

Deploy do Blog Pessoal

no



render

🔗 Deploy do Backend no Render

🔗 1. O que é Deploy?

O verbo **deploy**, em inglês, significa **implantar**.

Em programação, seu sentido está intimamente relacionado à sua tradução literal: fazer um deploy, em termos práticos, significa colocar na nuvem, ou seja, publicar na Internet alguma aplicação que teve seu desenvolvimento concluído.

Quando um site é finalizado por uma pessoa desenvolvedora, ele passa pelos últimos testes e finalmente ele é hospedado na nuvem através do processo chamado deploy.

Do mesmo modo, quando um sistema sofre alguma melhoria ou alteração em seu código-fonte, implementar essa alteração ao sistema que está publicado (em Produção), também é chamado de deploy, só que incremental.

🔗 2. O que veremos por aqui?

Esse documento é um passo a passo para você enviar a sua aplicação Spring, gratuitamente para a nuvem. Este processo irá gerar um link de acesso a sua aplicação, que poderá ser acessado de qualquer lugar, a partir de qualquer dispositivo com acesso a Internet.

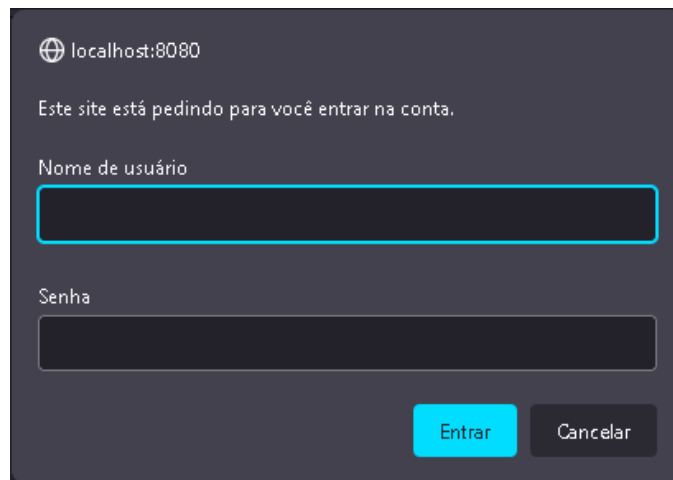
Para efetuar o Deploy vamos precisar fazer algumas modificações em nosso projeto, que serão detalhadas nos próximos tópicos.

🔗 🖱️ Passo 01 - Criar a Documentação da API

Para criar a Documentação da API no Swagger, utilize o [Guia de Configuração do Springdoc](#).

🔗 🖱️ Passo 02 - Testar a API no seu computador

1. Execute a sua aplicação localmente pelo STS
2. Abra o endereço: <http://localhost:8080/> no seu navegador
3. A sua aplicação deverá exibir a tela de **Login (Usuário e Senha)**. Utilize o **usuário:** root@root.com e a **Senha:** **rootroot**, que foram criados anteriormente.



4. Caso a aplicação **não** solicite o **Usuário** e a **Senha**, feche todas as janelas abertas do seu Navegador da Internet, abra novamente e acesse o endereço acima. Se o problema persistir, verifique a sua configuração do Swagger.
5. Verifique se após o login, o **Swagger** está inicializando automaticamente.
6. Caso você não tenha testado no **Insomnia**, execute os testes e verifique se tudo está funcionando.
7. Em especial, verifique se o Método **logar** está devolvendo o **Token**.
8. Antes de continuar a configuração do projeto para efetuar o Deploy, não esqueça de **parar a execução do Projeto no STS**.



IMPORTANTE: Não altere a senha do usuário root@root.com. Os instrutores da sua turma utilizarão este usuário para abrir, testar e corrigir a sua aplicação.



ATENÇÃO: Lembre-se que antes de fazer o Deploy é fundamental que a API esteja rodando e sem erros. Não faça os testes via Swagger porque o Swagger não utiliza todos os recursos da Spring Security, em especial o Token. Utilize o Insomnia para fazer os testes.

Passo 03 - Criar uma conta grátis no Render

3.1 O que é o Render?

O Render é uma plataforma unificada para criar e executar todos os seus aplicativos e sites. O Render permite criar e executar todos os seus aplicativos e sites com SSL gratuito, um CDN global, redes privadas e implantações automáticas do Git.

Uma **CDN** (Rede de Entrega de Conteúdo) é um grupo de servidores geograficamente distribuídos que aceleram a entrega do conteúdo da Web, aproximando-o de onde os usuários estão.

O Render é classificado como um **PaaS** (plataforma como serviço), ou seja, é um conjunto de serviços para criar e gerenciar aplicativos na nuvem. PaaS fornece os componentes de infraestrutura, que permitem que as pessoas Desenvolvedoras criem, integrem, migrem, implementem, protejam e gerenciem aplicativos móveis e da web, de forma simples e rápida.

Modelos de Serviços na Nuvem:

- **Plataforma como um serviço (PaaS):** Um provedor de serviços oferece acesso a um ambiente baseado em cloud no qual os usuários podem desenvolver e fornecer aplicativos. Além do **Render**, o **Render** e o **Azure** da Microsoft também utilizam este modelo.
- **Infraestrutura como um serviço (IaaS):** Um provedor de serviços fornece aos clientes acesso Pay As You Go (Pague pelo que você usar), para armazenamento, rede, servidores e outros recursos de computação na cloud. O **AWS da Amazon** e a **Digital Ocean** seguem este modelo.
- **Software como um serviço (SaaS):** Um provedor de serviços oferece softwares e aplicativos por meio da Internet. Os usuários subscrevem ao software e o acessam por meio da web ou de APIs do fabricante. O **Google Apps** e o **Microsoft Office 365** seguem este modelo.

Um grande diferencial do Render é que ele oferece **contas gratuitas**, com algumas limitações, que permitem hospedar aplicações desenvolvidas em diversas linguagens e **01 Banco de dados PostgreSQL**.

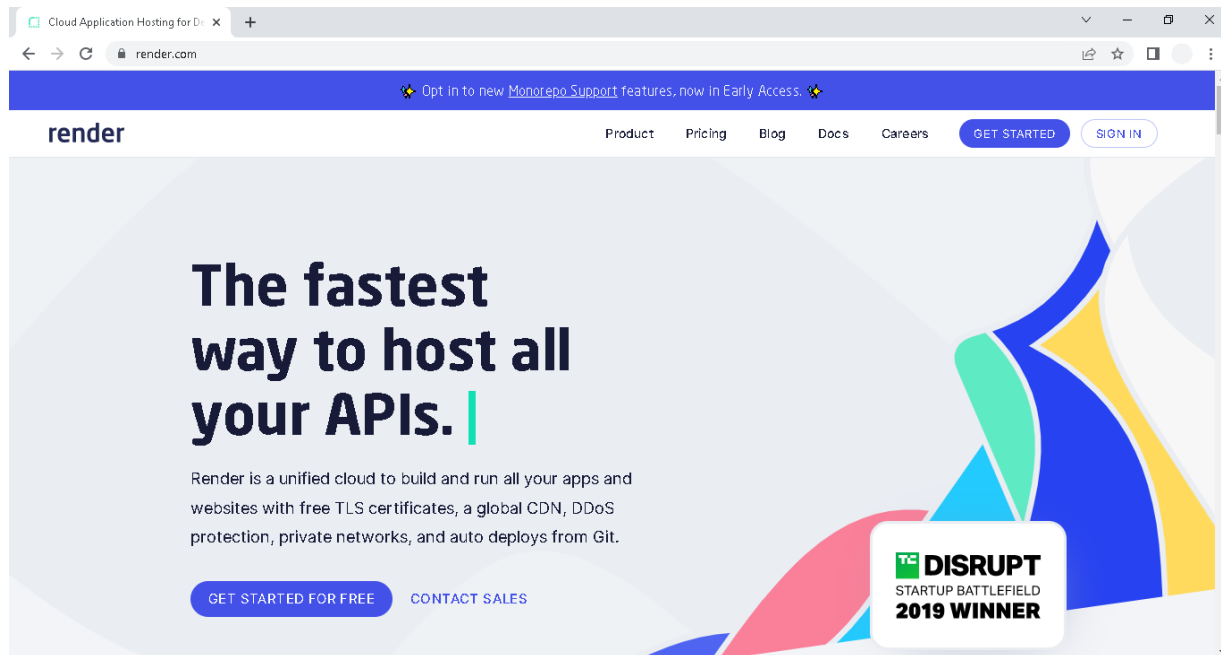
Principais Limitações do Plano Gratuito:

- Se a aplicação ficar **15 minutos sem receber nenhuma requisição**, o aplicativo é finalizado e será reiniciado somente quando receber outra requisição, para economizar os recursos da plataforma.
- Os servidores do Render estão disponíveis apenas na Europa, Ásia e nos Estados Unidos.
- Os recursos de Memória, Disco e Processamento são limitados, logo a aplicação e principalmente o Banco de dados não podem ser muito grandes.
- Aceita **apenas um Banco de dados por conta**;
- O **tráfego mensal é limitado a 750 horas somando todas as aplicações e o Banco de dados**, ou seja, se ultrapassar este valor antes do mês acabar, sua conta ficará suspensa até o mês seguinte;
- Dependendo da linguagem, o Deploy da aplicação deverá ser realizado via Docker.

