SÃO
PAULO
TECH
SCHOOL

Computação e sistemas distribuídos

Arquitetura e Containers pt.2

Eduardo Verri, Diego Brito

eduardo.verri@sptech.school

diego.brito@sptech.school

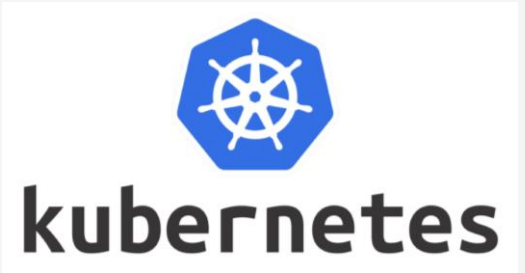
Orquestração de containers

Conforme as empresas começaram a adotar containers, geralmente como parte de arquiteturas modernas nativas da cloud, a simplicidade do container individual começou a entrar em conflito com a complexidade do gerenciamento de centenas (ou até milhares) de containers em um sistema distribuído.

Para superar esse desafio, a orquestração de containers surgiu como uma forma de gerenciar grandes volumes de containers ao longo de seu ciclo de vida

- Provisionamento
- Redundância
- Monitoramento do funcionamento
- Alocação de recursos
- Ajuste de escala e balanceamento de carga

Orquestração de containers

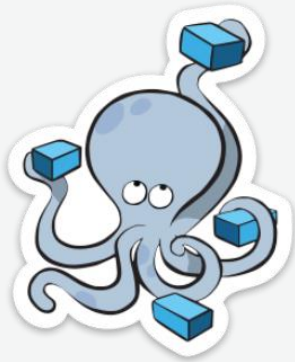


Outras orquestradores (como Apache Mesos, Nomad e Docker Swarm) foram criadas, entretanto o **Kubernetes (k8s)**, um projeto de código aberto apresentado pelo Google em 2014, se tornou a plataforma de orquestração de containers mais conhecida, e é usado como padrão pela maioria dos setores de mercado.

Ele permite que seja declarado um estado desejado do ambiente de container através de arquivos YAML e faz todo o trabalho para estabelecer e manter esse estado, com atividades que incluem a implementação de um número de instâncias de um determinado aplicativo ou carga de trabalho, a reinicialização do aplicativo em caso de falha, o balanceamento de carga, o ajuste de escala automático, etc.

Atualmente, é operado pela Cloud Native Computing Foundation (CNCF), um grupo independente de fornecedores do setor, sob os cuidados da Linux Foundation.

Docker Compose



- É uma ferramenta que permite executar ambientes de aplicativos com vários containers com base em definições definidas em um arquivo YAML. Ele usa definições de serviço para criar ambientes totalmente personalizáveis com vários containers que podem compartilhar redes e volumes de dados.

```
# Manualmente (escolhendo a versão)
sudo curl -L "https://github.com/docker/compose/releases/download/v2.26.1/docker-compose-$(uname -s)-$(uname -m)" -o
/usr/local/bin/docker-compose

sudo chmod +x /usr/local/bin/docker-compose

# plugin
sudo apt-get install docker-compose-plugin
docker-compose –version
```

- How to install compose
- Releases · docker/compose (github.com)
- Sample apps with Compose | Docker Docs
- docker/awesome-compose: Awesome Docker Compose samples (github.com)
- Try Docker Compose | Docker Docs

O arquivo Compose

- O caminho padrão para um arquivo Compose é **compose.yaml** (preferencial) ou **compose.yml** que é colocado no diretório de trabalho.
- O Compose também oferece suporte a **docker-compose.yaml** e **docker-compose.yml** para compatibilidade com versões anteriores.
- Se ambos os arquivos existirem, o Compose preferirá o **compose.yaml** canônico.
- Vários arquivos do Compose podem ser mesclados para definir o modelo do aplicativo. A combinação de arquivos YAML é implementada anexando ou substituindo elementos YAML com base na ordem do arquivo Compose que você definiu.

```yaml
services:
  frontend:
    image: example/webapp
    ports:
      - "443:8043"
    networks:
      - front-tier
      - back-tier
    configs:
      - httpd-config
    secrets:
      - server-certificate
```

Modelo de aplicação do Compose

- Os componentes de computação de um aplicativo são definidos como **serviços**. Um conceito abstrato implementado em plataformas executando a mesma imagem e configuração de container uma ou mais vezes.
- Os serviços se comunicam entre si por meio de **redes**. Uma rede é uma abstração de capacidade de plataforma para estabelecer uma **rota IP entre containers** dentro de serviços conectados entre si.
- Os serviços armazenam e compartilham dados persistentes em **volumes**. Dados persistentes são como uma montagem de sistema de arquivos de alto nível com opções globais.

