**DOCKER  
KUBERNETES**

**STEPHEN GRIDER**

**Table of Contents**

[**I] Dive into Docker** 5](#_Toc104643218)

[3. Why Use Docker 5](#_Toc104643219)

[4. What is Docker 5](#_Toc104643220)

[5. Docker for Mac/Windows 5](#_Toc104643221)

[6. Installing Docker on MacOS 5](#_Toc104643222)

[7. Installing Docker with WSL2 on Windows 10 Home and Pro 5](#_Toc104643223)

[8. Installing Docker for Windows Professional with HyperV 6](#_Toc104643224)

[9. More Windows Professional with HyperV 6](#_Toc104643225)

[10. More Windows Professional with HyperV 6](#_Toc104643226)

[11. Installing Docker on Linux 6](#_Toc104643227)

[12. Using the Docker Client 6](#_Toc104643228)

[13. But Really… What’s a Container? 6](#_Toc104643229)

[14.How’s Docker Running on Your Computer? 6](#_Toc104643230)

[**II] Manipulating Containers with Docker Client** 7](#_Toc104643231)

[15. Docker Run in Detail 7](#_Toc104643232)

[16. Overriding Default Commands 7](#_Toc104643233)

[17. Listing Running Containers 7](#_Toc104643234)

[18. Container Lifecycle 7](#_Toc104643235)

[19. Restarting Stopped Containers 7](#_Toc104643236)

[20. Removing Stopped Containers 7](#_Toc104643237)

[21. Retrieving Log Outputs 7](#_Toc104643238)

[22. Stopping Containers 7](#_Toc104643239)

[23. Multi-Command Containers 7](#_Toc104643240)

[24. Executing Commands in Running Containers 7](#_Toc104643241)

[25. The Purpose of IT Flag 7](#_Toc104643242)

[26. Getting a Command Prompt in a Container 7](#_Toc104643243)

[27. Starting with a Shell 7](#_Toc104643244)

[28. Container Isolation 7](#_Toc104643245)

[**III] Building Custom Images Through Docker Server** 8](#_Toc104643246)

[29. Creating Docker Images 8](#_Toc104643247)

[30. Buildkit for Docker Desktop 8](#_Toc104643248)

[31. Building a DockerFile 8](#_Toc104643249)

[32. Dockerfile Teardown 8](#_Toc104643250)

[33. What’s a Base Image? 8](#_Toc104643251)

[34. The Build Process in Detail 8](#_Toc104643252)

[35. A Brief Recap 8](#_Toc104643253)

[36. Rebuilds with Cache 8](#_Toc104643254)

[37. Tagging an Image 8](#_Toc104643255)

[38. Quick Note for Windows Users 8](#_Toc104643256)

[39. Manual Image Generation with Docker Commit 8](#_Toc104643257)

[**IV] Making Real Projects with Docker** 9](#_Toc104643258)

[40. Project Outline 9](#_Toc104643259)

[41. Node Server Setup 9](#_Toc104643260)

[42. A Few Planned Errors 9](#_Toc104643261)

[43. A required Node Base Image Version 9](#_Toc104643262)

[44. Base Image Issues 9](#_Toc104643263)

[45. A Few Missing Files 9](#_Toc104643264)

[46. Copying Build Files 9](#_Toc104643265)

[47. Container Port Mapping 9](#_Toc104643266)

[48. Specifying a Working Directory 9](#_Toc104643267)

[49. Unnecessary Rebuilds 9](#_Toc104643268)

[50. Minimizing Cache Busting and Rebuilds 9](#_Toc104643269)

[**IV] Making Real Projects with Docker** 10](#_Toc104643270)

[40. Project Outline 10](#_Toc104643271)

[41. Node Server Setup 10](#_Toc104643272)

[42. A Few Planned Errors 10](#_Toc104643273)

[43. A required Node Base Image Version 10](#_Toc104643274)

[44. Base Image Issues 10](#_Toc104643275)

[45. A Few Missing Files 10](#_Toc104643276)

[46. Copying Build Files 10](#_Toc104643277)

[47. Container Port Mapping 10](#_Toc104643278)

[48. Specifying a Working Directory 10](#_Toc104643279)

[49. Unnecessary Rebuilds 10](#_Toc104643280)

[50. Minimizing Cache Busting and Rebuilds 10](#_Toc104643281)

[**V] Docker Compose with Multiple Local Containers** 11](#_Toc104643282)

[51. App Overview 11](#_Toc104643283)

[52. App Server Starter Code 11](#_Toc104643284)

[53. Assembling a Dockerfile 11](#_Toc104643285)

[54. Introducing Docker Compose 11](#_Toc104643286)

[55. Docker Compose Files 11](#_Toc104643287)

[56. Networking with Docker Compose 11](#_Toc104643288)

[57. Docker Compose Commands 11](#_Toc104643289)

[58. Stopped Docker Compose Containers 11](#_Toc104643290)

[59. Container Maintenance with Docker Compose 11](#_Toc104643291)

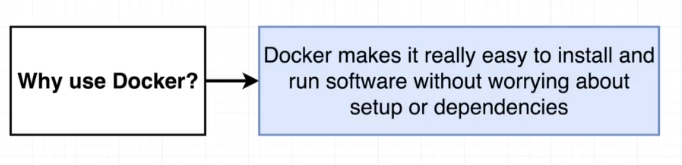
[60. Automatic Container Restarts 11](#_Toc104643292)

[61. Container Status with Docker Compose 11](#_Toc104643293)

# **I] Dive into Docker**

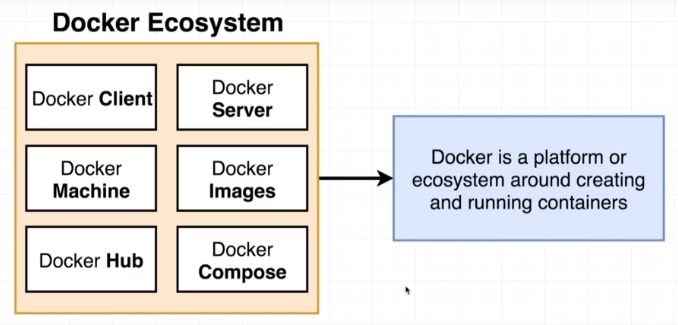
## 3. Why Use Docker

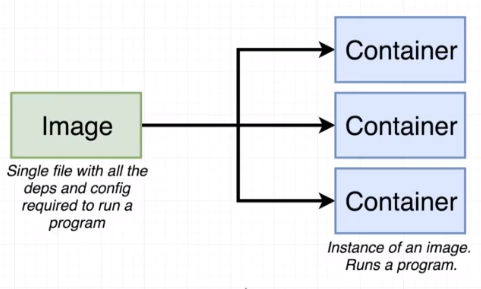
## 



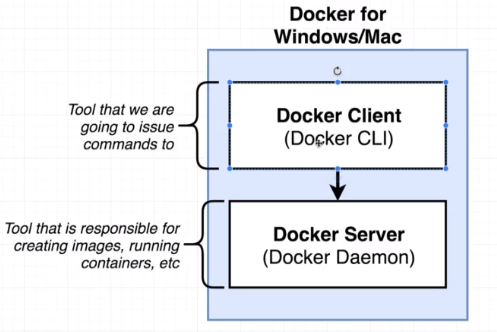
## 4. What is Docker

## 





## 5. Docker for Mac/Windows



## 6. Installing Docker on MacOS

## 7. Installing Docker with WSL2 on Windows 10 Home and Pro

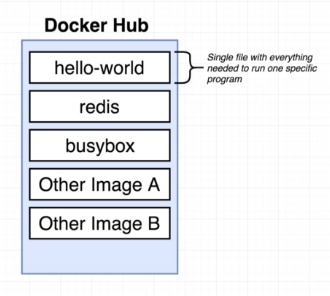
## Installing Docker for Windows Professional with HyperV

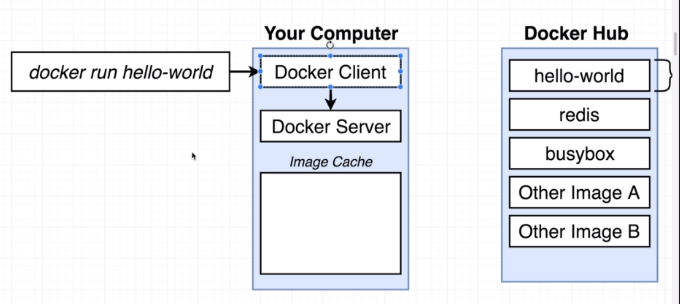
## 8. Installing Docker on Linux

## 9. Using the Docker Client

docker version

docker run hello-world





## 10. But Really… What’s a Container?

How OS works?

## 

With namespacing, we can look at all of the different hardware resources connected to our computer and we can essentially segment out portions of those resources.

So, we could create a segment of our hard disc specifically dedicated to housing Python version 2.

And we could make a second segment specifically dedicated to a housing Python version 3.

Then, to make sure that Chrome has access to this segment over here and Node.js has access to this segment over here, anytime that either of them issues a system call to read information off the hard drive, the kernel will look at that incoming system call and try to figure out which process it is coming from.

So, the kernel could say, "Okay, if Chrome is trying to read some information off the hard drive, I'm gonna direct that call over to this little segment of the hard disc over here, the segment that has Python version 2 and Node.js."

Anytime that makes a system call to read the hard drive,

the kernel can redirect that over to this segment

for Python version 3.

And so by making use of this kind of namespacing

or segmenting feature,

we can have the ability to make sure that Chrome

and Node.js are able to work on the same machine.

Now again, in reality,

neither of these actually needed installation of Python.

This is just a quick example.

So, this entire process of kind of segmenting a hard,

excuse me, a hardware resource based on the process

that is asking for it is known as namespacing.

