Documento Técnico — Infraestrutura de um MicroSaaS Contêinerizado (Docker) com Multi-Tenancy, Segurança, Rotinas Globais e Design Limpo

Objetivo

- Descrever uma arquitetura completa, pronta para produção, para um microSaaS em que o cliente apenas escolhe o nicho (domínio) a ser atendido.
- Abrange: contêineres (Docker), orquestração (Compose com caminho de crescimento para Kubernetes), controle de usuários e multi-tenant, rotinas globais (backup, observabilidade, migrações, billing, etc.), segurança e proteção de dados (LGPD/GDPR), banco relacional e vetorial, além de diretrizes de design de UI limpo e minimalista.

Princípios

- Segurança por padrão e por design.
- Multi-tenancy elástico, com isolamento lógico entre clientes.
- Modularidade: "kits de nicho" (plugins) que ativam dados, prompts, fluxos e layouts específicos por segmento.
- Observabilidade completa (logs, métricas, rastros).
- Infra-as-code, deploy reprodutível e rollback simples.
- Experiência mínima viável para iniciar com Docker Compose e evoluir para Kubernetes quando necessário.

Visão Geral da Arquitetura

Componentes

- Frontend: Next.js/React (SSR/SSG) + Tailwind/Chakra — design limpo e acessível (WCAG AA), tema claro/escuro.
- API (Backend): Node.js (NestJS/Express) ou Python (FastAPI) — módulos: auth, tenants, billing, users, niches, conteúdo, busca vetorial, relatórios.
- Banco relacional: PostgreSQL 16 (RLS ativável para multi-tenancy).
- Banco vetorial: pgvector (extensão no Postgres) ou Qdrant (opcional). Recomendação inicial: pgvector para simplificar backup/operacional.
- Cache/Rate limit/Filas leves: Redis 7 (com Redis Streams) ou RabbitMQ (opcional se necessário).
- Objeto/arquivos: MinIO (compatível com S3) para anexos e exportações.
- Proxy/Edge: Traefik (TLS, ACME, roteamento, rate limit, compressão).
- Observabilidade: Prometheus (métricas), Grafana (dashboards), Loki (logs), Tempo/Jaeger (tracing), OpenTelemetry nos serviços.
- Jobs/Workers: processadores assíncronos para embeddings, ETL, notificações, e-mails, billing (webhooks).

- CI/CD: GitHub Actions (build, testes, scan, SBOM, push de imagens, deploy).
- IaC: Terraform (rede, buckets, DB gerenciado, balanceador, TLS).
- Feature flags e configuração: Unleash/ConfigCat/LaunchDarkly (ou simples por BD/ENV).
- Billing: Stripe (assinaturas, metered billing, webhooks).
- E-mails/Comms: Postmark/SendGrid + WhatsApp/Telegram (opcional) via webhooks/filas.

Fluxo de Solicitação (alto nível)

- Request → Traefik → API → Auth + Tenant Resolver → Aplicação do contexto (tenant + nicho) → Regras de acesso (RBAC/ABAC) → Dados (Postgres) + Busca vetorial (pgvector) + Cache (Redis) → Resposta.
- Rotinas assíncronas: API envia mensagens → queue (Redis Streams/RabbitMQ) → workers (embeddings, ETL, notificações, faturamento).

Multi-Tenancy e "Niches" (Domínios)

Estratégia recomendada (default)

- Multi-tenant por colunas com Row Level Security (RLS) no Postgres:
    - Todas as tabelas de dados possuem tenant_id (UUID).
    - Política RLS garante que cada usuário só lê/escreve linhas do seu tenant.
    - Tabela de "membros" (user_id, tenant_id, role) com roles: owner, admin, analyst, viewer.
    - Vantagens: menos sobrecarga de migrações e pooling que schema-per-tenant, permite escalabilidade inicial mais simples.

Alternativa

- Schema-per-tenant (quando clientes exigem isolamento mais forte ou performance previsível). Requer automação de provisionamento de schemas e migrações por tenant.

Niches (kits de domínio)

- "Packs" versionados por nicho, contendo:
    - Configuração YAML/JSON: taxonomias, prompt templates, formulários, validações, workflows.
    - Recursos UI (ícones, páginas, componentes específicos).
    - Mapeamentos para ingestão de dados e pipelines (incluindo campos de pessoas — gênero, profissão, estado civil, nacionalidade — e metadados de origem/UF).
- Habilitados por tenant (tenant_niches) para que o administrador selecione o segmento.

