Habilitando Aplicações Nativas de Nuvem

Introdução a Contêineres e Kubernetes

2. Aplicativos Modernos e Contêineres

Sergio Rio
www.sergiorio.tech

Refrescando nossa memória

- Na aula passada estudamos os detalhes para criação, execução, registro e reuso de aplicativos em contêineres. Analisamos os três pilares do isolamento de aplicativos (*cgroups*, *namespaces* e *unionfs*), como funciona o motor de grafos de camadas na criação de imagens de contêineres e como o *runtime* as executa.
- Atividades preparatórias para a aula de hoje:
- Ler Storage Drivers in Docker: A Deep Dive e Docker overview
  - Assistir vídeos com tutoriais de números 8 ao 13 do Docker Tutorial:

8. Criar uma imagem customizada
9. Criar uma imagem a partir de um Dockerfile
10. COPY e ADD
11. Criar um projeto realístico
12. Depurar e executar um projeto
13. Rebuilds desnecessários e o arquivo dockerignore

Dúvidas, questões ou comentários?

Programa: Introdução a Contêineres e Kubernetes

1. Conceitos Básicos

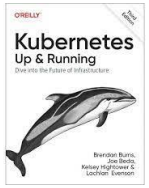
- ✓ Abstrações em Ciência da Computação
- ✓ Virtualização de Computadores
- ✓ MicroVMs e Unikernels

2. Contêineres

- Origem
- Fundamentos
- Criação e execução
- ➢ Registro e reuso
- ➢ Infraestrutura como Código
- ➢ Aplicativos Modernos

3. Kubernetes

- Origem
- Arquitetura
- Pods
- Abstrações de Recursos
- Descoberta de Serviços
- Serviços de Rede
- Instalação e administração básica
- Implantação de um caso de uso (exemplo)



Kubernetes Up & Running
Brendan Burns, Joe Beda, Kelsey Hightower, and Lachlan Evenson
Cortesia da VMware Inc.

Contêineres

Criação, execução, registro e reuso

Registro e Reuso
Criando imagens de contêineres - Dockerfile

- Método imperativo para criar imagens de contêiners:

```
FROM ubuntu: 22.04
RUN \
apt-get update && \
apt-get install -y --no-install-recommends zip unzip openjdk-8-
jdk expect && \
apt-get autoremove -qq -y --purge && \
apt-get clean && \
rm -rf /var/cache/apt /var/lib/apt/lists
COPY myExecutableFile.sh /exec/myExecutableFile.sh
RUN chmod a+rwx -R /exec/
ENTRYPOINT ["./exec/myExecutableFile.sh"]
```

Registro e Reuso
Executando imagens de contêineres

```
$ docker container run alpine echo “Hello World”
```

- Na primeira execução o Docker carrega a imagem do Dockerhub em seguida a executa.

```
[→  ~ docker container run alpine echo "hello world"
Unable to find image 'alpine:latest' locally
latest: Pulling from library/alpine
ba3557a56b15: Pull complete
Digest: sha256:a75afd8b57e7f34e4dad8d65e2c7ba2e1975c795ce1ee22fa34f8cf46f96a3be
Status: Downloaded newer image for alpine:latest
hello world
```

- A partir da segunda execução, a imagem já está disponível localmente (cache).

Registro e Reuso
Execução em modo Daemon

```
svianrio@SALVADOR: ~
(base) svianrio@SALVADOR:~$ docker container run alpine ping 127.0.0.1
PING 127.0.0.1 (127.0.0.1): 56 data bytes
64 bytes from 127.0.0.1: seq=0 ttl=64 time=0.073 ms
64 bytes from 127.0.0.1: seq=1 ttl=64 time=0.064 ms
64 bytes from 127.0.0.1: seq=2 ttl=64 time=0.061 ms
^C
--- 127.0.0.1 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.061/0.066/0.073 ms
```

```
svianrio@SALVADOR: ~
(base) svianrio@SALVADOR:~$ docker container run -d alpine ping 127.0.0.1
4fd2a3a2bb9e54474d27b10130a963b48d1352ff8b8fe6bd4ebd4a8ecc34efc6
```

```
svianrio@SALVADOR: ~
(base) svianrio@SALVADOR:~$ docker container ls
CONTAINER ID   IMAGE     COMMAND            CREATED              STATUS              PORTS     NAMES
4fd2a3a2bb9e   alpine    "ping 127.0.0.1"   About a minute ago   Up About a minute             musing_sinoussi
```

```
(base) svianrio@SALVADOR:~$ docker container exec -it 4fd2a3a2bb9e /bin/sh
/ # ps
PID   USER     TIME   COMMAND
    1 root       0:00 ping 127.0.0.1
    7 root       0:00 /bin/sh
   13 root       0:00 ps
/ #
```