With name spacing, we are allowed to isolate resources

per a process or a group of processes,

and we're essentially saying that anytime

this particular process asks for a resource,

we're gonna direct it to this one little specific area

of the given piece of hardware.

Now, namespacing is not only used for hardware,

it can be also used for software elements as well.

So for example, we can namespace a process

to restrict the area of a hard drive that'd be available,

or the network devices that are available,

or the ability to talk to in other processes,

or the ability to see other processes.

These are all things that we can use namespacing

for to essentially limit the resources

or kind of redirect request for resource

from a particular process.

Very closely related to this idea of namespacing

is another feature called control groups.

A control group can be used to limit the amount

of resources that a particular process can use.

So, namespacing is for saying,

"Hey, this area of the hard drive is for this process."

A control group can be used to limit the amount of memory

that a process can use, the amount of CPU,

the amount of hard drive input, input.

Or excuse me, input output,

and the amount of network bandwidth as well.

So, these two features put together can be used

to really kind of isolate a single process

and limit the amount of resources it can talk to,

and the amount of bandwidth essentially,

that it can make use of.

Now, as you might imagine,

this entire kind of little section right here,

this entire vertical of a running process,

plus this little segment of a resource that it can talk to

is what we refer to as a container.

And so, when people say,

"Oh yeah, I have a Docker Container."

You really should not think of these

as being like a physical construct

that exists inside of your computer.

Instead, a container is really a process

or a set of processes that have a grouping of resources

specifically assigned to it.

And so, this is the diagram

that we're gonna be looking at quite a bit

anytime that we think about a container.

We've got some running process

that sends a system call to a kernel.

The kernel is going to look at that incoming system call

and direct it to a very specific portion of the hard drive,

the RAM, CPU or whatever else it might need.

And a portion of each of these resources

is made available to that singular process.

Now, the last question you might have here is,

"Okay. Well, I get what a container is,

but with that in mind,

what is the real relation between one of those containers

or that kind of singular process

and grouping of resources to an image?

How is that single file eventually create this container?"

That's a good question.

One more quick diagram.

Anytime that we talk about an image,

we're really talking about a file system snapshot.

So, this is essentially kind of like a copy paste

of a very specific set of directories or files.

And so we might have an image

that contains just Chrome and Python.

An image will also contain a specific startup command.

So, here's what happens behind the scenes

when we take an image and turn it into a container.

First off,

the kernel is going to isolate a little section

of the hard drive

and make it available to just this container.

And so we can kind of imagine

that after that little subset is created,

the file snapshot inside the image is taken

and placed into that little segment of the hard drive.

And so, now,

inside of this very specific grouping of resources,

we've got a little section of the hard drive

that has just Chrome and Python installed

and essentially, nothing else.

The startup command is then executed

which we can kind of imagine

this case is like startup Chrome,

just Chrome for me.

And so Chrome is invoked,

we created a new instance of that process

and that created process is then isolated

to this set of resources inside the container.

So, that's pretty much it.

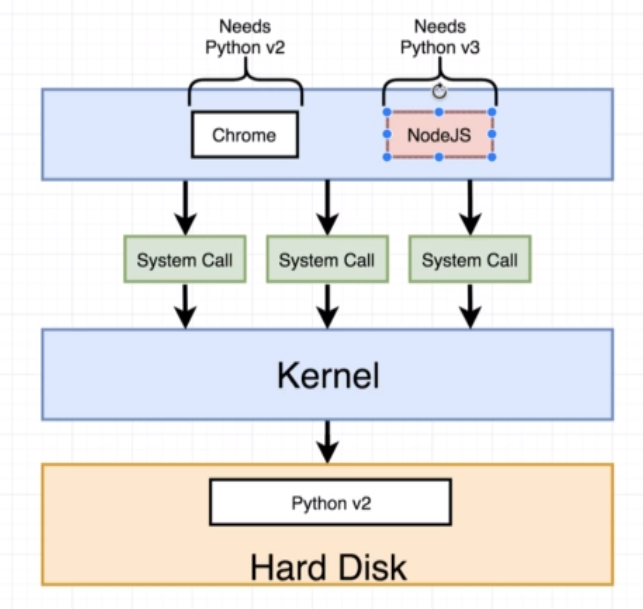
That is the relationship between a container and an image,

and it's how an image is eventually taken

and turned into a running container.

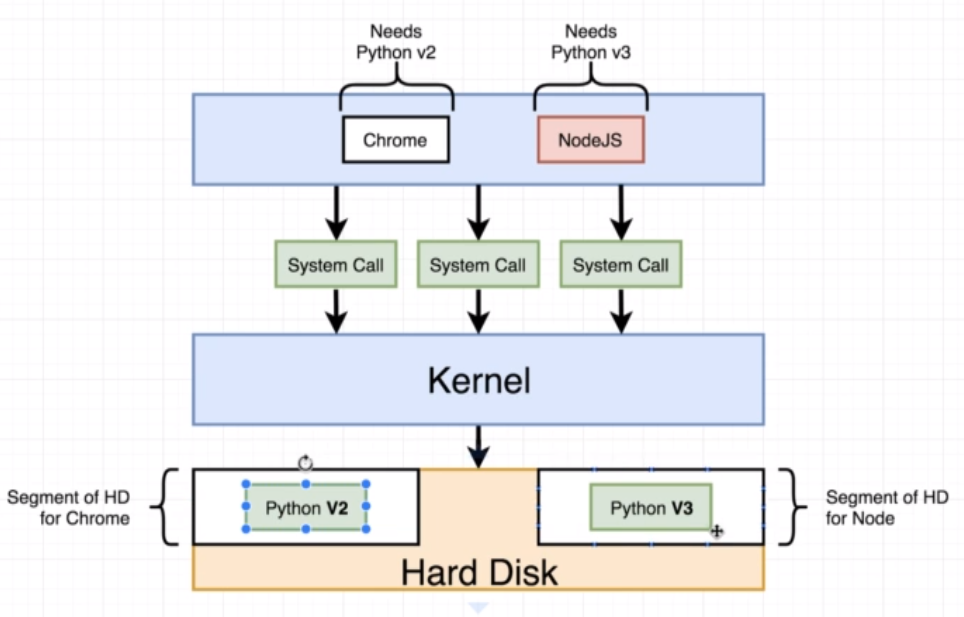
Now, there's still a tremendous amount more to learn

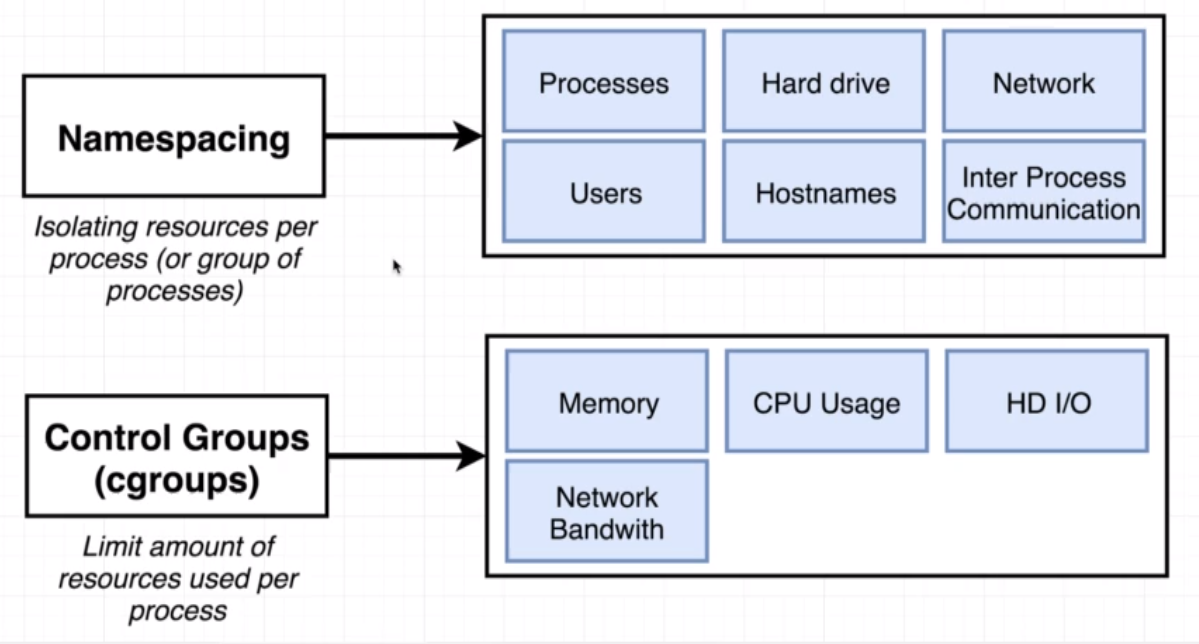
about containers and images.



