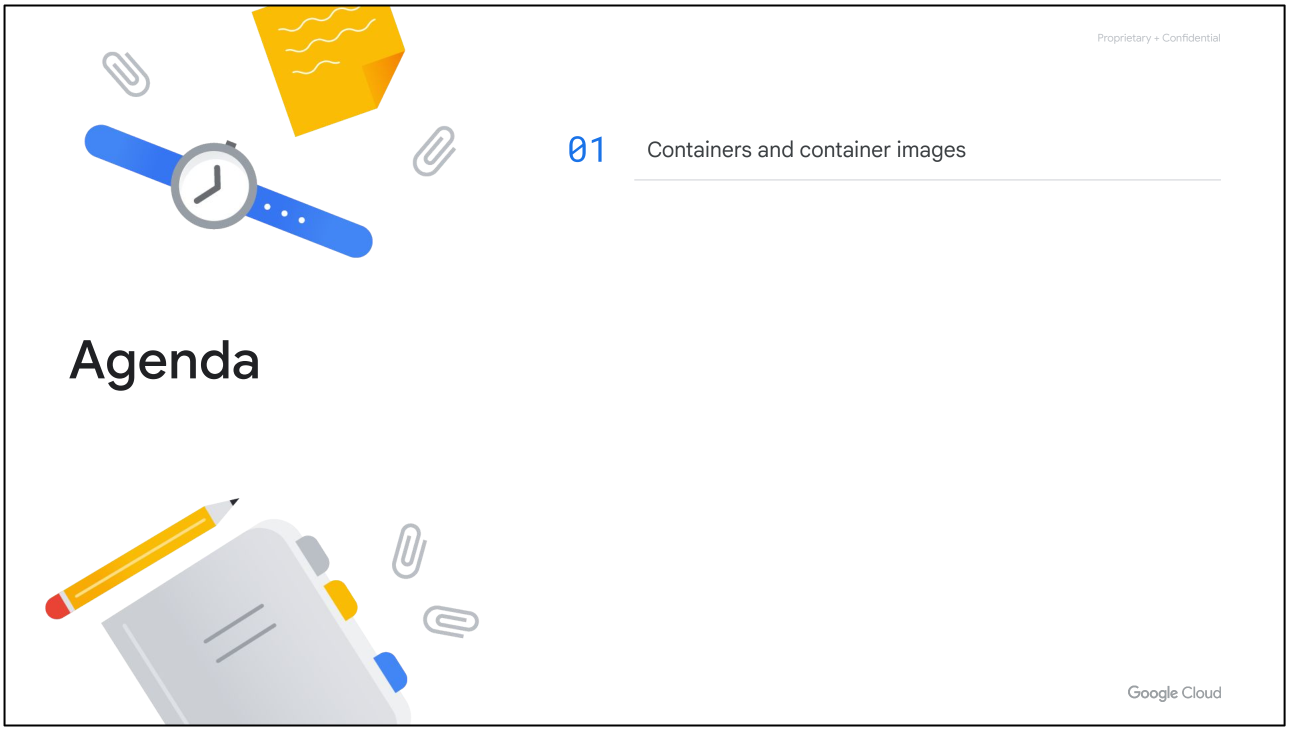
Google Cloud

Introduction to Containers

In this module, we introduce Containers and container images, discuss some of their features, and how to build and run them.

You also do a couple of labs in which you build containers with Docker and Google Cloud's buildpacks.

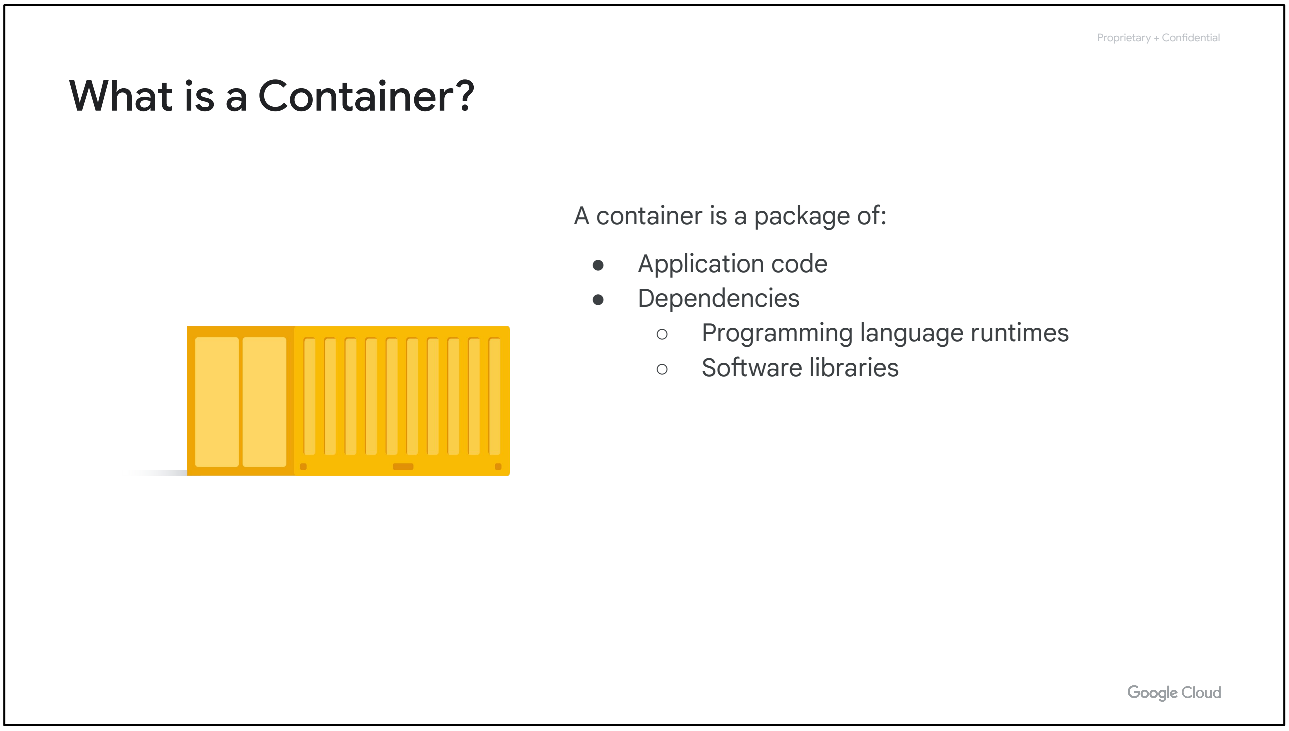


Agenda

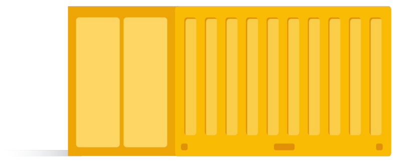
01 Containers and container images

Google Cloud

Let’s first discuss what are containers and container images.



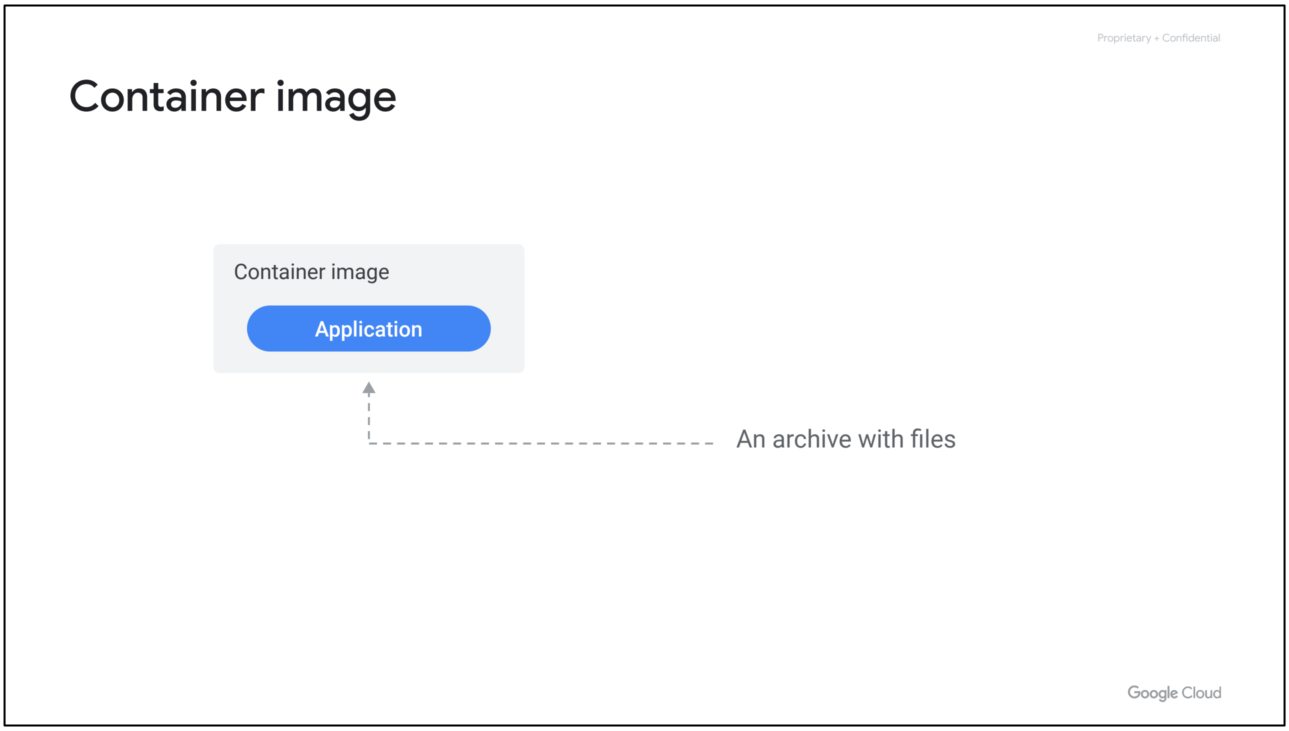
What is a Container?