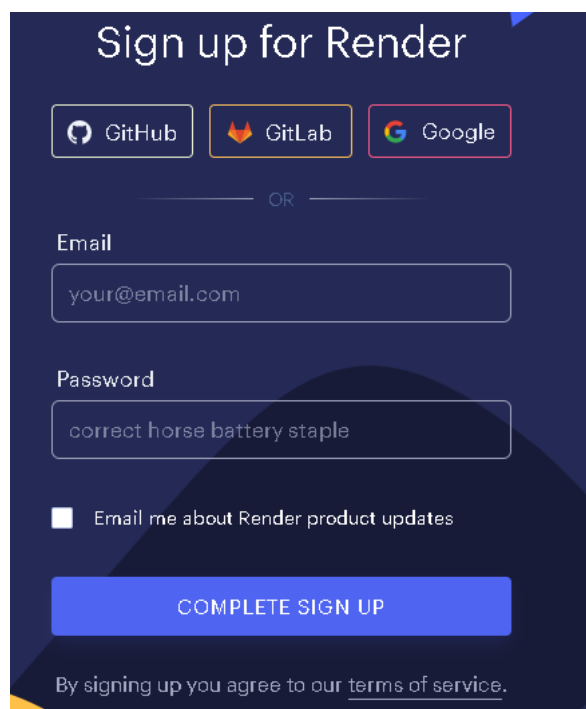
[Documentação: Render - Plano Gratuito](#)

Vamos criar a conta no Render para fazermos o Deploy:

1. Acesse o endereço: <https://www.render.com>




2. Existem diversas formas de criar uma conta no Render. Neste Guia utilizaremos a conta do Github. Clique no link **GitHub**, como mostra a figura abaixo:



3. Na próxima janela, digite o endereço do e-mail e a senha da sua conta do Github, e em seguida clique no botão **Sign in**





Sign in to GitHub
to continue to Render




Username or email address

Password


[Forgot password?](#)


Sign in


4. Na próxima janela, autorize o Render a acessar a sua conta do Github, clicando no botão **Authorize Render**.




Render by **Render** would like permission to:

 Verify your GitHub identity (rafaelpinfo)

 Know which resources you can access

 Act on your behalf
[Learn more](#)

Resources on your account


 Email addresses (read)
View your email addresses


[Learn more about Render](#)


Cancel

Authorize Render

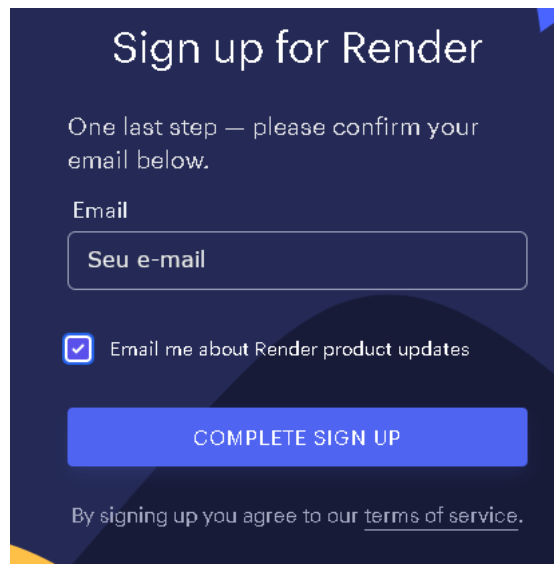
Authorizing will redirect to
<https://dashboard.render.com>

 Not owned or
operated by GitHub

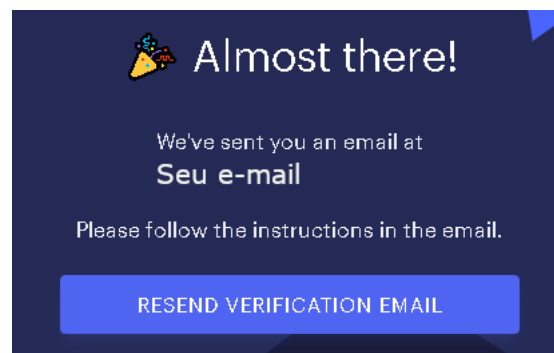
 Created 5 years ago

 More than 1K
GitHub users

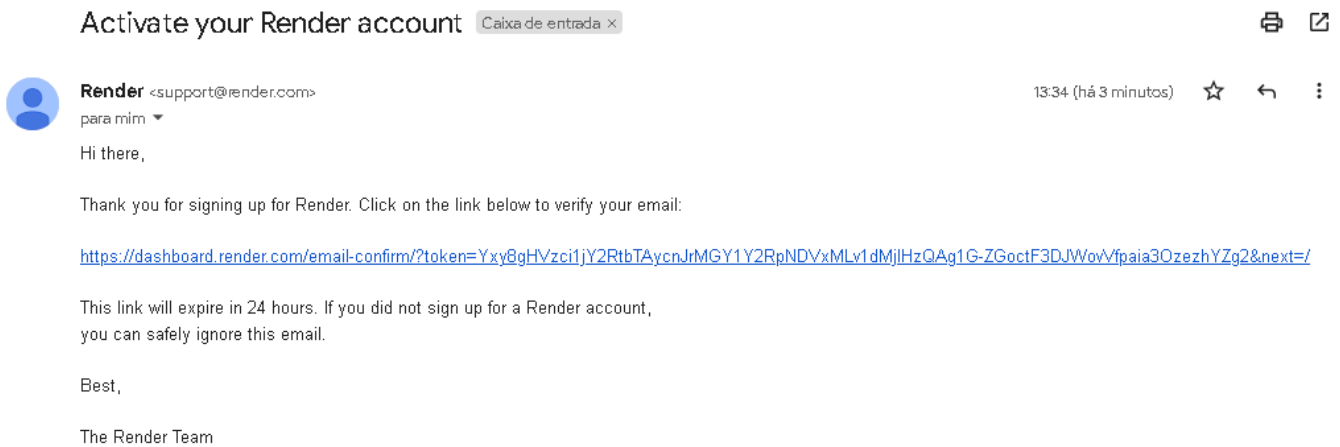
5. Na próxima janela, confirme se o endereço do e-mail está correto e clique no botão **COMPLETE SIGN UP**



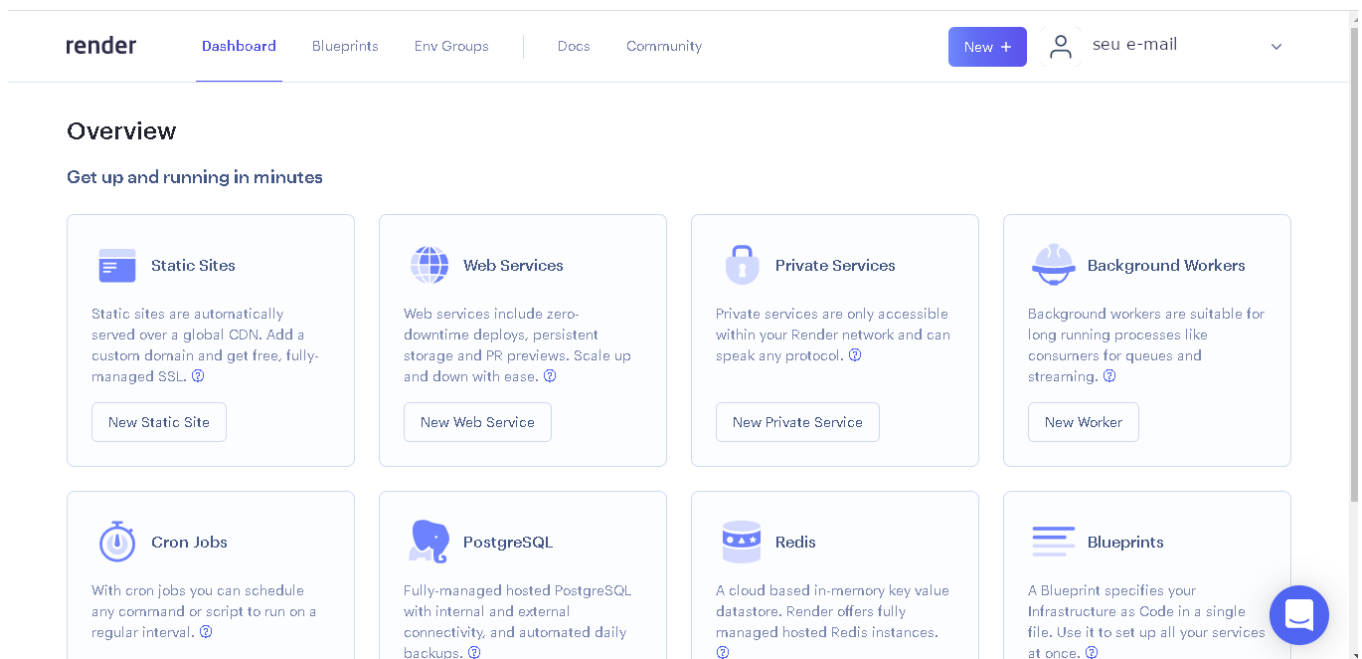
6. Na próxima janela, será exibida uma mensagem solicitando que você verifique se recebeu uma mensagem no seu e-mail para validar a sua conta no Render. Abra a sua conta de e-mail e verifique se o e-mail foi recebido. Caso não tenha recebido, clique no botão **RESEND VERIFICATION EMAIL**.



7. Abra o e-mail enviado pelo Render (semelhante a imagem abaixo) e **clique no link para validar a sua conta**.



8. Depois de clicar no link, sua conta será validada e será redirecionada para a tela do **Dashboard**.




ATENÇÃO: Conclua todas etapas do processo de criação da conta no Render antes de avançar para o próximo passo do Deploy.

Passo 04 - Adicionar a Dependência do PostgreSQL no pom.xml

O Render, na sua versão gratuita, utiliza o **PostgreSQL** como **SGBD** (Sistema Gerenciador de Banco de dados).

No Bloco 02 estamos utilizando o **MySQL** para desenvolver o Blog Pessoal. Ambos são Banco de dados Relacionais e graças ao **Spring Data JPA**, não será necessário realizar nenhuma alteração no código da nossa aplicação. A única mudança necessária, além de adicionar a **Dependência do PostgreSQL no pom.xml**, será necessário configurar a conexão com o Banco de dados PostgreSQL na nuvem.

 [Site Oficial: PostgreSQL](#)

No arquivo, **pom.xml**, vamos adicionar as linhas abaixo, com a dependência do **PostgreSQL**:

```
<dependency>
  <groupId>org.postgresql</groupId>
  <artifactId>postgresql</artifactId>
</dependency>
```

Passo 05 - Configurar o Banco de Dados na Nuvem

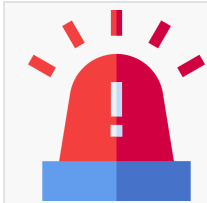
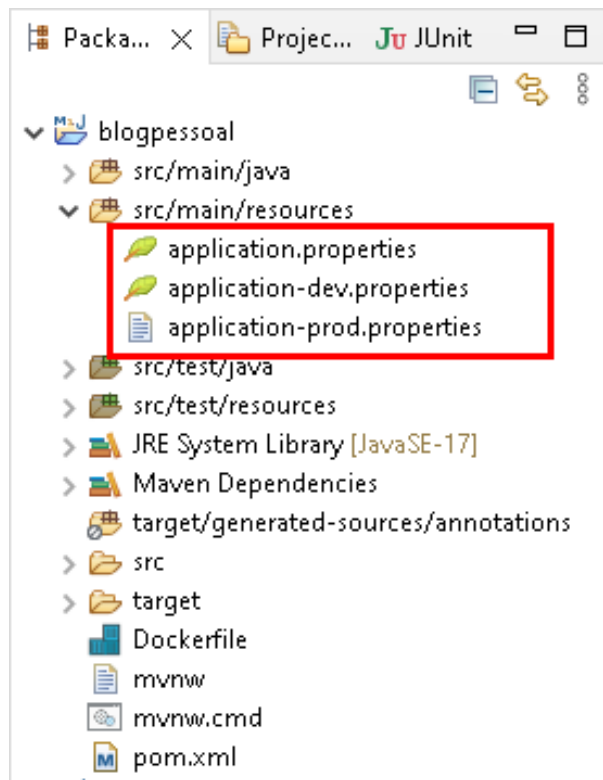
A Configuração do Banco de dados Local é diferente da configuração que será utilizada no Render.

