

**CURSO TECNÓLOGO EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS**

**NOME: Murillo Ferreira Ramos RM553315**

**Pedro Luiz Prado RM553874**

**William Kenzo Hayashi**  **RM552659**

**Challenge – OdontoPrev**

**OdontoPrev Resolução de problemas de Redução de Sinistros**

**São Paulo**

**2025**

**Murillo Ferreira Ramos**

**Pedro Luiz Prado William Kenzo Hayashi**

**Challenge – OdontoPrev**

**OdontoPrev Resolução de problemas de Redução de Sinistros**

Challenge apresentado pela empresa

OdontoPrev, como parte do desenvolvimento das mais diversas áreas de conhecimento do curso.

**São Paulo**

**2025**

**Sumário**

[Introdução 5](#_Toc198635501)

[Justificativa 6](#_Toc198635502)

[2.1 Relevância do Projeto 6](#_Toc198635503)

[2.2 Benefícios da Implementação 6](#_Toc198635504)

[Objetivo 8](#_Toc198635505)

[3.1 Objetivo Geral 8](#_Toc198635506)

[3.2 Objetivos Específicos 8](#_Toc198635507)

[Metodologia 10](#_Toc198635508)

[4.1 Tecnologias Utilizadas 10](#_Toc198635509)

[4.2 Arquitetura da Solução 11](#_Toc198635510)

[4.2.1 Desenho de Arquitetura 12](#_Toc198635511)

[4.2.1 Fluxo de Funcionamento da Aplicação 13](#_Toc198635512)

[4.3 Estrutura do Banco de Dados 14](#_Toc198635513)

[4.4 Processo de Desenvolvimento 15](#_Toc198635514)

[4.5 Pipeline CI/CD 16](#_Toc198635515)

[Etapas da Integração Contínua (CI) 16](#_Toc198635516)

[Etapas da Entrega Contínua (CD) 16](#_Toc198635517)

[4.6 Fluxo de Dados 18](#_Toc198635518)

[Rodagem da Aplicação 21](#_Toc198635519)

[5.1 Requisitos para Execução 21](#_Toc198635520)

[5.2 Instruções para Execução da Aplicação 21](#_Toc198635521)

[5.2.1 Derrubar Contêineres Existentes 21](#_Toc198635522)

[5.2.2 Construção das Imagens 22](#_Toc198635523)

[5.2.3 Inicialização dos Contêineres 22](#_Toc198635524)

[5.2.4 Verificação dos Contêineres em Execução 22](#_Toc198635525)

[5.2.5 Teste de Conectividade da API 23](#_Toc198635526)

[5.3 Procedimentos para Teste dos Endpoints 24](#_Toc198635527)

[5.3.1 Criar um Novo Cliente 24](#_Toc198635528)

[5.3.2 Listar Todos os Clientes 24](#_Toc198635529)

[5.3.3 Criar um Novo Atendimento 25](#_Toc198635530)

[5.3.4 Teste de Pagamento 25](#_Toc198635531)

[5.3.5 Listar os Dados Cadastrados 26](#_Toc198635532)

[5.4 Possíveis Erros e Soluções 27](#_Toc198635533)

[5.5 Considerações Finais sobre a Execução 27](#_Toc198635534)

[5.6 Considerações Finais sobre a Execução 28](#_Toc198635535)

[Arquivos de configuração utilizados: 28](#_Toc198635536)

[Resultados e Conclusão 29](#_Toc198635537)

[6.1 Análise dos Resultados 29](#_Toc198635538)

[6.2 Dificuldades Encontradas 31](#_Toc198635539)

[6.3 Conclusão 32](#_Toc198635540)

[6.4 Trabalhos Futuros 33](#_Toc198635541)

[6.5 Considerações Finais 33](#_Toc198635542)

[Links Importantes 35](#_Toc198635543)

[Vídeo de Apresentação (Sprint 03) 35](#_Toc198635544)

[(Ainda falta) Vídeo funcionamento da solução (Sprint 04) 35](#_Toc198635545)

[Documentação Técnica (Markdown): 35](#_Toc198635546)

[Referêcias 36](#_Toc198635547)

# Introdução

A transformação digital tem se consolidado como um fator essencial para a modernização dos processos organizacionais em diversas áreas, incluindo a odontologia. A crescente demanda por agilidade, eficiência e segurança no gerenciamento de atendimentos odontológicos impulsiona a necessidade de soluções tecnológicas que permitam a automação e a centralização de informações críticas.

Diante desse cenário, este projeto propõe o desenvolvimento de um sistema integrado para gestão de atendimentos odontológicos, que visa facilitar o gerenciamento de clientes, profissionais, agendamentos, pagamentos e sinistros. A solução adota uma abordagem baseada em arquitetura distribuída e computação em nuvem, permitindo maior escalabilidade, confiabilidade e segurança dos dados.

