Objectives Achieved

Website maintenance initially done with VM's  and Hashi Corp's Vagrant tool,  but found that maintaining virtual machines  on local machines was much more effort  than using Docker and Docker Compose.

To bridge the gap between production  and development a website is containerized with Docker and Docker Compose along with using RSpec, Capybara and Selenium  to do quick unit and integration tests  that can run anywhere at any time.

Terraform is used to demonstrate how rapidly we can deploy a working instance  of this website into the cloud lastly kept this all together  by writing a CICD pipeline with Jenkins  to build test and deploy website.

Steps to achieve the above objectives:

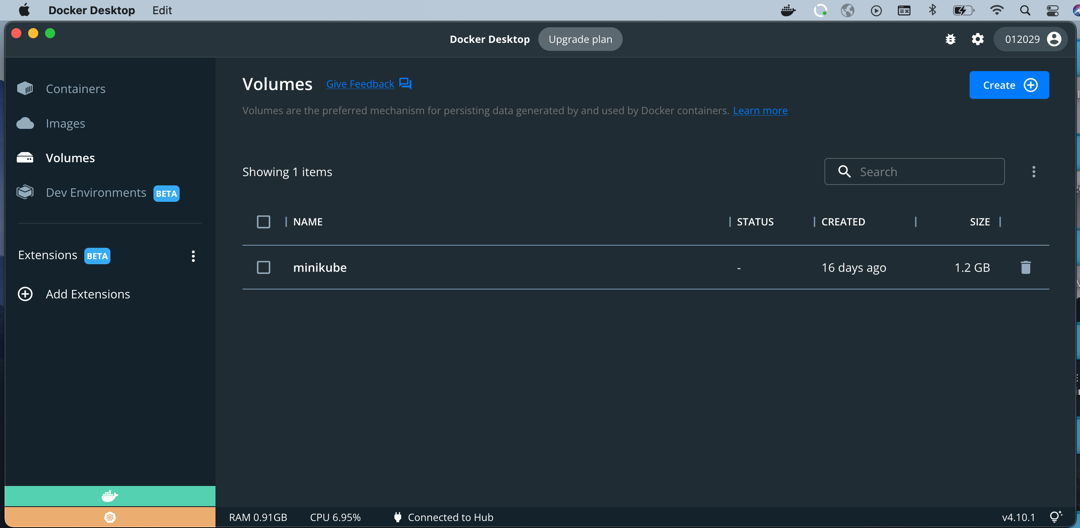
1. As I am using Mac machine installed the brew utility with the following command at terminal:

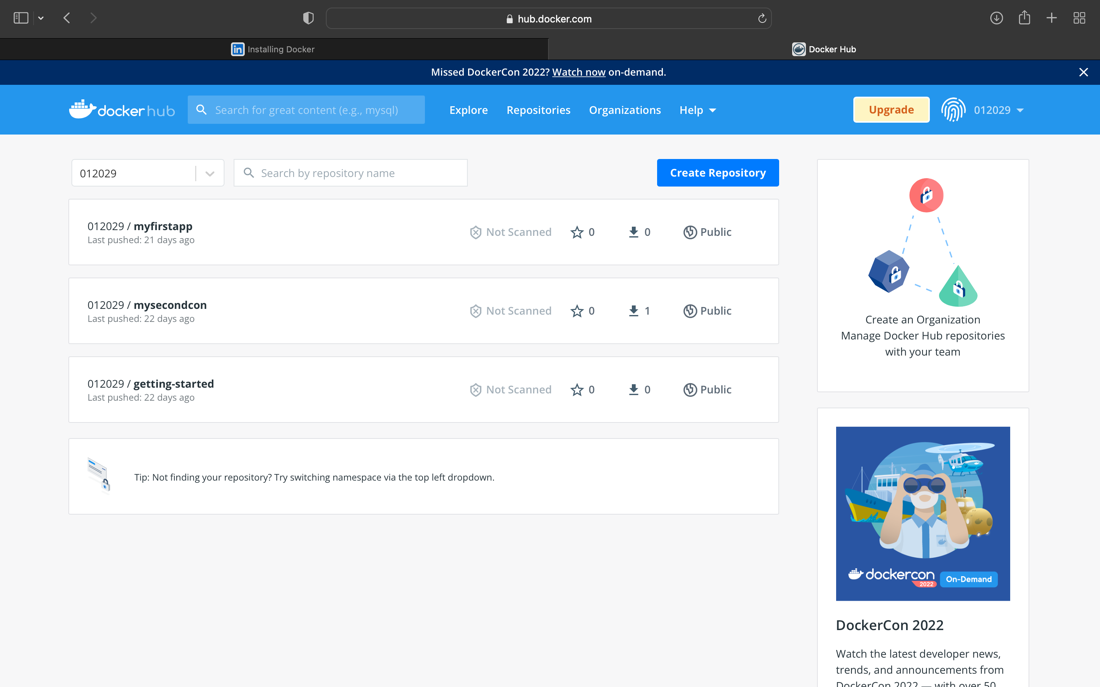
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"