Versão e nome dos elementos

- **Versão (opcional):** A propriedade ***version*** serve para compatibilidade com versões anteriores. É apenas informativo. O Compose não usa versão para selecionar um esquema exato para validar o arquivo do Compose, mas prefere o esquema mais recente quando ele é implementado.
- **Nome:** A propriedade ***name*** é o nome do projeto a ser usado se você não definir um explicitamente. O Compose oferece uma maneira de substituir esse nome e define um nome de projeto padrão a ser usado se o elemento de nome de nível superior não estiver definido.

```yaml
name: myapp

services:
  foo:
    image: busybox
    command: echo "I'm running ${COMPOSE_PROJECT_NAME}"
```

Serviços

- Um serviço é uma definição abstrata de um recurso de computação dentro de uma aplicação que pode ser dimensionado ou substituído independentemente de outros componentes. Os serviços são apoiados por um conjunto de containers, definidos por uma imagem Docker, executados pela plataforma de acordo com os requisitos de replicação e restrições de posicionamento.
- Cada serviço também pode incluir uma seção de construção, que define como criar a imagem Docker para o serviço. O Compose oferece suporte à construção de imagens do Docker usando esta definição de serviço.
- Cada serviço define restrições e requisitos de tempo de execução para executar seus containers.
- Services top-level elements | Docker Docs

Serviços - alguns parâmetros

- **image:** especifica a imagem a partir da qual iniciar o container. Se a imagem não existir na plataforma, o Compose tentará extraí-la com ***pull***.
- **ports:** expõem as portas do container. Se o IP do host não estiver definido, ele será vinculado a todas as interfaces de rede. As portas podem ser um valor único ou um intervalo. Especifique ambas as portas (HOST:CONTAINER) ou apenas a porta do container. Neste último caso, o tempo de execução do container aloca automaticamente qualquer porta não atribuída do host. HOST:CONTAINER deve sempre ser especificado como uma string (entre aspas), para evitar conflitos.
- **volumes:** os volumes definem caminhos de host de montagem ou volumes nomeados que são acessíveis por containers de serviço.
- **depends_on:** expressa dependências de inicialização e desligamento entre serviços. A variante de sintaxe curta especifica apenas nomes de serviços das dependências.

Serviços - alguns parâmetros

- **networks:** define as redes às quais os containers de serviço estão anexados, referenciando entradas na chave de redes de nível superior.
  - **aliases** declara nomes de host alternativos para o serviço na rede. Outros containers na mesma rede podem usar o nome do serviço ou um alias para se conectarem a um dos containers do serviço. O mesmo serviço pode ter aliases diferentes em redes diferentes.
- **deploy:** especifica a configuração para a implantação e o ciclo de vida dos serviços.
  - **resources:** configura restrições de recursos físicos para que o container seja executado na plataforma. Essas restrições podem ser configuradas como limites e reservas
    - **cpus:** configura um limite ou reserva para quantos recursos de CPU disponíveis, como número de núcleos, um container pode usar.
    - **memory:** configura um limite ou reserva na quantidade de memória que um container pode alocar, definido como uma string que expressa um valor de byte.

Redes

- O elemento de redes de nível superior permite configurar redes nomeadas que podem ser reutilizadas em vários serviços. Para usar uma rede em vários serviços, você deve conceder explicitamente acesso a cada serviço usando o atributo de redes no elemento de nível superior de serviços.
- No exemplo a seguir, em tempo de execução, as redes front-tier e back-tier são criadas e o serviço front-end é conectado às redes front-tier e back-tier.
- Networks top-level elements | Docker Docs

```
services:
  frontend:
    image: example/webapp
    networks:
      - front-tier
      - back-tier

networks:
  front-tier:
  back-tier:
```

