Guia de Deploy - Sistema de Consulta de Estoque

Parte 1: Subindo os Arquivos para o Git

Este guia detalha o processo de como você pode versionar e hospedar o código do seu Sistema de Consulta de Estoque em um repositório Git. Utilizaremos o GitHub como exemplo, mas os princípios são aplicáveis a qualquer serviço de hospedagem Git (GitLab, Bitbucket, Azure DevOps, etc.).

O que é Git e por que usá-lo?

Git é um sistema de controle de versão distribuído que rastreia as mudanças em qualquer conjunto de arquivos, geralmente usado para coordenar o trabalho entre programadores que desenvolvem colaborativamente o código-fonte durante o desenvolvimento de software. Em termos mais simples, ele permite que você:

- **Rastreie mudanças:** Saiba exatamente o que foi alterado, quando e por quem.
- **Colabore:** Trabalhe em equipe no mesmo projeto sem sobrescrever o trabalho uns dos outros.
- **Reverta versões:** Volte para versões anteriores do seu código a qualquer momento.
- **Gerencie diferentes versões:** Crie

diferentes "ramificações" do seu projeto para desenvolver novas funcionalidades sem afetar a versão principal.

Para este projeto, o Git será essencial para você manter seu código organizado e pronto para ser implantado em qualquer servidor.

Pré-requisitos

Antes de começar, certifique-se de ter o seguinte:

1. **Conta no GitHub (ou similar):** Se você ainda não tem uma, crie uma conta gratuita em github.com.

2. **Git Instalado:** Verifique se o Git está instalado em sua máquina local. Abra o terminal (ou prompt de comando) e digite:

```bash
git --version
```

Se o Git não estiver instalado, você pode baixá-lo e instalá-lo a partir do site oficial: git-scm.com/downloads.

Passo a Passo: Subindo seu Projeto para o GitHub

Siga estas etapas para criar um novo repositório no GitHub e enviar os arquivos do seu Sistema de Consulta de Estoque.

1. Criar um Novo Repositório no GitHub

1. Faça login na sua conta do GitHub.
2. No canto superior direito, clique no sinal de `+` e selecione `New repository` (Novo repositório).
3. **Nome do Repositório:** Escolha um nome descritivo para o seu projeto, por exemplo, `sistema-estoque-consulta`.
4. **Descrição (Opcional):** Adicione uma breve descrição do seu projeto.
5. **Público ou Privado:** Escolha `Public` (Público) se quiser que o código seja visível para todos, ou `Private` (Privado) se quiser mantê-lo apenas para você e colaboradores.
6. **Inicializar com README: Não** marque a opção `Add a README file` (Adicionar um arquivo README), `Add .gitignore` (Adicionar .gitignore) ou `Choose a license` (Escolher uma licença). Faremos isso manualmente para ter mais controle.
7. Clique em `Create repository` (Criar repositório).

Após a criação, o GitHub mostrará uma página com instruções sobre como configurar seu repositório localmente. Mantenha esta página aberta, pois você precisará do URL do repositório.

2. Inicializar o Git Localmente e Adicionar os Arquivos

Agora, você vai preparar a pasta do seu projeto local para ser versionada pelo Git e adicionar os arquivos.

1. **Navegue até a pasta raiz do seu projeto:** Abra o terminal (ou prompt de comando) e use o comando `cd` para ir até a pasta onde estão os arquivos do seu sistema. Por exemplo:

   ```bash
   cd /home/ubuntu/stock-api-deploy
   ```

Importante: Certifique-se de estar na pasta `stock-api-deploy`, que contém tanto o backend Flask quanto o frontend React (na pasta `src/static`).

2. **Inicialize o repositório Git:** Dentro da pasta do projeto, execute:

   `bash`
   `git init`

   Isso cria uma subpasta oculta chamada `.git`, que é onde o Git armazena todas as informações de versionamento.

