DOCKER

Docker é uma plataforma open source na linguagem de programação Go, que possui alto desempenho e é desenvolvida diretamente no Google.

A tecnologia Docker oferece mais do que a habilidade de executar containers: ela também facilita o processo de criação e construção de containers, o envio e o controle de versão de imagens, dentre outras coisas.

Assim, é desnecessário ajustar o ambiente para que o serviço funcione corretamente. Ou seja, ele é sempre igual e é configurado uma vez. A partir disso, basta replicá-lo.

Outras possibilidades interessantes são criar os containers prontos para deploy (ou seja, imagens) a partir de dockerfiles, que são arquivos de definição.
Uma imagem é um pacote leve e executável que engloba tudo que é preciso para executar um pedaço da aplicação, incluindo código, bibliotecas, variáveis de ambiente, arquivos de configuração e um ambiente de execução.

Um container é uma instancia desta imagem, isto é, dada a configuração da imagem, é o que todo o conjunto anterior se torna em memória quando é de fato executado. Um container roda de modo isolado do host por padrão, apenas acessando arquivos do host caso haja configuração para tal. (O container é um ambiente isolado.)

As imagens podem existir sem contêineres, enquanto um contêiner precisa executar uma imagem para existir. Portanto, os contêineres são dependentes de imagens e os usam para construir um ambiente de tempo de execução e executar um aplicativo.

COMANDOS DOCKER

docker run

Executar um container a partir de uma imagem docker

docker pull

Baixar imagem de um registry

docker exec

Enviar comandos para containers docker

docker push

Manda uma imagem para o registry

docker tag

Nomeia uma imagem docker

docker build

Builda uma imagem a partir do Dockerfile

docker images (docker image ls)

Mostra as imagens no host

docker ps (docker container ps)

Mostra containers rodando no host

docker inspect

Mostra informações sobre o objeto

docker login

Loga em um registry

docker logs(docker container logs)

Mostra os logs de um container

Dockerfile

O Dockerfile nada mais é do que o arquivo de configuração onde a imagem docker será referenciada para sua criação, executando todas as etapas no Dockerfile.

Para criar uma imagem com base no Dockerfile:

docker build

TAGS DOCKERFILE

FROM imagem base para criação (caso a imagem seja privada o docker pegará dentro do registry privado, caso seja publica, o repositório do dockerhub será utilizado.

RUN

Serve para executar comandos no processo de montagem da imagem que estamos construindo no Dockerfile

COPY/ADD

Comandos para copiar arquivos e pastas de um lugar específico na máquina local para uma pasta no container.

WORKDIR

Define uma pasta dentro do container onde os comandos serão executados.

DOCKER-COMPOSE

Docker compose é uma ferramenta para definição e execução de múltiplos containers Docker. Com ela é possível configurar todos os parâmetros necessários para executar cada container a partir de um **arquivo de definição**. Dentro desse arquivo, definimos cada container como **serviço**, ou seja, sempre que esse texto citar **serviço** de agora em diante, imagine que é a definição que será usada para iniciar um **container**, tal como portas expostas, variáveis de ambiente e afins.

Com o Docker Compose podemos também especificar quais **volumes** e **rede** serão criados para serem utilizados nos parâmetros dos **serviços**, ou seja, isso quer dizer que não preciso criá-los manualmente para que os **serviços** utilizem recursos adicionais de **rede** e **volume**.

Docker-compose.yml

```
version: "3.7"

services:
  app:
    # The app service definition
  mysql:
    image: mysql:5.7
    volumes:
      - todo-mysql-data:/var/lib/mysql
    environment:
      MYSQL_ROOT_PASSWORD: <your-password>
      MYSQL_DATABASE: todos

volumes:
  todo-mysql-data:
```

Representação da configuração do yml em forma de comando docker

```
docker run -d
  --network todo-app --network-alias mysql
  -v todo-mysql-data:/var/lib/mysql
  -e MYSQL_ROOT_PASSWORD=<your-password>
  -e MYSQL_DATABASE=todos
  mysql:5.7
```

docker-compose up

Cria e inicia os contêineres;

docker-compose build

Realiza apenas a etapa de build das imagens que serão utilizadas;

docker-compose logs

Visualiza os logs dos contêineres;

docker-compose restart

Reinicia os contêineres;

docker-compose ps

Lista os contêineres;

docker-compose scale

Permite aumentar o número de réplicas de um contêiner;

docker-compose start

Inicia os contêineres;

docker-compose stop

Paralisa os contêineres;

