Prompt Profissional: Sistema de Gerenciamento de Inventário Distribuído - Enterprise Grade

Contexto Detalhado do Projeto

Situação Atual da Empresa

Você é um arquiteto de software sênior contratado para resolver problemas críticos em uma **cadeia de varejo enterprise** com:

- **500+ lojas físicas** distribuídas nacionalmente
- **50,000+ produtos** no catálogo ativo
- **Sistema legado monolítico** rodando há 8 anos
- **Arquitetura obsoleta** sem padrões modernos de distribuição

Problemas Críticos do Sistema Atual

1. Arquitetura Legacy Problemática

- **Backend monolítico** em Java 8 + Spring Boot 1.x (desatualizado)
- **Banco de dados único** PostgreSQL 9.6 centralizado
- **Frontend web legacy** sem responsividade adequada
- **Sincronização manual** via batch jobs a cada 15 minutos
- **Zero observabilidade** - logs básicos apenas

2. Problemas de Consistência de Dados (CRÍTICO)

- **Discrepâncias de estoque**: Clientes veem produtos disponíveis que não existem
- **Overselling**: Vendas de produtos sem estoque suficiente
- **Underselling**: Produtos disponíveis marcados como esgotados
- **Conflitos de reserva**: Múltiplas lojas reservando o mesmo item
- **Perda de transações**: 15% das vendas online canceladas por inconsistência

3. Problemas de Performance e Latência

- **Consultas de estoque**: 2-5 segundos de resposta
- **Atualizações de inventário**: Delay de 15-45 minutos
- **Picos de tráfego**: Sistema trava durante promoções/Black Friday
- **Timeout errors**: 30% das consultas falham em horário de pico
- **Escalabilidade zero**: Não consegue adicionar novas lojas facilmente

4. Impactos no Negócio (Números Reais)

- **Perda de receita**: R$ 2.5M/mês por inconsistências de estoque
- **Experiência do cliente**: NPS caiu de 7.8 para 4.2 em 2 anos
- **Custos operacionais**: Time de suporte gasta 40h/semana resolvendo discrepâncias
- **Reputação**: Reclamações no Reclame Aqui aumentaram 300%
- **Competitive disadvantage**: Concorrentes com sistemas real-time ganhando market share

5. Limitações Técnicas Atuais

- **Sem distributed transactions**: Falha em operações entre múltiplas lojas
- **Sem event sourcing**: Impossível rastrear mudanças ou fazer audit
- **Sem caching strategy**: Toda consulta vai ao banco principal
- **Sem fault tolerance**: Uma falha derruba todo o sistema
- **Sem API versioning**: Mudanças quebram integrações existentes

6. Cenários Problemáticos Específicos

Cenário A - Black Friday:

- Sistema recebe 50x mais tráfego que normal
- Database locks causam timeout em 80% das transações
- Clientes abandonam carrinho por lentidão
- Perda estimada: R$ 8M em um dia

Cenário B - Nova Promoção:

- Marketing lança promoção de produto específico
- 50 lojas tentam reservar estoque simultaneamente
- Race conditions causam overselling de 300%
- Necessário cancelar vendas e compensar clientes

Cenário C - Operação Normal:

- Cliente consulta estoque online: "5 unidades disponíveis"
- Vai à loja física: "Produto esgotado há 2 dias"
- Sistema ainda não sincronizou a informação
- Cliente insatisfeito, venda perdida

Requisitos de Negócio para Nova Solução

Stakeholder Requirements

- **CEO**: ROI positivo em 12 meses, redução de custos operacionais
- **CTO**: Arquitetura moderna, escalável e observável
- **Head de Vendas**: Zero overselling, inventory real-time
- **Customer Experience**: Tempo de resposta < 200ms
- **Operations**: Zero-downtime deployments, self-healing system

SLAs Obrigatórios

- **Availability**: 99.9% uptime (máximo 8h downtime/ano)
- **Performance**: < 100ms para consultas de estoque
- **Consistency**: Zero discrepâncias entre canais online/offline
- **Scalability**: Suportar 100% crescimento sem degradação
- **Recovery**: RTO < 15min, RPO < 1min

Constraints e Limitações

Técnicas

- **Migration gradual**: Sistema atual deve continuar funcionando durante transição
- **Budget limitation**: Não pode contratar 50+ desenvolvedores
- **Timeline**: Solução MVP em 6 meses, completa em 12 meses
- **Compliance**: LGPD compliance obrigatório

Organizacionais

- **Team size**: 8 desenvolvedores Java disponíveis
- **Knowledge gap**: Time atual não tem experiência com microservices
- **Legacy integration**: Deve integrar com ERP SAP existente
- **Change management**: Resistência interna a mudanças grandes