3. **Adicionar um arquivo `.gitignore`:** Este arquivo informa ao Git quais arquivos e pastas ele deve ignorar (não rastrear). É crucial para não enviar arquivos desnecessários ou sensíveis (como ambientes virtuais, logs, etc.). Crie um arquivo chamado `.gitignore` na raiz da pasta `stock-api-deploy` com o seguinte conteúdo:

   ```gitignore

# Python

**pycache**/
.pyc
.pyo
.pyd
.Python
env/
venv/
.egg
*.egg-info/
dist/
build/

# Node.js

node_modules/
.env
   ```

Logs

*.log

Operating System files

.DS_Store
Thumbs.db

Consolidated stock data (if you regenerate it often)

consolidated_stock.csv
```

**Explicação do `.gitignore`:**
* `__pycache__/`, `*.pyc`, `*.pyo`, `*.pyd`: Arquivos de cache e compilados do Python.
* `env/`, `venv/`: Pastas de ambiente virtual Python. Elas contêm as bibliotecas instaladas e não devem ser versionadas, pois podem ser recriadas.
* `*.egg`, `*.egg-info/`, `dist/`, `build/`: Arquivos de empacotamento e distribuição do Python.
* `node_modules/`: Pasta de dependências do Node.js (usada pelo React). É muito grande e deve ser ignorada.
* `.env`: Arquivos de variáveis de ambiente (podem conter informações sensíveis).
* `*.log`: Arquivos de log.
* `.DS_Store`, `Thumbs.db`: Arquivos gerados por sistemas operacionais.
* `consolidated_stock.csv`: Se você gera este arquivo diariamente e não quer versionar cada nova versão, adicione-o aqui. Caso contrário, remova esta linha para que ele seja incluído no repositório.

4. **Adicionar todos os arquivos ao "staging area":**

`bash`
`git add .`

O `.` significa "todos os arquivos e pastas no diretório atual e subdiretórios". Este comando adiciona os arquivos que você deseja incluir no próximo commit.
```

3. Fazer o Primeiro Commit

Um commit é um "instantâneo" das mudanças que você fez. É como salvar o progresso do seu trabalho.

1. **Crie o commit:**

   ```bash
   git commit -m "Primeiro commit: Sistema de Consulta de Estoque"
   ```

   A flag `-m` permite que você adicione uma mensagem curta e descritiva sobre as mudanças feitas neste commit. Boas mensagens de commit são importantes para entender o histórico do projeto.

4. Conectar ao Repositório Remoto e Enviar (Push)

Agora você vai conectar seu repositório local ao repositório que você criou no GitHub e enviar seus arquivos.

1. **Adicionar o repositório remoto:** No terminal, execute o comando que o GitHub forneceu na página do seu novo repositório. Ele será algo parecido com:

   ```bash
   git remote add origin https://github.com/SEU_USUARIO/seu-repositorio.git
   ```

   Substitua `SEU_USUARIO` pelo seu nome de usuário do GitHub e `seu-repositorio` pelo nome que você deu ao seu repositório (ex: `sistema-estoque-consulta`). `origin` é o nome padrão para o repositório remoto principal.

2. **Renomear a branch principal (opcional, mas recomendado):** Por padrão, o Git inicializa a branch como `master`. O GitHub agora usa `main` como padrão. Para evitar problemas, renomeie sua branch local:

   ```bash
   git branch -M main
   ```

3. **Enviar os arquivos para o GitHub:**

   ```bash
   git push -u origin main
   ```

   Este comando envia (push) as mudanças da sua branch local `main` para a branch `main` no repositório remoto `origin`. A flag `-u` (ou `--set-upstream`) define `origin main` como o upstream padrão, o que significa que nas próximas vezes você poderá usar apenas `git push`.

Se for a primeira vez que você envia para o GitHub nesta máquina, ele pode pedir seu nome de usuário e senha (ou um Personal Access Token, que é o método recomendado para segurança).

Verificando no GitHub

Após o `git push` ser concluído, atualize a página do seu repositório no GitHub. Você deverá ver todos os seus arquivos lá! Agora seu código está versionado e seguro.

Próximos Passos

Com seu código no GitHub, você pode facilmente cloná-lo em sua VPS (Servidor Virtual Privado) para fazer o deploy. A próxima parte deste guia abordará exatamente isso.

Parte 2: Deploy do Sistema em uma VPS (Servidor Virtual Privado)

Esta seção detalha como implantar seu Sistema de Consulta de Estoque em um Servidor Virtual Privado (VPS). Uma VPS oferece um ambiente mais robusto e escalável para aplicações web em comparação com o ambiente de desenvolvimento local.