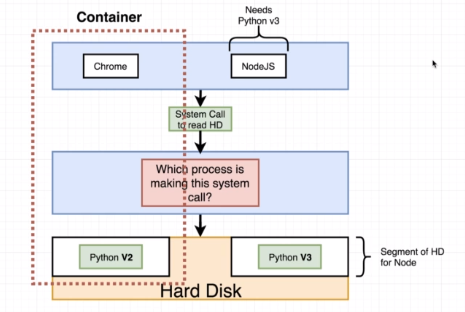
Solution:

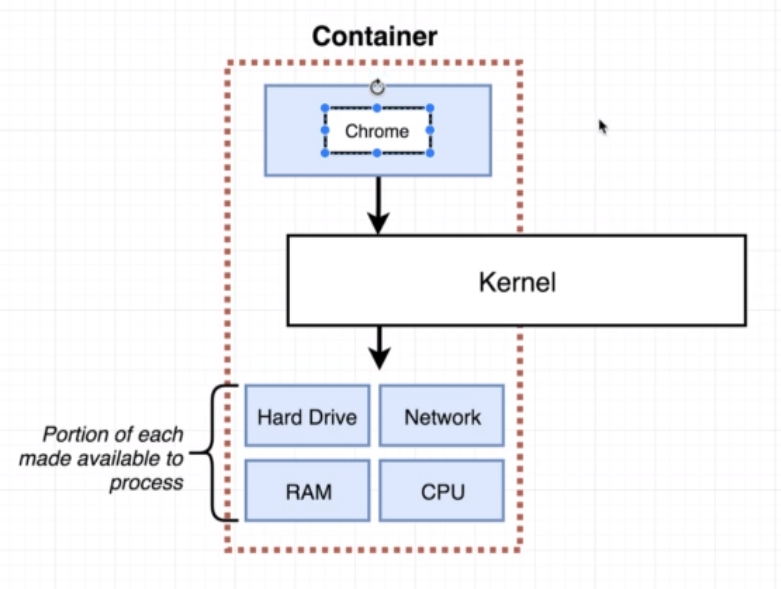
1. namespacing

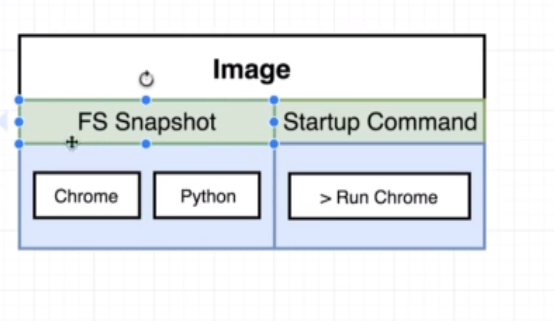




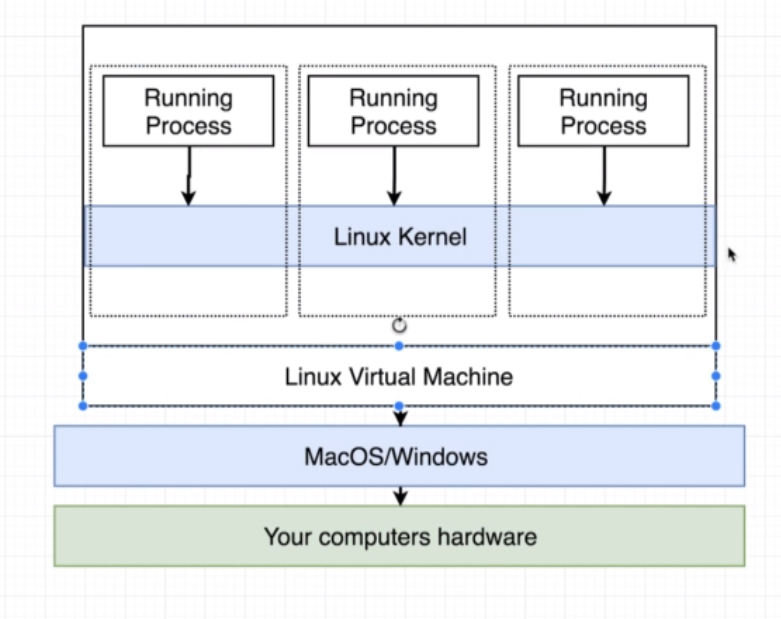
1. Control groups
2. Containers





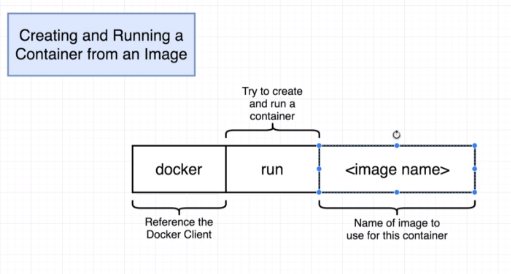


## 11.How’s Docker Running on Your Computer?

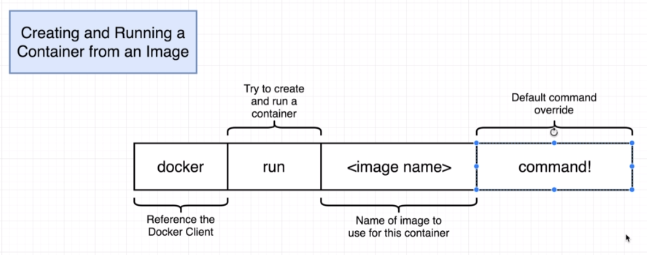


# **II] Manipulating Containers with Docker Client**

## 12. Docker Run in Detail



## 13. Overriding Default Commands



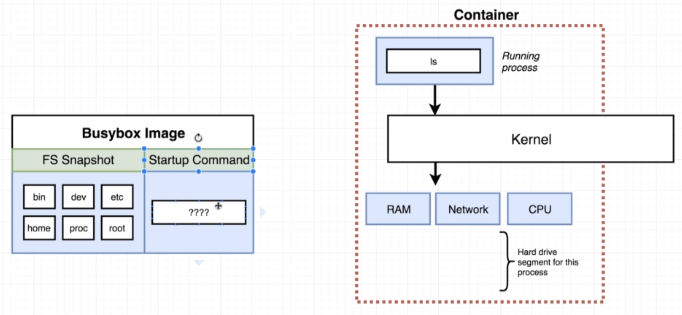
Docker run busybox ls

O/p: etc

Root

Etc

Var & so on



## 14. Listing Running Containers



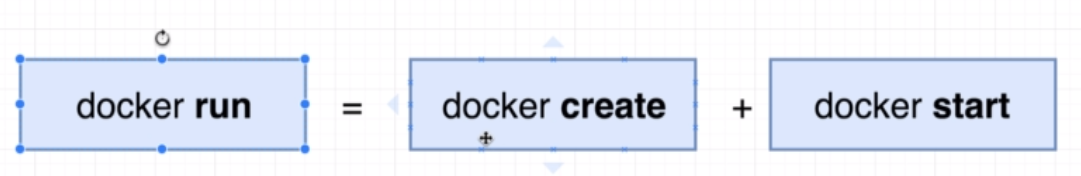
Docker run busybox ping google.com

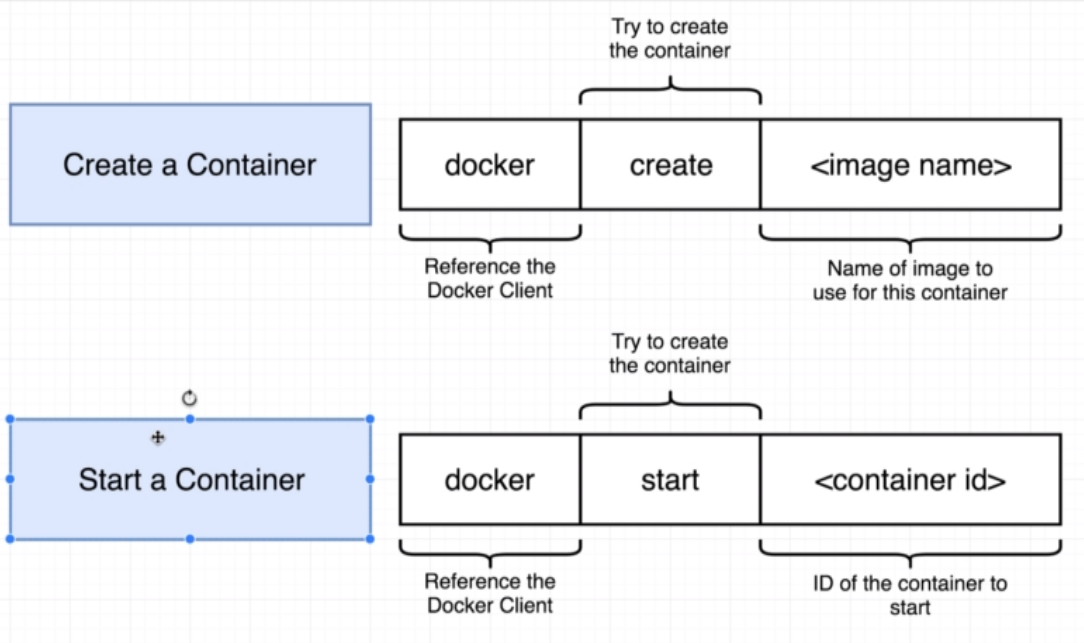
Docker ps



## 15. Container Lifecycle

Docker ps –all





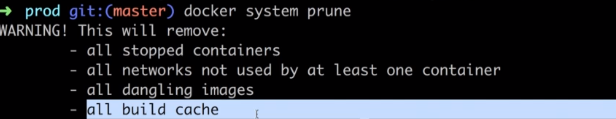
Docker start -a hello-world 🡺 will show output on container

