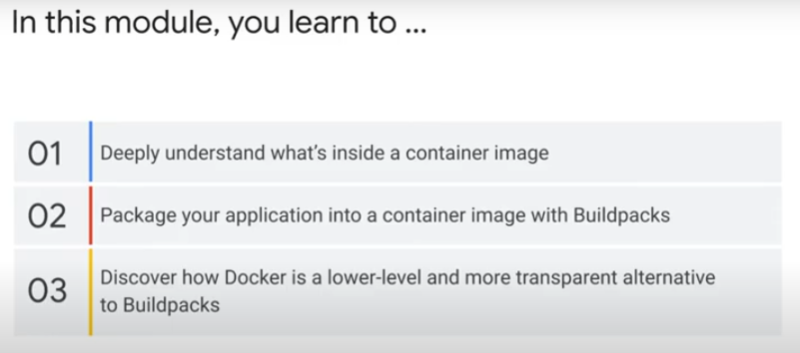
Application Development with Cloud Run

### Building Container Images

#### Introduction

Two (2) ways to deploy an application to Cloud Run:

1. use source code
2. use pre-built container image

Cloud Run does not decide how you package your application into a continer image.

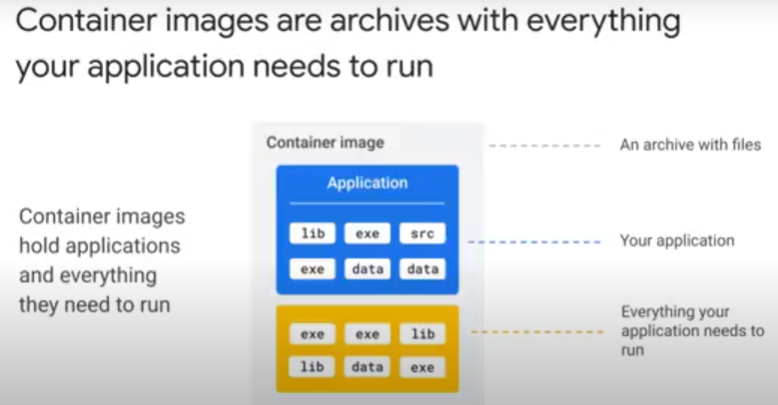
If deploy directly from source code, Cloud Run builds and packages your application using Buildpacks - https://buildpacks.io/

* Buildpacks is an open-source tool to transform application source code into container image that runs on any cloud.
  + Can use on local machine too

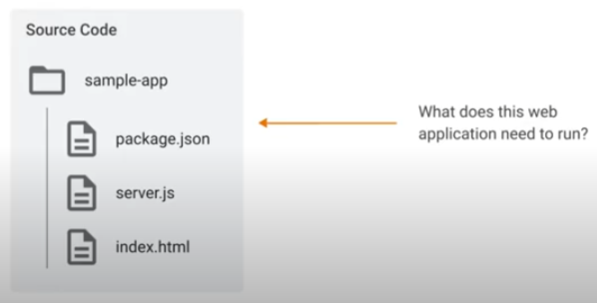
Docker is a tool that lets you build container images with a script.

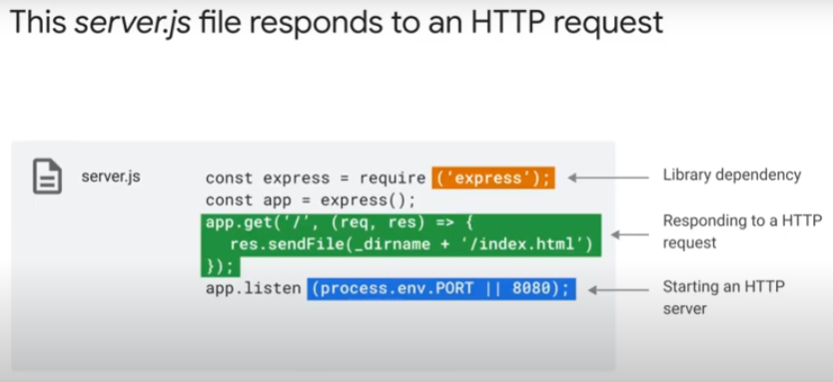
* The script puts you in full control of what goes into the container image and how it is built.
* Docker gives you more control with more transparency.

### Deep dive into a container image



A container image is an archive of your application and everything it needs to run.

We use a minimal Node.js application as an example.