Objetivos Principais

1. **Resolver problemas de consistência** através de arquitetura distribuída robusta
2. **Reduzir latência** de atualizações de estoque para < 100ms
3. **Diminuir custos operacionais** através de arquitetura eficiente
4. **Garantir observabilidade enterprise** com Dynatrace integration
5. **Aplicar Clean Architecture** com design patterns industriais

Stack Tecnológica Enterprise

Core Framework

- **Backend**: Java 17+ com Spring Boot 3.x
- **Arquitetura**: Clean Architecture com Domain-Driven Design (DDD)

Estratégia de Banco de Dados (CRÍTICO para Consistência)

- **Write Database**: PostgreSQL com particionamento por região/store
- **Read Database**: PostgreSQL read replicas + Redis para cache de consultas frequentes
- **Event Store**: Apache Kafka como event log para Event Sourcing
- **Distributed Coordination**: Redis Cluster para distributed locks e coordination
- **Time-Series Data**: InfluxDB para métricas de inventário e analytics
- **Search**: Elasticsearch para consultas complexas de produtos

Patterns de Consistência de Dados

- **Event Sourcing**: Para auditoria completa e replay de eventos
- **CQRS**: Separação total entre commands (writes) e queries (reads)
- **Saga Pattern**: Para transações distribuídas entre múltiplas stores
- **Distributed Locking**: Redis-based locks para operações críticas
- **Eventually Consistent Reads**: Com strong consistency para writes

Message Broker & Communication

- **Apache Kafka**: Event streaming para real-time inventory updates
- **Redis Streams**: Para notificações em tempo real
- **Apache Pulsar**: Alternativa ao Kafka para casos específicos

Observabilidade Enterprise (Dynatrace Integration)

- **APM**: Dynatrace como plataforma principal de observabilidade
- **Metrics Exposure**: Micrometer com Dynatrace registry (NÃO Prometheus)
- **Distributed Tracing**: OpenTelemetry com Dynatrace OneAgent integration
- **Custom Business Metrics**: Micrometer Timer, Counter, Gauge para KPIs
- **Structured Logging**: Logback JSON format para Dynatrace log ingestion
- **Real User Monitoring**: Dynatrace RUM para frontend performance

Ferramentas Complementares

- **API Documentation**: Swagger/OpenAPI 3

- **Testing**: JUnit 5, TestContainers, Mockito, ArchUnit para architecture tests
- **Build**: Maven com profiles enterprise
- **Containerization**: Docker + Kubernetes para produção

Arquitetura Clean Architecture + DDD

Estrutura de Projeto (Clean Architecture)

```
inventory-management-system/
├── shared/
│   ├── domain-events/        # Eventos compartilhados entre bounded contexts
│   ├── common/               # Utilities e cross-cutting concerns
│   └── observability/        # Dynatrace integration modules
├── inventory-service/
│   ├── domain/               # Domain Layer - Pure Business Logic
│   ├── application/          # Application Layer - Use Cases & Orchestration
│   ├── infrastructure/       # Infrastructure Layer - External Concerns
│   └── presentation/         # Presentation Layer - API Controllers
├── store-service/            # Mesma estrutura Clean Architecture
├── notification-service/
├── api-gateway/
├── observability/
│   ├── dynatrace-config/
│   └── custom-metrics/
└── deployment/
```

Design Patterns Obrigatórios (Enterprise Grade)

Domain Layer Patterns

- **Aggregate Pattern**: Garantir consistência transacional
- **Repository Pattern**: Abstração para persistência
- **Domain Events**: Comunicação entre bounded contexts
- **Value Objects**: Modelagem rica do domínio
- **Specification Pattern**: Regras de negócio complexas

Application Layer Patterns

- **Use Case Pattern**: Operações de negócio específicas
- **CQRS**: Command Query Responsibility Segregation
- **Saga Pattern**: Transações distribuídas de longa duração
- **Unit of Work**: Gerenciamento de transações
- **Port/Adapter (Hexagonal)**: Isolamento de dependências externas

Infrastructure Layer Patterns

- **Circuit Breaker**: Resilience4j para fault tolerance
- **Retry Pattern**: Políticas de retry com backoff exponencial
- **Bulkhead Pattern**: Isolamento de recursos críticos
- **Cache-Aside Pattern**: Estratégias de cache inteligente
- **Outbox Pattern**: Garantir entrega de eventos

Cross-Cutting Patterns

- **Decorator Pattern**: Observabilidade e logging
- **Chain of Responsibility**: Pipeline de validações
- **Strategy Pattern**: Algoritmos de sincronização
- **Factory Pattern**: Criação de objetos complexos
- **Observer Pattern**: Reação a eventos de domínio