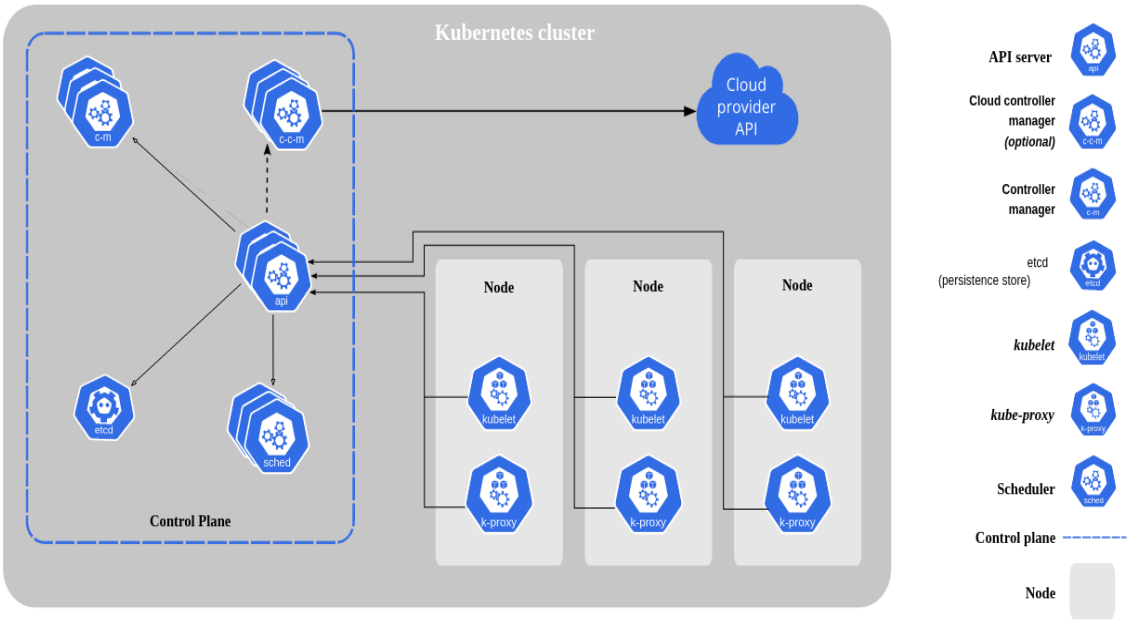
docker-compose down

Paralisa e remove todos os contêineres e seus componentes como rede, imagem e volume.

Kubernetes

Kubernetes é uma plataforma de código aberto que automatiza a implantação, dimensiona e gerencia aplicativos em contêineres.

Componentes kubernetes



CONTROL PLANE COMPONENTS

Os componentes do plano de controle tomam decisões globais sobre o cluster (por exemplo, agendamento), bem como detectam e respondem a eventos de cluster (por exemplo, iniciando um novo pod quando o campo de réplicas de uma implantação está insatisfeito).

KUBE-APISERVER

O servidor API é um componente do plano de controle Kubernetes que expõe a API kubernetes. O servidor API é a parte frontal do plano de controle Kubernetes.

A principal implementação de um servidor API Kubernetes é kube-apiserver. kube-apiserver é projetado para escalar horizontalmente — ou seja, ele escala implantando mais instâncias. Você pode executar várias instâncias de kube-apiserver e equilibrar o tráfego entre essas instâncias.

ETCD

Loja de valor de chave consistente e altamente disponível usada como loja de backup da Kubernetes para todos os dados de cluster.

Se o cluster Kubernetes usar etcd como sua loja de backup, certifique-se de ter um plano de backup para esses dados.

Você pode encontrar informações detalhadas sobre etcd na documentação oficial.

KUBE-SCHEDULER

Controle o componente do plano que observa pods recém-criados sem nó atribuído e seleciona um nó para que eles sejam executados.

Os fatores levados em conta para as decisões de agendamento incluem: requisitos de recursos individuais e coletivos, restrições de hardware/software/política, especificações de afinidade e anti-afinidade, localidade de dados, interferência entre carga de trabalho e prazos.

KUBE-CONTROLLER-MANAGER

Controle o componente do plano que executa os processos do controlador.

