BOOTSTRAP the world for your TESTS

Congreso esLibre
Vigo, 24 y 25 de Junio
Edición presencial y virtual
2022

esLibre 2022, Vigo, Galicia, España

Before swim in the deep sea..

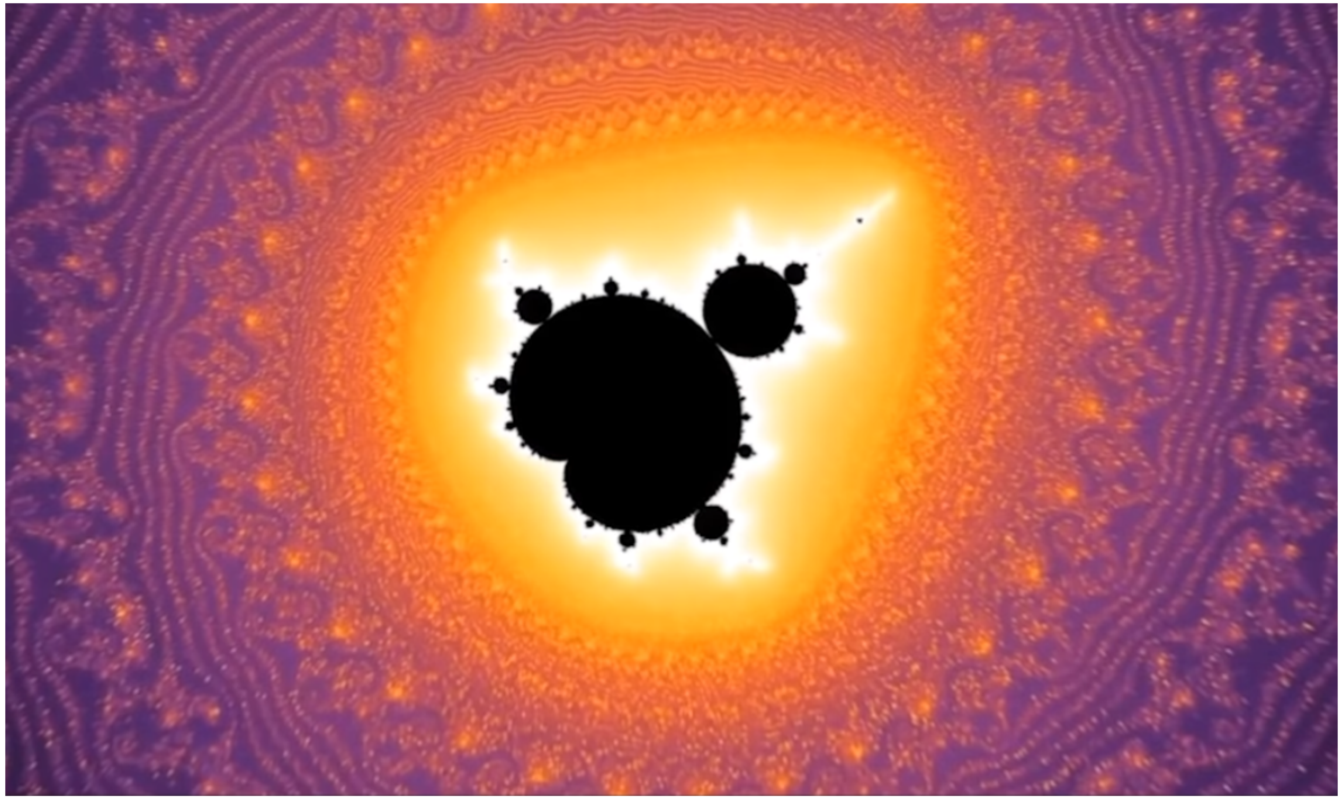
0. Not a Holy Grail

1. Eventually general purpose solution BUT stick to the particular technolgy and [maybe (application architecture)]

2. Keep your mind critical

Pay attention and **Enjoy it!**

Mativation



The Art of Code, Dylan Beattie, GOTO 2020. Retrieved 2022, from https://www.youtube.com/watch?v=yDB3wbkfEeI

“ System under test (SUT) refers to a system that is being tested for correct operation. ”

Source Wikipedia. Retrieved 2022, from https://en.wikipedia.org/wiki/System_under_test

Challenges

Challenges

1. Consistent environment/SUT state->"golden state"
2. Deterministic finite-state
3. Programmatically way to run-up
4. Resonable resource usage and time execution

more challenges...