A container is a package of:

- Application code
- Dependencies
  - Programming language runtimes
  - Software libraries

A container is a package of your application code together with dependencies such as specific versions of programming language runtimes and libraries that are required to run your software services.



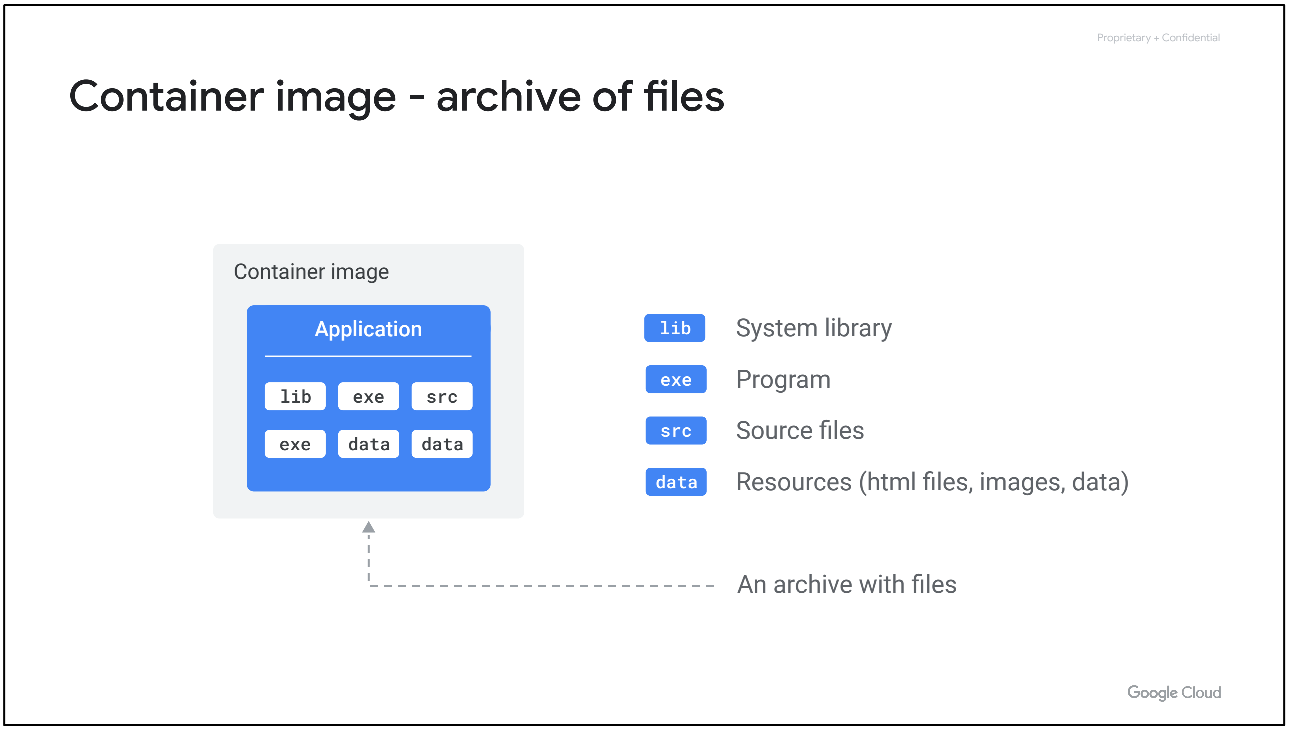
Container image

Container image
Application
An archive with files

A container image represents a template that defines how a container instance will be realized at runtime.

A container image packages your application along with everything your application needs to run.

For example, in a container image for a Java application, your application is packaged together with the appropriate Java Virtual Machine.



Container image - archive of files

lib	System library
exe	Program
src	Source files
data	Resources (html files, images, data)

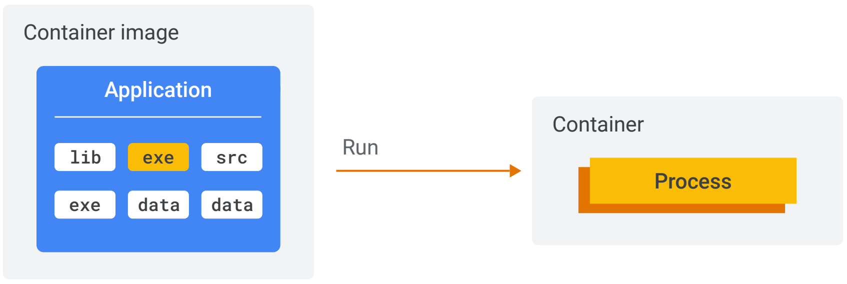
Practically, a container image is an archive with files that contains everything your application needs to run.

A container image includes system libraries, executable programs, and resources, such as (but not limited to) html files, images, binary blobs, and your application source files.

A container image can contain programs written in any programming language, for example Java, Python, JavaScript, PHP, or Go. It can include any binary dependency that you need.

This packaging turns your code and resources into something that you can store, download, and send somewhere else.

Container - runtime instance

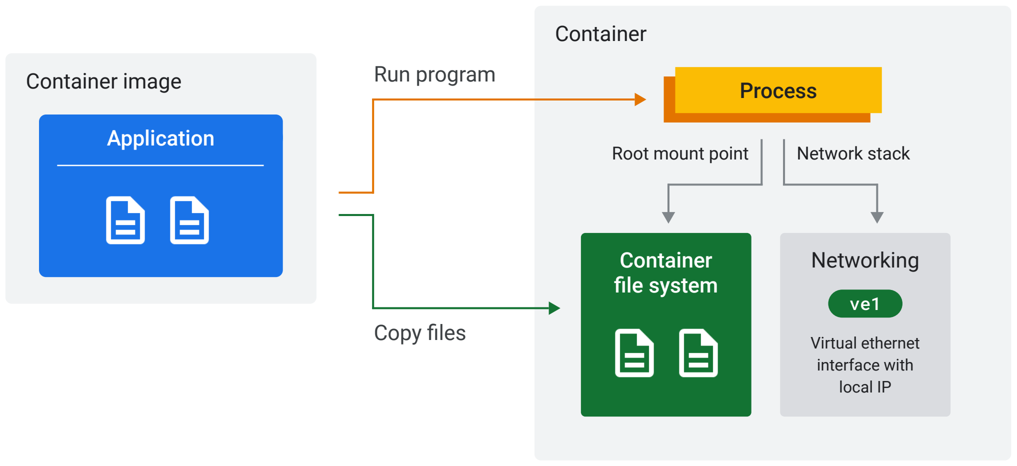


When you run a container image, you execute one of the programs inside the container image.

For example, if your container image contains a Java application, the program you execute is the Java Virtual Machine.

