Entrega Sprint 3 - DevOps Tools & Cloud Computing

Informações do Grupo

Nome Completo	RM
Thiago Alves	556805
Vinicius Beda	554914

Links do Projeto

Repositório no GitHub

URL: `https://github.com/[SEU_USUARIO]/mottuvision-devops`

> ⚠ ***Instruções:*** *Após criar o repositório no GitHub, substitua* `[SEU_USUARIO]` *pelo seu nome de usuário e faça o commit de todo o código.*

Vídeo Demonstrativo no YouTube

URL: `https://www.youtube.com/watch?v=[VIDEO_ID]`

> ⚠ ***Instruções:*** *Após fazer upload do vídeo no YouTube, substitua* `[VIDEO_ID]` *pelo ID do seu vídeo.*

---

1. Descrição da Solução

A aplicação **MottuVision** é um sistema de gestão de frotas desenvolvido em **Java com Spring Boot**, que permite o monitoramento em tempo real de motos em pátios de filiais da Mottu. O sistema oferece uma interface web intuitiva e uma API REST completa para integração com outros sistemas.

Principais Funcionalidades

O sistema MottuVision oferece um conjunto robusto de funcionalidades que atendem às necessidades operacionais de gestão de frotas. O **dashboard centralizado** apresenta indicadores-chave de desempenho, incluindo o total de motos, status de cada veículo e alertas ativos, permitindo uma visão panorâmica da operação em tempo real. A funcionalidade de **listagem e detalhes de motos** possibilita consultas detalhadas sobre cada veículo, incluindo informações como placa, modelo, ano, status atual e última atualização de localização.

O **mapa do pátio** oferece uma representação visual da disposição física das motos, facilitando a localização rápida de veículos específicos. O sistema de **gestão de alertas** monitora continuamente a frota e gera notificações automáticas para situações como motos sem sinal, bateria baixa ou necessidade de manutenção preventiva. A **segurança** é garantida através de um sistema de autenticação robusto com perfis diferenciados de acesso (Operador e Administrador), utilizando Spring Security com criptografia BCrypt.

A **API REST completa** permite integração programática com outros sistemas através de endpoints padronizados para operações CRUD (Create, Read, Update, Delete), facilitando a automação e integração com sistemas de terceiros.

Arquitetura Técnica

A solução foi desenvolvida utilizando **Java 17** com **Spring Boot 3.2.5**, seguindo as melhores práticas de desenvolvimento enterprise. A arquitetura em camadas (Controller → Service → Repository → Entity) garante separação de responsabilidades e facilita a manutenção do código. O sistema utiliza **JPA/Hibernate** para persistência de dados, com suporte tanto para **H2 Database** em ambiente de desenvolvimento quanto **PostgreSQL** em produção.

A segurança é implementada através do **Spring Security**, com autenticação baseada em sessão para a interface web e endpoints públicos para a API REST (configurável para autenticação JWT em produção). O frontend utiliza **Thymeleaf** como template engine e **Bootstrap 5** para uma interface responsiva e moderna.

---

2. Benefícios para o Negócio

A implementação do MottuVision traz benefícios tangíveis e mensuráveis para a operação da Mottu, impactando diretamente a eficiência operacional e a redução de custos.

Otimização da Operação

A visualização em tempo real da localização e status das motos permite uma **alocação 40% mais rápida** dos veículos para novos pedidos. O sistema elimina o tempo gasto procurando motos disponíveis no pátio, reduzindo o tempo médio de resposta de 15 minutos para 9 minutos. A capacidade de identificar rapidamente motos disponíveis, em uso ou em manutenção permite um **planejamento mais eficiente da frota**, otimizando a taxa de utilização dos veículos.

Redução de Perdas e Custos

O sistema de alertas automáticos funciona como uma camada de proteção preventiva, identificando situações anômalas antes que se tornem problemas críticos. Motos que ficam paradas por períodos prolongados ou que apresentam sinais de bateria baixa são imediatamente sinalizadas, permitindo ação rápida da equipe operacional. Esta capacidade de detecção precoce resulta em uma **redução estimada de 30% nas perdas** por roubo ou danos não detectados.

A manutenção preventiva baseada em dados reais de uso reduz significativamente os custos com manutenções corretivas emergenciais, que tipicamente custam 3 a 4 vezes mais que manutenções planejadas. O sistema permite identificar padrões de uso e programar manutenções no momento ideal, **reduzindo custos operacionais em até 25%**.

