**Faculdade de Informática e Administração Paulista**

**Nome dos Alunos:**

**Herbert Santos de Sousa: RM553227 João Pedro Pereira: RM553698 Enzo Franco Rocha: RM553643**

**Solução Odontoprev: DevOps Tools e Cloud Computing**

**São Paulo**

**Nome dos Alunos**

**Herbert Santos de Sousa: RM553227 João Pedro Pereira: RM553698 Enzo Franco Rocha: RM553643**

**Solução Odontoprev: DevOps Tools e Cloud Computing**

Atividade apresentada como exigência parcial para conclusão do Curso de Tecnologia em Análise e Desenvolvimento de Sistemas da Faculdade de Informática e Administração Paulista, na disciplina de *Devops Tools & Cloud Computing* sob a orientação do Prof. Thiago Rodrigues da Rocha

**São Paulo**

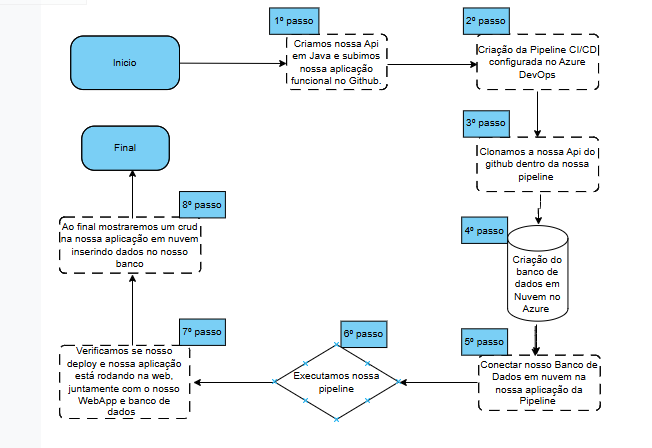
**Solução Odontoprev na Nuvem**

# Breve Descrição da Solução

A solução consiste em uma aplicação Java desenvolvida, hospedada em um WebApp no Azure através de uma pipeline CI/CD configurada no Azure DevOps. O pipeline automatiza as etapas de build, testes e deploy, garantindo entregas rápidas e consistentes. A aplicação está conectada a um banco de dados na nuvem, com duas tabelas relacionadas para persistência de dados.

O processo de integração e entrega contínua (CI/CD) foi implementado utilizando o Azure DevOps, permitindo que todo o ciclo de vida da aplicação — desde o build até o deploy em produção — ocorra de forma automatizada e padronizada. Isso reduz significativamente o esforço manual, minimiza erros humanos e garante que novas versões da aplicação sejam disponibilizadas de forma ágil e confiável.

# Desenho da Pipeline e detalhamento das etapas



O fluxo de Integração Contínua e Entrega Contínua (CI/CD) apresentado foi planejado para garantir agilidade, confiabilidade e automação no deploy da aplicação Java em ambiente de nuvem usando serviços da Microsoft Azure. A seguir, cada etapa do processo será descrita em detalhes.

**1º Passo – Criação da API em Java e publicação no GitHub**

Inicialmente, desenvolvemos uma API funcional utilizando Java com Spring Boot. Essa API é responsável por realizar operações básicas de um CRUD (Create, Read, Update, Delete). Após os testes locais, a aplicação foi publicada no GitHub, garantindo versionamento do código e preparação para integração com o Azure DevOps.

**2º Passo – Criação da Pipeline CI/CD no Azure DevOps**

No Azure DevOps, criamos uma pipeline de CI/CD. A integração contínua (CI) permite que a API seja automaticamente compilada e testada a cada alteração no repositório do GitHub. A entrega contínua (CD) garante que, ao ser aprovada, a nova versão da aplicação seja implantada automaticamente em um ambiente de produção ou homologação, garantindo agilidade e redução de erros manuais.

**3º Passo – Clonagem do repositório na pipeline**

A pipeline foi configurada para clonar o repositório da API diretamente do GitHub. Esse processo é automatizado e garante que a versão mais recente do código seja utilizada no processo de build e deploy, eliminando a necessidade de intervenção manual.

**4º Passo – Criação do banco de dados em nuvem (Azure SQL)**

Com a aplicação sendo preparada para o deploy, criamos um banco de dados em nuvem utilizando o serviço Azure SQL Database. Essa etapa é fundamental para armazenar os dados da aplicação e possibilitar o funcionamento completo do CRUD. O banco foi configurado com as credenciais necessárias, segurança básica e acessos restritos à aplicação.

**5º Passo – Conexão do banco de dados à aplicação na pipeline**

Configuramos a aplicação para se conectar ao banco de dados em nuvem. As credenciais de acesso e a URL do banco foram inseridas como variáveis de ambiente seguras na pipeline, garantindo que a aplicação pudesse acessar e manipular os dados corretamente após o deploy.

**6º Passo – Execução da pipeline**

Com todas as configurações concluídas, executamos a pipeline. O processo realiza o build da aplicação, os testes (caso existam), e o deploy automático em um WebApp do Azure, com a aplicação conectada ao banco de dados. Esta etapa é crucial para garantir a automação completa do processo.