O que é uma VPS?

Uma VPS é uma máquina virtual que simula um servidor físico, mas é hospedada em um servidor físico maior. Ela oferece recursos dedicados (CPU, RAM, armazenamento) e controle total sobre o sistema operacional, permitindo que você instale e configure qualquer software necessário para sua aplicação. Provedores populares de VPS incluem DigitalOcean, Linode, AWS EC2, Google Cloud Compute Engine, entre outros.

Pré-requisitos

Antes de iniciar o deploy, certifique-se de ter o seguinte:

1. **Acesso SSH à sua VPS:** Você precisará do endereço IP da sua VPS, nome de usuário e senha (ou chave SSH) para acessá-la via terminal.
2. **Repositório Git:** Seu código deve estar no GitHub (ou serviço Git similar), conforme detalhado na Parte 1 deste guia.
3. **Conhecimento Básico de Linux:** Familiaridade com comandos básicos de terminal Linux será útil.

Passo a Passo: Deploy na VPS

1. Conectar à sua VPS via SSH

Abra seu terminal local e conecte-se à sua VPS usando o comando SSH. Substitua `seu_usuario` pelo seu nome de usuário na VPS e `seu_ip_vps` pelo endereço IP da sua VPS.

```
ssh seu_usuario@seu_ip_vps
```

Se for a primeira vez que você se conecta, o terminal pode perguntar se você deseja continuar a conexão; digite `yes`.

2. Atualizar o Sistema e Instalar Dependências Essenciais

Após conectar, é uma boa prática atualizar os pacotes do sistema e instalar ferramentas essenciais.

```
sudo apt update
sudo apt upgrade -y
sudo apt install -y python3 python3-pip python3-venv git nginx
```

- `sudo apt update` e `sudo apt upgrade -y`: Atualiza a lista de pacotes e os pacotes instalados.
- `python3`, `python3-pip`, `python3-venv`: Instala o Python 3, o gerenciador de pacotes pip e a ferramenta para criar ambientes virtuais.
- `git`: Instala o Git para clonar seu repositório.
- `nginx`: Instala o servidor web Nginx, que atuará como um proxy reverso para sua aplicação Flask.

3. Clonar o Repositório Git

Navegue até um diretório onde você deseja armazenar seu projeto (por exemplo, `/var/www/`) e clone seu repositório do GitHub.

```
sudo mkdir -p /var/www/stock_system
cd /var/www/stock_system
sudo git clone https://github.com/SEU_USUARIO/seu-repositorio.git .
```

- `sudo mkdir -p /var/www/stock_system`: Cria o diretório para o seu projeto (se ainda não existir).
- `cd /var/www/stock_system`: Navega para o diretório criado.

- `sudo git clone ... .` : Clona seu repositório. O `.` no final significa que o conteúdo do repositório será clonado diretamente para o diretório atual, sem criar uma subpasta extra. Lembre-se de substituir `SEU_USUARIO` e `seu-repositorio` pelo seu usuário e nome do repositório no GitHub.

Após clonar, você precisará ajustar as permissões para que o usuário que executará a aplicação (geralmente `www-data` ou o seu próprio usuário) tenha acesso.

```
sudo chown -R $USER:$USER /var/www/stock_system
```

4. Configurar o Ambiente Python

Crie um ambiente virtual Python para isolar as dependências do seu projeto.

```
cd /var/www/stock_system
python3 -m venv venv
source venv/bin/activate
pip install -r requirements.txt
```

- `python3 -m venv venv` : Cria um ambiente virtual chamado `venv` .
- `source venv/bin/activate` : Ativa o ambiente virtual. Você verá `(venv)` no início da linha do seu terminal, indicando que o ambiente está ativo.
- `pip install -r requirements.txt` : Instala todas as dependências listadas no seu arquivo `requirements.txt` (que você gerou na Parte 1).

5. Testar a Aplicação Flask (Gunicorn)

Para servir sua aplicação Flask em produção, usaremos o Gunicorn (Green Unicorn), um servidor WSGI (Web Server Gateway Interface) robusto e eficiente.