A implementação foi estruturada utilizando Python com Flask para o desenvolvimento da API, SQL Server como banco de dados relacional e Docker para a containerização da aplicação, garantindo maior portabilidade e facilidade de implantação. Além disso, foram aplicadas metodologias de DevOps, com a automação de processos de deploy e monitoramento contínuo, assegurando a eficiência operacional e a qualidade do software.

Este documento apresenta a fundamentação teórica, as tecnologias empregadas, a arquitetura da solução e as etapas de desenvolvimento. A documentação tem como objetivo fornecer uma visão detalhada do projeto, consolidando as decisões técnicas adotadas e demonstrando como a integração entre DevOps, Cloud Computing e boas práticas de desenvolvimento de software contribui para a construção de sistemas eficientes e escaláveis no setor odontológico.

# Justificativa

## 2.1 Relevância do Projeto

A transformação digital tem impactado diversos setores, e a área da saúde não é uma exceção. A crescente necessidade de otimização dos processos administrativos e operacionais em clínicas odontológicas reforça a importância do uso de **tecnologias avançadas** para garantir eficiência, segurança e acessibilidade. Muitos desses estabelecimentos ainda realizam o **gerenciamento de consultas, pagamentos e sinistros manualmente**, o que pode levar a **erros administrativos, retrabalho e atrasos no atendimento ao paciente**.

Nesse contexto, a implementação de uma **API REST integrada a um banco de dados** surge como uma solução inovadora, possibilitando a **automatização do fluxo de atendimento, a organização centralizada de informações e o acompanhamento em tempo real**. Além disso, ao adotar princípios de **DevOps e Cloud Computing**, a infraestrutura da aplicação se torna mais **escalável, segura e acessível**, permitindo que clínicas odontológicas de diferentes portes possam usufruir dos benefícios dessa tecnologia.

## 2.2 Benefícios da Implementação

A adoção da solução proposta traz **diversos benefícios estratégicos e operacionais**, garantindo maior controle sobre os processos internos e promovendo uma **experiência mais eficiente para pacientes e profissionais**. Dentre as principais vantagens da implementação, destacam-se:

* **Otimização do gerenciamento de atendimentos**: Redução do tempo gasto na organização e controle de consultas e procedimentos odontológicos.
* **Automação do controle financeiro**: Registro de pagamentos e acompanhamento do status financeiro dos atendimentos, reduzindo inconsistências contábeis.
* **Facilidade no gerenciamento de sinistros**: Registro e acompanhamento de eventos adversos com histórico detalhado, permitindo análise e resolução mais ágil.
* **Maior segurança e integridade dos dados**: O uso de um banco de dados estruturado garante **persistência e proteção das informações**, minimizando riscos de perda ou inconsistência de registros.
* **Escalabilidade e portabilidade**: A adoção de **Docker e Cloud Computing** permite que a solução seja facilmente replicada e adaptada para diferentes ambientes, atendendo desde pequenas clínicas até redes odontológicas de grande porte.
* **Aprimoramento do processo decisório**: Relatórios e consultas estruturadas possibilitam análises detalhadas sobre a operação da clínica, facilitando a identificação de **pontos de melhoria e tomada de decisão estratégica**.

Com a implementação deste projeto, espera-se que clínicas odontológicas tenham **um sistema mais moderno e eficiente**, promovendo melhor gestão interna e atendimento qualificado ao paciente.

# Objetivo

## 3.1 Objetivo Geral

O objetivo geral deste projeto é **desenvolver e implementar uma API REST para a gestão de atendimentos odontológicos**, permitindo o cadastro, acompanhamento e controle de informações relacionadas a clientes, profissionais, atendimentos, pagamentos e sinistros. A solução será desenvolvida aplicando **princípios de DevOps e Cloud Computing**, garantindo automação, escalabilidade e eficiência na execução dos processos administrativos das clínicas odontológicas.

## 3.2 Objetivos Específicos

Para atender ao objetivo geral, os seguintes objetivos específicos foram definidos:

