CS 240 Computer Systems

Week 11: Docker and IaaS / PaaS / SaaS

CS 240, Spring 2021 - Week 10

Wade Fagen-Ulmschneider

As we build increasingly complex and distributed software, we need to think about what level of a system is most useful:

IaaS Infrastructure as a Service	**CaaS** Containers as a Service	**PaaS** Platform as a Service	**FaaS** Functions as a Service	**SaaS** Software as a Service
Data	Data	Data	Data	Data
Functions	Functions	Functions	Functions	Functions
Applications	Applications	Applications	Applications	Applications
Runtime	Runtime	Runtime	Runtime	Runtime
Containers*	Containers	Containers*	Containers*	Containers*
OS	OS	OS	OS	OS
Virtualization	Virtualization	Virtualization	Virtualization	Virtualization
Hardware	Hardware	Hardware	Hardware	Hardware

	Abstracted by Provider/Vendor		Customer Managed Unit of Scale		Customer Managed

---

Infrastructure as a Service (IaaS)

When you choose to host your technology on IaaS, you are provided virtualization running on top of managed hardware.

- **Example:** Amazon EC2 / Google Compute Engine

- Why IaaS?

Containers as a Service (CaaS)

When you choose to host your technology on CaaS, you are provided a container server running on top of managed virtualization.

- **Example:** Amazon ECS / Google App Engine

- Why CaaS?

Platform as a Service (PaaS)

When you choose to use PaaS, you are provided a fully functional platform ready to be used as infrastructure.

- **Database Platforms:** Amazon RDS / Google Cloud SQL / Amazon DocumentDB / Google DataStore

- **Software Runtime Platforms**: Amazon Elastic Beanstalk / AutoML / etc

- Why PaaS?

Functions as a Service (FaaS)

When you choose to use FaaS, you provide a source code function and its executed within a runtime environment on-demand.

- **Database Platforms:** Amazon Lambda / Google Cloud Functions

- Why FaaS?

Software as a Service (SaaS)
When you choose to use SaaS, you are provided a complete managed software solution where you will never see source code.

Running IaaS / Using a Virtual Machine:
Except for running the hardware yourself ("self-provisioned"), this is the most basic form of software infrastructure:

- You choose hardware characteristics of your host machine (# of CPUs, RAM, networking, storage, etc) and the operating system (ex: Ubuntu, Windows, etc).

- You are given "root" or "sudo" privileges to the OS and need to install the software stack that you want to run -- but you also manage the security and reliability of the OS!

Example: Google Compute Engine Pricing (April 2020)

For a VM running 24/7 that will never shutdown/get killed:

- **$0.021811** / vCPU hour == **$191.06** /vCPU year
- **$0.002923** / GiB hour == **$25.61** /GiB year
- **+$0.085** / GiB bandwidth == **$87.04** /TiB bandwidth

E2 General Purpose Machines, Sourced: https://cloud.google.com/compute/all-pricing

---

Running CaaS / Using a Container:
Suppose you are uninterested in running your own OS -- what if you just want to specify the runtime environment (binaries / libraries) you want your code to run in?

Solution: Containers!

- Many different container solutions, including **lxc** (Linux containers), **rkt**, and Docker.
- We will focus on Docker Containers in CS 240, but the principles of development are similar across all containers.

Docker Container: Basic Design
docker is a containerization engine *(and associated management tools)* that allows for container images to run a managed runtime within your operating system via a docker image file.

Example: Recall your MP6 web service that wrapped your MP2 solution into a web service:

Dockerfile

```
 1 FROM python:3.9
 2
 3 # Setup python libraries:
 4 COPY requirements.txt /
 5 RUN pip install -r /requirements.txt
 6
 7 # Copy needed files to run:
 8 COPY png-analyze /
 9 COPY app.py /
10 COPY templates /templates
11
12 # Run server:
13 CMD ["python", "-m", "flask", "run",
   "--host=0.0.0.0"]
```

Line 1 (FROM):

Lines 4, 8-10 (COPY):

Line 5 (RUN):

LINE 13 (CMD):

Effectively, a dockerfile simply specifies the base image, all files needed, any setup commands (that are run during the build phase), and then the command to run during the run phase. To build it:

```
$ docker build --tag cs240-mp6 .
```

Running a Docker Image

```
$ docker run --rm -it -p 5000:5000 cs240-mp6
```

- docker run
- --rm
- -it
- -p
- cs240-mp6

What do we expect to happen?

Mounting a Directory within Docker:

```
$ docker run --rm -it -v /home/waf/mp6-temp:/temp -p
5000:5000 cs240-mp6
```

- -v

When is using the -v option critical?

Docker Images as Building Blocks

Every dockerfile starts with a `FROM <image>` -- all the way down to `FROM scratch` (an image that contains no starting environment).

cs240-mp6 image:

```
FROM python:3.9
...
```

python:3.9 image:

```
FROM buildpack-deps:buster
...
```

buildpack-deps:buster image:

```
FROM buildpack-deps:buster-scm
...
```

buildpack-deps:buster-scm image:

```
FROM buildpack-deps:buster-curl
...
```

buildpack-deps:buster-curl image:

```
FROM debian:buster
...
```

debian:buster image:

```
FROM scratch
ADD rootfs.tar.xz /
CMD ["bash"]
```

Building Services for Consumption
Without knowing it, you have been writing Application Programming Interfaces -- commonly known as APIs -- to build services for others to consume your data:

- In MP6:
  - **/extract**, allows the extraction of a hidden "uiuc" GIF from a PNG image.
- In MP7:
  - **/:subject/:course**, returns the GPA and course credit information from the courses-microservice
  - **/scheduleGPA**, returns the GPA of a provided schedule

There are a lot of details in how to write a good API for others to use!

RESTful APIs:
Four Key Architectural Features:

- [Stateless]:

- [Client-Server]:

- [Explicit Caching]:

- [Layered System]:

Non-RESTful APIs:
Many other APIs exist outside of the RESTful API space -- particularly any if the request requires **state**.

Example Service: `MapReduce`

Example Input:

The	*quick*	*brown*	*fox*	*jumps*	*over*	*the*	*lazy*	*dog*
[0]	**[1]**	**[2]**	**[3]**	**[4]**	**[5]**	**[6]**	**[7]**	**[8]**

Map Function:

Reduce Function:

Example Input:

[0]	[1]	[2]	[3]
admin2: Champaign region: Illinois country: US cases: 6390 date: 2020-10-29	admin2: Champaign region: Illinois country: US cases: 6285 date: 2020-10-28	admin2: Champaign region: Sicilia country: Italy cases: 18325 date: 2020-10-28	admin2: Champaign region: Lombardia country: Italy cases: 162968 date: 2020-10-28

Source:

https://github.com/CSSEGISandData/COVID-19/tree/master/csse_covid_19_data/csse_covid_19_daily_reports