Cloud Computing

Gledson Scotti

Docker

SATC
EDUCAÇÃO E TECNOLOGIA

Engenharia da Computação

Engenharia da Computação

Cenário

Imagine que você desenvolveu um software, agora será necessário configurar e organizar a infraestrutura para fazer o deploy de sua aplicação e colocá-la em produção.

Você segue vários passos para esta atividade, desde instalação e configuração dos servers, até ajustes na aplicação para que rode em um webserver, por exemplo.

Muito tempo gasto nas etapas de ajustes e configuração, mesmo utilizando ferramentas de automação como Rundeck, Puppet, Chef, Jenkins ou Ansible, ainda teremos problemas com infraestrutura.

Imagine então sua aplicação separada em vária camadas, front-end, back-end, banco de dados e outros. Para fazer esta solução funcionar em vários ambientes você acaba instalando-a de várias formas, perdendo assim por muitas vezes controle do formato e configuração das instalações.

Engenharia da Computação

Conceitos

Visto o cenário anterior, entenda que o Docker pode isolar e manter a mesma configuração de cada serviço ou camada indiferentemente do ambiente em que esteja.

Docker é uma ferramenta para criar e manter containers, ou seja, ele é responsável por armazenar vários serviços de forma isolada do SO host, como: web server, banco de dados, aplicação etc. O seu back-end é baseado em LXC (LinuX Containers) que isola processos do sistema operacional host.

Como “**um navio cargueiro em que cada container leva várias mercadorias**”.

Não faz uso de emulação ou suporte a hardware, apenas proporciona a execução de vários sistemas Linux de forma isolada. Utiliza o mesmo Linux Kernel do servidor host, o que torna tudo muito rápido.

Engenharia da Computação

Conceitos

Nesse ecossistema temos os seguintes softwares:

- **Docker Engine**: É o software base de toda solução. É tanto o daemon responsável pelos containers como o cliente usado para enviar comandos para o daemon;

