Sistema de Governança de Dados V1.1 - Python 3.13

Visão Geral

O Sistema de Governança de Dados V1.1 é uma solução completa e moderna para governança corporativa de dados, desenvolvida especificamente para Python 3.13. O sistema oferece uma arquitetura de microserviços distribuída com 31 serviços especializados, 1042 endpoints REST e um modelo de dados robusto com 43 tabelas organizadas em 10 grupos funcionais.

Características Principais

Arquitetura Moderna

- **31 Microserviços** distribuídos nas portas 8000-8030
- **1042 Endpoints** REST com documentação OpenAPI automática
- **Compatibilidade** total com Python 3.13
- **Escalabilidade** horizontal e vertical
- **Alta disponibilidade** com observabilidade completa

Modelo de Dados Avançado

- **43 Tabelas** com campos de auditoria obrigatórios (created_at, updated_at)
- **Tipos modernos** (text, timestamptz) otimizados para PostgreSQL
- **10 Grupos funcionais** organizados por domínio
- **Comentários detalhados** sincronizados no modelo DBML
- **Scripts SQL** completos para criação e população

Funcionalidades Completas

- Contratos de dados com versionamento e SLAs
- Qualidade de dados com monitoramento contínuo
- Governança com políticas corporativas configuráveis
- Linhagem de dados com rastreamento end-to-end
- Catálogo de dados centralizado e descoberta automática

- Gerenciamento avançado de metadados
- Ciclo de vida dos dados com políticas de retenção/expurgo
- Compliance com LGPD, GDPR, SOX e outros frameworks
- Workflow de aprovação flexível e configurável
- Auditoria detalhada de todas as operações

Integrações Nativas

- **Databricks Unity Catalog** - Sincronização de metadados
- **Informatica Axon** - Governança empresarial
- **Azure Active Directory** - Autenticação corporativa
- **PostgreSQL** - Banco de dados principal
- **Redis** - Cache e sessões
- **Azure Service Bus** - Mensageria
- **Azure Event Hubs** - Streaming de eventos

Pré-requisitos

Sistema Operacional

- Linux (Ubuntu 20.04+ recomendado)
- Windows 10/11 (com WSL2)
- macOS 10.15+

Software Obrigatório

- **Python 3.13** (versão exata)
- **PostgreSQL 13+**
- **Redis 6+**
- **Git** para versionamento

Recursos Mínimos

- **CPU**: 4 cores
- **RAM**: 8GB (16GB recomendado)
- **Disco**: 20GB livres

- **Rede**: Conexão estável com internet

Instalação Rápida

1. Preparação do Ambiente

Bash

```bash
# Verificar Python 3.13
python3.13 --version

# Instalar PostgreSQL
sudo apt update
sudo apt install postgresql postgresql-contrib

# Instalar Redis
sudo apt install redis-server

# Iniciar serviços
sudo systemctl start postgresql
sudo systemctl start redis-server
```

2. Configuração do Banco de Dados

Bash

```bash
# Criar usuário e banco
sudo -u postgres createuser governance_user
sudo -u postgres createdb governance_system
sudo -u postgres psql -c "ALTER USER governance_user WITH PASSWORD
'governance_pass';"
sudo -u postgres psql -c "GRANT ALL PRIVILEGES ON DATABASE governance_system
TO governance_user;"
```

3. Instalação do Sistema

Bash

```bash
# Extrair pacote
tar -xzf sistema-governanca-v1.1-python-313.tar.gz
cd SISTEMA_GOVERNANCA_V2_1_PYTHON_3_13

# Instalar dependências
pip3.13 install -r requirements.txt
```

```
# Configurar ambiente
cp .env.example .env
# Editar .env com suas configurações

# Criar estrutura do banco
cd database/scripts
./run_scripts.sh
cd ../..
```

4. Execução do Sistema

Bash

```
# Executar sistema completo
python3.13 main.py

# Ou executar serviços individuais
cd apps/api-gateway
python3.13 main.py &

cd ../identity-service
python3.13 main.py &
```

Configuração Detalhada

Arquivo .env

Plain Text

```
# Banco de Dados
DATABASE_URL=postgresql://governance_user:governance_pass@localhost:5432/gove
rnance_system
REDIS_URL=redis://localhost:6379/0

# Segurança
SECRET_KEY=your-secret-key-here
JWT_SECRET=your-jwt-secret-here

# Azure (opcional)
AZURE_CLIENT_ID=your-azure-client-id
AZURE_CLIENT_SECRET=your-azure-client-secret
AZURE_TENANT_ID=your-azure-tenant-id

# Databricks (opcional)
```

```
DATABRICKS_HOST=your-databricks-host
DATABRICKS_TOKEN=your-databricks-token

# Informatica Axon (opcional)
AXON_HOST=your-axon-host
AXON_USERNAME=your-axon-username
AXON_PASSWORD=your-axon-password
```

Configuração de Portas

O sistema utiliza as seguintes portas por padrão:

- **8000**: API Gateway (principal)
- **8001**: Identity Service
- **8002**: Audit Service
- **8003**: Contract Service
- **8004**: Catalog Service
- **8005**: Quality Service
- **8006**: Governance Service
- **8007**: Lineage Service
- **8008-8030**: Demais microserviços

Verificação da Instalação

1. Health Check

Bash

```bash
# Verificar API Gateway
curl http://localhost:8000/health

# Verificar documentação
open http://localhost:8000/docs
```

2. Teste de Funcionalidades

Bash

```bash
# Executar testes automatizados
python3.13 test_all_endpoints_comprehensive.py
```

```bash
# Validar dados mock
python3.13 validate_solution_with_mock_data.py
```