Docker start hello-world

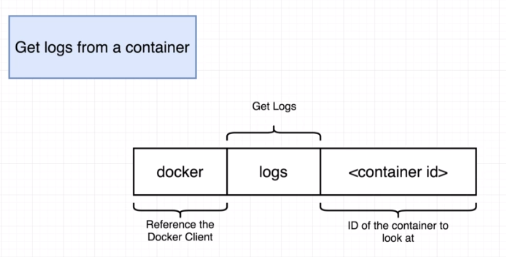
## 16. Restarting Stopped Containers

## 17. Removing Stopped Containers

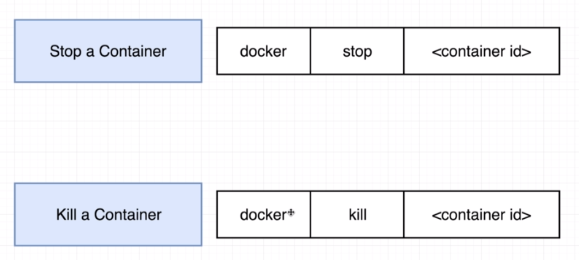
Docker system prune 🡺

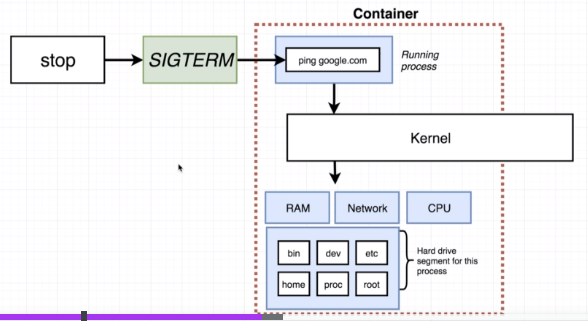


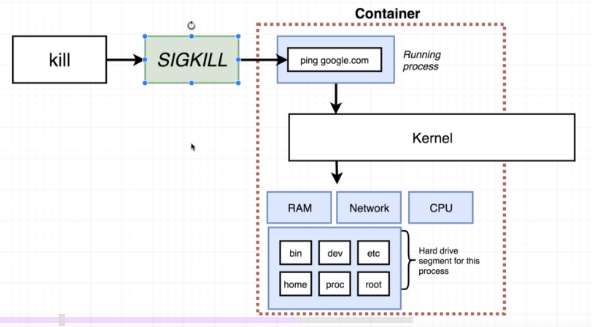
## 18. Retrieving Log Outputs



## 19. Stopping Containers



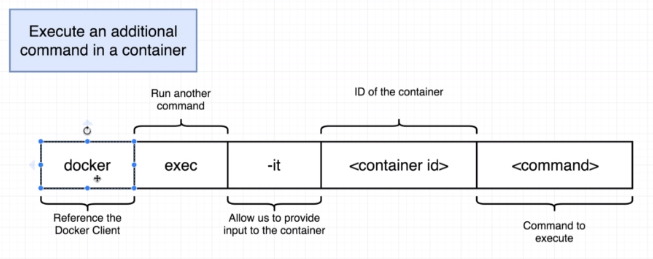


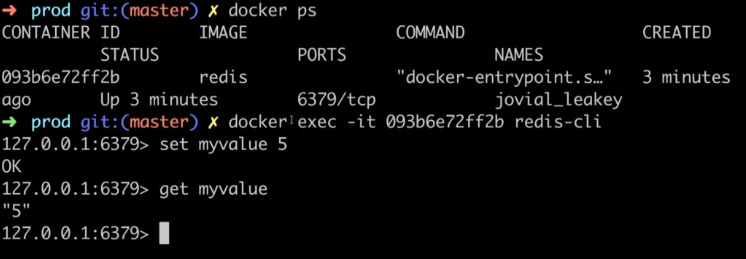


Kill instantly stops

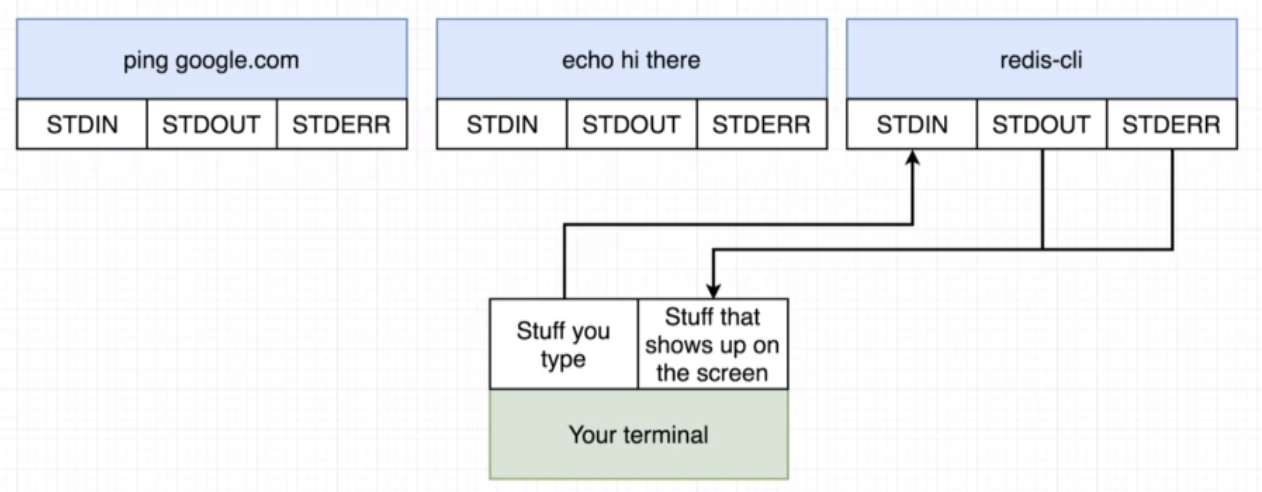
## 20. Multi-Command Containers

## 21. Executing Commands in Running Containers

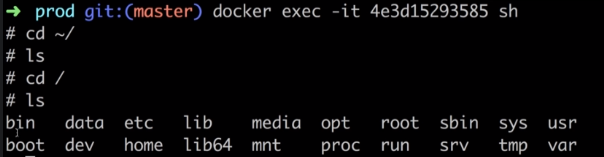


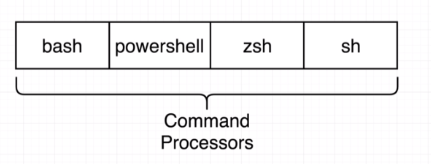


## 22. The Purpose of IT Flag



## 26. Getting a Command Prompt in a Container



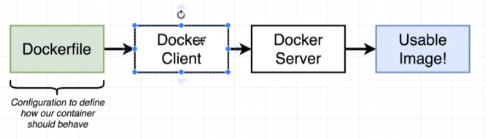


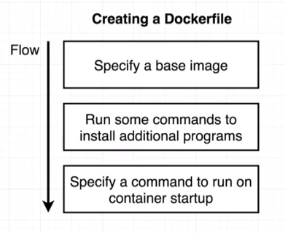
## 27. Starting with a Shell

## 28. Container Isolation

# **III] Building Custom Images Through Docker Server**

## 29. Creating Docker Images





## 30. Buildkit for Docker Desktop

## 31. Building a DockerFile

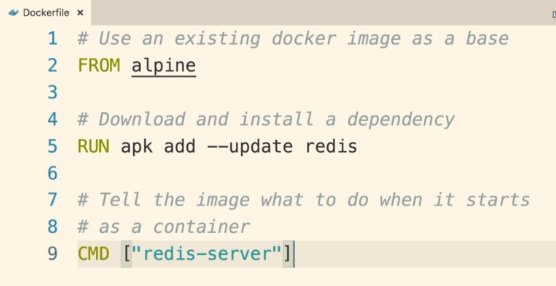
## Mkdir redis-image

Cd redis-image

Use vs-code

Open above dir

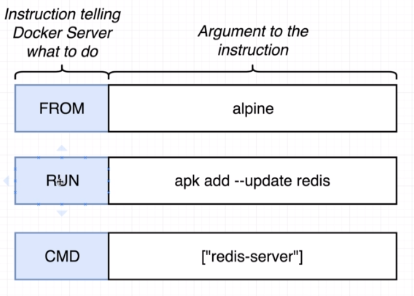
**Dockerfile**



Docker build . 🡺 ashdsagdhe73837

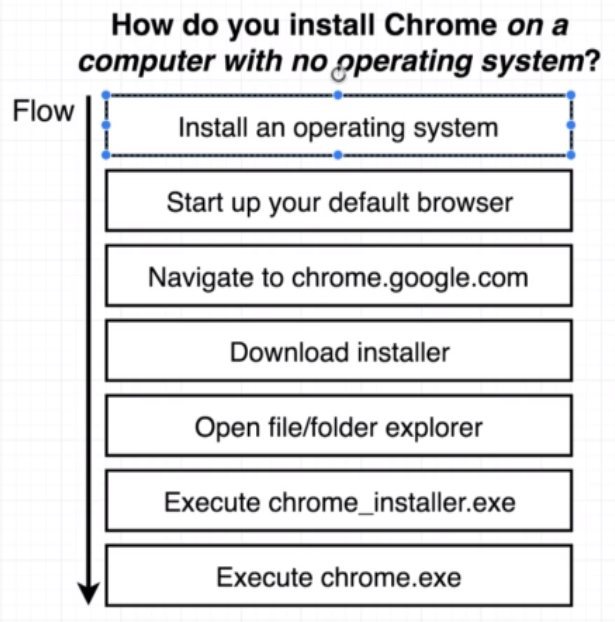