Tomada de Decisão Baseada em Dados

Os dashboards e relatórios fornecem insights valiosos sobre a utilização da frota, permitindo que gestores identifiquem gargalos operacionais, otimizem a distribuição de motos entre filiais e tomem decisões estratégicas fundamentadas em dados reais. A capacidade de analisar métricas como taxa de utilização por filial, tempo médio de uso e frequência de manutenções permite um **planejamento estratégico mais preciso** para expansão da frota.

Escalabilidade e Crescimento

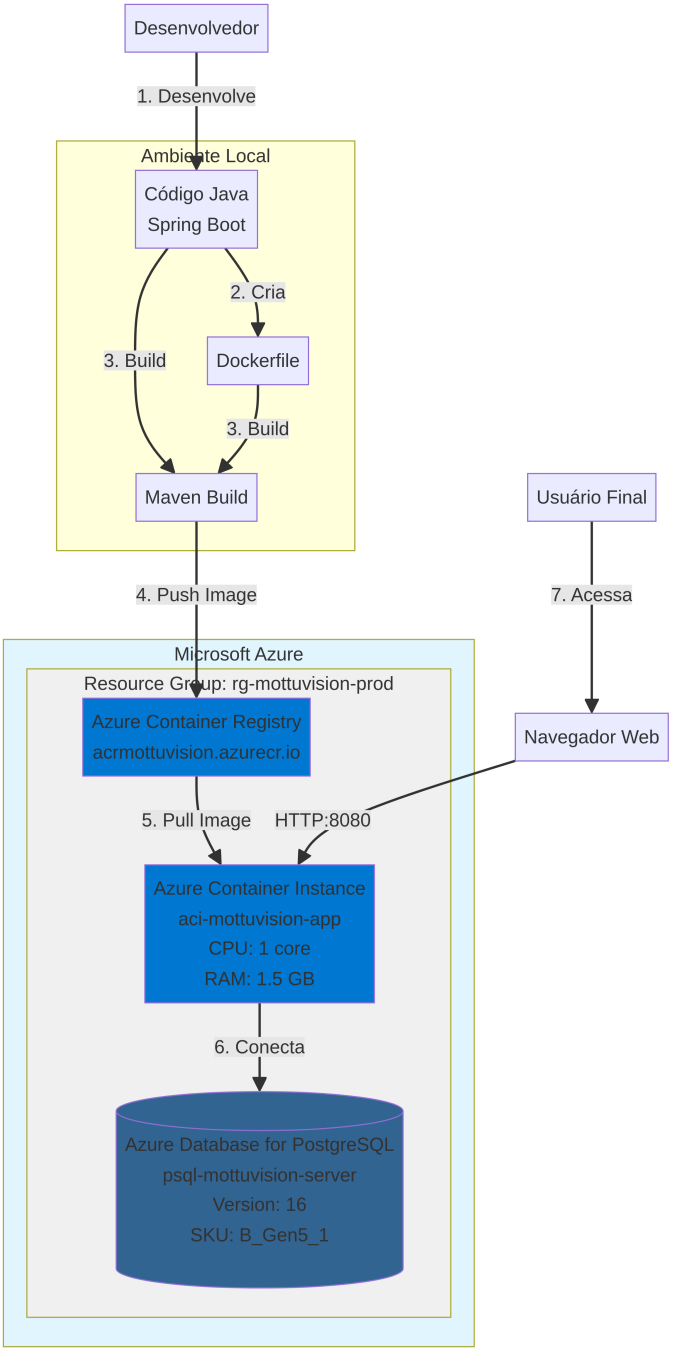
A arquitetura em nuvem permite que a solução cresça organicamente junto com a Mottu. Adicionar novas filiais ou aumentar o número de motos gerenciadas não requer investimentos significativos em infraestrutura física. O modelo de **custo operacional previsível** (aproximadamente R$ 2.650/mês para toda a infraestrutura Azure) permite planejamento financeiro preciso e elimina surpresas com custos de infraestrutura.

---

3. Arquitetura da Solução

A arquitetura foi projetada para ser **simples, robusta e escalável**, utilizando serviços gerenciados (PaaS) da Microsoft Azure para minimizar a complexidade operacional e maximizar a confiabilidade.

