Documento de Engenharia de Software – TrustShield

Este documento apresenta de forma sofisticada os princípios, padrões e práticas de engenharia de software que guiarão o desenvolvimento, implantação e manutenção do projeto **TrustShield** (Detecção e Prevenção de Fraudes em Tempo Real). Ele abrange desde arquitetura, qualidade de código, automação, segurança, até monitoramento e otimização contínua.

---

1. Visão Arquitetural

1.1 Estilo Arquitetural

- **Microserviços Orientados a Domínio**: cada componente (ingestão, processamento, detecção, inferência, monitoramento) expõe APIs independentes.
- **Event-Driven Architecture**: troca de mensagens via broker (Kafka/RabbitMQ) para desacoplamento e escalabilidade.
- **Pipelines DataOps**: uso de frameworks (Airflow/Kubeflow) para orquestração de fluxos ETL, feature engineering e treinamento.

1.2 Componentes Principais

1. **API Gateway**: roteamento inteligente, autenticação (OAuth2/JWT) e rate limiting.
2. **Serviço de Ingestão**: captura transações em tempo real, validação de schema (Avro/Protobuf) e publicação em tópicos.
3. **Serviço de Pré-Processamento**: limpeza, enriquecimento (mapeamento MCC), normalização e persistência em data lake (S3/HDFS).
4. **Engine de Features**: construção de features em streaming e batch, armazenadas em Feature Store (Feast).
5. **Módulo de Modelos**:
6. **Treino**: notebooks + jobs escaláveis em Kubernetes/GPU.
7. **Inferência**: model server (TensorFlow Serving / TorchServe) com quantização e autoscaling.
8. **Sistema de Alerta**: regras determinísticas e sugestões do ML ensemble para bloqueio ou alerta.
9. **Dashboard & Monitoramento**: Grafana / Kibana para métricas de saúde (latência, throughput) e performance (AUC, F1).
10. **Pipeline de CI/CD**: GitOps com ArgoCD e GitHub Actions / Jenkins.

---

2. Qualidade de Código e Padrões

2.1 Convenções de Código

- **PEP 8** para Python: indentação, nomes de variáveis, docstrings.
- **Arquitetura Hexagonal**: separação clara entre domínio, aplicação e infraestrutura.
- **DDD (Domain-Driven Design)**: entidades, agregados, repositórios e serviços de domínio.

2.2 Padrões de Design

- **Factory e Abstract Factory** para criação de instâncias de modelos e conectores de dados.
- **Strategy** para seleção dinâmica de algoritmos de detecção (Random Forest, XGBoost, redes neurais).
- **Observer / Event Sourcing** para auditabilidade de decisões de fraude.
- **Circuit Breaker** para proteção de serviços externos (SMS, e-mail).

2.3 Revisão de Código

- **Pull Requests** obrigatórios com mínimo de 2 revisores.
- **Checks automatizados**: lint (flake8), análise estática (Bandit, SonarQube), cobertura de testes (> 85%).
- **Pair Programming** em módulos críticos (segurança, inferência em tempo real).

---

3. Gerenciamento de Dependências e Ambientes

- **Conda** / **virtualenv** + `requirements.txt` versionado.
- **Dockerfiles** multistage: build leve e runtime otimizado.
- **Docker Compose** para ambientes locais (Kafka, PostgreSQL, MinIO).
- **Helm Charts** para deploy Kubernetes: configuração parametrizada por `values.yaml`.

---

4. Testes e Validação

4.1 Tipos de Testes

- **Unitários**: pytest com fixtures para simular dados (mocking de conexões).
- **Integração**: testes end-to-end com contêineres (Kafka, banco de dados).
- **Testes de Contrato**: Pact para garantir compatibilidade de APIs entre serviços.
- **Testes de Performance**: locust / k6 para medir throughput e latência em picos.

4.2 Automação de Testes

- Pipeline CI executa todos os testes em ambiente isolado.
- **Test Coverage** report automatizado e falha de build se abaixo do mínimo.

---

5. Configuração e Secret Management

- **12-Factor App**: configurações externas via variáveis de ambiente.
- **HashiCorp Vault** ou AWS Secrets Manager: armazenamento seguro de credenciais.
- **ConfigMaps** e **Secrets** no Kubernetes, integrados com RBAC.

---

6. Segurança e Compliance

- **Criptografia em Trânsito**: TLS 1.2+ para todas as comunicações.

- **Criptografia em Repouso**: dados sensíveis (logs, bancos) cifrados.
- **Políticas de Acesso**: IAM roles e policies mínimas (principle of least privilege).
- **Auditoria**: registro de eventos críticos e acesso a dados (ELK Stack).
- **Compliance**: aderência a LGPD/GDPR, revisões semestrais de privacidade.

---

7. Observabilidade e Monitoramento

- **Métricas Customizadas**: via Prometheus para contagem de transações, detecções e erros.
- **Logs Estruturados**: JSON com trace IDs, enviados para ELK ou Grafana Loki.
- **Tracing Distribuído**: OpenTelemetry para rastrear fluxos de eventos entre serviços.
- **Alertas**: regras no Grafana Alertmanager para latência alta, falhas de inferência, drift de dados.

---

8. Deploy e Infraestrutura

- **Infra as Code**: Terraform para provisionar VPC, clusters Kubernetes, buckets.
- **Blue/Green Deployments** ou **Canary Releases** para minimizar riscos de downtime.
- **Rollback Automático** em caso de falha de health checks.
- **Cluster Autoscaling** baseado em CPU/Memory e custom metrics (lag do Kafka).

---

9. Manutenção e Evolução Contínua

- **Pipeline de Retraining**: agendamento automático de re-treino com novos dados rotulados.
- **Feature Store Versioning**: controle de versões de features para reprodutibilidade.
- **Documentação Viva**: uso de MkDocs/Sphinx integrado ao repositório.
- **Governança de Dados**: catálogo de dados (Data Catalog) com linha de tempo de consumidores.

---

10. Métricas de Sucesso e SLAs

- **SLA de Inferência:** < 200 ms p/ request em 95º percentil.
- **Taxa de Detecção:** Recall $\geq$ 90% e Precision $\geq$ 85%.
- **Disponibilidade:** $\geq$ 99.9% dos serviços principais.
- **Taxa de Alerta Falso Positivo:** $\leq$ 2%.

Conclusão: Este documento consolida as melhores práticas de engenharia de software, garantindo que o projeto TrustShield seja robusto, seguro, escalável e mantenha excelência operacional ao longo de toda a sua vida útil.