Containers and Application Development

An Introduction to Docker and Python

Bob Rotsted

Agenda

- Why is Docker useful for deploying applications?
- What is a Dockerfile and what are the most important keywords
- How does docker build work? What is the role of a registry?
- Running a docker container on your laptop vs using orchestration

How do containers help?

- Docker containers encapsulate an application and its dependencies in a self-contained unit.
- They ensure the application runs consistently across different Linux environments, from development through production, thus eliminating the "it works on my machine" problem.

Docker and Python

Simplify Deployment and Repeatability

- Dependency management in Python can become complex, especially for larger projects.
- The intricacies of managing package versions, resolving dependency conflicts, and ensuring that all dependencies are compatible with the project's Python version can be daunting.
- Containers make dependency management and application deployment repeatable

Dockerfile Example

```
1 # Use the official Python 3.8 image as a base
2 FROM python:3.8
3
4 # Set the working directory in the container
5 WORKDIR /usr/src/app
6
7 # Copy the current directory contents into the container
8 COPY . .
9
10 # Command to run on container start
11 CMD [ "python", "./app.py" ]
```

Dockerfile

FROM keyword

- In the context of a Dockerfile, the FROM instruction specifies the base image from which you are building your Docker image.
- Docker images are built in layers. The base image specified by FROM contributes the initial layers, and each subsequent instruction in the Dockerfile adds new layers on top of it.
- This layering mechanism allows for efficient storage and rapid sharing of common layers between images.

Dockerfile

WORKDIR keyword

- In a Dockerfile, the WORKDIR instruction sets the working directory for any RUN, CMD, ENTRYPOINT, COPY, and ADD instructions that follow it in the Dockerfile.

- It's a way to specify the directory in which commands should be executed. If the specified directory doesn't exist, Docker will create it for you, even if it's not used in any subsequent Dockerfile instruction.

Dockerfile

ENV keyword

- In the context of a Dockerfile, the ENV keyword is used to set environment variables.
- Environment variables are commonly used to pass configuration values to a Docker container at runtime. They allow you to customize the behavior of the container without modifying the application or the Dockerfile directly.

Dockerfile

COPY keyword

- In the context of a Dockerfile, the COPY keyword is used to copy files and directories from the host machine into the filesystem of the image being built.
- This instruction is crucial for including application source code, configuration files, or any other necessary files into the Docker image.

Dockerfile

CMD keyword

- In the context of a Dockerfile, the CMD (command) keyword specifies the default command to be executed when a container is started from the Docker image.
- It is used to define the default behavior of the container, such as running an application or service contained within the image.
- This default command can be overridden by specifying a different command when launching the container with docker run.
- For instance, if a Docker image is built with a CMD instruction to run a specific application, but you want to start the container with an interactive shell instead, you can do so by running “docker run —it myimage /bin/bash”

Buildtime

docker build …

- Refers to the phase when a Docker image is being created from a Dockerfile.
- During this phase, Docker processes the instructions in the Dockerfile (such as RUN, COPY, and ADD) to construct the image layer by layer.
- Environment variables can be set with the ENV instruction and will be embedded into the image, affecting how software is installed and configured within the image.
- Build-time is when you install dependencies, compile code (if necessary), and configure the software that will be part of the Docker image.

Runtime

docker run …

- Begins once a Docker container is started from an image using commands like docker run.
- During run-time, the application within the container is executed, and the container interacts with its environment, data, and other containers or services.
- You can pass environment variables at run-time using the -e or --env flag with docker run, which can override some of the settings defined at build-time or provide additional configuration necessary for the application's execution.
- Run-time is when the containerized applications are actually running, serving requests, processing data, etc.

Docker Registries

- Docker registries are services that store and distribute Docker images. Users can push images to a registry for storage and pull images from the registry to deploy containers on any system.
- Docker Hub is the default public registry for Docker images, offering a vast collection of images from community and official sources. Organizations often use private registries to securely host proprietary images for internal use.
- Registries enable version control of images through tagging, allowing developers to release and track different versions of an application. They also facilitate sharing of images, making it easier for teams to collaborate on development and deployment.

Running a Docker Container

Local Management vs. Cluster Orchestration

- **On Your Laptop**: Running Docker containers directly on your laptop involves manual management using Docker CLI commands. You're responsible for starting, stopping, and managing the containers and their configurations individually.

- **In Kubernetes**: Kubernetes orchestrates containers at scale across a cluster of machines. It manages the deployment, scaling, and networking of containers automatically according to defined configurations (e.g., YAML files).

Running a Docker Container

Single Host vs. Multi-Host Deployment

- **On Your Laptop:** Containers run on a single host, your laptop. This setup is suitable for development, testing, or small-scale production environments where high availability and scalability are not critical concerns.

- **In Kubernetes:** Containers are deployed across multiple nodes in a cluster, offering high availability, load balancing, and scalability. Kubernetes ensures that applications can handle increased load or failures in the cluster by adjusting the number of running containers as needed.

Running a Docker Container

Manual Scaling vs. Automated Scaling and Self-healing

- **On Your Laptop:** Scaling involves manually starting more containers or stopping them as needed. There is no built-in mechanism for automatically adjusting the number of running containers based on demand or for replacing failed containers.

- **In Kubernetes:** Kubernetes supports automated scaling based on metrics like CPU usage or custom metrics. It also provides self-healing by automatically replacing containers that fail, are unresponsive, or don't meet the user-defined health criteria.

Discussion / Questions