

AULA 19 - IMPLANTAÇÃO E CONTAINERS

Disciplina de Backend - Professor Ramon Venson - SATC 2024

Maven CLI

O comando `mvn` pode ser usado para rodar a ferramenta maven e interagir com o projeto. Um `wrapper` é geralmente disponibilizado na pasta de projetos criados pela IDE para que seja possível rodar um projeto sem o Maven instalado.

Esse comando permite rodar perfis de execução específicos, como `test` para rodar testes de aplicação e `compile` para compilação.

Plugins para o Maven como o do projeto `spring-boot` permitem a definição de perfis de execução alternativos, como o `mvn spring-boot:run` para rodar uma aplicação `spring-boot`.

Lista de Comandos

comando	descrição
<code>mvn install</code>	Instala dependências especificadas no arquivo <code>pom.xml</code>
<code>mvn test</code>	Roda todos os casos de teste declarados no projeto
<code>mvn compile</code>	Compila o código fonte do projeto
<code>mvn clean</code>	Remove todos os arquivos compilados (pasta <code>target</code>)
<code>mvn package</code>	Cria um pacote WAR ou JAR para distribuição
<code>mvn spring-boot:run</code>	Roda um projeto spring-boot

JAR

O comando `java -jar meuapp.jar` pode ser utilizado para rodar uma aplicação distribuível no formato `jar`.

O `jar` é um tipo de arquivo compactado executável usado para distribuição das classes java e outros recursos de um projeto. Esse tipo de pacote também pode ser "ofuscado" para que o conteúdo não seja (facilmente) identificável.

Docker

O *Docker* é um padrão de encapsulamento de bibliotecas e ambientes em **containers padronizados**

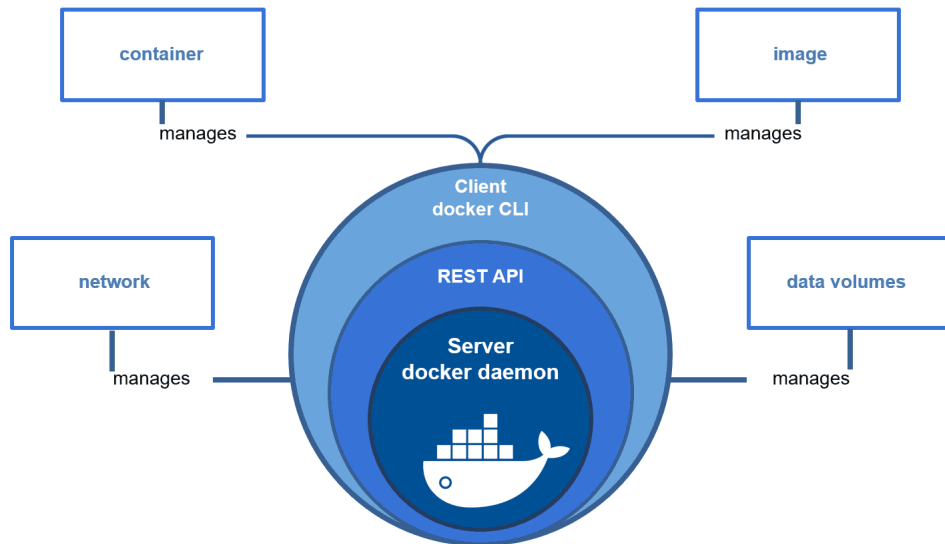
Isso permite que ambientes de execução sejam compartilhados sem diferenças e com o mínimo de esforço.



Componentes do Docker

O Docker pode ser dividido em diferentes componentes:

- Imagem
- Container
- Volume
- Rede



Imagens

Uma imagem Docker define o ambiente e o conjunto de instruções que serão executadas em um **container**.

Podemos consultar as imagens disponíveis localmente no Docker usando o comando:

```
docker image ls
```

Para instalar uma imagem vinda do repositório hub.docker.com:

```
docker pull docker # imagem oficial do docker
```

Containers

Containers são instâncias de ambientes e aplicações. **Containers não são máquinas virtuais**, visto que estas requerem um sistema operacional completo.

Diferentes containers podem rodar simultaneamente na mesma máquina hospedeira (*host*) de forma completamente isolada ou compartilhando recursos.

Podemos consultar os containers rodando localmente:

```
docker container ps
```

Para rodar um container a partir de uma imagem:

```
docker run docker:latest
```


Volumes

Um container não possui persistência dos dados após o fim do seu ciclo de vida. O uso de volumes implementa um sistema de arquivos virtual para os containers armazenarem e compartilharem dados entre si e a máquina hospedeira.

Redes

Outro recurso importante e poderoso do docker é a possibilidade de criar redes virtuais, que interagem com o hospedeiro e entre os containers. Isso permite que a comunicação de rede de diferentes containers seja encapsulada em uma estrutura mais segura.