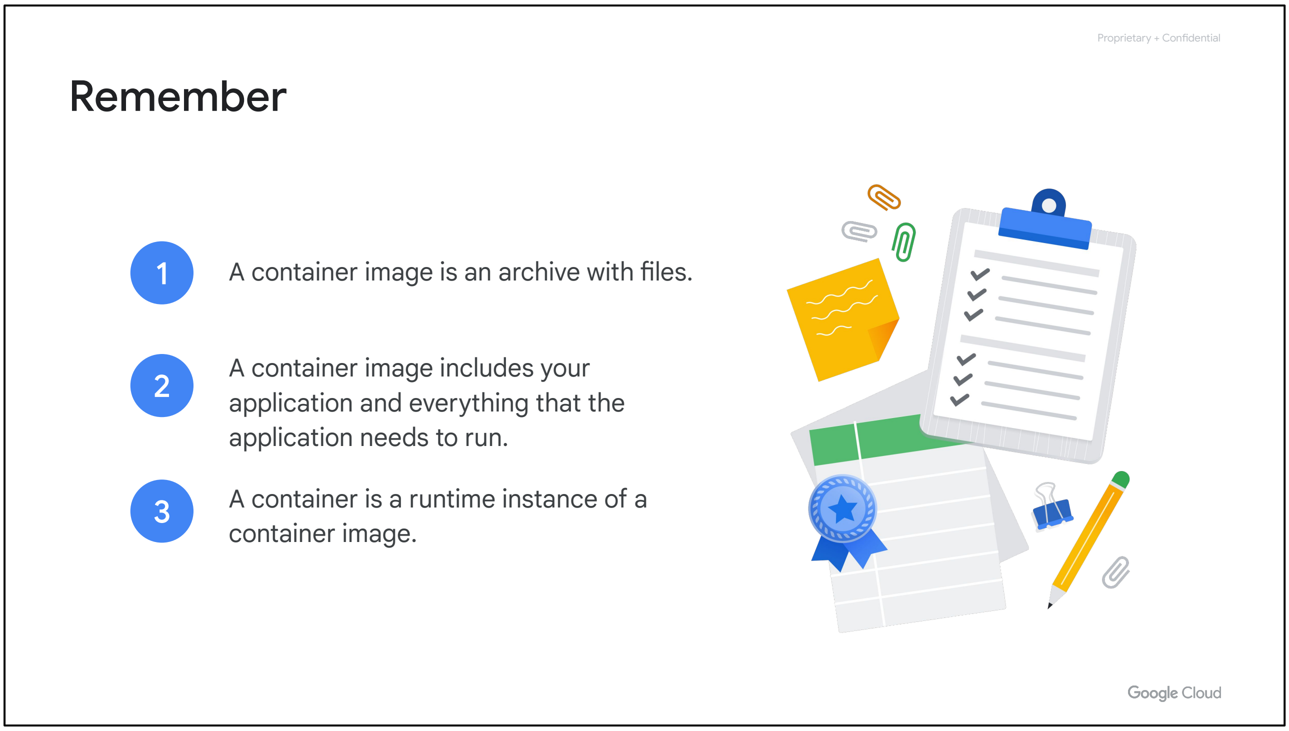
A container represents the running processes of your application and only exists at runtime. If there are no running processes, there is no container.

Container - file system and network stack



Two other relevant things happen when a container runs:

- The contents of the container image are used to seed a private file system for the container. (All files that the processes of your application see)
- The processes in the container have access to a *virtual network interface* with a local IP, so your application can bind to this interface and start listening on a port for incoming traffic.



Proprietary + Confidential

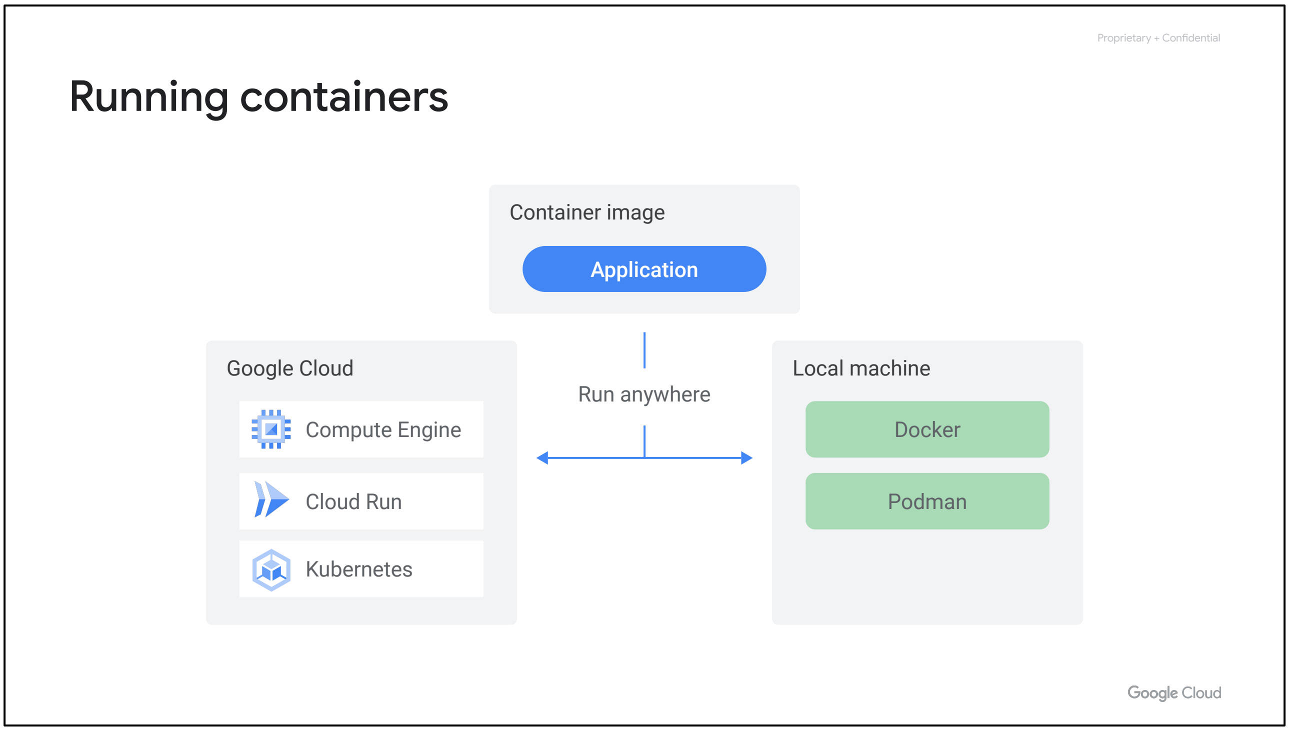
Remember

1. A container image is an archive with files.
2. A container image includes your application and everything that the application needs to run.
3. A container is a runtime instance of a container image.

Google Cloud

Here are a few things to remember:

- A **container image** is an archive with files: it includes executables, system libraries, data files, and more.
- A container image is **self-contained.** Everything that your application needs to run is inside the container image. If your application is a Node.js application, for example, your source files are in the image along with the Node.js runtime.
- A **container** is a runtime instance of a container image and represents the running processes of your container.



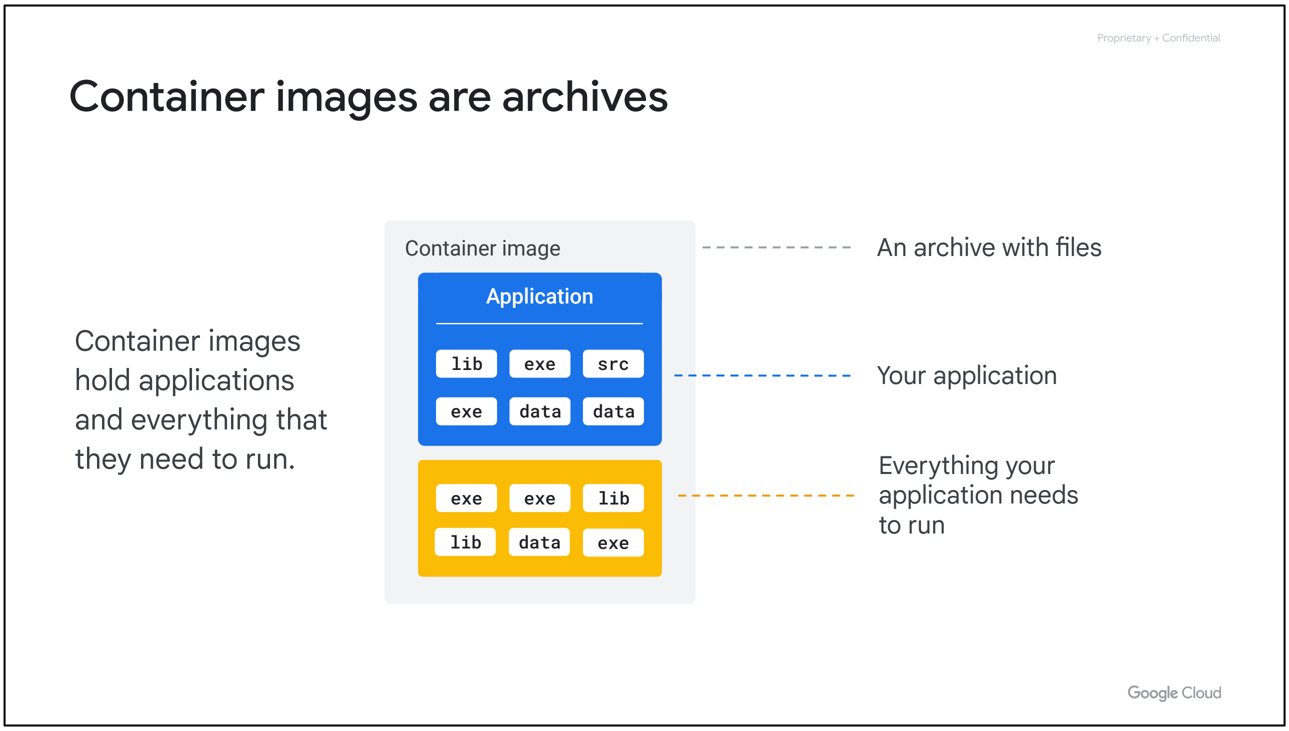
Running containers

After your application is packaged into a container image, you can run it anywhere. On Google Cloud, you can run containers on a Compute Engine in a virtual machine, or on a Kubernetes cluster, or on Cloud Run. On your local machine, you can use the Docker or Podman container runtimes.