* **Desenvolver um sistema estruturado para cadastro e gerenciamento de dados**: Criar endpoints REST para permitir a **inserção, atualização, exclusão e recuperação de registros** de clientes, profissionais, atendimentos e pagamentos.
* **Implementar práticas de DevOps para automação do ambiente**: Utilizar **Docker e Docker Compose** para containerização da aplicação, facilitando a implantação e escalabilidade do sistema.
* **Utilizar um banco de dados relacional para armazenamento seguro das informações**: Estruturar um **banco de dados SQL Server**, garantindo **integridade, consistência e segurança dos dados**.
* **Garantir qualidade e confiabilidade do sistema por meio de testes automatizados**: Implementar **testes unitários e testes de integração**, assegurando a estabilidade da API e a validação correta dos dados inseridos.
* **Criar uma infraestrutura escalável e replicável em ambientes de Cloud Computing**: Proporcionar flexibilidade para a aplicação ser executada em **diferentes servidores e plataformas em nuvem** sem necessidade de grandes adaptações.
* **Assegurar a rastreabilidade e controle das transações**: Garantir que cada atendimento e pagamento possa ser rastreado dentro do banco de dados, permitindo auditoria e análise futura.
* **Facilitar a consulta e análise de dados por meio de endpoints otimizados**: Criar operações eficientes que possibilitem a rápida recuperação de informações e aprimorem a **tomada de decisão na gestão da clínica**.

Esses objetivos refletem a proposta do projeto e asseguram que a solução desenvolvida atenda às necessidades do setor odontológico, promovendo eficiência operacional, segurança de dados e automação dos processos administrativos.

# Metodologia

A metodologia adotada para o desenvolvimento do Challenge DevOps - OdontoPrev baseia-se em práticas modernas de desenvolvimento de software, com ênfase na integração e entrega contínuas (CI/CD), automação de infraestrutura, versionamento de código e uso de contêineres para escalabilidade da solução. Esta seção detalha as tecnologias utilizadas, a arquitetura do sistema, a estrutura do banco de dados e os processos de desenvolvimento e testes.

## 4.1 Tecnologias Utilizadas

A escolha das tecnologias seguiu critérios de **eficiência, escalabilidade, compatibilidade e facilidade de integração**. A seguir, são apresentadas as principais ferramentas utilizadas no projeto:

* **Linguagem de Programação**: Python (com Flask, para a construção da API REST)
* **Banco de Dados**: Microsoft SQL Server (para armazenamento e gerenciamento de dados transacionais)
* **Containerização**: Docker e Docker Compose (para isolamento e replicação do ambiente de execução)
* **Versionamento de Código**: GitHub (controle de versão e colaboração entre os integrantes)
* **Gerenciamento de Infraestrutura**: Cloud Computing (com suporte à execução da API em ambiente distribuído)
* **Testes Automatizados**: Pytest (para validação de endpoints e consistência dos dados)
* **Ferramentas de Documentação**: Markdown (README), Draw.io (Diagramas) e Microsoft Word (para documentação formal)

## 4.2 Arquitetura da Solução

O projeto segue uma arquitetura modular baseada em **microserviços**, garantindo **independência entre componentes, escalabilidade e fácil manutenção**. A API REST atua como **intermediadora** entre os usuários e o banco de dados, gerenciando requisições e fornecendo respostas estruturadas.

A solução foi desenvolvida utilizando **princípios de DevOps**, garantindo **automação na implantação e execução da aplicação**. A arquitetura geral do sistema pode ser representada pelo seguinte diagrama:

Diagrama

O conteúdo gerado por IA pode estar incorreto.

### Interface gráfica do usuário O conteúdo gerado por IA pode estar incorreto.4.2.1 Desenho de Arquitetura

### 4.2.1 Fluxo de Funcionamento da Aplicação

O fluxo de funcionamento da API segue as seguintes etapas principais:

1. **Recebimento da requisição HTTP** do usuário via **Postman, navegador ou terminal**.
2. A requisição é processada pelo **servidor Flask**, que identifica e executa a ação correspondente (criação, consulta, atualização ou exclusão de dados).
3. A API se conecta ao **banco de dados SQL Server**, realizando a operação solicitada.
4. **A resposta é retornada ao usuário** no formato JSON, garantindo **facilidade de integração com outras aplicações**.

Diagrama

O conteúdo gerado por IA pode estar incorreto.

## 4.3 Estrutura do Banco de Dados

O banco de dados foi projetado para armazenar e gerenciar informações de forma **segura, consistente e escalável**. As principais tabelas do sistema são:

* **Clientes**: Armazena informações de pacientes cadastrados na clínica.
* **Profissionais**: Contém os dados dos odontologistas e suas especializações.
* **Atendimentos**: Registra consultas e procedimentos odontológicos realizados.
* **Pagamentos**: Controla os valores, métodos de pagamento e status financeiros.
* **Sinistros**: Gerencia ocorrências de problemas durante os atendimentos.

Diagrama

O conteúdo gerado por IA pode estar incorreto.A modelagem foi desenvolvida considerando **normalização, integridade referencial e otimização de consultas**, garantindo **eficiência na manipulação dos dados**.