Redes - alguns parâmetros

- **driver:** especifica qual driver deve ser usado para esta rede. O Compose retornará um erro se o driver não estiver disponível na plataforma.
  - **bridge:** O driver de rede padrão. Comumente usadas quando seu aplicativo é executado em um container que precisa se comunicar com outros containers no mesmo host.
  - **host:** remove o isolamento de rede entre o container e o host Docker e use a rede do host diretamente. Consulte Driver de rede host.
  - **overlay:** redes de sobreposição conectam vários daemons Docker e permitem que serviços e container Swarm se comuniquem entre nós (funciona apenas em Linux).
  - **ipvlan:** As redes IPvlan oferecem aos usuários controle total sobre o endereçamento IPv4 e IPv6
  - **macvlan:** As redes Macvlan permitem atribuir um endereço MAC a um container, fazendo-o aparecer como um dispositivo físico na sua rede.
  - **none:** isola completamente o container do host e outros containers

Redes - alguns parâmetros

- **attachable:** se setado como **true**, os containers autônomos deverão ser capazes de se conectar a essa rede, além dos serviços. Se um container independente for conectado à rede, ele poderá se comunicar com serviços e outros containers independentes que também estejam conectados à rede.
- **name:** define um nome personalizado para a rede. O campo de nome pode ser usado para fazer referência a redes que contêm caracteres especiais. O nome é usado como está e não tem como escopo o nome do projeto.

```yaml
networks:
  mynet1:
    driver: overlay
    attachable: true
```

```yaml
networks:
  network1:
    name: my-app-net
```

Redes - resumo driver

- A rede bridge padrão é boa para executar containers que não requerem recursos especiais de rede
- As redes bridge definidas pelo usuário permitem que containers no mesmo host Docker se comuniquem entre si. Uma rede definida pelo usuário normalmente define uma rede isolada para vários containers pertencentes a um projeto ou componente comum.
- A rede host compartilha a rede do host com o container. Quando você usa esse driver, a rede do container não fica isolada do host.

Redes - resumo driver

- Overlay são melhores quando você precisa de containers executados em diferentes hosts Docker para se comunicar ou quando vários aplicativos trabalham juntos usando serviços Swarm.
- Macvlan são melhores quando você está migrando de uma configuração de VM ou precisa que seus containers se pareçam com hosts físicos em sua rede, cada um com um endereço MAC exclusivo.
- IPvlan é semelhante ao Macvlan, mas não atribui endereços MAC exclusivos aos containers. Considere usar IPvlan quando houver uma restrição no número de endereços MAC que podem ser atribuídos a uma interface ou porta de rede.

Volumes

- Os volumes são armazenamentos de dados persistentes implementados pelo mecanismo de container.
- A declaração de volumes permite configurar volumes nomeados que podem ser reutilizados em vários serviços. Para usar um volume em vários serviços, você deve conceder explicitamente acesso a cada serviço usando o atributo volumes no elemento de nível superior de serviços.
- Uma entrada na seção de volumes de nível superior pode estar vazia; nesse caso, ela usa a configuração padrão do mecanismo de container para criar um volume.
- Volumes top-level elements | Docker Docs

Volumes

- O exemplo a seguir mostra uma configuração de dois serviços em que o diretório de dados de um banco de dados é compartilhado com outro serviço como um volume, denominado db-data, para que possa ser feito backup periodicamente.
- O volume db-data é montado nos caminhos do container /var/lib/backup/data e /etc/data para backup e back-end, respectivamente.
- A execução do **docker compose up** cria o volume, caso ele ainda não exista, senão, o volume existente será usado

```
services:
  backend:
    image: example/database
    volumes:
      - db-data:/etc/data

  backup:
    image: backup-service
    volumes:
      - db-data:/var/lib/backup/data

volumes:
  db-data:
```

Comandos básicos Docker compose

`docker compose ps`: mostra informações sobre os containers questão rodando, e seus estados

`docker compose logs`: para checar os logs produzidos pela sua aplicação conteinerizada

