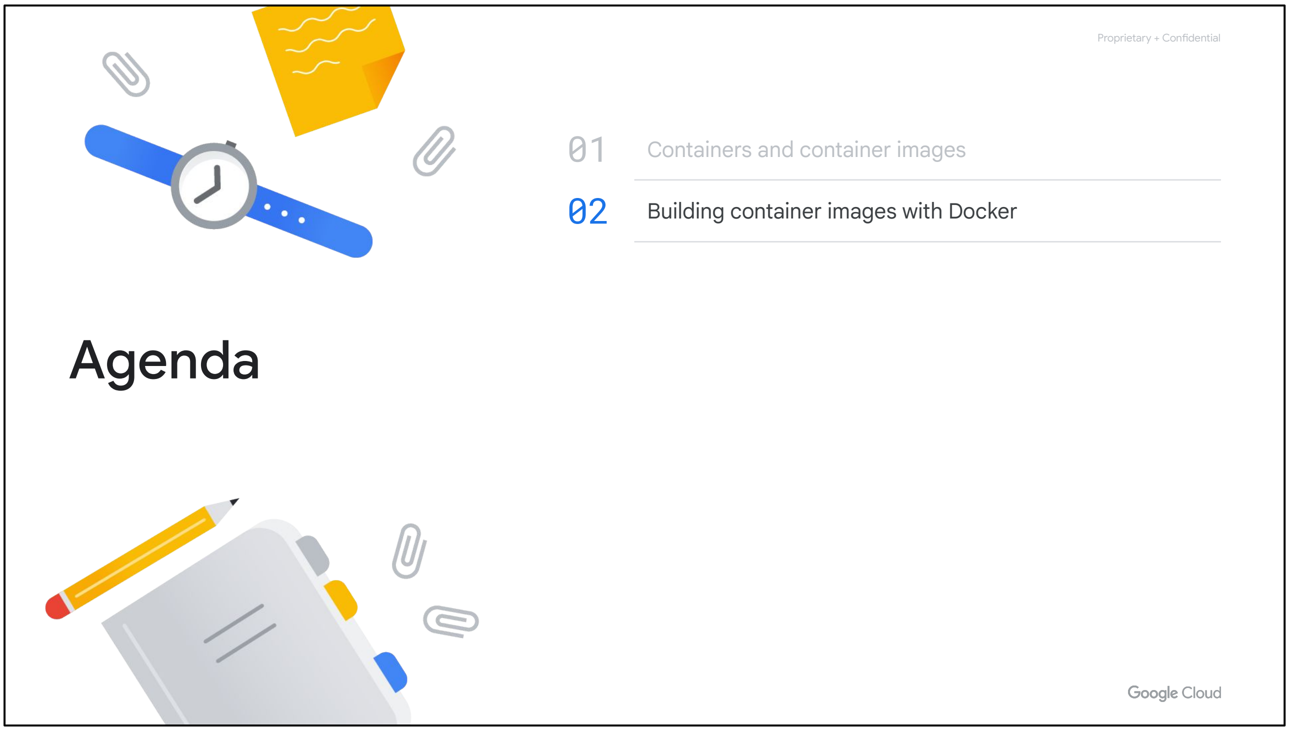
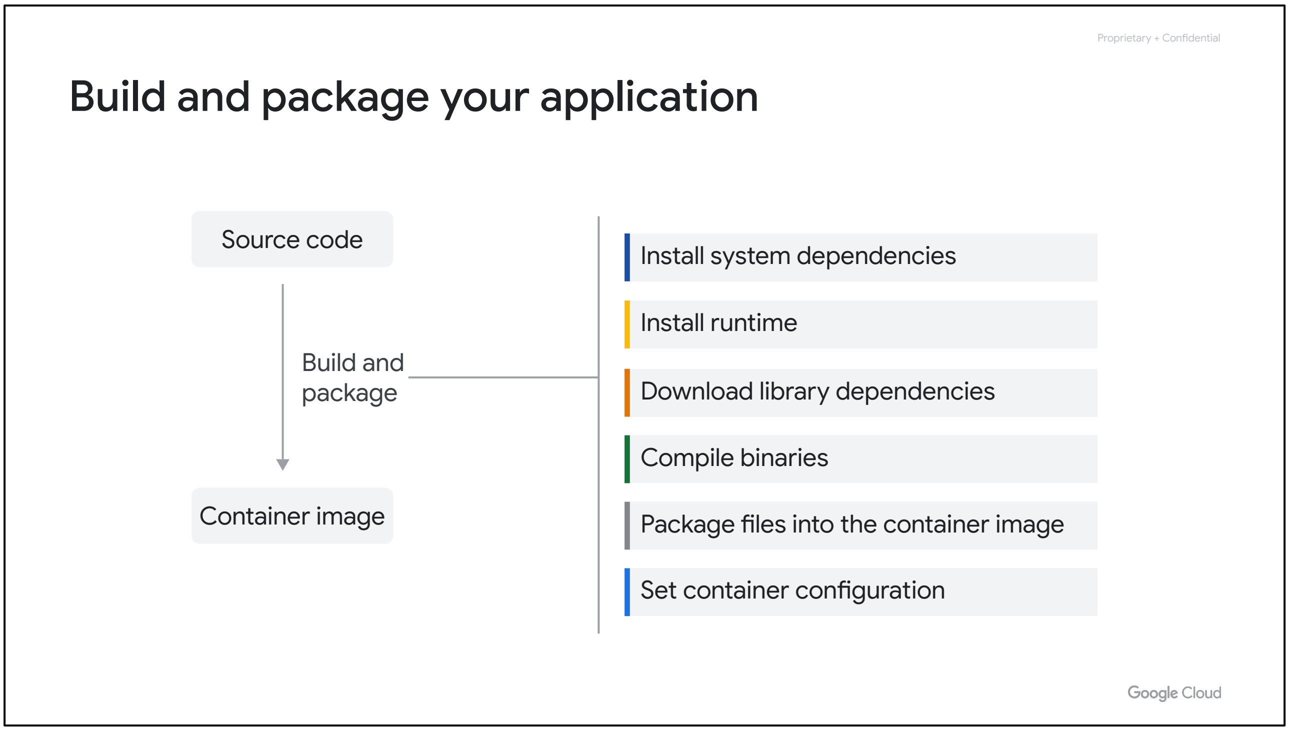


