**A Orquestração Contêineres Com Docker VM Container**

* Compartilham o mesmo sistema operacional;
* Tudo que for realizado dentro da aplicação em um ambiente de container é imutável;
* Por exemplo, subir uma nova imagem, e se caso estiver tudo certo pode ser alterada a imagem.;
* Possibilidade de controlar a utilização de memórias/cpu em nossos containers;
* O backup se ocorrer algum problema torna-se nossa própria imagem;
* Diferencial é a utilização de um escalonamento horizontal e fácil atualização;
* Existe uma camada de proxy/servidor entre app e os containers, por exemplo, intercalando a entrega desses serviços, mediante a capacidade de uso;
* Utilização do DockerSWARM/DockerCOMPOSE e o Kubernetes;
* Escalonamento vertical é a ação de aumentar os recursos utilizados como CPU e Memória.

**Arquitetura do DOCKER**

* Container Image é o pacote com todas as dependências para criar nosso container;
* Dockerfile é um arquivo de texto que contém todas as instruções para fazer o buid da nossa imagem;
* Build é a ação que cria a imagem a partir do Dockfile;
* A partir das instruções do dockerfile e a geração da imagem do build, é instanciada o container;
* Container então é a instância da nossa imagem que representa a execução da nossa aplicação ou um processo ou um serviço;
* Volumes permite que armazenam dados em disco, ou seja se o container “morrer” os dados em salvos em volume não se perdem;
* Tag é a ajuda no versionamento das nossas imagens;
* Multi-Stage Build contém múltiplos estágios do build;
* Repository nada mais é do que uma caixa com as imagens;
* Registry é um serviço que prove o acesso do docker ao repository;
* Docker hub então seria um repositório publico, podem terem arquivos publicos e privados;
* Compose é uma ferramenta que podemos criar múltiplos container com simples comando.

**Processo de Funcionamento**

Docker\_Client (docker build, docker pull, docker run) - Docker\_Host(Docker Daemon) - Registry - Images - Containers

**Criar Conteiner**

<https://labs.play-with-docker.com/> Utlizar o docker sem instalação na máquina.

Comandos:

**docker run –name nome-do-projeto -p 80:80 -d nginx** (Cria o container)

**docker images** (Lista as Imagens contidas no servidor)

**docker ps** (Lista detalhes)

**docker rm nome-do-projeto** (Remove o Container)

**docker rm -f nome-do-projeto** (Remove a força o container caso esteja rodando)

**docker rmi nginx** (Remove a imagem)

**Instalação do Docker no Windows**

Baixar direto do site <https://www.docker.com/products/docker-desktop>

Documentação do docker <https://docs.docker.com/>

Comandos Testes (Power Shell)

**docker ps**

**docker run** projeto-teste

***Observação:***

* *Para rodar em produção utilizar ambiente em nuvem.*
* *Para desenvolvimento utilizar desktop.*

**Principais Comandos**

**run** - Criar o container;

**ps** - Lista os containers;

**info** - Traz as informações do nosso docker;

**images** -Imagens para criar nossos containers;

**exec** - Usado para executar um outro binário em nosso container;

**stop** - Parar execução;

**start** - iniciar execução;

**logs** - Lista os logs de nosso container;

**inspect** - Lista todas as configurações do nosso container;

**pull** - Baixar as imagens do nosso repositório público ou privado;

**commit** - Para comitar modificações nos nossos containers;

**tag** - melhorar versionamento;

**login/logout** - Usar para logar em nosso repositório privado ou público;

**push** - Após realizar o build, possibilita compartilhar ou armazenar em nosso repositório;

**search** - Procurar uma imagem;

**rm** - Remover um container;

**rmi** - Remover uma imagem;

**export/import** - Exportar ou importar um container

**save/load** - Salvar uma imagem

**Tipos de Rede**

**Bridge**: É a rede default do Docker, utilizado para comunicação entre containers.

**Host:** Remove o isolamento de rede, o container responde diretamente pela placa de rede do host.

**Overlay**: Permite a comunicação entre containers de hosts diferentes.

**Macvlan**: Permite atribuir um endereço MAC ao container tornando ele visível como um dispositivo físico na rede.

**Tipos de Armazenamento no Docker**

**Volume**

Analogia a um disco virtual, onde a docker engine tem total autonomia sobre ele sendo assim o host não precisa conter uma estrutura de arquivos, tornando um carregamento um pouco mais demorado a leitura do disco no sistema virtual. Aconselhado para utilização em ambientes replicados.

**Bind Mounts**

É mapeado um arquivo/pasta do host para um arquivo/pasta dentro do container. Seria uma analogia de um compartilhamento de pastas do host com o container.

***Obs.:***

***Diante dos dois tipos de armazenamento acima, se o container for removido ou reiniciado as informações são persistidas em disco.***

**tmpfs Mounts**

Armazenamento temporário. Quando for necessário reiniciar esses arquivos são deletados, permitindo a utilização de seus recursos.

**Outras Observações:**

* O parâmetro ***run*** nos permite instalar aplicação do build da nossa imagem;
* Os parâmetros do Docker run, que utilizamos para limitação de CPU e Memória são: ***--cpus*** e ***--memory***;
* O Docker-Compose.yaml guarda todas as definições de quais container precisamos e suas características;
* Precisamos de apenas 1 node para iniciarmos um cluster de swarm;
* O parâmetro **expose** do Dockefile, permite a conexão na porta da aplicação criada
* A **memória** é um recurso importante a ser limitado;
* O **DockerHub** é um repositório de imagens.