# **Docker**

Terminology

An **image** is a lightweight, stand-alone, executable package that includes everything needed to run a piece of software.

A **container** is a runtime instance of an image. It runs completely isolated from the host environment by default, only accessing host files and ports if configured to do so.

**Services** are really just “containers in production.” A service only runs one image, but it codifies the way that image runs—what ports it should use, how many replicas of the container should run so the service has the capacity it needs, and so on. Scaling a service changes the number of container instances running that piece of software, assigning more computing resources to the service in the process.

Set up

<https://docs.docker.com/get-started/#setup>

Dockerfile

Dockerfile will define what goes on in the environment inside your container. Access to resources like networking interfaces and disk drives is virtualized inside this environment, which is isolated from the rest of your system, so you have to map ports to the outside world, and be specific about what files you want to “copy in” to that environment.

docker-compose.yml

A docker-compose.yml file is a YAML file that defines how Docker containers should behave in production

ex. from the set up

https://docs.docker.com/get-started/part3/#your-first-docker-composeyml-file

Develop with docker engine SDKs and API

<https://docs.docker.com/develop/sdk/> a list of unofficial libraries can be found (including erldocker)

Docker api for Python

<https://docs.docker.com/apidocs/docker-cloud/?python#start-a-container>

erldocker for using the api in erlang

<https://github.com/ElaineQvarnstrom/erldocker> (forked since the dependencies were inaccurate)

Scale a service in sequential mode

<https://docs.docker.com/docker-cloud/apps/service-scaling/#sequential-deployment-and-scaling> necessary to use api for scaling in production