Proprietary + Confidential

Agenda

Google Cloud

Now that you have an understanding of what a container image is, let’s discuss how to build and package an application into a container image.

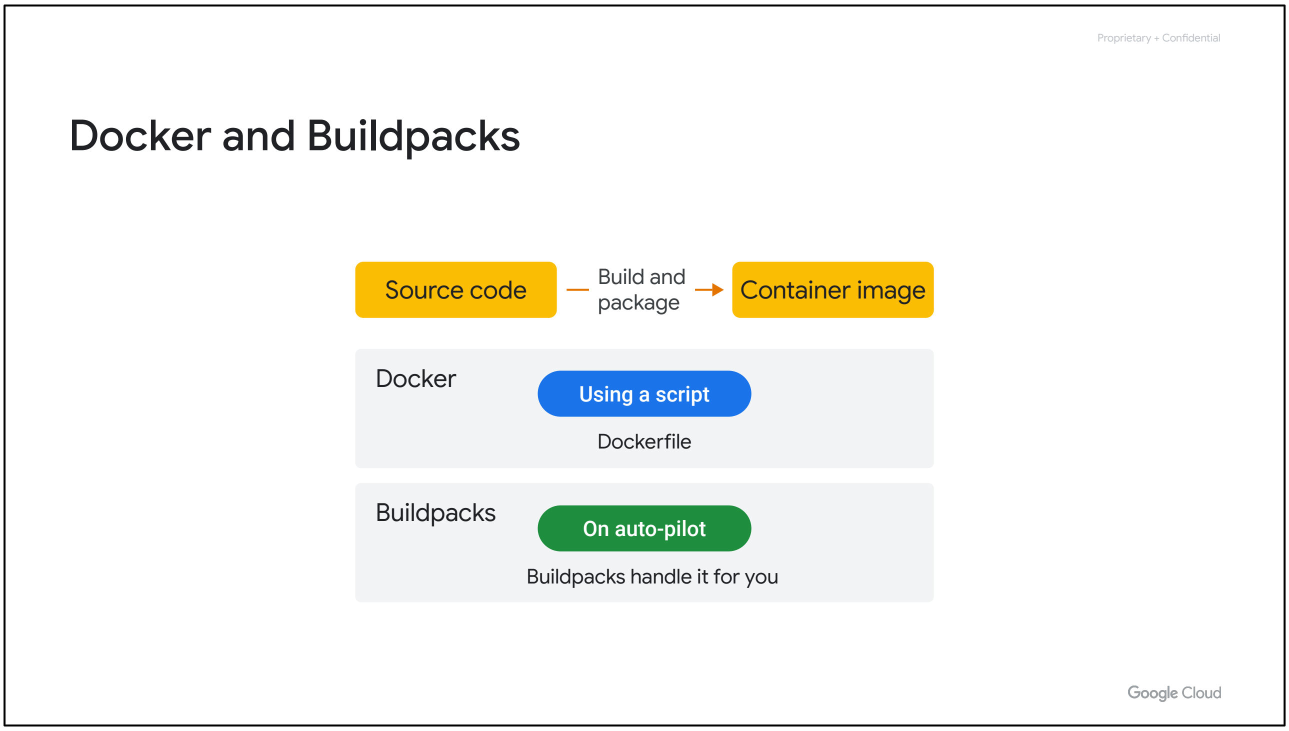


Build and package your application

Source code
Build and package
Container image
Install system dependencies
Install runtime
Download library dependencies
Compile binaries
Package files into the container image
Set container configuration

To build and package your application into a container image, perform these steps:

- Install any system dependencies (if your application depends on them).
- Install a runtime (for example Node.js or Python).
- Download your application's dependencies (npm install, go get, pip install, or invoking your package manager of choice).
- Compile the binaries (or process / bundle the source code).
- Package the files into the image.
- Set the container configuration.