**7º Passo – Validação do deploy e do ambiente em produção**

Após o deploy, verificamos se a aplicação estava de fato rodando corretamente na Web. Validamos se o WebApp foi publicado, se a API responde às requisições e se está conectada ao banco de dados, garantindo a integridade do ambiente e da comunicação entre os serviços.

**8º Passo – Demonstração do CRUD na aplicação em nuvem**

Por fim, realizamos uma demonstração completa do funcionamento da aplicação hospedada na nuvem, executando operações de inserção, consulta, atualização e exclusão de dados via API. Essa demonstração valida todo o pipeline e comprova o sucesso do deploy automatizado com integração ao banco de dados.

# Configuração das Pipelines (CI/CD) no Azure DevOps

* + Configurar o repositório GitHub no Azure DevOps.
  + Criar pipeline YAML usando o código fornecido.
  + Criar conexão de serviço MyAzureSubscription no Azure DevOps.
  + Configurar variáveis (rm, location, resourceGroup, app-name etc.).
  + Garantir permissões para deploy no Azure Web App.
  + Configurar banco de dados no Azure (SQL Database, MySQL, ou CosmosDB) e apontar a aplicação para esse banco via application.properties ou application.yml.

Segue código YAML para criação da pipeline (CI/CD):

trigger:

branches:

include:

- main

- master

- minharelease

pool:

vmImage: "ubuntu-latest"

variables:

- name: rm

value: 553227

- name: location

value: eastus

- name: resourceGroup

value: rg-OdontoPrev-Sprint

- name: service-plan

value: odontoprevSprint

- name: app-name

value: Odonto-PrevSolution-$(rm)

- name: runtime

value: JAVA:17-java17

- name: sku

value: F1

- name: nome-artefato

value: OdontoSolution

stages:

- stage: CriarInfra

jobs:

- job: criaWebApp

displayName: Criar ou atualizar o Serviço de Aplicativo

steps:

- task: AzureCLI@2

inputs:

azureSubscription: 'MyAzureSubscription'

scriptType: 'bash'

scriptLocation: 'inlineScript'

inlineScript: |

# Criar o Grupo de Recursos

az group create --location $(location) --name $(resourceGroup)

# Criar o Plano de Serviço com SO igual a Linux

az appservice plan create -g $(resourceGroup) -n $(service-plan) --is-linux --sku $(sku)

# Criar um Serviço de Aplicativo com JAVA 17 SE como runtime

az webapp create -g $(resourceGroup) -p $(service-plan) -n $(app-name) --runtime "$(runtime)"

visibleAzLogin: false

- stage: BuildApp

variables:

- name: mavenPOMFile

value: 'pom.xml'

jobs:

- job: buildWebApp

displayName: Realizar o Build do App

steps:

- task: Maven@4

displayName: 'Build PetClinic'

inputs:

mavenPomFile: '$(mavenPOMFile)'

testRunTitle: 'Testes Unitários'

jdkVersionOption: 1.17

- task: CopyFiles@2

displayName: 'Copiar a aplicação PetClinic'

inputs:

SourceFolder: '$(system.defaultworkingdirectory)'

Contents: 'target/\*.jar'

TargetFolder: '$(build.artifactstagingdirectory)'

- task: PublishBuildArtifacts@1

displayName: 'Publicar artefato do Build Petclinic'

inputs:

PathtoPublish: '$(build.artifactstagingdirectory)'

ArtifactName: $(nome-artefato)

- stage: DeployApp

jobs:

- job: DeployWebApp

displayName: Deploy no Serviço de Aplicativo

steps:

- task: DownloadBuildArtifacts@1

displayName: 'Baixar artefato gerado'

inputs:

buildType: 'current'

downloadType: 'specific'

downloadPath: '$(System.DefaultWorkingDirectory)'

- task: AzureRmWebAppDeployment@4

displayName: 'Deploy Petclinic'

inputs:

azureSubscription: 'MyAzureSubscription'

appType: 'webApp'

WebAppName: $(app-name)

packageForLinux: '$(System.DefaultWorkingDirectory)/$(nome-artefato)/target/\*.jar'

**Funcionamento mostrado no vídeo.**

# Tabelas do banco com relacionamentos entre elas

CREATE TABLE usuarios (

id\_user INT PRIMARY KEY,

nome VARCHAR(100) NOT NULL,

cpf VARCHAR(11) NOT NULL UNIQUE,

senha VARCHAR(255) NOT NULL,

data\_criacao DATE NOT NULL

);

CREATE TABLE fotos (

id\_fotos INT PRIMARY KEY,

usuario\_id INT NOT NULL,

caminho\_foto VARCHAR(255) NOT NULL,

data\_envio DATE NOT NULL,

FOREIGN KEY ( usuario\_id )

REFERENCES usuarios ( id\_user )

ON DELETE CASCADE

);

CREATE TABLE erros (

id\_erro INT PRIMARY KEY,

usuario\_id INT NULL,

mensagem VARCHAR2(50) NOT NULL,

data\_ocorrencia DATE NOT NULL,

FOREIGN KEY ( usuario\_id )