Dockerfile

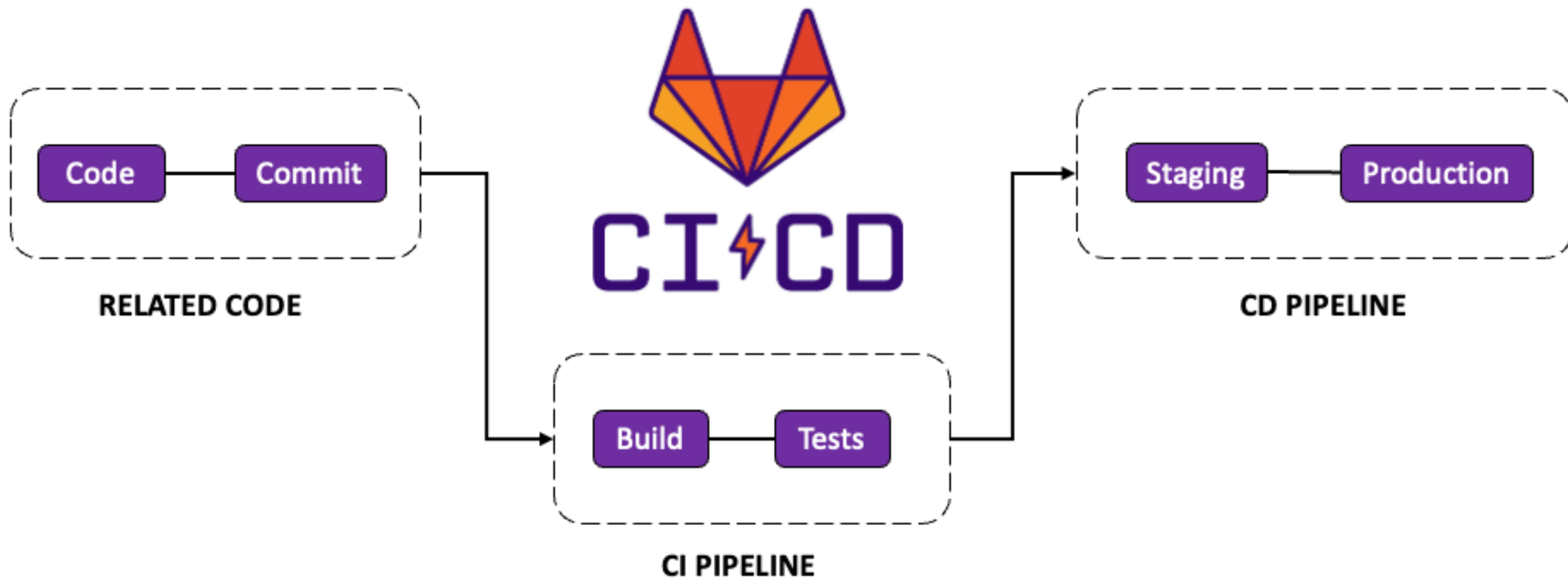
O `Dockerfile` é um arquivo de texto usado para definir a criação de uma imagem. Ele permite partir de uma imagem previamente criada, determinando novas características e pacotes para a imagem resultante:

```
FROM debian:latest
RUN apt-get update && apt-get upgrade -y
RUN apt-get install nginx -y
CMD ["nginx", "-g", "daemon off;"]
```

CI/CD

Continuous Integration (CI) funciona enviando pequenos pedaços de código para a base de código do seu aplicativo hospedada em um repositório Git e, a cada envio, execute um pipeline de scripts para criar, testar e validar as alterações de código antes de mesclá-las na ramificação principal.

Continuous Delivery (CD) consiste em um passo adicional da CI, implantando seu aplicativo na produção a cada push na ramificação padrão do repositório.



O CI/CD do GitLab é configurado por um arquivo chamado `.gitlab-ci.yml`, colocado na raiz do repositório. Os scripts definidos neste arquivo são executados pelo GitLab Runner.

Os Runners podem ser configurados em ambiente privado (mesmo que o repositório esteja hospedado no [GitLab](#)). O GitLab também disponibiliza *Shared Runners* que são compartilhados entre os projetos hospedados na plataforma.

Exemplo de `.gitlab-ci.yml`

```
image: node:8.10.0
cache:
  paths:
    - node_modules/
stages:
  - deploy_production
production:
  image: ruby:latest
  only:
    - master
  stage: deploy_production
  script:
    - apt-get update -qy
    - apt-get install -y ruby-dev
    - gem install dpl
    - dpl --provider=heroku --app=$HEROKU_APP_NAME --api-key=$HEROKU_API_KEY
```

Configurando o `.gitlab-ci.yml`

O arquivo `.gitlab-ci.yml` deve ser adicionado na pasta raiz do projeto para que o Gitlab possa reconhecer as configurações e executar os *runners*.

É nesse arquivo que vamos especificar os **Stages**, as fases do CI que geralmente serão executadas depois de um commit no repositório.