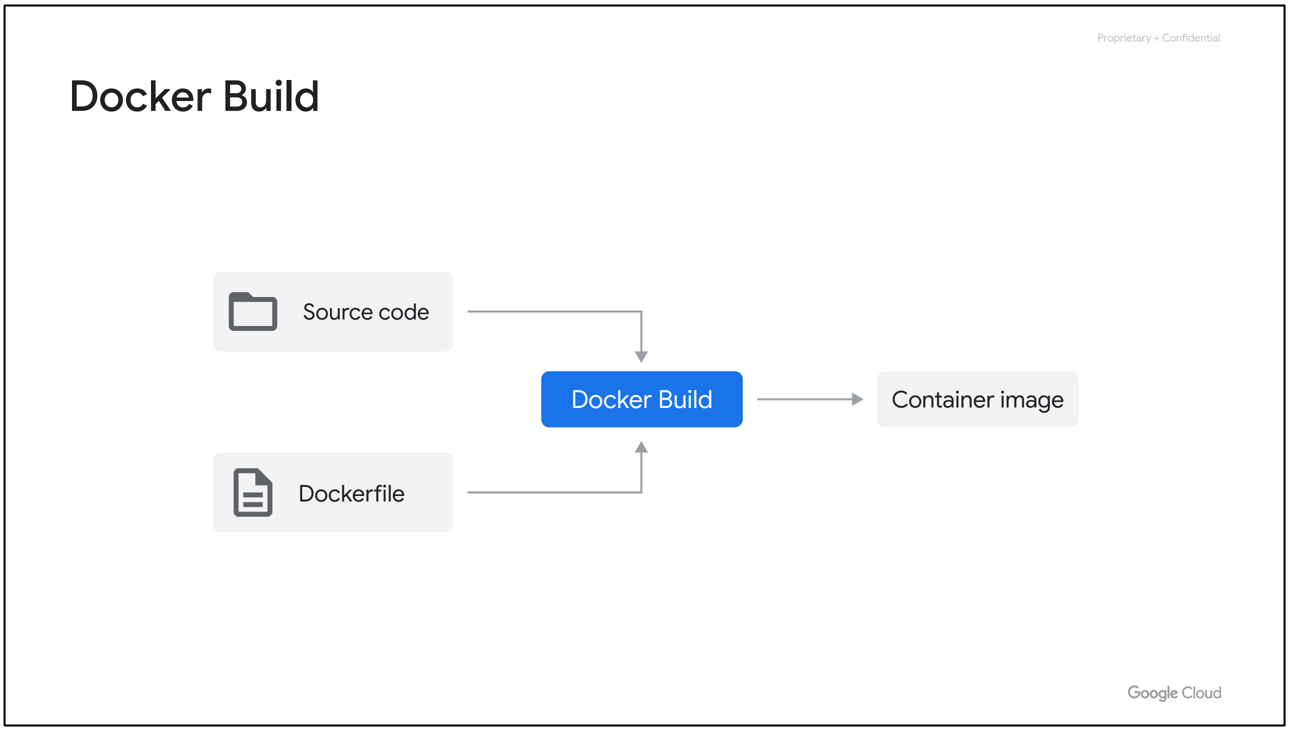
Docker and Buildpacks

Docker is an open platform that enables you to package and run applications in containers. It provides the tools to manage the lifecycle of your containers, from development and packaging to deployment.

Docker lets you express the application build process using a script, called a Dockerfile. Dockerfiles provide a low-level approach that offers flexibility at the cost of complexity.

The Dockerfile is a manifest that details how to turn your source code into a container image.

Buildpacks are another approach for building container images. They are different from Docker, and provide a convenient approach to building container images by using heuristics to build and package the source code. We also discuss how to use buildpacks in this module.