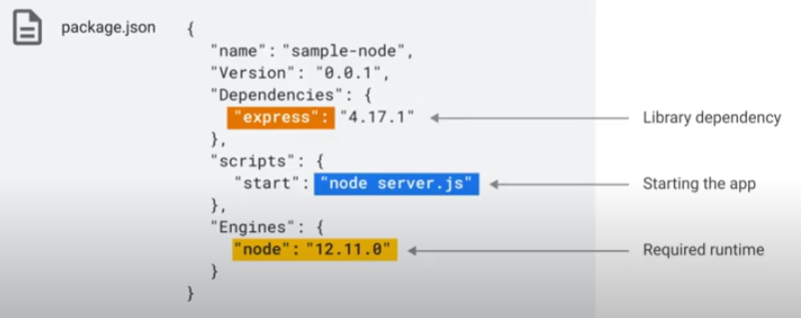
* consists of 3 files only:
  + server.js
  + package.json
  + index.html
* **server.js** is the main configuration file
* requires a library dependency for Express.
  + Express is a web application framework for Node.js
    - Generally, a Web Application Framework is a set to tools, services, resources and APIs for creating a web application.
    - Similar to CMS but need programming knowledge and skills
    - Express is mainly a routing and middleware web framework. An Express application is essentially a series of middleware function calls.
* Define routing.
  + Responds to GET requests on “/” route path by return index.html file.
  + Reference: https://expressjs.com/en/guide/routing.html
  + Routing is how an application’s endpoints (URIs) respond to clients’ requests.
  + Use methods of Express *app* object that correspond to the HTTP methods.
  + app.get handles GET requests.
  + General syntax: *app.METHOD(route path, callback/handler function/s)*
  + Express application often include a series of middleware function calls. Middleware functions have access to the *request object (req)*, the *response object (res)* and the next middleware function. Middleware function often denoted by variable name *next*.
  + https://expressjs.com/en/guide/using-middleware.html

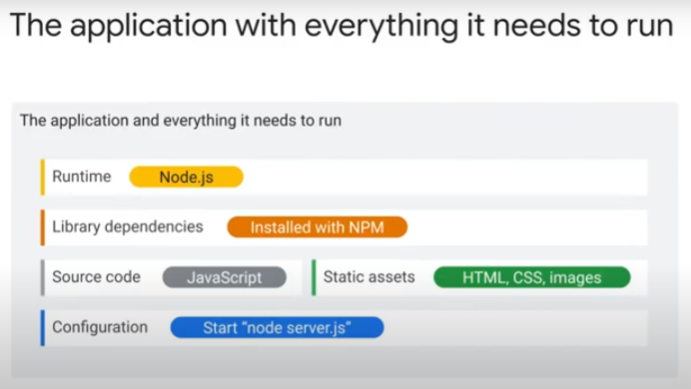
Example: app.get(‘/’, (req, res, next) => {

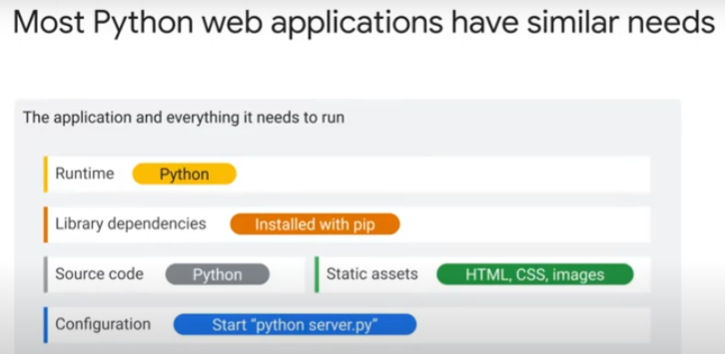
console.log(“...”)

next()

})

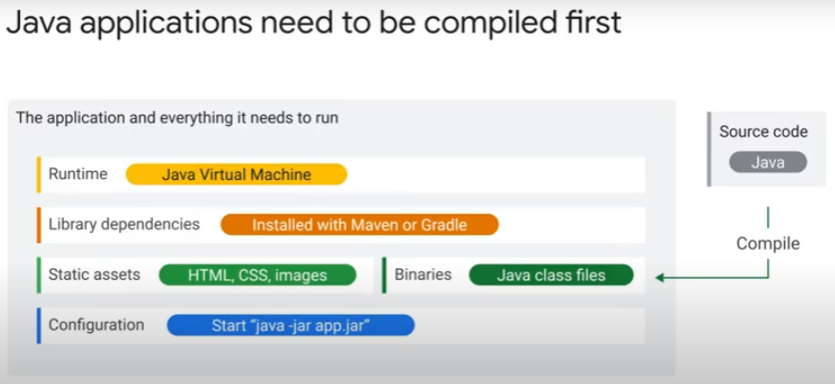
* application listens to port defined in PORT environment variable, and if none, port 8080.
* **package.json** is read by npm (node.js package manager)
* 3 key features:
  + this file specifies library dependencies for npm to download and install
    - in this example, it is a specific version of the Express package
  + value for the “start” property of “scripts” object.
    - Gives the command to run for application to start.
    - Points to the main Javascript configuration file.
  + Specifies the required node.js runtime

In summary, the 5 components of an Express node.js application:

The components of a Python application:

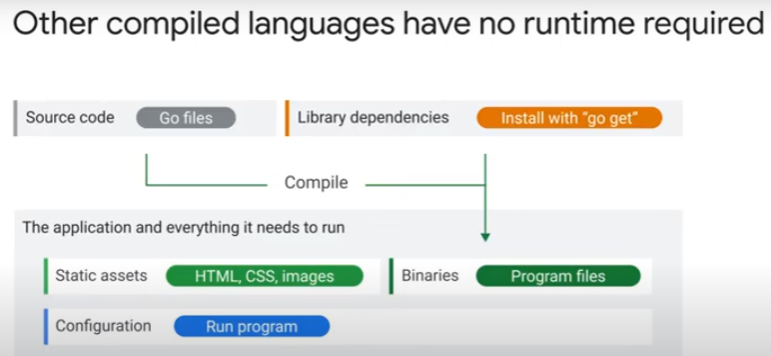
* Python is an interpreted language so need a runtime (Python or Python3)
  + source code (written by dev, .py) is lexed, parsed and compiled into intermediate language (python byte code, .pyc)
    - which is translated into native machine codes for execution
    - executed much faster than .py source codes.
    - executed by a Python virtual machine (the runtime) .. just a big loop that iterates through the instructions in the machine bytecodes, one by one, and carry out the operations.
* Dependencies specified with *requirements.txt* file.
  + Installed with *pip*, the python package manager.
* Source code: python files (.py)
* static assets: HTML, CSS, images
* configuration: to start > “python server.py”

Container image for Java application:



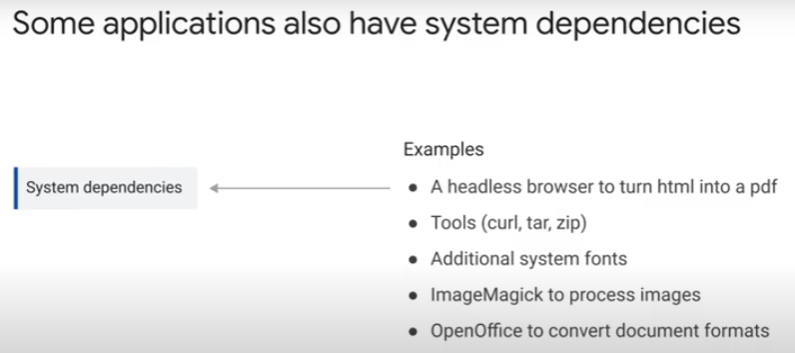
* Java is a compiled language.
  + We don’t need source code to run application. So, no need to include in container image.
  + Instead, include the compiled binaries (.jar files).

Container image for Go application:

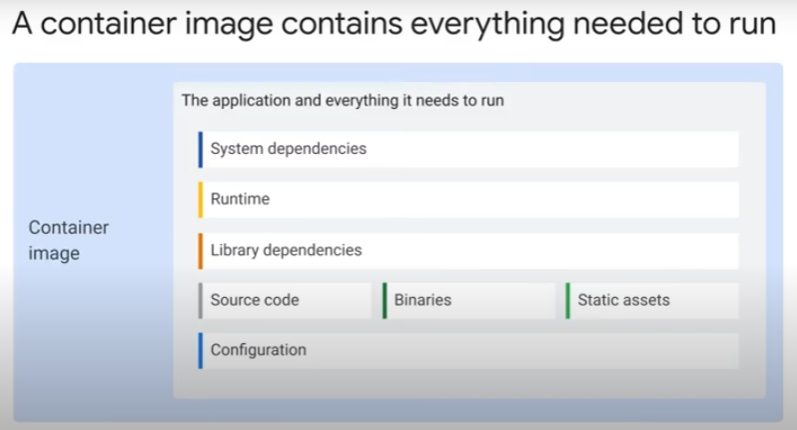


* compiled language (like Java) but **no** runtime
* Copy source code to container image + download & install dependencies > compile into a binary. Assets can be embeded into the binary too.

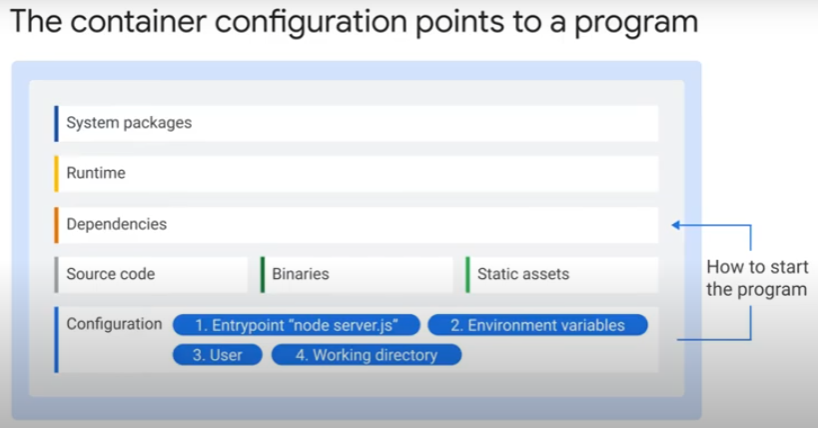
Some applications (by entension, Cloud Run services) need additional tools to function:

* these are not application dependencies/libraries.
* These tools are included in the container image and deployed to Cloud Run.
* Examples of these tools:
  + headless browser (a brawser without GUI elements).
    - Most commonly used for automation tests (of webpage forms, mouse clicks, keyboard inputs etc), layout checks, website performance tests, web scrapping data extraction.
    - Chrome & Firefox have headless modes.
  + tools to download and process files (zip, curl, tar)
  + additional system fonts
  + ImageMagick – open-source suite for editing and converting images from one format to another.
    - https://imagemagick.org/index.php
  + LibreOffice and OpenOffice to convert document formats
  + tools to help development and debugging
    - e.g. Xdebug for PHP. https://xdebug.org/

