**Docker Document**

O que é Container:

 Tudo pode ser instalado no servidor e é armazenado nos containers. Dessa forma, os mesmos softwares e as suas versões podem ter uma execução facilitada em qualquer ambiente de desenvolvimento.

Facilitando atualização, manutenção, instalação e uso dos softwares.

Emulação da sua aplicação.

Ordem de aplicação em container:

* Server
* Server OS
* Container Engine:

1. Container – ap1
2. Container – ap2
3. Container – ap3

O Docker utiliza apenas um Kernial para o docker.

Portabilidade com container funciona em qualquer máquina.

A função do container é emular sua aplicação.

**Camadas e o Copy-on-Write**

O Docker ele utiliza 2 storage driver.

**AUFS** – Primeiro File system para o docker, funcionando em camada. Para criar uma imagem voe vai sobrepondo. Sempre a ultima camada é onde pode ler e escrever.

Caso um arquivo que seja de leitura ou que precise ser reescrito o Docker o joga para a camada onde se pode ler e escrever (ultima camada) para que consiga ser lida e escrita.

**Copy-on-Write –** Quando se pega um arquivo e ao invés de salvar diretamente no original ele cria uma copia e salva suas alterações nelas.

**Docker internas, Namespaces, CGROUP, Netfilter Container**

**Docker internas:**

**NameSpaces:** Ele que permite fazer o isolamento de certos componentes do sistema operacional. Responsavel por fazer o container ter seu próprio ambiente, tendo ip, ponto de montam.

**PADnamespace**: Responsavel por isolar, fazendo o container ter sua propia arvore de processo de identificação.

**NetNameSpace**: Isolamento de rede. Responsavel por fazer a comunicação.

**MNTNameSpace** – Isolamento de file system.

**IPCNameSpace** –

**UTSNameSpace** – Isolamento de host name do container, versão e tudo mais.

**UserNameSpace** – Responsável por fazer isolamento de usuário.

**CGROUPS** – Isolamento de CPU, memoria. Responsável por limitar, liberar recursos para os container.

**NetFilter**- Modulo de iptables. Não mecha nas regas do iptable.

**Instalando Docker**

**Docker so roda em processador 64 bits.**

**Kernial v3.8**

Linhas de comando:

<https://docs.docker.com/engine/installation/>

curl –fsSL <https://get.docker.com/> | sh

**Administrando Container**

Docker run hello word – Utilizado para executar um container

Docker ls –a - mostra todos os containers da maquina

Docker images – mostra todas as imagens que se tem no container

Docker run –ti – Para que traga um terminal e interação com o container

Docker run -d – Para que o container rode em background

Cntr+d – Terminar o container

Cntr+p+q – Sai do container e deixa ele ainda funcionando

Docker attach ID – Para voltar para dentro do container, colocano o ID do container assim voltando para dentro do mesmo.

Docker creat – Ele cria o container mas ele fica parado.

Docker run –ti – para iniciar o container que estava parado.

Docker stop ID – para para o container.

Docker start ID – para iniciar o container

Docker pause ID – para pausar o container

Docker unpause ID – para despausar o container

Docker status ID – para ter informações da CPU, Memoria, Rede.

Docker top id – Processos que estão rodando nesse continer

Docker logs ID – Vai trazer as informações de loger.

Docker rm ID – Para removar o container.

Docker rm –f ID – para remover um container que esta em execução

Docker inspect ID- Consegue visualizar todas as informações incluindo memoria e cpu.

Docker –name nometeste – Escolhe um nome para o container

**Configurar CPU e MEM dos Containers**

**Limitar memoria**

**Na inicialização**

Primeiro na linha de criação do container coloca essa linha para definir o numero memoria e o nome do container. Para já no inicio da criação do container.

docker run -ti –memory ou -m 512m --name teste2 debian

Para ver se esta funcionando corretamente pode usa-se a seguinte linha

docker inspect 516b6035415a | grep -i mem

E ele vai mostrar apenas o consume de memoria que foi limitado na hora da criacao do container.

**Quando o contêiner já foi criado**

Docker update –m 256m teste2 ( nome ou ID) e assim faz a alteração .

Para verificar a alteração usa-se

Docker inspect teste | grep –i mem

**Limitar CPU**

Para estipular o valor usa-se a porcentagem em cima do valor total da CPU.

Comando

Docker run –ti --cpu-shares 1020 –nome contaner1 debian

Para ver se esta funcionando pode ver nessa linha alterando o nome

docker inspect maq3 | grep -i cpu

Caso a operacao esteja em operacao basta usar o commando

Docker update –cpu-shares 512 maq1

**Volumes e container data-only**

**O que e volume**

E uma forma de colocar um diretório dentro do container fora do filesistem do container. Compartilhamento do diretório com o container.

**Para localizar um volume pode-se utilizar**

Docker inspect –f {{.Mounts}} ID

**Mapeamento de dockerfiles**

**//** Criando a pasta dentro do root

1 - mkdir /root/primeiro\_dockerfiles

// Apontando a pasta do docker dentro do docker

2 - docker run -ti -v /root/primeiro\_dockerfiles:/volume debian

**Data Only**

Não precisa esta rodando e ele vai compartilhando volume com outros containers.

Docker create –v /data –name dbdados centos

--volume-from – Ele importa outro volume de outro container para o container que ele ta criando agora.

**Docker Files**

Instrucao de como bildar/construir a imagem.

Na pasta destino do projeto criar o Dockerfile.

Comandos Dockerfile:

FROM: Imagem que esta usando como base. Ex: debian, node, php.

RUN : Comando usado para instalar depedencias. Mas quanto menos melhor. Mas quando e feito em uma linha fica com uma quantidade menor de camadas e assim deixando mais fácil de fazer manutenção.

**ADD:** Jogas os arquivos da maquina host para dentro do container.

**CMD**: Comando para entry point. Principal processo do container. Quando não passa o entrey point ele cria automaticamente com o bash.

**LABEL**: Descricao, versão, fabricante.

**COPY**: copia os arquivos da maquina host para dentro do container.

**ENTRYPOINT**: Permite que deixe um processo seja o principal processo dentro do container. Caso ele ”morra” o container também morre. [“usr/bin/apachc2ctl”]

**ENV**: Determina variáveis de ambiente para enviar para o container.

EXPOSE 80 – Coloca para ele ouvir a porta que esta exposta.

**USER** : Usuario default da maquina. Quando não declaro e o root

**Workdir**: Falando que o diretório e a raiz do container.

Vomlume : Para criar um volume no container

**MAINTAINER:** o Nome de quem crio o container.

**Criando a imagem em cima do Dockerfile**

Docker bild –t imagem:1.0 .

Tem que ter o ponto no final caso esteja na pasta destino para assim conseguir acessar a pasta do diretório correto.