

CloudWalk Monitoring System

Desafio de Monitoring Analyst

Slide 1: Introdução ao Projeto

CloudWalk Transaction Monitoring System

- **Objetivo:** Sistema de monitoramento em tempo real para transações com detecção de anomalias
- **Desafio:** Monitoring Analyst Test da CloudWalk
- **Repositório:** <https://github.com/fabenejr/cloudwalk-monitoring-test>
- **Status:** 🟢 **SISTEMA 100% OPERACIONAL**

"Where there is data smoke, there is business fire." — Thomas Redman

Slide 2: Arquitetura do Sistema

Componentes Principais

1. Servidor de Monitoramento (`server.js`)

- API REST para estatísticas, séries temporais e alertas
- Endpoint para receber transações em tempo real
- WebSocket para broadcast de dados em tempo real
- SQLite em memória carregado dos CSVs fornecidos
- Agendamento automático a cada 5 minutos para detecção de anomalias

2. Dashboard Web (`public/index.html`)

- Gráficos Chart.js interativos
- Métricas resumidas em tempo real
- Lista de alertas dinâmica

3. Análise de Dados (analysis/dataAnalysis.js)

- Análise exploratória dos CSVs
- Queries SQL otimizadas
- Insights e relatórios

4. Testes (tests/)

- Scripts automatizados de teste de API
- Validação de funcionalidades

Slide 3: Stack Tecnológica

Tecnologias Implementadas

Categoria	Tecnologia	Uso
Backend	Node.js + Express	Servidor principal e APIs
Database	SQLite (em memória)	Armazenamento de dados
Real-time	WebSocket (ws)	Comunicação em tempo real
Frontend	HTML5, CSS3, JavaScript	Dashboard interativo
Visualização	Chart.js	Gráficos e métricas
Processamento	csv-parser	Leitura de dados CSV
Agendamento	node-cron	Execução de tarefas periódicas
Testes	Axios	Cliente HTTP para testes

Slide 4: Estrutura Organizada do Projeto

```
cloudwalk-monitoring-test/
├── server.js           # * Servidor principal
├── package.json        # Configuração do projeto
├── data/               # [] Dados CSV
│   ├── transactions.csv
│   ├── transactions_auth_codes.csv
│   ├── checkout_1.csv
│   └── checkout_2.csv
├── public/             # [] Dashboard web
│   └── index.html
├── analysis/           # [] Análise de dados
│   └── dataAnalysis.js
├── tests/              # [] Testes
│   ├── api-tester.js
│   ├── quickTest.js
│   └── simpleTest.js
```

```
├── docs/                                # 📄 Documentação
│   ├── README.md
│   └── EXECUTION_SUMMARY.md
└── node_modules/                       # Dependências
```

Slide 5: Funcionalidades Implementadas

✓ Requisitos Obrigatórios Atendidos

1. Endpoint para Receber Transações

- POST `/api/transaction`
- Retorna recomendação: "normal" ou "alert"
- Detecta anomalias em tempo real

2. Queries SQL Otimizadas

- Múltiplas queries para análise de dados
- Agregações por status, horário e período
- Análise de séries temporais

3. Gráficos em Tempo Real

- Dashboard web responsivo
- Gráficos de status de transações
- Análise temporal de volume
- Comparação de checkouts por horário

4. Modelo de Detecção de Anomalias

- Algoritmo híbrido (rule-based + statistical)
- Z-score para detecção estatística
- Limiares dinâmicos por tipo de transação

5. Sistema de Alertas Automáticos

- Alertas em tempo real via WebSocket
- Histórico de alertas armazenado
- Níveis de severidade (low, medium, high)

Slide 6: APIs Disponíveis

Endpoints REST Implementados

Método	Endpoint	Descrição	Parâmetros
GET	/health	Status do servidor	-
GET	/api/stats	Estatísticas de transações	?hours=24
GET	/api/alerts	Alertas recentes	?limit=50
GET	/api/checkout-analysis	Análise de checkout	-
GET	/api/timeseries	Dados temporais	?hours=6
POST	/api/transaction	Enviar transação	JSON body