General overview of what you may need in a container image:



* may need some/all parts to run an application

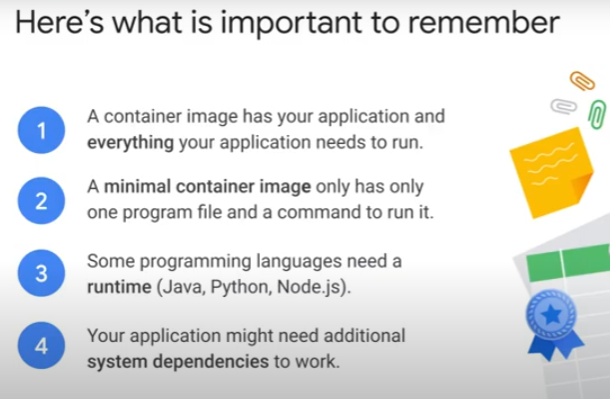


* Configuration of container image

=> contains details to turn a container image into a container (running processes)

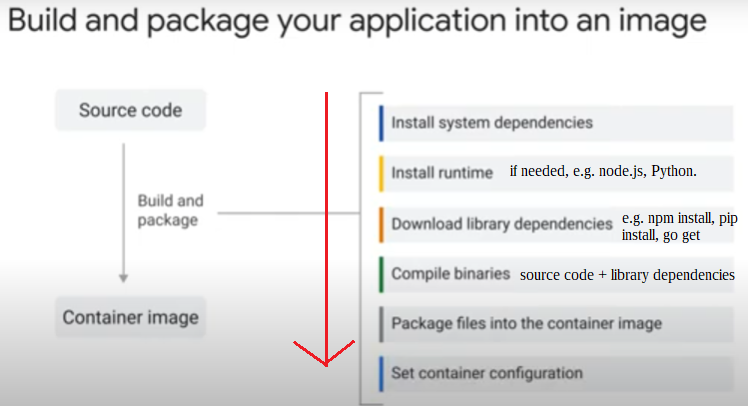
* + command to run application – called “entrypoint”
  + environment variables – context settings for application
  + working directory – where to start the app
  + user to start program with.
    - Default is root user, system admin.
    - Risky from security perspective because root user role has more/additional permissions, beyond those needed to run application.
    - So, important to customize this parameter.
* You can override the program arguments and environment variables in the image when you are deploying and starting the container.

Summary:

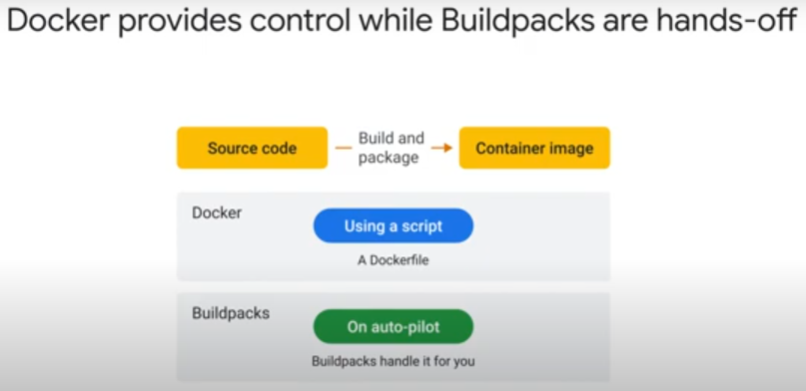


### Building images with Docker

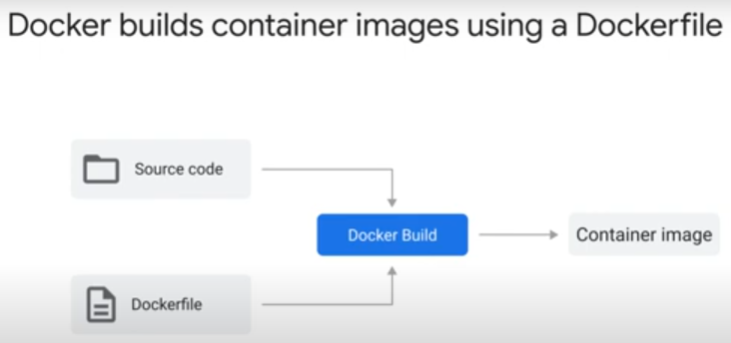
To build and package an application into a container image, need to go through these steps:



Two ways to build container image from source code:



|  |  |
| --- | --- |
| Docker | Buildpacks |
| https://www.docker.com/ | https://buildpacks.io/ |
| uses a text-based file with no extension called **Dockerfile**. Contains a script of instructions. | use heuristics to build and package source code.  E.g. if it detects *package.json* file, it will run “npm install”. |
| low-level, transparent, flexible. | convenient, automated |

