Docker 101: Getting to know Docker

Alexandru Rosianu @aluxian
Entrepreneurs Club

Thanks everyone for coming

I hope you'll enjoy this talk and hopefully LEARN DOCKER!!!!

$ whoami

- Student — Computer Science with AI
- Campus Ambassador + Certified Associate
- Co-founder & CTO @ futuristico.xyz
- @aluxian on Twitter, Facebook, everywhere

🔷 Geek 🤓 Student 👨‍🎓 Developer 👨‍💻 Hacker 📲

student at the uni

i signed up to be a campus ambassador, i got accepted, they offered free exams, so I also became “CERTIFIED”

i also have a startup, right now we’re playing with crypto stuff. of course we use docker there

my username is aluxian everywhere

$ whoareyou

- Why do you want to learn Docker?

I want to know what to focus on during the presentation

so can i ask you guys, why do you want to learn docker?

Agenda

and stickers

- What is Docker?
- But why?
- Basic docker commands
- The mighty Dockerfile
- Cat gifs demo
- Now you try it!
- Q&A
- Workshop
- Pizza

and t-shirts

and here's what's on the show tonight

What is Docker?

docker

some would say it’s like this… it’s pretty similar to their logo, right?

a good visualization, but docker is actually a tool that makes developers’ life easier — it helps you **build, ship and run apps**

this is actually a photo from Southampton’s docks, btw

Docker containers are NOT VMs

- Easy connection to make
- Fundamentally different architectures
- Fundamentally different benefits

docker

i’m going to tell you what containers are… by telling you what they are not. in fact, we’ll use an analogy

if you need a refresher, a VM, or virtual machine, is some piece of software that emulates a machine. think about your laptop, then imagine you have a program that pretends it’s a computer itself. then in this virtual computer you can install another operating system

containers are usually compared to VMs because it’s an easy connection to make; however, they are quite different

VMs

this is a VM; think of a house: it is fully self-contained and offers protection from unwanted guests

it also has its own infrastructure: plumbing, heating, electrical, etc

furthermore, in the vast majority of cases, houses are all going to have at a minimum a bedroom, living area, bathroom, and kitchen. It's difficult to find a "studio house"

similarly, VMs are a full copy of an operating system, with its own dedicated resources. If you decide you need a new house, you build a whole new house with its own foundation, plumbing, etc. VMs stand alone, and this leads to duplicating resources – every instance has its own full copy of the OS, for instance

finally, the smallest VMs are typically at least hundreds of MBs in size, and they take several minutes to boot up (partially due to the fact that you're starting a full copy of the OS)

Containers

and these are containers—more like apartments

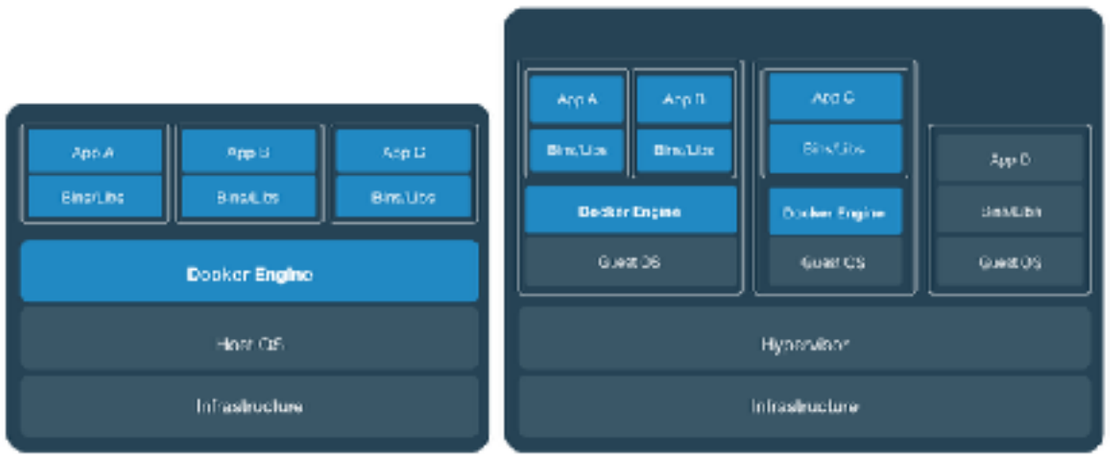
they also offer protection from unwanted guests, but they are built around shared infrastructure
- the apartment building offers shared plumbing, heating, electrical, etc. to each apartment
- additionally, apartments are offered in several sizes—from studio to multi-bedroom penthouses
- you're only renting exactly what you need