1. Once brew is installed successfully docker is installed by typing the command:

brew cask install docker

1. Docker is running …





1. Initial docker file created is : (Reference https://docs.docker.com/engine/reference/builder)

**Dockerfile**

#base image selection

FROM nginx:alpine

#copying the website files and images

COPY website /website

#copying the configuration

COPY nginx.conf /etc/nginx/nginx.conf

# port used run the container

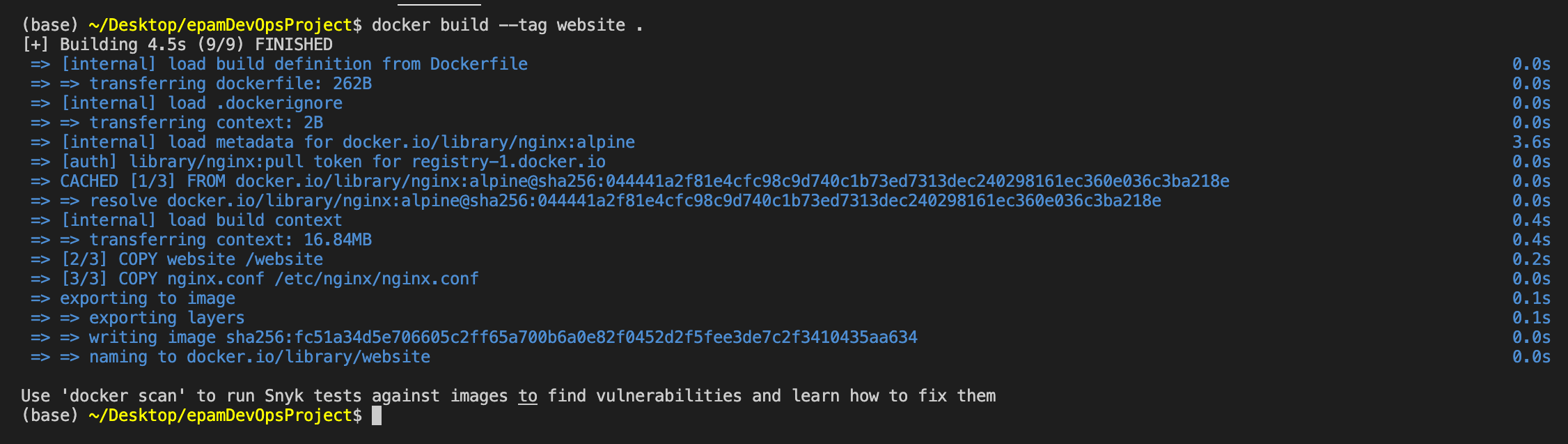
EXPOSE 80

1. The contents of the working directory are :

(base) ~/Desktop/epamDevOpsProject$ ls

Dockerfile nginx.conf website

1. Docker image is created is created by running the command :



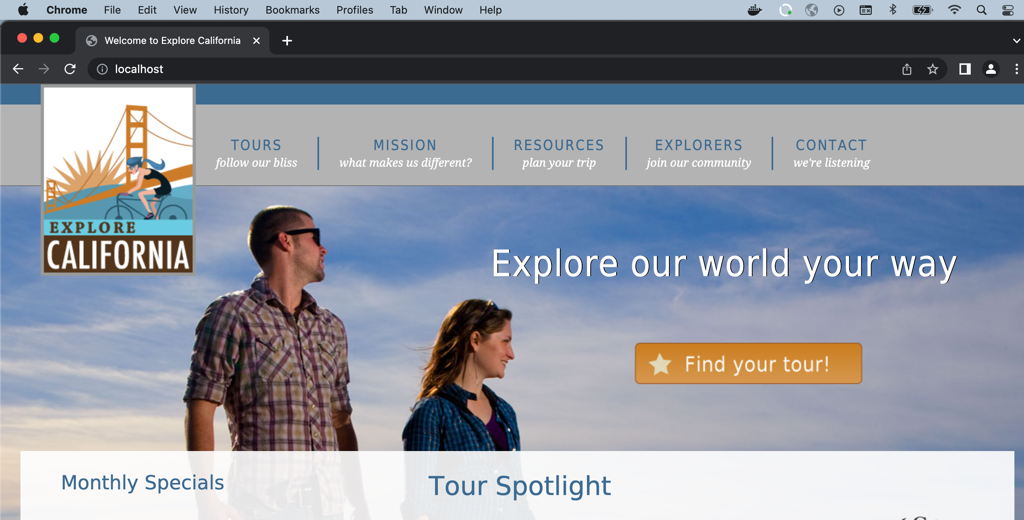
1. So created a docker image, build it and ran a container from it.

We have a docker file that when built  runs a local copy of our Explore California website  which is very similar to production environment.  We can use docker commands to do that.  We can use docker build to build our docker file  and create an image from it.  And docker run to run that image  and expose the port to make it necessary  to access the website locally.

The website output from the container is obtained with the command:

docker run --publish 80:80 website

Website viewed from the browser is shown below:



1. Docker compose is a simple light weight platform for running multiple container applications in a single stack. But here like Dockerfile we use docker composed manifest listing all the services etc. Docker compose utility is installed in step 1 and is available for use.

The docker compose manifest file name “docker-compose.yml” is created and as shown below:

#include latest docker-compose version number

version: '3.7'

#website container service that runs my website container image

services:

website:

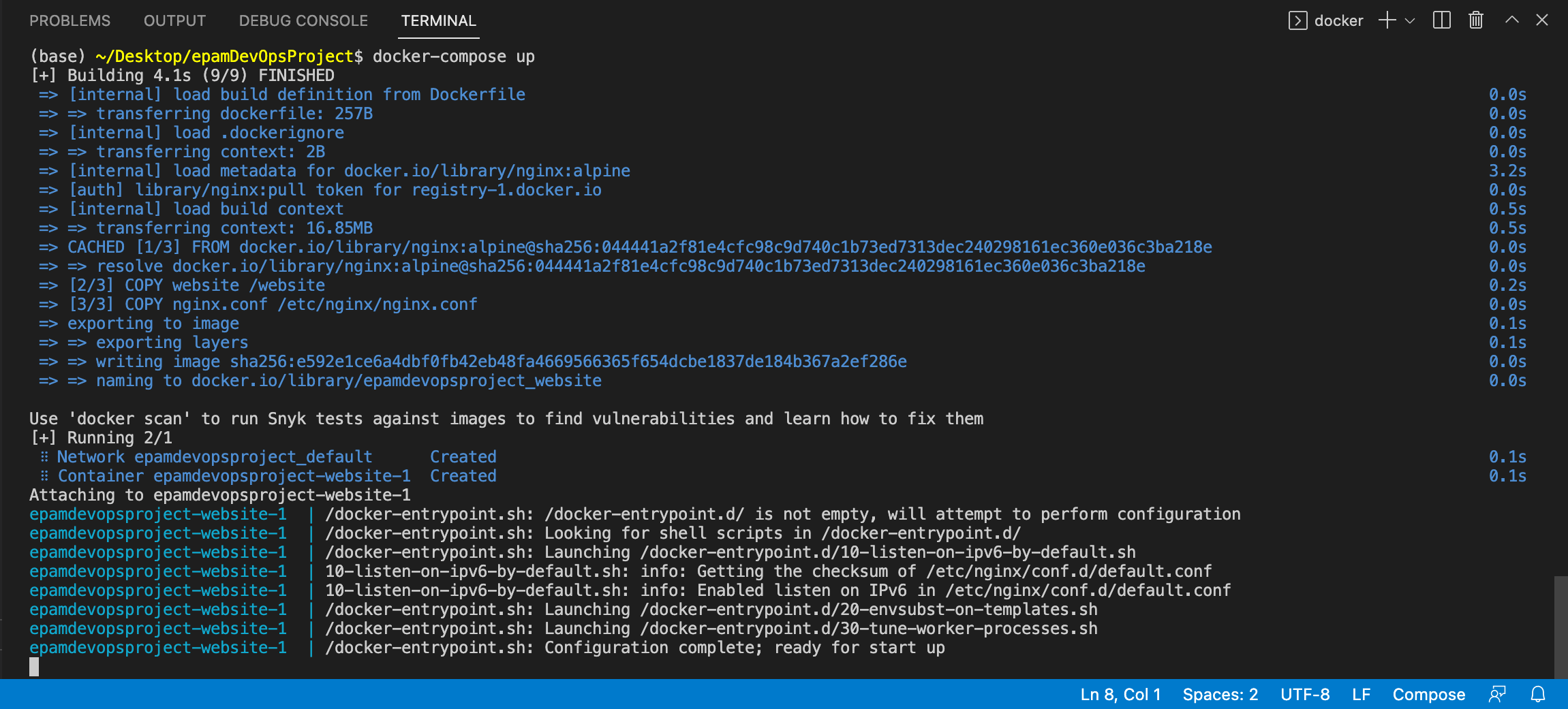
build:

context: .

ports:

- 80:80

1. Run the command “docker-compose up” in turn will run the website by mapping the ports automatically. The website can be viewed in the browser as all the services listed in the manifest file are started.



(base) ~/Desktop/epamDevOpsProject$ docker-compose ps

NAME COMMAND SERVICE STATUS PORTS

epamdevopsproject-website-1 "/docker-entrypoint.…" website exited (0)

1. Now that website works inside a docker and will do some automated tests with tools RSpec, Capybara and Selenium for it by modifying the “docker-compose.yml” as below:

version: '3.7'

services:

website:

build:

context: .

ports:

- 80:80

unit-tests:

build:

dockerfile: rspec.dockerfile

context: .

volumes:

- $PWD:/app

command:

- --pattern

- /app/spec/unit/\*\_spec.rb

A separate folder “spec” containing folders for unit and integration tests.

“rspec.dockerfile” contents:

FROM ruby:alpine

MAINTAINER Carlos Nunez <dev@carlosnunez.me>

RUN apk add --no-cache build-base ruby-nokogiri

RUN gem install rspec capybara selenium-webdriver

ENTRYPOINT [ "rspec" ]

1. Updated unit test “page\_spec.rb”

require 'capybara'

require 'capybara/dsl'

require 'selenium-webdriver'

include Capybara::DSL

Capybara.app\_host = "http://website" # Using Selenium; connect over network

Capybara.run\_server = false # Disable Rack since we are using Selenium.

Capybara.register\_driver :selenium do |app|

Capybara::Selenium::Driver.new(

app,

browser: :remote,

url: "http://#{ENV['SELENIUM\_HOST']}:#{ENV['SELENIUM\_PORT']}/wd/hub",

desired\_capabilities: Selenium::WebDriver::Remote::Capabilities.chrome(

"chromeOptions" => {

"args" => ['--no-default-browser-check']

}

)

)

end

Capybara.default\_driver = :selenium

describe "Example page render unit tests" do

it "Shows the Explore California logo" do

visit('/')

expect(page.has\_selector? '.logo').to be true

end

end

1. To the unit tests run the following command:

docker-compose up -d website

docker-compose run --rm website unit-tests

(base) ~/Desktop/epamDevOpsProject$ docker-compose run --rm unit-tests

1 examples found.