1. Declarative way of SUT assembling
2. Dependency and dependency ordering resolve
3. Order of execution
4. ...

Let’s stop for a while!

The same old story..

"dependency hell", execution modes {in memory, dummy,...}, configurations...

run-up and keeping correct state <u>in a not trivial environment</u> such a microservices...

=>

all boil down to the difficulties of managing state

Proposal

Encapsulate as much as possible the state of the environment!

Solution

Dockerize it

Solution explanation

- An application ~ a single docker image
- Single point of true
- Reference point
- Speaking in terms of "Unit of change"
- Clear borderlines and responsiblity
- Perfect suite for microservice base architecture

Let’s stop again!

Little by Little putting details!

Execution Unit explentation

Speaking about computer systems, we all, have the same sort of issues/challenges to sort out.

1. Unit without any dependencies - only raw data input

```
unit :: s -> f
unit = f(s) -- simple to execute
```

2. Dependencies in between units - state dependencies

```
-- for such kind a complex and non trivial environment
-- we need a simple powerfull DSL.
```

Study, study and one more time study

systemd

- Offers simple declarative schema for definition, execution order and process magnage.

```
# kafka.target file
[Unit]
Description=Kafka
```

```
# app.target
[Unit]
Description=App
Wants=kafka.target
[Service]
ExecStart=java -jar ...
```

```
systemctl start app.target
```