Segurança e Proteção de Dados (LGPD/GDPR-ready)

Autenticação e Autorização

- Password hashing: Argon2id (ou scrypt com parâmetros fortes).
- MFA/TOTP opcional por tenant/usuário.
- OAuth2/OIDC com provedores (Google/Microsoft) e SSO corporativo (opcional).
- Sessões curtas e refresh tokens rotacionados; revogação em logout e alteração de senha.
- RBAC granular (entidade, ação). ABAC opcional (atributos de niche/tenant).
- Rate limiting por IP, por usuário, e por rota sensível (Traefik e Redis).

Proteção de Dados

- Criptografia em trânsito: TLS obrigatório (HSTS, TLS 1.2+).
- Criptografia em repouso:
    - Postgres em disco (criptografia de volume) + colunas sensíveis com criptografia de campo (pgcrypto) quando necessário.
    - MinIO com SSE.
- Segredos: nunca em repositório; usar variáveis de ambiente e/ou gerenciadores (Vault/KMS).
- Minimização: coletar apenas o imprescindível; mascarar PII em logs; logs de auditoria com retenção definida.
- DPO, Aviso de Privacidade, Política de Retenção e Procedimentos de Atendimentos de Solicitações de Titulares (acesso, correção, exclusão).
- Geolocalização de dados: parametrizar região; se necessário, isolar dados por região.
- Backups cifrados, testados periodicamente (DR test).

Hardening de Aplicação e Edge

- CSP, X-Content-Type-Options, X-Frame-Options, Referrer-Policy.
- Proteção CSRF (rotas stateful), sanitização de inputs, validação de payloads.
- SSRF e RCE: bloqueios de egress, noProxy para metadados cloud, não executar uploads.
- Contêineres sem root, imagens minimalistas, read-only FS onde possível, capabilities mínimas.
- SAST/DAST/Dependabot/Syft/Grype; assinaturas de imagens e SBOM.
- Chaves rotacionadas; auditoria de acessos administrativos e produção.

Conformidade Operacional

- Matriz de controle de acesso (least privilege) para times e serviços.
- Retenção de logs por 90 a 180 dias (ajustável por plano).
- Subprocessadores documentados e acordos DPA com terceiros.

Rotinas Globais

- Migrações: Prisma/Drizzle/Knex (Node) ou Alembic (Python). Política “migrate on startup” controlada por flag.
- Backups: Postgres (PITR com WAL), MinIO (versões), Redis (RDB snapshot). Rotação e testes de restauração.
- Reindex e VACUUM: agendados semanalmente; ANALYZE pós-carga massiva.
- Rebuild de embeddings: fila dedicada, idempotente (com hashing do conteúdo).
- Limpeza/retention: purge de dados expirados e arquivos órfãos.
- Custos: alarmes de uso (S3/MinIO, egress, CPU/RAM), budget alerts.
- Observabilidade: dashboards pré-prontos (latência, p95/p99, erros, uso de CPU/RAM/IO, fila).

Design Limpo e Minimalista

- UI: tipografia simples, espaçamento generoso, cores neutras com 1 cor de acento (modo claro/escuro).
- Acessibilidade: contraste adequado, foco visível, navegação por teclado, labels e ARIA.
- Componentes reutilizáveis: botões, inputs, cards, listas, tabelas, empty states, feedback (toasts/modais).
- Documentação de microcópia (textos claros, sem jargão).
- Performance: SSR/ISR, code splitting, imagens otimizadas, cache HTTP, CDN.
- Internacionalização pronta (i18n) e “white-label” por tenant (logo/cores).

Estrutura de Pastas (exemplo)

/app

/frontend

/src

/components

/pages

/layouts

/styles

/lib

/niches # UI e configs por nicho

/backend

/src

/modules

/auth

/users

/tenants

/billing

/niches

/search # vetorial + keyword

/files

/webhooks

/common

/jobs

/config

/migrations

/scripts

/workers

/embeddings

/ingestion

/deploy

docker-compose.yml

/k8s

/terraform

/observability

/grafana-dashboards

.github/workflows

Variáveis de Ambiente (exemplo)

