Minecraft Server Manager - Análise Técnica e Roadmap

Objetivo Geral do Projeto

Dashblock é uma solução de gerenciamento remoto de servidores Minecraft que permite aos usuários criar, configurar, monitorar e controlar múltiplos servidores Minecraft através de uma interface web intuitiva. O sistema utiliza conexões SSH para gerenciar servidores hospedados remotamente, oferecendo funcionalidades completas de administração sem necessidade de acesso direto ao servidor.

Casos de Uso Principais:

- **Hosting Providers**: Empresas que oferecem hospedagem de servidores Minecraft
- **Game Communities**: Comunidades que gerenciam múltiplos servidores temáticos
- **Administradores**: Pessoas que precisam gerenciar vários servidores de forma centralizada
- **Desenvolvedores**: Testagem de mods e configurações em diferentes ambientes

---

Arquitetura Atual (MVP)

Stack Tecnológico

- **Backend**: Python com Litestar (framework web assíncrono)
- **SSH Management**: Fabric (biblioteca Python para SSH)
- **Process Management**: tmux (para sessões persistentes dos servidores)
- **Cache**: Sistema simples em memória com TTL
- **CORS**: Configurado para frontend local

Características Técnicas

- **Connection Pooling**: Reutilização de conexões SSH para performance
- **Thread Safety**: Operações SSH executadas em threads separadas
- **Caching**: Cache com TTL de 30 segundos para listagens de diretório
- **Error Handling**: Tratamento robusto de erros SSH e de rede

---

Documentação dos Endpoints

1. Gerenciamento de Arquivos Remotos

GET /list-directory

Funcionalidade: Lista conteúdo de diretórios remotos via SSH

- **Parâmetros**: `path` (opcional, default: "~")
- **Retorno**: Lista de arquivos/diretórios com permissões
- **Cache**: 30 segundos TTL
- **Uso**: Navegação pelo sistema de arquivos do servidor

`POST /change-directory`

Funcionalidade: Navega entre diretórios remotos

- **Body**: `current_path`, `target_name`
- **Retorno**: Novo conteúdo do diretório
- **Uso**: Interface de navegação tipo explorador de arquivos

`GET /list-home` (Legacy)

Funcionalidade: Lista diretório home (compatibilidade)

- **Retorno**: Array simples com nomes dos arquivos
- **Status**: Mantido para compatibilidade com versões anteriores

2. Lifecycle dos Servidores

`POST /create-server`

Funcionalidade: Criação automatizada de novos servidores Minecraft

- **Body**: `server_name`, `version`
- **Processo**:
  1. Cria diretório do servidor
  2. Download do server.jar oficial
  3. Aceita EULA automaticamente
  4. Gera server.properties padrão
  5. Cria script de inicialização
- **Retorno**: Status e caminho do servidor criado

`POST /start-server`

Funcionalidade: Inicia servidor em sessão tmux

- **Body**: `server_name`
- **Verificações**: Existência do servidor, se já está rodando
- **Processo**: Cria sessão tmux dedicada executando o servidor
- **Retorno**: Status da operação e session ID

`POST /stop-server`

Funcionalidade: Para servidor de forma segura

- **Body**: `server_name`
- **Processo**:
  1. Envia comando "stop" via tmux
  2. Aguarda 3 segundos
  3. Força fechamento da sessão se necessário
- **Retorno**: Status da operação

3. Monitoramento e Status

`GET /server-status/{server_name}`

Funcionalidade: Status completo de um servidor específico

- **Parâmetros**: `server_name` na URL
- **Retorno**:
  - Status de execução (tmux session ativa)
  - Existência do servidor
  - Informações detalhadas (tipo, versão, mods, IP, porta)
- **Detecção Automática**: Identifica Vanilla, Fabric, Forge

`GET /list-servers`

Funcionalidade: Lista todos os servidores disponíveis

- **Processo**: Busca por arquivos server.jar no sistema
- **Retorno**: Array com informações completas de cada servidor
- **Dados**: Nome, caminho, status, tipo, versão, contagem de mods

4. Configuração de Servidores

`GET /server-properties/{server_name}`

Funcionalidade: Lê configurações do servidor

- **Retorno**: Objeto com todas as propriedades parseadas
- **Formato**: Chave-valor das configurações do Minecraft

`POST /server-properties/{server_name}`

Funcionalidade: Atualiza configurações do servidor

- **Body**: `properties` (objeto chave-valor)
- **Processo**: Reescreve arquivo server.properties
- **Uso**: Configuração de MOTD, dificuldade, limite de jogadores, etc.