Docker is an open platform for developing, and running container-based applications.

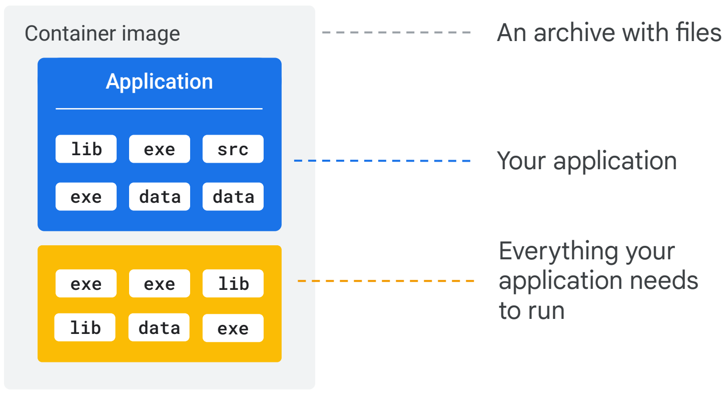
Podman is an open source Linux tool that is used to build, and run applications using containers that are based on the Open Containers Initiative (OCI).

We discuss Cloud Run in more detail later in this course.



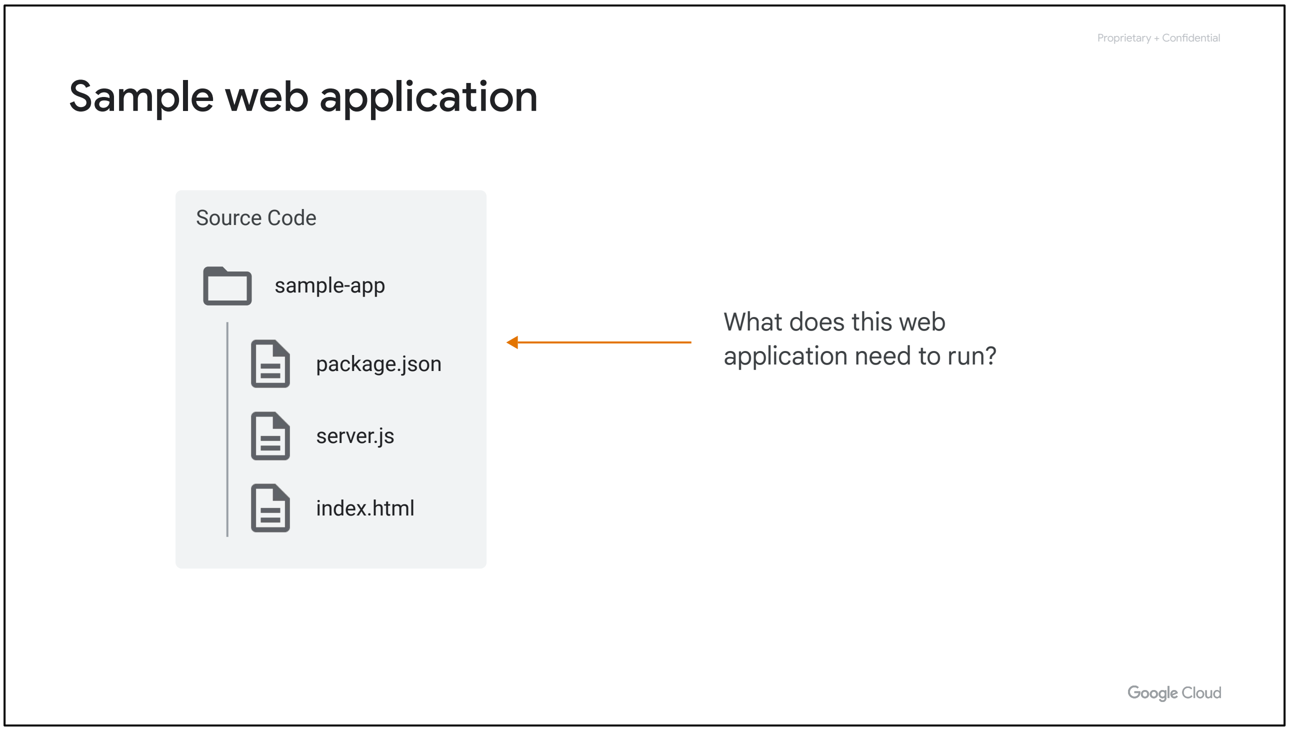
Container images are archives

Container images hold applications and everything that they need to run.



As mentioned previously, a container image is an archive with files that includes your application and everything it needs to run.

To discover the different types of files that you might find in a container image, let's look at a very simple example application.

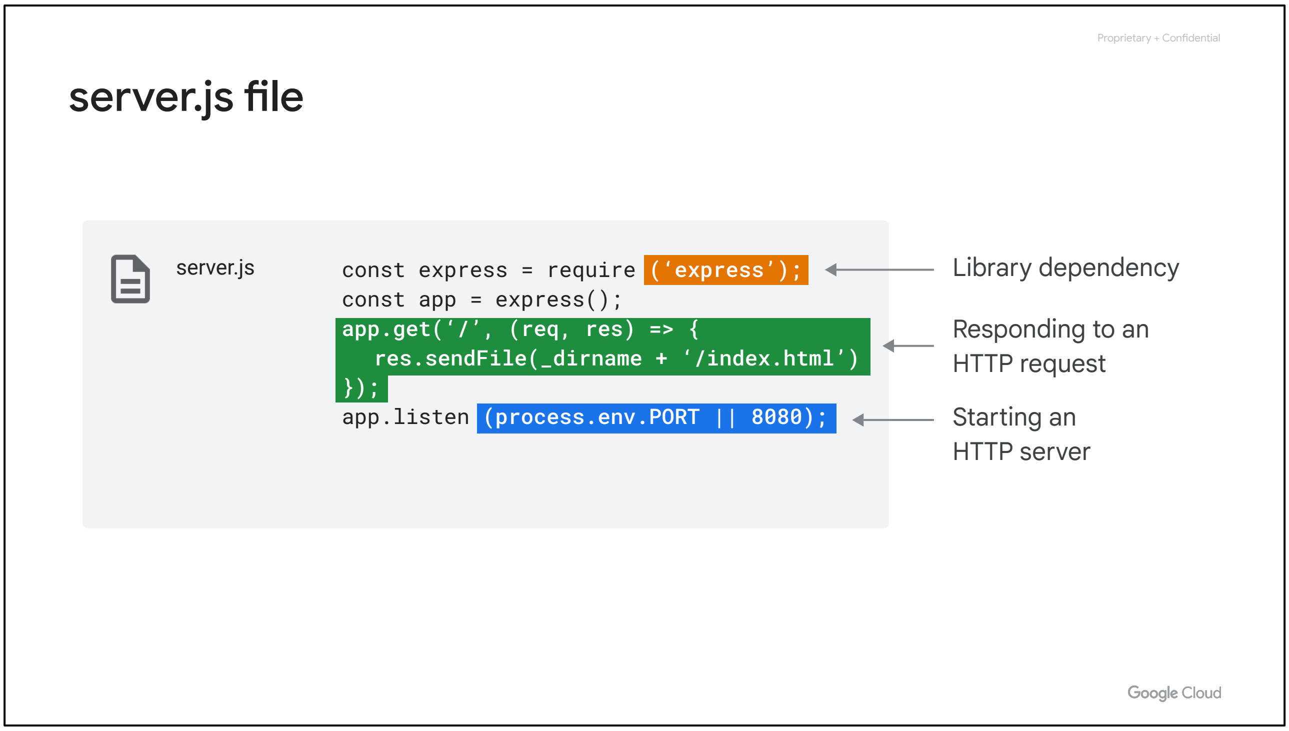


Sample web application

Here’s a minimal Node.js sample web application.

The application is made up of three files: server.js, package.json, and index.html.

