Deploying to the cloud

To run your containers in the cloud with either Azure or AWS, check out our docs on getting started with cloud deployments.

* [Deploying with Docker and AWS](https://docs.docker.com/engine/context/ecs-integration/)
* [Deploying with Docker and Azure](https://docs.docker.com/engine/context/aci-integration/)

Sample Dockerfile

Writing a Dockerfile is the first step to containerizing an application. You can think of these Dockerfile commands as a step-by-step recipe on how to build up your image. The Dockerfile in the bulletin board app looks like this:

# Use the official image as a parent image.

FROM node:current-slim

# Set the working directory.

WORKDIR /usr/src/app

# Copy the file from your host to your current location.

COPY package.json .

# Run the command inside your image filesystem.

RUN npm install

# Add metadata to the image to describe which port the container is listening on at runtime.

EXPOSE 8080

# Run the specified command within the container.

CMD [ "npm", "start" ]

# Copy the rest of your app's source code from your host to your image filesystem.

COPY . .

The dockerfile defined in this example takes the following steps:

* Start FROM the pre-existing node:current-slim image. This is an *official image*, built by the node.js vendors and validated by Docker to be a high-quality image containing the Node.js Long Term Support (LTS) interpreter and basic dependencies.
* Use WORKDIR to specify that all subsequent actions should be taken from the directory /usr/src/app *in your image filesystem* (never the host’s filesystem).
* COPY the file package.json from your host to the present location (.) in your image (so in this case, to /usr/src/app/package.json)
* RUN the command npm install inside your image filesystem (which will read package.json to determine your app’s node dependencies, and install them)
* COPY in the rest of your app’s source code from your host to your image filesystem.

You can see that these are much the same steps you might have taken to set up and install your app on your host. However, capturing these as a Dockerfile allows you to do the same thing inside a portable, isolated Docker image.

The steps above built up the filesystem of our image, but there are other lines in your Dockerfile.

The CMD directive is the first example of specifying some metadata in your image that describes how to run a container based on this image. In this case, it’s saying that the containerized process that this image is meant to support is npm start.

The EXPOSE 8080 informs Docker that the container is listening on port 8080 at runtime.

What you see above is a good way to organize a simple Dockerfile; always start with a FROM command, follow it with the steps to build up your private filesystem, and conclude with any metadata specifications. There are many more Dockerfile directives than just the few you see above. For a complete list, see the [Dockerfile reference](https://docs.docker.com/engine/reference/builder/).

CLI references

Further documentation for all CLI commands used in this article are available here:

* [docker image](https://docs.docker.com/engine/reference/commandline/image/)
* [docker container](https://docs.docker.com/engine/reference/commandline/container/)
* [Dockerfile reference](https://docs.docker.com/engine/reference/builder/)