Docker run ashdsagdhe73837

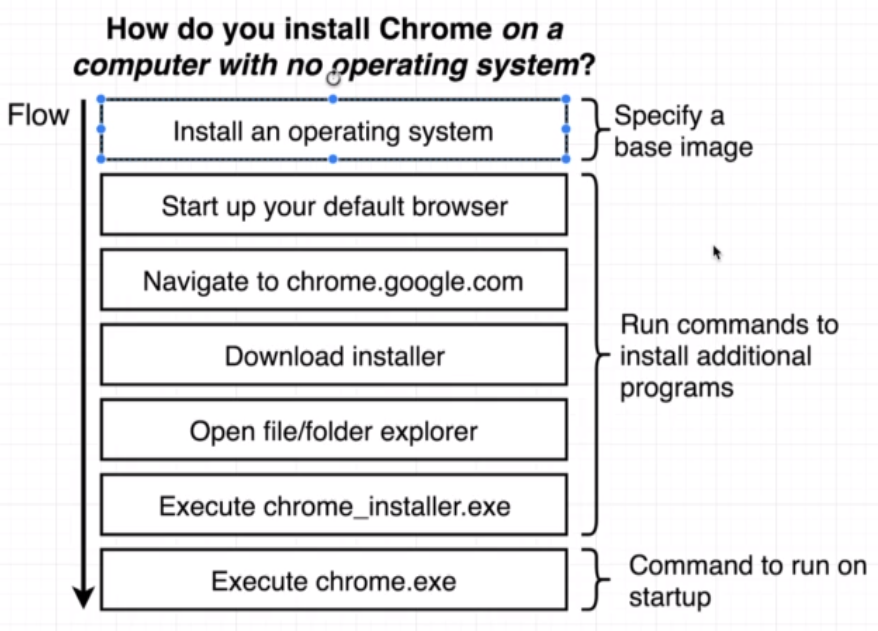
## 32. Dockerfile Teardown

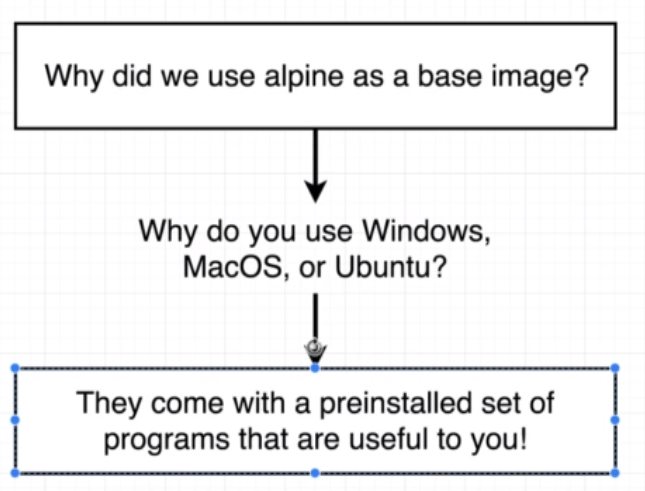


## 33. What’s a Base Image?



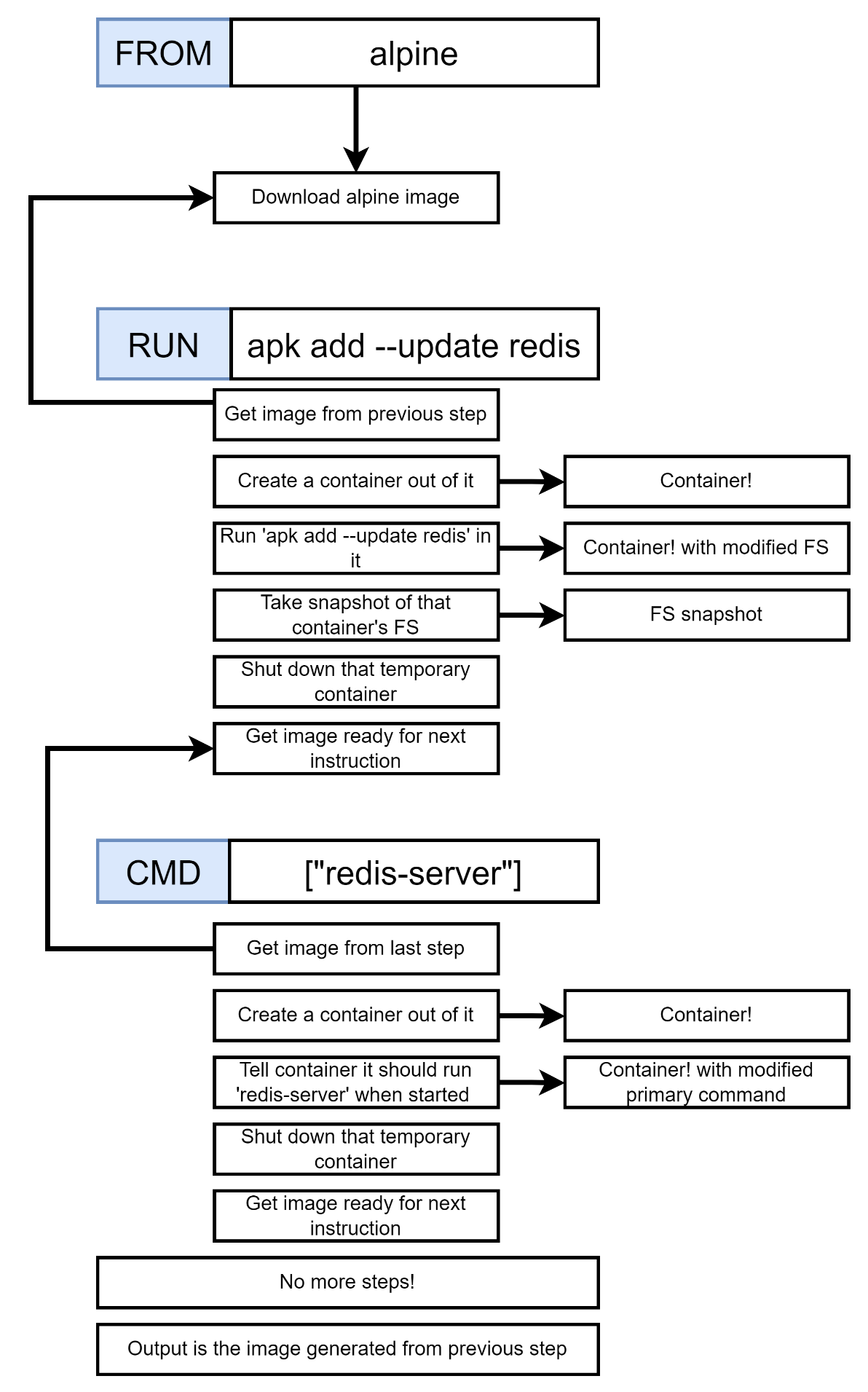






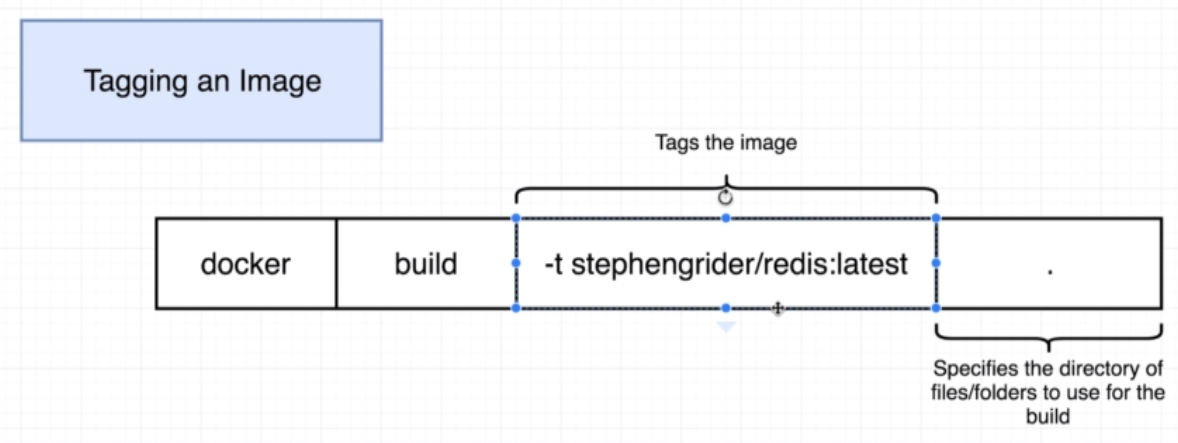
## 34. The Build Process in Detail

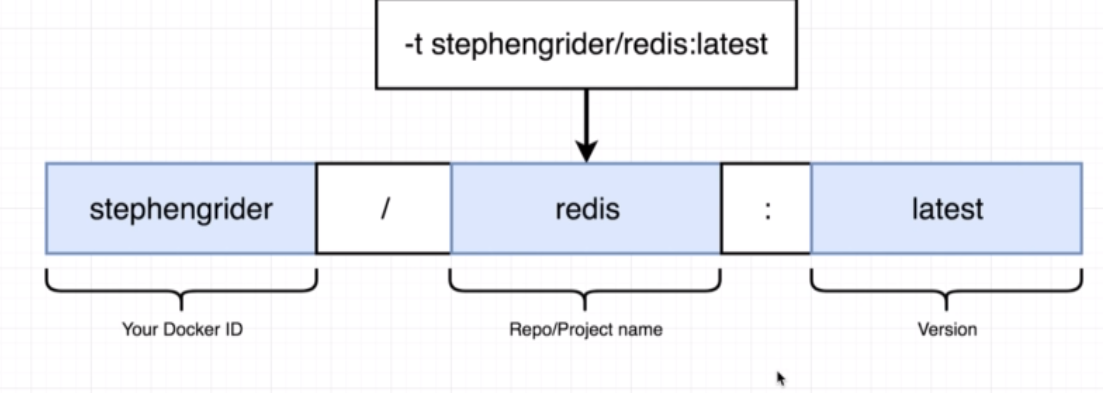
## 35. A Brief Recap



## 36. Rebuilds with Cache

## 37. Tagging an Image



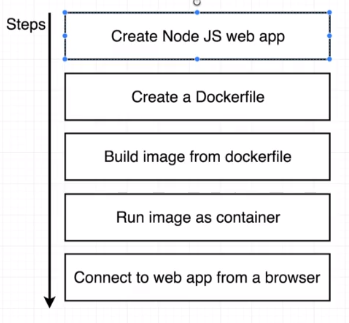


## 38. Quick Note for Windows Users

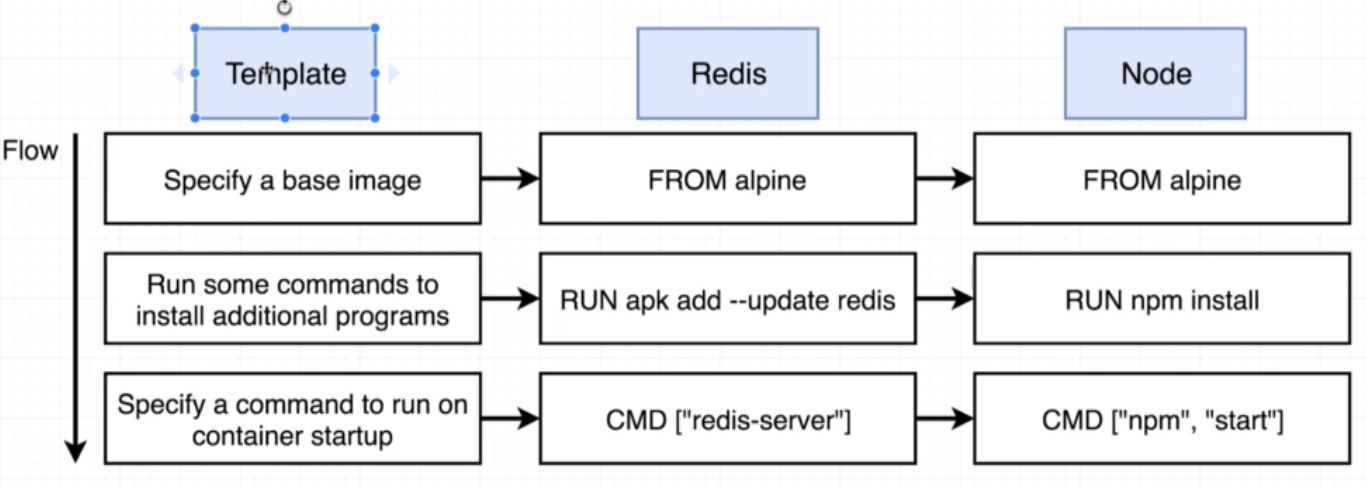
## 39. Manual Image Generation with Docker Commit

# **IV] Making Real Projects with Docker**

## 40. Project Outline



## 41. Node Server Setup



## 42. A Few Planned Errors

n the upcoming lecture, we will be adding a Node base image to our Dockerfile. To properly follow along with the lectures, please add this specific version:

Change this:

FROM node:alpine

to this:

FROM node:14-alpine

If you do not specify a Node version

If you do not specify a version, you will meet a number of errors caused by changes in the newest versions of Node:

npm ERR! idealTree already exists

This can be resolved by adding a WORKDIR right after the FROM instruction: (we will be adding this in the Specifying a Working Directory lecture anyway):

