Parte 1 – Oi, eu sou o Docker

Porquê?

Isolamento/”enjaulamento” de aplicações com as configurações de sistema necessárias para funcionar.
Imutabilidade do estado inicial.
Independência da aplicação em relação ao Host. (*)

Como?

Compartilhando os recursos no nível do kernel.
Usando o UnionFS.

Não confundir:

- **Imagem:** a configuração preparada. (o executável compilado)
- **Conteiner:** a execução da configuração preparada. (o programa rodando)

A funcionalidade de conteinerização existe no Linux há muito tempo (2.6.24 – Janeiro/2008 para os cgroups; o LXC é de Agosto/2008). O Docker é uma ferramenta que automatiza e facilita o uso. Existem outras.

Não é VM. Parece, olhando de fora. Mas não é.

Processo de uso:

Simples:

1. Obtém a imagem (`base image`)
2. Cria o conteiner (com o `docker run`) e usa.

Exemplo: rodar um mysql

```
$ docker run -d –name=mysqlapp \
 -e MYSQL_ROOT_PASSWORD=senhadificil -e MYSQL_DATABASE=meudb \
 -e MYSQL_USER=meuuser -e MYSQL_PASSWORD=minhasenha \
 --name=mysqlapp mysql:latest
```

Completo:

1. Obtém a imagem inicial (`base image`)
2. Monta uma configuração usando o Dockerfile, incrementando a imagem inicial.
3. Cria a imagem personalizada (`docker build`)
4. Cria o conteiner baseado na imagem criada

Alguns comandos

`pull`	Obtém uma imagem
`image`	Gerencia as imagens disponíveis
`run`	Cria um container baseado em uma imagem
`build`	Constrói uma imagem baseada em uma configuração (Dockerfile)
`ps`	Lista os containers
`exec`	Roda um comando em um conteiner ativo
`inspect`	Obtém dados sobre um conteiner
`kill`	Encerra a execução de um conteiner

Cursos/treinamentos de Docker

http://birthday.play-with-docker.com/

http://dockr.ly/2gpE6ax

Parte 2 – docker volume e docker network

Docker volume

Porquê?

Isolar e reutilizar os dados em diferentes conteiners.
Um volume pode ser usado em vários conteiners ao mesmo tempo!

Usar diferentes drivers, que permitem diversos tipos de armazenamento de modo transparente dentro do conteiner. (NFS, GlusterFS, etc)

Como funciona?

Cria “volumes” nomeados para armazenar os arquivos.

Surgiu como uma alternativa a alguns padrões anteriores:

- Conteiner de dados (reaproveitados com o parametro `--volumes-from` do `docker run`)
- Montagem de diretório do hospedeiro.

Onde ficam os volumes?

No Linux: /var/lib/docker/volumes/

Comandos:

create: Cria um volume

```
$ docker volume create nome_volume
```

ls: Listar os volumes existentes

```
$ docker volume ls
```

inspect: Verificar os dados de um volume

```
$ docker volume inspect nome_volume
```

rm: Exclui um conteiner

```
$ docker volume rm nome_volume
```

Usando um volume criado

```
$ docker run (...) -v «nome_do_volume»:«ponto_de_montagem» (...)
```

Docker networks

Porquê?

Isolar conteiners em diferentes redes.
Se você precisa rodar várias aplicações, compostas de vários conteiners, em um único host, todos os conteiners ficam na mesma rede. Por isso, pode criar diferentes redes para isolar os conteiners.

Como funciona?

Redes preexistentes:

- **bridge:** rede padrão
- **none:** nenhuma rede disponível para o conteiner
- **host:** as interfaces de rede do host ficam disponíveis para o conteiner

Novas redes podem usar drivers que permitam comportamentos de rede mais complexos, como conexão entre conteiners existentes em diferentes hospedeiros.

Cuidado para não confundir a funcionalidade de links entre conteiners (parâmetro `--link` do `docker run`) com a funcionalidade de redes. O link é uma facilidade para mapeamento de nomes entre os conteiners.

Comandos:

create: Cria uma rede

```
$ docker network create rede_001
```

ls: Listar as redes existentes

```
$ docker network ls
```

inspect: Verificar os dados de um volume

```
$ docker network inspect rede_001
```

rm: Exclui um conteiner

```
$ docker network rm rede_001
```

connect/disconnect: conecta um conteiner a uma rede

```
$ docker network connect rede_001 meu_conteiner
```

Usando uma rede:

```
$ docker run (...) --net rede_001 (...)
```

Parte 3 – Docker Compose

Porquê?

Porque conteiners, normalmente, não funcionam de forma separada. Uma aplicação completa depende de vários componentes, que podem ser isolados em conteiners. E o Docker Compose define um formato de arquivo e um conjunto de comandos que possibilitam isso.

Instalação:

```
$ sudo pip install docker-compose
```

Como funciona:

Define um arquivo `docker-compose.yml` com a descrição dos serviços e suas configurações. Este arquivo pode apontar tanto para imagens já existentes quanto para contextos de construção (diretório com Dockerfile + configurações).

Com o arquivo existente e válido, o comando `docker-compose up` inicia todos os serviços (conteineres) descritos.

Exemplo:

```
version: "2"
services:
  db:
    image: mysql:5.6
    volumes:
      - ./data:/var/lib/mysql
    environment:
      MYSQL_ROOT_PASSWORD: senha1234
      MYSQL_DATABASE: dbcrud
      MYSQL_USER: crud
      MYSQL_PASSWORD: senhacrud
  web:
    build: ./webserver-container/
    volumes:
      - .:/var/www/html
    ports:
      - 8088:80
    depends_on:
      - db
```

Parte 4 – Conteinerizando aplicações

Alguns pré-requisitos:

- conhecer **bem** as configurações do stack usado. O que compõe, como os serviços conversam entre si, quais os desafios que apresenta.
- saber configurar **todos** os serviços envolvidos.
- saber diferenciar volumes montados em conteiners de arquivos copiados para os conteiners.
- conhecer o suficiente sobre Dockerfile para construir as máquinas customizadas.

Dockerfile:

Define uma imagem personalizada, descrevendo os passos para a criação.
Diretivas importantes:

FROM	Define a "base image" que será usada
COPY	Copia arquivos do contexto para a imagem
RUN	Executa um comando durante a criação do conteiner
ENV	Exporta uma variável de ambiente dentro do conteiner
CMD	Define um comando que pode ser executado quando o conteiner for criado, caso não for definido um comando na linha `docker run`
ENTRYPOINT	Define o comando que será executado quando o conteiner for criado. Se outro comando for informado no momento do `docker run`, esse comando é passado como parâmetro para o comando definido em **ENTRYPOINT**
VOLUME	Informa um ponto de montagem no conteiner que será usado como volume
WORKDIR	Define um diretório onde rodar comandos durante a criação da máquina
USER	Define o usuário padrão do conteiner quando rodar um comando **CMD**, **RUN** ou **ENTRYPOINT**
EXPOSE	Define as portas que o conteiner irá expor

Quando estiver construindo uma imagem personalizada, preste atenção às configurações pedidas pela base image. Estas configurações tem que ser informadas no Dockerfile, ou no momento de criar o conteiner usando o `docker run`.

Comando "docker run":

Define os parâmetros de execução do conteiner. Dependendo da imagem usada, alguns parâmetros podem ser obrigatórios.

docker-compose:

Coordena a execução completa do stack de desenvolvimento. É aqui que você define como cada componente da aplicação funciona.