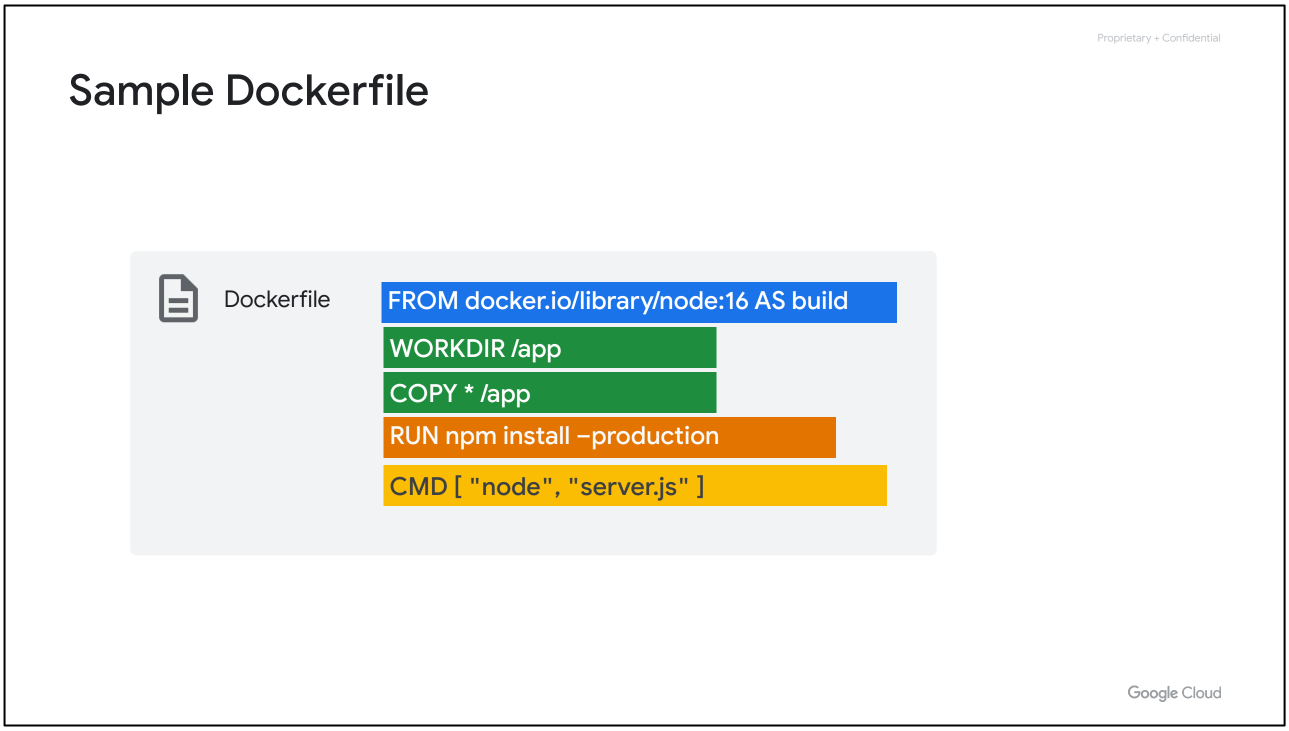


Docker Build

Let’s first dive into Docker. Docker is a container engine—you can use it to run containers on your local machine. You can also use it to build container images.

Docker Build is a set of features and tools in Docker that enable you to build and package your applications into container images.

Docker Build takes your source code and a Dockerfile. You express the building and packaging of your source code using a set of instructions in the Dockerfile.



Sample Dockerfile

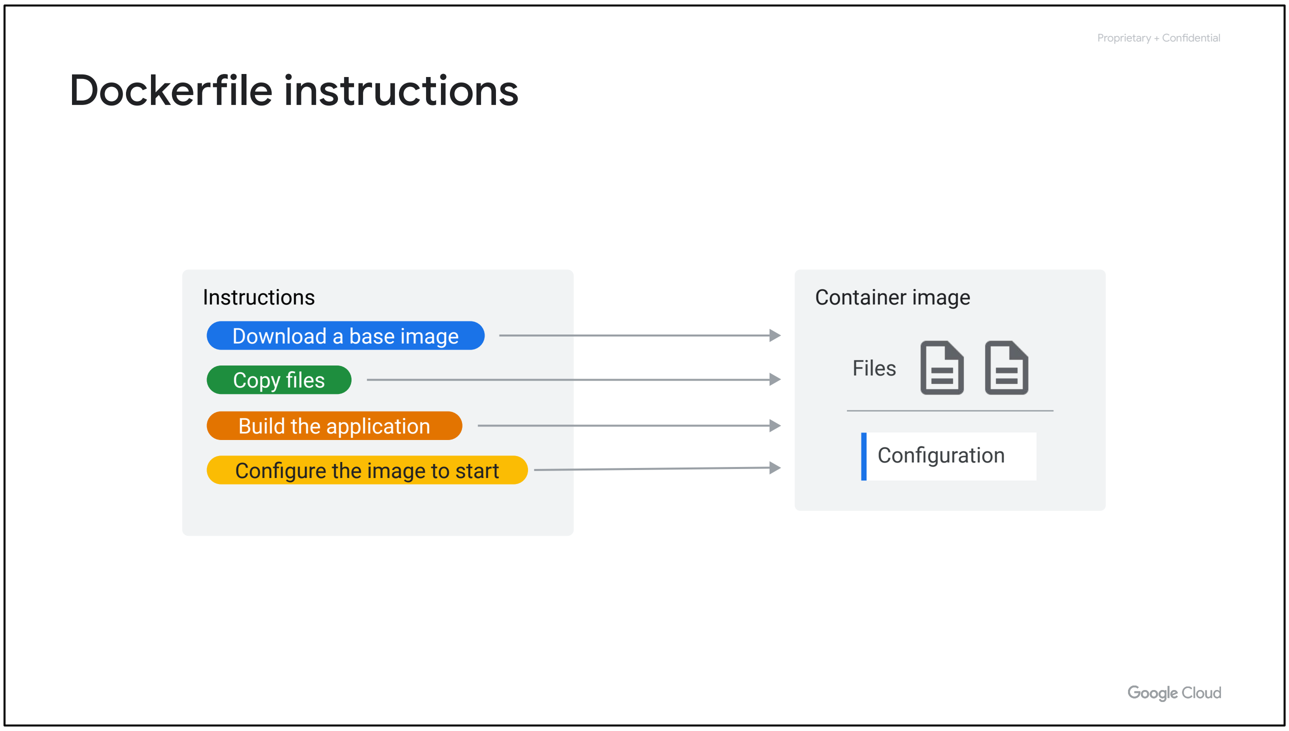
Dockerfile

```
FROM docker.io/library/node:16 AS build
WORKDIR /app
COPY * /app
RUN npm install –production
CMD [ "node", "server.js" ]
```

