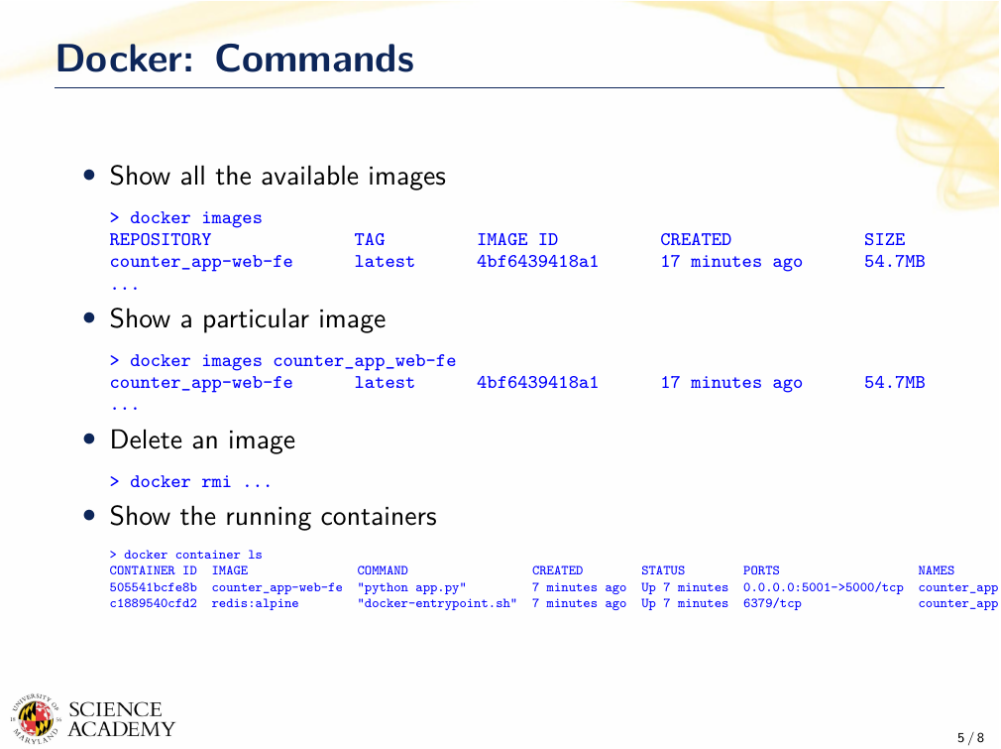
```
FROM python:3.8-slim-buster
LABEL maintainer="gsaggese@umd.edu"

WORKDIR /app

COPY requirements.txt requirements.txt
RUN pip3 install -r requirements.txt

COPY . .

CMD ["python3", "-m", "flask", "run", "--host=0.0.0.0"]
```

4 / 8

- **FROM python:3.8-slim-buster**: This line specifies the base image for our Docker container. We're using a lightweight version of Python 3.8, which is based on Debian Buster. This choice helps keep the container size small while providing the necessary Python environment.
- **LABEL maintainer="gsaggese@umd.edu"**: Here, we're adding metadata to the image. The `maintainer` label indicates who is responsible for maintaining the Dockerfile. This can be useful for others who might need to contact the maintainer for questions or issues.
- **WORKDIR /app**: This command sets the working directory inside the container to `/app`. Any subsequent commands will be run in this directory. It helps organize the file structure within the container.
- **COPY requirements.txt requirements.txt**: This line copies the `requirements.txt` file from the host machine into the container. This file typically contains a list of Python packages that the application depends on.
- **RUN pip3 install -r requirements.txt**: After copying the `requirements.txt` file, this command installs the required Python packages listed in it. Using `pip3` ensures that the packages are installed for Python 3.
- **COPY . .**: This command copies all files from the current directory on the host machine into the working directory of the container. This includes the application code and any other necessary files.
- **CMD ["python3", "-m", "flask", "run", "–host=0.0.0.0"]**: This is the command that will be executed when the container starts. It runs a Flask application, which is a popular