Core

NODE_ENV=production

APP_PORT=8080

```
APP_BASE_URL=https://app.suaempresa.com

# Postgres
POSTGRES_HOST=postgres
POSTGRES_PORT=5432
POSTGRES_DB=appdb
POSTGRES_USER=appuser
POSTGRES_PASSWORD=supersecret

# Redis
REDIS_HOST=redis
REDIS_PORT=6379
REDIS_PASSWORD=redispass

# MinIO
MINIO_ENDPOINT=minio:9000
MINIO_ACCESS_KEY=minioadmin
MINIO_SECRET_KEY=minioadmin
MINIO_BUCKET=app-bucket

# Auth
JWT_ACCESS_TTL=900
JWT_REFRESH_TTL=2592000
JWT_SECRET=change_me
PASSWORD_PEPPER=change_me_2

# Stripe
STRIPE_SECRET_KEY=sk_live_xxx
STRIPE_WEBHOOK_SECRET=whsec_xxx

# Observability
```

OTEL_EXPORTER_OTLP_ENDPOINT=http://otel-collector:4317

Feature flags

FEATURE_FLAGS_PROVIDER=local

Dockerfiles (exemplos)

Backend (Node.js + NestJS)

Stage 1 - Build

FROM node:20-alpine AS build

WORKDIR /app

RUN addgroup -S app && adduser -S app -G app

COPY backend/package*.json ./

RUN npm ci --include=dev

COPY backend ./backend

WORKDIR /app/backend

RUN npm run build

Stage 2 - Runtime

FROM gcr.io/distroless/nodejs20-debian12

WORKDIR /app

USER 1000:1000

ENV NODE_ENV=production

COPY --from=build /app/backend/dist ./dist

COPY --from=build /app/backend/node_modules ./node_modules

EXPOSE 8080

CMD ["dist/main.js"]

Frontend (Next.js)

Build

FROM node:20-alpine AS build

WORKDIR /app

COPY frontend/package*.json ./

RUN npm ci

COPY frontend ./frontend

WORKDIR /app/frontend

RUN npm run build

Runtime (nginx estático ou next start)

FROM node:20-alpine

WORKDIR /app/frontend

ENV NODE_ENV=production

COPY --from=build /app/frontend ./

EXPOSE 3000

CMD ["npm", "run", "start"]

Workers (embeddings com Node/Python, exemplo Node)

FROM node:20-alpine

WORKDIR /app

COPY workers/package*.json ./

RUN npm ci

COPY workers ./workers

CMD ["node", "workers/embeddings/index.js"]

Observações

- Executar como usuário não-root.
- Imagens minimalistas, sem ferramentas desnecessárias.
- Ativar scans de vulnerabilidade de imagens e gerar SBOM.

docker-compose.yml (produção pequena / staging)

version: "3.9"

services:

traefik:

image: traefik:v3.0

command:

- "--api.dashboard=true"