Here’s an example of a Dockerfile that builds a sample Node.js application into a container image.

The instructions in the Dockerfile:

- Starts from a Node.js base image.
- Creates the application directory in the container file system.
- Copies the source code and other files to the container image.
- Installs the application dependencies excluding any devDependencies listed in the package.json file.
- Sets configuration to run the application when it starts. (node server.js)



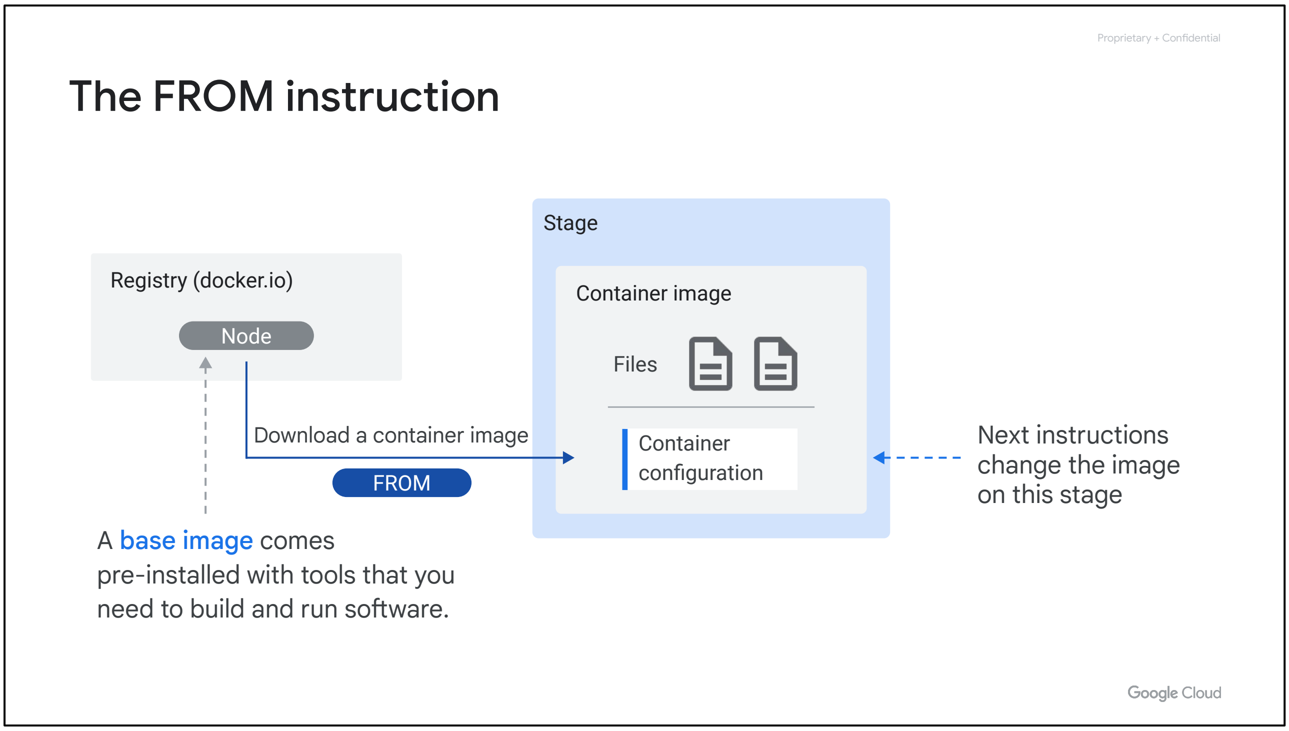
Dockerfile instructions

To understand how this works, it's important to realize that with Docker you build your application *inside* the container image.

You start with putting a container image on a stage, and every Dockerfile instruction changes that staged container image.

The general process is:
- Start with a base image, which contains tooling to build your application.
- Pull your source code and other required files into the container image.
- To build your application, run a program to update files in the image.
- Configure the image to start your application.

Dockerfiles combine the building and packaging of a container image into a single process.