Funcionalidades Inteligentes

Detecção Automática de Servidor

O sistema detecta automaticamente:

- **Tipo**: Vanilla, Fabric, Forge
- **Versão**: Extrai de version.json ou manifesto do JAR
- **Mods**: Conta arquivos .jar na pasta mods
- **Configurações**: Porta personalizada, IP de bind

Sistema de Cache Inteligente

- **Cache de Diretórios**: Evita múltiplas chamadas SSH
- **Connection Pooling**: Reutiliza conexões por 5 minutos
- **Thread Safety**: Operações seguras em ambiente multi-thread

Error Handling Robusto

- **SSH Timeouts**: Timeouts configuráveis para operações
- **Connection Recovery**: Reconexão automática em caso de falha
- **Graceful Degradation**: Valores padrão quando detecção falha

---

Recomendações Tecnológicas para o Projeto Futuro

Stack Recomendada

Frontend

Next.js 14+ (App Router)

- **Razões**:
  - Server-Side Rendering para melhor SEO
  - API Routes integradas
  - Excelente developer experience
  - TypeScript nativo
  - Otimizações automáticas de performance

UI Framework:

- **shadcn/ui + Tailwind CSS**: Componentes modernos e customizáveis
- **Framer Motion**: Animações suaves para feedback visual
- **React Query/TanStack Query**: Gerenciamento de estado servidor

Backend

Node.js com Fastify ou Express

- **Fastify** (Recomendado):
  - Performance superior ao Express
  - Validation/Serialization integrados
  - Plugin ecosystem robusto
  - TypeScript first-class support

Alternativas modernas:

- **Bun**: Runtime JavaScript ultrarrápido
- **Deno**: Segurança e TypeScript nativo

Gerenciamento SSH

node-ssh + ssh2

- **node-ssh**: Interface high-level para SSH
- **ssh2**: Controle low-level quando necessário
- **ssh2-sftp-client**: Transfer de arquivos

Database & Cache

PostgreSQL + Redis

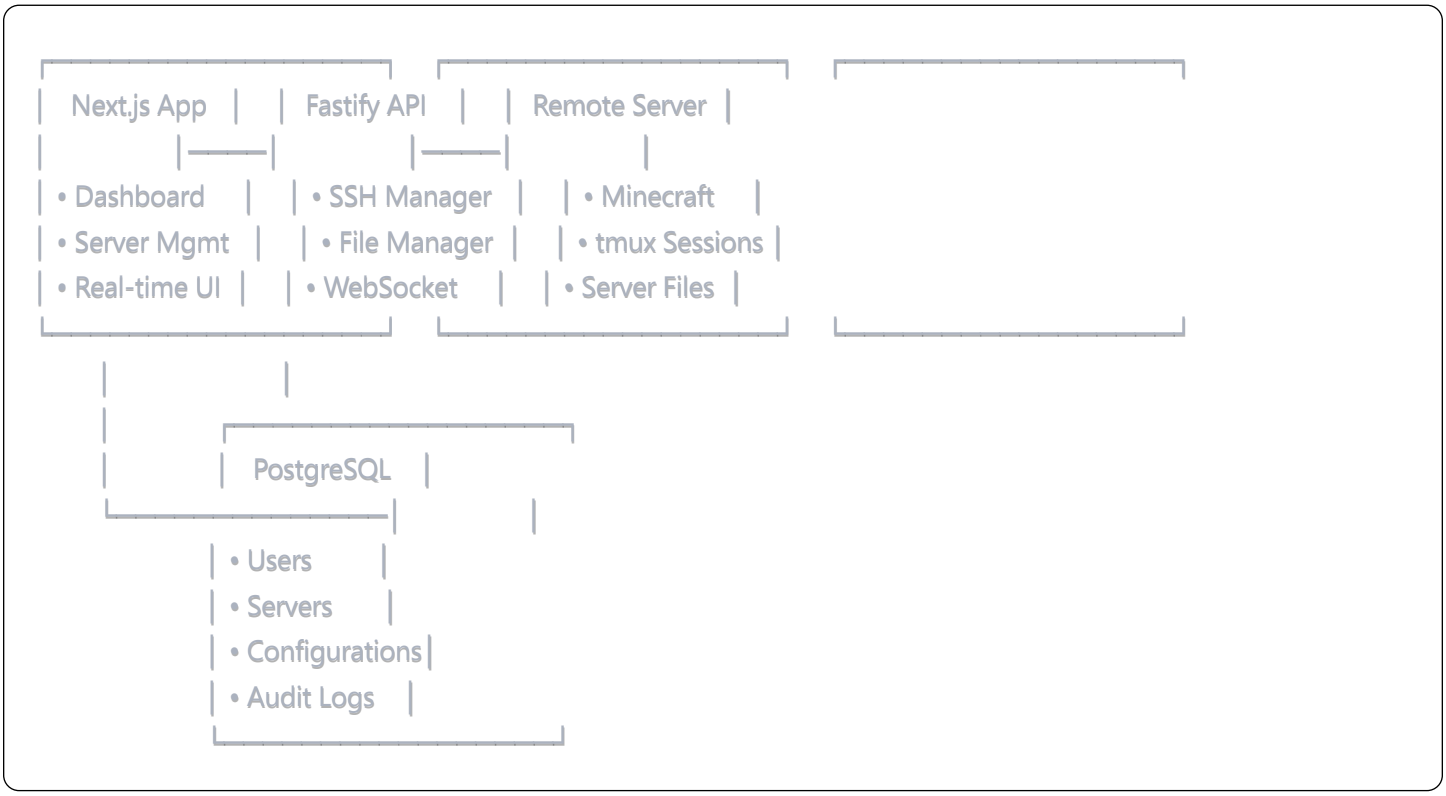
- **PostgreSQL**: Dados relacionais (configurações, usuários, logs)
- **Redis**: Cache e session management
- **Prisma**: ORM moderno com excelente TypeScript support