FROM node:alpine

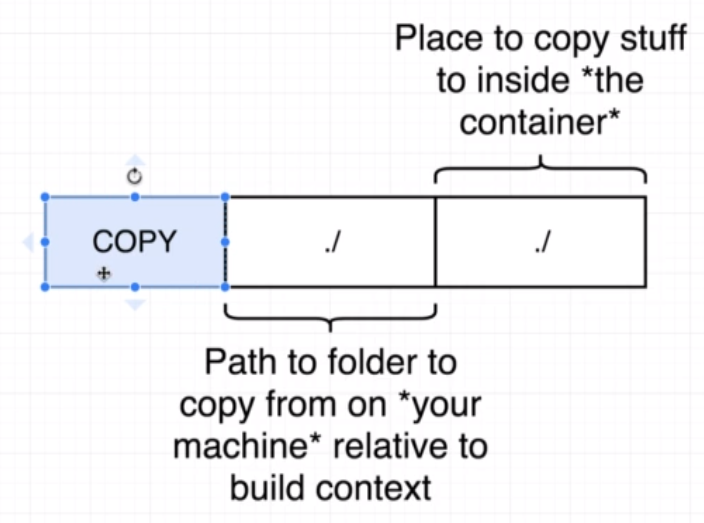
WORKDIR /usr/app

## 43. A required Node Base Image Version

## 44. Base Image Issues

## 45. A Few Missing Files

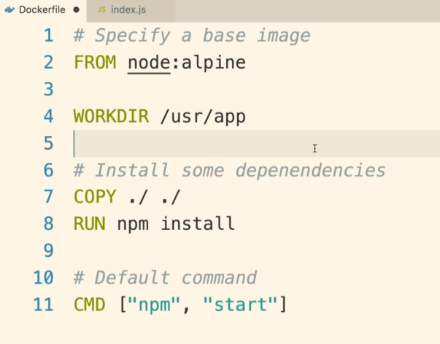
## 46. Copying Build Files



## 47. Container Port Mapping

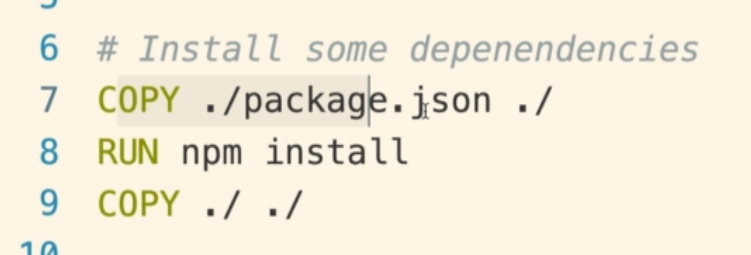


## 48. Specifying a Working Directory



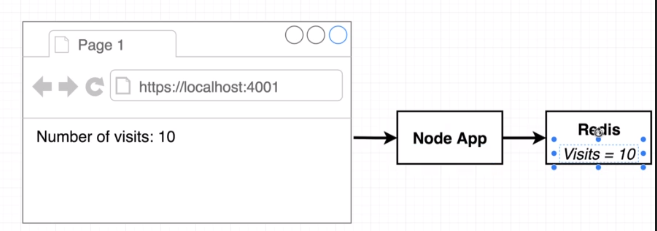
## 49. Unnecessary Rebuilds

## 50. Minimizing Cache Busting and Rebuilds



# **V] Docker Compose with Multiple Local Containers**

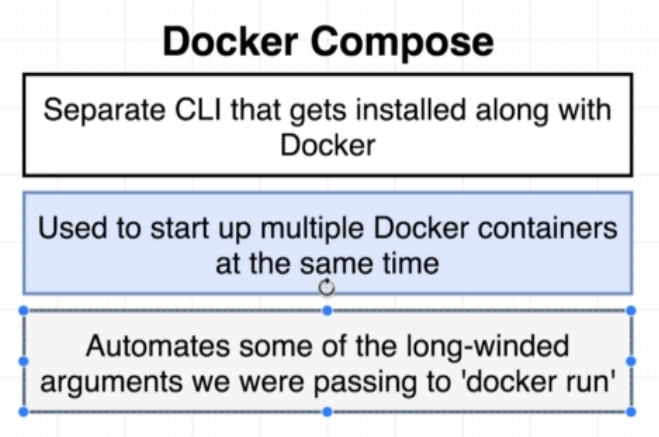
## 51. App Overview



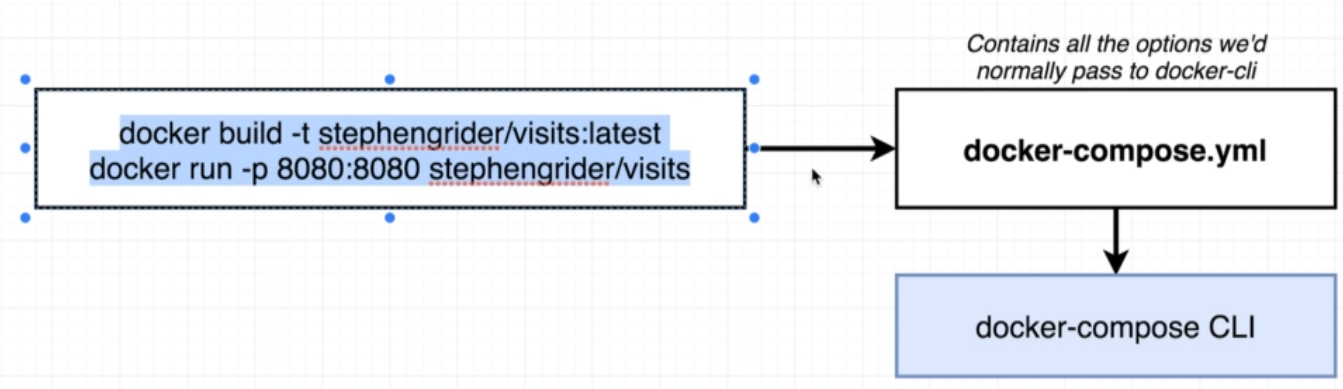
## 52. App Server Starter Code

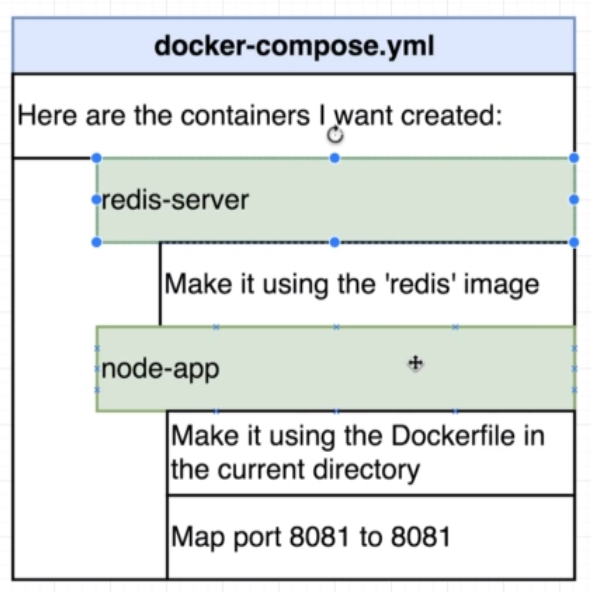
## 53. Assembling a Dockerfile

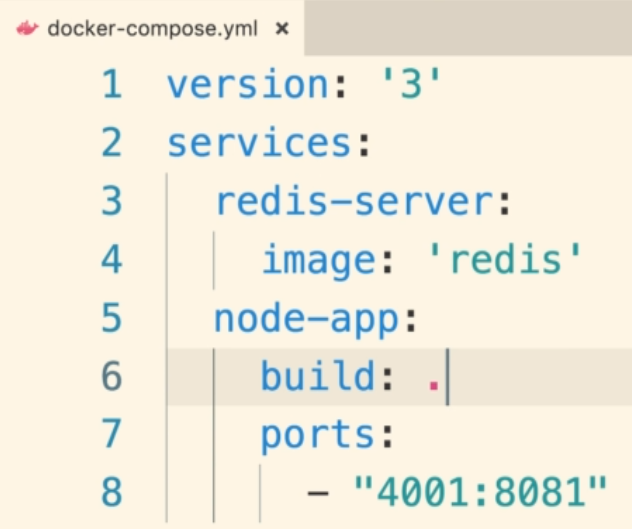