Logicamente, cada controlador é um processo separado, mas para reduzir a complexidade, todos eles são compilados em um único binário e executados em um único processo.

Alguns tipos desses controladores são:

Controlador de nó: Responsável por perceber e responder quando os nódulos caírem.

Controlador de trabalho: Relógios para objetos de trabalho que representam tarefas pontuais e criam Pods para executar essas tarefas até a conclusão.

Controlador de pontos finais: preenche o objeto Endpoints (ou seja, junta-se a Services & Pods).

Controladores de conta de serviço & token: Crie contas padrão e tokens de acesso a API para novos namespaces.

NODE COMPONENTES

Os componentes do nó são executados em cada nó, mantendo as cápsulas em execução e fornecendo o ambiente de tempo de execução de Kubernetes.

KUBELET

Um agente que corre em cada nó no aglomerado. Ele garante que os contêineres estão funcionando em um Pod.

O kubelet pega um conjunto de PodSpecs que são fornecidos através de vários mecanismos e garante que os recipientes descritos nesses PodSpecs estejam funcionando e saudáveis. O kubelet não gerencia contêineres que não foram criados por Kubernetes.

KUBE-PROXY

kube-proxy é um proxy de rede que é executado em cada nó em seu cluster, implementando parte do conceito de Serviço Kubernetes.

kube-proxy mantém regras de rede sobre nódulos. Essas regras de rede permitem a comunicação de rede com seus Pods a partir de sessões de rede dentro ou fora do seu cluster.

kube-proxy usa a camada de filtragem do pacote operacional se houver um e estiver disponível. Caso contrário, kube-proxy encaminha o tráfego em si.

OBJETOS KUBERNETES

WORKLOADS

Uma carga de trabalho é um aplicativo em execução no Kubernetes. Se sua carga de trabalho é um único componente ou vários que trabalham juntos, em Kubernetes você executá-lo dentro de um conjunto de pods. Em Kubernetes, um Pod representa um conjunto de recipientes em execução em seu cluster.

POD

Vamos começar falando da menor unidade no cluster k8s e ir subindo a hierarquia, desta forma vai ficar mais fácil de entender o "por que" e como os controladores funcionam.

O Pod é onde de fato sua aplicação será executada, e sim você consegue criar/rodar sua aplicação criando diretamente um pod com tudo o que você precisa

se o pod "morrer" sua aplicação para! não tem escalabilidade e nenhuma inteligência para sua aplicação.

REPLICASET

Como vimos é possível executar a aplicação através do objeto pod, porém isso está longe de ser o ideal. Precisamos de um objeto que cuide dos pods, que mantenha os estados e que aumente a quantidade de pods quando necessário e para isso temos o ReplicaSet:

Se deletar manualmente o pod, ele será recriado automaticamente. Se a aplicação der algum problema, e o pod "morrer" o ReplicaSet vai lá e cria outro!

Além de garantir esse estado, o ReplicaSet é o responsável por escalar sua aplicação.

Se houver alteração na configuração do POD o mesmo só subirá será autalizado caso seja criado um pod novo(ele não sobe automáticamente).

DEPLOYMENT

Para resolver esse gerenciamento e controle de versão, temos o Deployment. Com ele conseguimos realizar deploy de novas versões mais facilmente e até realizar rollback.

Só uma nota, quando temos um artefato do tipo Deployment automaticamente é criado o ReplicaSet e os pods de acordo com número de réplicas, se especificado.

Service é um balanceador de carga, que realiza uma requisição para a kube-apiserver , recebendo uma lista de endpoints das aplicações, direcionando o tráfego para os pods.

Ingress é um objeto Kubernetes que nos permite externalizar services de um cluster, ou seja é uma forma de abstrair services internos do cluster para o público externo (cliente), com Ingress podemos trabalhar com protocolos HTTP/HTTPS, SSL/TLS, roteamento de tráfego, balanceamento de carga, controle de acesso, websocket e etc. Tudo isso utilizando apenas uma única integração com o serviço de loadbalancer.

STATEFULSET

Permite executar um ou mais pods relacionados que rastreiam o estado de alguma forma. Por exemplo, se sua carga de trabalho registra dados de forma persistente, você pode executar um statefulset que corresponda a cada pod um com um arquivo PersistentVolume. Seu código, executado no pod, pode replicar dados para outros pods no mesmo statefulset para melhorar a resiliência geral.