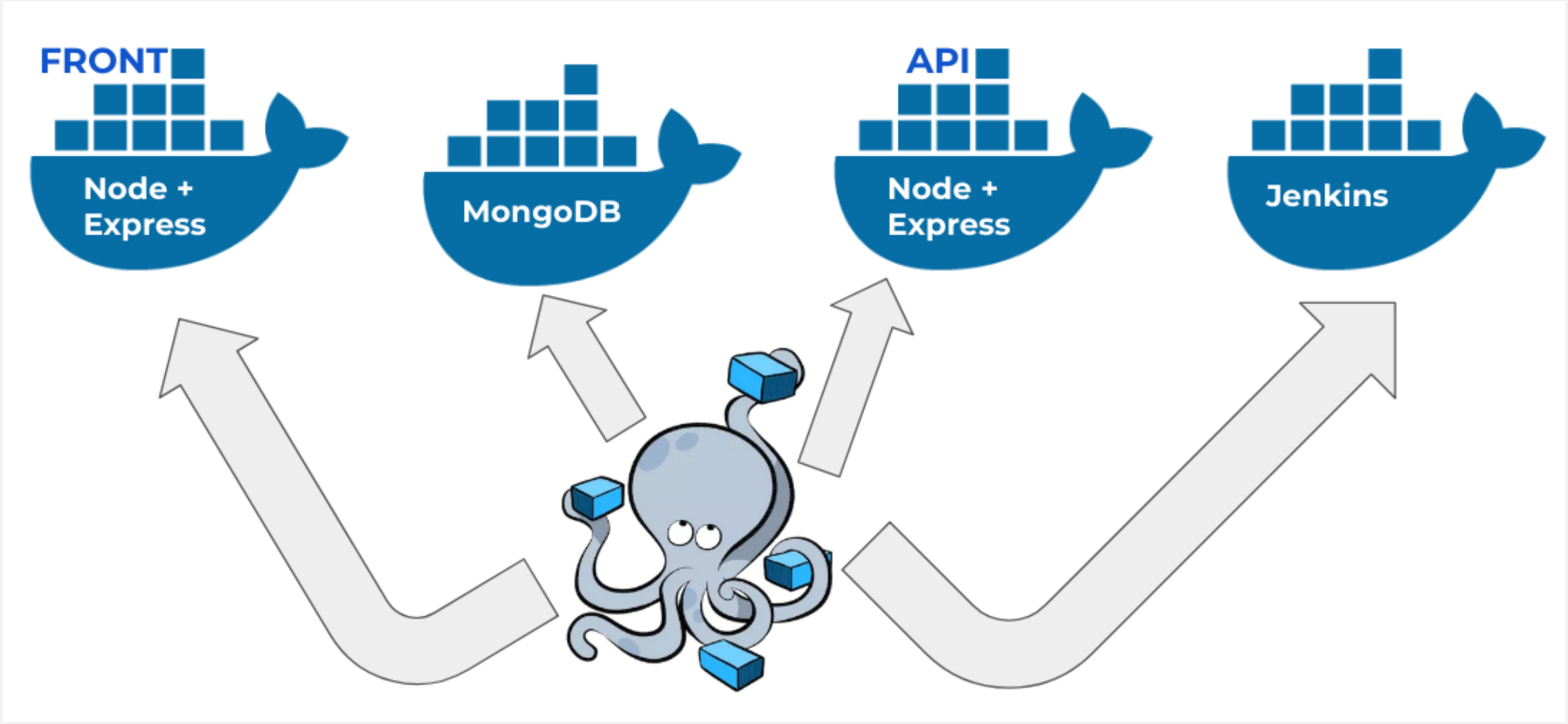
`docker compose pause`: pausar o ambiente de execução sem alterar o estado atual do container

`docker compose unpause`: para retornar ao estado antes da pausa

`docker compose stop`: encerrará a execução do container, mas não destruirá qualquer dado associado ao container