## 4.4 Processo de Desenvolvimento

O desenvolvimento do projeto foi conduzido em **sprints**, seguindo a abordagem **Ágil e DevOps**, com ciclos iterativos de melhoria contínua. As etapas principais foram:

1. **Planejamento**: Definição dos requisitos e criação do backlog de tarefas.
2. **Desenvolvimento**: Implementação incremental dos endpoints e funcionalidades.
3. **Testes**: Validação dos endpoints com **testes unitários e testes integrados**.
4. **Containerização**: Empacotamento da aplicação utilizando **Docker e Docker Compose**.
5. **Implantação**: Subida dos containers e configuração da infraestrutura.
6. **Monitoramento e Ajustes**: Execução de testes finais e correção de eventuais falhas.

A adoção dessa metodologia permitiu um **desenvolvimento ágil, controlado e alinhado às boas práticas do mercado**, assegurando **qualidade e confiabilidade na solução entregue**.

## 4.5 Pipeline CI/CD

O projeto utiliza uma esteira de integração contínua e entrega contínua (CI/CD) configurada na plataforma Azure DevOps, com o objetivo de garantir agilidade, confiabilidade e rastreabilidade no processo de entrega da aplicação.

A aplicação foi completamente containerizada utilizando Docker, e todo o ciclo de vida da API é gerenciado por meio de pipelines CI/CD configuradas no Azure DevOps. O pipeline de CI realiza o build da imagem Docker a partir do Dockerfile presente no projeto e publica a imagem diretamente no Azure Container Registry (ACR). Em seguida, a pipeline de CD realiza o deploy automático dessa imagem no Azure App Service, configurado para rodar contêineres Docker personalizados. Essa abordagem garante maior portabilidade, escalabilidade e controle sobre o ambiente de execução da aplicação, alinhando-se às práticas modernas de DevOps e Cloud Computing.

### Etapas da Integração Contínua (CI)

A pipeline de CI foi configurada no arquivo azure-pipelines.yml, com as seguintes etapas:

1. **Instalação das dependências**: o ambiente Python é preparado com as bibliotecas necessárias especificadas no requirements.txt.
2. **Execução de testes automatizados**: os testes com pytest validam os principais endpoints da API, garantindo estabilidade antes do deploy.
3. **Publicação do artefato**: ao final da execução, é gerado e publicado um pacote .zip contendo a aplicação pronta para ser implantada.

### Etapas da Entrega Contínua (CD)

A pipeline de CD está definida no arquivo azure-deploy.yml, com foco no deploy contínuo e automatizado da API para o Azure App Service:

1. **Build final**: é realizado o empacotamento da aplicação para produção.
2. **Deploy no App Service**: a aplicação é enviada diretamente para a instância configurada, ficando disponível para acesso externo via URL pública.
3. **Verificação de sucesso**: após o deploy, os endpoints da API são testados manualmente ou via Postman para garantir que a aplicação esteja rodando

corretamente com conexão ativa ao banco de dados SQL Server.Essa arquitetura simplificada, sem uso de contêineres, garantiu maior velocidade na entrega e menor fricção na configuração de infraestrutura, mantendo todos os requisitos técnicos atendidos e melhorando a produtividade do time de desenvolvimento.

A seguir, apresentamos evidências visuais da execução bem-sucedida da pipeline CI/CD configurada no Azure DevOps. As imagens comprovam o funcionamento da integração e entrega contínua, desde o push de código até o deploy da aplicação no ambiente de produção.

**Diagrama

O conteúdo gerado por IA pode estar incorreto.**

## 4.6 Fluxo de Dados

1. Um push ou pull request para a branch main no repositório GitHub do projeto OdontoPrev inicia o fluxo de integração contínua. Essa ação representa a atualização do código-fonte por parte do desenvolvedor.

2. O repositório GitHub está integrado ao Azure DevOps por meio de uma conexão de serviço (Service Connection), permitindo que qualquer alteração enviada à branch monitorada seja detectada automaticamente pela plataforma de CI/CD.

3. Essa detecção aciona automaticamente a execução da pipeline de Integração Contínua (CI), definida no arquivo azure-pipelines.yml. O processo é iniciado sem intervenção manual, garantindo agilidade no ciclo de entrega.

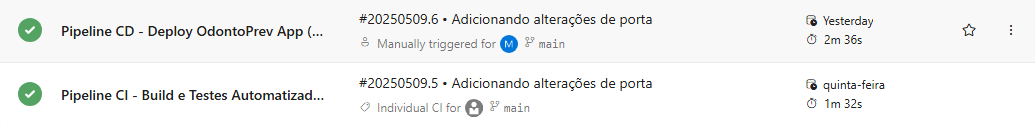
4. A pipeline de CI executa a preparação do ambiente com a instalação da versão correta do Python, do driver ODBC para acesso ao banco de dados, das dependências listadas em requirements.txt e da inicialização da API Flask. Em seguida, os testes automatizados são executados com pytest para garantir a estabilidade da aplicação antes da entrega.