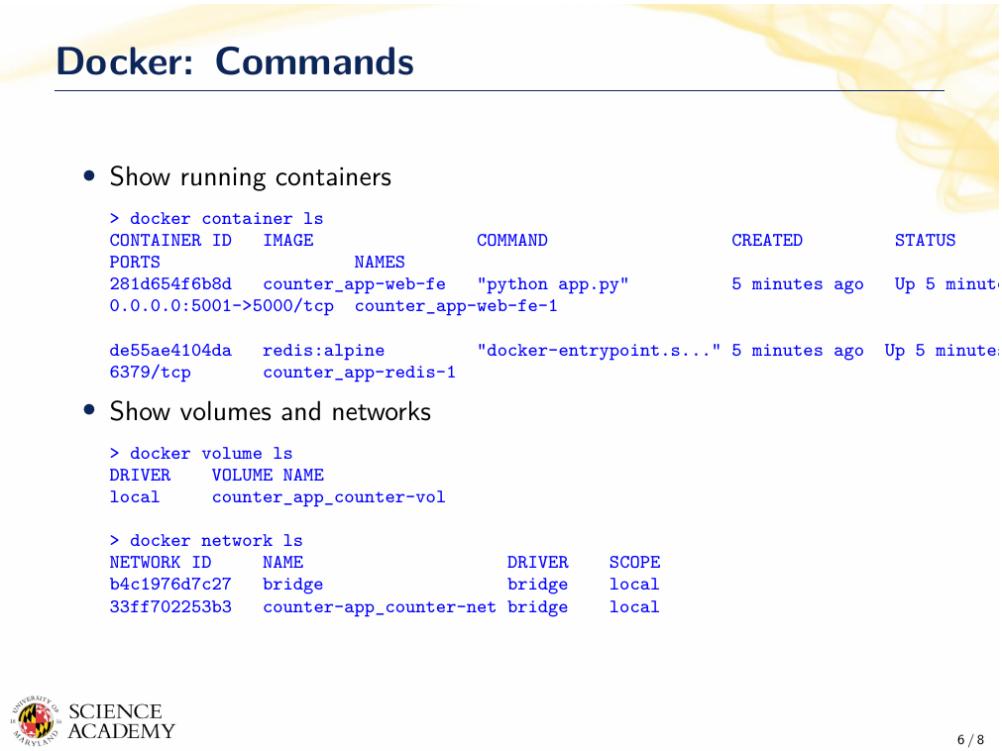
web framework for Python. The `--host=0.0.0.0` option makes the server accessible from outside the container, which is important for testing and deployment.

5 / 8: Docker: Commands



- **Show all the available images**
  - The command `docker images` is used to list all Docker images stored on your system. Each image is a snapshot of a container's file system and can be used to create running containers. The output includes details such as the repository name, tag, image ID, creation date, and size. This helps you manage and keep track of the images you have downloaded or created.
- **Show a particular image**
  - To view details of a specific image, use `docker images` followed by the image name. This command filters the list to show only the specified image, making it easier to find information about a particular image when you have many stored on your system.
- **Delete an image**
  - The command `docker rmi` is used to remove one or more Docker images from your system. This is useful for freeing up disk space or removing outdated images. You need to specify the image ID or name to delete it. Be cautious, as deleting an image will prevent you from creating new containers from it unless you re-download or rebuild it.
- **Show the running containers**
  - Use `docker container ls` to list all currently running containers. This command provides details such as container ID, image name, command being executed, creation time, status, ports, and container names. This information is crucial for monitoring and managing active containers, ensuring they are running as expected and troubleshooting any issues.