Proprietary + Confidential

The FROM instruction

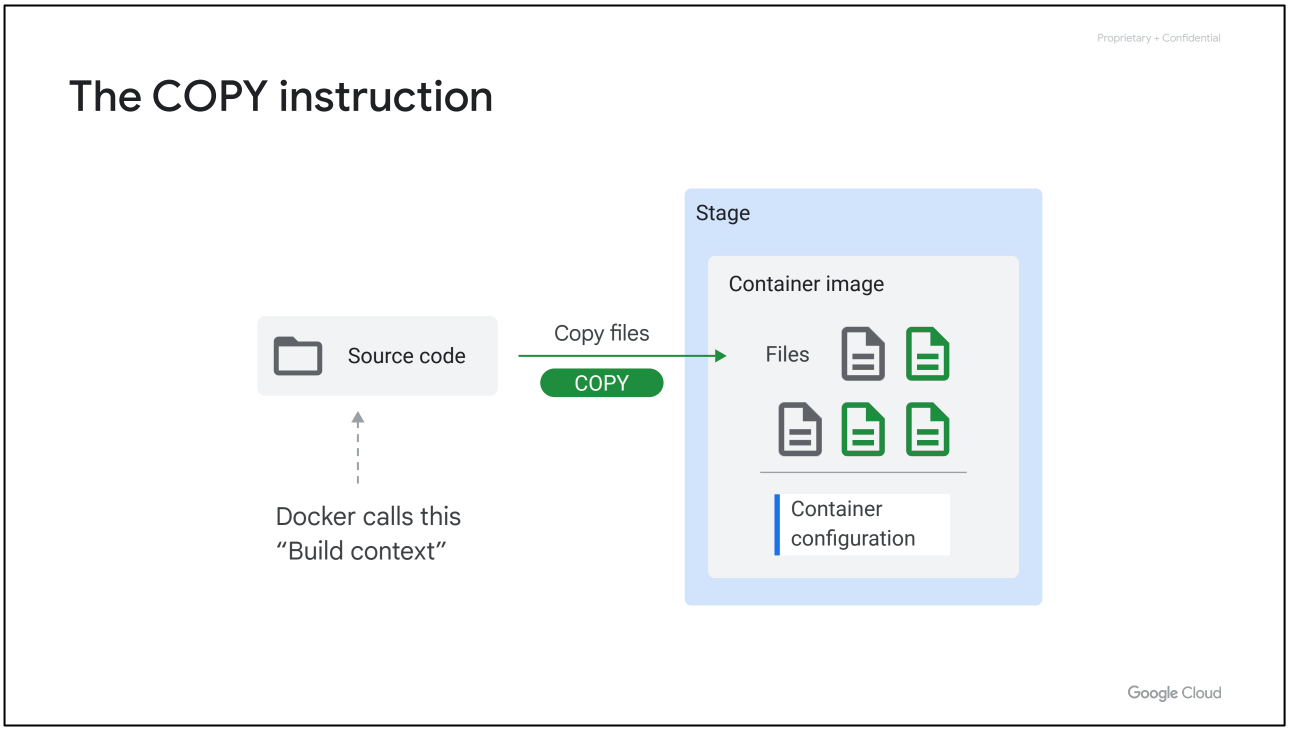
A base image comes pre-installed with tools that you need to build and run software.

Google Cloud

The FROM instruction downloads a base image from a registry and puts that on the stage, to be modified by subsequent instructions.

Examples of base images are:

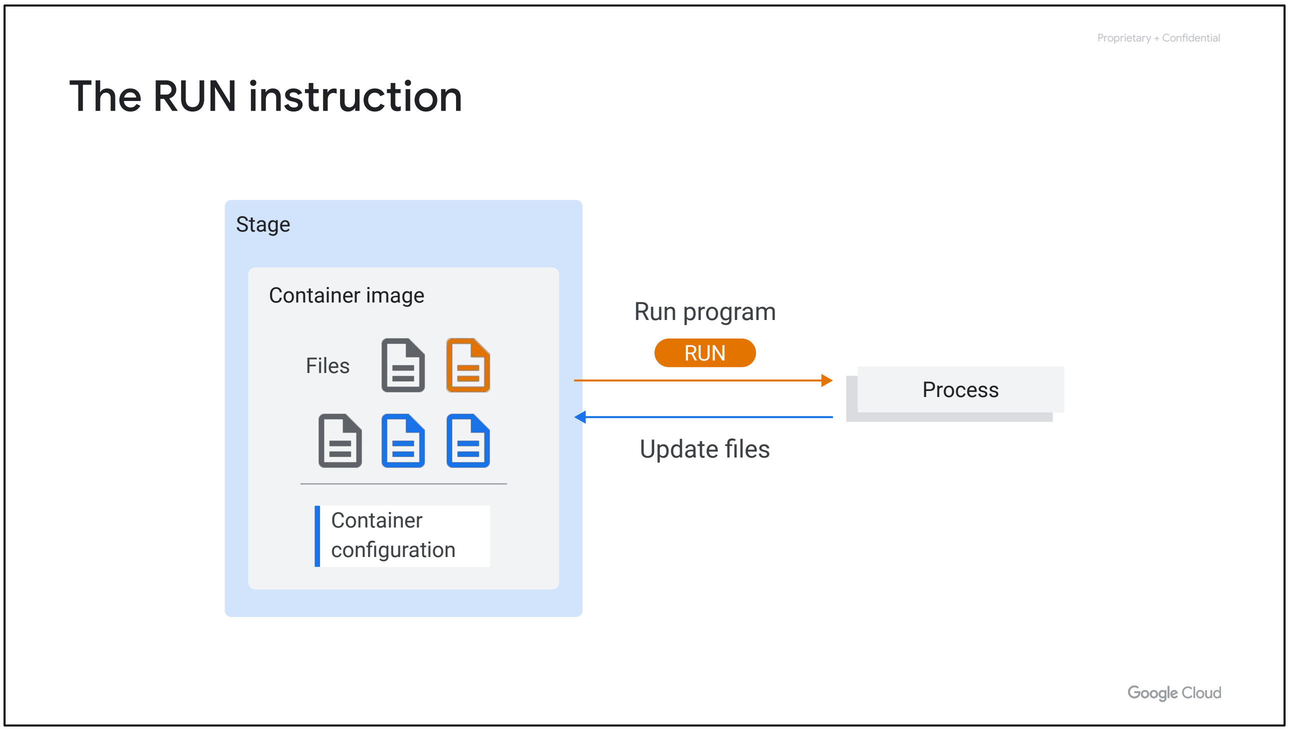
- golang (it has tools to build *go* programs)
- nodejs (it has tools to install and build node programs)



