Part 1:

Installing Docker on Ubuntu Server

1. Update Your System: Ensure your system package database is up-to-date.

```
$ sudo apt update
$ sudo apt upgrade
```

2. Install Docker: Install Docker using the convenience script provided by Docker.

```
$ curl -fsSL https://get.docker.com -o get-docker.sh
$ sudo sh get-docker.sh
```

3. Add User to Docker Group (Optional): If you want to run Docker commands without sudo, add your user to the docker group.

```
$ sudo usermod -aG docker ${USER}
```

Log out and log back in for the group changes to take effect.

4. Start and Enable Docker: Ensure Docker starts on boot.

```
$ sudo systemctl enable docker
$ sudo systemctl start docker
```

5. Verify Docker Installation: Check the Docker version to ensure it's installed correctly.

```
$ docker --version
```

6. Deploying a Sample Web Application using Docker

6.1 Pull a Sample Web Application Image: For this guide, we'll use a simple HTTP server image from Docker Hub.

```
$ docker pull httpd
```

6.2 Run the Web Application: Start a container using the httpd image. This will run the web server on port 8080.

```
$ docker run -d -p 8080:80 --name sample-webapp httpd
```

6.3 Access the Web Application: If you're accessing the server locally, open a web browser and navigate to: (Since you are connected via SSH lets install a text-based web browser lynx.)

```
$ sudo apt-get install lynx
$ lynx http://localhost:8080
```

6.4 Stop and Remove the Web Application (Optional):
When you're done testing the web application, you can stop and remove the container.

```
$ docker stop sample-webapp
$ docker rm sample-webapp
```

Extra Ref:
https://linuxhint.com/best_linux_text_based_browsers/
https://romanzolotarev.com/ssh.html

Basic Docker Commands and Their Usage

- docker --version
  Usage: Displays the Docker version installed.
  Example: docker --version
- docker info
  Usage: Provides detailed information about the Docker installation.
  Example: docker info
- docker pull <image_name>
  Usage: Downloads a Docker image from Docker Hub.
  Example: docker pull nginx
- docker build -t <image_name>:<tag> <path>
  Usage: Builds a Docker image from a Dockerfile located at <path>.
  Example: docker build -t myapp:latest .
- docker images
  Usage: Lists all available Docker images on the system.
  Example: docker images
- docker run <options> <image_name>
  Usage: Creates and starts a container from a Docker image.
  Example: docker run -d -p 80:80 nginx
- docker ps
  Usage: Lists running containers.
  Example: docker ps
- docker ps -a
  Usage: Lists all containers, including stopped ones.
  Example: docker ps -a
- docker stop <container_id/container_name>
  Usage: Stops a running container.
  Example: docker stop my_container
- docker rm <container_id/container_name>
  Usage: Removes a stopped container.
  Example: docker rm my_container

- docker rmi <image_name>
  Usage: Removes a Docker image.
  Example: docker rmi nginx
- docker logs <container_id/container_name>
  Usage: Displays logs from a running or stopped container.
  Example: docker logs my_container

Troubleshooting Common Docker Container Issues

- Container Fails to Start
  Check Logs: Use docker logs <container_name> to check for any error messages.
  Inspect Configuration: Ensure that the Docker run command has the correct parameters, such as port mappings and volume mounts.
- Networking Issues
  Check IP Address: Use docker inspect <container_name> | grep IPAddress to find the container's IP address.
  Check Port Bindings: Ensure that the ports inside the container are correctly mapped to the host using the -p option.
  You may use docker port <container_name> to further check the port mapping.
- File or Directory Not Found in Container
  Check Volumes: Ensure that directories or files from the host are correctly mounted into the container using the -v option.
  You may use docker volume ls to list all volumes mapped and docker volume inspect <volume_name> to inspect a selected volume.
  Inspect Image: Use docker image inspect <image_name> to see the image's layers and ensure the required files are present.
- Container Performance Issues
  Check Resources: Containers might face performance issues if they're not allocated enough resources. Use docker stats to check the resource usage of running containers.
  Limit Resources: When running a container, you can use flags like --cpus and --memory to limit its resources.
  You can use docker top <container_name> to see some stats.
- Image-Related Issues
  Pull Latest Image: Ensure you have the latest version of the image using docker pull <image_name>.
  Check Dockerfile: If you're building your own image, ensure that the Dockerfile has the correct instructions.
- Permission Issues
  User Mappings: If a containerized application can't access certain files, it might be a user permission issue. Ensure that the user inside the container has the necessary permissions.
  Use --user Flag: When running a container, you can specify which user the container should run as using the --user flag.

Part 2:

What is a Dockerfile?

