Lecture 13: Containers

Advanced Practical Data Science, MLOps

Pavlos Protopapas
Institute for Applied Computational Science, Harvard
Outline

1. Recap
2. Motivation / Tutorial
3. What is a Container
4. Tutorial: Building & Running Containers using Docker
5. Why use Containers?
Virtual Environments

**Pros**
- Reproducible research
- Explicit dependencies
- Improved engineering collaboration

**Cons**
- Difficulty setting up your environment
- Not isolation
- Does not always work across different OS
Virtual Machines

Pros
• Full autonomy
• Very secure
• Lower costs
• Used by all Cloud providers for on demand server instances

Cons
• Uses hardware in local machine
• Not very portable since size of VMs are large
• There is an overhead associated with virtual machines
Wish List

We want a system that:

● Automatically set up (installs) all OS and extra libraries and set up the python environment
● It is isolated
● Uses less resources
● Startups quickly
What is a CONTAINER

- Extremely **portable** and lightweight
- **Fully packaged** software with all dependencies included
- Can be used for development, training, and deployment
- Development teams can easily **share** containers

**Docker** is an open source platform for building, deploying, and managing containerized applications.
Environments vs Virtualization vs Containerization

- **Virtual Environments**
  - Python
  - Operating System
  - Infrastructure
  - Virtual Environments

- **Virtualization**
  - Hypervisor
  - Operating System
  - Infrastructure
  - Virtualization

- **Containerization**
  - Docker
  - Operating System
  - Infrastructure
  - Containerization
Tutorial

- Create a GCS Bucket and read/write files to it
- Let us run the simple-translate app using Docker
- For this we will do the following:
  - Create a VM Instance
  - SSH into the VM
  - Install Docker inside the VM
  - Run the Containerized simple-translate app
- Full instructions can be found [here](#)
What is a Container

- **Standardized** packaging for software dependencies
- **Isolate** apps from each other
- **Works** for all major Linux distributions, MacOS, Windows
What Makes Containers so Small?

Container = User Space of OS

- User space refers to all of the code in an operating system that lives outside of the kernel
How to run a docker container

• We use a simple text file, the **Dockerfile**, to build the **Docker Image**, which consists of an iso file and other files.

• We run the Docker Image to get **Docker Container**.
What is the difference between an image and container

Docker Image is a template aka a blueprint to create a running Docker container. Docker uses the information available in the Image to create (run) a container.

Image is like a recipe, container is like a dish.

Alternatively, you can think of an image as a class and a container is an instance of that class.
Inside the Dockerfile

FROM: This instruction in the Dockerfile tells the daemon, which base image to use while creating our new Docker image. In the example here, we are using a very minimal OS image called alpine (just 5 MB of size). You can also replace it with Ubuntu, Fedora, Debian or any other OS image.

RUN: This command instructs the Docker daemon to run the given commands as it is while creating the image. A Dockerfile can have multiple RUN commands, each of these RUN commands create a new layer in the image.

ENTRYPONT: The ENTRYPONT instruction is used when you would like your container to run the same executable every time. Usually, ENTRYPONT is used in scenarios where you want the container to behave exclusively as if it were the executable it's wrapping.

CMD: The CMD sets default commands and/or parameters when a docker container runs. CMD can be overwritten from the command line via the docker run command.
Multiple containers from same image

How can you run multiple containers from the same image?
Yes, you could think of an image as instating a class.

Wouldn’t they all be identical?
Not necessarily. You could instate it with different parameters using the CMD and therefore different containers will be different.

```
FROM ubuntu:latest
RUN apt-get update
ENTRYPOINT ["/bin/echo", "Hello"]
CMD ["world"]
```

> docker build -t hello_world_cmd:first -f Dockerfile_cmd .
> docker run -it hello_world_cmd:first
> Hello world
> docker run -it hello_world_cmd:first Pavlos
> Hello Pavlos
When we execute the build command, the daemon reads the Dockerfile and creates a layer for every command.
Image Layering

- **Container**: (Writable, running application)
- **Layered Image 2**
- **Layered Image 1**
- **Platform Image**: (Runtime Environment)

**A application sandbox**
- Each container is based on an image that holds necessary config data
- When you launch a container, a writable layer is added on top of the image

**A static snapshot of the container configuration**
- Layer images are read-only
- Each image depends on one or more parent images

**An Image that has no parent**
- Platform images define the runtime environment, packages and utilities necessary for containerized application to run
Docker layers for a container running Debian and a Python environment using Pipenv.
Some Docker Vocabulary

**Docker Image**
The basis of a Docker container. Represent a full application.

**Docker Container**
The standard unit in which the application service resides and executes.

**Docker Engine**
Creates, ships and runs Docker containers deployable on a physical or virtual, host locally, in a datacenter or cloud service provider.