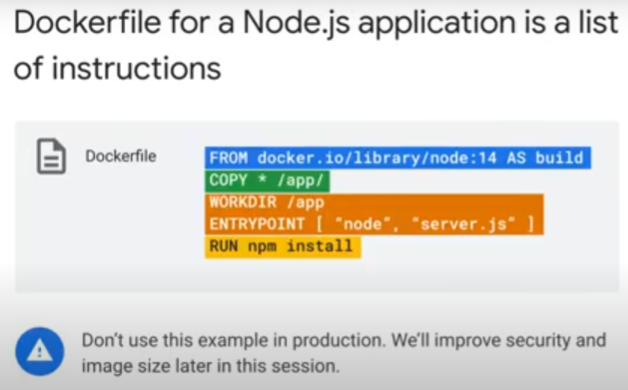


Docker

- a container engine

- can run containers on local machine

- can build container images

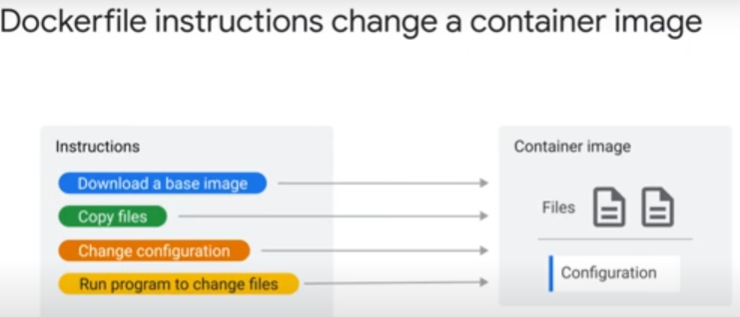
Dockerfile for building Node.js application into a container image.

1. Download node.js base image from docker hub.

2. copy source code into image.

3. install app dependencies.

4. changes configuration to run application when starts “node server.js”.

Docker builds app inside container image.

1. pulls **base** image which **include toolings** to build application.

2. pull source code into downloaded image’s working directory.

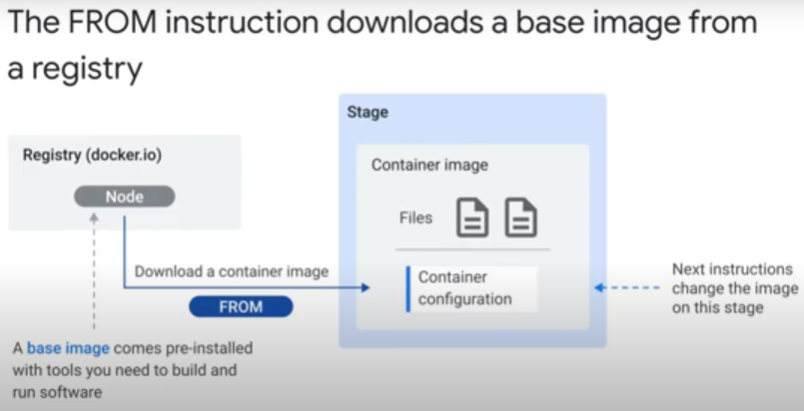
3. to build app, run program inside image to changes files.

4. configure image to start application.

Docker combines build and package into one process. This is an issue as the base image with toolings is usually very large in size.

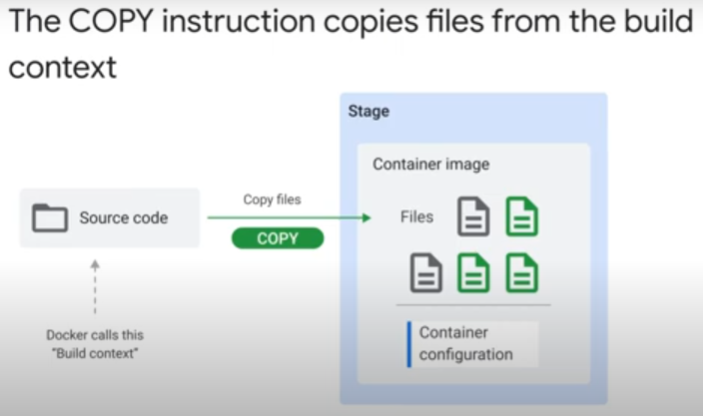
*Instructions in Dockerfile in depth.*

The FROM instruction in Dockerfile



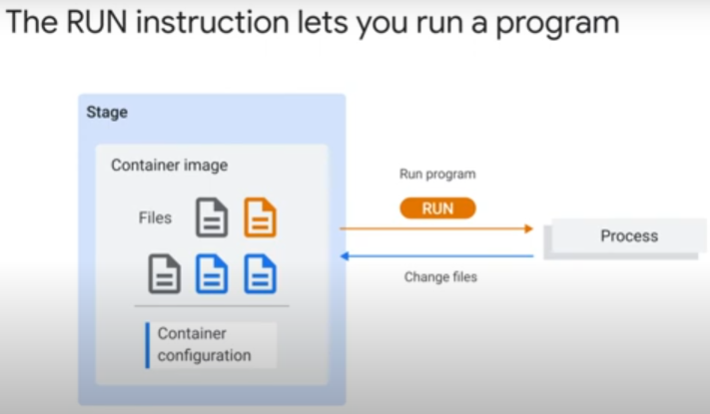
FROM instruction downloads base image from a registry, and puts on a stage, to be modified by later instructions.

E.g. Golang (Go programs), node.js (node programs)



COPY instruction copies source code from \* to /app directory of the downloaded base image.