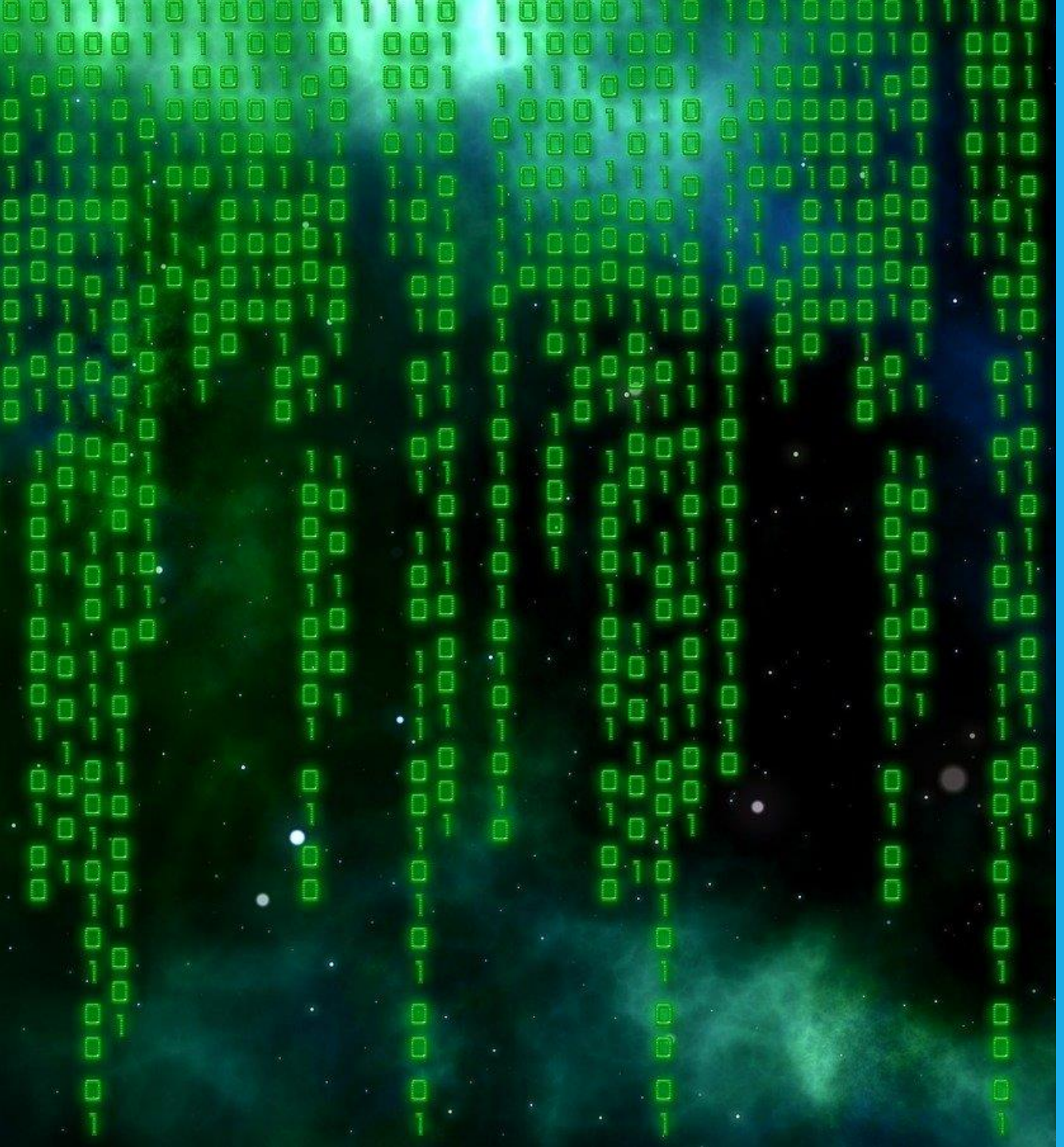
Registro e Reuso
Dockerfile multi-estágios