WebSocket

- **URL:** ws://localhost:8080
- **Função:** Broadcast de transações e alertas em tempo real

Slide 7: Detecção de Anomalias

■ Critérios de Alerta Configurados

Transações com Falha

- Alerta se taxa > 5% ou picos > 2.5x média
- Contagem mínima: 10 transações

Transações Negadas

- Alerta se taxa > 10% ou picos > 2.0x média
- Contagem mínima: 5 transações

Transações Revertidas

- Alerta se picos > 3.0x média
- Inclui reversões backend
- Contagem mínima: 8 transações

Checkouts Anômalos

- Volume > 2.5x média semanal ou 3x média mensal
- Quedas para zero quando esperado > 5
- Volumes < 30% da média quando esperado > 10

Metodologia Híbrida

1. **Rule-Based:** Limiares pré-definidos
2. **Statistical:** Z-score e médias móveis
3. **Contextual:** Padrões históricos e sazonais

Slide 8: Análise dos Dados

Principais Insights Identificados

Padrões Temporais

- **Horário de pico:** 10h-16h (horário comercial)
- **Baixa atividade:** 02h-06h (madrugada)
- **Padrões consistentes** entre dias da semana

Indicadores de Saúde

- **Taxa de aprovação:** >80% (saudável)
- **Taxa de falha:** <5% (aceitável)
- **Taxa de negação:** <10% (normal)
- **Taxa de reversão:** <3% (baixo risco)

Anomalias Críticas Detectadas

- **Checkout 2:** Períodos de zero atividade suspeitos às 15h-17h
- **Checkout 1:** Picos incomuns às 10h (55 vs média 29.42)
- **Divergências** entre os dois datasets de checkout

Slide 9: Queries SQL Implementadas

Exemplos de Queries Utilizadas

1. Resumo de Status de Transações

```
SELECT
  status,
  COUNT(*) as record_count,
  SUM(count) as total_transactions,
  AVG(count) as avg_per_minute,
  ROUND(SUM(count) * 100.0 /
    (SELECT SUM(count) FROM transactions), 2) as percentage
FROM transactions
GROUP BY status
ORDER BY total_transactions DESC;
```

2. Detecção de Anomalias em Falhas

```
WITH failed_stats AS (
  SELECT
    timestamp,
    count as failed_count,
    AVG(count) OVER (ORDER BY timestamp
      ROWS BETWEEN 10 PRECEDING AND CURRENT ROW) as moving_avg,
    STDDEV(count) OVER (ORDER BY timestamp
      ROWS BETWEEN 10 PRECEDING AND CURRENT ROW) as moving_stddev
  FROM transactions
  WHERE status = 'failed'
)
SELECT
  timestamp,
  failed_count,
  moving_avg,
  CASE
    WHEN failed_count > (moving_avg + 2 * moving_stddev)
      THEN 'HIGH_ANOMALY'
    WHEN failed_count > (moving_avg + moving_stddev)
      THEN 'MEDIUM_ANOMALY'
    ELSE 'NORMAL'
  END as anomaly_level
FROM failed_stats
WHERE moving_stddev > 0
ORDER BY failed_count DESC;
```

Slide 10: Dashboard e Visualizações

Interface do Dashboard

Métricas em Tempo Real

- Total de transações (24h)
- Taxa de aprovação
- Transações falhadas
- Alertas ativos

Gráficos Interativos

1. **Distribuição de Status:** Gráfico de pizza mostrando proporção de approved/failed/denied/reversed
2. **Volume Temporal:** Linha temporal mostrando volume de transações ao longo do tempo
3. **Análise de Checkout:** Comparação horária entre hoje, ontem e médias históricas

Sistema de Alertas

- Lista de alertas recentes
- Níveis de severidade com cores
- Timestamps e descrições detalhadas

Acessos

- **Dashboard:** <http://localhost:3000>
- **Health Check:** <http://localhost:3000/health>
- **WebSocket:** <ws://localhost:8080>