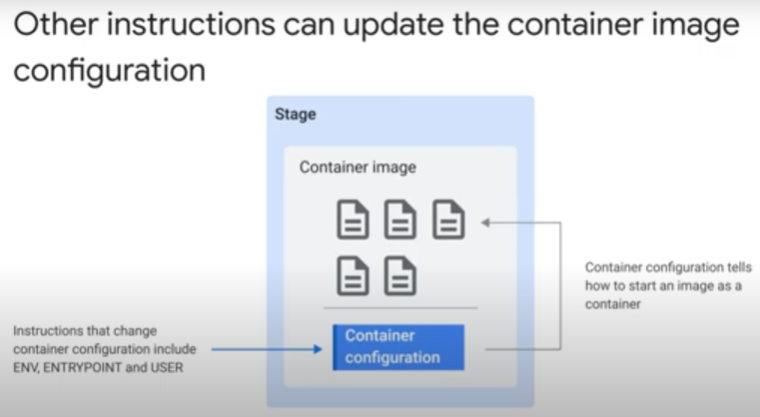
Docker has concept of “build context”, which is usually directory with the Dockerfile.

RUN instruction runs a program from the image. Means program must be present within base image.

Program makes changes to files. It can see only the files on the image.

Can RUN instruction to

1. download system pacakage/s needed to build application.
2. download application libraries/dependencies
3. compile source code into binaries

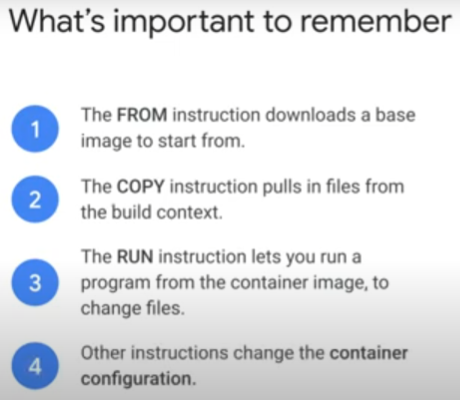
Configuration instructions tell container runtime (Docker or Cloud Run) which is the program file to start and the starting parameters.

Important instructions:

ENTRYPOINT – command and file to start app

ENV – sets environment variable

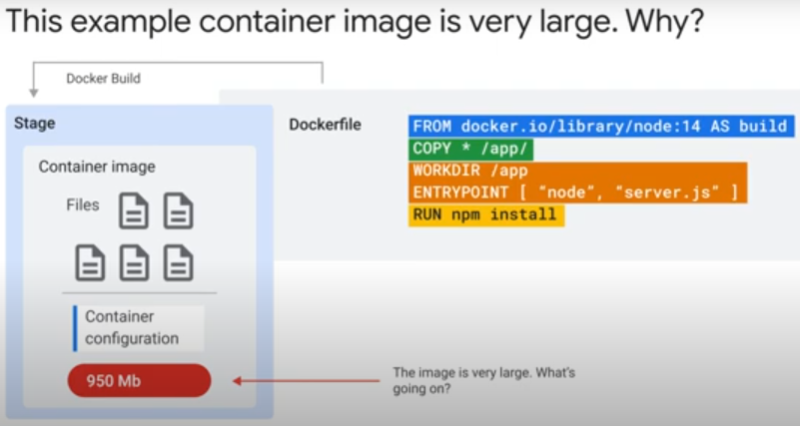
WORKDIR – sets working directory of program and user.

USER – sets user for program. Default is that of base image. If not set in base image, it may be root / system admin. So, impt to specify for security reason.

Dockerfile instructions

Start with downloading a base image and putting it on a stage.

Every later instruction changes that stage container image.

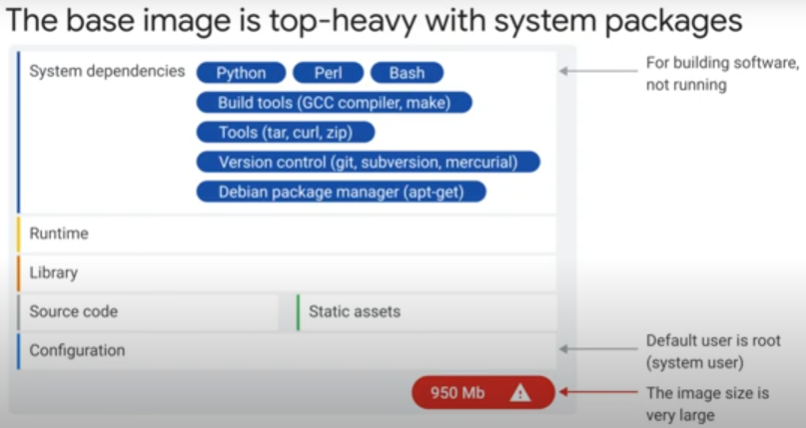


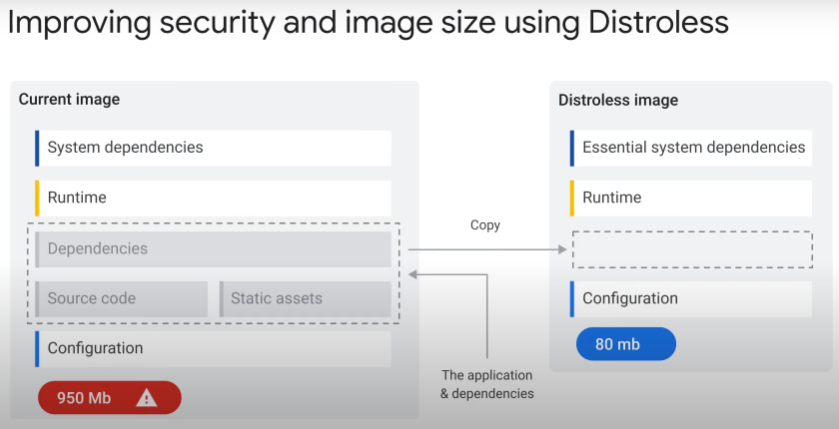
Using minimal node.js application as an example:

node.js runtime is ~80MB.