**Registry Service (Docker Hub or Docker Trusted Registry)**
Cloud or server-based storage and distribution service for your images.
Installing Docker Desktop

1. Install **Docker Desktop**. Use one of the links below to download the proper Docker application depending on your operating system.
   - For Mac users, follow this link- [https://docs.docker.com/docker-for-mac/install/](https://docs.docker.com/docker-for-mac/install/).
   - For Windows users, follow this link- [https://docs.docker.com/docker-for-windows/install/](https://docs.docker.com/docker-for-windows/install/)
     Note: You will need to install Hyper-V to get Docker to work.
   - For Linux users, follow this link- [https://docs.docker.com/install/linux/docker-ce/ubuntu/](https://docs.docker.com/install/linux/docker-ce/ubuntu/)

2. Once installed run the docker desktop.

3. Open a Terminal window and type `docker run hello-world` to make sure Docker is installed properly.
Let us build the simple-translate app **Docker Container**

For this we will do the following:
- Clone or download [code](#)
- Build a container
- Run a container
- Pavlos will update a container on Docker Hub
- You will pull the new container and run it

For detail instruction go [here](#)
Tutorial: Docker commands

Check what version of Docker

```
docker --version
```

Get version of Docker CLI
List all running docker containers

- `docker container ls`
List all docker images

Docker command: `docker image ls`

Docker command for image

Docker command option to list all images
Tutorial: Docker commands

Build an image based on a Dockerfile

```
docker build -t ac215-d1 -f Dockerfile .
```

- **docker command**: `docker build -t ac215-d1 -f Dockerfile .`
- **Build the image**
- **Name the image**: `ac215-d1`
- **Name of docker file and “.” means look at the current working directory**
Run a docker container using an image from Docker Hub

```
docker run --rm --name ac215-d1 -ti --entrypoint /bin/bash ac215-d1
```

- **Run the container**
  - `docker run`: Command to run a container
  - `--rm`: Automatically clean up the container and remove the file system when the container exit
  - `--name ac215-d1`: Name of the container
  - `-ti`: 't' is to give us a terminal and 'i' is for interactive mode
  - `--entrypoint /bin/bash`: Default command to execute on startup
  - `ac215-d1`: Name of the image to use
Open another command prompt and check how many container and images we have

```
docker container ls
```

```
docker image ls
```
Exit from all containers and let us clear of all images

Docker command

Docker command for system

Docker command option to remove all images not referenced by any containers
Check how many containers and images we have currently

```
docker container ls
```

```
docker image ls
```
Docker Image as Layers

```bash
> docker build -t hello_world_cmd -f Dockerfile_cmd .
```

Sending build context to Docker daemon  34.3kB

Step 1/4 : FROM ubuntu:latest

```
latest: Pulling from library/ubuntu
54ee1f796a1e: Already exists
f7bfea53ad12: Already exists
46d371e02073: Already exists
b66c17bbf772: Already exists
```

Digest: sha256:31dfb10d52ce76c5ca0aa19d10b3e6424b830729e32a89a7c6eee2cda2be67a5
Status: Downloaded newer image for ubuntu:latest

```bash
---

Step 2/4 : RUN apt-get update
    ---> Running in e3e1a87e8d6e
    Get:1 http://archive.ubuntu.com/ubuntu focal InRelease [265 kB]
    Get:2 http://security.ubuntu.com/ubuntu focal-security InRelease [107 kB]
    Get:3 http://security.ubuntu.com/ubuntu focal-security/universe amd64 Packages [67.5 kB]
    Get:4 http://archive.ubuntu.com/ubuntu focal-updates InRelease [111 kB]
    Get:5 http://archive.ubuntu.com/ubuntu focal-backports InRelease [98.3 kB]
    Get:6 http://security.ubuntu.com/ubuntu focal-security/main amd64 Packages [231 kB]
    Get:7 http://archive.ubuntu.com/ubuntu focal/restricted amd64 Packages [33.4 kB]
    Get:8 http://archive.ubuntu.com/ubuntu focal/main amd64 Packages [1275 kB]
    Get:9 http://security.ubuntu.com/ubuntu focal-security/multiverse amd64 Packages [1078 B]
```

...
>docker build -t hello_world_cmd -f Dockerfile_cmd .