- "--providers.docker=true"

```
    - "--entrypoints.web.address=:80"
    - "--entrypoints.websecure.address=:443"
    - "--certificatesresolvers.le.acme.tlschallenge=true"
    - "--certificatesresolvers.le.acme.email=devops@suaempresa.com"
    - "--certificatesresolvers.le.acme.storage=/letsencrypt/acme.json"
  ports:
    - "80:80"
    - "443:443"
  volumes:
    - letsencrypt:/letsencrypt
    - /var/run/docker.sock:/var/run/docker.sock:ro

postgres:
  image: postgres:16-alpine
  environment:
    POSTGRES_DB: appdb
    POSTGRES_USER: appuser
    POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
  volumes:
    - pgdata:/var/lib/postgresql/data
  healthcheck:
    test: ["CMD-SHELL", "pg_isready -U appuser -d appdb"]

# Habilitar pgvector no init
postgres-init:
  image: postgres:16-alpine
  depends_on: [postgres]
  command: >
    sh -c "
    until pg_isready -h postgres -U appuser; do sleep 1; done &&
    psql postgresql://appuser:${POSTGRES_PASSWORD}@postgres:5432/appdb
```

```
    -c 'CREATE EXTENSION IF NOT EXISTS vector;'
    "
  environment:
    POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}

redis:
  image: redis:7-alpine
  command: ["redis-server", "--save", "60", "1", "--loglevel", "warning"]
  volumes:
    - redisdata:/data

minio:
  image: minio/minio:latest
  command: server /data --console-address ":9001"
  environment:
    MINIO_ROOT_USER: ${MINIO_ACCESS_KEY}
    MINIO_ROOT_PASSWORD: ${MINIO_SECRET_KEY}
  ports:
    - "9000:9000"
    - "9001:9001"
  volumes:
    - miniodata:/data

backend:
  build:
    context: .
    dockerfile: deploy/Dockerfile.backend
  env_file:
    - .env
  depends_on:
    - postgres
```

```
    - redis
    - minio
  labels:
    - "traefik.enable=true"
    - "traefik.http.routers.api.rule=Host(`api.suaempresa.com`)"
    - "traefik.http.routers.api.entrypoints=websecure"
    - "traefik.http.routers.api.tls.certresolver=le"
    - "traefik.http.middlewares.api-ratelimit.rateLimit.average=100"
    - "traefik.http.middlewares.api-ratelimit.rateLimit.burst=200"
    - "traefik.http.routers.api.middlewares=api-ratelimit@docker"

frontend:
  build:
    context: .
    dockerfile: deploy/Dockerfile.frontend
  env_file:
    - .env
  depends_on:
    - backend
  labels:
    - "traefik.enable=true"
    - "traefik.http.routers.web.rule=Host(`app.suaempresa.com`)"
    - "traefik.http.routers.web.entrypoints=websecure"
    - "traefik.http.routers.web.tls.certresolver=le"

worker-embeddings:
  build:
    context: .
    dockerfile: deploy/Dockerfile.worker
  env_file:
    - .env
```

```
  depends_on:
    - backend
    - postgres
    - redis

prometheus:
  image: prom/prometheus:latest
  volumes:
    - ./observability/prometheus.yml:/etc/prometheus/prometheus.yml:ro
  depends_on:
    - backend

grafana:
  image: grafana/grafana:latest
  ports:
    - "3001:3000"
  depends_on:
    - prometheus

loki:
  image: grafana/loki:2.9.0
  command: -config.file=/etc/loki/local-config.yaml
  ports:
    - "3100:3100"

promtail:
  image: grafana/promtail:2.9.0
  volumes:
    - /var/log:/var/log
    - /var/lib/docker/containers:/var/lib/docker/containers:ro
    - ./observability/promtail-config.yml:/etc/promtail/config.yml:ro
```

```
  command: -config.file=/etc/promtail/config.yml
  depends_on:
   - loki

volumes:
 letsencrypt:
 pgdata:
 redisdata:
 miniodata:
```

Notas

- Para produção gerenciada, substitua Postgres/Redis/MinIO por serviços gerenciados (RDS/Cloud SQL, Memorystore/ElastiCache, S3).
- Para banco vetorial dedicado, pode-se usar Qdrant em contêiner separado (ou serviço gerenciado) — caso o volume vetorial cresça significativamente.

Banco de Dados: Relacional e Vetorial

- PostgreSQL com RLS (multi-tenant).
- pgvector para embeddings (CREATE EXTENSION vector).
- Índices:
    - B-Tree para chaves e buscas exatas.
    - GIN/Trigram para buscas textuais.
    - ivfflat/HNSW (pgvector) para similaridade.
- Criptografia de campo para PII sensível, quando necessário (pgcrypto).
- Backup PITR e planos de retenção.

Exemplo de políticas RLS (pseudocode SQL)

```
ALTER TABLE public.documents ENABLE ROW LEVEL SECURITY;

CREATE POLICY tenant_isolation ON public.documents
 USING (tenant_id = current_setting('app.tenant_id')::uuid);

-- Antes de cada request, a API seta:
-- SET app.tenant_id = '<tenant_uuid>';
```

Observação: Use connection pool (pgBouncer) quando escalar ainda mais.

Controle de Usuários e Acesso

- Fluxos:
    - Cadastro com verificação de e-mail.
    - Convite por tenant, aceitação via token.
    - Esqueci minha senha (tokens de uso único).
    - MFA opcional (TOTP).
- RBAC:
    - owner (billing e tudo), admin (gerencia usuários/dados), analyst (lê/escreve dados operacionais), viewer (somente leitura).
- Auditoria:
    - Tabela audit_log (quem fez, o quê, quando, onde — IP, user agent).
    - Exportável por tenant.
- Rate limiting por plano:
    - Limites de requisições/min, jobs/hora, armazenamento, e tamanho de upload.

Billing (Assinaturas e Planos)

- Stripe:
    - Planos: Starter, Pro, Enterprise (limites diferentes).
    - Metered billing: contagem de tokens/consultas/armazenamento.
    - Webhooks: container “webhooks” com verificação de assinatura e idempotência.
- Dunning, trial, cupom, faturas, notas fiscais (com integrações locais se necessário).
- Feature flags por plano.