- *Dockerfiles* multi-estágios permite a não inclusa de camadas intermediarias desnecessárias:

```
• FROM alpine:latest AS build
RUN apk update && apk add --update alpine-sdk
RUN mkdir /app
WORKDIR /app
COPY . /app
RUN mkdir bin
RUN gcc test.c -o bin/test
FROM alpine:latest
COPY --from=build /app/bin/test /app/test
CMD /app/test
```

- A imagem resultante terá 4MB os invés de mais de 200MB!

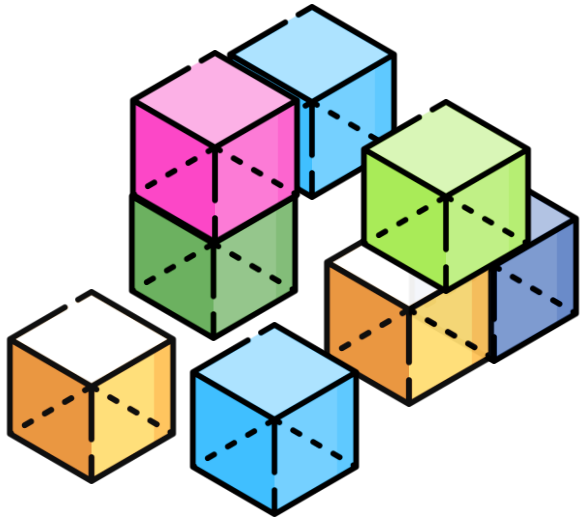
Dúvidas, questões ou comentários?



“A Matrix está em toda parte. Está em tudo a nossa volta. Agora mesmo nessa sala.”

Morpheus (Lawrence Fishburn), “The Matrix”, 1999.

Aplicativos Modernos
Arquitetura Genérica de Aplicativos Distribuídos



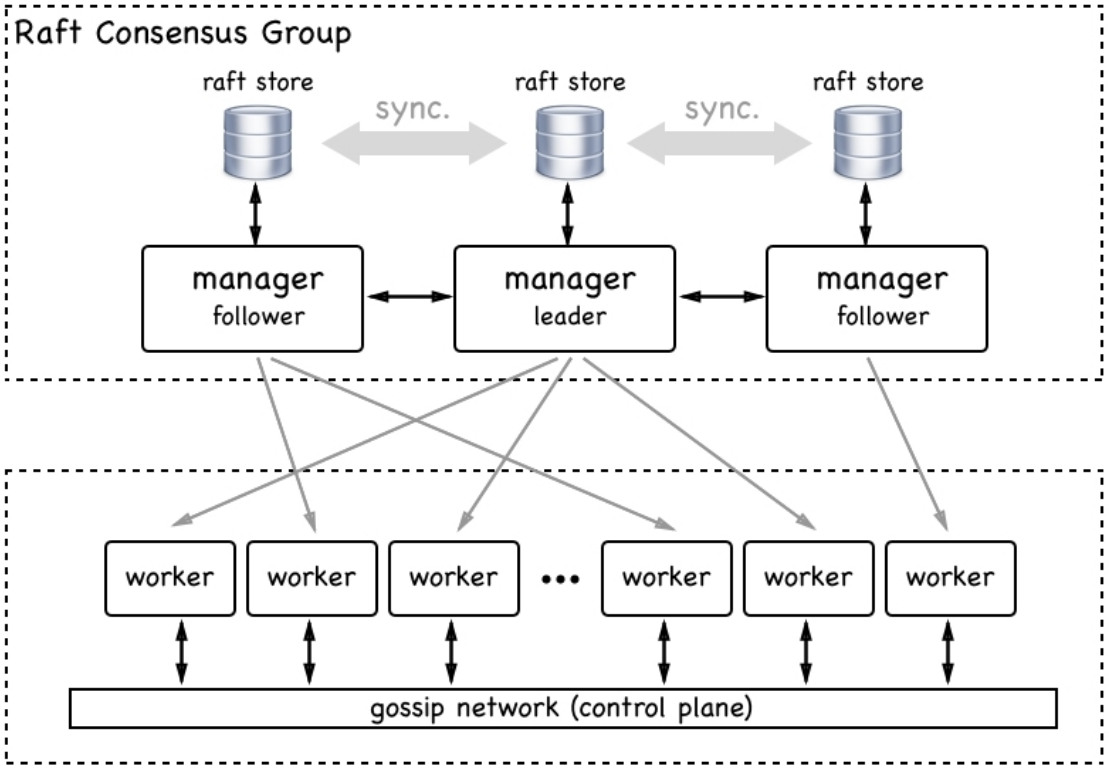
- Em geral, sistemas distribuídos consistem de vários componentes que precisam ser executados em mais de um computador.
- Usando contêineres é possível construir e implementar cada componente em seu próprio contêiner.
- Existem milhares de componentes pré-construídos disponíveis!