....

Step 3/4 : ENTRYPOINT ["/bin/echo", "Hello"]
--->
Running in 52c7a98397ad
Removing intermediate container 52c7a98397ad
--->
7e4f8b0774de

Step 4/4 : CMD ["world"]
--->
Running in 353adb968c2b
Removing intermediate container 353adb968c2b
--->
a89172ee2876
Successfully built a89172ee2876
Successfully tagged hello_world_cmd:latest
### Docker Image as Layers

```bash
> docker images

<table>
<thead>
<tr>
<th>REPOSITORY</th>
<th>TAG</th>
<th>IMAGE ID</th>
<th>CREATED</th>
<th>SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>hello_world_cmd</td>
<td>latest</td>
<td>a89172ee2876</td>
<td>7 minutes ago</td>
<td>96.7MB</td>
</tr>
<tr>
<td>ubuntu</td>
<td>latest</td>
<td>4e2eef94cd6b</td>
<td>3 weeks ago</td>
<td>73.9MB</td>
</tr>
</tbody>
</table>
```

```bash
> docker image history hello_world_cmd

<table>
<thead>
<tr>
<th>IMAGE</th>
<th>CREATED</th>
<th>CREATED BY</th>
<th>SIZE</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>a89172ee2876</td>
<td>8 minutes ago</td>
<td>/bin/sh -c #(nop) CMD [&quot;world&quot;]</td>
<td>0B</td>
<td></td>
</tr>
<tr>
<td>7e4f8b0774de</td>
<td>8 minutes ago</td>
<td>/bin/sh -c #(nop) ENTRYPOINT [&quot;/bin/echo&quot; &quot;…&quot;]</td>
<td>0B</td>
<td></td>
</tr>
<tr>
<td>cfc0c414a914</td>
<td>8 minutes ago</td>
<td>/bin/sh -c apt-get update</td>
<td>22.8MB</td>
<td></td>
</tr>
<tr>
<td>4e2eef94cd6b</td>
<td>3 weeks ago</td>
<td>/bin/sh -c #(nop) CMD [&quot;/bin/bash&quot;]</td>
<td>1.01MB</td>
<td></td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>3 weeks ago</td>
<td>/bin/sh -c mkdir -p /run/systemd &amp;&amp; echo 'do…'</td>
<td>7B</td>
<td></td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>3 weeks ago</td>
<td>/bin/sh -c set -xe &amp;&amp; echo '#!/bin/sh' &gt; /…</td>
<td>811B</td>
<td></td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>3 weeks ago</td>
<td>/bin/sh -c [-z &quot;$(apt-get index-targets)&quot; ]</td>
<td>1.01MB</td>
<td></td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>3 weeks ago</td>
<td>/bin/sh -c #(nop) ADD file:9f937f4889e7bf646…</td>
<td>72.9MB</td>
<td></td>
</tr>
</tbody>
</table>
```
Why Layers

Why build an image with multiple layers when we can just build it in a single layer? Let’s take an example to explain this concept better, let us try to change the Dockerfile_cmd we created and rebuild a new Docker image.

```
> docker build -t hello_world_cmd -f Dockerfile_cmd .
Sending build context to Docker daemon  34.3kB
Step 1/4 : FROM ubuntu:latest
  ---> 4e2ee94c9d6b
Step 2/4 : RUN apt-get update
  ---> Using cache
  ---> cfc0414a914
Step 3/4 : ENTRYPOINT ["/bin/echo", "Hello"]
  ---> Using cache
  ---> 7e4f8b0774de
Step 4/4 : CMD ["world"]
  ---> Using cache
  ---> a89172ee2876
Successfully built a89172ee2876
Successfully tagged hello_world_cmd:latest
```

As you can see that the image was built using the existing layers from our previous docker image builds. If some of these layers are being used in other containers, they can just use the existing layer instead of recreating it from scratch.
Why use Containers?