Docker containers share the underlying resources of the server they are running on. Furthermore, developers build a Docker image that includes exactly what they need to run their application: starting with the basics and adding in only what is needed by the application

containers are very small (some are less than 3MBs) and start up very quickly (less than 1s) because you're not booting a full operating system, but only a process

so something like this

They're different, not mutually exclusive



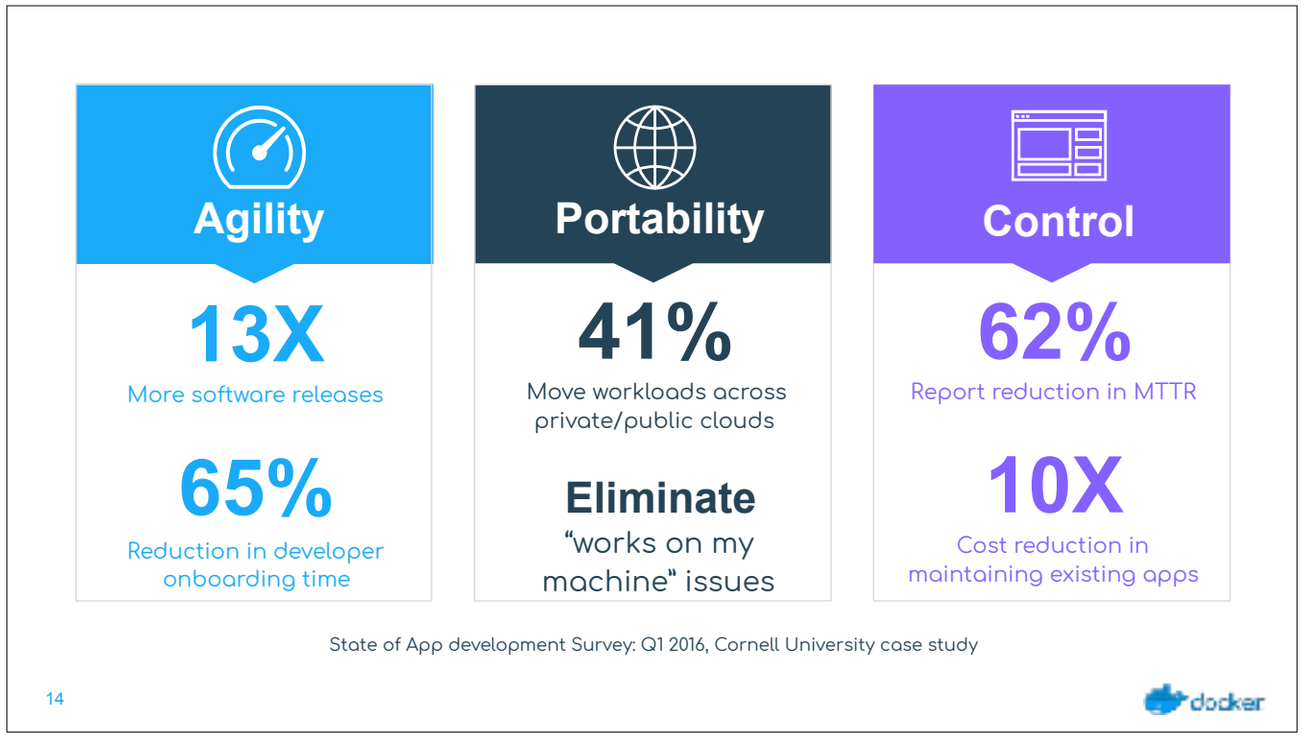
this doesn't mean they cannot co-exist, though

But why?

but… why?

you know what’s coming…

yep, another gif



the truth is that Docker is very appealing to developers and organisations

first of all, the speed and simplicity of Docker is what originally drew developers to Docker (they were able to ship more software, faster)