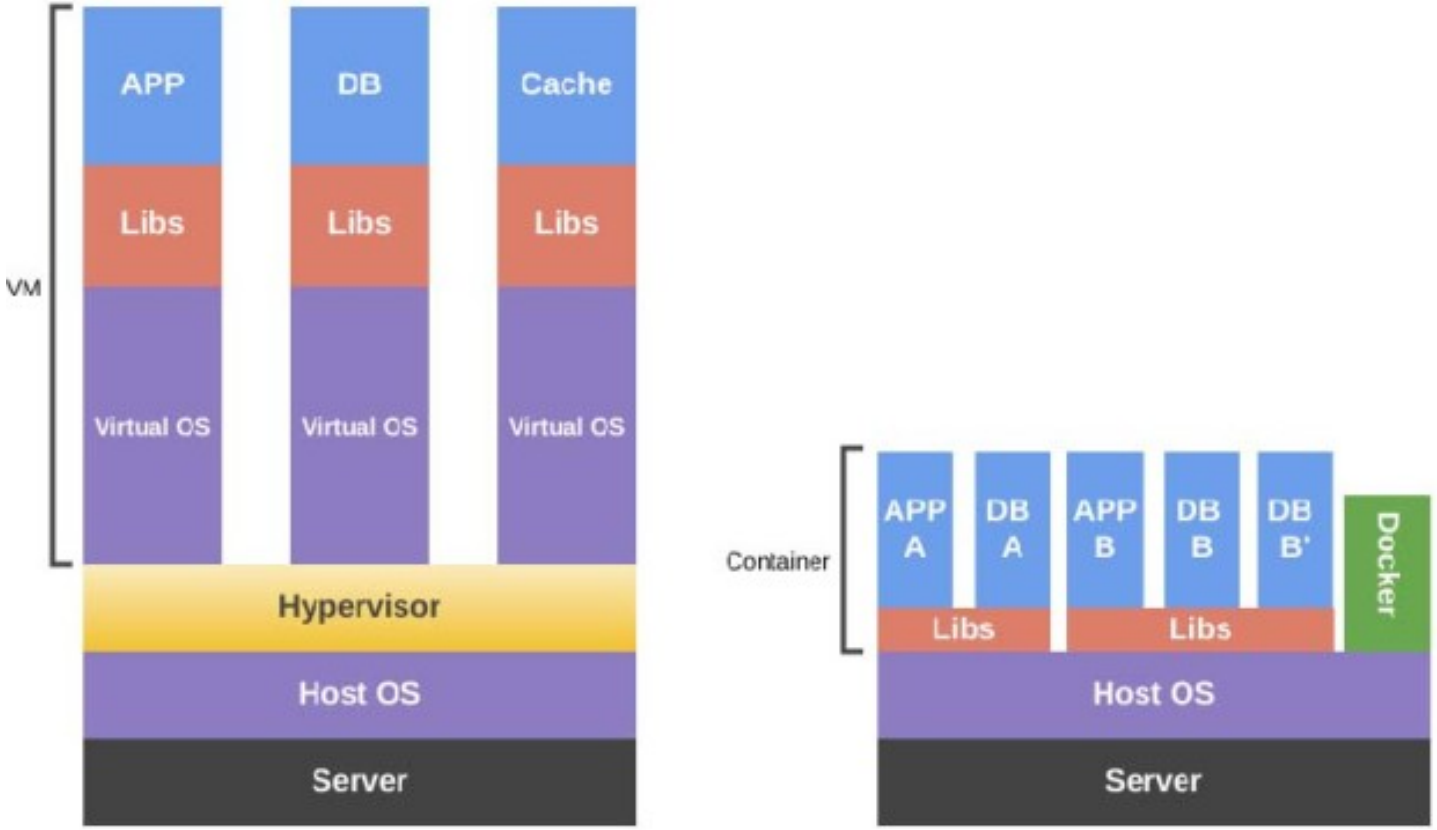
- **Docker Compose**: É a ferramenta responsável pela definição e execução de múltiplos containers com base em arquivo de definição;

- **Docker Machine**: é a ferramenta que possibilita criar e manter ambientes docker em máquinas virtuais, ambientes de nuvem e até mesmo em máquina física.

Container versus máquina virtual

- A **virtualização** proporciona um isolamento entre maquinas virtuais, mas ao mesmo tempo, uma sobrecarga, pois cada máquina virtual que é criada executará seu próprio kernel e instância do sistema operacional.

- A **virtualização por containers**, ocorre de forma menos isolada, pois compartilha algumas partes do kernel do host, fazendo com que a sobrecarga seja menor.



Engenharia da Computação

Comandos básicos

Antes de executar um container é necessário verificar a partir de qual imagem ele será executado. Para listar as imagens disponíveis em seu Docker host possui localmente digite:

$ docker image list

As imagens que retornam são as que estão presentes em seu host docker. Caso seja necessário utilizar uma imagem que seu docker host não possui, esta deve ser baixada da nuvem publica do Docker.

$ docker image pull **python**

Como exemplo utilizamos a imagem chamada **python**, mas caso deseje atualizar qualquer outra imagem, basta trocar o nome **python** pela imagem de seu interesse.

Engenharia da Computação

Comandos básicos

A imagem que acabou de atualizar ou baixar pode ser verificada em detalhes, para isso, basta usar o comando abaixo:

$ docker image inspect python

O comando inspect é responsável por informar todos os dados referentes à imagem. Após inspecionar a imagem, podemos executar um container utilizando com o base a sintaxe abaixo:

$ docker container run <parâmetros> <imagem> <CMD> <argumentos>

Os parâmetros mais utilizados na execução do container são:

-d	Execução do container em background.
-i	Modo interativo. Mantém o STDIN aberto mesmo sem console anexado.
-t	Aloca uma pseudo TTY.
-rm	Automaticamente remove o container após finalização (**Não funciona com -d**)
-name	Nomear o container

Comandos básicos

-v	Mapeamento de volume
-p	Mapeamento de porta
-m	Limitar o uso de memória RAM
-c	Balancear o uso de CPU

Exemplo simples de execução de container:

$ docker container run -it --rm --name meu_python python bash

O comando acima, será iniciado um container com o nome meu_python, criado a partir da imagem python e o processo executado nesse container será o bash.