Orquestração: execução coordenada de todos os componentes.

Necessária para obter comportamento esperado de um sistema distribuído como um todo.

Aplicativos Modernos
Docker Swarm[1]

- Permite provisionamento automático de clusters de aplicativos em contêineres.
- Primeira versão liberada em 2014.
- Docker Swarm Mode em 2016.
- Testes realizados com clusters com aproximadamente 2.300 nós rodando 96.000 contêineres.

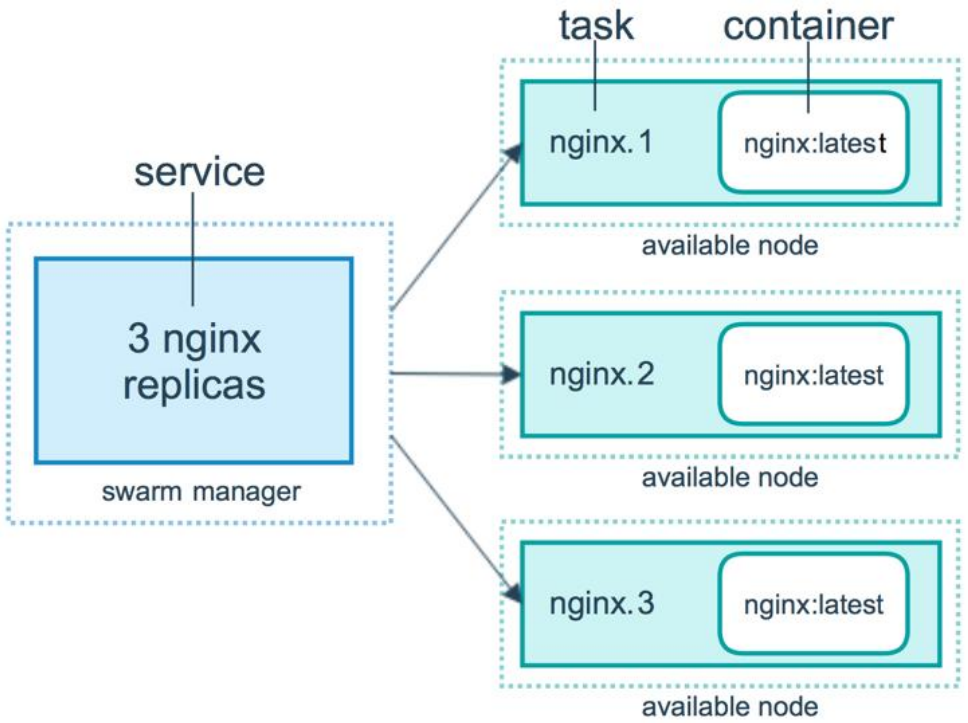


1 - Swarm mode overview

Raft é um algoritmo de consenso projetado como uma alternativa à família de algoritmos Paxos. Ele foi concebido para ser mais compreensível que o Paxos por meio da separação da lógica, mas também é formalmente comprovado como seguro e oferece alguns recursos adicionais. Raft oferece uma maneira genérica de distribuir uma máquina de estado em um cluster de sistemas de computação, garantindo que cada nó do cluster concorde com a mesma série de transições de estado.