- ING, for example, went from shipping once every 9 months to shipping over 1,500 times a year by adopting Docker and DevOps

then there’s portability

- in companies, containerized apps can move from dev to test, and ultimately to production without incidents
- no more finger pointing or “works on my machine” issues, because the app and its dependencies packaged together in a self-contained and independent unit
- what this also means is that if you want to run e.g. MongoDB, you don’t need to install it directly on your machine. Just download the container image
- if you want to run something like ownCloud, which has several components (a database, a web server, some storage controller), or any software that’s complicated to install and set up, Docker makes this process a little easier

Build, Ship, Run — that’s Docker’s slogan

essentially, it boils down to the fact that developers can choose whatever languages or components they want to build their application (.NET, Java, Ruby, Redis, etc)

then wherever that code is deployed, Physical/virtual/cloud, it makes no difference

an application written on a developer’s laptop will run exactly the same on a large server, as on a Raspberry Pi

in other words…

docker

… it’s so you can feel like this

Basic docker commands

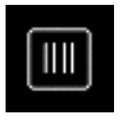
ok, enough theory

don’t leave yet, now it’s getting more fun

Some Docker vocabulary

Docker Image

The basis of a Docker container, represents a full application



Docker Container

The standard unit in which the application resides and executes

Docker Engine

It creates and runs Docker containers

Registry Service (Docker Hub or Docker Trusted Registry)

A storage and distribution service for your images

docker

you know by now what a container is:

Container
- the “standard unit” in which the application service resides, where
- the app and its dependencies sit together in an isolated environment

Image
- the image is like a template, or starting point, from which containers are created
- they contain EVERYTHING an application needs to run, and
- should always be built via a Dockerfile (which we’ll talk about in a bit)

Docker Engine
- it’s the program that creates images, runs containers from images
- it also communicates with Docker Hub, where you download most images from, and you can even push your own

Registry
- Docker Hub is an image registry, i.e.
- a service that stores and distributes container images
- it’s like GitHub for git, or like npmjs.com for npm

Basic docker Commands

```
$ docker pull aluxian/catweb:1.0

$ docker images

$ docker run -d -p 5000:5000 --name catweb aluxian/catweb:latest

$ docker ps

$ docker stop catweb (or <container id>)

$ docker rm catweb (or <container id>)

$ docker rmi aluxian/catweb:latest (or <image id>)

$ docker build -t aluxian/catweb:2.0 .

$ docker push aluxian/catweb:2.0
```


now let's see how you can actually use **docker**

docker pull pulls an image from the registry to the local host

docker images will list all the images on your docker host

docker run will start a new container
- in this case we are instructing the docker engine to run the aluxian/catweb image we pulled earlier
- **-d** tells docker to start the application in detached mode (running in the background)
- **-p 5000:5000** tells docker engine that any requests coming into port 5000 on the host should be directed to port 5000 on this container
- **-name catweb** specifies a name for our running container
- if you do not specify a name, Docker will generate one (and they can be pretty funny)
- finally, we specify the image from which we're running the container

docker ps shows running containers

docker stop stops a running container, but does not delete it

docker rm deletes a stopped container

docker rmi removes an image from your host

docker build will create a new image from a Dockerfile
- in this example we are creating an image called aluxian/catweb and we're tagging it with a 2.0 version number
- the period essentially says to build the image from the current directory

docker push is the opposite of Docker pull — it pushes an image up to a registry

The mighty Dockerfile

i mentioned a “Dockerfile” at least twice until now

Dockerfile: Linux example

```
1  # our base image
2  FROM alpine:latest
3
4  # Install python and pip
5  RUN apk add --update py-pip
6
7  # upgrade pip
8  RUN pip install --upgrade pip
9
10 # install Python modules needed by the Python app
11 COPY requirements.txt /usr/src/app/
12 RUN pip install --no-cache-dir -r /usr/src/app/requirements.txt
13
14 # copy files required for the app to run
15 COPY app.py /usr/src/app/
16 COPY templates/index.html /usr/src/app/templates/
17
18 # tell the port number the container should expose
19 EXPOSE 5000
20
21 # run the application
22 CMD ["python", "/usr/src/app/app.py"]
```