Finished in 0.00052 seconds (files took 0.10818 seconds to load)

1 examples, 0 failures

1. Update the “docker-compose.yml” with selenium driver

version: '3.7'

services:

selenium:

image: selenium/standalone-chrome-debug

ports:

- 4444:4444

- 5900:5900

website:

build:

context: .

ports:

- 80:80

unit-tests:

build:

dockerfile: rspec.dockerfile

context: .

environment:

SELENIUM\_HOST: selenium

SELENIUM\_PORT: 4444

volumes:

- $PWD:/app

entrypoint: rspec

command:

- --pattern

- /app/spec/unit/\*\_spec.rb

1. Now run the container with following command:

docker-compose up -d website selenium

base) ~/Desktop/epamDevOpsProject$ docker-compose up -d website selenium

[+] Running 2/2

⠿ Container epamdevopsproject-selenium-1 Started 0.8s

⠿ Container epamdevopsproject-website-1 Started 0.7s

1. Run the unit-tests once again using the vnc client to run the tests against browser.

Explore California website has a Docker file  that allows us to run our website locally for testing. Those tests are written with our spec  and run as well in Docker and Rspec uses Capybara  and Selenium to talk to a real browser  and actually run those tests against a real browser  and test elements from it.  And done all of this in a Docker compose network  all defined in the Docker compose manifest.

Here comes a need for tool that allows us  to provision infrastructure easily  and also keep track of that infrastructure and to do that is through Terraform.  Terraform is a Golang based tool that allows us to deploy  any sort of infrastructure into any environment.  So we want AWS Terraform can communicate with AWS  but just as easily, it can deploy stuff into Azure,  Google Cloud or Bluemix or any other provider  that you may use or want to use it  for deploying Explorer California into it.

1. Create a file named “terraform.Dockerfile” to the project folder with the contents shown below:

FROM alpine

#to fetch the terraform binary from internet

RUN wget -O /tmp/terraform.zip https://releases.hashicorp.com/terraform/0.12.9/terraform\_0.12.9\_linux\_amd64.zip && \

unzip /tmp/terraform.zip -d /

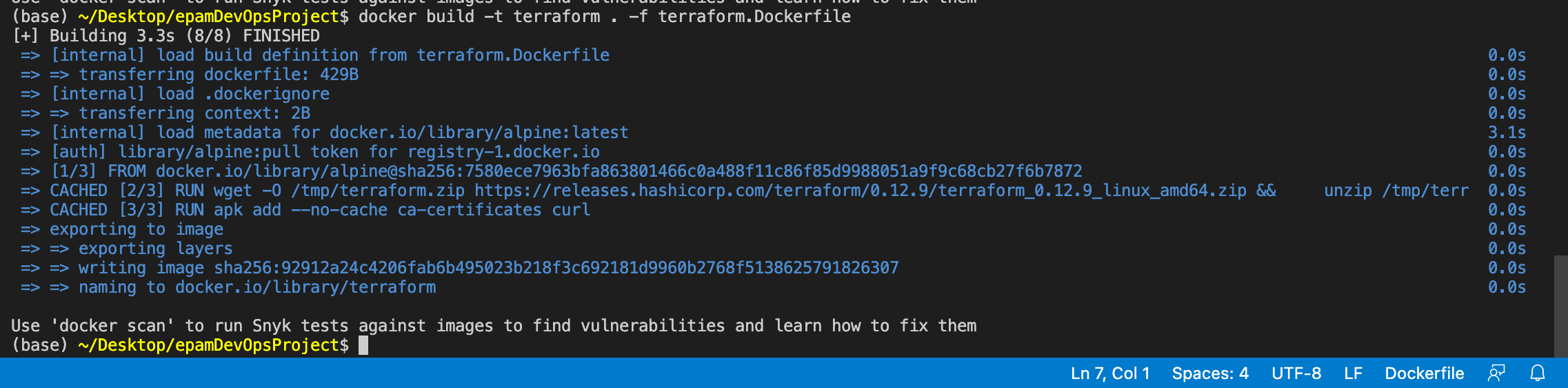
RUN apk add --no-cache ca-certificates curl

USER nobody

ENTRYPOINT [ "/terraform" ]

1. Now build the image by using command:

docker build -t terraform .