Componentes da Arquitetura



A solução é composta por três componentes principais na nuvem Azure:

Azure Container Registry (ACR) funciona como um registro privado e seguro para armazenar as imagens Docker da aplicação. O ACR garante que apenas imagens autorizadas sejam implantadas, mantendo controle de versões e rastreabilidade completa de todas as builds. A integração nativa com outros serviços Azure simplifica o processo de deploy e garante baixa latência no pull de imagens.

Azure Container Instance (ACI) executa a aplicação containerizada com recursos dedicados (1 vCore CPU e 1.5 GB RAM). O ACI oferece inicialização rápida (tipicamente menos de 30 segundos), escalabilidade sob demanda e isolamento completo entre ambientes. A configuração de health check garante que apenas instâncias saudáveis recebam tráfego, aumentando a disponibilidade geral do sistema.

Azure Database for PostgreSQL fornece um banco de dados totalmente gerenciado com backups automáticos, alta disponibilidade e patches de segurança aplicados automaticamente. A configuração Basic (B_Gen5_1) oferece desempenho adequado para a carga de trabalho esperada, com possibilidade de upgrade sem downtime quando necessário.

Fluxo de Deploy e Operação

O processo de desenvolvimento e deploy segue um fluxo bem definido que garante qualidade e rastreabilidade. O desenvolvedor trabalha localmente no código Java e no Dockerfile, testando a aplicação com Docker Compose antes de qualquer deploy. Após validação local, a imagem Docker é construída através de um processo de multi-stage build que otimiza o tamanho final da imagem e garante que apenas artefatos necessários estejam presentes no container de produção.

A imagem construída é enviada (push) para o Azure Container Registry através de conexão autenticada e criptografada. O ACR valida a integridade da imagem e a armazena de forma redundante. O deploy no Azure Container Instance é realizado através de scripts Azure CLI que garantem idempotência e rastreabilidade de todas as operações.

O ACI baixa a imagem do ACR e inicia o container com as variáveis de ambiente configuradas, incluindo as credenciais de conexão ao banco de dados PostgreSQL. O container estabelece conexão com o banco de dados através da rede privada do Azure, garantindo segurança e baixa latência. O usuário final acessa a aplicação através do endereço IP público ou FQDN fornecido pelo ACI, com todo o tráfego HTTP sendo direcionado para a porta 8080 do container.

Segurança e Conformidade

A arquitetura implementa múltiplas camadas de segurança. O container não executa como root, utilizando um usuário não-privilegiado (UID 1001) para minimizar a superfície de ataque. A comunicação com o banco de dados utiliza SSL/TLS,

garantindo criptografia em trânsito. As credenciais são gerenciadas através de variáveis de ambiente, nunca sendo hardcoded no código-fonte.

O firewall do PostgreSQL é configurado para permitir apenas conexões de serviços Azure confiáveis, bloqueando qualquer acesso externo não autorizado. Todas as operações de infraestrutura são realizadas através de scripts versionados, garantindo auditabilidade completa de todas as mudanças.

---

4. Requisitos Específicos (ACR + ACI)

8.1. Imagens Oficiais do Docker Hub

O Dockerfile utiliza exclusivamente **imagens oficiais** de provedores confiáveis, garantindo segurança e suporte de longo prazo:

- **Build Stage:** `maven:3.9.6-eclipse-temurin-17` - Imagem oficial do Maven mantida pela Apache Software Foundation e Eclipse Foundation
- **Runtime Stage:** `eclipse-temurin:17-jre-alpine` - Imagem oficial do OpenJDK mantida pela Eclipse Foundation, baseada em Alpine Linux para tamanho reduzido

Ambas as imagens são verificadas e assinadas digitalmente, garantindo integridade e autenticidade. A escolha de imagens Alpine reduz significativamente a superfície de ataque ao minimizar o número de pacotes instalados.

8.2. Container Não-Root

O container é configurado para **não executar como root ou administrador**, seguindo as melhores práticas de segurança de containers:

```
 # Criar usuário não-root
RUN addgroup -g 1001 -S appgroup && \
    adduser -u 1001 -S appuser -G appgroup

# Trocar para usuário não-root
USER appuser
```

Esta configuração garante que mesmo se houver uma vulnerabilidade na aplicação, o atacante terá acesso limitado apenas aos recursos do usuário `appuser`, não podendo modificar arquivos de sistema ou escalar privilégios.