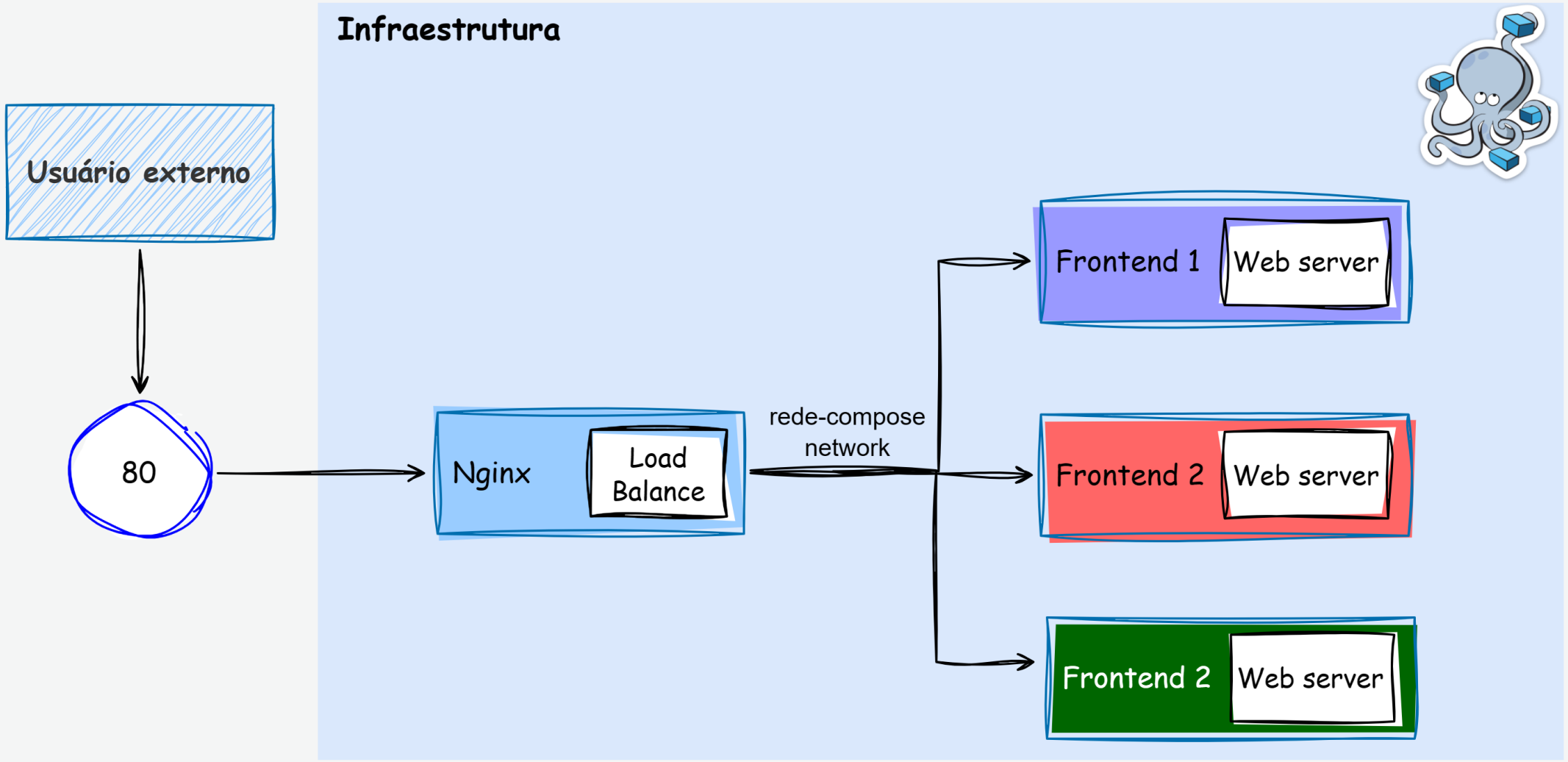
`docker compose down`: remover o container, as redes e volumes associados com o ambiente