## 54. Introducing Docker Compose



## 55. Docker Compose Files

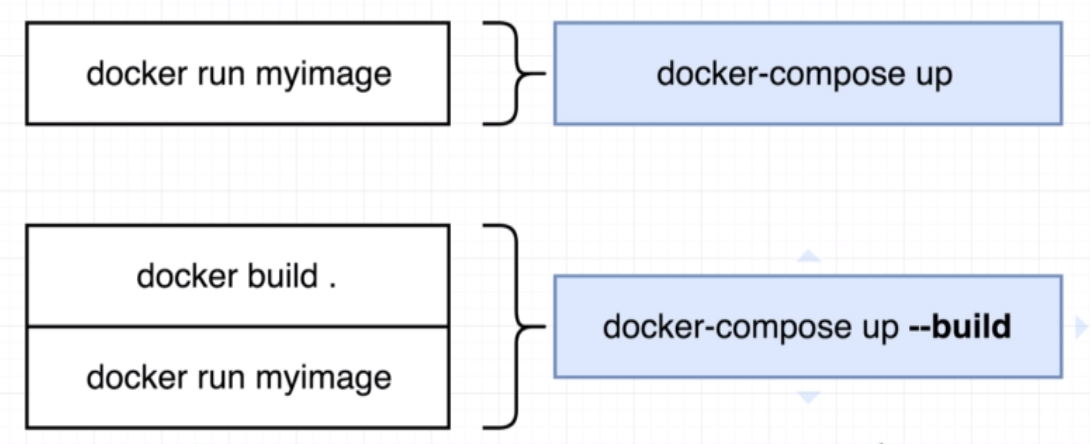






## 56. Networking with Docker Compose

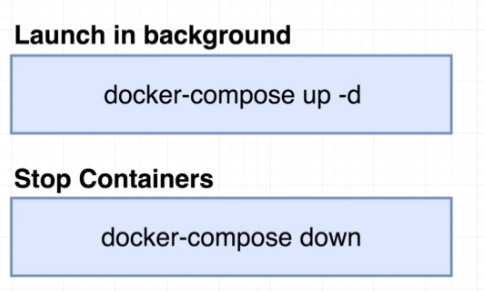
## 57. Docker Compose Commands



Docker-compose.yml in pwd

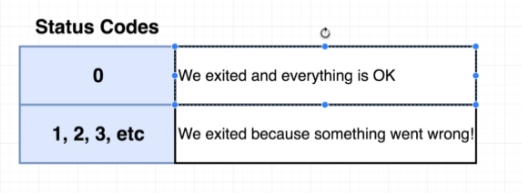
Docker-compose up

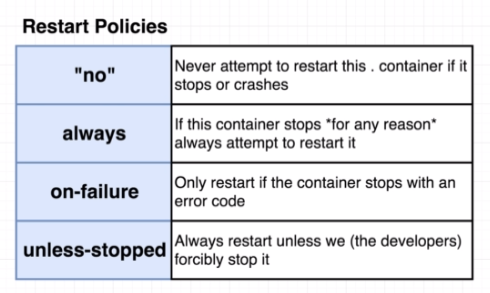
## 58. Stopped Docker Compose Containers



## 59. Container Maintenance with Docker Compose

## 60. Automatic Container Restarts

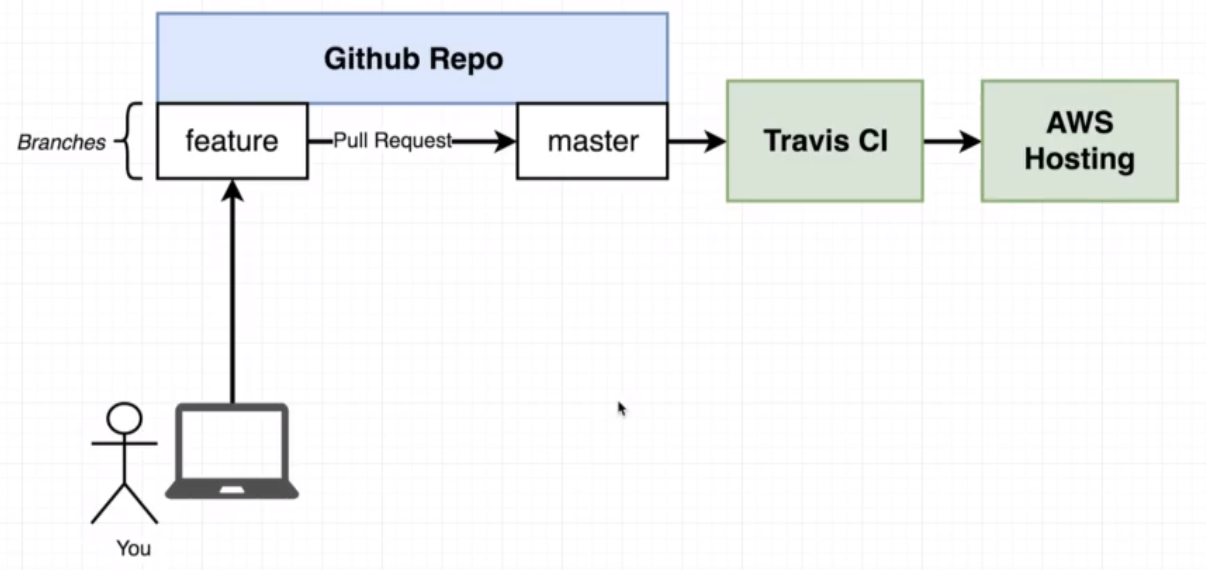


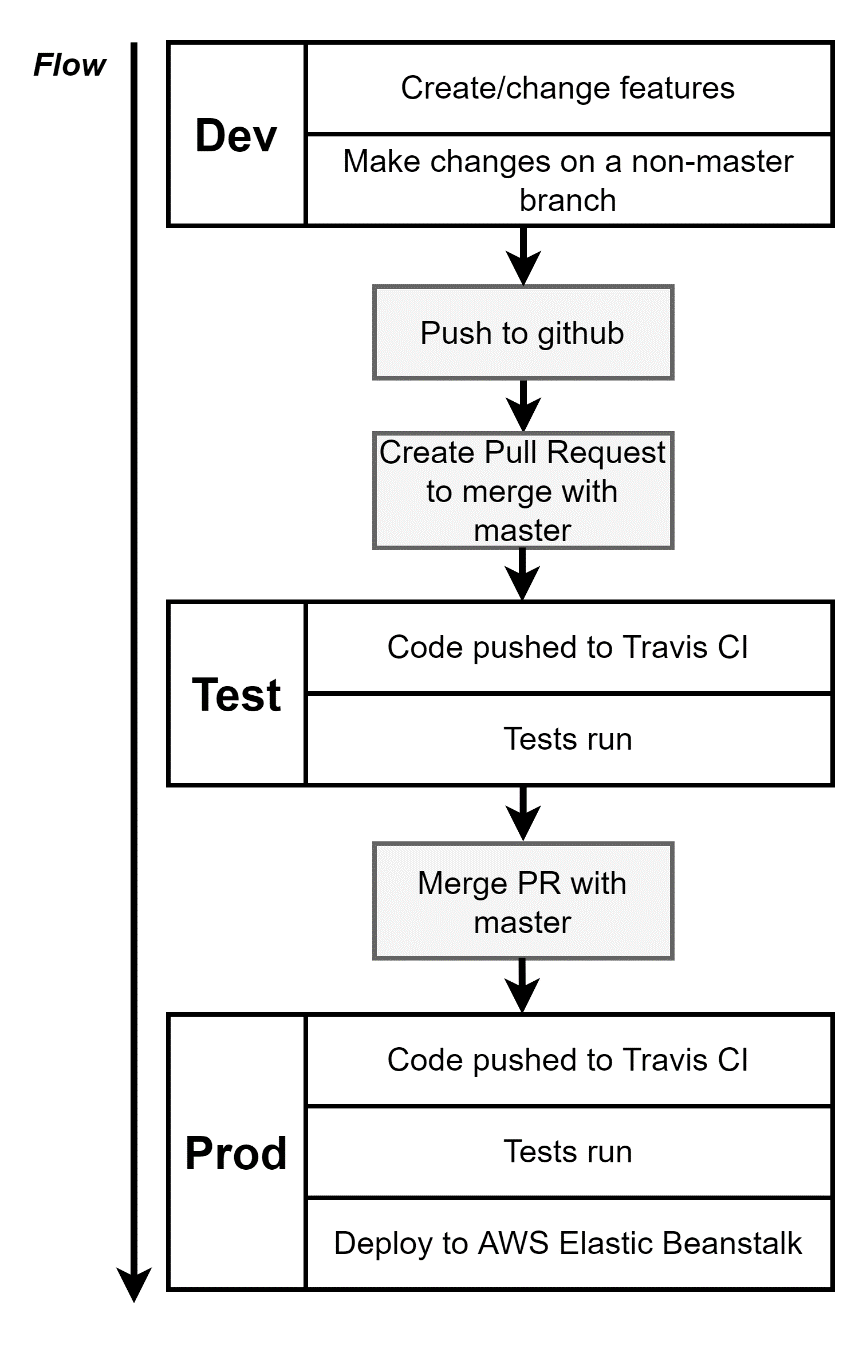


## 61. Container Status with Docker Compose

# **VI] Creating A Production-Grade Workflow**

## 62. Development Workflow





## 63. Flow Specifics

## 64. Docker’s Purpose

## 65. Project Generation

## 66. Create React App Generation

Instead of this:

npm install -g create-react-app

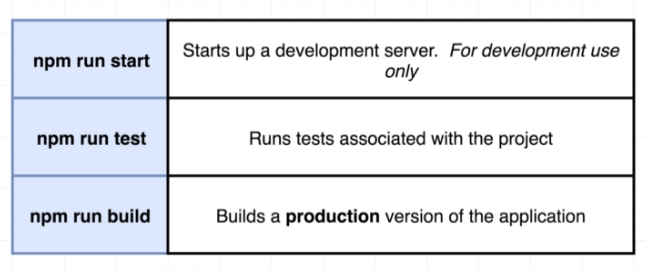
create-react-app frontend

We need to run this command:

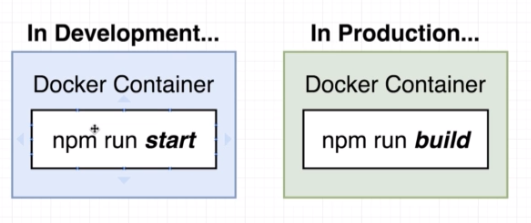
npx create-react-app frontend

## 67. More on Project Generation

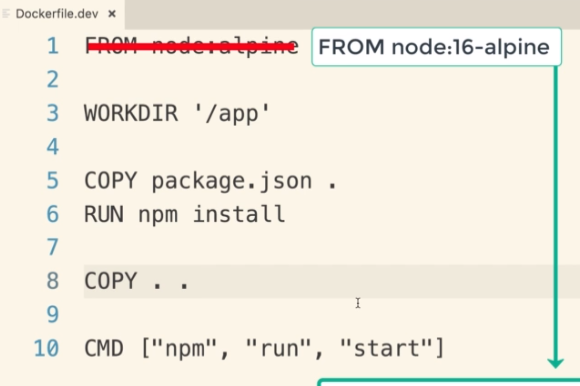
## 68. Necessary Commands



## 69. creating the Dev Dockerfile



Dockerfile.dev

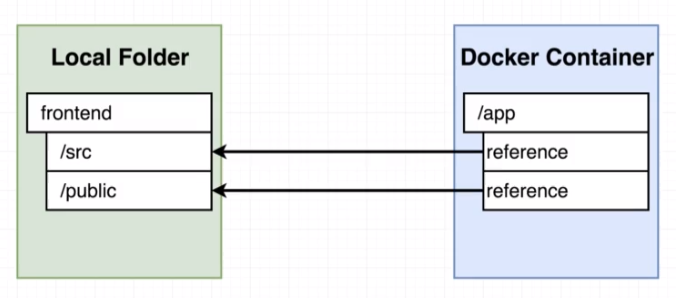


Docker build **-f** Dockerfile.dev

## 70. Duplicating Dependencies

## 71. Starting the Container

## 72. Docker Volumes





## 73. WSL2 and Windows users note

## 74. Bookmarking Volumes

## 75. Shorthand with Docker Compose

## 76. Overriding Dockerfile Selection

## 77. Do we need Copy?

## 79. Executing Tests

## 80. Docker Compose for Running Tests

## 81. Shortcomings on testing

## 82. Need for Nginx

## 83. Multi-step Docker Builds

## 84. Implementing Multi-Step Builds

## 85. Running Nginx

# **VII] Continuous Integration and Deployment with AWS**

## 86. Services Overview

## 87. Github Setup

## 88. Imp Info about Travis and Account Registration

## 89. Travis CI Setup

## 90. Travis YML File Config

## 91. Required Travis Script updates

## 92. A Touch More Travis Setup

## 93. Automatic build Creation

## 94. Required Updates for Amazon Linux 2 Platform

## 95. AWS Elastic Beanstalk

## 96. More On Elastic Beanstalk

## 97. Travis Config for Deployment

## 98. Travis Keys Update

## 99. Automated Deployments

## 100. Exposing Ports Through Dockerfile

## 101. Workflow with Github

## 102. Redeploy on Pull Request Merge

## 103. Deployment Wrap up

## 104. Environment Cleanup

## 105. AWS Config Cheat Sheet

## 106. Finished Project with Updates Applied

# **VIII] Building a Multi-Container Application**

## 107. Single Container Deployment Issues

## 108. Application Overview

## 109. A Quick Note

## 110. Application Architecture

## 111. Worker Process Setup

## 112. Express API Setup

## 113. Imp Update for Table Query

## 114. Connecting to Postgres

## 115. More Express API Setup

## 116. Create React App Generation

## 117. Generating React App

## 118. Fetching Data in The React app

## 119. Rendering Logic in the App

## 120. Exporting the Fib Class

## 121. Routing in the React App

# **IX] “Dockerizing” Multiple Services**

## 122. Checkpoint Files

## 123. Checkpoint Catchup

## 124. Dockerizing a React App

## 125. DockerizingGeneric Node Apps

## 126. Adding Postgres as a Service

## 127. Docker-Compose Config

## 128. Postgres Database Required Fixes and Updates

## 129. Environment Variables with Docker Compose

## 130. Required Worker Environment Variables

## 131. The Worker and Client Services

## 132. Nginx Path Routing

## 133. Routing with Nginx

## 134. Building a Custom Nginx Image

## 135. Starting up Docker Compose

## 136. Nginx connect() failed – Connection refused while connecting to upstream

## 137. Troubleshooting Startup Bugs

## 138. Websocket Connection to ‘ws://localhost:3000/ws’ failed

## 139. Opening Websocket Connections

# **X] A Continuous Integration Workflow for Multiple Images**

## 140.

## 141.

## 142.

## 143.

## 144.

## 145.

## 146.

## 147.

## 148.

## 149.

## 150.

# **XI] Multi-Container Deployments to AWS**

## 151.

## 152.

## 153.

## 154.

## 155.

## 156.

## 157.

## 158.

## 159.

## 160.

## 161.

## 162.

## 163.

## 164.

## 165.

## 166.

## 167.

## 168.

## 169.

## 170.

## 171.

## 172.

## 173.

## 174.

## 175.

## 176.

# **XII] Onwards to Kubernetes**

## 177. The Why’s and What’s of Kubernetes

## 178. Kubernetes in Development and Production

## 179. Docker Desktop’s Kubernetes Setup and Production - MacOS

## 180. Docker Desktop’s Kubernetes Setup and Production - Windows

## 181. Updated Minikube Install and Setup Info - MacOS

## 182. Minikube Setup - MacOS

## 183. Minikube Setup – Windows

## 184. Minikube Setup - Linux

## 185. Mapping Existing Knowledge

## 186. Quick Note to prevent error

## 187. Adding Config Files

## 188. Object Types and API versions

## 189. Running Container in Pods

## 190. Service Config Files in Depth

## 191. Connecting to Running Containers

## 192. The Entire Deployment Flow

## 193. Imperative vs Declarative Deployments

# **XIII] Maintaining Sets of Containers with Deployments**

## 194.

## 195.

## 196.

## 197.

## 198.

## 199.

## 200.

## 201.

## 202.

## 203.

## 204.

## 205.

## 206.

## 207.

## 208.

## 209.

## 210.

## 211.

# **XIV] A Multi-Container App with Kubernetes**

## 212.

## 213.

## 214.

## 215.

## 216.

## 217.

## 218.

## 219.

## 220.

## 221.

## 222.

## 223.

## 224.

## 225.

## 226.

## 227.

## 228.

## 229.

## 230.

## 231.

## 232.

## 233.

## 234.

## 235.

## 236.

## 237.

## 238.

## 239.

## 240.

## 241.

# **XV] Handling Traffic with Ingress Controllers**

## 242.

## 243.

## 244.

## 245.

## 246.

## 247.

## 248.

## 249.

## 250.

## 251.

## 252.

## 253.

## 254.

## 255.

# **XVI] Kubernetes Production Deployment**

## 256.

## 257.

## 258.

## 259.

## 260.

## 261.

## 262.

## 263.

## 264.

## 265.

## 266.

## 267.

## 268.

## 269.

## 270.

## 271.

## 272.

## 273.

## 274.

## 275.

## 276.

## 277.

## 278.

## 279.

## 280.

## 281.

## 282.

## 283.

## 284.

## 285.

## 286.

## 287.

## 288.

## 289.

## 290.

## 291.

## 292.

## 293.

## 294.

## 295.

## 296.

## 297.

## 298.

# **XIV] HTTPS Setup with Kubernetes**

## 299.

## 300.

## 301.

## 302.

## 303.

## 304.

## 305.

## 306.

## 307.

## 308.

## 309.

## 310.

## 311.

## 312.

## 313.

## 314.

## 315.

## 316.

# **XVIII] Local Development with Skaffold**

## 317.

## 318.

## 319.

## 320.

## 321.

## 322.

## 323.

# **XIX] Extras**

## 324.