Observabilidade

- OpenTelemetry em backend e workers.
- Prometheus: métricas (latência API, taxa de erro, consumo Redis, tempo de fila, CPU/RAM).
- Grafana: dashboards prontos (SLO p95/p99).
- Loki/Promtail: logs estruturados (JSON), com PII mascarada.
- Alertas: uptime (heartbeats), erro 5xx, backlog de filas, espaço disco, reconstrução de embeddings atrasada.

CI/CD (GitHub Actions — exemplo)

```yaml
name: ci-cd

on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]

jobs:
  build-test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with:
          node-version: '20'
      - name: Install deps
        run: npm ci --workspaces
      - name: Lint
        run: npm run lint --workspaces
      - name: Test
        run: npm test --workspaces
      - name: Build
        run: npm run build --workspaces
      - name: SBOM
        uses: anchore/sbom-action@v0
        with:
          path: .

  docker-publish:
    needs: build-test
```

```
runs-on: ubuntu-latest
permissions:
 contents: read
 packages: write
steps:
 - uses: actions/checkout@v4
 - name: Login to GHCR
  uses: docker/login-action@v3
  with:
   registry: ghcr.io
   username: ${{ github.actor }}
   password: ${{ secrets.GITHUB_TOKEN }}
 - name: Build & Push backend
  uses: docker/build-push-action@v6
  with:
   context: .
   file: deploy/Dockerfile.backend
   push: true
   tags: ghcr.io/suaorg/app-backend:latest
 - name: Build & Push frontend
  uses: docker/build-push-action@v6
  with:
   context: .
   file: deploy/Dockerfile.frontend
   push: true
   tags: ghcr.io/suaorg/app-frontend:latest
 - name: Build & Push worker
  uses: docker/build-push-action@v6
  with:
   context: .
   file: deploy/Dockerfile.worker
```

push: true

tags: ghcr.io/suaorg/app-worker:latest

- Opcional: step de segurança (Trivy/Grype) e assinar imagens (cosign).
- Deploy: usar ambiente de staging antes de produção, com gates manuais.

Infraestrutura como Código (Terraform – visão)

- Módulos:
    - Rede (VPC, sub-redes privadas/públicas).
    - Banco gerenciado (RDS/Cloud SQL Postgres).
    - Redis gerenciado (ElastiCache/Memorystore).
    - Storage (S3) com políticas.
    - Balanceador (ALB/NLB) + ACM.
    - Secrets (KMS/Vault).
    - Observabilidade (Managed Grafana/Cloud Logging).
- Saídas (outputs) integradas ao CI para preencher .env de produção.

Rotinas Operacionais (Runbooks)

- Incidente de indisponibilidade:
    - Checar Traefik, upstream, latência DB, fila de jobs, picos de CPU/RAM.
    - Rollback com imagens anteriores.
- Vazamento de segredos:
    - Revogar chaves, rotacionar tokens, invalidar sessões.
- Restaurar backup:
    - Checklist de RTO/RPO; testes mensais simulados.
- Rebuild de índices vetoriais:
    - Job dedicado, rate limitado, com progress tracking e métricas.
- Migrações:
    - Plano com dry-run em staging; toggle “migrate_on_startup=true” com backoff.

Evolução para Kubernetes (quando escalar)

- Benefícios: autoscaling, deploy canário, isolamentos, secrets gerenciados, storage classes.
- Recursos:
    - Deployments para backend, frontend, workers.

    - Ingress Controller (Traefik/NGINX).
    - HPA com métricas de CPU/latência.
    - Secrets (Kubernetes Secrets + KMS).
    - Service Mesh (Linkerd/Istio) para mTLS interno e observabilidade.
- Ferramentas: Helm charts, ArgoCD/Flux para GitOps.

Checklist de Boas Práticas

Segurança

- TLS em todos os domínios; HSTS.
- Políticas CSP e rate limiting.
- RLS ativado e testado; colunas tenant_id em todas as tabelas de dados.
- Hash de senha com Argon2id; MFA opcional.
- Logs sem PII; mascaramento de e-mails e CPFs nos eventos.
- Segredos fora do repositório; chaves rotacionadas.
- Contêineres sem root; imagens minimalistas; scans ativos.
- Backups testados e cifrados.