To learn what you need to run the application, let’s review the content in these files.



server.js file

server.js

```
const express = require ('express');
const app = express();
app.get('/', (req, res) => {
   res.sendFile(_dirname + '/index.html')
});
app.listen (process.env.PORT || 8080);
```

Library dependency

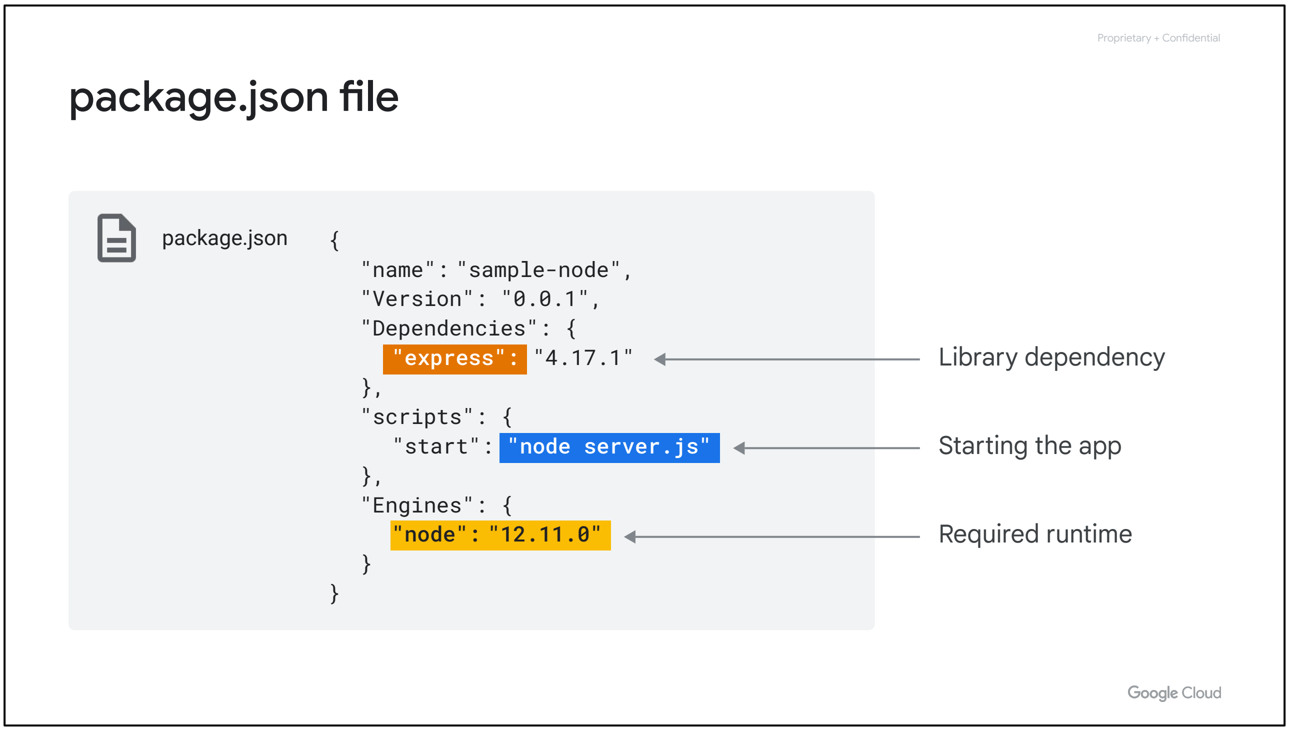
Responding to an HTTP request

Starting an HTTP server

The server.js file is the main file of the web application.

It refers to a library or module dependency (express), which is a web application framework for Node.js.

The Express module is used to create an endpoint that returns the contents of the index.html file. The app starts to listen for requests on port 8080.



package.json file

package.json

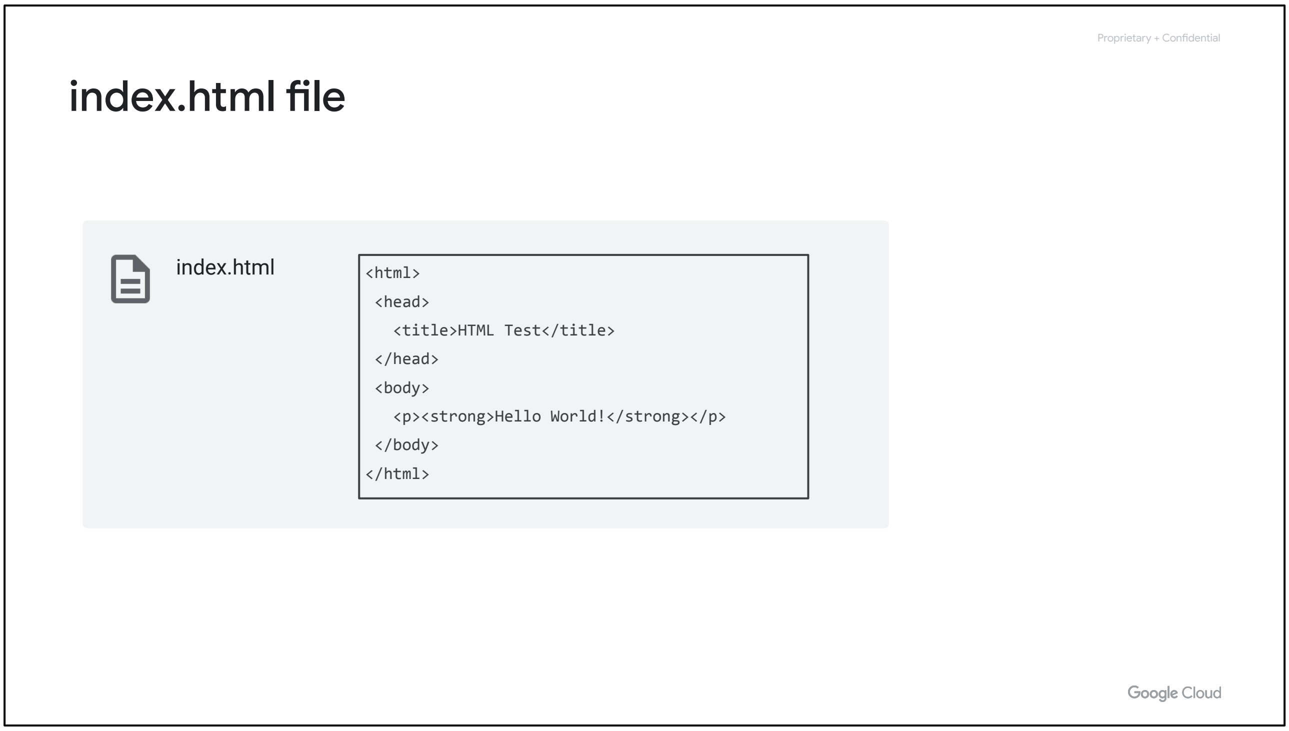
```
{
   "name": "sample-node",
   "Version": "0.0.1",
   "Dependencies": {
     "express": "4.17.1"
   },
   "scripts": {
      "start": "node server.js"
   },
   "Engines": {
      "node": "12.11.0"
   }
}
```

Library dependency
Starting the app
Required runtime

The package.json file is read by the tool **npm** (the Node.js package manager) to install dependencies.

In addition to the application name and version, the package.json specifies the:

- *Library dependency*: which is the express module, with the exact version number, so that npm knows which version to download and install.
- *Start command*: that instructs how to run the application with the node executable.
- *Required Node.js runtime*: that specifies the version of Node.js to use.

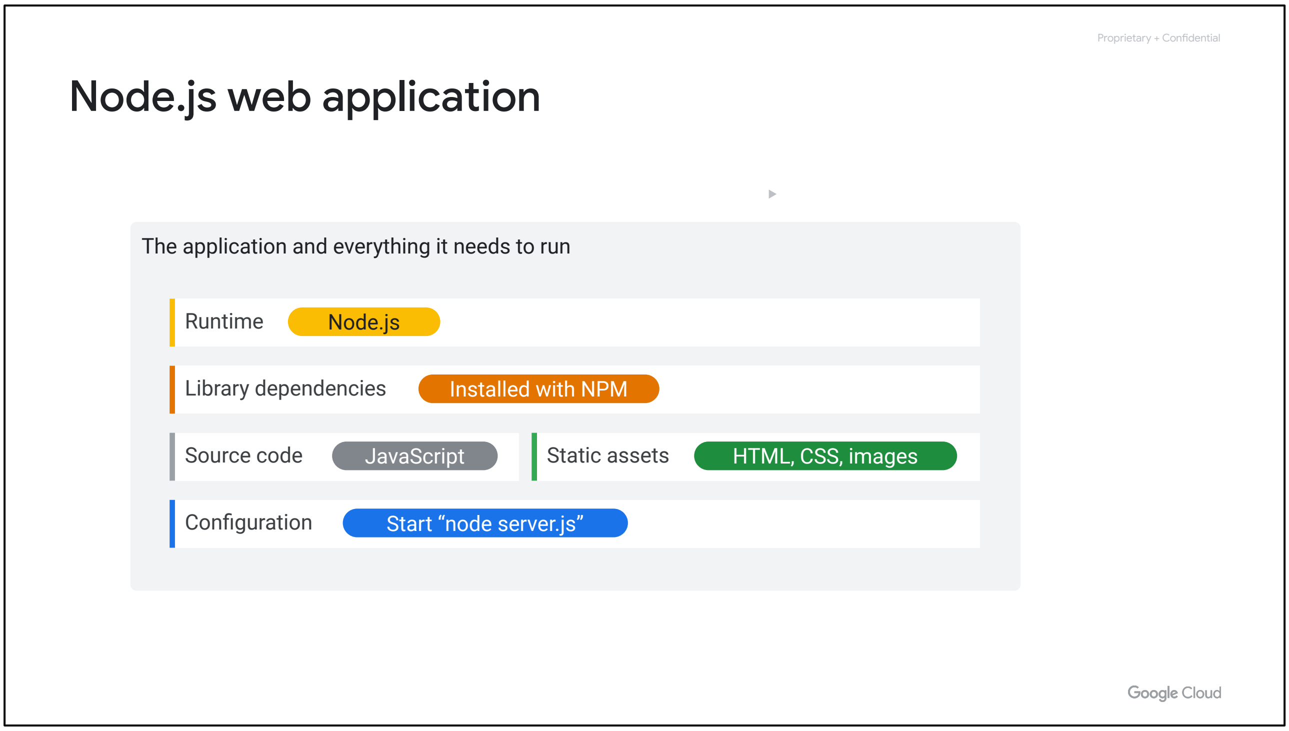


index.html file

index.html

```
<html>
 <head>
   <title>HTML Test</title>
 </head>
 <body>
   <p><strong>Hello World!</strong></p>
 </body>
</html>
```