1. **Instalar Gunicorn:**

   `bash`
   `pip install gunicorn`

2. **Testar o Gunicorn:** Execute sua aplicação Flask usando o Gunicorn. Certifique-se de que o `main.py` esteja no diretório `src` e que o objeto `app` (sua instância Flask) esteja definido nele.

```bash
gunicorn --bind 0.0.0.0:5000 src.main:app &
```

- `--bind 0.0.0.0:5000`: Faz com que o Gunicorn escute em todas as interfaces de rede na porta 5000.
- `src.main:app`: Indica ao Gunicorn para carregar o objeto `app` do módulo `main.py` dentro da pasta `src`.
- `&`: Coloca o processo em segundo plano, liberando seu terminal.

Você pode verificar se a aplicação está rodando acessando `http://seu_ip_vps:5000` no seu navegador. Se tudo estiver correto, você verá a interface do seu sistema de estoque.

6. Configurar o Nginx como Proxy Reverso

O Nginx atuará como um proxy reverso, encaminhando as requisições da porta 80 (HTTP padrão) para a sua aplicação Gunicorn na porta 5000. Isso permite que você acesse sua aplicação sem precisar especificar a porta no navegador e também facilita a configuração de HTTPS (SSL/TLS) posteriormente.

1. **Criar um arquivo de configuração para o Nginx:**

   ```bash
   sudo nano /etc/nginx/sites-available/stock_system
   ```

   Cole o seguinte conteúdo no arquivo. Substitua `seu_dominio.com` pelo seu endereço IP da VPS ou, se você tiver um domínio configurado, pelo seu domínio.

   ```nginx
   server {
   listen 80;
   server_name seu_ip_vps;

   location / {
      proxy_pass http://127.0.0.1:5000;
      proxy_set_header Host $host;
      proxy_set_header X-Real-IP $remote_addr;
      proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
      proxy_set_header X-Forwarded-Proto $scheme;
   }
   }
   ```

   - `listen 80`: O Nginx escutará na porta HTTP padrão.

- `server_name seu_ip_vps` : Define o nome do servidor. Use o IP da sua VPS ou seu domínio.
- `location /` : Todas as requisições para a raiz do seu servidor serão encaminhadas.
- `proxy_pass http://127.0.0.1:5000` : Encaminha as requisições para o Gunicorn, que está rodando localmente na porta 5000.
- `proxy_set_header ...` : Configura os cabeçalhos para que a aplicação Flask receba as informações corretas sobre a requisição original.

Salve e feche o arquivo (Ctrl+X, Y, Enter no nano).

2. **Ativar a configuração do Nginx:** Crie um link simbólico do arquivo de configuração para o diretório `sites-enabled` .

```bash
sudo ln -s /etc/nginx/sites-available/stock_system /etc/nginx/sites-enabled/
```

3. **Remover a configuração padrão do Nginx:**

```bash
sudo rm /etc/nginx/sites-enabled/default
```

4. **Testar a configuração do Nginx e reiniciar:**

```bash
sudo nginx -t
sudo systemctl restart nginx
```

- `sudo nginx -t` : Testa a sintaxe do arquivo de configuração do Nginx. Se houver erros, corrija-os antes de reiniciar.
- `sudo systemctl restart nginx` : Reinicia o serviço Nginx para aplicar as novas configurações.

7. Configurar o Gunicorn para Rodar como um Serviço Systemd

Para garantir que o Gunicorn inicie automaticamente com o servidor e seja gerenciado de forma robusta, configuraremos um serviço Systemd.

1. **Criar um arquivo de serviço Systemd:**

```bash
sudo nano /etc/systemd/system/stock_system.service
```

Cole o seguinte conteúdo no arquivo. Certifique-se de ajustar os caminhos e o nome de usuário conforme necessário.

```ini
[Unit]
Description=Gunicorn instance to serve stock_system
After=network.target

[Service]
User=seu_usuario
Group=www-data
WorkingDirectory=/var/www/stock_system
ExecStart=/var/www/stock_system/venv/bin/gunicorn --workers 3 --bind unix:/var/www/stock_system/stock_system.sock src.main:app
Restart=always

[Install]
WantedBy=multi-user.target
```