DAEMONSET

Define pods que fornecem recursos no local do nó. Eles podem ser fundamentais para a operação de seu cluster, como uma ferramenta auxiliar de rede.Toda vez que você adiciona um nó ao cluster que corresponde à especificação em a DaemonSet, o plano de controle agenda a Pod para isso DaemonSet no novo nó.

JOB/CRONJOB

Definem tarefas que são executadas até a conclusão e, em seguida, param. Os Jobs representam tarefas pontuais, enquanto CronJobs se repetem de acordo com um cronograma.

SERVICE

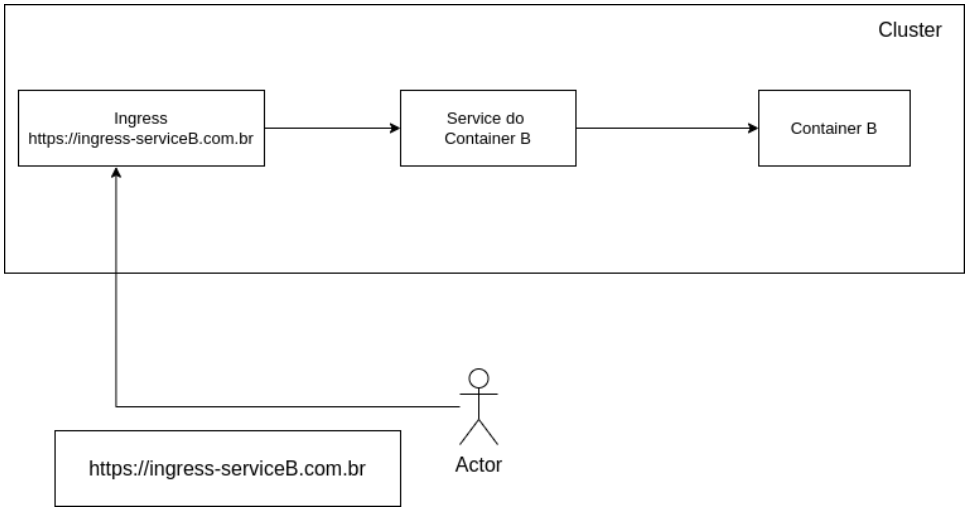
Um Service agrupa um conjunto de endpoints de pod em um único recurso. É possível configurar várias maneiras de acessar o agrupamento. Por padrão, você recebe um endereço IP de cluster estável que os clientes dentro do cluster podem usar para contatar pods no serviço. Um cliente envia uma solicitação ao endereço IP estável e a solicitação é encaminhada a um dos pods no serviço.

Um serviço identifica seus pods membro com um seletor. Para que um pod seja membro do serviço, ele precisa ter todos os rótulos especificados no seletor. Um rótulo é um par de chave-valor arbitrário anexado a um objeto.

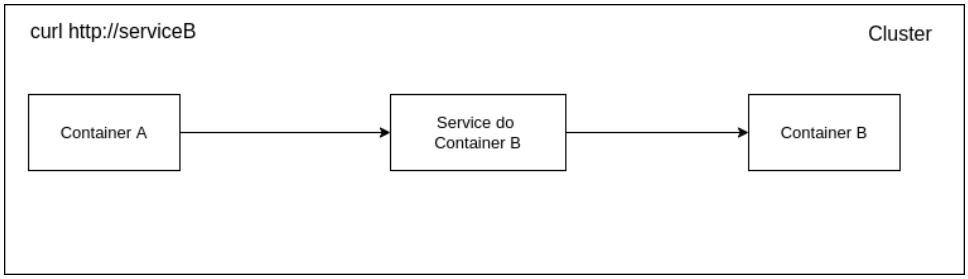
INGRESS

Kubernetes Ingress é um objeto de API que fornece regras de roteamento para gerenciar o acesso de usuários externos aos serviços em um cluster Kubernetes, normalmente via HTTPS/HTTP. Com o Ingress, você pode configurar facilmente regras para rotear o tráfego sem criar vários balanceadores de carga ou expor cada serviço no nó. Isso o torna a melhor opção para uso em ambientes de produção.

Comunicação fora do cluster



Comunicação dentro do cluster



NGINX

O NGINX é um servidor web. Ele é um serviço, é um programa que roda e que serve para responder requisições web. Então, é muito importante que você entenda como a web funciona. Por isso, o pré-requisito deste treinamento é o curso de HTTP. Você precisa entender o protocolo HTTP para saber qual é o papel do NGINX.