5. Ao final da execução bem-sucedida da CI, é gerado e publicado automaticamente um artefato .zip, que contém a versão empacotada da aplicação. Esse artefato fica disponível no Azure DevOps e é utilizado como entrada para o processo de entrega contínua.

6. A publicação do artefato aciona a pipeline de Entrega Contínua (CD), definida no arquivo azure-deploy.yml. Essa pipeline é responsável por consumir o artefato gerado anteriormente e realizar o processo de implantação.

7. A pipeline de CD realiza o deploy automático da aplicação no serviço gerenciado Azure App Service, utilizando a task AzureWebApp@1, que envia o artefato .zip para o ambiente de produção. A aplicação torna-se acessível publicamente por meio de uma URL gerada pelo Azure.

8. Uma vez publicada, a API Flask estabelece conexão com o banco de dados relacional Azure SQL Database utilizando uma string de conexão segura via ODBC. A aplicação passa a processar requisições e realizar operações de leitura e escrita diretamente no banco hospedado em nuvem.

Execução da pipeline no Azure DevOps após push na branch main

Texto

O conteúdo gerado por IA pode estar incorreto.Texto

O conteúdo gerado por IA pode estar incorreto.

Uma imagem contendo Interface gráfica do usuário

O conteúdo gerado por IA pode estar incorreto.Etapas de testes e publicação sendo executadas com sucesso

Aplicação disponível no ar via App Service da Azure

# Rodagem da Aplicação

Esta seção apresenta os procedimentos necessários para a configuração e execução da aplicação, garantindo que o ambiente esteja corretamente configurado para o funcionamento da API REST. Serão abordados os passos de inicialização, execução dos contêineres e testes dos endpoints.

A solução foi desenvolvida para ser executada de maneira **automatizada e escalável**, utilizando contêineres Docker, permitindo a portabilidade e reprodutibilidade do ambiente.

## 5.1 Requisitos para Execução

Antes de rodar a aplicação, é necessário garantir que os seguintes requisitos estejam instalados no ambiente:

* **Docker** e **Docker Compose** para a execução dos contêineres.
* **Python 3.12** (caso precise testar a aplicação localmente sem container).
* **Postman** ou **curl/Invoke-RestMethod** para testar os endpoints da API.

Texto

O conteúdo gerado por IA pode estar incorreto.

## 5.2 Instruções para Execução da Aplicação

A aplicação pode ser inicializada seguindo os passos descritos abaixo:

### 5.2.1 Derrubar Contêineres Existentes

Antes de iniciar a API, é recomendável garantir que não há contêineres em execução. Isso evita conflitos de porta e possíveis resíduos de versões anteriores da aplicação.

**Comando:**

*docker-compose down*

Texto

O conteúdo gerado por IA pode estar incorreto.

### 5.2.2 Construção das Imagens

O próximo passo é a construção das imagens da aplicação. O uso do parâmetro --no-cache garante que todas as dependências sejam baixadas novamente, evitando problemas com versões desatualizadas.

**Comando:**

docker-compose build --no-cache

Texto

O conteúdo gerado por IA pode estar incorreto.

### 5.2.3 Inicialização dos Contêineres

Após a construção das imagens, os contêineres da aplicação podem ser iniciados em **modo desacoplado (-d)**, garantindo que a execução ocorra em segundo plano.

**Comando:**

docker-compose up -d

Texto

O conteúdo gerado por IA pode estar incorreto.

### 5.2.4 Verificação dos Contêineres em Execução

Para verificar se os contêineres foram iniciados corretamente e estão rodando, utilize o seguinte comando:

**Comando:**

docker ps

Texto

O conteúdo gerado por IA pode estar incorreto.

### 5.2.5 Teste de Conectividade da API

Após a inicialização dos contêineres, a API pode ser testada acessando o endpoint principal diretamente pelo navegador ou utilizando um cliente HTTP.

**Opção 1: Acessando via Navegador**  
Abra um navegador e acesse a URL:

http://localhost:5000/

Se a aplicação estiver funcionando corretamente, a página inicial exibirá uma mensagem confirmando a execução do servidor.