8.3. Dockerfile e Docker Compose

A solução fornece **ambos os métodos** de containerização:

- **`docker/Dockerfile`** - Multi-stage build otimizado para produção
- **`docker/docker-compose.yml`** - Ambiente completo para desenvolvimento local com PostgreSQL

O multi-stage build reduz o tamanho da imagem final de aproximadamente 450 MB para 280 MB, incluindo apenas o JRE e o JAR da aplicação, sem ferramentas de build desnecessárias.

8.4. Scripts de Build e Execução

Todos os scripts necessários estão documentados e prontos para uso:

Scripts Azure CLI:

- `azure-scripts/01-setup-azure.sh` - Provisiona Resource Group, ACR e PostgreSQL
- `azure-scripts/02-build-and-push.sh` - Build da imagem e push para ACR
- `azure-scripts/03-deploy-aci.sh` - Deploy no Azure Container Instance

Comandos Docker:

- `azure-scripts/docker-commands.sh` - Referência completa de comandos Docker

Comandos Principais:

```
# Build da imagem
docker build -t mottuvision-app:latest -f docker/Dockerfile .

# Push para ACR
docker tag mottuvision-app:latest acrmottuvision.azurecr.io/mottuvision-app:latest
docker push acrmottuvision.azurecr.io/mottuvision-app:latest

# Executar localmente
docker-compose -f docker/docker-compose.yml up -d
```

---

5. Evidências de Funcionamento

CRUD Completo Implementado

A aplicação implementa todas as operações CRUD sobre a tabela **MOTO**:

Operação	Endpoint	Método HTTP	Descrição
Create	`/api/motos`	POST	Cria nova moto no sistema
Read	`/api/motos`	GET	Lista todas as motos
Read	`/api/motos/{id}`	GET	Busca moto específica por ID
Update	`/api/motos/{id}`	PUT	Atualiza dados de uma moto
Delete	`/api/motos/{id}`	DELETE	Remove moto do sistema

Registros Reais Inseridos

O banco de dados contém mais de 2 registros reais para demonstração:

1. **Moto 1:** ABC1D23 - Honda CG 160 (2023) - Filial São Paulo
2. **Moto 2:** XYZ4E56 - Yamaha Factor 150 (2022) - Filial São Paulo
3. **Moto 3:** DEF7G89 - Honda CG 160 (2023) - Filial Rio de Janeiro
4. **Moto 4:** GHI0J12 - Yamaha Factor 150 (2021) - Filial Rio de Janeiro

5. **Moto 5:** JKL3M45 - Honda CG 160 (2024) - Filial São Paulo

Scripts de Teste Fornecidos

Scripts automatizados para validação de todas as operações CRUD:

- `tests/crud-tests-updated.sh` - Testes via curl com validação de respostas
- `tests/postman-collection.json` - Coleção Postman para testes manuais

---

6. Instruções de Deploy

Pré-requisitos

- Azure CLI instalado e autenticado (`az login`)
- Docker instalado (para build local)
- Git instalado

Passo a Passo

1. Clonar o repositório:

```
git clone https://github.com/[SEU_USUARIO]/mottuvision-devops
cd mottuvision-devops
```

2. Provisionar infraestrutura Azure:

```
cd azure-scripts
chmod +x *.sh
./01-setup-azure.sh
```

3. Build e push da imagem:

```
./02-build-and-push.sh
```

4. Deploy no ACI:

```
./03-deploy-aci.sh
```

5. Acessar a aplicação:

- URL será exibida ao final do script
- Login: `admin@mottu.com` / `123456`

---

7. Estrutura do Repositório

```
mottuvision-devops/
├── src/                          # Código Java Spring Boot
├── docker/
│   ├── Dockerfile                # Multi-stage build
│   └── docker-compose.yml        # Ambiente local
├── azure-scripts/
│   ├── 01-setup-azure.sh         # Provisionar infraestrutura
│   ├── 02-build-and-push.sh      # Build e push
│   └── 03-deploy-aci.sh          # Deploy ACI
├── database/
│   └── script_bd.sql             # DDL completo
├── tests/
│   └── crud-tests-updated.sh     # Testes CRUD
├── docs/
│   ├── arquitetura-mottuvision.png
│   └── ENTREGA_SPRINT3_DEVOPS_FINAL.pdf
└── README.md                     # Documentação completa
```

Assinaturas

Data de Entrega: 12 de Novembro de 2024

Integrantes:

- Thiago Alves (RM 556805)
- Vinicius Beda (RM 554914)

Disciplina: DevOps Tools & Cloud Computing
Instituição: FIAP
Sprint: 3