Então super simplificando milhares de conceitos que você já deve ter aprendido no curso de HTTP, a web funciona com uma arquitetura cliente-servidor. Temos aqui o cliente e o servidor, então imagine que o cliente é um navegador web; e nós temos um servidor rodando, uma aplicação em Java, em PHP, C Sharp, Python, Ruby ou a linguagem que for.

Então, o cliente vai digitar a URL do nosso site: “alura.com.br”, por exemplo, e ele vai fazer uma requisição para o computador que tem o nome “alura.com.br”.

Esse computador precisa saber receber essa requisição web e fazer o que tem que fazer - ou exibir o HTML, a imagem, devolver o arquivo CSS - ou então mandar para uma aplicação processar essa requisição. Uma aplicação em Java, em PHP, em Python ou no que for.

Essa tarefa de receber a requisição e decidir o que vai fazer, se vai mandar para uma aplicação, se vai devolver algum arquivo direto - essa é a tarefa de um servidor web.

Então é aí que entra o NGINX! Esse é o papel dele. O que o NGINX vai fazer dentro de um servidor é ficar ouvindo uma porta. Quando falamos de HTTP, a porta padrão é a “80”. Então se você não digita uma porta e está utilizando HTTP, essa requisição vai cair na porta 80.

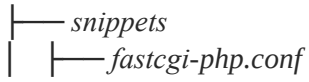
Então um servidor web fica lá ouvindo, por exemplo, a porta 80 e ele fica esperando alguma requisição ou algum pedido entrar pedido chegar nesse computador. Quando chega, esse servidor web tem que saber o que fazer.

Então, ele vai analisar. Se o que você está pedindo parece ser um arquivo de imagem, então não preciso fazer nada com mais ninguém. Eu só vou te devolver essa imagem e o navegador vai exibir para você.

Ao acessar o local de instalação (/etc/nginx), veremos uma estrutura parecida com esta:

tree /etc/nginx

```
/etc/nginx
├── conf.d
├── fastcgi.conf
├── fastcgi_params
├── mime.types
├── modules-available
├── modules-enabled
├── nginx.conf
├── proxy_params
├── sites-available
│   ├── default
├── sites-enabled
│   ├── default -> /etc/nginx/sites-available/default
```



conf.d/

Pasta com as configurações extras do Nginx. Nela, é possível criar um arquivo de configuração que será incluído automaticamente nas configurações gerais.

fastcgi.conf, fastcgi_params

Configurações do fastcgi. Com eles, é possível incluir, excluir ou remover parâmetros usados pela interface entre o servidor da web e os aplicativos.

mime.types

Funciona como um map para identificar o mimetype dos arquivos conforme a extensão dele.

modules-available/

Configuração dos módulos disponíveis. (Veremos sobre os módulos mais adiante)

nginx.conf

Configuração geral do Nginx. Nele, contém configuração básica de formatação de log, SSL, upload, gzip, pid (Process Identifier, no Unix), número de conexões simultâneas por processo.

proxy_params

Configurações usada com o recurso de proxy reverso. (Veremos um pouco ao configurar o ExpressJs com o NodeJs).

sites-available/

Nesta pasta, ficam as configurações dos servidores virtuais.

sites-enabled/

Nesta pasta, ficam os servidores virtuais ativos. Caso o arquivo de configuração esteja em sites-available, mas não esteja em sites-enabled, o Nginx não irá carregá-lo.

snippets/

Configurações extras.