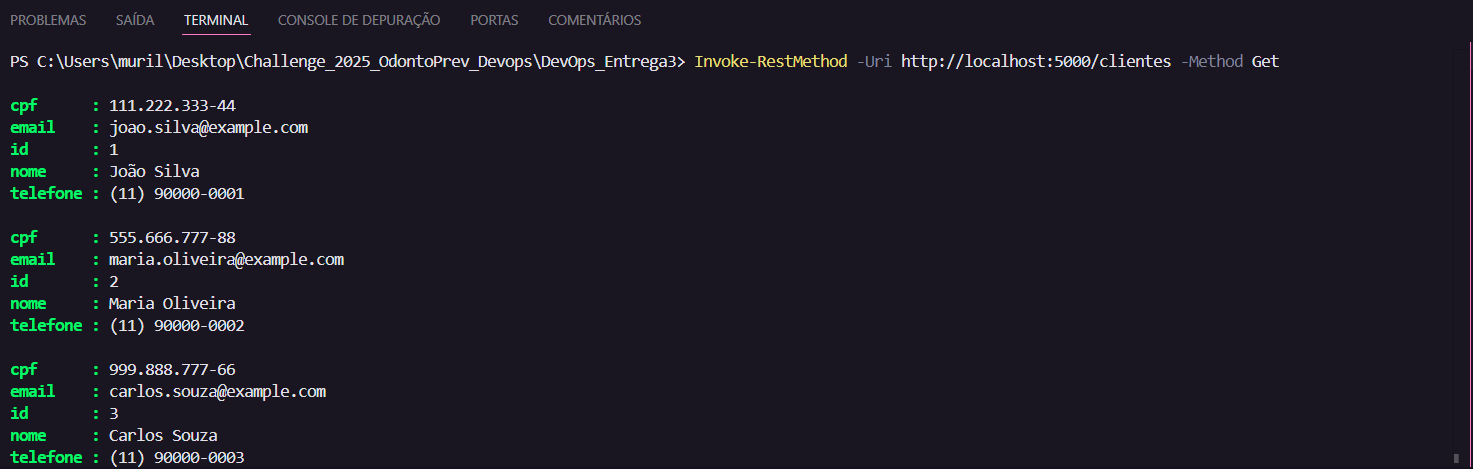
Interface gráfica do usuário

O conteúdo gerado por IA pode estar incorreto.

**Opção 2: Testando via Terminal (PowerShell)**  
Para testar o endpoint de listagem de clientes, execute:

Invoke-RestMethod -Uri http://localhost:5000/clientes -Method Get

Se houver clientes cadastrados, a resposta será um JSON com a lista de registros.



## 5.3 Procedimentos para Teste dos Endpoints

Para validar os principais endpoints da aplicação, os seguintes testes podem ser realizados utilizando **Postman** ou **cURL**:

### 5.3.1 Criar um Novo Cliente

**Comando:**

Invoke-RestMethod -Uri http://localhost:5000/clientes -Method Post -Body '{

"nome": "Carlos Souza",

"email": "carlos@email.com",

"cpf": "321.654.987-00",

"telefone": "(11) 90000-0000"

}'

Texto

O conteúdo gerado por IA pode estar incorreto.

### 5.3.2 Listar Todos os Clientes

**Comando:**

Invoke-RestMethod -Uri http://localhost:5000/clientes -Method Get

Texto

O conteúdo gerado por IA pode estar incorreto.

### 5.3.3 Criar um Novo Atendimento

**Comando:**

Invoke-RestMethod -Uri http://localhost:5000/atendimentos -Method Post -Body '{

"cliente\_id": 1,

"profissional\_id": 1,

"descricao": "Consulta ortodôntica",

"status": "Pendente"

}'

Texto

O conteúdo gerado por IA pode estar incorreto.

### 5.3.4 Teste de Pagamento

**Comando:**

Invoke-RestMethod -Uri http://localhost:5000/pagamentos -Method Post -Body '{

"atendimento\_id": 1,

"valor": 200.00,

"metodo\_pagamento": "PIX",

"status": "Pago"

}'

Texto

O conteúdo gerado por IA pode estar incorreto.

### 5.3.5 Listar os Dados Cadastrados

**Comando:**

Invoke-RestMethod -Uri http://localhost:5000/sinistros -Method Get

Texto

O conteúdo gerado por IA pode estar incorreto.

## 5.4 Possíveis Erros e Soluções

Durante a execução da aplicação, podem ocorrer erros relacionados à configuração do ambiente. Alguns dos problemas mais comuns e suas respectivas soluções são:

|  |  |  |
| --- | --- | --- |
| Erro | Causa Possível | Solução |
| Erro bind: address already in use | A porta **5000** já está em uso por outro serviço. | Finalize o processo que está utilizando a porta ou altere a configuração no **docker-compose.yml**. |
| Erro Cannot connect to the database | O SQL Server não está configurado corretamente ou não foi iniciado. | Verifique se o container do banco de dados está rodando e se as credenciais de acesso estão corretas. |
| Erro Failed to decode JSON object | Problema na formatação da requisição JSON. | Confirme se os dados enviados possuem **aspas duplas corretamente formatadas**. |

## 5.5 Considerações Finais sobre a Execução

A API foi projetada para **ser executada de forma automatizada e replicável**, garantindo a **portabilidade do ambiente** e facilitando a integração com diferentes sistemas. O uso de **Docker e DevOps** permitiu um fluxo de trabalho ágil e eficiente, reduzindo **tempo de configuração e mitigando possíveis falhas de ambiente**.