No passo anterior, adicionamos a Dependência do PostgreSQL no arquivo pom.xml, neste passo vamos configurar a aplicação para acessar o Banco de dados remoto que será criado no Render.

Para simplificar o processo, vamos utilizar um recurso do Spring chamado **Profiles** (perfis), que nada mais é do que criar um modelo de configuração para cada etapa do desenvolvimento da aplicação, ou seja, uma configuração para usar localmente (**application-dev.properties**) e outra configuração para usar na nuvem (**application-prod.properties**).

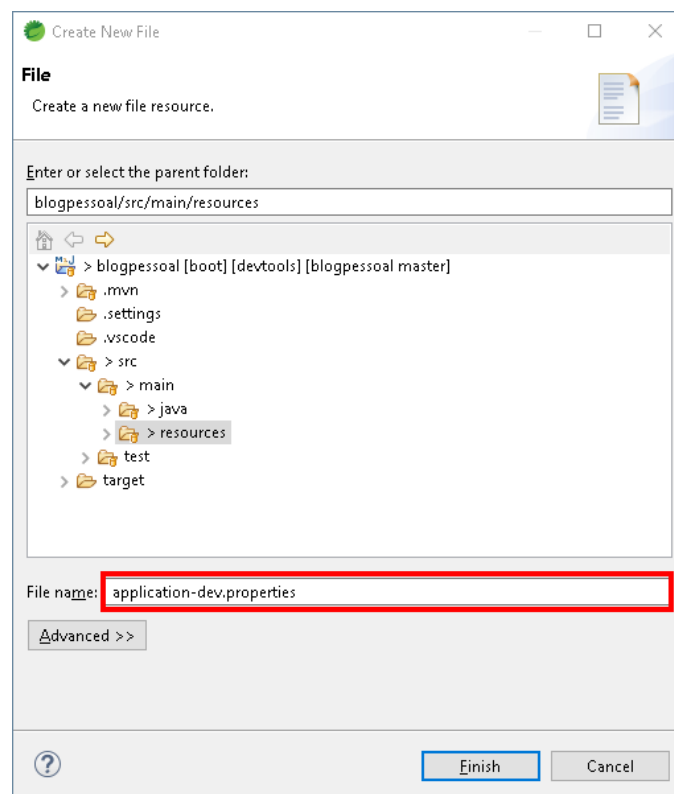
O grande benefício dos Profiles é simplificar a troca entre a configuração Local (**MySQL**) e a configuração Remota do Render (**PostgreSQL**).

1. Na Source Folder **src/main/resources**, crie os arquivos **application-dev.properties** (Configuração do Banco de dados local) e **application-prod.properties** (Configuração do Banco de dados na nuvem), como mostra a imagem abaixo:

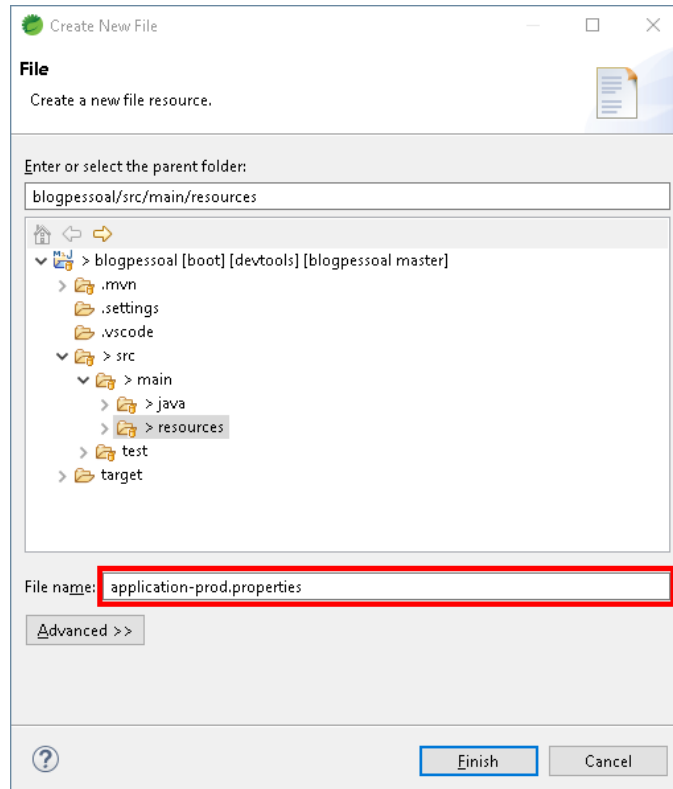


ALERTA DE BSM: Mantenha a atenção aos detalhes ao criar os arquivos *application-dev.properties* e *application-prod.properties*. Cuidado para não se equivocar ao nomear os arquivos ou criar em uma pasta diferente.

2. Vamos criar o primeiro arquivo. No lado esquerdo superior, na **Guia Package explorer**, na Source Folder **src/main/resources**, clique com o botão direito do mouse e clique na opção **New → File**.
3. Em **File name**, digite o nome do primeiro arquivo (**application-dev.properties**) e clique no botão **Finish**.



4. Vamos criar o segundo arquivo da mesma forma que criamos o primeiro.
5. Em **File name**, digite o nome do segundo arquivo (**application-prod.properties**) e clique no botão **Finish**.



Agora vamos configurar os 3 arquivos **.properties**:

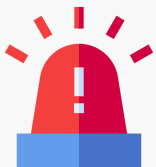
5.1 Configuração do arquivo `application.properties`

1. Abra o arquivo **application.properties**, apague **todo o conteúdo do arquivo** e substitua pelas linhas abaixo e salve o arquivo.

```
springdoc.api-docs.path=/v3/api-docs
springdoc.swagger-ui.path=/swagger-ui.html
springdoc.swagger-ui.operationsSorter=method
springdoc.swagger-ui.disable-swagger-default-url=true
springdoc.swagger-ui.use-root-path=true
springdoc.packagesToScan=com.generation.blogpessoal.controller">

spring.profiles.active=prod

springdoc.api-docs.path=/v3/api-docs
springdoc.swagger-ui.path=/swagger-ui.html
springdoc.swagger-ui.operationsSorter=method
springdoc.swagger-ui.disable-swagger-default-url=true
springdoc.swagger-ui.use-root-path=true
springdoc.packagesToScan=com.generation.blogpessoal.controller
```



ALERTA DE BSM: Mantenha a atenção aos detalhes ao configurar o arquivo `application.properties`. Cuidado para não apagar a configuração do Swagger (SpringDoc).



ALERTA DE BSM: Mantenha a atenção aos detalhes ao configurar o arquivo `application.properties`. Na última linha (`springdoc.packagesToScan`), verifique se os pacotes da sua aplicação estão configurados da mesma forma que o trecho de código acima, caso contrário, altere a configuração.

5.2 Configuração do arquivo `application-dev.properties`

1. Abra o arquivo **application-dev.properties**, insira as linhas abaixo (configuração original do **application.properties**) e salve o arquivo.
2. Não esqueça de alterar a senha do usuário root caso a sua senha do MySQL não seja root.

```
spring.jpa.show-sql=true
```

```
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL8Dialect
```



```
spring.jackson.date-format=yyyy-MM-dd HH:mm:ss
spring.jackson.time-zone=Brazil/East">

spring.jpa.hibernate.ddl-auto=update
spring.jpa.database=mysql
spring.datasource.url=jdbc:mysql://localhost/db_blogpessoal?createDatabaseIfNotExist=true&serverTimezone=America/Sao_Paulo&useSSL=false
spring.datasource.username=root
spring.datasource.password=root

spring.jpa.show-sql=true

spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL8Dialect

spring.jackson.date-format=yyyy-MM-dd HH:mm:ss
spring.jackson.time-zone=Brazil/East
```

🔗 5.3 Configuração do arquivo application-prod.properties

1. No arquivo, **application-prod.properties**, insira as linhas abaixo e salve o arquivo:

```
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.PostgreSQL95Dialect

spring.jackson.date-format=yyyy-MM-dd HH:mm:ss
spring.jackson.time-zone=Brazil/East">

spring.jpa.generate-ddl=true
spring.jpa.database=postgresql
spring.datasource.url=jdbc:postgresql://${POSTGRESHOST}:${POSTGRESPOST}/${POSTGRESDATABASE}
spring.datasource.username=${POSTGRESUSER}
spring.datasource.password=${POSTGRESPASSWORD}
spring.jpa.show-sql=true

spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.PostgreSQL95Dialect

spring.jackson.date-format=yyyy-MM-dd HH:mm:ss
spring.jackson.time-zone=Brazil/East
```

Observe no código acima que nas configurações referentes aos dados para acesso ao Banco de dados (endereço, usuário, senha, entre outros), foram inseridas **variáveis de ambiente**, que serão configuradas no Render com os dados de acesso.



ATENÇÃO: Depois de finalizar as configurações dos 3 arquivos, recomendamos executar o comando *Update Project* para atualizar as configurações do projeto.

🔗 5.4 Alternando entre os perfis no arquivo application.properties

1. Para alternar entre as configurações Local e Remota, abra o arquivo **application.properties** e utilize uma das 2 opções abaixo:

`spring.profiles.active=dev` → O Spring executará a aplicação com a configuração do Banco de dados local (MySQL)

`spring.profiles.active=prod` → O Spring executará a aplicação com a configuração do Banco de dados na nuvem (Render)

Para o Deploy, devemos deixar a linha **spring.profiles.active** configurada com a opção **prod**.



ALERTA DE BSM: Mantenha a atenção aos detalhes ao criar os perfis do Banco de Dados. Um erro muito comum é tentar executar o seu projeto no STS com o Perfil prod habilitado no arquivo *application.properties*. Com o perfil prod habilitado, o projeto não será inicializado.



ATENÇÃO: Caso o projeto seja inicializado normalmente com o perfil prod, verifique se os nomes dos arquivos de perfil e as configurações estão corretas. Se persistir, execute o comando *Update Project*.

