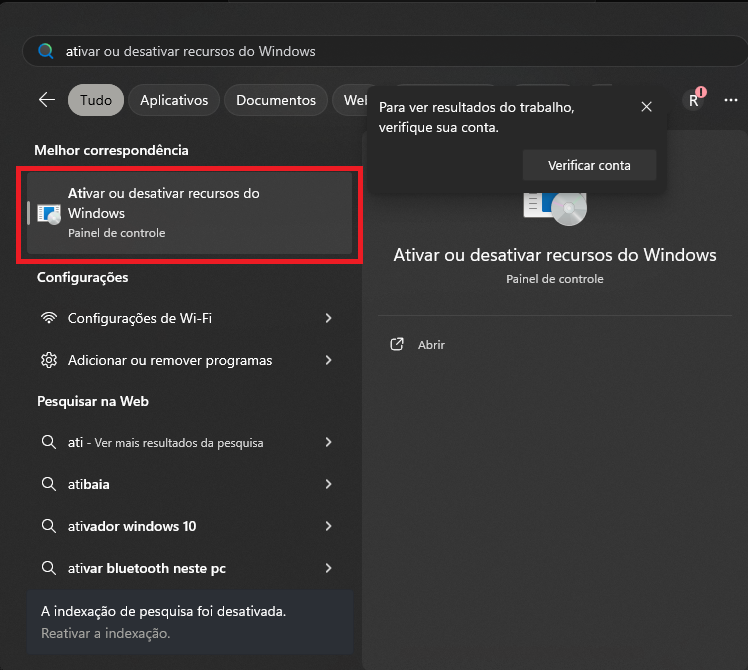
1. **CONFIGURANDO O AMBIENTE**

O Docker Engine é o core do Docker. Ele é responsável pela criação e gerenciados container dentro do SO. E esse serviço é baseado no kernel Linux. Por isso precisamos de instalar o WSL do Linux para executarmos o Docker Engine. Esse serviço é executado no Linux, sendo necessário a instalação do Docker Engine no Linux. No caso do Windows e do MacOS necessitamos a instalação do Docker desktop que será responsável por rodar o Docker no SO. Assim sendo o Docker Desktop não necessita da instalação de uma VM Linux, ela utilizaria o WSL para fazer esse papel de SO.

* 1. Devemos verificar a instalação do WSL. Para isso devemos pesquisar no menu Iniciar do Windows por Ativar ou desativar recursos do Windows.