The minimal node application and dependencies <1MB.

Base image was DockerHub node image.

* Comes with many system packages needed to build software.
* even Debian tool “apt-get” is included in case you need to download more system packages.
* Many packages not needed by a running container.
  + More security vulnerabilities with more system packages.
  + In production, smaller images are better for security.
* Image more suitable for building software than for a running application/container.

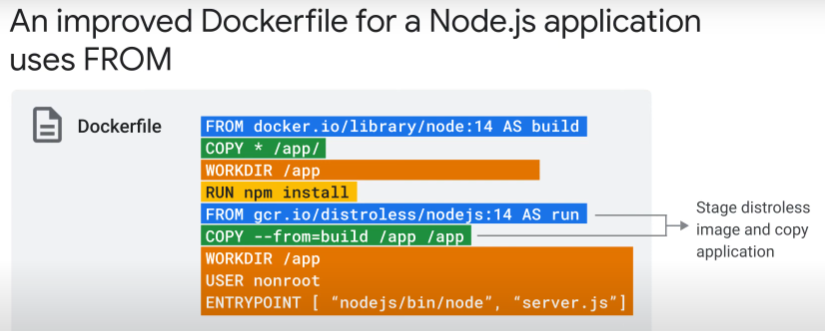


Solution by Docker is to do **multi-stage image build**.

Stage 1: Download initial build image, copy source code, static assets and application depdendencies.

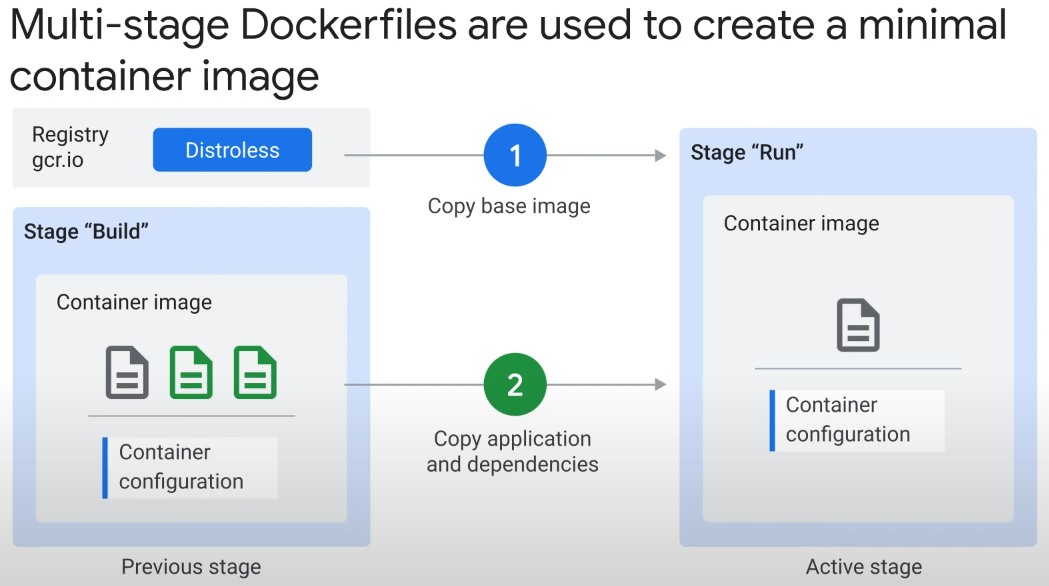
Stage 2: Download an image that only has dependencies essential for running container.

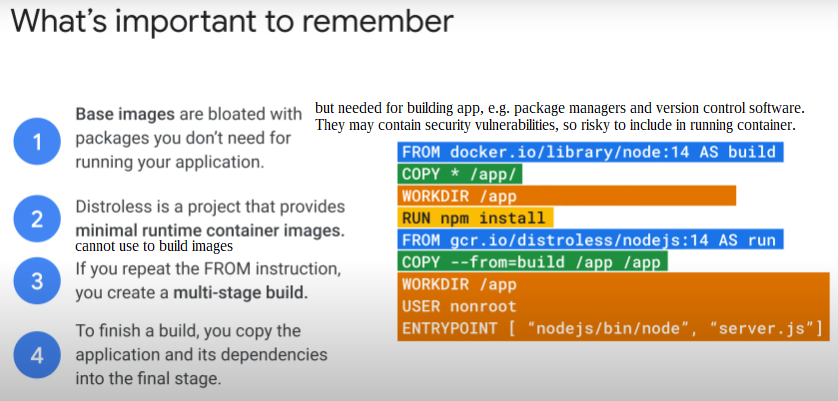
* Then copy the application and dependencies to this distroless image.
* Image from Distroless project > only application, runtime and runtime dependencies. No build-time dependencies.