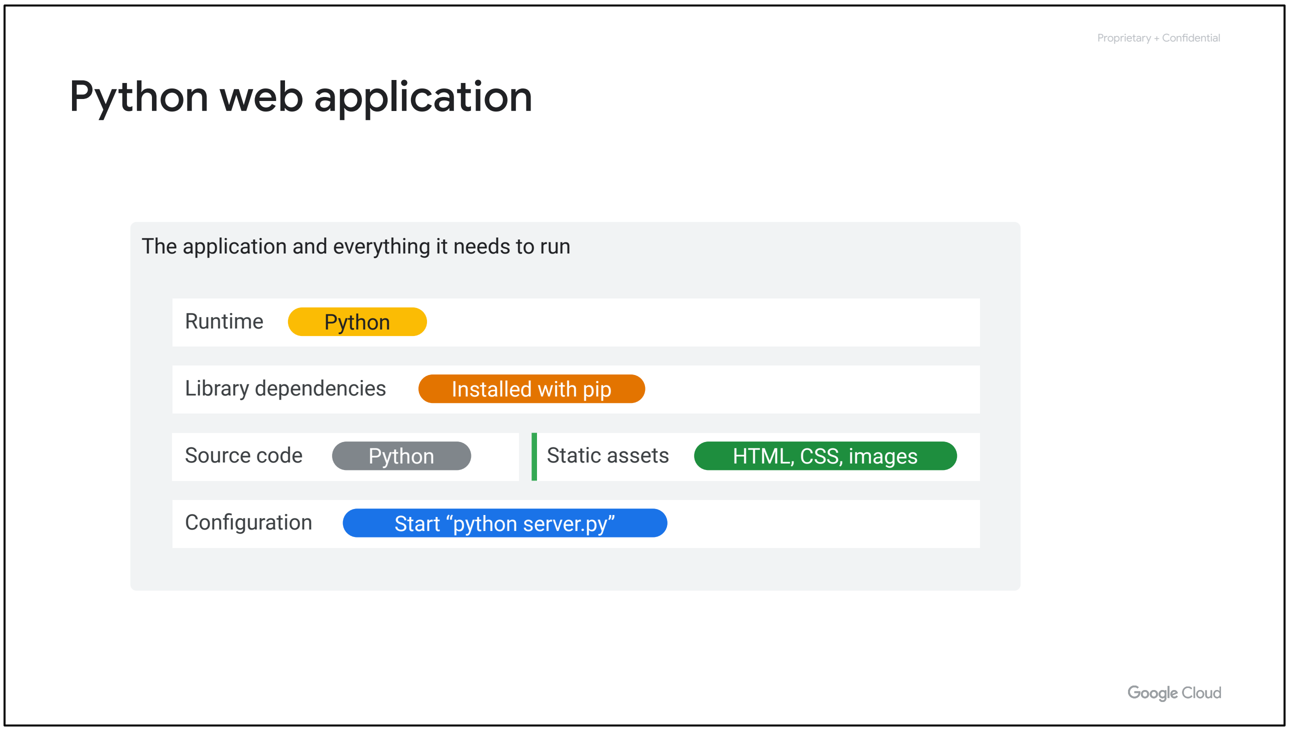
The index.html is a static HTML file that is returned by the web application on an HTTP request.



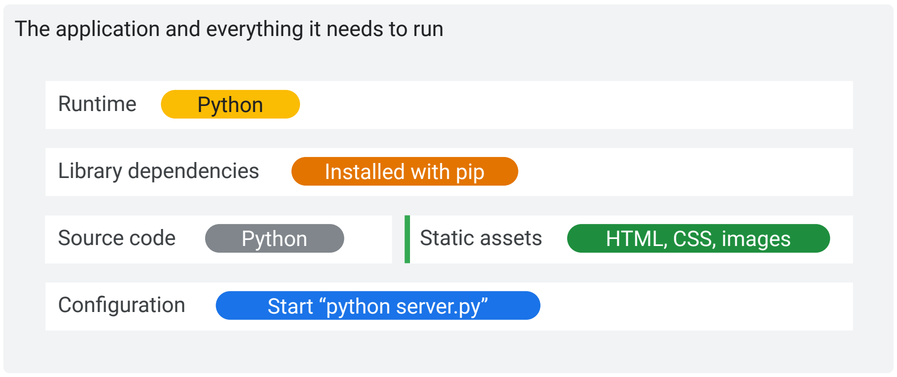
Node.js web application

To summarize, this sample application requires these five components to run the application:

- The runtime
- Dependencies, which you need to install before you can run the app.
- The source code (javascript files),
- The index.html file (or static assets in general). In reality, you’re likely to have images, CSS files, and more HTML files.
- Configuration, that includes a way to start the application. In this example, it’s “node server.js”, but it can also be more complicated.



Python web application

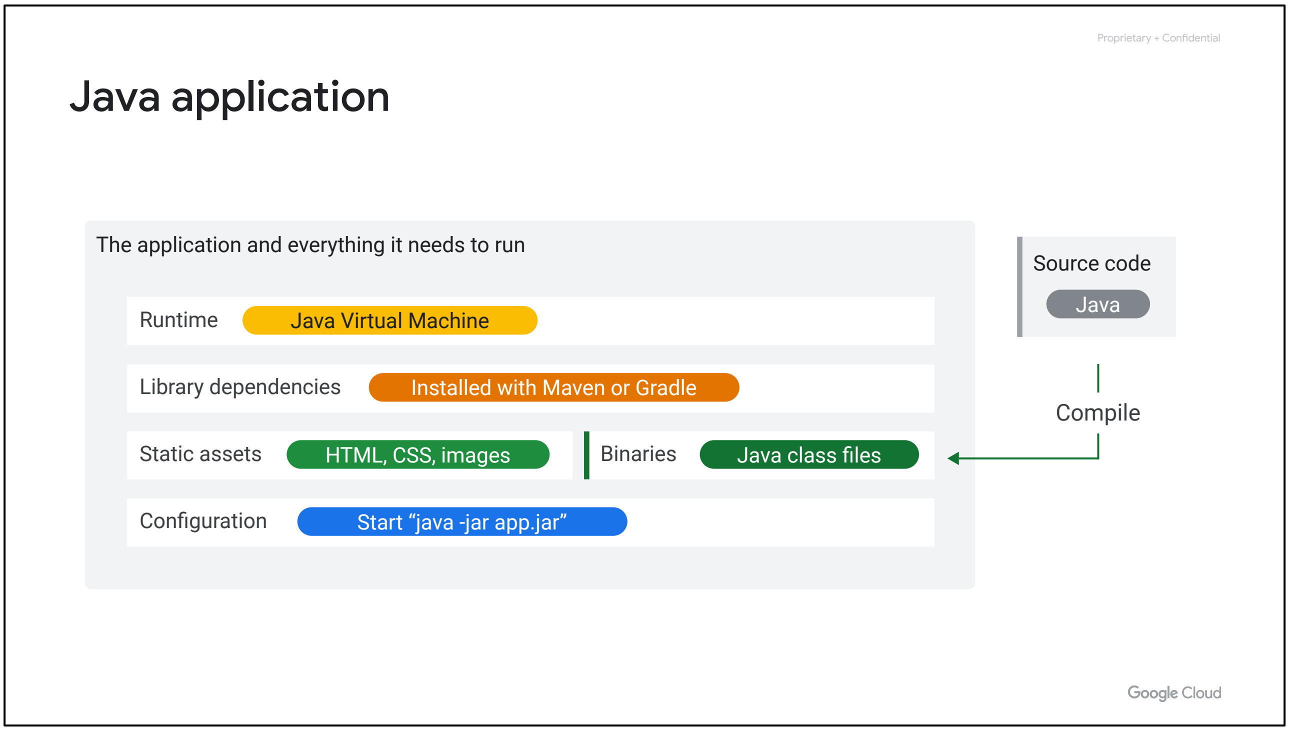


Most web applications written in Python have similar requirements.

Python is an interpreted language, so you need a *runtime* (Python) to run it.

In Python applications, you specify dependencies with a requirements.txt file, and you usually install dependencies using the package manager **pip**.