Para futuras implementações, podem ser incluídos **testes automatizados mais robustos**, **monitoramento com ferramentas como Prometheus e Grafana** e **orquestração de contêineres com Kubernetes** para garantir **alta disponibilidade da aplicação**.

## 5.6 Considerações Finais sobre a Execução

O projeto conta com uma esteira de automação criada com a ferramenta Azure DevOps, dividida em duas etapas principais: integração contínua (**CI**) e deploy contínuo (**CD**).

A pipeline de **CI** executa os testes automatizados utilizando o framework *pytest* sempre que um novo commit é realizado na branch principal (*main*) do repositório. Isso garante que as funcionalidades da API estejam funcionando corretamente antes de serem publicadas.

A pipeline de **CD** realiza o deploy automático da aplicação para um Web App Linux hospedado na Azure, utilizando a imagem Docker definida no projeto. O deploy é feito de forma direta, após o CI passar com sucesso ou mediante execução manual no Azure DevOps.

### Arquivos de configuração utilizados:

* **azure-pipelines.yml**: Define os passos da integração contínua.
* **azure-deploy.yml**: Responsável por orquestrar o deploy via Docker no Azure Web App.
* **Dockerfile:** Define a estrutura do ambiente de execução da aplicação Flask.
* **docker-compose.yml:** Permite executar a aplicação em ambiente local com múltiplos serviços (API e banco de dados SQL Server).

**URL do aplicativo publicado:**

<https://odontoprev-rm553315.azurewebsites.net>

**Repositório com o código-fonte:**

<https://github.com/mulliru/Challenge_2025_OdontoPrev_Devops>

Atualmente, a execução automática da pipeline depende da liberação de recursos gratuitos (*Hosted* *Parallelism*) pela *Microsoft*. O pedido já foi enviado, e assim que liberado, a execução completa da esteira será demonstrada no vídeo da apresentação.

# Resultados e Conclusão

## 6.1 Análise dos Resultados

A implementação do **Challenge DevOps - OdontoPrev** permitiu validar a aplicação de conceitos fundamentais de **DevOps e Cloud Computing** no desenvolvimento e entrega contínua de uma API REST para gestão de atendimentos odontológicos.

Os testes realizados demonstraram que a aplicação **opera de maneira eficiente e confiável**, permitindo o gerenciamento de **clientes, profissionais, atendimentos, pagamentos e sinistros**. Os endpoints da API foram testados utilizando **PowerShell, Postman e navegador**, garantindo a integridade das operações **CRUD** e a consistência das informações no banco de dados SQL Server.



A utilização de **Docker** facilitou a **padronização do ambiente de desenvolvimento**, eliminando inconsistências entre diferentes máquinas e garantindo a replicabilidade da solução. Além disso, o uso de **versionamento no GitHub** assegurou a rastreabilidade do código e das alterações realizadas ao longo do projeto.

Os principais resultados observados foram:

* **Automação da infraestrutura**: A utilização do Docker e Docker Compose permitiu que a API fosse rapidamente implantada em qualquer ambiente.
* **Escalabilidade e modularidade**: A arquitetura do sistema foi projetada para permitir futuras expansões, como a adição de novos serviços e integrações com outras plataformas.
* Texto

  O conteúdo gerado por IA pode estar incorreto.**Melhoria na gestão de processos**: A API centralizou os dados e operações, reduzindo riscos de inconsistência e otimizando o fluxo de trabalho das clínicas odontológicas.

A entrega final do projeto utiliza contêineres Docker desde o desenvolvimento até o ambiente de produção, garantindo ambientes idênticos, implantação automatizada e escalabilidade. O uso do Azure Container Registry (ACR) como repositório de imagens e a execução em um App Service baseado em contêineres comprovaram a eficiência do pipeline implementado.

## 6.2 Dificuldades Encontradas

Durante o desenvolvimento da aplicação, algumas dificuldades foram identificadas e solucionadas ao longo do processo. Entre os desafios enfrentados, destacam-se:

|  |  |
| --- | --- |
| Desafio | Solução Aplicada |
| Configuração inicial do banco de dados SQL Server no contêiner | Foi realizada a configuração adequada do **ODBC Driver 18** e ajustes no **docker-compose.yml** para garantir conectividade. |
| Problemas na serialização de dados JSON em requisições POST | A implementação de validação e **tratamento de erros** melhorou a confiabilidade na comunicação com a API. |
| Conflitos de portas ao subir os contêineres | Foi necessária a verificação de processos ativos utilizando docker ps e a redefinição de portas no docker-compose.yml. |
| Ajustes na estrutura do banco de dados | A estruturação inicial foi revisada e normalizada para evitar duplicação de informações. |

Texto

O conteúdo gerado por IA pode estar incorreto.