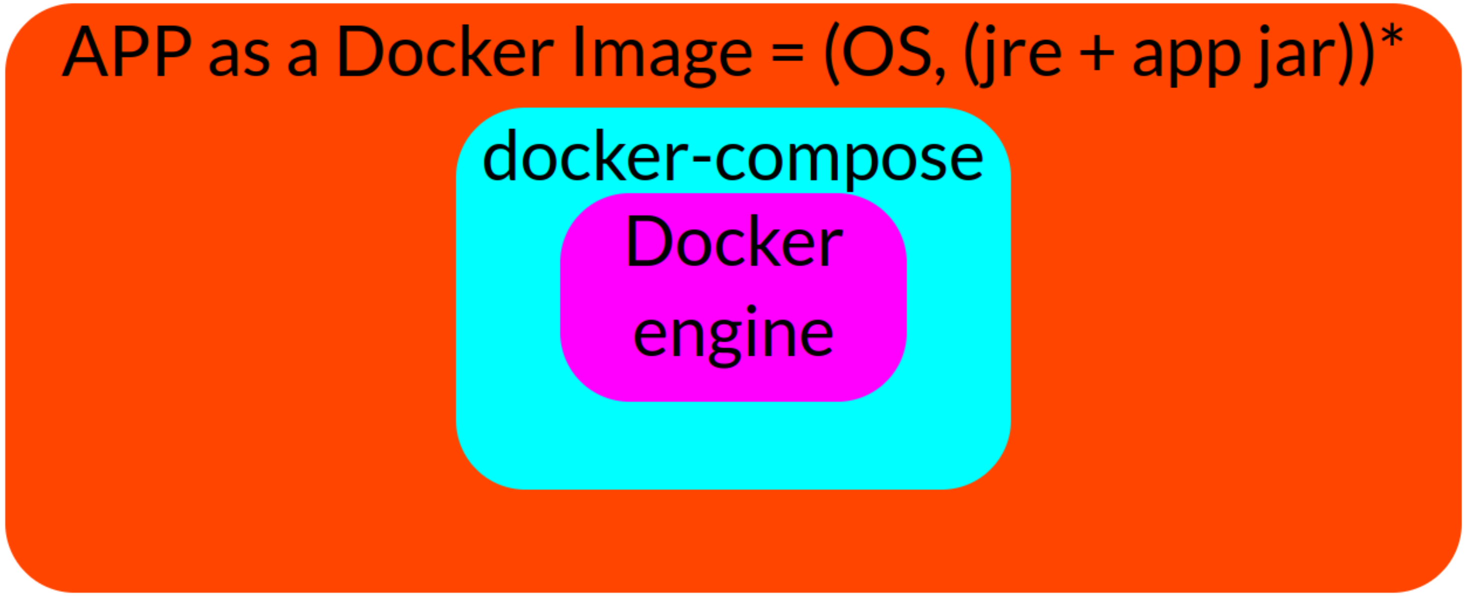
Dockerize it

Docker engine

docker-compose

Classic usage I

Dockerize it



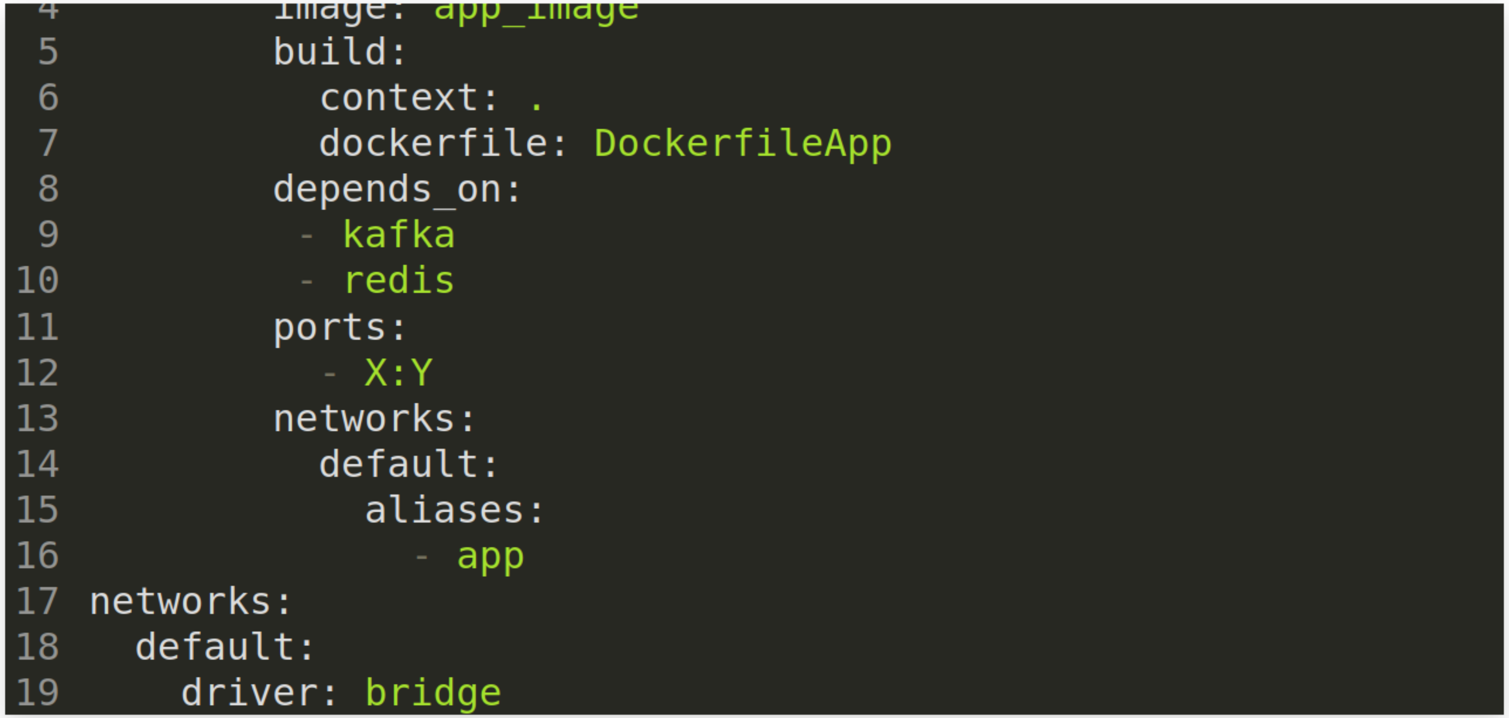
Classic usage II

Conclusions I

1. Docker engine and Dockerfile as a great technology candidate

```dockerfile
FROM _
RUN mkdir -p /usr/ourApp
COPY _ # binary, run script, *conf
ENTRYPOINT["sh", "_.sh"]
```

2. docker-compose CLI has a great DSL (**depends_on**, **networks**)



```yaml
        image: app_image
        build:
          context: .
          dockerfile: DockerfileApp
        depends_on:
         - kafka
         - redis
        ports:
          - X:Y
        networks:
          default:
            aliases:
              - app
networks:
  default:
    driver: bridge
```

Docker compose manage links easier. Treat container as a single entity.