REFERENCES usuarios ( id\_user )

ON DELETE SET NULL

);

CREATE TABLE processo (

id\_processo INT PRIMARY KEY,

usuario\_id INT NULL,

analise\_id INT NOT NULL,

data\_analise DATE NOT NULL,

FOREIGN KEY ( usuario\_id )

REFERENCES usuarios ( id\_user )

ON DELETE SET NULL

);

CREATE TABLE notificacoes (

id\_notificacoes INT PRIMARY KEY,

usuario\_id INT NOT NULL,

mensagem VARCHAR(50) NOT NULL,

data\_notificacao DATE NOT NULL,

lida NUMBER(1) DEFAULT 0, -- 0 pra FALSE e 1 pra TRUE,

FOREIGN KEY ( usuario\_id )

REFERENCES usuarios ( id\_user )

ON DELETE CASCADE

);

CREATE TABLE analise\_preditiva (

id\_analise NUMBER(10) PRIMARY KEY,

resultado\_analise VARCHAR2(255) NOT NULL,

frequencia\_sinistros NUMBER(10) NOT NULL,

foto\_id NUMBER(10) NOT NULL,

data\_analise DATE NOT NULL,

CONSTRAINT fk\_analise\_foto FOREIGN KEY (foto\_id) REFERENCES foto(id)

);

CREATE TABLE funcionario (

id NUMBER(19) PRIMARY KEY,

nome VARCHAR2(100) NOT NULL,

cpf VARCHAR2(11) NOT NULL UNIQUE,

cargo VARCHAR2(50) NOT NULL,

salario NUMBER(15,2) NOT NULL,

data\_admissao DATE NOT NULL,

telefone VARCHAR2(15),

email VARCHAR2(50) UNIQUE

);

BEGIN

FOR rec IN (

SELECT

u.nome,

COUNT(f.id\_fotos) AS total\_fotos

FROM

usuarios u

INNER JOIN fotos f ON u.id\_user = f.usuario\_id

GROUP BY

u.nome

ORDER BY

total\_fotos DESC

) LOOP

dbms\_output.put\_line('Usu rio: '

|| rec.nome

|| ' - Total de Fotos: '

|| rec.total\_fotos);

END LOOP;

END;

/

BEGIN

FOR rec IN (

SELECT

u.nome,

COUNT(n.id\_notificacoes) AS total\_notificacoes

FROM

usuarios u

LEFT JOIN notificacoes n ON u.id\_user = n.usuario\_id

GROUP BY

u.nome

ORDER BY

total\_notificacoes DESC

) LOOP

dbms\_output.put\_line('Usu rio: '

|| rec.nome

|| ' - Total de Notifica es: '

|| rec.total\_notificacoes);

END LOOP;

END;

/

BEGIN

FOR rec IN (

SELECT

n.mensagem,

COUNT(n.id\_notificacoes) AS total\_notificacoes

FROM

notificacoes n

RIGHT JOIN usuarios u ON n.usuario\_id = u.id\_user

GROUP BY

n.mensagem

ORDER BY

total\_notificacoes DESC

) LOOP

dbms\_output.put\_line('Mensagem: '

|| nvl(rec.mensagem, 'Notifica o sem Usu rio')

|| ' - Total: '

|| rec.total\_notificacoes);

END LOOP;

END;

/

DECLARE

v\_id\_user INT := 1;

v\_novo\_nome VARCHAR2(100) := 'Novo\_Nome';

v\_nova\_senha VARCHAR2(255) := 'nova\_senha123';

BEGIN

UPDATE usuarios

SET

nome = v\_novo\_nome,

senha = v\_nova\_senha

WHERE

id\_user = v\_id\_user;

IF SQL%rowcount > 0 THEN

dbms\_output.put\_line('Usu rio atualizado com sucesso: ID ' || v\_id\_user);

ELSE

dbms\_output.put\_line('Nenhum usu rio encontrado com o ID ' || v\_id\_user);

END IF;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

dbms\_output.put\_line('Erro ao atualizar usu rio: ' || sqlerrm);

ROLLBACK;

END;

/

DECLARE

v\_id\_user INT := 1;

BEGIN

DELETE FROM usuarios

WHERE

id\_user = v\_id\_user;

IF SQL%rowcount > 0 THEN

dbms\_output.put\_line('Usu rio deletado com sucesso: ID ' || v\_id\_user);

ELSE

dbms\_output.put\_line('Nenhum usu rio encontrado com o ID ' || v\_id\_user);

END IF;

COMMIT; -- Confirma as altera es

EXCEPTION

WHEN OTHERS THEN

dbms\_output.put\_line('Erro ao deletar usu rio: ' || sqlerrm);

ROLLBACK; -- Desfaz as altera es em caso de erro

END;

/

# Repositório no GitHub

³ Link do GITHUB:

# Demonstração em Vídeo

Uma gravação será realizada para demonstrar o funcionamento da solução, contendo:

* + Clone do repositório e configuração do ambiente.

³ Link do vídeo: <https://youtu.be/Q4zXpiB7lR0>