Slide 11: Como Executar o Sistema

Instalação e Execução

Pré-requisitos

- Node.js 16+
- NPM (ou Yarn)

Passos de Instalação

```
# Instalar dependências
npm install

# Iniciar servidor
```

```
npm start

# Executar testes
npm test

# Análise de dados
npm run analyze

# Verificar saúde
npm run health
```

Métodos de Inicialização

1. **NPM Script** (Recomendado): `npm start`
2. **Node direto**: `node server.js`
3. **Scripts de sistema**: `start.bat` (Windows) / `./start.sh` (Linux/Mac)

Slide 12: Testes e Validação

Sistema de Testes Implementado

Testes Automatizados

- **API Testing**: `tests/api-tester.js`
- **Testes Rápidos**: `tests/quickTest.js`
- **Testes Simples**: `tests/simpleTest.js`

Exemplo de Teste de API

```
const axios = require('axios');

async function testAPIs() {
  const base = 'http://localhost:3000';

  // Testar health check
  const health = await axios.get(`${base}/health`);

  // Testar estatísticas
  const stats = await axios.get(`${base}/api/stats?hours=24`);

  // Testar envio de transação
  const transaction = await axios.post(`${base}/api/transaction`, {
    timestamp: new Date().toISOString(),
    status: 'approved',
    count: 5
  });
}
```


Slide 13: Problemas Resolvidos

✓ Melhorias Implementadas

1. ✓ Servidor Estável

- Sem encerramento prematuro
- Error handling robusto

2. ✓ Dados Organizados

- CSVs organizados na pasta data/
- Carregamento automático na inicialização

3. ✓ Testes Estruturados

- Scripts organizados na pasta tests/
- Validação de todas as APIs

4. ✓ Documentação Completa

- README detalhado
- Guias de execução
- Documentação de APIs

5. ✓ Sistema Limpo

- Duplicatas removidas
- Apenas arquivos essenciais

6. ✓ Dashboard Funcional

- Dados em tempo real
- Interface responsiva
- Gráficos interativos

Slide 14: Recomendações Futuras

□ Próximos Passos e Melhorias

Melhorias Técnicas

1. **Persistência de Dados:** Migrar para PostgreSQL/MongoDB
2. **Escalabilidade:** Implementar Redis para cache
3. **Segurança:** Adicionar autenticação e autorização
4. **Monitoramento:** Integrar com Prometheus/Grafana

Funcionalidades Avançadas

1. **Machine Learning:** Modelos mais sofisticados de detecção
2. **Notificações:** Integração com Slack/Email/SMS
3. **Multi-tenancy:** Suporte a múltiplos clientes
4. **Histórico:** Análise de tendências de longo prazo

Operacional

1. **Deploy:** Containerização com Docker
2. **CI/CD:** Pipeline automatizado
3. **Logs:** Sistema estruturado de logging
4. **Backup:** Estratégia de backup e recovery

Slide 15: Conclusão

▮ Objetivos Alcançados

Desafio Completamente Atendido

- ✓ Análise exploratória dos dados CSV
- ✓ Sistema de monitoramento em tempo real
- ✓ Detecção automática de anomalias
- ✓ Dashboard interativo funcional
- ✓ APIs RESTful documentadas
- ✓ Sistema de alertas automático

Diferenciais Implementados

- **Arquitetura limpa** e bem organizada
- **Documentação completa** e detalhada
- **Testes automatizados** para validação
- **Interface moderna** e responsiva
- **Sistema híbrido** de detecção de anomalias

Status Final: ▮ SISTEMA 100% OPERACIONAL

Links Importantes

- **Repositório:** <https://github.com/fabenejr/cloudwalk-monitoring-test>
- **Dashboard:** <http://localhost:3000>
- **Documentação:** Disponível na pasta `/docs`

Obrigado!

Perguntas e Demonstração

Desenvolvido por: Fabio Nepomuceno Jr.

Para: CloudWalk - Monitoring Analyst Test

Tecnologias: Node.js, Express, SQLite, WebSocket, Chart.js

Status: Pronto para produção e deploy