Aplicativos Modernos
Serviços no Docker Swarm

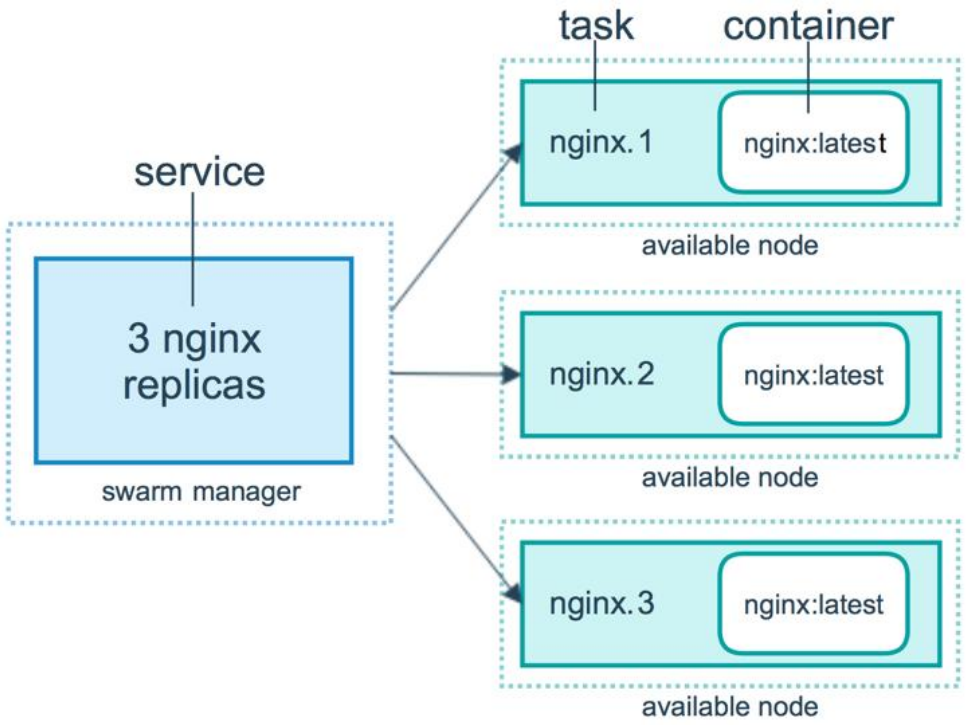
- Serviços Docker usam um modelo declarativo que define o estado desejado do serviço.
- O Docker Swarm se encarrega da manutenção do estado desejado.
- Definição de Estado
  - Nome da imagem e etiqueta (*tag*).
  - Quantidade de contêineres (*tasks*) no serviço.
  - Portas IP expostas para clientes fora do *swarm*.



Aplicativos Modernos
Serviços no Docker Swarm

```
$ docker service create --name
my_web --replicas 3 --publish
published=8080,target=80 nginx
```

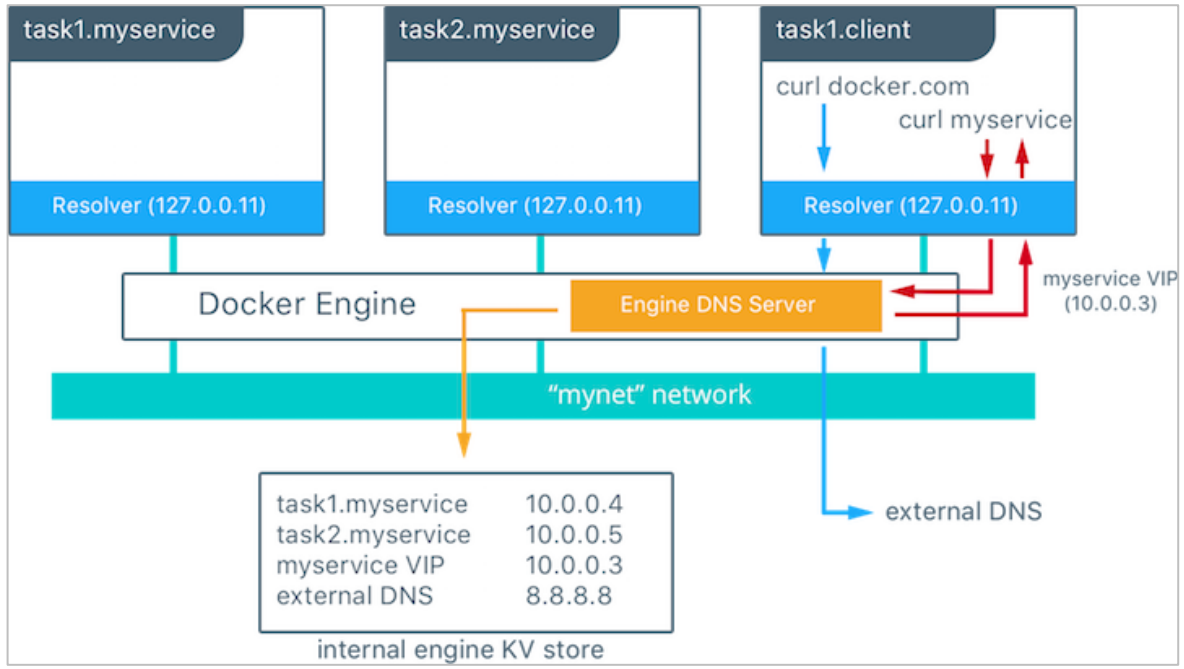
- Três tarefas (tasks) executadas em até três nos.
- Não é necessário saber quais nós executam quais tarefas.
- *Routing Mesh*: conexão na porta 8080 de qualquer um dos nós é roteada para uma das três tarefas de nginx.



