Week 1 – Container basics

What is a container?

Containers provide a standard way to package your application's code, configurations, and dependencies into a single object. Containers share an operating system that's installed on the server. They run as resource-isolated processes, and are designed to provide quick, reliable, and consistent deployments—regardless of environment. For more information about containers and their benefits, see Containers Deep Dive.

Namespaces and cgroups

With containers, you can isolate processes from each other, which is a key component of how containers work. For more information about how namespaces and control groups (cgroups) work with containers, see What are Namespaces and cgroups, and how do they work? on the NGINX blog.

OCI

In this course, we mostly talk about using Docker containers. However, in 2015, Docker and other industry leaders came together to form the Open Container Initiative (OCI). The purpose of the OCI is to create standards for container formats and runtimes. For more information about the OCI, see About the Open Container Initiative.

Benefits of containers

- Optimization of resource utilization: You can fit multiple, lightweight containers on a single virtual machine, and increase the efficiency and density of your resources.
- Automation: The standard packaging and interaction with containers can make it easier to automate software development lifecycle tasks, such as building, deploying, and scaling applications.
- Speed: Containers are quick to run. You can start, scale, or delete containers in seconds.
- Scalability: Because containers facilitate microservices architectures, you can meet demand by using containers to scale out horizontally both quickly and efficiently.
- Increased developer productivity:
- Developers can spend less time on operations and environment debugging, and spend more time on application development.

- Code portability: Having confidence that your workload will behave consistently across environments is one of the major benefits of containers.

What is the Docker daemon?

The Docker daemon runs on a host and listens for API calls to manage Docker objects. For example, if you want to build an image, run a container, or create a volume, you can use the Docker command line interface (CLI) to create the request. The request will then be submitted to the Docker daemon to be managed.

What is the Docker CLI?

The Docker command line interface (CLI) is the client interface for interacting with Docker objects. The Docker CLI is how you issue commands to Docker to build your containers, run your containers, or stop your containers.

What is a Dockerfile?

A Dockerfile is a text document that contains instructions on how to build your container image. A Dockerfile includes instructions for what base layer to use (for example, whether you want to use a minimal version of Linux or use an image that has preinstalled software, such as a database engine). Other instructions include running commands in the container, or copying your application data, dependencies, and configurations into the container. For more information, see Best Practices for writing Dockerfiles.

What is a Docker image?

A container image is created when you run the build command from the Docker CLI and it follows the instructions in a Dockerfile. A container image is made up of a series of read-only layers, with one writable layer on top where files can be added, modified, or deleted. Each layer is created from an instruction in the Dockerfile.

What is an image registry?

An image registry is a repository where you can store container images. You can store the images either publicly or privately. From the repository, images can be pulled and deployed, or used by other developers.