Cada **Stage** possui diferentes **Jobs**, que definem uma tarefa rodada pelos *runners*.

Sintaxe do `.gitlab-ci.yml`

Exemplo

A cada commit adicionado em nosso repositório, vamos garantir que o software continua passando em todos os testes e ainda é possível ser compilado. Dessa forma podemos definir inicialmente dois **Stages**: `test` e `build`

```
stages:  
  - test  
  - build
```

No mesmo arquivo vamos definir um job que fará parte do primeiro stage, e fará os testes unitários no nosso projeto:

```
testes-unitarios:  
  stage: test  
  image: maven:3.8.3-openjdk-17  
  script:  
    - mvn clean test
```

O parâmetro `image` define uma imagem docker que será utilizada para rodar o *job* que estamos chamando de `testes-unitarios`.

O parâmetro `script` define uma lista de comandos que serão executados pelo *runner*. Nesse caso, vamos limpar e testar o projeto.

Vamos adicionar também um *job* que fará a compilação da nossa aplicação:

```
pacote-jar:  
  stage: build  
  image: maven:3.8.3-openjdk-17  
  script:  
    - mvn clean package -DskipTests  
  artifacts:  
    paths:  
      - ./target/*.jar  
    when: always  
    expire_in: 1 hrs  
  only:  
    - master
```

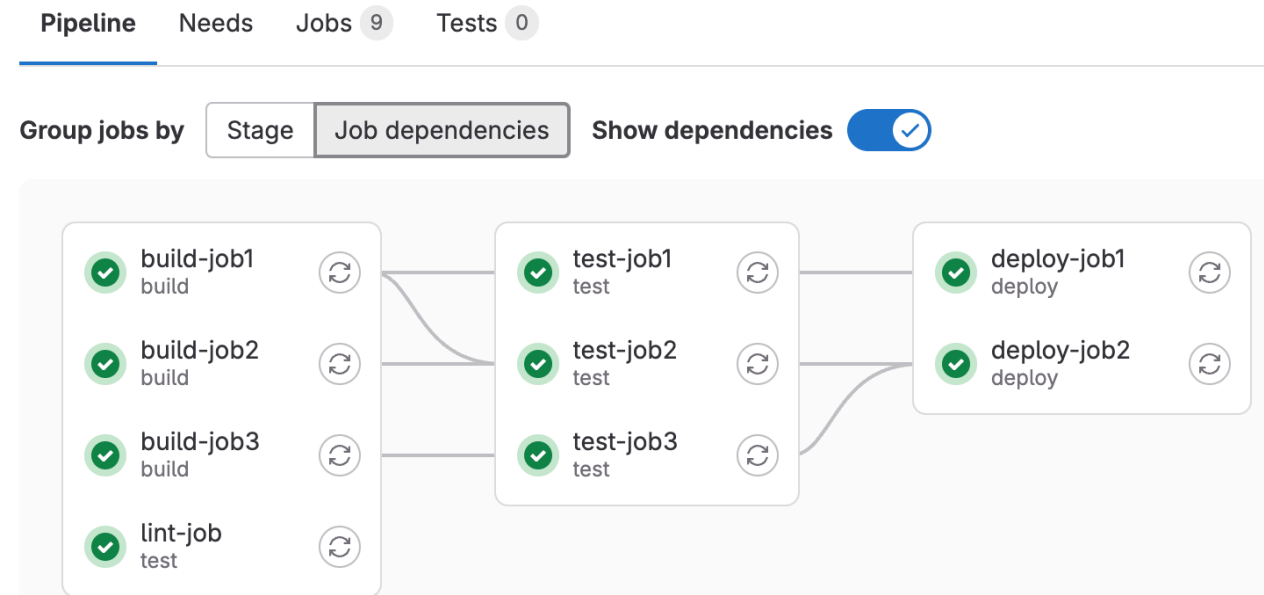
Aqui temos um novo parâmetro: o `artifacts` salva o arquivo `.jar` gerado para uso em outros *runners* e também pode ser acessado na página do projeto.

Os parâmetros `when` e `expire_in` definem que esse artefato será armazenado sempre, mesmo em caso de insucesso, por pelo menos uma hora.

Repare temos ainda um parâmetro chamado `only`, que serve para que este *job* seja executado apenas quando o *commit* for relacionado ao *branch* `master`

Executando o CICD

Após criar o arquivo `.gitlab-ci.yml` na raiz do projeto, vamos commitar e enviar as alterações para a página do projeto no Gitlab. Dessa forma, a pipeline deve ser inicializada imediatamente (ou assim que os *runners* estiverem disponíveis).