Aplicativos Modernos
Serviços de Rede no Docker Swarm

- São configuráveis usando *drivers* (*bridge, host, overlay, macvlan, none*):

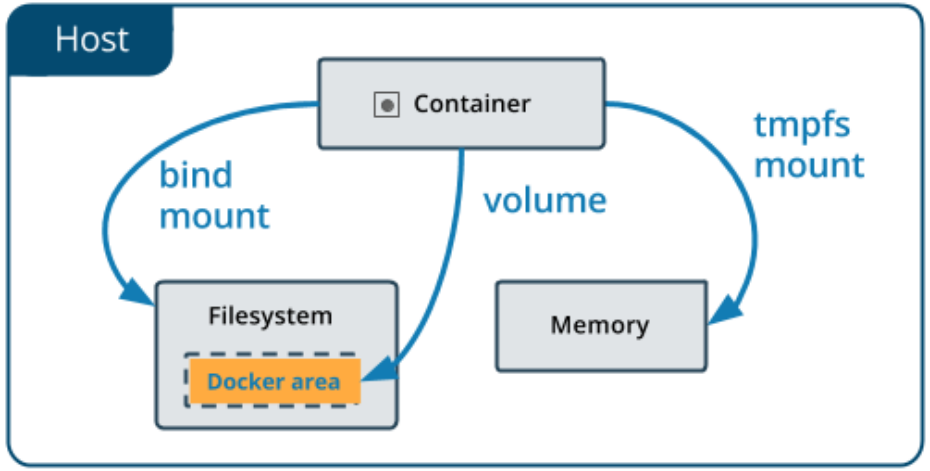
```
$ docker network create my-net
$ docker create --name my-nginx \
--network my-net \
--publish 8080:80 \
nginx:latest
```



- Balanceamento de carga é habilitado automaticamente quando um serviço é criado.
- Em geral o descobrimento de serviços é habilitado pelo serviço de nomes (DNS).

Aplicativos Modernos
Volumes de Dados no Docker Swarm

- Como os contêineres usam *unionfs*, o sistema de arquivos em tempo de execução é **efêmero:** quando a execução do contêiner termina o ou é abortado o conteúdo de seu sistema de arquivos é apagado.
- Docker Swarm permite três tipos de sistemas de arquivos persistentes:
  - Bind mount (diretório ou arquivo do sistema *host*)
  - Volume (volumes locais ou *NFS* por exemplo)
  - *tmpfs* (dados não persistem no *host* e tampouco no contêiner, usando comumente confidencialidade de dados ou quando o aplicativo gera e escreva volume muito grande de informações não persistentes)




```
$docker volume create my-vol
$docker volume ls
$docker volume inspect my-vol
$docker volume rm my-vol
$ docker run -d \
--name devtest \
--mount source=my-vol,target=/app \
nginx:latest
```