In Dockerfile, repeat FROM instruction to download distroless container image after COPYing source code and RUNning package manager to download app dependencies.

Stage 2 COPY instruction to copy built app from precious image onto current active stage.

Set USER as nonroot .. no system administrator permissions

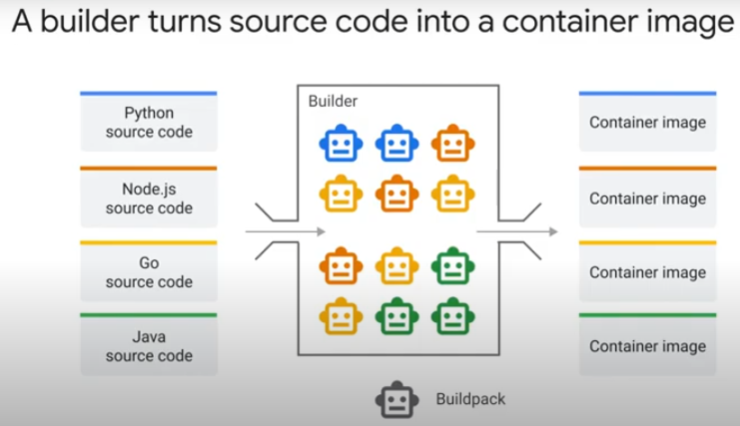




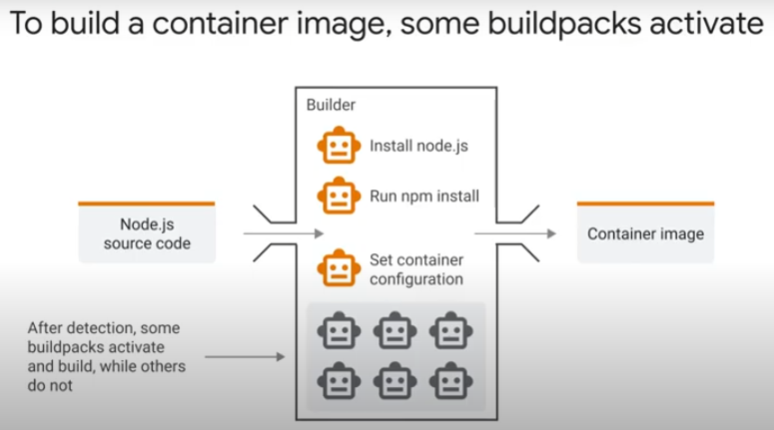
### Building Images with Buildpacks

### 

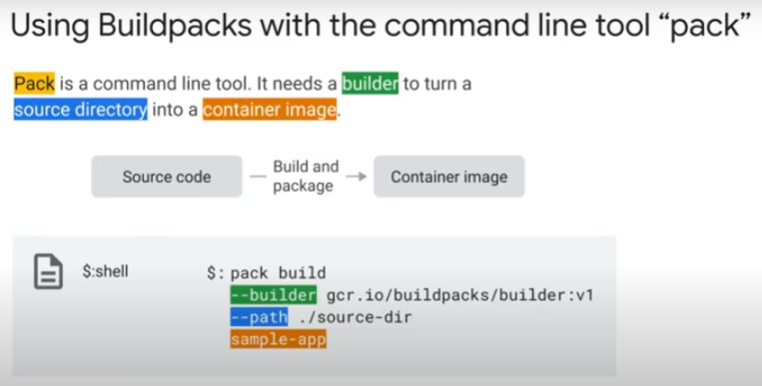
* A way to turn source code into container image **without** writing a Dockerfile.
* Buildpacks are built into Cloud Run for the source-based development workflow
* open-source
* Multiple vendors/projects, e.g.
  + **Heroku** buildpacks
    - Heroku is known for cloud-native PaaS for modern app development.
    - https://elements.heroku.com/buildpacks
  + **Paketo** buildpacks
    - by Cloud Foundry Foundation, dedicated to maintaining vendor-neutral governance
    - https://paketo.io/



* a Builder
  + contains a collection of buildpacks (little robots).
    - Buildpacks do the actual work of building and packaging into container images
* Builder can process source codes in multiple different programming languages, e.g. Python, Node.js, Java, Go.
* Builder executes in 2 lifecycle phrases
  1. Detect phrase
     + Builder asks all buildpacks if they can process the source.
     + e.g. npm buildpack will look for package.json file in source. If have, it will perform npm install in build phase.
  2. Build phase
     + All buildpacks that can contribute are activated to perform a part
     + e.g. 3 buildpacks activated.
       - One installs node.js runtime.
       - Another runs npm install (package).
       - Final buildpack configures image to start node.js runtime
  + net result is image that can deploy to Cloud Run or run with Docker locally



Build container images with CLI tool “Pack”



* pack is from the open-source Cloud Native Buildpacks project.
* https://buildpacks.io/docs/tools/pack/
* pack CLI works with Heroku and Paketo too

