UNIVERSIDADE ESTADUAL PAULISTA JÚLIO DE MESQUITA FILHO

FACULDADE DE CIÊNCIAS

BACHARELADO EM SISTEMAS DE INFORMAÇÃO

Trabalho de Conclusão de Curso

Autor
Bruno Pirozzi Venancio

Orientador
Kelton Augusto Pontara Costa

VIGIA

Visualização Integrada de Gestão e Inteligência de Anomalias

Bauru, 2025

UNIVERSIDADE ESTADUAL PAULISTA JÚLIO DE MESQUITA FILHO

FACULDADE DE CIÊNCIAS

BACHARELADO EM SISTEMAS DE INFORMAÇÃO

Trabalho de Conclusão de Curso

Autor
Bruno Pirozzi Venancio

Orientador
Kelton Augusto Pontara Costa

VIGIA

Visualização Integrada de Gestão e Inteligência de Anomalias

Trabalho apresentado como exigência para a conclusão do Curso de Bacharelado em Sistemas de Informação da Faculdade de Ciências – UNESP
Campus Bauru.

Bauru, 2025

BRUNO PIROZZI VENANCIO

V448v

Venancio, Bruno Pirozzi
VIGIA : Visualização Integrada de Gestão e Inteligência de Anomalias / Bruno Pirozzi Venancio. -- Bauru, 2025
61 p. : fotos

Trabalho de conclusão de curso (Bacharelado - Sistemas de Informação) - Universidade Estadual Paulista (UNESP), Faculdade de Ciências, Bauru
Orientador: Kelton Augusto Pontara Costas

1. Monitoramento de Servidores. 2. Sistema de Recomendação. 3. Gerenciamento de Incidentes. 4. Automação de TI. 5. Zabbix. I. Título.

Sistema de geração automática de fichas catalográficas da Unesp. Dados fornecidos pelo autor(a).

Bauru, 2025

RESUMO

Este trabalho apresenta o desenvolvimento do VIGIA (Visualização Integrada de Gestão e Inteligência de Anomalias), um sistema web de monitoramento proativo de servidores Linux integrado ao Zabbix, voltado para pequenas e médias empresas. O sistema foi desenvolvido para automatizar a detecção, análise e resolução de incidentes em ambientes de TI, reduzindo o tempo de resposta a problemas críticos e minimizando o impacto operacional. A arquitetura implementa uma aplicação web responsiva utilizando React no *backend* e Node.js no *backend*, com banco de dados MySQL para persistência de dados. A integração com a API do Zabbix 7.4 permite sincronização automática de alertas a cada 2 minutos, processando eventos de múltiplos servidores simultaneamente. O diferencial do sistema reside no mecanismo de recomendação inteligente de soluções, que utiliza análise de similaridade baseada em palavras-chave para identificar as três melhores soluções para cada incidente, classificando-as por score de compatibilidade (0-100%) em três categorias: match exato (≥80%), parcial (40-79%) e similar (<40%). O sistema implementa notificações multi-canal via Telegram Bot API e email SMTP, garantindo que a equipe de TI seja alertada imediatamente sobre incidentes críticos. Durante os testes, o sistema demonstrou capacidade de processar 127 incidentes ao longo de 6 meses, com taxa de resolução de 92,1% e tempo médio de resposta de 42 minutos. A interface web oferece *dashboards* em tempo real com visualização de métricas de CPU e memória, histórico completo de incidentes resolvidos e ativos, e um sistema de busca e filtros avançados por severidade e servidor. Os resultados obtidos comprovam a eficácia do sistema em automatizar processos de monitoramento e resolução de problemas, contribuindo significativamente para a redução do tempo de inatividade e melhoria da disponibilidade dos serviços de TI em ambientes corporativos.

Palavras-chave: Monitoramento de Servidores. Zabbix. Sistema de Recomendação. Gerenciamento de Incidentes. Automação de TI. Node.js. React.

ABSTRACT

This work presents the development of VIGIA (Integrated Visualization for Management and Anomaly Intelligence), a proactive web-based monitoring system for Linux servers integrated with Zabbix and designed for small and medium-sized *enterprise*s. The system was created to automate the detection, analysis, and resolution of IT incidents, reducing response times to critical issues and minimizing operational impact. The architecture consists of a responsive web application developed with React on the *backend* and Node.js on the *backend*, supported by a MySQL database for data persistence. Integration with the Zabbix 7.4 API enables automatic synchronization of alerts every two minutes, allowing the processing of events from multiple servers simultaneously. The system's key innovation lies in its intelligent solution recommendation engine, which uses keyword-based similarity analysis to identify the three most relevant resolutions for each incident. These recommendations are ranked by compatibility score (0–100%) across three categories: exact match (≥80%), partial match (40–79%), and similar (<40%). VIGIA also implements multi-channel notifications through the Telegram Bot API and SMTP email, ensuring that IT teams are promptly alerted to critical incidents. During testing, the system successfully processed 127 incidents over a six-month period, achieving a resolution rate of 92.1% and an average response time of 42 minutes. The web interface provides real-time *dashboards* displaying CPU and memory metrics, a complete history of active and resolved incidents, and advanced search and filtering capabilities by severity and server. The results demonstrate the system's effectiveness in automating monitoring and incident resolution workflows, significantly reducing *downtime* and improving service availability in corporate IT environments.