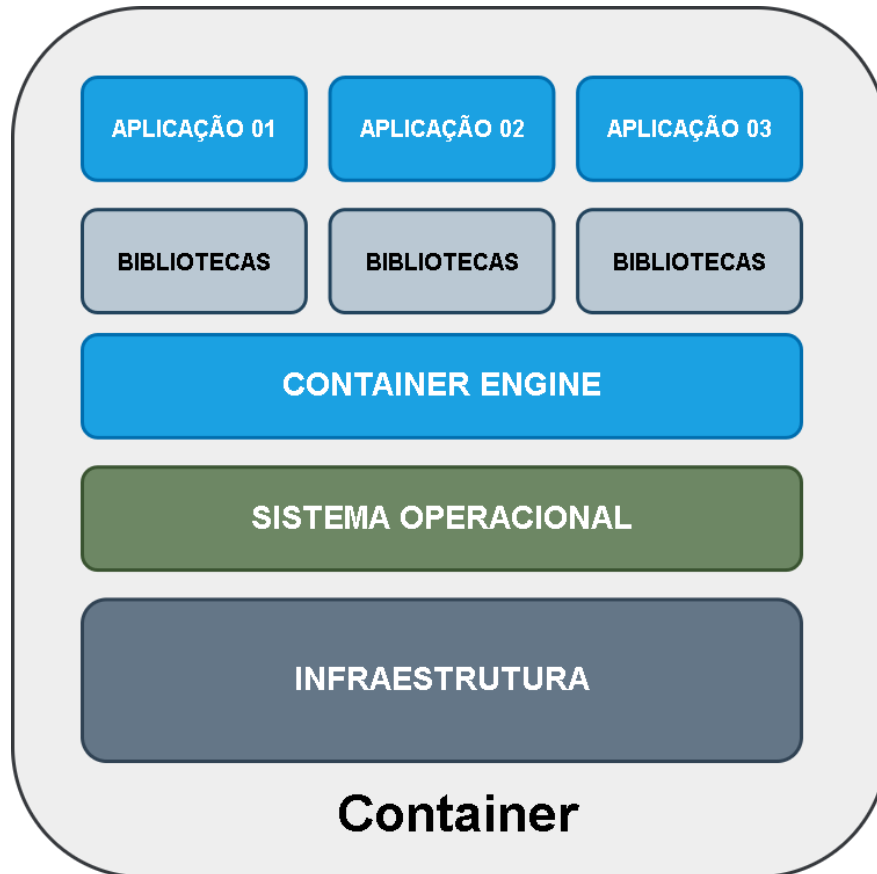
🔗 🖱️ Passo 06 - Criar o arquivo Dockerfile

O Render não possui nativamente o suporte ao Java. Como opção para fazer o Deploy de aplicações Java, o Render oferece a possibilidade de utilizar o **Docker**, sem a necessidade de instalar o Docker Desktop no computador.

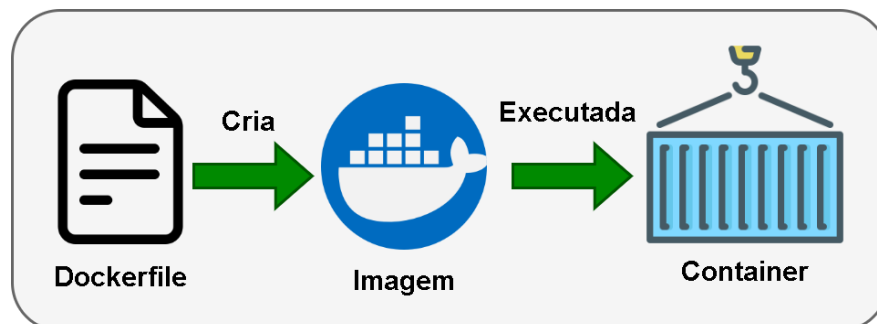
O **Docker** é um projeto para automatizar a implantação de aplicativos como **Containers** autossuficientes e portáteis, que podem ser executados na nuvem ou localmente, a partir de **imagens** contendo o ambiente mínimo para executar uma aplicação.

Uma **Imagem** nada mais é do que uma **representação imutável** de como será efetivamente construído um container. **Exemplo:** uma imagem contendo o Ubuntu Linux, uma imagem contendo o Java 17, entre outros.

Como uma imagem é imutável, nós não podemos executar a imagem de forma direta, nós fazemos isso através de um **Container**, que é uma tecnologia que permite empacotar e isolar aplicações com o seu ambiente de tempo de execução, ou seja, com todos os arquivos necessários para executar.



O ponto principal que precisamos compreender para efetuarmos o Deploy da nossa aplicação no Render é que escrevemos um **Dockerfile**, construímos uma imagem a partir do Dockerfile, e por fim, criamos e executamos o container. **O container é o fim enquanto a imagem é o meio.**



O **Dockerfile** nada mais é do que **um meio que utilizamos para criar nossas próprias imagens**. Em outras palavras, ele serve como uma **receita para construir um container**, permitindo definir um ambiente personalizado e próprio para meu projeto pessoal ou empresarial.

No Projeto Blog Pessoal nós iremos **construir o Dockerfile e o Render se encarregará de construir a Imagem e o Container com o ambiente necessário para executar o projeto Blog Pessoal**.

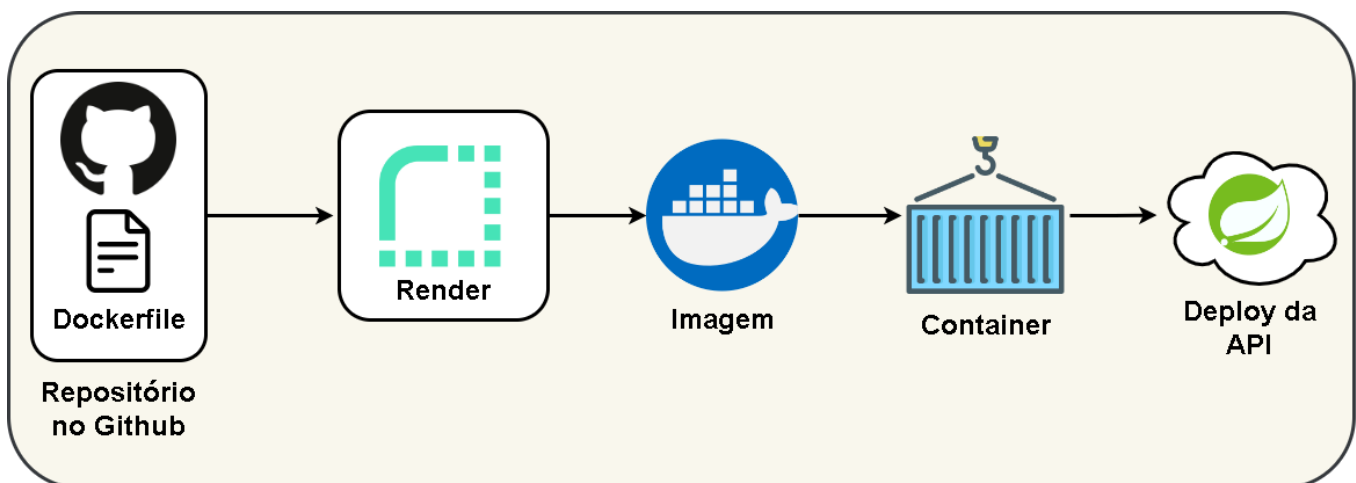


ATENÇÃO: O nosso foco não é estudar o Docker. O foco neste momento é compreender como criaremos o Dockerfile necessário para efetuarmos o Deploy da nossa aplicação no Render.

Veja na tabela abaixo os comandos do **Dockerfile** que vamos utilizar:

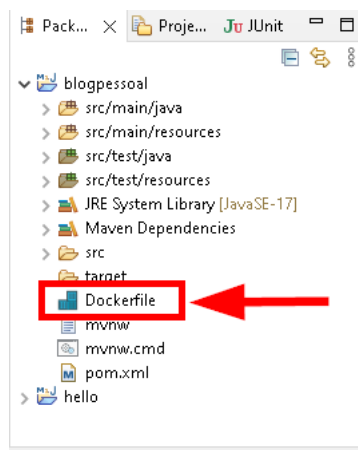
Comandos	Descrição
FROM	Deve ser o primeiro comando de um Dockerfile, com ele configuramos qual a imagem que queremos utilizar como base na nossa imagem. Podemos criar uma imagem a partir de uma imagem do Java ou de um sistema Linux e assim por diante.
RUN	Serve para executar comandos no processo de montagem da imagem que estamos construindo no Dockerfile, ele é executado durante o <i>build</i> (construção da imagem) e não durante a construção do container. Em um Dockerfile é possível ter mais de um comando RUN .
COPY	Comando para copiar arquivos e pastas de um lugar específico na máquina local para uma pasta no container. O COPY é utilizado para copiar arquivos e pastas locais.
VOLUME	Quando adicionamos VOLUME no Dockerfile estamos informando um ponto de montagem, criando uma pasta que ficará disponível entre o <i>container</i> e o <i>host</i> .
WORKDIR	Define a pasta dentro do container onde os comandos serão executados.
ENTRYPOINT	É um comando que não pode ser sobrescrito, ele sempre será executado e o container irá rodar como um executável. Quando este comando "morrer" (finalizar), o container morrerá também.
ARG	A instrução ARG define uma variável que os usuários podem passar em tempo de compilação do container. Se um usuário especificar um argumento de compilação que não foi definido no Dockerfile, a compilação gerará um aviso.

O Render ao receber o Dockerfile através do repositório do Github, vai gerar a Imagem e o Container automaticamente, e caso não aconteça nenhum erro, a sua aplicação será implantada na nuvem e o Render lhe oferecerá um endereço (<https://meuprojeto.onrender.com>) para acessar externamente. Veja um resumo do processo na imagem abaixo:



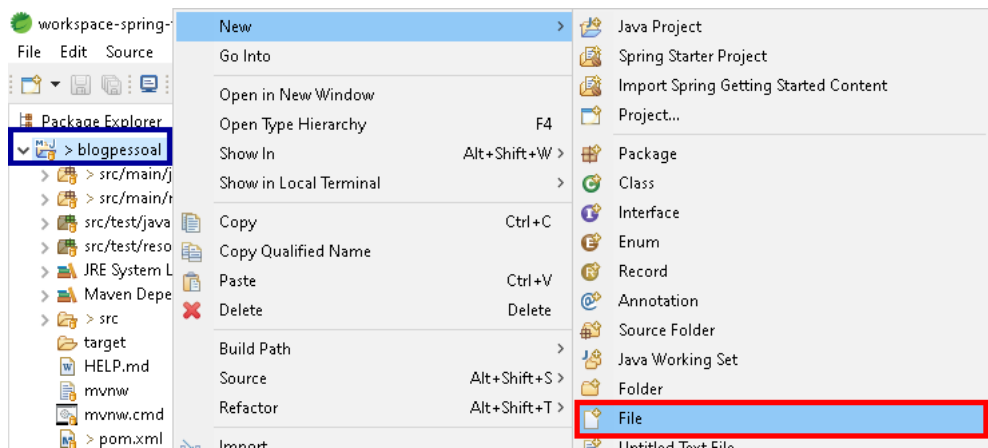
Vamos criar o arquivo **Dockerfile** no projeto Blog Pessoal:

1. Na **raiz do seu projeto**, na pasta **blogpessoal** (como mostra a figura abaixo), crie o arquivo **Dockerfile**.

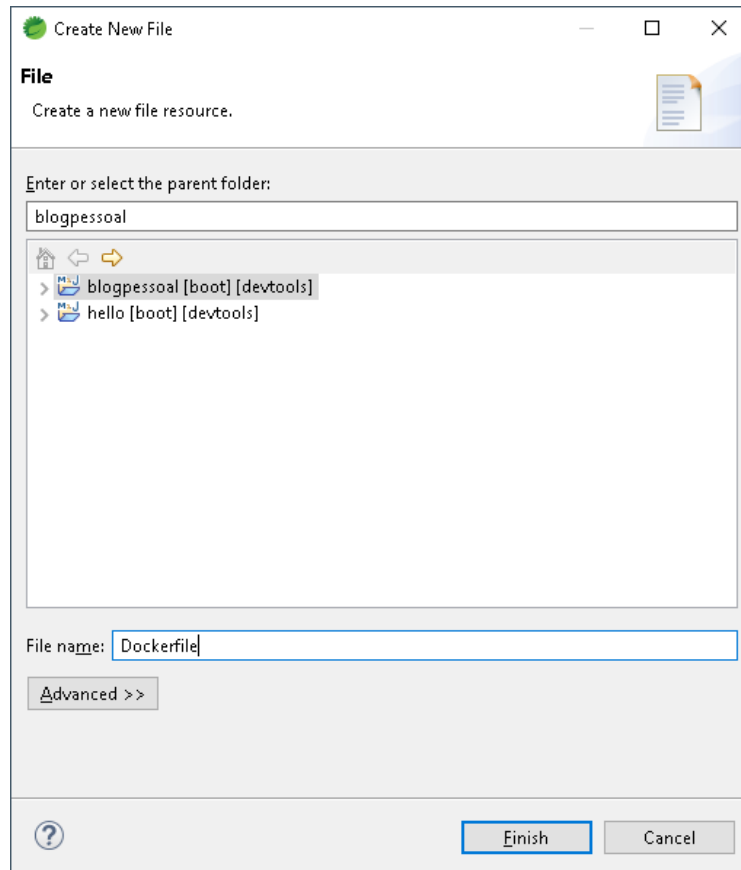


ALERTA DE BSM: Mantenha a atenção aos detalhes ao criar o arquivo Dockerfile. Um erro muito comum é não criar o arquivo na pasta raiz do projeto. Outro erro comum é digitar o nome do arquivo com letras minúsculas. Observe que o arquivo inicia com letra maiúscula

2. Na Guia **Package Explorer**, clique com o botão direito do mouse sobre a pasta do projeto (indicada em azul) e clique na opção **New** → **File**.



3. Em **File name**, digite: **Dockerfile** e clique no botão **Finish**.



4. Copie o código abaixo dentro do arquivo **Dockerfile** que você criou:

```
WORKDIR /workspace/app

COPY mvnw .
COPY .mvn .mvn
COPY pom.xml .
COPY src src

RUN chmod -R 777 ./mvnw

RUN ./mvnw install -DskipTests

RUN mkdir -p target/dependency && (cd target/dependency; jar -xf ../*.jar)

FROM openjdk:17.0.1-jdk-oracle

VOLUME /tmp

ARG DEPENDENCY=/workspace/app/target/dependency

COPY --from=build ${DEPENDENCY}/BOOT-INF/lib /app/lib
COPY --from=build ${DEPENDENCY}/META-INF /app/META-INF
COPY --from=build ${DEPENDENCY}/BOOT-INF/classes /app

ENTRYPOINT ["java", "-cp", "app:app/lib/*;", "com.generation.blogpessoal.BlogpessoalApplication"]>
```

```

FROM openjdk:17.0.1-jdk-oracle as build

WORKDIR /workspace/app

COPY mvnw .
COPY .mvn .mvn
COPY pom.xml .
COPY src src

RUN chmod -R 777 ./mvnw

RUN ./mvnw install -DskipTests

RUN mkdir -p target/dependency && (cd target/dependency; jar -xf ../*.jar)

FROM openjdk:17.0.1-jdk-oracle

VOLUME /tmp

ARG DEPENDENCY=/workspace/app/target/dependency

COPY --from=build ${DEPENDENCY}/BOOT-INF/lib /app/lib
COPY --from=build ${DEPENDENCY}/META-INF /app/META-INF
COPY --from=build ${DEPENDENCY}/BOOT-INF/classes /app

ENTRYPOINT ["java", "-cp", "app:app/lib/*", "com.generation.blogpessoal.BlogpessoalApplication"]

```

Vamos entender o código:

```

1 FROM openjdk:17.0.1-jdk-oracle as build
2
3 WORKDIR /workspace/app
4
5 COPY mvnw .
6 COPY .mvn .mvn
7 COPY pom.xml .
8 COPY src src
9
10 RUN chmod -R 777 ./mvnw
11
12 RUN ./mvnw install -DskipTests
13
14 RUN mkdir -p target/dependency && (cd target/dependency; jar -xf ../*.jar)
15
16 FROM openjdk:17.0.1-jdk-oracle
17
18 VOLUME /tmp
19
20 ARG DEPENDENCY=/workspace/app/target/dependency
21
22 COPY --from=build ${DEPENDENCY}/BOOT-INF/lib /app/lib
23 COPY --from=build ${DEPENDENCY}/META-INF /app/META-INF
24 COPY --from=build ${DEPENDENCY}/BOOT-INF/classes /app
25
26 ENTRYPOINT ["java", "-cp", "app:app/lib/*", "com.generation.blogpessoal.BlogpessoalApplication"]

```

A Criação do Container é dividida em 2 estágios:

1. Criar o Build da aplicação, ou seja, gerar o arquivo JAR.
2. Executar a Aplicação dentro do Container

Vamos detalhar as etapas:

Etapas 01 - Build da aplicação

Linha 1: Estamos indicando que o Build da aplicação será gerado a partir de uma imagem contendo a versão 17 do Java, rodando sobre o Linux numa versão minimalista, ou seja, apenas o necessário para executar o Java 17. Neste momento será feito o download da imagem contendo o Java 17. Observe que no final do nome da imagem, temos o alias **as build**, indicando que o Java 17 será executado nesta etapa em modo Build.

Linha 3: Estamos indicando que dentro do nosso container, o Build da nossa aplicação será gerado dentro da pasta **/workspace/app**.

Linha 5 a 8: Copia o projeto do **Repositório do Github** para a pasta **/workspace/app**.

Linha 10: Autoriza a execução do **Maven** (mvnw) dentro da pasta **/workspace/app**. Sem este comando, o Render devolverá o erro: **Permission Denied! (Permissão negada)**

Linha 12: Gera o Build da nossa aplicação, excluindo os testes, porquê em produção eles são desnecessários. Neste momento será feito o download de todas as dependências do projeto do Repositório central do Maven e o arquivo JAR será criado.

Linha 14: Cria a pasta **target/dependency**, e na sequência descompacta o arquivo .JAR que foi gerado. Ao descompactar o JAR, está sendo extraída uma versão compilada do código, ou seja, pronta para executar.

Etapas 02 - Executar aplicação

Linha 16: Indica que a nossa aplicação será executada através da imagem da versão 17 do Java, que já foi obtida via download.

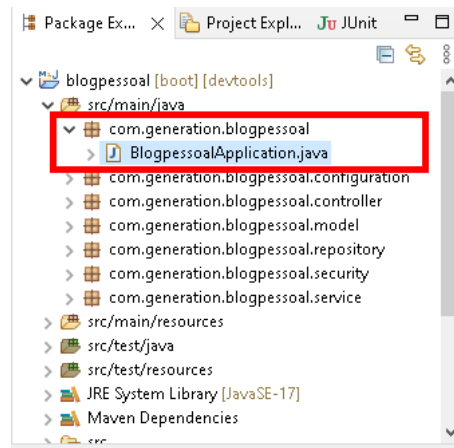
Linha 18: Cria um volume chamado **/tmp** para armazenar os arquivos temporários.

Linha 20: Indica onde os arquivos e pastas do Build do projeto serão gravados.

Linha 22 a 24: Copia todos os arquivos e pastas gerados no Build da aplicação, na pasta `/workspace/app/target/dependency`, dentro da pasta `/workspace/app`.

Linha 26: Executa o projeto Blog Pessoal.

Importante: A instrução `"com.generation.blogpessoal.BlogpessoalApplication"` deve ser igual ao caminho da Classe Main da aplicação, como mostra a imagem abaixo:



Observe na imagem acima que:

- `com.generation.blogpessoal` → Pacote principal da aplicação.
- `BlogpessoalApplication` → Nome da Classe que contém o método `main()`.

Salve o arquivo antes de continuar.



[Documentação: Dockerfile](#)

[Documentação: Maven Wrapper - MVNW](#)

[Documentação: Linux - CHMOD](#)



[Código fonte: Projeto finalizado](#)



Passo 07 - Atualizar o repositório do projeto no Github

1.

Envie as atualizações do seu projeto para o repositório do Github, através do **Git Bash**, utilizando os comandos abaixo:

```
git commit -m "Deploy do Projeto Blog Pessoal"
```

```
git push origin main">
```

```
git add .
```

```
git commit -m "Deploy do Projeto Blog Pessoal"
```

```
git push origin main
```

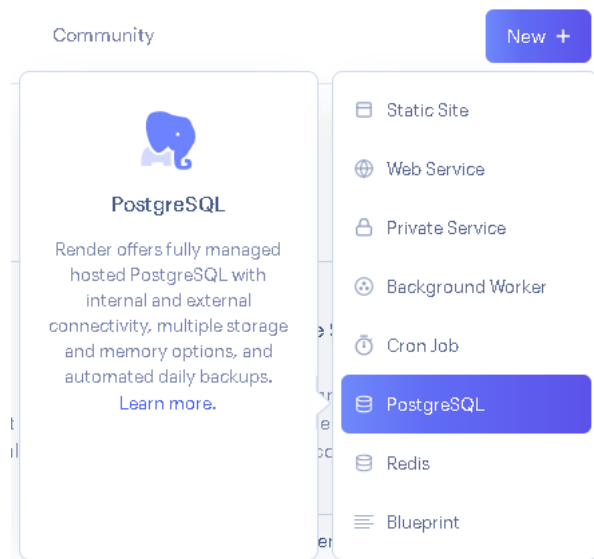


ATENÇÃO: Para efetuar o Deploy, o projeto Spring **OBRIGATORIAMENTE** precisa estar em um Repositório **EXCLUSIVO** e não pode estar **DENTRO DE UMA PASTA**, ou seja, ao abrir o repositório do projeto no Github, o conteúdo exibido será semelhante ao da imagem abaixo. Se estiver diferente da imagem abaixo será necessário refazer o Repositório do Github.

<div> main 1 branch 0 tags </div> <div> <div>Go to file</div> <div>Add file</div> <div>Code</div> </div>		
<div> rafaelq80 Deploy <div>93be532 7 minutes ago 2 commits</div> </div>		
.mvn/wrapper	Deploy	7 minutes ago
src	Deploy	7 minutes ago
.gitignore	Deploy	7 minutes ago
README.md	Initial commit	8 minutes ago
mvnw	Deploy	7 minutes ago
mvnw.cmd	Deploy	7 minutes ago
pom.xml	Deploy	7 minutes ago
system.properties	Deploy	7 minutes ago

Passo 08 - Criar o Banco de dados no Render

1. Para adicionar um novo Banco de dados, no Dashboard do Render, clique no botão **New +** e em seguida clique na opção **PostgreSQL**.