Real-time Features

WebSockets (Socket.io)

- **Server logs em tempo real**
- **Status updates instantâneos**
- **Chat integration com servidores**

Arquitetura Proposta



Funcionalidades Expandidas

Autenticação & Autorização

- **NextAuth.js**: Múltiplos providers (Google, Discord, GitHub)
- **RBAC**: Role-based access control (Admin, Moderator, User)
- **Multi-tenancy**: Suporte a múltiplas organizações

Monitoramento Avançado

- **Métricas em tempo real**: CPU, RAM, players online
- **Alertas**: Notificações via webhook/email
- **Logs centralizados**: Agregação e pesquisa de logs
- **Performance analytics**: Gráficos de performance histórica

Automação

- **Scheduled tasks**: Backups automáticos, restarts programados
- **Auto-scaling**: Start/stop baseado em players online
- **Plugin management**: Install/update automático de plugins
- **World management**: Backup/restore de mundos

Interface Moderna

- **Dashboard interativo**: Gráficos em tempo real
- **File manager**: Editor de código integrado
- **Console web**: Terminal web para comandos diretos
- **Mobile responsive**: Design mobile-first

Estrutura do Projeto

```
minecraft-manager/
├── apps/
│   ├── web/              # Next.js frontend
│   └── api/              # Fastify backend
├── packages/
│   ├── ui/               # Shared UI components
│   ├── database/         # Prisma schema & utilities
│   ├── ssh/              # SSH management utilities
│   └── types/            # Shared TypeScript types
├── tools/
│   ├── eslint-config/    # Shared ESLint config
│   └── typescript-config/  # Shared TS config
└── deploy/
    ├── docker/           # Docker configurations
    └── kubernetes/       # K8s manifests
```

Roadmap de Desenvolvimento

Fase 1: Core Migration (4-6 semanas)

1. **Setup do monorepo** (Turbo/Nx)
2. **API Backend básica** (Fastify + TypeScript)
3. **Migração dos endpoints** existentes
4. **Frontend básico** (Next.js + shadcn/ui)
5. **Autenticação simples**

Fase 2: Enhanced Features (6-8 semanas)

1. **Database integration** (PostgreSQL + Prisma)
2. **Real-time features** (WebSockets)
3. **File management** avançado
4. **Sistema de logs**
5. **Mobile responsiveness**

Fase 3: Advanced Features (8-10 semanas)

1. **Métricas e monitoring**
2. **Sistema de alertas**
3. **Backup automation**
4. **Plugin management**
5. **Multi-server orchestration**

Fase 4: Enterprise Features (10-12 semanas)

1. **Multi-tenancy**
2. **Advanced RBAC**
3. **API rate limiting**
4. **Audit logging**
5. **Enterprise integrations**

Considerações de Produção

Segurança

- **SSH Key Management**: Vault ou similar para chaves
- **Network Security**: VPN/Private networks
- **Input Validation**: Sanitização rigorosa de inputs
- **Rate Limiting**: Proteção contra abuse

Escalabilidade

- **Horizontal scaling**: Load balancing da API
- **Connection pooling**: Pool de conexões SSH otimizado
- **Caching strategy**: Redis para cache distribuído
- **Database optimization**: Índices e queries otimizadas

Observabilidade

- **Logging**: Structured logging (Winston/Pino)
- **Metrics**: Prometheus + Grafana
- **Tracing**: OpenTelemetry
- **Health checks**: Endpoints de saúde da aplicação

Este documento serve como base técnica para o desenvolvimento da versão completa do sistema, mantendo a funcionalidade core do MVP atual enquanto expande significativamente as capacidades e a experiência do usuário.