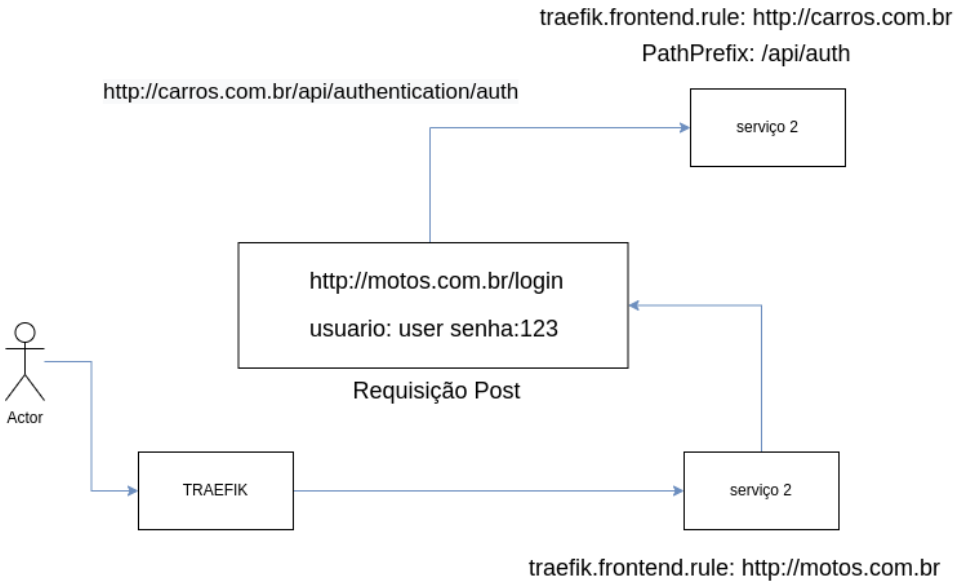
Traefik

O Traefik é um proxy reverso HTTP moderno e um balanceador de carga para microsserviços. O Traefik facilita a implantação de todos os microsserviços, integrado a componentes de infraestrutura existentes, como o Docker, o Swarm Mode, o Kubernetes, o Amazon ECS, o Rancher, o Etcd, o Consul etc.

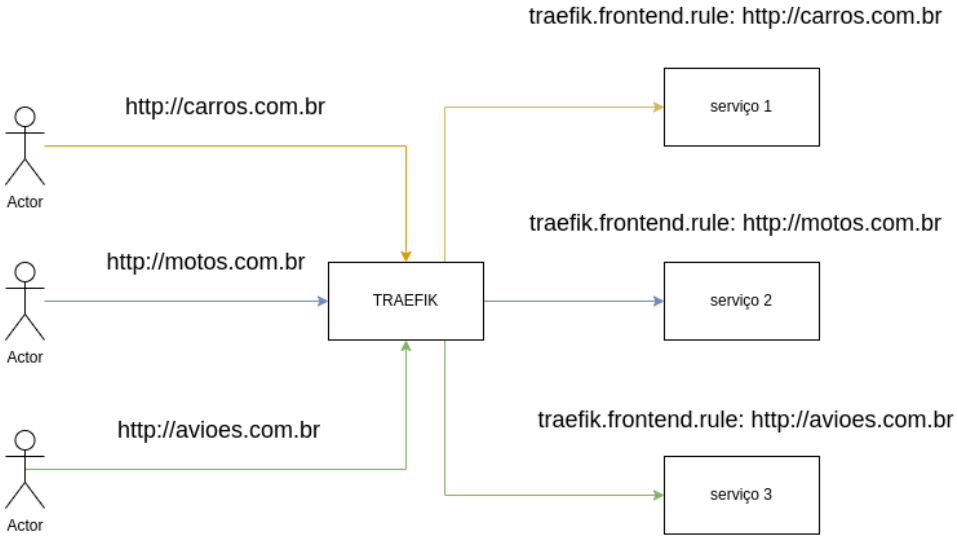
O Traefik funciona como um roteador para todos os seus aplicativos de microsserviços, roteando todas as solicitações do cliente para corrigir o destino dos microsserviços.

No docker,rancher(1.6) e docker-compose, o traefik funciona a partir de labels que configuram todas as requisições para o container.

Exemplo 1



Exemplo 2



Labels:

traefik.enabled : habilita o traefik.

traefik.frontend.rule=Host:blog.your_domain Traefik para examinar o host solicitado e se ele corresponder ao padrão de blog.your_domain ele deve encaminhar o tráfego para o contêiner do blog.

Path: /products/, /articles/{category}/{id:[0-9]+}, combine o caminho exato da solicitação. Aceita uma sequência de caminhos de expressão literal e regular.

PathStrip: /products/, Combine o caminho exato e retire o caminho antes de encaminhar a solicitação para o back-end. Aceita uma sequência de caminhos literais.

PathStripRegex: /articles/{category}/{id:[0-9]+}, combine o caminho exato e retire o caminho antes de encaminhar a solicitação para o back-end. Aceita uma sequência de caminhos de expressão literal e regular.

PathPrefix: /products/, /articles/{category}/{id:[0-9]+}, Caminho de prefixo de solicitação de correspondência. Ele aceita uma sequência de caminhos prefixos de expressão literal e regular.