Containerizando uma Aplicação

Para criar um container que rode uma aplicação, primeiro iremos criar uma imagem docker para contê-la, incluindo todos os pacotes e dependências necessárias para o seu funcionamento. Para isso vamos criar um arquivo

`Dockerfile` **na pasta raiz** da aplicação:

```
FROM openjdk:17-alpine # imagem base
COPY /target/*.jar app.jar # copia apenas o jar
EXPOSE 8080 # documenta o uso da porta 8080
ENTRYPOINT ["java", "-jar", "/app.jar"] # comando ao iniciar o container
```

Criando uma imagem nova

Para criar uma imagem nova, na raiz da aplicação, onde está definido o `Dockerfile`, vamos rodar o seguinte comando:

```
docker build -t meuapp:latest .
```

Verifique se a imagem foi gerada corretamente:

```
docker image ls
```

E para rodar a imagem recém criada, use:

```
docker run meuapp:latest
```

Gitlab Registry

O Gitlab Registry é uma funcionalidade do Gitlab que permite atribuir imagens à um projeto. Desde que essa funcionalidade esteja habilitada no projeto, podemos realizar usar o terminal para realizar o login nesse repositório.

O login pode ser realizado com usuário e senha:

```
docker login -u meu_usuario_gitlab -p minha_senha registry.gitlab.com
```

No entanto, é interessante criar um Token de acesso para realizar essa operação.

Criando um Token

Para criar um **Token de Autenticação**, vá até as configurações do seu projeto e acesse a opção `Tokens de Acesso` > `Adicionar novo token`. Não esqueça de marcar os escopos relacionados ao envio e leitura de registros: `read_registry` e `write_registry`.

Com o token em mãos, utilize ele ao invés da sua senha:

```
docker login -u meu_usuario_gitlab -p meu_token registry.gitlab.com
```

CICD e Registry

É possível configurar um **Job** para registrar imagens automaticamente no Registry do seu projeto, para isso, adicione a seguinte tarefa no seu `.gitlab-ci.yml`.

Utilizando *Runners* é possível utilizar variáveis de ambiente padronizadas do Gitlab para autenticar e acessar a URL do Registry do seu projeto automaticamente.

Segue o exemplo:

```
deploy-gitlab:
  image: docker
  stage: deploy
  services:
    - docker:dind
  before_script:
    - docker login -u $SCI_REGISTRY_USER -p $SCI_JOB_TOKEN $SCI_REGISTRY
  script:
    - docker build -t $SCI_REGISTRY_IMAGE .
    - docker push $SCI_REGISTRY_IMAGE
  dependencies:
    - build
  only:
    - master
```

Ambiente de Produção

Existem diversas maneiras diferentes de tornar uma aplicação disponível na internet a partir da pasta do projeto, do projeto empacotado (jar ou war) ou do container docker.

Quando uma aplicação encontra-se em ambiente operacional, onde pode ser utilizado para executar suas tarefas pretendidas em ambiente final, chamamos isso de **Ambiente de Produção**.

Existem diversas maneiras de realizar a implantação de um software a partir do seu código fonte, projeto compactado (`jar` ou `war`) ou seu container docker.

Uma estrutura de nuvem (*cloud*) pode ser utilizada como implantação definitiva para um software, contendo todas as ferramentas necessárias para escalonar, controlar e monitorar o funcionamento da aplicação.

No entanto, dezenas de configurações incluem também o uso de servidores locais, com servidores dedicados e/ou virtualizados, além de nuvens privadas em configurações chamadas de ***On-Premise***.

Implantação na Nuvem

Serviços populares em nuvem oferecem diferentes configurações para implantação de aplicações dependendo das necessidades de uso e escalonamento. Em algumas das nuvens populares podem ser utilizados:

- **Amazon Web Services:** Beanstalk (EB), Kubernetes (EKS), Container (ECS), Lambda
- **Azure:** App Service, Kubernetes (AKS), Spring Apps, Functions
- **Google Cloud:** App Engine, Kubernetes (GKE), Cloud Run, Cloud Functions
- **Digital Ocean:** App Plataforma, Kubernetes, Droplets
- **Oracle:** Cloud Infrastructure (OCI), Kubernetes (OKE), Functions
- **IBM:** Cloud Foundry, Kubernetes, Functions

Google Cloud Run

Implantar um novo serviço usando o Cloud Run da Google exige que a aplicação esteja no formato de um container docker, mas permite liberdade para definir os serviços e a configuração de plataforma que será utilizada.

NOTA IMPORTANTE : A maioria dos serviços em nuvem é pago por uso e exige uso de cartão de crédito. Não instancie novos serviços sem ter ideia dos custos gerados na sua conta.

Alguns serviços contam com um **Free Tier**, que estabelece cotas de uso que não são tarifados pelas plataformas.

