Docker

Docker simplifies the management of runtime environment of applications, to ensure the run consistently and efficiently regardless of compatibility (typical DevOps practices).

One of the benefits is that it uses less resource intensive (uses less RAM…).

* Docker containers : containers are standardized unit that allows grouping application code and all its dependencies and configurations.
* Dockerfile : A Dockerfile is a script that contains a series of commands used to build a Docker image. When you build a Docker image, you start with a base image (typically an operating system or a base image with some common dependencies) and then use the Dockerfile to specify how to set up and configure the environment for your application.

The Dockerfile typically includes instructions for things like:

1. \*\*Setting a base image:\*\* You specify the starting point for your image, such as a specific Linux distribution or a base image with certain libraries.

2. \*\*Installing dependencies:\*\* You can use commands to install software packages or libraries that your application needs.

3. \*\*Copying files:\*\* You can copy your application code and other necessary files into the image.

4. \*\*Configuring environment variables:\*\* You can set environment variables that your application will use.

5. \*\*Running commands:\*\* You can specify the commands that should be executed when the container is started, like starting your application.

6. \*\*Exposing ports:\*\* If your application listens on specific network ports, you can specify which ports should be exposed.

Once you have written the Dockerfile, you use the `docker build` command to build the Docker image. This command reads the Dockerfile, executes each of the specified instructions, and generates an image. This image can then be used to create and run Docker containers.

* We can use the same Docker image to run multiple containers. That's one of the key benefits of Docker's containerization technology. Each container is an instance of the same image, but they are isolated from each other and can run independently.

Here's how it works:

1. \*\*Image as a Blueprint:\*\* A Docker image is like a blueprint for a container. It contains all the necessary files, libraries, and configurations to run your application.

2. \*\*Container Instances:\*\* When you run a Docker image, it creates a container instance. You can create multiple container instances from the same image.

3. \*\*Isolation:\*\* Each container is isolated from the others and behaves as if it has its own environment. They don't interfere with each other, and changes made in one container do not affect others.

4. \*\*Resource Sharing:\*\* Containers created from the same image share the image's underlying file system and resources (unless you specify custom resource constraints).

This ability to run multiple containers from the same image is particularly useful in scenarios like:

- \*\*Scaling:\*\* You can run multiple instances of your application to handle increased load. Each instance is a container created from the same image.

- \*\*Development and Testing:\*\* Developers can work on different parts of an application by running separate containers from the same image. This ensures consistency in the development and testing environments.

- \*\*Microservices:\*\* In a microservices architecture, different services are often implemented as separate containers from the same image, allowing for easy management and scaling of individual services.

- \*\*Continuous Integration and Deployment (CI/CD):\*\* CI/CD pipelines can use the same image to create containers for testing and deployment, ensuring that what was tested is exactly what gets deployed.

* Docker registry : A Docker registry is a centralized repository for storing and distributing Docker images. It serves as a storage and distribution system for Docker images, allowing users to easily share, access, and manage container images.

Here are some key points about Docker registries:

1. \*\*Storage:\*\* Docker images can be quite large, and Docker registries provide a secure and efficient way to store these images. They act as a central hub where you can keep your images in a well-organized and easily accessible manner.

2. \*\*Distribution:\*\* Docker registries make it easy to distribute images to different environments. You can push images to a registry, and then other users or systems can pull those images from the registry to run containers on their own machines or servers.

3. \*\*Security:\*\* Docker registries often support authentication and access control, which means you can control who can push images to the registry and who can access them. This is important for securing your container images, especially in production environments.

4. \*\*Public and Private Registries:\*\* There are both public and private Docker registries. Public registries, like Docker Hub, allow anyone to access and use images. Private registries, often set up by organizations, are used to store images that are meant for internal use only.

5. \*\*Custom Registries:\*\* You can also set up your own custom Docker registry if you want to have complete control over your image distribution. The open-source Docker Registry project can be used to set up a private registry.

6. \*\*Image Versioning:\*\* Docker registries allow you to version your images, making it possible to access specific versions of an image. This is crucial for maintaining consistency and reproducibility in your deployments.

7. \*\*Proxying and Caching:\*\* Docker registries can be set up to proxy and cache images from other registries. This is useful for faster image pulls and to reduce the load on public registries like Docker Hub.

Link for docker explanation: <https://www.youtube.com/watch?v=J0NuOlA2xDc>