The COPY instruction

The COPY instruction pulls in source code. Docker has the concept of a “build context,” which is the set of files in the source code directory.

Use it to bring source code into the staged image that you’ve just downloaded with the FROM instruction.



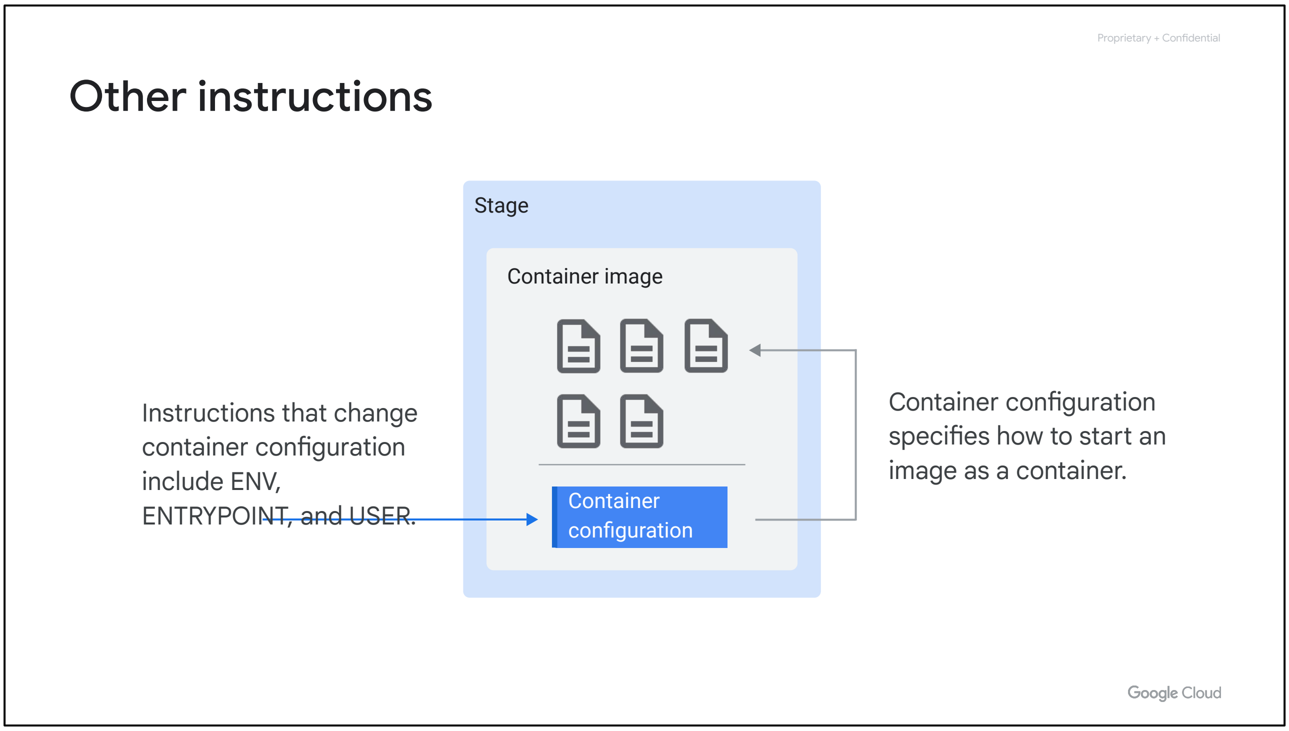
The RUN instruction

The RUN instruction lets you run a program **from** the image, **on** the image. This means:

- The program file that you execute needs to be present in the container image.
- The only files accessible to the program are those that exist on the container image.