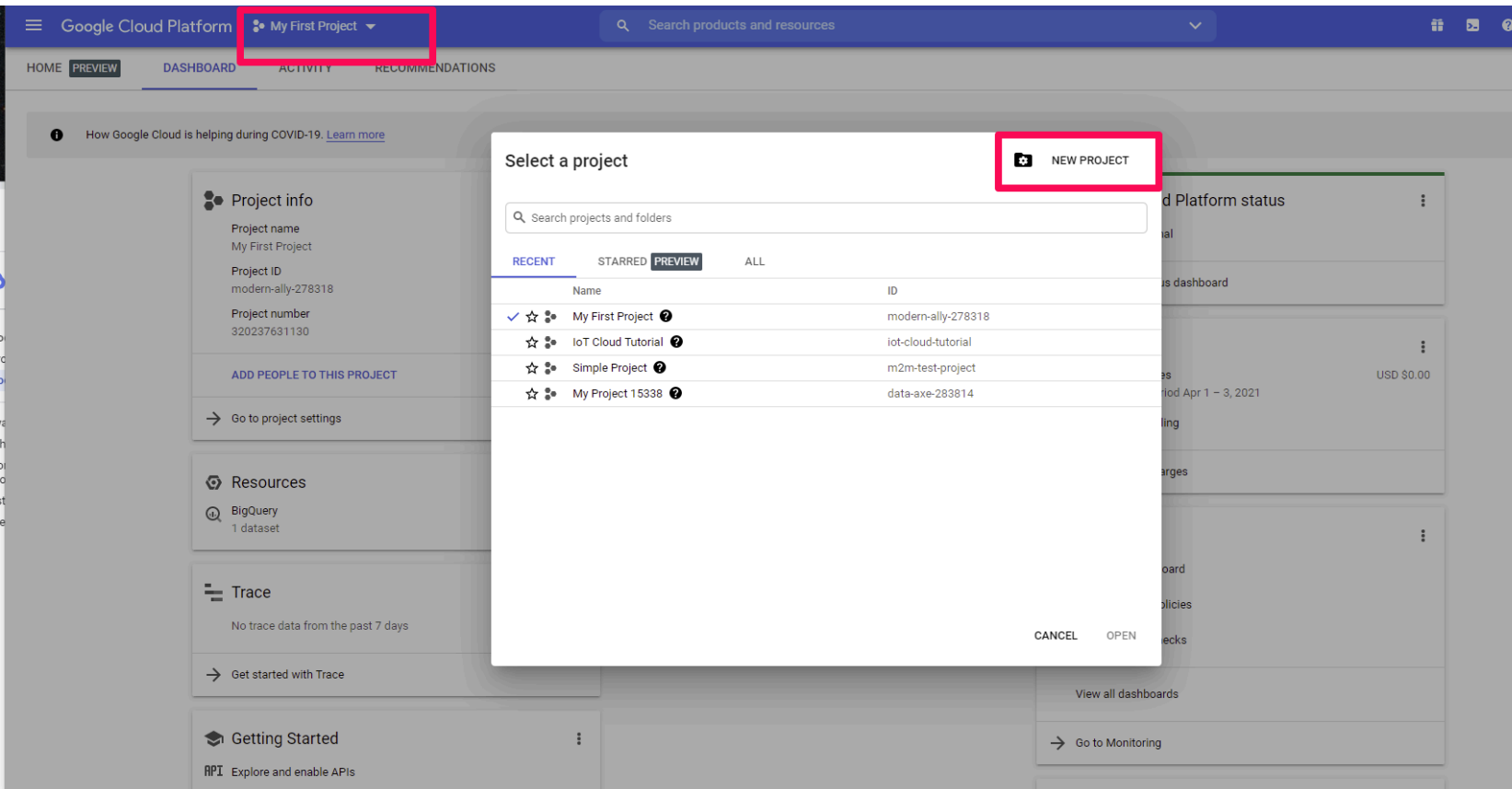
Outra vantagem do Cloud Run é o escalonamento automático, que inicia novas instâncias do container em função da quantidade do uso atual da aplicação.

Para configurações com maior complexidade e controle, como redes específicas e padrões de escalonamento mais rígidos, pode-se migrar do Cloud Run para o **Google Kubernetes Engine** (GKE)

Configurando Google Cloud

Vamos seguir os seguintes passos para configurar nossa conta no Google Cloud:

- Criar/Acessar uma conta Google e acessar o site: console.google.com;
- Criar um novo projeto;
- Ativar o `Cloud Run Admin API`, `Service Usage API` `Cloud Build API` ;
- Criar uma nova conta de serviço (opcional);
- Criar uma nova chave para a conta de serviço e salvar;
- Fazer a compilação do jar;
- Adicionar o Dockerfile;
- Usar o Cloud CLI e fazer o upload da aplicação;



Tela inicial do
Google Cloud

Para criar um novo
projeto, acesse o
menu superior e
clique no botão **New
Project**

Novo projeto

Crie um novo projeto

Nessa tela apenas especifique o nome do projeto. De preferência à nomes únicos.