`docker image rm <nome_da_imagem>`: para remover a imagem base do sistema


FRONT
Node +
Express
MongoDB
API
Node +
Express
Jenkins

Atividade Docker Compose

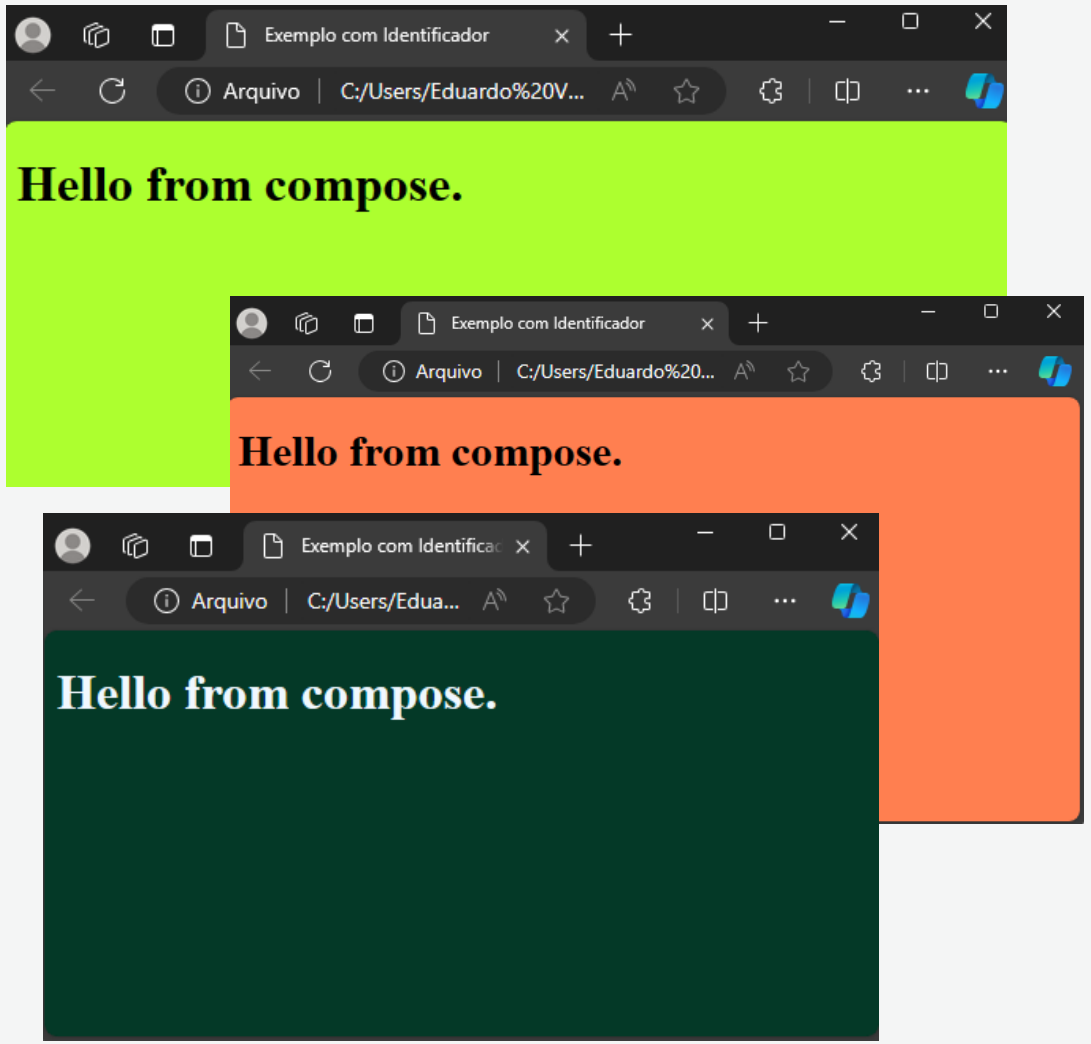
O que buscamos?



Atividade Docker Compose – Passo 1

Localmente

- Ajuste o Dockerfile criado anteriormente para construir **três** imagem simples com um servidor web e um html com background diferentes **(NGINX + index.html),** pode taggear as imagens como **USER_DOCKER/REPO:COR_n**;
- Print da imagem rodando localmente + terminal aberto com o comando whoami (bash);
- Fazer login no Docker hub e criar um repositório;
- Realizar um Docker push no Docker Hub;
- Criar o **docker-compose.yaml;**
- Anexar link da imagem na entrega;



Atividade Docker hub - Passo 2

Na VM de sub-rede pública (AWS)

- Criar um diretório no home do usuário `~/front-lb`
- Copiar o arquivo **docker-compose.yaml** para a VM via **scp** dentro do diretório criado;
- Realizar um **docker compose up** com o arquivo transferido;
- Checar as imagens Docker na máquina;
- Testar os servidores web e o load balance;
- Print do navegador com container executando na VM + terminal aberto com o comando whoami (bash) ;

Atividade Docker hub – dica básico

Estrutura local

```
└──atividade-compose
    │   docker-compose.yml
    │   Dockerfile
    │   index.html
    │
    └──load-balancer
            nginx.conf
```

Dockerfile

```dockerfile
FROM nginx:latest
COPY index.html /usr/share/nginx/html/index.html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

```
docker build -t eduverri/nginx:azul .
```

```
docker compose up
```

docker-compose.yaml

```yaml
services:
  load_balancer:
    image: nginx:latest
    ports:
      - "80:80"
    volumes:
      - ./load-balancer/nginx.conf:/etc/nginx/nginx.conf
      #Arquivo com config do NGINX + Load Balancer
    networks:
      - rede-compose
    depends_on:
      - site1
      - site2

  site1:
    image: dibrito/nginx:green #Substitua pelo nome de sua
    imagem + tag
    networks:
      - rede-compose
```

```yaml
networks:
  rede-compose:
    driver: bridge
```

Agradeço
a sua atenção!

Eduardo Verri, Diego Brito

eduardo.verri@sptech.school

diego.brito@sptech.school

SÃO PAULO TECH SCHOOL