ATENÇÃO: *O Plano Gratuito do Render autoriza a criação de apenas 1 Banco de dados PostgreSQL por conta. Outro ponto importante é que em 90 dias o Banco será apagado (Drop Database).

2. Na próxima tela, informe o nome do Banco de dados (**db_blogpesoal**), na propriedade **Name**, como mostra a imagem abaixo:

New PostgreSQL

Name ⓘ

Database ⓘ

User

Region ⓘ
The region where your Database runs.

PostgreSQL Version

Datadog API Key ⓘ

Please [enter your payment information](#) to select a plan with higher limits.



3. Role a tela para baixo e verifique se o Plano Gratuito (**Free**) está selecionado e na sequência, clique no botão **Create Database**, como mostra a imagem abaixo:

Current	Starter	Standard	Standard Plus	Pro	Pro Plus
Free RAM 256 MB CPU shared STORAGE 1 GB \$0/month	RAM 256 MB CPU shared STORAGE 1 GB \$7/month	RAM 1 GB CPU 1 STORAGE 16 GB \$20/month	RAM 2 GB CPU 1 STORAGE 48 GB \$45/month	RAM 4 GB CPU 2 STORAGE 96 GB \$95/month	RAM 8 GB CPU 4 STORAGE 256 GB \$185/month

Free plan databases will expire in 90 days and will be deleted if not upgraded. [Learn more about free plan limits.](#)

Need a custom plan? We support up to 512 GB RAM, 64 CPUs, and 5 TB storage. Contact us at sales@render.com.

Create Database



ATENÇÃO: *Caso seja selecionado um plano diferente, o Render exigirá o Cartão de Crédito para emitir a fatura do serviço.

4. Na próxima tela, aguarde até que o **Status** esteja **Available**, como mostra a imagem abaixo:

General

Name db_blogpessoal [Edit](#)

Created a minute ago

Expiration December 8, 2022 ⓘ

Status Available

PostgreSQL Version 14

Region Oregon (US West)



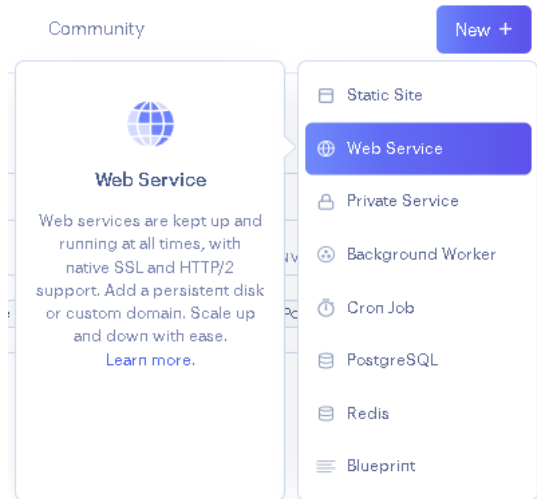
ATENÇÃO: *O processo do Deploy enviará apenas a sua aplicação para a nuvem, logo o Banco de dados que será criado nesta etapa estará vazio.

Passo 09 - Criar o Web Service no Render

1. Na barra de menus principal do Render, clique no item Dashboard, como mostra a imagem abaixo:



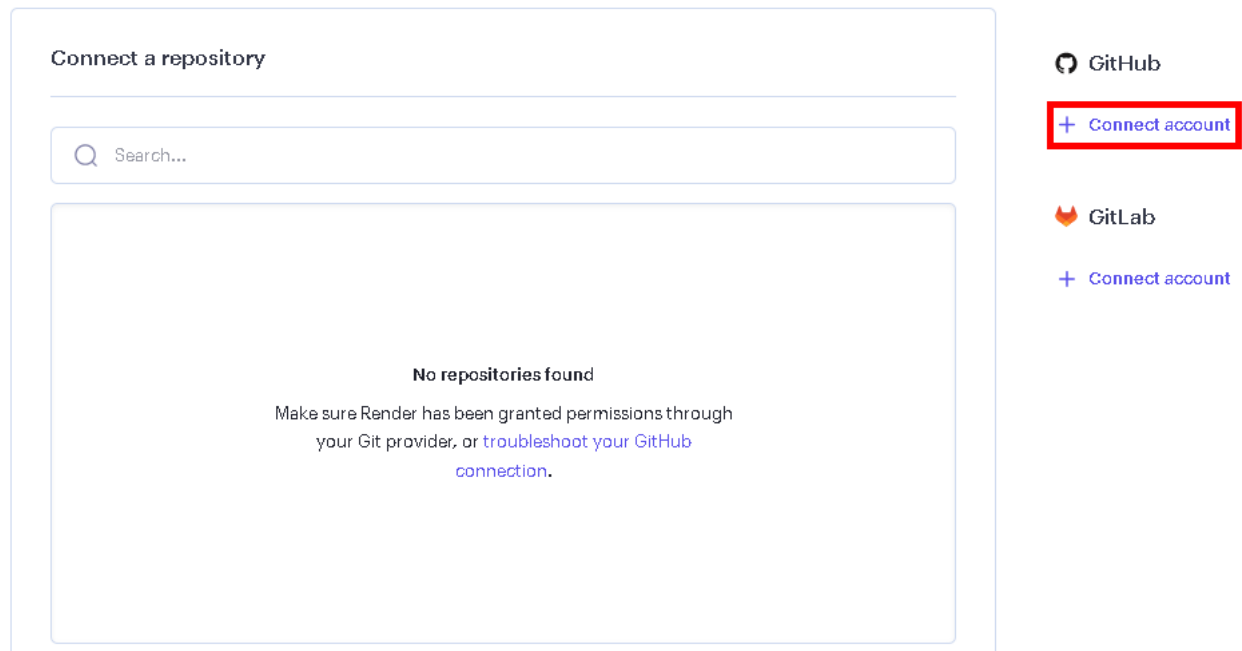
2. Para adicionar um novo Web Service, no Dashboard do Render, clique no botão **New +** e em seguida clique na opção **Web Service**.



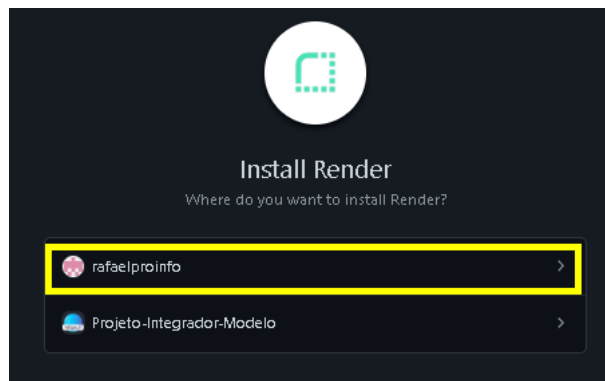
3. No item **GitHub**, clique no link **+ Connect account**, para conectar a sua conta do Render com a sua Conta do Github.

Create a new Web Service

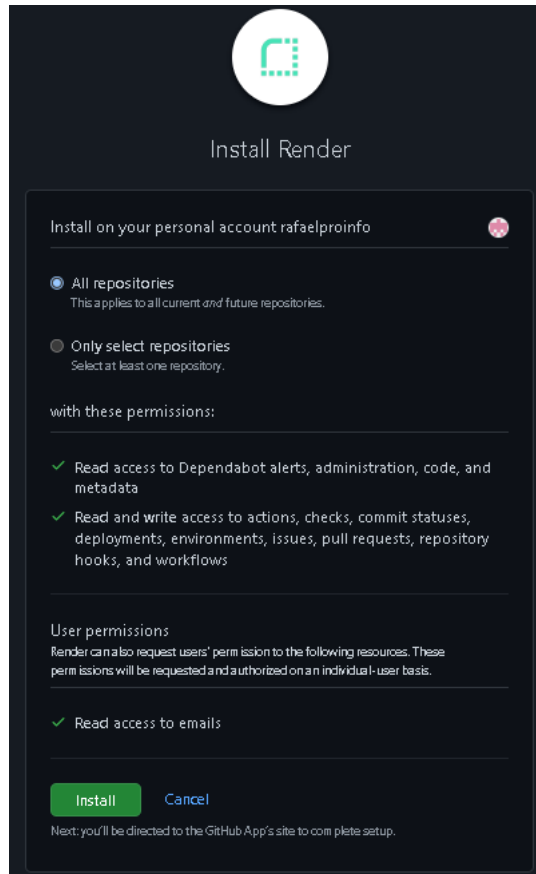
Connect your Git repository or use an existing public repository URL.



4. Na tela, **Install Render**, clique no seu usuário do Github (no exemplo, rafaelproinfo), como mostra a figura abaixo:



5. Na próxima tela, clique no botão **Install**, para concordar que o Render acesse a sua Conta do Github.




6. Conecte o Render com o Repositório onde você enviou o Blog Pessoal, clicando no botão **Connect**, localizado ao lado do Repositório.


Create a new Web Service

Connect your Git repository or use an existing public repository URL.


Connect a repository

 rafaelprounfo / blogpessoal • 5 months ago


Connect

 rafaelprounfo / helloworld • 7 months ago

Connect

 rafaelprounfo / projeto_integrador • 8 months ago

Connect

 rafaelprounfo / spring_react • 10 months ago

Connect

7. Na próxima tela, informe o nome da sua aplicação na propriedade **Name** (blogpessoal) e verifique se a propriedade **Environment** está com a opção **Docker** selecionada.

You seem to be using **Docker**, so we've autofilled some fields accordingly. Make sure the values look right to you!

Name

A unique name for your web service.

blogpessoal

Environment

The runtime environment for your web service.

Docker

Region

The **region** where your web service runs. Services must be in the same region to communicate privately and you currently have services running in **Oregon**.

Oregon (US West)

Branch

The repository branch used for your web service.

main



ATENÇÃO: O NOME DO PROJETO NÃO PODE CONTER LETRAS MAIUSCULAS, NUMEROS OU CARACTERES ESPECIAIS.

8. Role a tela para baixo e verifique se o Plano Gratuito (**Free**) está selecionado.

Plans

Please [enter your payment information](#) to select a plan with higher limits.

Current

Free

RAM 512 MB
CPU shared

\$0/month

Starter

RAM 512 MB
CPU 0.5

\$7/month

Starter Plus

RAM 1 GB
CPU 1

\$15/month

Standard

RAM 2 GB
CPU 1

\$25/month

Standard Plus

RAM 3 GB
CPU 1.5

\$50/month

Pro

RAM 4 GB
CPU 2

\$85/month

Pro Plus

RAM 8 GB
CPU 4

\$175/month

Pro Max

RAM 16 GB
CPU 4

\$225/month

Pro Ultra

RAM 32 GB
CPU 8

\$450/month



ATENÇÃO: *Caso seja selecionado um plano diferente, o Render exigirá o Cartão de Crédito para emitir a fatura do serviço.