* `User=seu_usuario`: Substitua `seu_usuario` pelo seu nome de usuário na VPS (ou `www-data` se preferir).
* `Group=www-data`: O grupo que terá permissão para acessar o **socket do** Gunicorn.
* `WorkingDirectory`: O diretório raiz **do** seu projeto.
* `ExecStart`: O comando para iniciar o Gunicorn. Note que estamos usando um **socket** Unix (`unix:/var/www/stock_system/stock_system.sock`) em vez de uma porta TCP. Isso é mais eficiente e seguro para comunicação entre Nginx e Gunicorn **no** mesmo servidor.
* `--workers 3`: Define o número de processos de worker **do** Gunicorn. Um bom ponto de partida é `(2 * número de núcleos da CPU) + 1`.

Salve e feche o arquivo.

1. **Ajustar a configuração do Nginx para usar o socket Unix:**

   Edite o arquivo de configuração do Nginx que você criou anteriormente:

   `bash`
   `sudo nano /etc/nginx/sites-available/stock_system`

   Altere a linha `proxy_pass` para usar o socket Unix:

   ```nginx
   server {
   listen 80;
   server_name seu_ip_vps;
   ```

```
location / {
  include proxy_params;
  proxy_pass http://unix:/var/www/stock_system/stock_system.sock;
}
}
```

Adicione também o arquivo `proxy_params` para incluir as configurações de cabeçalho padrão. Crie este arquivo:

```bash
sudo nano /etc/nginx/proxy_params
```

E cole o seguinte conteúdo:

```nginx
proxy_set_header Host $host;
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
proxy_set_header X-Forwarded-Proto $scheme;
```

Salve e feche ambos os arquivos.

2. **Recarregar Systemd e Iniciar o Serviço Gunicorn:**

```bash
sudo systemctl daemon-reload
sudo systemctl start stock_system
sudo systemctl enable stock_system
```

   - `sudo systemctl daemon-reload` : Recarrega o Systemd para reconhecer o novo arquivo de serviço.
   - `sudo systemctl start stock_system` : Inicia o serviço Gunicorn.
   - `sudo systemctl enable stock_system` : Garante que o serviço Gunicorn inicie automaticamente a cada reinicialização do servidor.

3. **Testar a configuração do Nginx e reiniciar novamente:**

```bash
sudo nginx -t
sudo systemctl restart nginx
```

Agora, ao acessar `http://seu_ip_vps` (ou seu domínio) no navegador, você deverá ver seu Sistema de Consulta de Estoque funcionando perfeitamente, servido pelo Nginx e Gunicorn.

Considerações Finais e Próximos Passos

- **HTTPS (SSL/TLS):** Para segurança em produção, é **altamente recomendado** configurar HTTPS. Você pode usar o Certbot para obter certificados SSL gratuitos do Let's Encrypt e configurá-los com o Nginx. O processo é geralmente simples:

  ```
  bash
  sudo apt install certbot python3-certbot-nginx
  sudo certbot --nginx -d seu_dominio.com
  ```

  Siga as instruções do Certbot.

- **Firewall:** Configure um firewall (como o UFW no Ubuntu) para permitir apenas as portas necessárias (80 para HTTP, 443 para HTTPS, 22 para SSH).

  ```
  bash
  sudo ufw allow OpenSSH
  sudo ufw allow 'Nginx HTTP'
  sudo ufw allow 'Nginx HTTPS'
  sudo ufw enable
  ```

- **Monitoramento e Logs:** Monitore os logs do Nginx (`/var/log/nginx/access.log` , `/var/log/nginx/error.log`) e do Gunicorn (via `journalctl -u stock_system.service`) para depurar problemas.

- **Atualizações:** Para atualizar seu código, basta ir ao diretório do projeto na VPS (`/var/www/stock_system`), fazer um `git pull` , e reiniciar o serviço Gunicorn (`sudo systemctl restart stock_system`).

Este guia fornece uma base sólida para o deploy do seu sistema. Lembre-se de que cada ambiente e provedor de VPS pode ter particularidades, mas os conceitos centrais permanecem os mesmos. Com este setup, você terá uma aplicação web robusta e pronta para uso em produção.