PathPrefixStrip: /products/, Combine o prefixo de solicitação e retire o prefixo do caminho antes de encaminhar a solicitação para o backend. Ele aceita uma sequência de caminhos prefixos literais. Começando com Traefik 1.3, o caminho do prefixo despojado estará disponível no cabeçalho X-Forwarded-Prefix.

PathPrefixStripRegex: /articles/{category}/{id:[0-9]+}, combine o prefixo de solicitação e retire o prefixo do caminho antes de encaminhar a solicitação para o backend. Ele aceita uma sequência de caminhos prefixos de expressão literal e regular. Começando com Traefik 1.3, o caminho do prefixo despojado estará disponível no cabeçalho X-Forwarded-Prefix.

traefik.port especifica a porta exposta que Traefik deve usar para direcionar o tráfego para este contêiner.

Como funciona o proxy reverso?

Um proxy reverso é um servidor que fica na frente de um ou mais servidores da web, interceptando solicitações de clientes. Com um proxy reverso, quando os clientes enviam solicitações ao servidor de origem de um site, essas solicitações são interceptadas na extremidade da rede pelo servidor proxy reverso.

Gitlab

Trata-se de um arquivo no formato YAML onde são definidos estágios e tarefas para execução das etapas necessárias da integração e implantação contínua.

A primeira parte a se pensar são os estágios (stages) das tarefas (jobs). Os estágios são grupos de tarefas e são executados sequencialmente. São grupos de ações que precisam ser executadas para executar as verificações de código e por fim, a implantação.

```
stages:
  - check
  - Build
  - deploy
```

Definidos os estágios, a próxima etapa é definir as tarefas para cada um. Ainda seguindo o mesmo exemplo, no estágio de checagem de código podemos executar o ESLint e os testes. Cada tarefa possui seu próprio bloco no arquivo YAML, também é necessário definir quais comandos serão executados (bloco script). Por exemplo:

```
ESLint:
  stage: check
  script:
    - npm run eslint

Test:
  stage: check
  script:
    - npm test
```

Gradle

o Gradle trata-se de uma ferramenta de build open source bastante poderosa que nos permite configurar arquivos de build por meio da sua DSL (Domain Specific Language) baseada na linguagem Groovy.

Gradlew

A maneira recomendada de executar qualquer compilação do Gradle é com a ajuda do Gradle Wrapper (em resumo, apenas "Wrapper"). O Wrapper é um script que invoca uma versão declarada do Gradle, baixando-a previamente se necessário. Como resultado, os desenvolvedores podem começar a trabalhar com um projeto Gradle rapidamente sem ter que seguir os processos de instalação manual.

Build.gradle

Por padrão, o arquivo Gradle do projeto principal usa um buildscript para definir os repositórios e dependências do Gradle, isso permite que diferentes projetos usem diferentes versões do Gradle

Node

Node.js é um ambiente de execução JavaScript que permite executar aplicações desenvolvidas com a linguagem de forma autônoma, sem depender de um navegador. Com ele, é possível criar praticamente qualquer tipo de aplicações web, desde servidores para sites estáticos e dinâmicos, até APIs e sistemas baseados em microserviços.

Package.json

O arquivo package.json é o ponto de partida de qualquer projeto NodeJS. Ele é responsável por descrever o seu projeto, informar as engines (versão do node e do npm), url do repositório, versão do projeto, dependências de produção e de desenvolvimento dentre outras coisas.

NPM

O npm é uma ferramenta de linha de comando que ajuda a interagir com plataformas online, como navegadores e servidores. Essa utilidade auxilia na instalação e desinstalação de pacotes, gerenciamento das versões e gerenciamento de dependências necessárias para executar um projeto.

Comandos:

Npm install –instala as dependencias que são configuradas no package.json

NVM

É um gerenciador de “instâncias” Node.js, ou seja, uma ferramenta que nos permite possuir múltiplas instalações do Node.js em nossa máquina.

nvm ls - lista todas as versões instaladas em sua máquina.

nvm ls-remote - lista todas as versões disponíveis para download do Node.js.

nvm install vX.X.X - baixa e instala uma versão do Node.js. Obs.: troque "vX.X.X" por "v0.10.22".

nvm uninstall vX.X.X - desinstala uma versão Node.js de sua máquina.

nvm use vX.X.X - define uma versão Node para uso.