- Instructions on how to build a Docker image
- Looks very similar to “native” commands
- Versioned and 100% reproducible

docker

a Dockerfile tells Docker how to build a docker image

the commands in the file are a mix of commands you’d actually run to install an application locally, and specific keywords that tell Docker what to do (RUN a command, COPY a file, etc)

an important point about Dockerfiles is that they live with your source code, which means that Docker images are 100% reproducible

this Dockerfile builds a simple python web app

Line 1: build this new Docker image based on the official Alpine Linux base image
Line 5: install python and pip (the python package manager)
Line 8: we use pip to ensure that pip is the latest version
Line 11: requirements.txt holds a list of libraries the app will need
Line 12: uses that file and pip to install the dependencies into the container
Line 15: copy my application code (app.py) into the /usr/src/app directory of the image
Line 16: copy our index.html file into the /usr/src/app/templates directory of the image
Line 19: the web server listens for connections on port 5000, so we tell Docker to listen on that port
Line 22: when the container starts up—we fire up python and pass it out application code to start the app

Dockerfile: Windows example

19 lines (15 sloc) | 832 Bytes — Raw | Blame | History

```
FROM microsoft/windowsservercore

ENV NPM_CONFIG_LOGLEVEL info
ENV NODE_VERSION 6.5.0
ENV NODE_SHA256 0c0562809b6c7104ce6643302b2592172183676e34397323ce3978b5ee34cf2

RUN powershell -Command \
    $ErrorActionPreference = 'Stop' ; \
    (New-Object System.Net.WebClient).DownloadFile('https://nodejs.org/dist/v%NODE_VERSION%/node-v%NODE_VERSION%-win-x64.zip',
    if ((Get-FileHash node.zip -Algorithm sha256).Hash -ne $env:NODE_SHA256) {exit 1} ; \
    Expand-Archive node.zip -DestinationPath C:\ ; \
    Rename-Item 'C:\node-v%NODE_VERSION%-win-x64' 'C:\nodejs' ; \
    New-Item '%APPDATA%\npm' ; \
    $env:PATH = 'C:\nodejs;%APPDATA%\npm;' + $env:PATH ; \
    [Environment]::SetEnvironmentVariable('PATH', $env:PATH, [EnvironmentVariableTarget]::Machine) ; \
    Remove-Item -Path node.zip

CMD [ "node.exe" ]
```


docker

and… i don’t know why you’d want your servers to blue screen, but you can do so if you please—there’s Docker for Windows as well

Cat gifs demo

What about data persistence?

- **Volumes** allow you to map a directory from your host into a container
  - e.g. map **/home/docker/mydbdata** on the host to the **/data** directory inside my MongoDB container
- They can also be used to share data between containers

docker

one of the tricky things about containers is that when a container is destroyed, any changes that were made to the container are removed in the process; they are not persistent

clearly this is suboptimal, you might want to save some logs from your application or, maybe your container was running a database and you want the data to last after the container is destroyed

the solution to this problem is a Volume; a volume is simply a subdirectory in your container that is mapped to a subdirectory on your host

e.g. mapping /home/docker/mydbdata from my host to /data inside my MongoDB container

and volumes can also be used to share data between containers

Now you try it!

now it’s your turn to learn, there are a few more things i’d like to say and then we can start the workshop

Try at home

- Visit https://docs.docker.com/installation
- Install the right version for your machine: Mac, Windows, Linux
- After Docker is installed, run catweb
  - docker run –d –p 5000:5000 --name catweb aluxian/catweb:1.0
- Browse to port 5000 on your machine
  - http://localhost:5000

Learn at home

- https://docs.docker.com
- https://github.com/docker/labs
- https://training.play-with-docker.com
- I'll post these slides on the event page

Thank You ;)
Q&A

Feedback

- I'd love to know if you enjoyed this
- https://goo.gl/P3BVGC

- Another presentation/workshop on 13 March
- Send us suggestions!

Next from Entrepreneurs Club

docker

Next from Entrepreneurs Club

docker

Play With Docker — Workshop

- Go to

  https://training.play-with-docker.com/beginner-linux/

- Do as much as possible
- If you finish it, you'll get a t-shirt

docker

The End (ok, now you can leave)