In this scenario, we'll look at how you can optimise Dockerfile using the OnBuild instruction.

The environment has been configured with an example Node.js application however the approaches can be applied to any image. The machine name Docker is running on is called docker. If you want to access any of the services, then use docker instead of localhost or 0.0.0.0.

**Step 1 - OnBuild**
While Dockerfile's are executed in order from top to bottom, you can trigger an instruction to be executed at a later time when the image is used as the base for another image.

The result is you can delay your execution to be dependent on the application which you're building, for example the application's package.json file.

Below is the Node.js OnBuild Dockerfile. Unlike in our previous scenario the application specify commands have been prefixed with ONBUILD.

```
FROM node:7
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app
ONBUILD COPY package.json /usr/src/app/
ONBUILD RUN npm install
ONBUILD COPY . /usr/src/app
CMD [ "npm", "start" ]
```

The result is that we can build this image but the application specific commands won't be executed until the built image is used as a base image. They'll then be executed as part of the base image's build.

We'll see how this base image looks in the next step.

**Step 2 - Application Dockerfile**
With all of the logic to copy the code, install our dependencies and launch our application the only aspect which needs to be defined on the application level is which port(s) to expose.

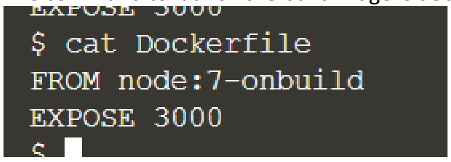
The advantage of creating OnBuild images is that our Dockerfile is now much simpler and can be easily re-used across multiple projects without having to re-run the same steps improving build times.

This will be created in the environment for you. The steps to build and launch the Dockerfile are covered in the next step.

**Step 3 - Building & Launching Container**
The Dockerfile from the previous step has been created for you. Building the images based on the OnBuild docker file is the same as before. The OnBuild commands will be executed as if they were in the base Dockerfile.

Example: Build & Launch
The command to build the image is docker build -t my-nodejs-app .

The command to launch the built image is docker run -d --name my-running-app -p 3000:3000 my-nodejs-app



```
EXPOSE 3000
$ cat Dockerfile
FROM node:7-onbuild
EXPOSE 3000
$
```

Testing Container
You can test the container is accessible using curl. If the application responds when you know that everything has correctly started.

curl http://docker:3000

In this step we explored how to use the OnBuild instruction to optimise your Dockerfile's to increase re-use between different applications by moving common parts into a base image.