Passo 10 - Configurar as Variáveis de Ambiente

Nesta etapa, serão criadas 5 variáveis de ambiente, como mostra a imagem abaixo:

Environment Variables

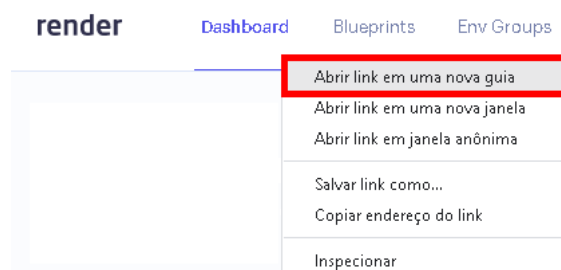
Use environment variables to store API keys and other configuration values and secrets. You can access them in your code like regular environment variables, for example with `os.getenv()` in Python or `process.env` in Node.

Key	Value	
POSTGRESDATABASE	
POSTGRESHOST	
POSTGRESPASSWORD	
POSTGRESPOST	
POSTGRESUSER	

[Create Environment Group](#)[Add Environment Variable](#)[Save Changes](#)

Antes de criarmos as variáveis, precisamos identificar os dados que serão inseridos nestas variáveis, que estão disponíveis no **Banco de dados** criado no passo anterior.

1. Na **Barra Principal** do Render, clique com o botão direito mouse no item **Dashboard** e clique na opção **Abrir link em uma Nova Guia**, como mostra a imagem abaixo:



2. Na nova guia que foi aberta, clique no Banco de dados **db_blogpessoal**, como mostra a imagem abaixo:

Overview

Search services						
NAME	STATUS	TYPE	ENVIRONMENT	REGION	LAST DEPLOYED	
db_blogpessoal	Available	PostgreSQL	PostgreSQL 14	Oregon	5 days ago	

3. Role a tela para baixo e localize o item: **Connections**, como mostra a imagem abaixo:

Connections

Hostname	dpg-ccdmqr6n6mpu7e89atrg-a
Port	5432
Database	db_blogpessoal
Username	db_blogpessoal_user
Password	<input type="password"/>
Internal Database URL	<input type="password"/>
External Database URL	<input type="password"/>
PSQL Command	<input type="password"/>

4. Na imagem abaixo, está indicada qual variável receberá cada item da configuração do Banco de dados na nuvem:

Connections

Hostname	dpg-ccdmqr6n6mpu7e89atrg-a	POSTGRESHOST
Port	5432	POSTGRESPOST
Database	db_blogpessoal	POSTGRESDATABASE
Username	db_blogpessoal_user	POSTGRESUSER
Password	<input type="password"/>	POSTGRESPASSWORD

5. Ao criar a variável **POSTGRESHOST**, acrescente no final do **Hostname**: **.oregon-postgres.render.com**. Caso contrário o Sprinh não conectará com o banco de dados.

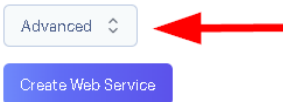
6. Nas demais variáveis, utilize os mesmos dados que estão na tela do Banco de dados.

7. No exemplo acima, a configuração ficou da seguinte forma:

Variável	Conteúdo
POSTGRESHOST	dpg-ccdmqr6n6mpu7e89atrg-a.oregon-postgres.render.com
POSTGRESPOST	5432
POSTGRESDATABASE	db_blogpessoal
POSTGRESUSER	db_blogpessoal_user
POSTGRESPASSWORD	senha do banco de dados

Agora vamos criar as variáveis:

1. Volte para a Guia onde o **Web Service** está sendo criado e role a tela para baixo e clique no botão **Advanced**, como mostra a imagem abaixo:



2. Na sequência, clique no botão **Add Environment Variable**

Use environment variables to store API keys and other configuration values and secrets. You can access them in your code like regular environment variables, for example with `os.getenv()` in Python or `process.env` in Node.



3. No item **Key**, informe o nome da 1ª variável: **POSTGRESHOST**

Environment Variables

Use environment variables to store API keys and other configuration values and secrets. You can access them in your code like regular environment variables, for example with `os.getenv()` in Python or `process.env` in Node.

Key	Value
<input type="text" value="POSTGRESHOST"/>	<input type="text" value="value"/> Generate

4. Cole no item **Value** da Variável de Ambiente **POSTGRESHOST** o Hostname do Banco de dados, conforme detalhado nos itens acima, como mostra a imagem abaixo:

Environment Variables

Use environment variables to store API keys and other configuration values and secrets. You can access them in your code like regular environment variables, for example with `os.getenv()` in Python or `process.env` in Node.

Key	Value
<input type="text" value="POSTGRESHOST"/>	<input type="text" value="dpg-ccdmqr6n6mpu7e89atrg-a.oregon-postgres.render.com"/>

5. Repita os passos 2, 3 e 4 para criar as demais variáveis.

6. Verifique se a propriedade **Auto-Deploy** está com a opção **Yes** selecionada

Auto-Deploy

Automatic deploy on every push to your repository or changes to your service? Select "No" to handle your deploys manually.

7. Clique no botão **Create Web Service** para criar o Web Service e iniciar o Deploy.



8. Acompanhe o processo do Deploy no **Log do Web Service**, como mostra a imagem abaixo:

Builds too slow? Upgrade to a paid plan to go faster. Learn more about free plan limits.

September 14, 2022 at 4:47 PM *** In progress

67a7a92 Deploy inicial

Search logs

Search

Maximize

Scroll to top

```
Sep 14 04:47:28 PM ==> Checking out commit 67a7a925de4326c7f9e9319889e60e2a626043c0 in branch main
Sep 14 04:47:33 PM ==> Downloading cache...
Sep 14 04:47:59 PM #1 [internal] load .dockignore
Sep 14 04:47:59 PM #1 transferring context: 2B done
Sep 14 04:47:59 PM #1 DONE 0.1s
Sep 14 04:47:59 PM
Sep 14 04:47:59 PM #2 [internal] load build definition from Dockerfile
Sep 14 04:47:59 PM #2 transferring dockerfile: 644B done
Sep 14 04:47:59 PM #2 DONE 0.1s
Sep 14 04:47:59 PM
Sep 14 04:47:59 PM #3 [internal] load metadata for docker.io/library/openjdk:17.0.1-jdk-oracle
Sep 14 04:48:00 PM #3 ...
```

9. Ao finalizar a criação da imagem e do Container Docker, será exibida a mensagem: **DONE**, como mostra a imagem abaixo e na sequência a aplicação será iniciada.

```
Sep 14 04:25:02 PM #18 exporting layers 3.0s done
Sep 14 04:25:02 PM #18 exporting manifest sha256:4f612ae30c6237dce99c0967f024ec3ccfe019fe92645edf81327ab86f445414 0.0s done
Sep 14 04:25:02 PM #18 exporting config sha256:56d1b359b596fe76a81256751baaf6c17e2f3692243c3884aae5d56fe2fafe8d 0.0s done
Sep 14 04:25:08 PM #18 DONE 9.0s
Sep 14 04:25:08 PM
Sep 14 04:25:08 PM #19 exporting cache
Sep 14 04:25:08 PM #19 preparing build cache for export
Sep 14 04:25:38 PM #19 DONE 29.6s
Sep 14 04:26:02 PM Pushing image to registry...
Sep 14 04:26:10 PM DONE
```

10. Observe na imagem abaixo, que caso a aplicação inicialize sem erros, será exibida a mensagem informando que a aplicação está em execução. A mensagem é a mesma que é exibida no **Console do STS**.

```
tyContextPersistenceFilter@18ad085a, org.springframework.security.web.header.HeaderWriterFilter@147c4523, org.springframework.web.filter.CorsFilter@37ad818e, org.springframework.security.web.authentication.logout.LogoutFilter@1390a415, org.springframework.security.web.authentication.www.BasicAuthenticationFilter@91c11b6, org.springframework.security.web.savedrequest.RequestCacheAwareFilter@7f3a92fd, org.springframework.security.web.servletapi.SecurityContextHolderAwareRequestFilter@504274c1, org.springframework.security.web.authentication.AnonymousAuthenticationFilter@41271320, org.springframework.security.web.session.SessionManagementFilter@17635531, org.springframework.security.web.access.ExceptionTranslationFilter@14d6ede3, org.springframework.security.web.access.intercept.AuthorizationFilter@495e1ad1]
Sep 14 04:28:51 PM 2022-09-14 19:28:51.967 INFO 1 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
Sep 14 04:28:52 PM 2022-09-14 19:28:52.251 INFO 1 --- [main] c.g.blogpessoal.BlogpessoalApplication : Started BlogpessoalApplication in 139.801 seconds (JVM running for 147.988)
```

11. Para confirmar que o Deploy foi concluído com êxito, verifique na parte superior da tela de **Log do Webservice**, se apareceu a mensagem **Live**, como mostra a figura abaixo:

September 19, 2022 at 4:19 PM

Live

367f207 Deploy

12. Se apareceu esta mensagem, o Deploy foi finalizado com sucesso!

13. Para abrir a aplicação no Navegador da Internet, clique no link da aplicação, indicado na imagem abaixo:

WEB SERVICE



Docker

Free Plan

 rafaelproinfo/blogpessoal 

<https://blogpessoal.onrender.com> 



ATENÇÃO: Ao clicar no link da aplicação, o projeto não abrirá automaticamente no navegador. Como o Docker precisa finalizar alguns processos, ele pode demorar alguns minutos para abrir. O tempo médio do processo de deploy completo do Blog Pessoal pode demorar um pouco.



ALERTA DE BSM: Mantenha a atenção aos detalhes. Caso o nome do projeto já seja um endereço em uso no Render, ele acrescentará caracteres aleatórios depois do nome do projeto ao criar o endereço da aplicação. Exemplo: `blogpessoal-wrtc.onrender.com`.

O Deploy não Abriu!

Se o **Deploy no Render não abrir** e/ou aparecer a mensagem **Failed**, como mostra a figura abaixo:

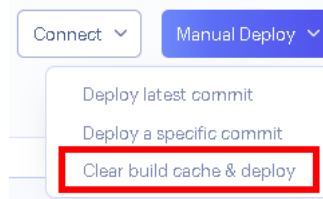
September 19, 2022 at 4:10 PM

Failed

Rollback to this deploy

367f207 Deploy

- Caso não tenha aparecido mensagens de erro no **Console**, na tela **Log do Webservice** (O projeto Spring inicializou, mas o deploy falhou), clique no botão **Manual Deploy** e em seguida, clique na opção **Clear build cache & deploy** para refazer o deploy, como mostra a imagem abaixo:



- Se o erro persistir, verifique se as **Variáveis de Ambiente estão configuradas corretamente**. Caso seja necessário atualizar o valor de qualquer Variável de ambiente, o Render iniciará um novo Deploy automaticamente após a atualização;
- Caso tenha aparecido algum erro no **Log do Web Service**, identifique o tipo do erro:
 - Se for erro no **Dockerfile**, corrija o erro e atualize o seu repositório no Github;
 - Se for erro de código (erro do Spring), siga as instruções do tópico: [3. Como atualizar o Deploy no Render?](#) e corrija o erro.