Já o mapeamento de volumes se dá em especificar qual origem do dado no host e onde deve ser montado dentro do container.

$docker container run -it --rm -v “<host>:<container>” python

Engenharia da Computação

Comandos básicos

O mapeamento de portas basta saber qual porta será mapeada no host e qual deve receber essa conexão dentro do container.

$ docker container run -it --rm -p "<host>:<container>" python

Exemplo com a porta 80 do host para uma porta 8080 dentro do container tem o seguinte comando:

$ docker container run -it --rm -p 80:8080 python

Para um container utilizar somente 512MB de RAM.

$ docker container run -it --rm -m 512M python

Para balancear o uso da CPU pelos containers, utilizamos especificação de pesos, quanto menor o peso, menor sua prioridade no uso. Os pesos podem oscilar de 1 a 1024. Não sendo especificado, ele usará o maior peso possível, **1024**. usaremos como exemplo o peso 512.

$ docker container run -it --rm -c 512 python

Engenharia da Computação

Comandos básicos

Vamos imaginar que três containers foram colocados em execução. Um deles tem o peso padrão 1024 e dois têm o peso 512. Caso os três processos demandem toda CPU o tempo de uso deles será dividido da seguinte maneira:

- O processo com peso 1024 usará 50% do tempo de processamento;
- Os dois processos com peso 512 usarão 25% do tempo de processamento, cada.

Para visualizar a lista de containers de um determinado Docker host utilizamos o comando **docker ps**. Responsável por mostrar todos os containers, mesmo aqueles não mais em execução.

$ docker container list <parâmetros>

-a	Lista todos os containers, inclusive os desligados.
-l	Lista os últimos containers, inclusive os desligados.
-n	Lista os últimos N containers, inclusive os desligados.
-q	Lista apenas os ids dos containers, ótimo para utilização em scripts.

Comandos básicos

Uma vez iniciado o container a partir de uma imagem é possível gerenciar a utilização com novos comandos. O comando **stop** envia um sinal **SIGTERM**, caso não desligue, o container receberá um **SIGKILL** após 10 segundos. Para iniciar o container novamente se utiliza o comando **start**.

$ docker container **stop** meu_python

$ docker container **start** meu_python

Engenharia da Computação

Dockerfile

Dockerfile é um arquivo que aceita rotinas em shell script para serem executadas. A seguir veja como ficaria o processo de instalação do Nginx.

```
FROM ubuntu
LABEL name="satcnginx"
MAINTAINER Gledson Scotti <gledson.scotti@satc.edu.br>
RUN apt update
RUN apt install nginx -y
```

Analisando as partes:

- FROM – estamos escolhendo a imagem base para criar o container;
- MAINTAINER – especifica o nome de quem vai manter a imagem;
- RUN – permite a execução de um comando no container.

Engenharia da Computação

Dockerfile

Vamos gerar a nossa primeira imagem com a opção build, note que o sinal de “.” especifica o local do arquivo dockerfile.

$ sudo docker build -t nginx .

$ sudo docker images

Criando container com mapeamento de portas em seguida testando dentro do container:

$ docker run -d -p 8000:80 nginx /usr/sbin/nginx -g "daemon off;"

$ curl -IL http://localhost:8000

Redes no Docker

O docker possui três redes configuradas por padrão. Essas redes oferecem configurações específicas para gerenciamento do tráfego de dados. Para visualizar essas interfaces, basta utilizar o comando abaixo:

$ docker network ls

NETWORK ID	NAME	DRIVER	SCOPE
9e5de7d4b862	bridge	bridge	local
F9f060462a07	host	host	local
D507b5afc525	none	null	local

Todo container iniciado no docker é associado a uma rede específica. **Bridge** é a rede padrão para qualquer container, a menos que o associemos a outra rede. A rede confere ao container uma interface que faz bridge com a interface **docker0** do docker host. Essa interface recebe, automaticamente, o próximo endereço disponível na rede IP 172.17.0.0/16. A rede **None** tem como objetivo isolar o container para comunicações externas. Já a rede **Host**, tem como objetivo entregar para o container todas as interfaces existentes no docker host.