Aplicativos Modernos
Secrets no Docker Swarm

- Docker administra de maneira centralizada dados confidenciais como senhas, chaves privadas SSH e certificados SSL que são chamados de *Secrets*.
- Não podem ser transmitidos pela rede, armazenados em arquivos *Dockerfile* ou no código do aplicativo.
- Quando autorizados os Secrets são montados de maneira não criptografada na memória do contêiner em execução:
  - /run/secrets/<secret_name> in Linux containers
  - C:\ProgramData\Docker\secrets in Windows containers
- Secrets estão disponíveis somente para *Serviços Swarm*. Não existem para contêineres isolados.

```
$docker secret create
$docker secret inspect
$docker secret ls
$docker secret rm

--secret flag for docker service
create
--secret-add and --secret-rm
flags for docker service update
```

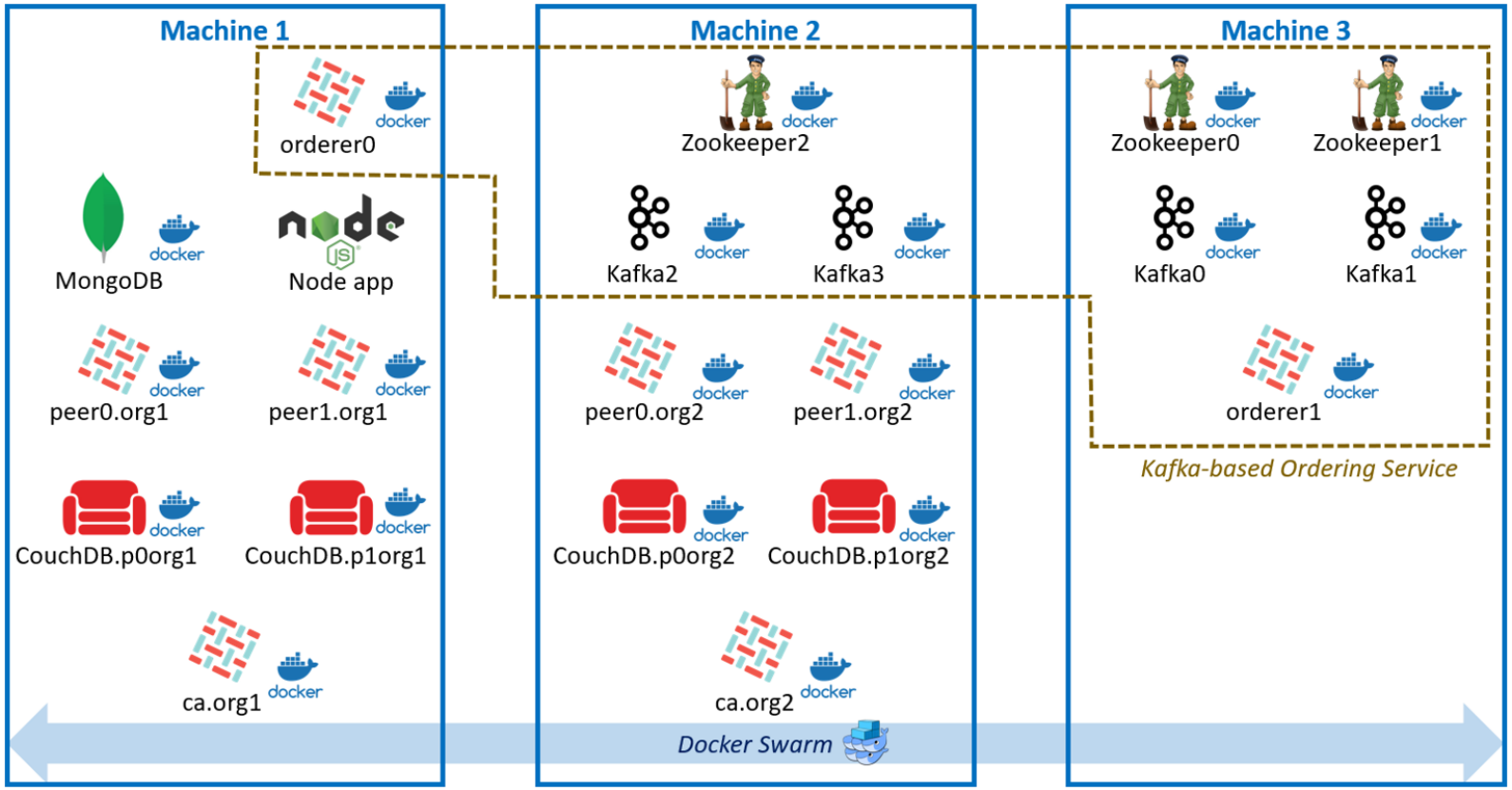
Aplicativos Modernos
Configs no Docker Swarm