Confiabilidade

- Observabilidade com painéis de SLO (p95/p99).
- Healthchecks e readiness/liveness.
- Retentativa e idempotência em workers.
- Filas dimensionadas e monitoradas.

Escalabilidade

- Cache adequado (Redis), e tag de invalidação.
- Uso de CDN para assets.
- Banco vetorial com índices apropriados; partição de dados quando crescer.

Produto

- “Kits de nicho” versionados e testados.
- RBAC/ABAC prontos para customização por plano.
- Onboarding guiado; empty states úteis.
- Internacionalização e white-label por tenant.

Exemplos de Scripts Úteis

Script simples de backup diário (Postgres + MinIO)

```
#!/usr/bin/env bash

set -euo pipefail

DATE=$(date +"%Y%m%d-%H%M")

FILE="pgdump-$DATE.sql.gz"


PGPASSWORD="$POSTGRES_PASSWORD" pg_dump -h $POSTGRES_HOST -U $POSTGRES_USER -d $POSTGRES_DB | gzip > /tmp/$FILE


mc alias set minio http://$MINIO_ENDPOINT $MINIO_ACCESS_KEY $MINIO_SECRET_KEY

mc cp /tmp/$FILE minio/$MINIO_BUCKET/backups/$FILE

mc retention set --days 30 minio/$MINIO_BUCKET/backups/

rm /tmp/$FILE

echo "Backup concluído em $DATE"
```

Job de reindex vetorial (pseudo-código Node)

```
// Recria embeddings quando a versão do modelo muda

for await (const doc of streamDocsNeedingRebuild(modelVersion)) {

 const emb = await embedder.encode(doc.text);

 await db.query(

  `UPDATE documents SET embedding=$1, embedding_model=$2, embedding_updated_at=now()

   WHERE id=$3 AND tenant_id=$4`,

   [emb, modelVersion, doc.id, doc.tenant_id]

 );

}
```

Considerações Específicas ao Domínio (Nicho)

- O sistema deve permitir que o “tenant admin” escolha e ative um ou mais kits de nicho:
    - Define campos e validações específicas de pessoas (ex.: gênero, profissão, estado civil, nacionalidade), estados de origem e de identidade (UF), além de taxonomias de domínio.
    - Ajusta formulários e relatórios padrão.
    - Habilita buscas vetoriais e filtros específicos (ex.: perfis por UF/papel).

- Cada kit mantém versionamento para possibilitar mudanças sem quebrar tenants existentes.
- Conteúdos e embeddings são segregados por tenant e por nicho (p.ex., coleções vetoriais com chaves (tenant_id, niche_id)).

Roadmap de Lançamento

Fase 1 (MVP com Compose)

- Backend, Frontend, Postgres+pgvector, Redis, MinIO, Traefik.
- Autenticação básica + RBAC + RLS.
- Um kit de nicho inicial para validação.
- Observabilidade básica (Prometheus/Grafana, Loki).

Fase 2 (Produção inicial)

- Stripe (planos/limites), webhooks.
- Backups PITR configurados e testados.
- MFA, políticas CSP, scans CI.
- Pipeline de embeddings assíncrono e busca vetorial robusta.

Fase 3 (Escala)

- Infra gerenciada (RDS, S3, Redis gerenciado).
- GitOps/Kubernetes, HPA, canários.
- Feature flags por plano, kits múltiplos e marketplace interno de nichos.
- Auditoria avançada, exportações de dados e relatórios.

Fase 4 (Enterprise)

- SSO corporativo, SCIM.
- ABAC, masking dinâmico, data residency.
- DR multi-região, RTO/RPO baixos.

Conclusão

Este blueprint cobre, ponta a ponta, como erguer um microSaaS contêinerizado, seguro, observável e extensível por nicho — começando com Docker Compose e com um caminho claro para Kubernetes. A combinação Postgres + pgvector simplifica operações iniciais, enquanto Redis, MinIO, Traefik e a pilha de observabilidade garantem fundamentos sólidos. Com RLS, RBAC e boas práticas de segurança, você atende LGPD/GDPR e prepara o terreno para escalar com eficiência.

Se quiser, posso gerar:

- Templates completos dos kits de nicho (estrutura YAML/JSON).
- Scripts SQL de RLS por tabela do seu domínio.

- Dashboards do Grafana prontos para colar.
- Helm Charts para migração a Kubernetes quando chegar a hora.