Passo 11 - Abrir o Deploy no Navegador

- Ao abrir a sua aplicação no Navegador, será exibida a tela de login abaixo. Como o Banco de dados criado no Render está vazio, precisamos criar uma conta de usuário e efetuar o login com esta conta antes de exibir a sua documentação no Swagger.

Fazer login

<https://blogpessoal.onrender.com>

Nome de usuário

Senha

- Abra o **Insomnia** e acesse a Workspace **Blog Pessoal**.
- Crie uma pasta chamada **Blog Pessoal** e arraste as 3 pastas (Postagem, Tema e Usuario) para dentro dela.
-

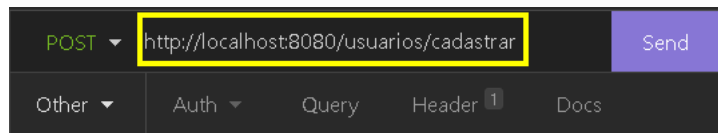
Duplicate a pasta **Blog Pessoal**.

5. Na próxima janela, defina o nome da nova pasta como **Blog Pessoal - Render**.

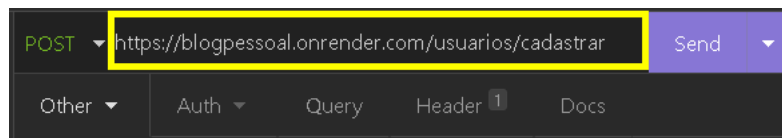


6. Abra a requisição **Cadastrar Usuário** na pasta **Blog Pessoal - Render**.

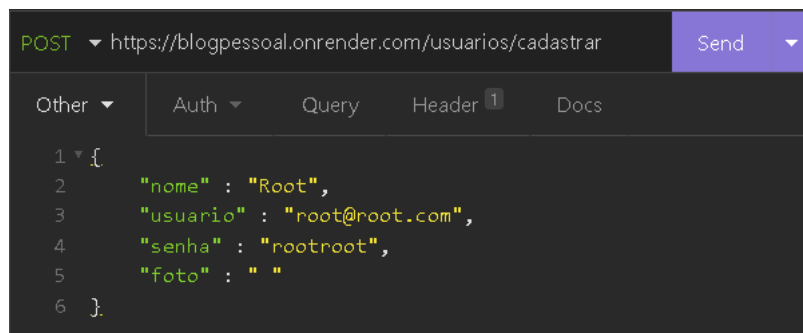
7. Altere o caminho atual: <http://localhost:8080/usuarios/cadastrar>



8. Para o endereço do Render: <https://meuprojeto.render.com/usuarios/cadastrar> (No exemplo acima: <https://blogpessoal.onrender.com/usuarios/cadastrar>)

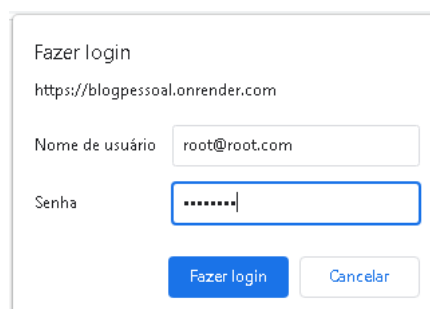


9. Após efetuar as alterações, crie o usuário root@root.com com os dados da imagem abaixo:



ALERTA DE BSM: Mantenha a atenção aos detalhes. Crie o usuário root exatamente como mostra a figura acima. Será através deste usuário que os Instrutores da sua turma irão corrigir o seu projeto.

10. Volte para o Navegador da Internet e efetue o login com o usuário root@root.com.



11. A Documentação no Swagger será exibida como a página inicial.

 Swagger
powered by SMARTBLAR

v3/api-docs

Explore

Projeto Blog Pessoal v0.0.1 OAS3

v3/api-docs

Projeto Blog Pessoal - Generation Brasil

[Conteúdo Generation - Website](#)
[Send email to Conteúdo Generation](#)
[Generation Brasil](#)
[Github](#)

Servers

https://blogpessoal.onrender.com - Generated server url


usuario-controller

GET	/usuarios/{id}	▼
GET	/usuarios/all	▼
POST	/usuarios/login	▼
POST	/usuarios/cadastrar	▼

12. Outra forma de abrir a aplicação é digitando o endereço: <https://nomedoprojeto.onrender.com> no navegador. (No exemplo acima: <https://blogpessoal.onrender.com>)


Passo 12 - Testar o Deploy no Insomnia

1. Volte para o **Insomnia**
2. Atualize o endereço de todas requisições da pasta **Blog Pessoal - Render**, assim como foi feito na requisição **Cadastrar Usuário**
3. Execute a requisição **Login** para acessar a API
4. Teste todas requisições seguindo as orientações do **Checklist do Projeto Blog Pessoal**.



ALERTA DE BSM: *Mantenha a atenção aos detalhes e a persistência. Insira dados na API através do Insomnia em todos os recursos (Postagem, Tema e Usuário). No recurso Postagem, não esqueça de testar o Relacionamento entre as Classes.*

3. Como atualizar o Deploy no Render?



ALERTA DE BSM: *Mantenha a atenção aos detalhes. Este item você utilizará apenas se você precisar alterar alguma coisa no seu projeto Spring e atualizar a aplicação na nuvem.*



1. Para fazer alterações no código do projeto e executar localmente, volte para o **STS** e altere a primeira linha do arquivo, **application.properties** conforme o código abaixo:
`spring.profiles.active=dev`
2. Faça as alterações necessárias no código do seu projeto, execute localmente e verifique se está tudo funcionando **sem erros**.
3. Antes de refazer o Deploy, altere novamente a primeira linha do arquivo, **application.properties** conforme o código abaixo:
`spring.profiles.active=prod`
4. Envie as atualizações do seu projeto para o repositório do Github, através do **Git Bash**, utilizando os comandos abaixo:

```
git add .
git commit -m "Atualização do Deploy - Blog Pessoal"
git push origin main
```

5. Ao finalizar o **git push**, o Render começará a refazer o Deploy. Acompanhe o processo pelo **Dashboard do Render**.

Overview

Q Search services

NAME	STATUS	TYPE	ENVIRONMENT	REGION	▼ LAST DEPLOYED
 blogpessoal 	*** Deploy in progress	Web Service	Docker	Oregon	a few seconds ago

6. Verifique se a Aplicação abre no Navegador e faça os testes no Insomnia.



```
</div>
```



```
</main>
```

```
<footer class="footer width-full container-xl p-responsive">
```

Footer



```
<nav aria-label='footer' class="col-12 col-lg-8">
  <h3 class='sr-only' id='sr-footer-heading'>Footer navigation</h3>
  <ul class="list-style-none d-flex flex-wrap col-12 flex-justify-center flex-lg-justify-between mb-2 mb-lg-0" aria-labelledby='sr-footer-heading'>
    <li class="mr-3 mr-lg-0"><a href="https://docs.github.com/en/github/site-policy/github-terms-of-service" data-analytics-event="{&quot;category&quot;:&quot;Footer&quot;,&quot;action&quot;:&quot;go to terms&quot;,&quot;label&quot;:&quot;text:terms&quot;}">Terms</a></li>
    <li class="mr-3 mr-lg-0"><a href="https://docs.github.com/en/github/site-policy/github-privacy-statement" data-analytics-event="{&quot;category&quot;:&quot;Footer&quot;,&quot;action&quot;:&quot;go to privacy&quot;,&quot;label&quot;:&quot;text:privacy&quot;}">Privacy</a></li>
    <li class="mr-3 mr-lg-0"><a data-analytics-event="{&quot;category&quot;:&quot;Footer&quot;,&quot;action&quot;:&quot;go to security&quot;,&quot;label&quot;:&quot;text:security&quot;}" href="https://github.com/security">Security</a></li>
    <li class="mr-3 mr-lg-0"><a href="https://www.githubstatus.com/" data-analytics-event="{&quot;category&quot;:&quot;Footer&quot;,&quot;action&quot;:&quot;go to status&quot;,&quot;label&quot;:&quot;text:status&quot;}">Status</a></li>
    <li class="mr-3 mr-lg-0"><a data-ga-click="Footer, go to help, text:Docs" href="https://docs.github.com">Docs</a></li>
    <li class="mr-3 mr-lg-0"><a href="https://support.github.com/tags=dotcom-footer" data-analytics-event="{&quot;category&quot;:&quot;Footer&quot;,&quot;action&quot;:&quot;go to contact&quot;,&quot;label&quot;:&quot;text:contact&quot;}">Contact GitHub</a></li>
    <li class="mr-3 mr-lg-0"><a href="https://github.com/pricing" data-analytics-event="{&quot;category&quot;:&quot;Footer&quot;,&quot;action&quot;:&quot;go to Pricing&quot;,&quot;label&quot;:&quot;text:Pricing&quot;}">Pricing</a></li>
    <li class="mr-3 mr-lg-0"><a href="https://docs.github.com" data-analytics-event="{&quot;category&quot;:&quot;Footer&quot;,&quot;action&quot;:&quot;go to api&quot;,&quot;label&quot;:&quot;text:api&quot;}">API</a></li>
    <li class="mr-3 mr-lg-0"><a href="https://services.github.com" data-analytics-event="{&quot;category&quot;:&quot;Footer&quot;,&quot;action&quot;:&quot;go to training&quot;,&quot;label&quot;:&quot;text:training&quot;}">Training</a></li>
    <li class="mr-3 mr-lg-0"><a href="https://github.blog" data-analytics-event="{&quot;category&quot;:&quot;Footer&quot;,&quot;action&quot;:&quot;go to blog&quot;,&quot;label&quot;:&quot;text:blog&quot;}">Blog</a></li>
    <li><a data-ga-click="Footer, go to about, text:about" href="https://github.com/about">About</a></li>
  </ul>
</nav>
```

```
<template id="site-details-dialog">
```



X

```
<div class="Popover js-hovercard-content position-absolute" style="display: none; outline: none;" tabindex="0">
```

```
<template id="snippet-clipboard-copy-button">
```



```
<style>
  .user-mention[href$="/geandrol"] {
    color: var(--color-user-mention-fg);
    background-color: var(--color-user-mention-bg);
    border-radius: 2px;
    margin-left: -2px;
    margin-right: -2px;
    padding: 0 2px;
  }
</style>
```