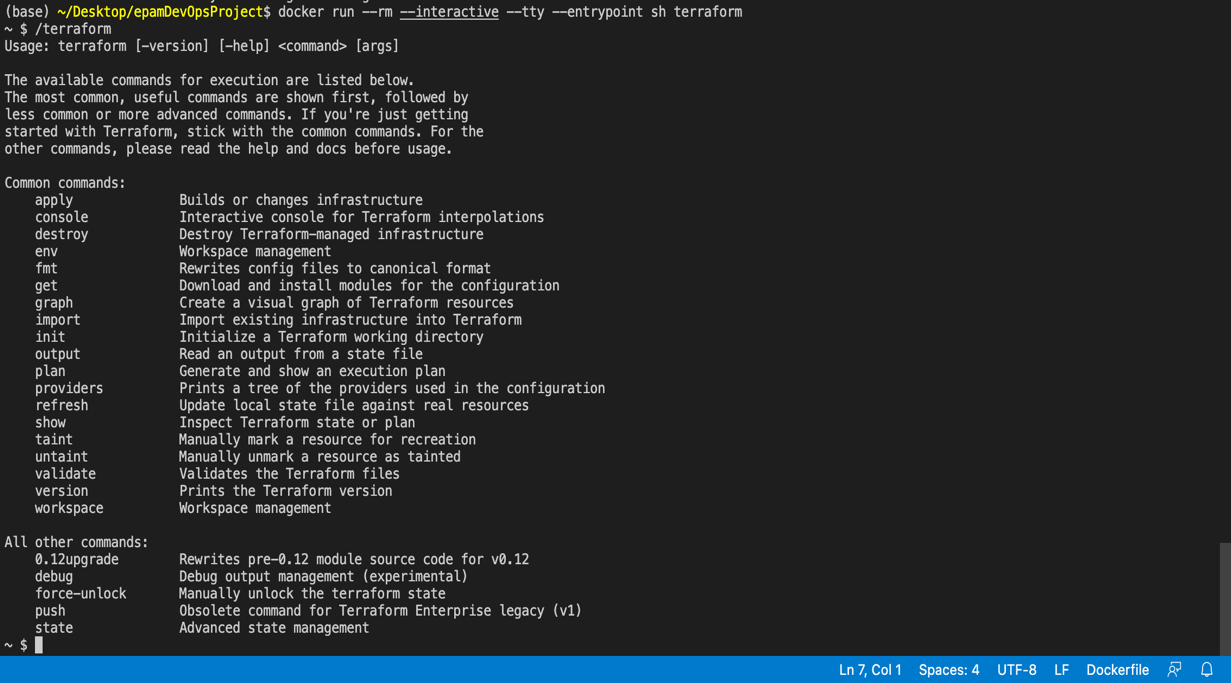


And tested using the command

(base) ~/Desktop/epamDevOpsProject$ docker run --rm --interactive --tty --entrypoint sh terraform

~ $

The above output shows and says working fine with terraform image created.



1. As the terraform image is built with necessary context run the image in container using the command :

docker run --rm terraform

1. The updated “docker-compose.yml” is liste below:

version: '3.7'

services:

terraform:

build:

dockerfile: terraform.Dockerfile

context: .

environment:

AWS\_ACCESS\_KEY\_ID: "${AWS\_ACCESS\_KEY\_ID}"

AWS\_SECRET\_ACCESS\_KEY: "${AWS\_SECRET\_ACCESS\_KEY}"

AWS\_REGION: "${AWS\_REGION}"

volumes:

- $PWD:/app

working\_dir: /app

selenium:

image: selenium/standalone-chrome-debug

ports:

- 4444:4444

- 5900:5900

website:

build:

context: .

ports:

- 80:80

unit-tests:

build:

dockerfile: rspec.dockerfile

context: .

environment:

SELENIUM\_HOST: selenium

SELENIUM\_PORT: 4444

volumes:

- $PWD:/app

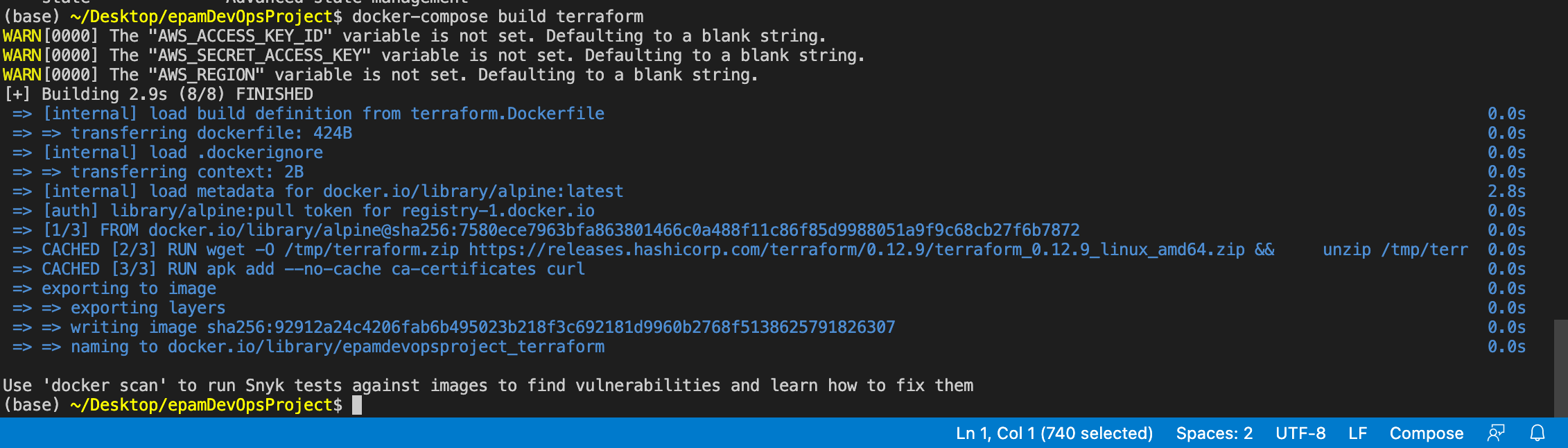
entrypoint: rspec

command:

- --pattern

- /app/spec/unit/\*\_spec.rb

1. The status of terraform image using docker-compose is shown below



Version : Terraform v0.12.9

1. Now the image created terraform will be deployed into AWS.
2. Create the terraform file “main.tf” with following content for creating the infrastructure in AWS.

data "aws\_iam\_policy\_document" "bucket\_policy" {

statement {

sid = "PublicReadGetObject"

effect = "Allow"

actions = [ "s3:GetObject" ]

principals {

type = "\*"

identifiers = [ "\*" ]

}

resources = [ "arn:aws:s3:::explorecalifornia.org/\*" ]

}

}

/\* We can access properties from data sources using this format:

${data.<data\_source\_type>.<data\_source\_name>.<property>.

In this case, we need the JSON document, which the documentation

says can be accessed from the .json property. \*/

resource "aws\_s3\_bucket" "website" {

bucket = "explorecalifornia.org" // The name of the bucket.

acl = "public-read" /\* Access control list for the bucket.

Websites need to be publicly-available

to the Internet for website hosting to

work. \*/

policy = "${data.aws\_iam\_policy\_document.bucket\_policy.json}"

website {

index\_document = "index.htm" // The root of the website.

error\_document = "error.htm" // The page to show when people hit invalid pages.

}

}

output "website\_bucket\_url" {

value = "${aws\_s3\_bucket.website.website\_endpoint}"

}

1. Now that we have created the configuration file to create the infrastructure, we need to initialize the terraform.

docker-compose run --rm terraform init

The above command initializes the backend successfully.