A Dockerfile is a script containing a set of instructions used by Docker to automate the process of building a new container image. It defines the environment inside the container, installs necessary software, sets up commands, and more.
Basic Structure of a Dockerfile

A Dockerfile consists of a series of instructions and arguments. Each instruction is an operation used to build the image, like installing a software package or copying files. The instruction is written in uppercase, followed by its arguments.
Key Dockerfile Instructions

FROM: Specifies the base image to start from. It's usually an OS or another application.
Example: FROM ubuntu:20.04

LABEL: Adds metadata to the image, like maintainer information.
Example: LABEL maintainer="name@example.com"

RUN: Executes commands in a new layer on top of the current image and commits the result.
Example: RUN apt-get update && apt-get install -y nginx

CMD: Provides defaults for the executing container. There can only be one CMD instruction in a Dockerfile.
Example: CMD ["nginx", "-g", "daemon off;"]

ENTRYPOINT: Configures the container to run as an executable. It's often used in combination with CMD.
Example: ENTRYPOINT ["nginx"]

COPY: Copies files or directories from the host machine to the container.
Example: COPY ./webapp /var/www/webapp

ADD: Similar to COPY, but can also handle URLs and tarball extraction.
Example: ADD https://example.com/app.tar.gz /app/

WORKDIR: Sets the working directory for any subsequent RUN, CMD, ENTRYPOINT, COPY, and ADD instructions.
Example: WORKDIR /app

EXPOSE: Informs Docker that the container listens on the specified network port at runtime.
Example: EXPOSE 80

ENV: Sets environment variables.
Example: ENV MY_VARIABLE=value

VOLUME: Creates a mount point for external storage or other containers.
Example: VOLUME /data

Let's create a Dockerfile for a basic web server using Nginx:

First, create a folder called my-webserver and go inside it cd my-webserver

Then create another folder inside that called website and a file called index.html within the folder website with any content of your choice.

Create a file dockerfile with the following content within the my-webserver folder.

```
# Use the official Nginx image as a base
FROM nginx:latest

# Set the maintainer label
LABEL maintainer="name@example.com"

# Copy static website files to the Nginx web directory
COPY ./website /usr/share/nginx/html

# Expose port 80 for the web server
EXPOSE 80

# Default command to run Nginx in the foreground
CMD ["nginx", "-g", "daemon off;"]
```

Building an Image from a Dockerfile

To build a Docker image from your Dockerfile, navigate to the directory containing the Dockerfile and run:

```
docker build -t my-webserver:latest .
```

This command tells Docker to build an image using the Dockerfile in the current directory (.) and tag it as my-webserver:latest.

Best Practices

- Minimize Layers: Try to reduce the number of layers in your image to make it lightweight. For instance, chain commands using && in a single RUN instruction.
- Use .dockerignore: Just like .gitignore, you can use .dockerignore to exclude files that aren't needed in the container.
- Avoid Installing Unnecessary Packages: Only install the packages that are necessary to run your application.
- Clean Up: Remove temporary files and caches to reduce image size.

Part 3:

What is Docker Compose?

Docker Compose is a tool for defining and running multi-container Docker applications. With Compose, you can define a multi-container application in a single file, then spin up your application with a single command (`docker-compose up`).

Key Concepts

Services: Each container started by Docker Compose is a service. Services are defined in the docker-compose.yml file.

Networks: By default, Docker Compose sets up a single network for your application. Each container for a service joins the default network and is discoverable via a hostname identical to the container name.

Volumes: Volumes can be used to share files between the host and container or between containers.

Basic docker-compose Commands

- `docker-compose up`: Starts up the services defined in the docker-compose.yml file.
- `docker-compose down`: Stops and removes all the containers defined in the docker-compose.yml file.
- `docker-compose ps`: Lists the services and their current state (running/stopped).
- `docker-compose logs`: Shows the logs from the services.

Deploying WordPress with Docker Compose

Let's deploy a WordPress application using two containers: one for WordPress and another for the MySQL database.

Create a docker-compose.yml file:

```
version: '3'

services:
  # Database Service
  db:
    image: mysql:5.7
    volumes:
      - db_data:/var/lib/mysql
    environment:
      MYSQL_ROOT_PASSWORD: somewordpress
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress

  # WordPress Service
  wordpress:
    depends_on:
      - db
    image: wordpress:latest
    ports:
      - "8080:80"
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress
      WORDPRESS_DB_NAME: wordpress
    volumes:
      - wordpress_data:/var/www/html

volumes:
    db_data: {}
    wordpress_data: {}
```

Start the WordPress and Database Containers: Navigate to the directory containing the docker-compose.yml file and run:

```
docker-compose up -d
```

This command will start the services in detached mode. Once the services are up, you can access the WordPress site by navigating to http://<Floating_IP>:8080 from your browser.

Stopping the Services: To stop the services, navigate to the same directory and run:

```
docker-compose down
```

Best Practices

- Explicit Service Names: Give your services explicit names to make it clear what each service does.
- Environment Variables: Use environment variables for sensitive information and configurations.
- Service Dependencies: Use the depends_on option to ensure services start in the correct order.

Part 4:

Deploy any web app as per your wish and showcase its usage of it. You need to use more than one docker container eg: you can use three containers, one to run a web app and the others to run a database and other data storage respectively. You may use the docker hub to get any existing containers. What we evaluate is your ability to deploy the containers and bringing up a working web app.