Docker (codecademy)

Docker is an open platform container technology that uses the host operating system's kernel, which is isolated from the application. So, your applications will run in a sandbox. This allows Docker to really accelerate your development time because everything that your application needs to run is contained in your image, and these images are lightweight and they're portable, so you can share these images, send them from one development team to another. Docker images are also very fast to deploy and fast to build.

You can use Docker to run microservices in a microservice architecture. Docker containers are lightweight. They have everything that you need for the application. So any dependencies are already going to be there, so during the build process you don't have to go and download. And because of that, they're shareable.

Docker has a product called **Docker Hub** which is a container image library, and you can create and manage users and grant access to all your repositories that will hold your Docker images and then you can just go out, pull that image down from Docker Hub and incorporate it into your application or deploy it as you need.

With Docker, you develop your application and all the components using your containers. You can then deploy your application, you can deploy it to a

development environment and since they're so lightweight, you can deploy them at ease. When you're ready, you deploy that into production as a container or as an orchestrated service.

Docker Uses:

- Rapid Application Delivery
- Continuous Integration
- Continuous Deployment
- Faster to get fixes out to the end users
- Run multiple workloads on one physical machine
- Good for development, deployment and scaling

You can run containers on a local machine, on physical machines, or on virtual machines and it's very portable and lightweight, so we can scale up and scale down our applications very quickly with containers and we can really handle our traffic flows in real-time.

Docker Workflow:

1. Write & test the application:

Two ways to write and test the applications:

▼ Steps to Develop using the usual way:

1. If you prefer to develop the usual way and then Integrate that and deploy that out into a Docker image using a Dockerfile.
2. If you were going to develop your usual way and your local machine, you can use what they call a bind mount and that's a Docker feature.
3. And then that makes it available to sync your files from your local machine and any changes are going to be reflected in real-time.

▼ Steps to Develop using a Docker container:

1. Start a container and leave it running.
2. Edit your source code and every time you make changes, these changes will appear in the container.

2. Build a container image:

Steps to build a container image:

▼ Build an artifact

1. Build an artifact and this artifact is your container image or your Docker image.
2. We're going to deploy that out to our server.

▼ Write a Dockerfile

1. Dockerfile is a simple text-based file that has all the commands a user could call on the command line to assemble an image.
2. It's basically just build instructions. You can do things like pull your base image, install PHP extensions, set environment variables.
3. Build all that up, create an image from that, so then everything is self-contained.

▼ Use Tools

1. There's tools like Buildpacks, Source-to-Image, Jib, and other tools that allow you to build your containers.

3. Push images to a server:

- Allows us to visualize stored images as layers
- Use docker push to move images
- If you want to move an image from your library, you can use docker push command
- You can also move from libraries to an image library, so you probably already have a registry
- You can push images to Docker Hub, from GitHub, or from your local image registry in the cloud provider

4. Start the application

1. Use **Podman** or a **System D** to start your containers automatically.
2. Use **docker-compose file** for your application. So, you can run **docker-compose up** and that'll run your application.

Feature	Dockerfile	docker-compose.yml
Primary Purpose	**Building** a single Docker image.	**Running** and orchestrating one or more containers.
Scope	Focuses on a single image/environment.	Manages an entire application stack (multiple services).
File Format	Plain text script (commands like `FROM`, `RUN`, `COPY`).	YAML file (defining `services`, `networks`, `volumes`).
Main Command	`docker build .`	`docker compose up .`

Starting Docker Containers:

- To start a docker container we make use of **docker run command**. This command is used to **build and run your containers**.
- Every single Docker run command is going to create a new container or separate resources. And executes all the commands that are in the Docker file.
- If we add attributes to our basic docker run syntax, we can configure a container, we can have it run in detached mode, we can set a container name, and so on.
- When we're running the docker run command. So, the docker run syntax at its very basic is docker run.

Docker run Syntax - docker run [OPTIONS] IMAGE[:TAG|@DIGEST] [COMMAND] {ARG...}

- The docker run syntax at its very basic is **docker run**.
- You can add options, you can specify your image, an image tag, and a digest, any commands, and arguments.

▼ **OPTIONS** - OPTIONS operator allows us to have exclusive options and only the person executing this docker run command can set these up.

- **-d for detached** because you can run a container in detached mode. If you start it in detached mode, containers exit when the root process used to run the container exits.
- **-rm,** and if you use that with d, the container is also removed when it exits.

▼ **IMAGE** - With a **TAG or DIGEST**

- Images using version two or later have a content addressable identifier called DIGEST.
- As long as the input user generate that images on change, the digest value will be unchanged as well. So, you can specify a tag like the image with a tag or an image with a digest.

▼ **COMMAND** - Place to run your commands.

- These are for additional commands, not for the docker run command.
- And most of the commands are already present in the Docker file, so no need to specify additional commands again here.