3. Verificar Logs

Bash

```bash
# Logs do sistema
tail -f logs/system.log

# Logs de microserviços
tail -f logs/microservices.log
```

Estrutura do Projeto

Plain Text

```
SISTEMA_GOVERNANCA_V2_1_PYTHON_3_13/
├── apps/                          # 31 microserviços
│   ├── api-gateway/              # Gateway principal
│   ├── identity-service/         # Autenticação
│   ├── contract-service/         # Contratos de dados
│   ├── catalog-service/          # Catálogo de dados
│   ├── quality-service/          # Qualidade de dados
│   └── ...                       # Demais serviços
├── config/                       # Configurações
│   └── settings.py              # Configuração centralizada
├── database/                     # Modelo de dados
│   ├── modelo_estendido.dbml    # Modelo DBML completo
│   └── scripts/                 # Scripts SQL
├── docs/                        # Documentação
│   ├── DOCUMENTO_TECNICO_COMPLETO.md
│   ├── DOCUMENTO_FUNCIONAL_COMPLETO.md
│   └── DOCUMENTO_GERENCIAL_COMPLETO.md
├── fluxos_drawio/              # Diagramas
├── evidencias_teste/           # Evidências de teste
├── mock_data/                  # Dados de teste
├── main.py                     # Executor principal
├── requirements.txt            # Dependências
└── .env                        # Configurações de ambiente
```

Uso do Sistema

Interface Web

Acesse `http://localhost:8000` para a interface principal:

- **Dashboard**: Visão geral do sistema
- **Catálogo**: Explorar datasets disponíveis
- **Contratos**: Gerenciar contratos de dados
- **Qualidade**: Monitorar métricas de qualidade
- **Linhagem**: Visualizar fluxo de dados
- **Governança**: Configurar políticas

API REST

Documentação completa disponível em `http://localhost:8000/docs`:

- **1042 endpoints** organizados por microserviço
- **Autenticação JWT** para segurança
- **Rate limiting** para proteção
- **Versionamento** de APIs
- **Documentação OpenAPI** automática

SDK e Integrações

Python

```python
# Exemplo de uso do SDK Python
from governance_sdk import GovernanceClient

client = GovernanceClient(base_url="http://localhost:8000")

# Criar contrato de dados
contract = client.contracts.create({
    "name": "Customer Data Contract",
    "schema": {...},
    "quality_rules": [...]
})

# Monitorar qualidade
quality_report = client.quality.get_report(dataset_id="customers")
```

Monitoramento e Observabilidade

Métricas Disponíveis

- **Performance**: Latência, throughput, disponibilidade
- **Qualidade**: Completude, consistência, precisão
- **Uso**: Acessos, usuários ativos, datasets populares
- **Compliance**: Violações, auditorias, conformidade

Dashboards

- **Executivo**: KPIs de alto nível
- **Operacional**: Métricas técnicas
- **Qualidade**: Indicadores de dados
- **Compliance**: Status regulatório

Troubleshooting

Problemas Comuns

Erro de conexão com banco:

```bash
# Verificar se PostgreSQL está rodando
sudo systemctl status postgresql

# Testar conexão
psql -h localhost -U governance_user -d governance_system
```

Microserviço não inicia:

```bash
# Verificar logs
tail -f logs/microservices.log

# Verificar porta em uso
netstat -tulpn | grep :8000
```

Erro de dependências:

```bash
```

```
# Reinstalar dependências
pip3.13 install -r requirements.txt --force-reinstall

# Verificar versão Python
python3.13 --version
```

Logs e Diagnóstico

Bash

```
# Logs detalhados
export LOG_LEVEL=DEBUG
python3.13 main.py

# Verificar saúde do sistema
python3.13 -c "from config.settings import Settings;
print(Settings().dict())"
```

Próximos Passos

Implementação por Fases

Fase 1 (6-8 semanas): Contratos de Dados + Unity Catalog

- Implementar contratos básicos
- Configurar sincronização Unity Catalog
- Treinar equipe inicial

Fase 2 (8-10 semanas): Qualidade e Governança

- Configurar regras de qualidade
- Implementar políticas corporativas
- Estabelecer workflows de aprovação

Fase 3 (10-12 semanas): Linhagem e Analytics

- Mapear linhagem completa
- Implementar dashboards executivos
- Configurar alertas automáticos

Fase 4 (6-8 semanas): Segurança Avançada

- Implementar classificação automática

- Configurar controles de acesso granulares
- Estabelecer auditoria completa

Evolução Arquitetural

Event-Driven Architecture: Migração planejada para arquitetura orientada a eventos para melhor performance e escalabilidade.

Machine Learning: Integração de modelos para classificação automática e detecção de anomalias.

Multi-Cloud: Suporte a múltiplos provedores de nuvem para alta disponibilidade.

Suporte e Documentação

Documentação Completa

- **Técnica**: `docs/DOCUMENTO_TECNICO_COMPLETO.md`
- **Funcional**: `docs/DOCUMENTO_FUNCIONAL_COMPLETO.md`
- **Gerencial**: `docs/DOCUMENTO_GERENCIAL_COMPLETO.md`
- **Instruções**: `INSTRUCOES_COMPLETAS_V1_1.md`

Recursos Adicionais

- **Diagramas**: `fluxos_drawio/` - Arquitetura e fluxos
- **Apresentações**: Slides para diferentes públicos
- **Evidências**: `evidencias_teste/` - Relatórios de teste
- **Mock Data**: `mock_data/` - Dados de exemplo

Contato e Suporte

Para suporte técnico e dúvidas sobre implementação, consulte a documentação técnica completa ou entre em contato com a equipe de desenvolvimento.

Sistema de Governança de Dados V1.1

Desenvolvido para Python 3.13

Validado e testado em ambiente limpo