Requisitos Funcionais Detalhados

Core Domain Operations

1. **Real-time Stock Reservation**: Reserva atômica com timeout
2. **Multi-store Inventory Sync**: Sincronização eventual consistente
3. **Concurrent Stock Updates**: Handling de updates simultâneos
4. **Stock Replenishment**: Algoritmos de reposição inteligente
5. **Inventory Reconciliation**: Correção de discrepâncias automática

Performance Requirements

- **Latência**: < 100ms para operações críticas
- **Throughput**: > 10,000 operações/segundo
- **Availability**: 99.9% uptime
- **Consistency**: Strong consistency para writes críticos
- **Cache Hit Rate**: > 95% para consultas de estoque

Observabilidade Requirements (Dynatrace Specific)

- **Custom Business Metrics**: Inventory levels, reservation rates, sync failures
- **Distributed Tracing**: End-to-end transaction visibility
- **Real-time Alerting**: Dynatrace alerts para anomalias de inventário
- **Service Dependencies**: Mapeamento automático via OneAgent

- **Performance Baselines**: Dynatrace AI para detection de anomalias

Cenários de Uso Críticos para Resolver

Problema 1: Concurrent Updates

- **Solução**: Optimistic locking + Event sourcing + Conflict resolution strategies

Problema 2: Network Partitions

- **Solução**: Saga pattern + Compensation transactions + Local-first approach

Problema 3: Consistency vs Availability

- **Solução**: Hybrid approach - Strong consistency para reservas, eventual para consultas

Problema 4: High-Traffic Scenarios

- **Solução**: Cache layers + Read replicas + Load balancing + Circuit breakers

Problema 5: Cross-Store Inventory Visibility

- **Solução**: Event-driven sync + CQRS views + Real-time notifications

Estratégias de Consistência de Dados

Write Path (Strong Consistency)

- **Distributed Locks**: Redis-based para operações críticas
- **Optimistic Concurrency**: Version-based conflict detection
- **Event Sourcing**: Complete audit trail e replay capability
- **Saga Orchestration**: Para workflows de múltiplas etapas

Read Path (Eventual Consistency)

- **CQRS Views**: Read models otimizados
- **Multi-level Caching**: L1 (local) + L2 (Redis) + L3 (CDN)
- **Read Replicas**: Geographic distribution
- **Cache Invalidation**: Event-driven cache updates

Entregáveis Esperados

1. Código Base Completo

- **Clean Architecture** implementation completa
- **Design Patterns** implementados corretamente
- **SOLID Principles** aplicados consistentemente

- **DDD Practices** com bounded contexts claros

2. Observabilidade Enterprise

- **Dynatrace Integration** completa com custom metrics
- **OpenTelemetry** setup para distributed tracing
- **Structured Logging** para análise automatizada
- **Health Checks** customizados para business logic

3. Testes Abrangentes

- **Unit Tests**: > 80% coverage
- **Integration Tests**: TestContainers para cenários reais
- **Architecture Tests**: ArchUnit para validar clean architecture
- **Performance Tests**: JMeter scenarios para load testing

4. Documentação Enterprise

- **Architecture Decision Records (ADRs)**: Para decisões técnicas
- **API Documentation**: OpenAPI 3 specification
- **Runbook**: Operational procedures
- **Deployment Guide**: Kubernetes manifests

5. DevOps & Deployment

- **Docker**: Multi-stage builds otimizados
- **Kubernetes**: Manifests para produção
- **CI/CD**: Pipeline completo com quality gates
- **Infrastructure as Code**: Terraform para cloud resources

Critérios de Avaliação

Qualidade Técnica

- **Clean Architecture**: Separação clara de responsabilidades
- **Design Patterns**: Aplicação apropriada e consistente
- **Code Quality**: SonarQube metrics + clean code practices
- **Performance**: Benchmarks demonstrando melhorias

Funcionalidade

- **Problem Resolution**: Solução clara para problemas de consistência
- **Scalability**: Demonstração de capacidade de escalar

- **Resilience**: Fault tolerance e recovery mechanisms
- **Business Value**: Métricas que demonstram melhorias

Observabilidade

- **Dynatrace Integration**: Custom dashboards e alerts
- **Operational Metrics**: KPIs de negócio monitoráveis
- **Troubleshooting**: Capacidade de debug em produção
- **Performance Monitoring**: APM insights acionáveis

IMPORTANTE: Este sistema deve demonstrar expertise em arquitetura de sistemas distribuídos enterprise, com foco especial na resolução dos problemas de consistência através de patterns e tecnologias adequadas. A integração com Dynatrace deve ser nativa e fornecer visibilidade completa do comportamento do sistema em produção.