Voce tem 11 projects restantes na sua cota. Solicite um aumento ou exclua projetos. [Saiba mais](#)

[MANAGE QUOTAS](#)

Nome do projeto *

rvenson-professor



ID do projeto: rvenson-professor. Não é possível mudar depois. [EDITAR](#)

Local *

 Sem organização

[PROCURAR](#)

Pasta ou organização pai

CRIAR

CANCELAR

The screenshot displays the Google Cloud console interface. At the top, the Google Cloud logo and the project name 'rvenson-satc-teste' are visible. The left sidebar contains a menu with the following items: 'APIs e serviços' (selected), 'Biblioteca', 'Credenciais', 'Tela de permissão OAuth', and 'Contratos de uso de página'. The main content area is titled 'APIs e serviços' and includes a button '+ ATIVAR APIS E SERVIÇOS'. Below this, there is a 'Tráfego' section with a timeline showing 'UTC-3', '18:00', '27 de mai.', and '06:00'.

Adicione novas APIs

Procure pelo menu
API e Serviços e
em APIs e
serviços ativados
selecione o botão
Ativar APIS e
Serviços

Adicione as APIs
necessárias

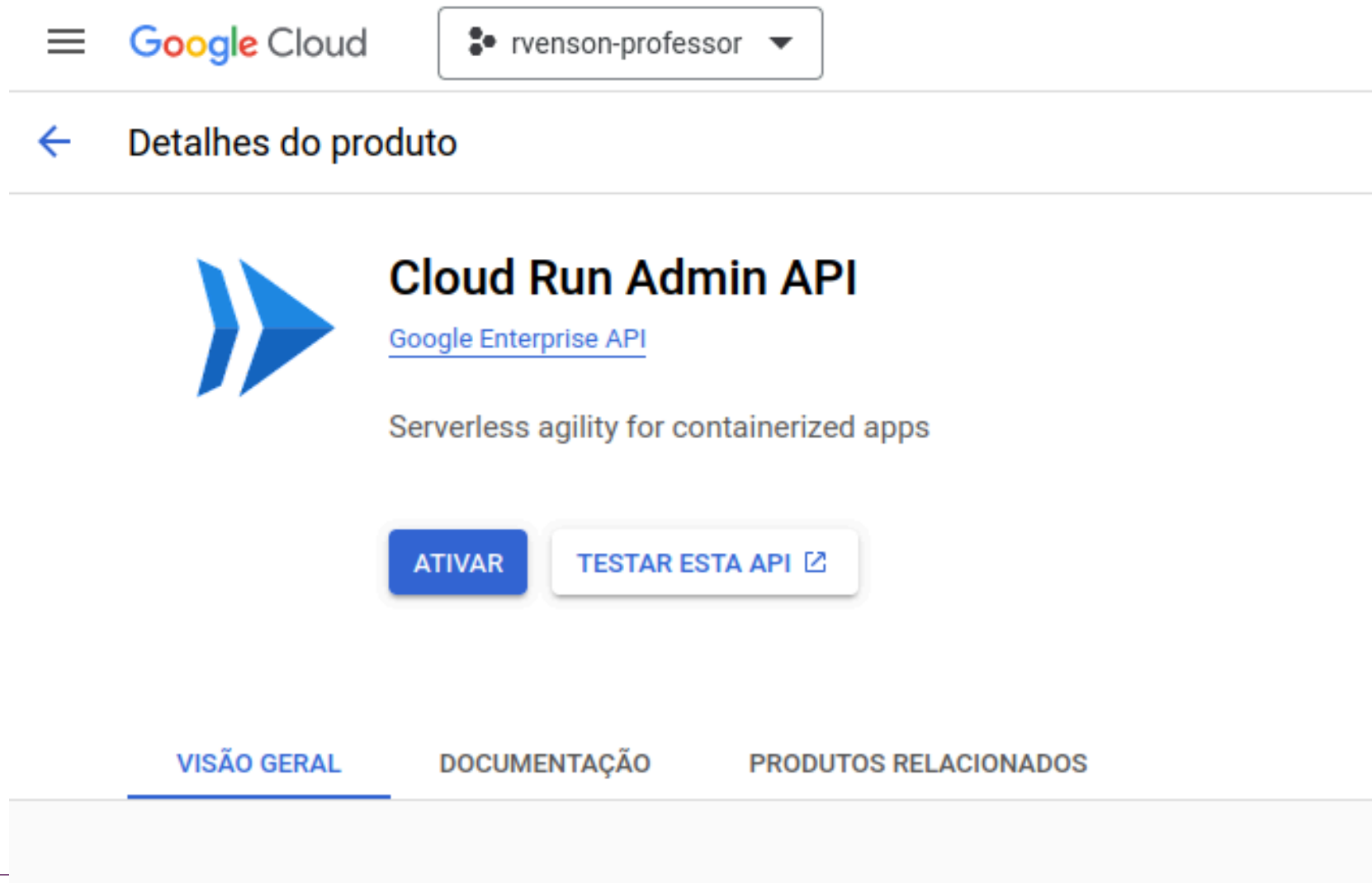
Para essa
implantação, vamos
precisar de três APIs:

Cloud Run Admin ,

Service Usage API

Cloud Build API .