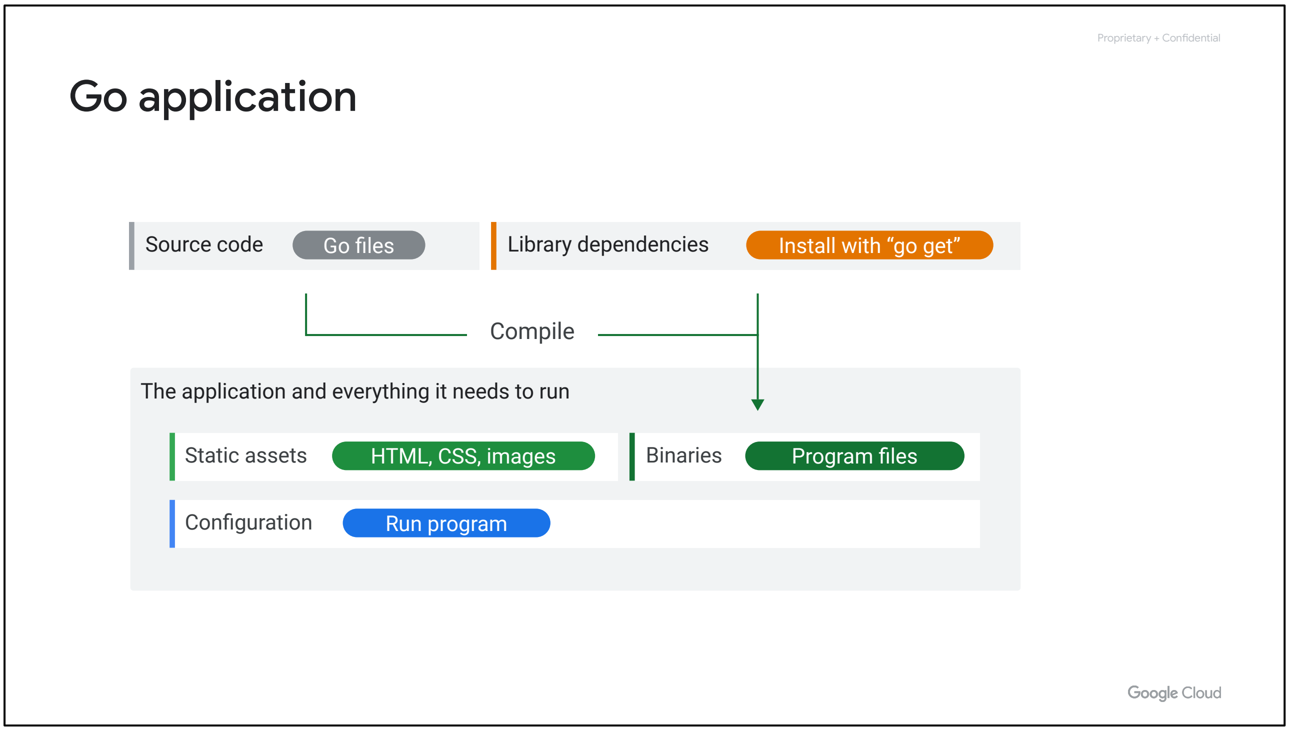
Your source code will be Python files, and the command to start your application is "python server.py" or something similar.



Java application

An application written in Java needs to be compiled first.

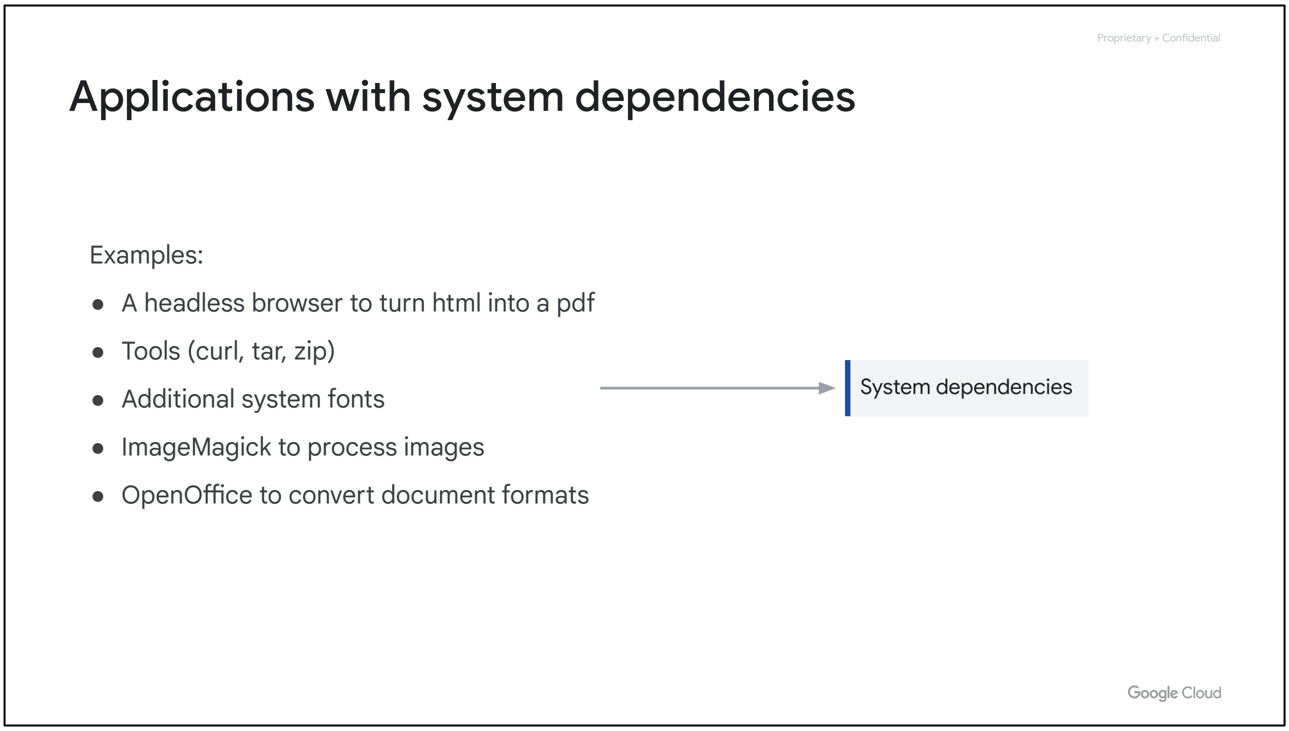
The source code is no longer required to run the application, but the compiled binaries are. You’ll need to compile the sources first.



Go application

When building an application in Go, you install dependencies and compile them together with the source code to a binary.

You can also choose to embed the assets into the binary.



Applications with system dependencies

Examples:

- A headless browser to turn html into a pdf
- Tools (curl, tar, zip)
- Additional system fonts
- ImageMagick to process images
- OpenOffice to convert document formats

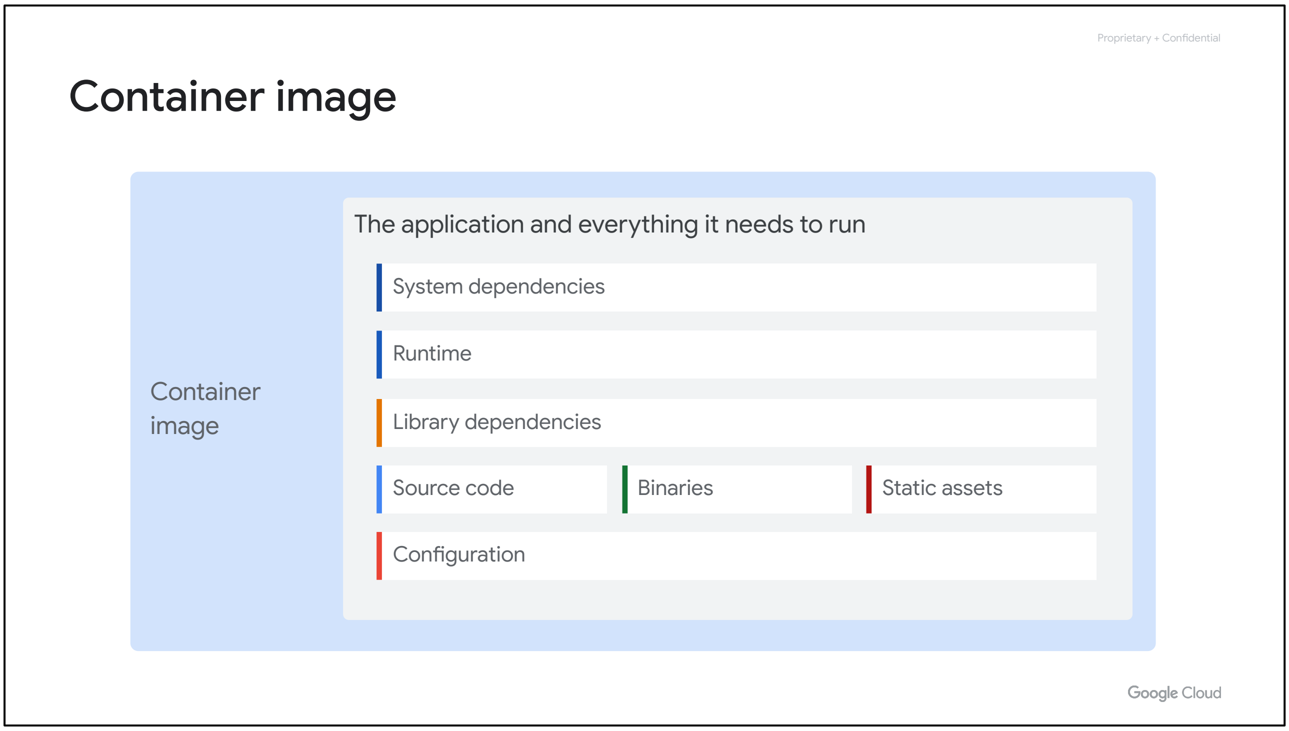
System dependencies

Some applications have dependencies on system tools that can't be expressed as an application library dependency.