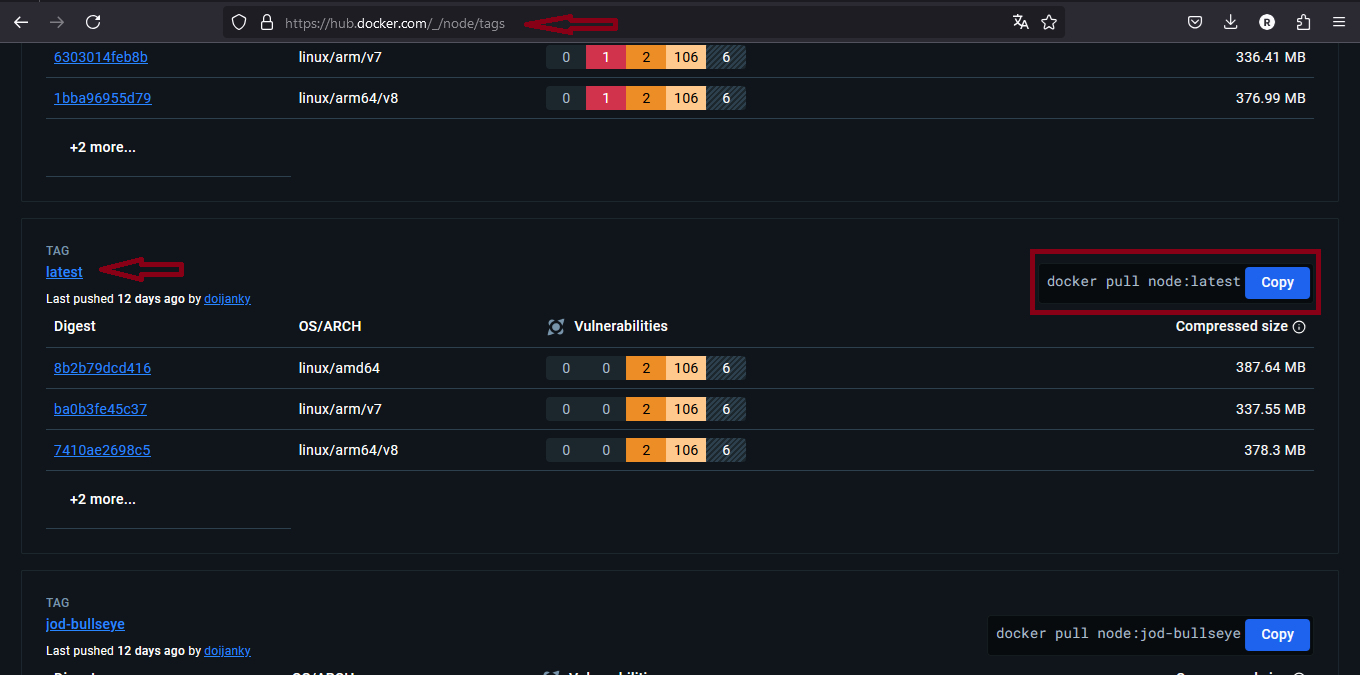


* + 1. Devemos verificar se o recurso está habilitado. No caso de não estar habilitado é aconselhável que reiniciemos o SO mesmo se não for solicitado.

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

* + 1. Devemos acessar o PowerShell como administrador e iremos executar o comando “wsl –status”. Esse comando irá verificar se o WSL está instalado no SO. Após essa verificação devemos atualizar o WSL através do comando “wsl –update”.
  1. Próximo passo é instalar o Docker Desktop através de download no site do Docker. Aconselhável reiniciar o SO após a instalação do Docker Desktop.
  2. Comando “docker –version” no PowerShell ou CMD, confirma que o Docker está instalado e mostra a versão dessa instalação.
  3. No caso do Docker compose, devemos verificar a versão através do comando “docker compose version”. Nesse caso, iremos instalar o VCSCode, realizando o download na página do VSCode (code.visualstudio.com), para realizar os comandos do Docker. Instalando as extensões WSL e Docker no VSCode. Selecione no momento da instalação as opções “Registre Code como editor para todos os tipos de arquivos suportados” e “Adicione em PATH (disponível após reiniciar)”, com a segunda opção podemos utilizar os comandos de programação via linha de comando, pode facilitar dependendo da tarefa que deseja realizar.

1. **INTRODUÇÃO AO DOCKER**
   1. Docker cliente e Docker Server (deamon) – Client é a interface de linha de comando do Docker, através dele vamos disparar os comandos que serão enviados para o Docker Server. No caso do SO Windows e macOS, esses comandos serão enviados para a Docker desktop que foi instalado e ele será o responsável para enviar para o Docker Server e em ambos os casos uma API Rest é responsável por esse envio dos comandos para o Server seja de forma direta como no Linux ou através do Docker Desktop.
   2. Configurando o acesso ao Docker Sock no Linux – o Docker Sock fica entre o Client e o Docker Server no Linux. Nele está contido por exemplo as configurações de usuários do Docker. Ele fica localizado na pasta /var/run/docker.sock. No caso dos usuários por exemplo do Docker, eles são criados dentro do grupo de usuários Docker no Linux. Portanto se quisermos configurar um usuário para o Docker, deverá ser feito dentro desse grupo.
   3. Imagens do Docker Hub – Imagem é um arquivo executável e autocontido com tudo que é necessário para executar o seu objetivo. O Docker Server já vem por padrão programado para se comunicar com o Docker hub (hub.docker.com). Esse repositório temos diversas versões de programas que as empresas disponibilizam além de imagens disponibilizada pelas comunidades. Na página do serviço, por exemplo Nginx podemos baixar a imagem através do comando docker pull nginx.
   4. Containers – instância de uma imagem em execução. Uma imagem pode gerar vários containers.
2. **IMAGES**
   1. Ao acessarmos o Docker Hub, temos muitas versões de uma mesma aplicação que pode ser utilizada nos projetos. Uma boa prática é criar uma imagem com a última versão da aplicação e inserir o app desenvolvido que utiliza a aplicação para realização de testes de compatibilidade. A tag latest é sempre a versão mais atual e lts é a versão estável da aplicação. 

Como pode ser visto na imagem temos a tag latest, essa é a última versão do node e para realizarmos o download da imagem utilizamos o comando Docker pull node:latest.

Uma imagem do Docker é imutável e a utilizaremos no template que será a base para colocarmos o container em execução. Dockerfile é o arquivo que é automaticamente reconhecido pelo Docker para configurar uma imagem que será recuperada do registry (base do Docker).

Ao excluirmos os arquivos locais, não mudará nada, pois tudo ficará gravado no Docker.

TARGET IMAGE – Nome dados para a imagem dentro do Docker Server

SOURCE IMAGE – ID da imagem, atrelado a versão da imagem

No momento da criação do repositório no Docker hub, podemos escolher se iremos criar um repositório público (qualquer pessoa com conta no docker hub pode acessar) ou privada (somente o usuário da conta pode acessar o conteúdo do repositório). Dependendo da conta, podemos criar somente um repositório privado.