Ative as três nessa
sequência.



Google Cloud

rvenson-professor

← Detalhes do produto

 **Cloud Run Admin API**

[Google Enterprise API](#)

Serverless agility for containerized apps

ATIVAR TESTAR ESTA API

VISÃO GERAL DOCUMENTAÇÃO PRODUTOS RELACIONADOS

Adicione as APIs
necessárias

Procure pelo menu
IAM e
administração e em
Contas de serviço
clique na conta de
serviço Default
Compute service
account (ou crie
uma nova)

The screenshot shows the Google Cloud IAM and Administration console. The left sidebar contains the navigation menu with 'Contas de serviço' (Service Accounts) highlighted. The main content area shows the 'Contas de serviço' page for the project 'rvenson-professor'. At the top, there is a search bar and a dropdown menu for the project. Below the header, there is a notification about Google's policy change regarding service account keys. The main section is titled 'Contas de serviço para o projeto "rvenson-professor"' and includes a description of service accounts. Below this, there is a table listing the service accounts. The table has columns for 'E-mail', 'Status', and 'Nome'. One service account is listed: 'Default compute service account' with the email '623986932235-compute@developer.gserviceaccount.com' and a status of 'Ativado' (Active).

Google Cloud

rvenson-professor

Pesquise (/) recursos, documentos, produtos e m

IAM e administra...

Contas de serviço

+ CRIAR CONTA DE SERVIÇO

EXCLUIR

GERENCIAR

Starting June 16, 2024, Google will automatically disable service account keys detected in public policy. [Learn more](#)

DISMISS

Contas de serviço para o projeto "rvenson-professor"

Uma conta de serviço representa uma identidade de serviço do Google Cloud, como o código que é executado nas v

Políticas da organização podem ser usadas para proteger contas de serviço e bloquear recursos arriscados de cont [sobre políticas da organização da conta de serviço](#)

Filtro Insira o nome ou o valor da propriedade

<input type="checkbox"/>	E-mail	Status	Nome ↑
<input type="checkbox"/>	623986932235-compute@developer.gserviceaccount.com	Ativado	Default compute service account

Gere uma Chave

Faça a geração de uma nova chave aba

Chaves da conta de serviço, botão

Adicionar chave e opção Criar nova

chave. Crie a chave no formato json e faça o download.

The screenshot shows the Google Cloud IAM and Admin console. The left sidebar lists various services, with 'Contas de serviço' (Service Accounts) selected. The main panel displays the 'Default compute service account' page, specifically the 'CHAVES' (Keys) tab. A warning message states: 'As chaves da conta de serviço podem representar um risco de segurança caso sejam comprometidas sobre a melhor maneira de autenticar contas de serviço no Google Cloud neste link'. Below this, an information message notes: 'Starting June 16, 2024, Google will automatically disable service account keys detected in public'. The page instructs to 'Adicione um novo par de chaves ou faça upload de um certificado de chave pública do par existente.' and provides a link to 'Bloqueie a criação de chaves da conta de serviço usando as políticas da organização'. A dropdown menu for 'ADICIONAR CHAVE' is open, showing two options: 'Criar nova chave' and 'Fazer upload de uma chave atual'.

Compile sua aplicação

Agora vá até a pasta do seu projeto e realize a compilação do pacote `jar` executando o comando na raiz da aplicação:

```
mvn package
```

ou

```
./mvnw package
```

Certifique-se de que o `.jar` foi gerado corretamente na pasta `target`.

Criando o Dockerfile

Crie um novo arquivo na raiz do projeto chamado `Dockerfile` com o conteúdo:

```
FROM openjdk:17-alpine
COPY /target/*.jar app.jar
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "/app.jar"]
```

Esse arquivo será usado pelo docker para construir uma imagem que executa a sua aplicação.

Você pode substituir o openjdk:17-alpine por openjdk:22-alpine caso use uma versão mais nova da JDK na sua aplicação.

Google Cloud CLI

Primeiro vamos usar o Google Cloud para realizar a *build* da nossa aplicação e armazená-la no **Container Registry**. Isso permite que aplicações possam ser executadas rapidamente com a imagem nos servidores da Google.

Antes de tudo, vamos precisar de uma ferramenta chamada `gcloud`, que pode ser instalada a partir do site Google Cloud. Também podemos usar uma imagem docker que já possui essa ferramenta instalada.

Iniciando um container docker com `gcloud`

Para iniciar um novo container docker, vamos primeiro organizar nosso projeto em uma pasta, junto com a chave `json` que baixamos da conta de serviço:

```
- meu_aplicativo
  - src <-- pasta raiz do seu projeto
  - chave.json <-- chave da conta de serviço
```

De preferência pra hierarquia acima para os próximos passos para não se perder. Abra o terminal na pasta `meu_aplicativo`.