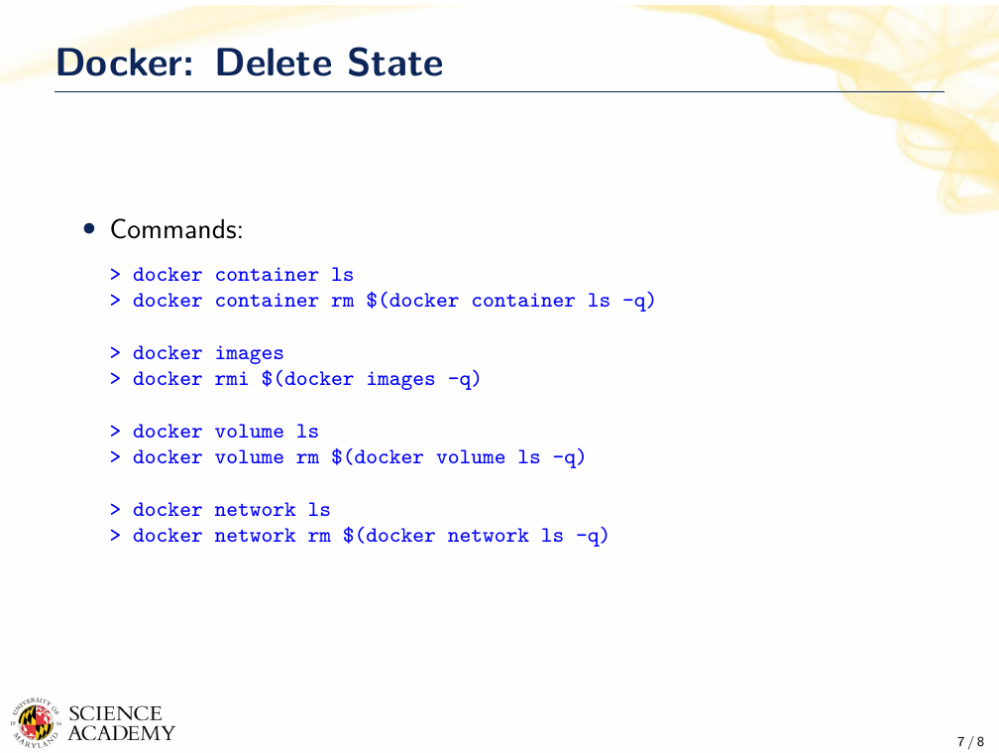
6 / 8: Docker: Commands



- **Show running containers**
  - The command `docker container ls` is used to list all the currently running Docker containers on your system. This is a useful command when you want to see which applications are actively running in your Docker environment.
  - The output provides several details:
    * **CONTAINER ID**: A unique identifier for each running container.
    * **IMAGE**: The Docker image from which the container was created.
    * **COMMAND**: The command that is being executed inside the container.
    * **CREATED**: How long ago the container was started.
    * **STATUS**: The current state of the container, such as "Up" and the duration.
    * **PORTS**: Information about port mappings between the host and the container.
    * **NAMES**: The name assigned to the container, which can be used to reference it in other commands.
- **Show volumes and networks**
  - The command `docker volume ls` lists all the Docker volumes on your system. Volumes are used to persist data generated by and used by Docker containers.
    * **DRIVER**: The type of driver used for the volume, typically "local".
    * **VOLUME NAME**: The name of the volume, which can be used to reference it in other commands.
  - The command `docker network ls` lists all the Docker networks. Networks allow containers to communicate with each other and with the outside world.
    * **NETWORK ID**: A unique identifier for each network.
    * **NAME**: The name of the network.
    * **DRIVER**: The type of network driver, such as "bridge", which is the default net-

work driver.
    * **SCOPE**: Indicates whether the network is local or global.