Tela de celular com aplicativo aberto

Descrição gerada automaticamente

É possível a criação de um registry privado, ou seja, instalar o registry dentro de uma VM Linux para utilizar em sua rede local sem a necessidade de acesso ao docker hub. Basta instalar a extensão a seguir na VM.

Tela de celular com publicação numa rede social

Descrição gerada automaticamente

**Volumes:** São diretórios externos aos containers e servem para resolver problemas de persistência e de compartilhamento de dados.

Quando criamos uma imagem (Dockerfile), essa camada é somente leitura, isso quer dizer que esse arquivo não sofrerá mudanças para ser transformado em container. Quando realizamos o build dessa imagem, é transformado em uma nova camada de escrita, que pode ser alterado conforme necessidade. O problema é quando excluímos esse container, todas as informações e dados nele contidos são excluídos junto com ele. Para isso é que utilizamos os volumes. Para armazenar dados que não podem ser eliminados juntamente com o container, pois o container é excluído constantemente no processo de utilização.

Tipos de montagem:

* Bind Mount: Armazenamento persistente na máquina física onde está o container (Host), nesse caso não será perdida a informação no caso de reiniciar do host. É muito utilizado em período de desenvolvimento que os containers devem ser alterados constantemente, com esse tipo de armazenamento, alteramos as informações do container e atualizamos o modelo no volume persistente que automaticamente será atualizado no container. Podemos utilizar o comando a seguir para manter uma conexão entre o arquivo do host e o container, assim quando atualizarmos um automaticamente essa atualização reflete no outro, tanto do host para o container quanto do container para o host.

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

No caso a seguir é a configuração quando queremos que a alteração seja feita do host para o container, mas não do container para o host.

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

O Bind Mount tende a ser um pouco lento quando utilizado com o Windows e o macOS, pois o container é realizados todas as suas configurações através do docker desktop, assim quando ele utiliza todo o caminho para as alterações ele tende a ficar lento. Portanto não é muito aconselhável a utilização em produção. Para ser mais produtiva a utilização do docker no Windows um bom conselho seria a utilização do WSL.

* Docker volume: Armazenamento persistente na máquina física onde está o container (Host), nesse caso não será perdida a informação no caso de reiniciar do host. Esse é o modo de armazenamento mais utilizado no docker. Podemos criar um volume a atachá-lo em vários containers. Também é utilizado quando não temos o diretório do disco no host, assim sendo, criamos um volume, pois ele não é associado ao host e sim ao docker server. Utilizado quando formos associar o volume a um host na nuvem, assim nós criamos o volume em uma VM na nuvem e associamos ao docker server. Quando estamos trabalhando com o Windows e macOS, utilizamos o docker volume, pois assim diminui a latência no container. Quando a aplicação deve ter controle total sober as operações de entrada e saída.

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Pode ser criado da mersma forma que o mount um volume somente leitura, assim a alteração realizada no docker não vai influenciar na imagem base no volume.

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

* Tmpfs: armazenamento volátil na RAM da máquina, nesse caso ao reiniciar o host, perderemos a informação, porém esse caso o acesso à informação será bem mais rápido do que o anterior, útil em um caso de alto volume de dados para serem acessados.

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Redes do Docker: Utilizado para a comunicação entre containers e com a internet. Existem drivers padrão para a rede do Docker. São elas:

* Bridge – mais utilizado
* Host
* Overlay
* Ipvlan
* Macvlan
* None

Assim como as images, container e volumes as redes são gerenciadas pelo Docker Deamon.

Tem 2 características:

* Comunicação isolada, ou seja, eles se comunicam entre os containers que estão na mesma rede e a internet.
* A rede funciona como um NAT, traduzindo os endereções do host hospedeiro, no caso a VM local, essa característica existe somente na rede bridge. E quando criamos uma rede, automaticamente é criado um gateway que é responsável pelo tráfego no container e internet.

O driver host funciona como se estivesse comunicando diretamente com o host, sem a necessidade de NAT, a principal vantagem é o ganho de performance além de eliminar os custos na realização do NAT.

A rede null tem como característica o isolamento completo na rede, não havendo comunicação com internet e nem com outro container na mesma rede.

Docker Compose: gerenciamento de múltiplos containers, é utilizado por exemplo em um ambiente completo com bd, frontend, backend etc, assim temos múltiplos containers, cada um com um serviço instalado.

O arquivo principal desse orquestrador é o Docker-compose ou somente compose e a extensão pode ser yaml ou yml. O mais utilizado é o Docker-compose.yaml.

A seguir temos as configurações de condition do depends on health

Diagrama

Descrição gerada automaticamente