Para rodar um container docker com o gcloud instalado, rode o seguinte comando no terminal enquanto estiver na pasta `meu_aplicativo` (que contém a pasta do projeto + chave):

```
docker run -it --mount src=.,target=/app,type=bind google/cloud-sdk:alpine bash
```

A imagem do gcloud será baixada na primeira vez e um novo container será iniciado. Se tudo correr bem, você estará dentro do container assim que o processo for finalizado.

Verifique que o seu projeto e a chave são acessíveis de dentro do container usando os comandos `cd /app` e `ls -lA`.

Autenticando no gcloud

Vamos autenticar no gcloud usando o comando:

```
gcloud auth activate-service-account --key-file nome-do-arquivo-da-chave.json
```

Se tudo correr bem, configure o caminho do seu projeto usando o comando:

```
gcloud config set project id-do-projeto
```

Verifique o ID do projeto na página do Google Cloud. Por fim, configure o docker:

```
gcloud auth configure-docker
```

Realizando a Build

Para realizar a build do projeto e a criação da imagem que será armazenada na nuvem da Google, use o comando:

```
gcloud builds submit --tag gcr.io/id-do-projeto/nome-do-servico
```

Não esqueça de substituir o id do seu projeto e o nome do servico, que representa um nome da imagem que será armazenada. Você pode utilizar o mesmo nome da sua aplicação (ex.: `meu-app`)

Rodando Cloud Run

Por fim, com a imagem armazenada na nuvem, utilize o mesmo nome de serviço e execute o comando a seguir para criar o *runner* para a sua aplicação no Google Cloud Run.

```
gcloud run deploy nome-do-servico --image gcr.io/id-do-projeto/nome-do-servico --region=us-central1 --platform managed --allow-unauthenticated
```

E pronto! Sua aplicação deve estar disponível na web =)

Caso a aplicação não seja acessível (FORBIDDEN), altere a configuração no Google Run para permitir o acesso não autenticado.

Gitlab CI/CD e Google Run

Para automatizar o deploy usando as pipelines de CI/CD do Gitlab, podemos adicionar (ou criar caso não exista) ao arquivo `gitlab-ci.yml` o seguinte modelo de *job*:

```
deploy-gcloud:
  image: google/cloud-sdk:alpine
  stage: deploy
  script:
    - echo $SERVICE_ACCOUNT_KEY > service-account-key.json
    - gcloud auth activate-service-account --key-file service-account-key.json
    - gcloud config set project $PROJECT_ID
    - gcloud auth configure-docker
    - gcloud builds submit --tag gcr.io/$PROJECT_ID/$SERVICE_ID
    - gcloud run deploy $SERVICE_ID --image gcr.io/$PROJECT_ID/$SERVICE_ID --region=us-central1 --platform managed --allow-unauthenticated
  dependencies:
    - build
  when: manual
  only:
    - master
```

Repare o uso de algumas variáveis no script. Dessa forma vamos precisar inicializar essas variáveis diretamente na configuração do projeto no Gitlab, em `Configurações > CI/CD > Variáveis`.

Insira cada uma das três variáveis:

- Chave `SERVICE_ID` : valor `nome_da_aplicacao` (escolha um nome)
- Chave `PROJECT_ID` : valor `slug-projeto` (nome encontrado na página inicial do projeto no Google Cloud, chamada `ID do projeto`);
- Chave `SERVICE_ACCOUNT_KEY` : copie o conteúdo da chave `.json` gerada anteriormente.

Docker Compose

O Docker Compose é uma ferramenta do Docker (antigamente plugin) que permite definir e rodar múltiplos containers, facilitando a execução de ambientes complexos e permitindo uma criação mais fácil de redes e volumes.

Para usar o Docker Compose, podemos definir na raiz do projeto um arquivo de texto chamado `docker-compose.yml` (esse não é mais um padrão).

```
services:

  postgres:
    image: postgres
    ports:
      - "5432:5432"
    networks:
      - app-network

  meu-app:
    image: meu-app:latest
    container_name: meu-app
    volume: ./data:meu-app
    networks:
      - app-network
    build:
      dockerfile: ./Dockerfile
    ports:
      - "3001:3001"
    depends_on:
      - postgres
```

```
networks:
  app-network:
    driver: nat

volumes:
  meu-app:
```

Essa configuração (exemplo) permite abrir dois containers, um para o banco de dados e outro para a aplicação, conectando-os em uma rede virtual. Para iniciar a configuração, use:

```
docker compose up -D
```

O que aprendemos hoje

- Executar comandos básicos do maven;
- O que é o Docker Engine e como criar imagens e containers;
- Como automatizar pipelines usando o CICD do Gitlab;
- Como criar containers usando o CICD;
- Como fazer deploy de um container no Google Cloud;