## 6.3 Conclusão

O desenvolvimento do **Challenge DevOps - OdontoPrev** demonstrou que a adoção de práticas **DevOps e Cloud Computing** pode **agilizar o desenvolvimento de software e melhorar a escalabilidade das soluções**. A utilização de contêineres Docker e a integração contínua permitiram a entrega de um sistema **modular, seguro e de fácil replicação**.

Além disso, o projeto proporcionou **aprendizado prático na utilização de tecnologias modernas**, incluindo **Python (Flask), SQL Server, Docker, e versionamento com GitHub**, possibilitando a construção de uma solução funcional e adaptável a novas necessidades.

A API está estruturada para suportar **expansões futuras**, incluindo a implementação de **autenticação de usuários, dashboards interativos e integração com serviços externos** para otimização do fluxo de trabalho das clínicas odontológicas.

Texto

O conteúdo gerado por IA pode estar incorreto.Dessa forma, a solução proposta **atende aos objetivos do projeto e contribui para a digitalização e automação da gestão odontológica**, demonstrando a viabilidade de uma abordagem DevOps aplicada a esse contexto.

## 6.4 Trabalhos Futuros

Com base nos aprendizados adquiridos ao longo deste desafio, algumas melhorias podem ser implementadas no projeto para ampliar suas funcionalidades e garantir maior robustez:

* **Implementação de autenticação e autorização** utilizando JWT para controle de acesso.
* **Criação de dashboards interativos** para visualização de dados estatísticos sobre os atendimentos.
* **Monitoramento e logging avançado**, utilizando ferramentas como Prometheus e Grafana.
* **Orquestração de contêineres com Kubernetes**, permitindo escalabilidade automática da aplicação.
* **Otimização do desempenho do banco de dados**, com indexação avançada e tuning de queries.

## 6.5 Considerações Finais

O projeto permitiu a aplicação prática de conceitos fundamentais do ciclo de vida de aplicações **DevOps e Cloud Computing**, reforçando a importância da **automação, versionamento e boas práticas de desenvolvimento**. A API desenvolvida representa uma solução eficiente para a gestão de atendimentos odontológicos e pode ser aprimorada e expandida para atender a novos requisitos no futuro.

O uso de **ferramentas modernas e boas práticas de desenvolvimento** foram determinantes para o sucesso da entrega, garantindo **escalabilidade, confiabilidade e segurança** à solução proposta.

Com isso, o **Challenge DevOps - OdontoPrev** conclui-se como uma entrega **satisfatória e aplicável ao contexto profissional**, contribuindo para o desenvolvimento de habilidades técnicas e metodológicas na área de tecnologia.

Texto

O conteúdo gerado por IA pode estar incorreto.Texto

O conteúdo gerado por IA pode estar incorreto.

# Links Importantes

### Vídeo de Apresentação (Sprint 03)

<https://www.youtube.com/watch?v=60iEipKjBGI>

### (Ainda falta) Vídeo funcionamento da solução (Sprint 04)

### Documentação Técnica (Markdown):

<https://github.com/mulliru/Challenge_2025_OdontoPrev_Devops>

# Referêcias

**MICROSOFT**. *Documentação oficial do SQL Server*. 2024. Disponível em: <https://learn.microsoft.com/pt-br/sql/>. Acesso em: 14 mar. 2025.

**MICROSOFT**. *Docker e Contêineres no Azure*. 2024. Disponível em: <https://learn.microsoft.com/pt-br/azure/container-instances/>. Acesso em: 14 mar. 2025.

**MICROSOFT**. *Documentação sobre APIs REST e Desenvolvimento de Back-end*. 2024. Disponível em: <https://learn.microsoft.com/pt-br/rest-api/>. Acesso em: 14 mar. 2025.

**MICROSOFT**. *Gerenciamento de Banco de Dados com SQL Server*. 2024. Disponível em: <https://learn.microsoft.com/pt-br/sql/database-engine/>. Acesso em: 14 mar. 2025.

**ALURA**. *DevOps na Prática: Containers e Automação com Docker e Kubernetes*. 2024. Disponível em: https://www.alura.com.br/curso-online-docker-fundamentos. Acesso em: 14 mar. 2025.

**ALURA**. *APIs REST com Python e Flask: Construção e Boas Práticas*. 2024. Disponível em: https://www.alura.com.br/curso-online-api-rest-python-flask. Acesso em: 14 mar. 2025.

**ALURA**. *Banco de Dados SQL: Modelagem, Normalização e Indexação*. 2024. Disponível em: https://www.alura.com.br/curso-online-sql-modelagem. Acesso em: 14 mar. 2025.