Examples of tasks that RUN is used for, include:

- Installing another system package that you need to build your application.
- Downloading library dependencies.
- Compiling your source code into binaries.



Other instructions

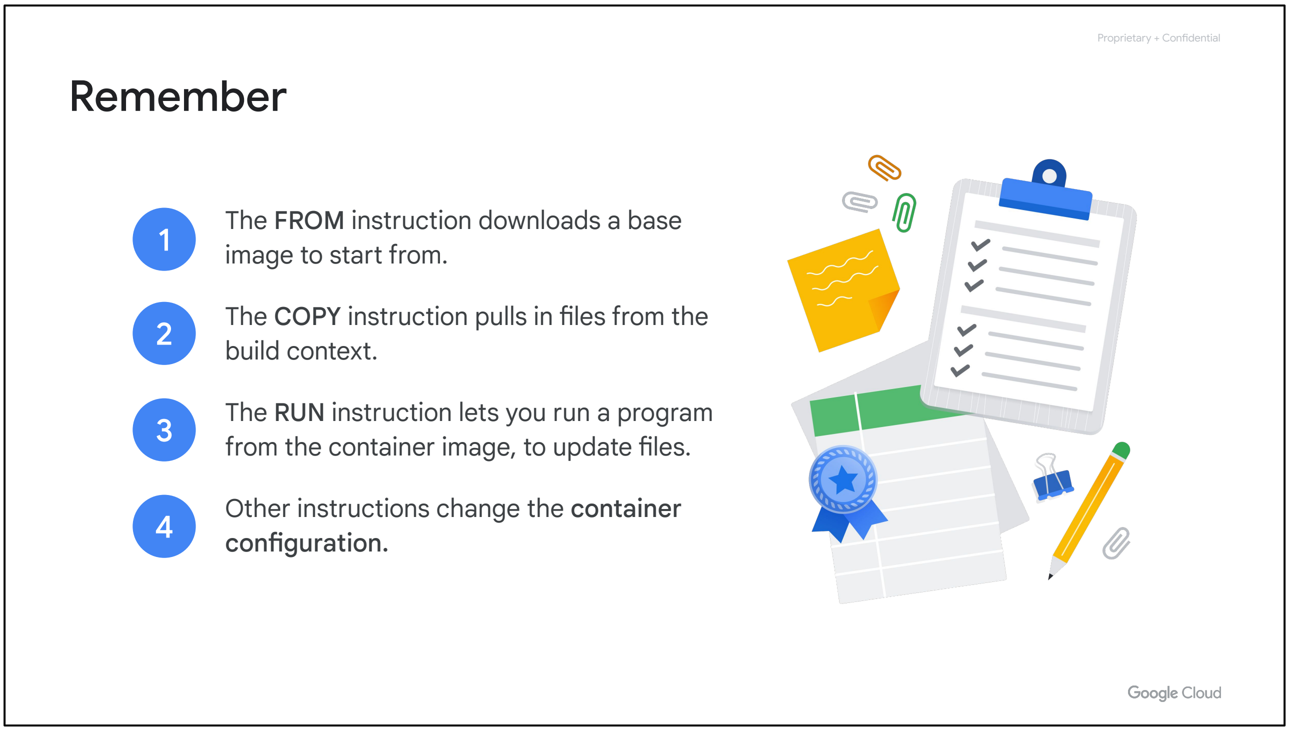
Stage
Container image
Container configuration
Instructions that change container configuration include ENV, ENTRYPOINT, and USER.
Container configuration specifies how to start an image as a container.

Finally, *container configuration* tells the container runtime (such as Docker, or Cloud Run), what program file to start from the container image, and with what parameters.

There are several instructions that can change the container configuration. Examples include:

ENTRYPOINT: points to the program file to start and run the container as an executable.
CMD: provides defaults for an executing container, that includes the command to run when the container is started. If the executable command is not specified, then the ENTRYPOINT instruction is required.
ENV: is used to set environment variables.
WORKDIR: sets the working directory of the program.
USER: sets the user to use when starting the program.

A full reference of all Dockerfile instructions can be found here:
https://docs.docker.com/engine/reference/builder/



Remember

1. The **FROM** instruction downloads a base image to start from.
2. The **COPY** instruction pulls in files from the build context.
3. The **RUN** instruction lets you run a program from the container image, to update files.
4. Other instructions change the **container configuration.**

Here's what's important to remember about Dockerfile instructions:

You start with putting a container image on a stage, and every Dockerfile instruction changes that staged container image.

- The **FROM** instruction downloads a base image to start from. A base image can be 'golang' for example, and includes tools you need to build and run your software.
- The **COPY** instruction pulls in files from the build context, which is usually the directory that contains the Dockerfile.
- The **RUN** instruction lets you run a program **from** the container image, to change files **in the** image.
- Other instructions change the container configuration, which points out which program file to start, and how.