Engenharia da Computação

Redes no Docker

O docker possibilita que o usuário crie redes e estas redes são associadas ao elemento que o docker chama de driver de rede. Cada rede criada por usuário deve estar associada a um determinado driver.

$ docker network create --driver bridge isolated_satc

$ docker network list

Iniciando um container na rede isolated_satc.

$ docker container run -itd --net isolated_satc ubuntu bash

Inspecionando a rede criada.

$ docker network inspect isolated_satc

Engenharia da Computação

Lista de comandos docker

$ sudo docker info => informações do nosso Docker Host.

$ sudo docker version => verificar informações sobre versão de docker client/server.

$ sudo docker images => verificar imagens disponíveis localmente.

$ sudo docker search <X> => procura por uma imagem descrita em “X” (ubuntu, apache, ngnix).

$ sudo docker pull <X> => baixa a imagem descrita em “X”.

$ sudo docker run <X> => executa a imagem descrita em “X”. Se não possui a imagem o docker baixa.

$ sudo docker ps => mostra os container sendo executados. Junto com “-a” mostra todos.

$ sudo docker stats <ID> => mostra status (CPU, MEM,...) do container. Pode ser especificado o ID ou apelido.

$ sudo docker inspect <ID> => mostra maiores detalhes de seu container em formato “json”.

$ sudo docker rmi <img> => remove a imagem, onde “img” é o nome da imagem ou ID.

$ sudo docker rm <ctn> => remove um container, onde “ctn” é o nome do container ou ID.

$ sudo docker attach <ID> => para entrar no container desejado, onde pode ser especificado o ID ou nome do container.

$ sudo docker exec <ID> => executa comando no container remotamente sem precisar estar na console do mesmo. Pode ser o ID ou nome do container.

$ sudo docker start/stop <ID> => comando para iniciar ou parar um container.

Engenharia da Computação

Lista de comandos docker

Vejamos alguns dos parâmetros que podemos utilizar com docker:

-i permite interagir com o container

-t associa o seu terminal ao terminal do container

-it é apenas uma forma reduzida de escrever -i -t

--name algum-nome permite atribuir um nome ao container em execução

-p 8080:80 mapeia a porta 80 do container para a porta 8080 do host

-d executa o container em background

-v /pasta/host:/pasta/container cria um volume '/pasta/container' dentro do container com o conteúdo da pasta '/pasta/host' do host

$ sudo docker run -it -d ubuntu /bin/bash

(Inicia o ubuntu em bash, porém detached ou separado)

$ sudo docker run -it -p 8080:80 nginx

(Inicia o nginix na porta 80 atendendo “externamente” na porta 8080.)

Lista de comandos docker

O nosso container com o nome app01 está com o limite para memória definido e será exibida em bytes.

docker run -it -m 512M — name app01 debian

docker inspect app01 | grep -i mem

Para subir um container com um limite de CPU.

docker run -it --cpus=0.5 --name app02 debian

docker inspect app02 | grep -i cpu

O Docker oferece o comando “docker update” que nos permite a alteração em configurações.

docker container update -m 256m — cpus=1 app_novo01

Exportar container docker:

docker export meucontainer | gzip > meucontainer.gz.

zcat meucontainer.gz | docker import – meucontainer

docker run -it meucontainer bash

Engenharia da Computação

Ferramentas Visuais...

1. Kitematic
2. Portainer
3. DockStation
4. Shipyard
5. Docker Compose UI
6. Rancher