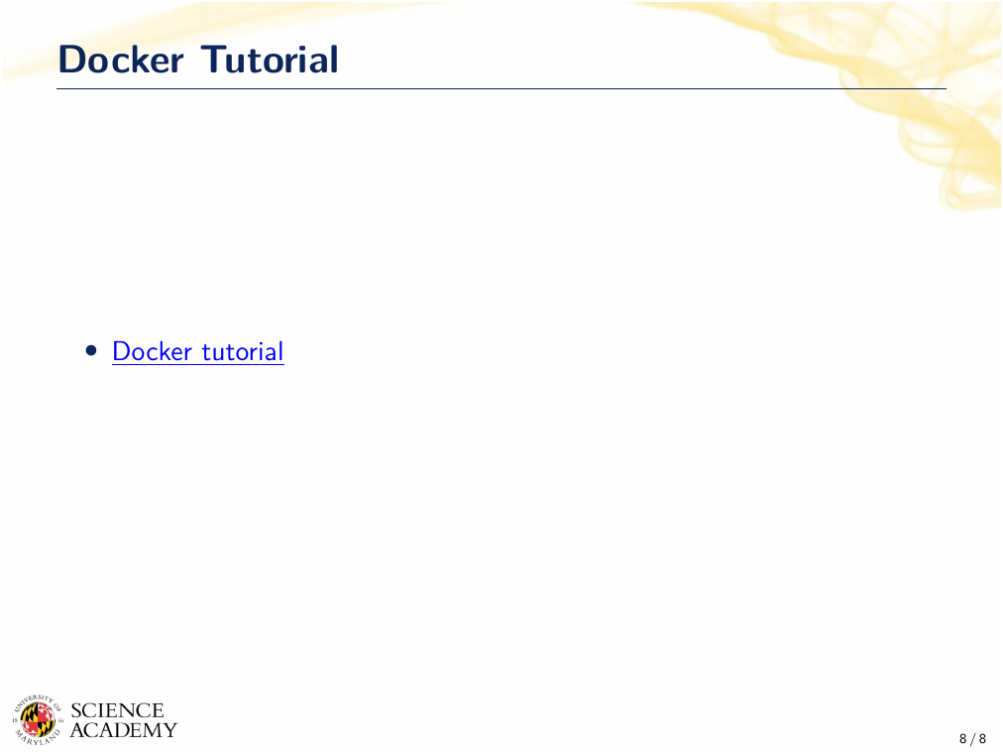
7 / 8: Docker: Delete State



- **Docker: Delete State**
  - This slide is about removing different types of Docker resources. Docker is a tool that helps developers create, deploy, and run applications in containers. Sometimes, you need to clean up these resources to free up space or start fresh.
- **Commands:**
  - **List and Remove Containers:**
    * `docker container ls`: This command lists all running containers. Containers are like lightweight virtual machines that run your applications.
    * `docker container rm $(docker container ls -q)`: This command removes all running containers. The `-q` option lists only the container IDs, which are then passed to the `rm` command to delete them.
  - **List and Remove Images:**
    * `docker images`: This command lists all Docker images. Images are the blueprints for containers, containing everything needed to run an application.
    * `docker rmi $(docker images -q)`: This command removes all images. Like with containers, the `-q` option lists only the image IDs for removal.
  - **List and Remove Volumes:**
    * `docker volume ls`: This command lists all Docker volumes. Volumes are used to persist data generated by and used by Docker containers.
    * `docker volume rm $(docker volume ls -q)`: This command removes all volumes. Removing volumes will delete any data stored in them, so use this command with

caution.
  - **List and Remove Networks:**
    * `docker network ls`: This command lists all Docker networks. Networks allow containers to communicate with each other.
    * `docker network rm $(docker network ls -q)`: This command removes all networks. Be careful when removing networks, as it can disrupt communication between containers.

8 / 8: Docker Tutorial



- **Docker Tutorial**
  - This slide introduces a tutorial on Docker, which is a platform used to develop, ship, and run applications inside containers. Containers are lightweight, portable, and ensure that software runs consistently across different computing environments.
- **Docker tutorial**
  - The link provided directs you to a detailed tutorial on Docker. This tutorial is likely to cover the basics of Docker, including how to install it, create Docker images, and run containers. It might also delve into more advanced topics such as Docker Compose, which is used for defining and running multi-container Docker applications.
  - Understanding Docker is crucial for anyone involved in software development or deployment, as it simplifies the process of managing application dependencies and environments. This tutorial is a valuable resource for beginners and those looking to deepen their understanding of containerization.
  - *Key takeaway*: Docker helps in creating a consistent environment for applications, making it easier to develop, test, and deploy software across different platforms.