• Imagine you are building a large complex application (e.g. Online Store)

• Traditionality you would build this using a Monolithic Architecture
Monolithic Architecture

Browser Apps

HTML / REST / JSON

Server

Storefront UI Module
Catalog Module
Reviews Module
Orders Module

REST / JSON

Mobile Apps

Database
Monolithic Architecture

Browser Apps

Mobile Apps

Server

Storefront UI Module
Catalog Module
Reviews Module
Orders Module

Database

Oracle

Java

HTML / REST / JSON

REST / JSON
Monolithic Architecture - Advantages

Simple to **Develop, Test, Deploy** and **Scale**:

1. Simple to develop because all the tools and IDEs support the applications by default
2. Easy to deploy because all components are packed into one bundle
3. Easy to scale the whole application
Monolithic Architecture - Disadvantages

1. Very difficult to maintain
2. One component failure will cause the whole system to fail
3. Very difficult to create the patches for monolithic architecture
4. Adapting to new technologies is challenging
5. Take a long time to startup because all the components need to get started
Applications have changed dramatically

A decade ago
Apps were monolithic
Built on a single stack (e.g. .NET or Java)
Long lived
Deployed to a single server

Today
Apps are constantly being developed
Build from loosely coupled components
Newer versions are deployed often
Deployed to a multitude of servers
Applications have changed dramatically

A decade ago

Apps were monolithic
Built on a single stack (e.g. .NET or Java)
Long lived
Deployed to a single server

Today

Apps are constantly being developed
Build from loosely coupled components
Newer version are deployed often
Deployed to a multitude of servers

Data Science

Apps are being integrated with various data types/sources and models
Today: Microservice Architecture

- **Browser Apps**
  - REST / JSON
  - HTML

- **Mobile Apps**
  - REST / JSON

- **Edge Device Apps**
  - REST / JSON

**API Service**

- **Browser Apps**
- **Mobile Apps**
- **Edge Device Apps**

**Storefront UI**

- **Catalog Module**
- **Reviews Module**
- **Orders Module**
- **Recommendation Module**

**Database**
- **Cloud Store**
- **Models**
Software Development Workflow (no Docker)

OS Specific installation in every developer machine
Software Development Workflow (no Docker)

Every team member moves code to source control

OS Specific installation in every developer machine
Every team member moves code to source control.

Build server needs to be **installed** with all required softwares/frameworks.

Production build is performed by pulling code from source control.

OS Specific **installation** in every developer machine.
Software Development Workflow (no Docker)

Every team member moves code to source control

Build server needs to be **installed** with all required softwares/frameworks

Production build is performed by pulling code from source control

Production server needs to be **installed** with all required softwares/frameworks

Production server will be different OS version than development machines

OS Specific installation in every developer machine
Software Development Workflow (with Docker)

Development machines only need Docker installed

Containers need to be setup only once
Software Development Workflow (with Docker)

Development machines only needs **Docker installed**

**Containers** need to be setup only once

Every team member moves code to source control

Source Control

GitHub
Software Development Workflow (with Docker)

Every team member moves code to source control.

Build server only needs Docker installed.

Docker images are built for a release and pushed to container registry.

Containers need to be setup only once.
Software Development Workflow (with Docker)

Every team member moves code to source control.

Build server only needs Docker installed.

Docker images are built for a release and pushed to container registry.

Production server only needs Docker installed.

Production server pulls Docker images from container registry and runs them.

Containers need to be setup only once.

Development machines only needs Docker installed.

Source Control

GitHub

Build Server

Production/ Test Servers

Production server

Linux

Linux

Production server

Linux

Production server

Linux

Build server

Docker installed

Docker images

container registry

Docker images

container registry

Docker images

container registry

Docker images

container registry

Docker images

container registry

Docker images

container registry

Docker images

container registry
Comparison

<table>
<thead>
<tr>
<th></th>
<th>Virtual ENV</th>
<th>Docker</th>
<th>VM</th>
<th>JH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computational Cost</td>
<td>LOW</td>
<td>MEDIUM</td>
<td>HIGH</td>
<td>?</td>
</tr>
<tr>
<td>Memory Footprint</td>
<td>LOW</td>
<td>LOW</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deployment</td>
<td>EASY</td>
<td>MEDIUM</td>
<td>INIT HIGH THEN EASY</td>
<td>N/A</td>
</tr>
<tr>
<td>Versatility (Types of Apps)</td>
<td>MEDIUM HIGH</td>
<td>MEDIUM HIGH</td>
<td>MEDIUM HIGH</td>
<td>LOW</td>
</tr>
<tr>
<td>Portability</td>
<td>MEDIUM</td>
<td>HIGH</td>
<td>HIGH</td>
<td>HIGH</td>
</tr>
</tbody>
</table>

- **Computational Science**
- **DevOps**
- **Data Science (No Pipeline)**
- **Data Science (Pipelines)**
THANK YOU