- Operam de maneira similar aos Secrets mas não são encriptados (*at rest*) e são montados diretamente no sistema de arquivos do contêiner sem o uso de RAM disk (ou *tmpfs*).
- *Configs* estão disponíveis somente para *Serviços Swarm*. Não existem para contêineres isolados.

```
$docker config create
$docker config inspect
$docker config ls
$docker config rm
```

Aplicativos Modernos
Um exemplo de *Hyperledger Fabric*[2]

Hyperledger Fabric é uma estrutura modular de blockchain que atua como base para o desenvolvimento de produtos, soluções e aplicativos baseados em blockchain usando componentes *plug-and-play* destinados ao uso em empresas privadas.



Fonte: Deploy a Kafka-based Hyperledger Fabric network with Docker Swarm on multiple hosts

2 – Hyperledger Foundation

“Door on your left…”

Tank, The Operator (Marcus Chong), “The Matrix”, 1999.

Infrastructure as a Code (IaaC) [3] Objetivos e abordagens

- *IaaC* é o provisionamento, gestão e a disposição de infraestrutura (servidores, maquinas virtuais, redes, armazenamento de dados, contêineres, clusters, etc.) através de programas e arquivos de configuração com os seguintes objetivos:
  1. Redução de custos: eliminação de atividades manuais repetitivas através de automação.
  2. Velocidade: acelera a entrega de valor disponibilizando a infraestrutura requerida para aplicativos de negócios.
  3. Risco: mitiga riscos de atrasos e custos adicionais diminuindo a probabilidade de erros humanos.
- Duas abordagens atualmente:
  - Imperativa (***Como?***) define a sequência de comandos que devem ser executados para levar a cabo o provisionamento e a configuração desejados.
  - Declarativa (***Que?***) define o estado desejado e o sistema executa o que for necessário para alcançá-lo e mantê-lo de maneira estável.
- Exemplos: *Terraform (OpenTF), Puppet, SaltStack, Docker,* Doc*ker Swarm* e *Kubernetes*.

3 - https://en.wikipedia.org/wiki/Infrastructure_as_code

Infrastructure as a Code (IaaC)
Objetivos e abordagens

Imperativo

```
$ docker network create –d overlay my-network
$ docker container create --name web-service --publish 80:80 --replicas 3 --network my-network nginx:latest
$ docker container create --name db-service --replicas 1 --network my-network --volume /usr/bin:/usr/local/bin
```

Declarativo

```
version: “3.9”
        service:
                web-service:
                image: nginx:latest
                 ports: “80:80”
                 deploy:
                 replicas: 3
                 networks:
                 front-end
...
networks:
        front-end:
```

Infrastructure as a Code (IaaC) Exemplo em Docker Swarm

```
version: "3.9”
services:
        frontend:
                image: web-app
                deploy:
                        replicas: 3
                ports:
                        - "443:8043"
                networks:
                        - front-tier
                        - application-tier
                configs:
                        - httpd-config
                secrets:
                        - server-certificate
        backend:
                image: example-registry.com:4000/mysql:8.0
                restart: on_failure
                environment:
                        #MYSQL_ROOT_PASSWORD: example
                        #.env
                        MYSQL_ROOT_PASSWORD_FILE=/run/secrets/mysql-root
        volumes:
                - db-data:/etc/data
        networks:
                - back-tier
        secrets:
                - mysql-password
```

```
application:
        build:
                context: ./dir #path to the build context
                dockerfile: Dockerfile-alternate
                args:
                        - buildno: 1
        image: application-logic
        deploy:
                mode: replicated
                replicas: 4
        networks:
                - application-tier
                - back-tier
```

Dúvidas, questões ou comentários?

Atividades para a próxima aula

- Ler Infrastructure As Apps: The GitOps Future of Infra-as-code e “The Legacy Trap”.
- Assistir vídeo Tanzu Talk: The right mindset for starting application modernization.