Some application examples include:

- A headless browser
- Tools that you might want to use to download or process files
- Additional system fonts
- ImageMagick (a suite of open source programs that can be used to convert images from one format to another, and process them)

In development, you might also need additional tools to aid in debugging.



Proprietary + Confidential

Container image

Google Cloud

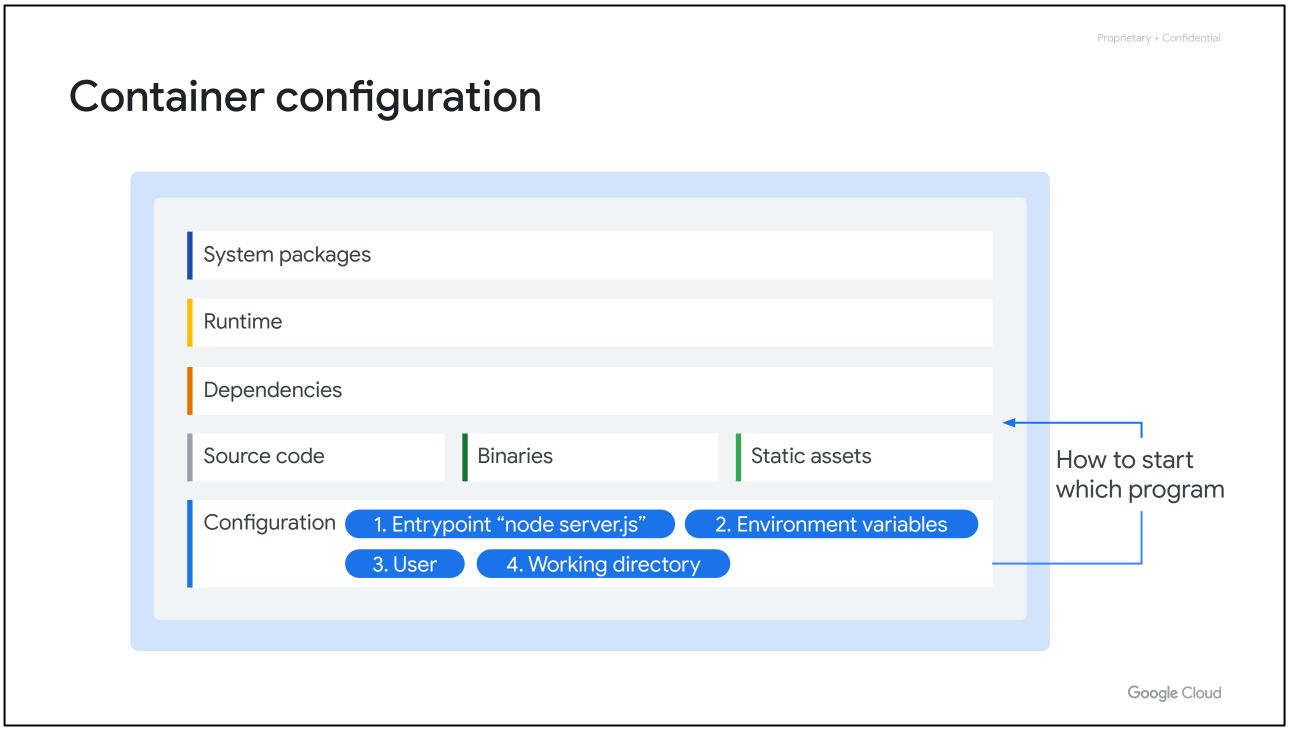
To summarize, the types of files you might need to run your application and be included in a container image are:

- System packages
- A runtime
- Library dependencies
- Source code
- Binaries
- Static assets
- Configuration

Your application might not need everything on this list. Sometimes you might just have a single binary.

The last item is the container configuration. The container configuration details how to turn a container image into a container—a running process.

In the previous examples, the command needed to run the application was specified as ‘configuration,’ which is usually an important component. But there’s more!



Proprietary + Confidential

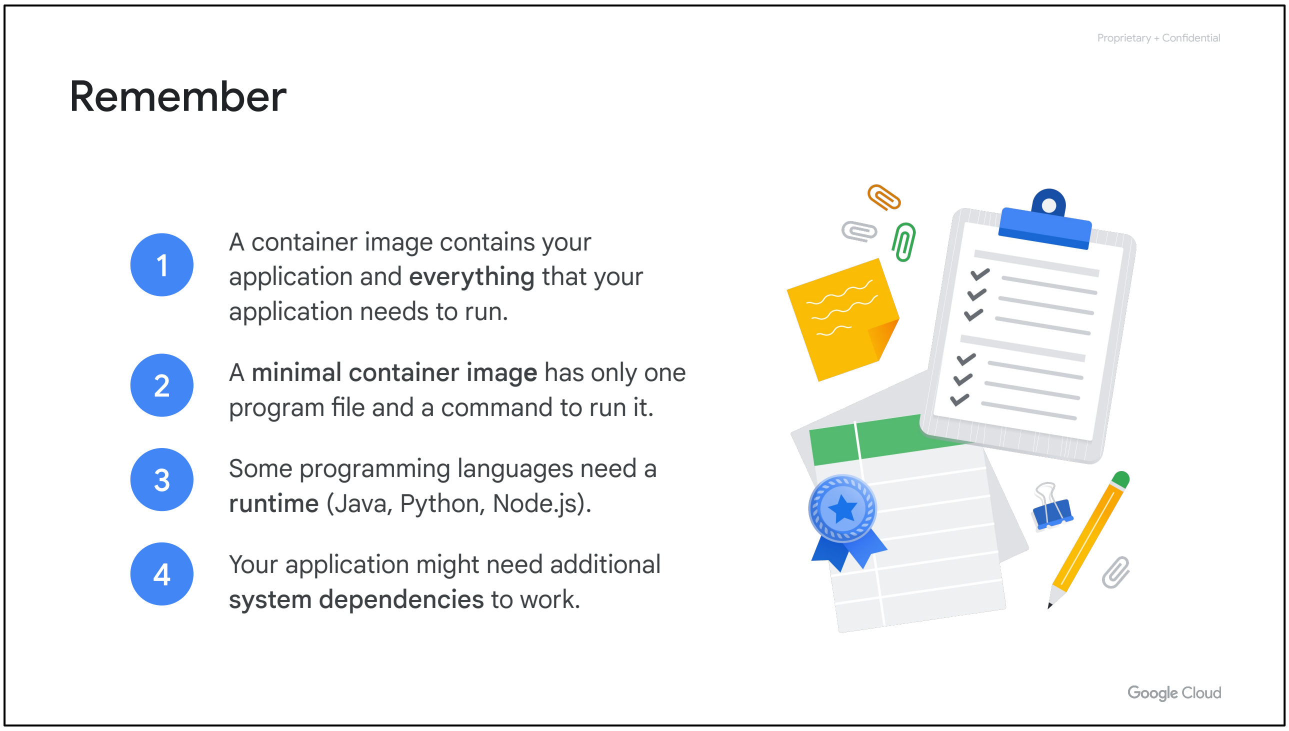
Container configuration

The command to be run when a container is started is called the entrypoint. Some other important settings are:

- Environment variables, which are used to pass configuration settings to your application.
- A working directory
- The user to run the program with.

It’s important to set the user. If not set, the root user (or system administrator) is used as the default, which is not a best practice for security reasons.

When you start the container, you can override the values of application arguments and environment variables.



Remember

1. A container image contains your application and **everything** that your application needs to run.
2. A **minimal container image** has only one program file and a command to run it.
3. Some programming languages need a **runtime** (Java, Python, Node.js).
4. Your application might need additional **system dependencies** to work.

Here's what's important to remember about container images.

A container image is a package which can include an entire runtime, source code, binaries, library dependencies, assets, and container configuration that specifies what program to start and how.

A minimal container image has only one program file and a command to run it.

Your application might need additional system dependencies to work that are also included in the image.

The container image is a self-contained package with your application and everything that it needs to run.