Keywords: Server Monitoring. Zabbix. Recommendation System. Incident Management. IT Automation. Node.js. React.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

A crescente dependência de sistemas computacionais nas operações empresariais tornou o monitoramento de infraestrutura de TI atividade crítica para organizações de todos os portes. A indisponibilidade de serviços, mesmo por períodos curtos, pode resultar em prejuízos financeiros significativos. Segundo dados consolidados por Atlassian (2024), o custo médio de indisponibilidade é de US$ 5.600 a US$ 9.000 por minuto, podendo alcançar valores ainda maiores para empresas de comércio eletrônico durante períodos de alta demanda.

Embora grandes corporações disponham de equipes especializadas e orçamentos robustos para implementar soluções sofisticadas de monitoramento, pequenas e médias empresas frequentemente enfrentam barreiras significativas. A complexidade técnica das ferramentas disponíveis, aliada à escassez de profissionais qualificados e recursos financeiros limitados, cria cenário desafiador para adoção de práticas adequadas de monitoramento.

Ferramentas de código aberto como Zabbix, Nagios e Prometheus oferecem funcionalidades poderosas sem custos de licenciamento (TURNBULL, 2014), porém sua configuração e operação demandam conhecimento técnico especializado. Soluções proprietárias como Datadog e Dynatrace, embora mais acessíveis em termos de usabilidade, possuem custos que iniciam em US$ 15 por servidor mensalmente e podem ultrapassar US$ 58 por servidor com funcionalidades completas (DATADOG DOCUMENTATION, 2024; DYNATRACE, 2024), tornando-se proibitivas para organizações menores.

O presente trabalho propõe o VIGIA (Visualização Integrada de Gestão e Inteligência de Anomalias), sistema que visa democratizar o acesso a monitoramento de infraestrutura através de três pilares fundamentais: interface intuitiva seguindo princípios de usabilidade (NIELSEN, 1993), sistema de notificações em tempo real via Telegram, e base de conhecimento estruturada com procedimentos guiados de resolução de problemas, reduzindo dependência de conhecimento especializado.

A motivação surgiu da experiência prática como administrador de sistemas em empresa que atende cooperativas de crédito, onde foi possível observar dificuldades enfrentadas por organizações de menor porte na manutenção de infraestruturas e na resposta rápida a incidentes.

Este documento estrutura-se da seguinte forma: o Capítulo 2 detalha o problema a ser resolvido; o Capítulo 3 apresenta soluções existentes no mercado; o Capítulo 4 analisa criticamente essas soluções identificando lacunas; o Capítulo 5 define os objetivos do projeto; o Capítulo 6 descreve o sistema desenvolvido; o Capítulo 7 apresenta procedimentos de validação; e o Capítulo 8 traz as conclusões.

2 DETALHAMENTO DO PROBLEMA

O problema central que motiva este trabalho pode ser caracterizado pela lacuna existente entre a disponibilidade de ferramentas robustas de monitoramento e a capacidade efetiva de pequenas e médias organizações em utilizá-las adequadamente. Esta seção detalha as diferentes dimensões deste problema.

2.1 COMPLEXIDADE DAS FERRAMENTAS DE MONITORAMENTO

Ferramentas como Zabbix, Nagios e Prometheus oferecem funcionalidades extensas para monitoramento de infraestrutura, porém apresentam curvas de aprendizado acentuadas. A configuração adequada envolve compreensão profunda de conceitos como *templates*, *triggers*, macros, expressões regulares e protocolos de comunicação como SNMP e IPMI. Para organizações sem equipes dedicadas de TI, este conhecimento técnico frequentemente não está disponível internamente.

A interface nativa dessas ferramentas, embora funcional, foi projetada para administradores experientes. Métricas são apresentadas em formato técnico que exige interpretação especializada. Por exemplo, um alerta "TCP: Connection refused on port 3306" pode ser claro para um DBA, mas para um gestor de TI generalista, a conexão entre este alerta e o serviço MySQL pode não ser imediata.

2.2 ESCASSEZ DE PROFISSIONAIS QUALIFICADOS

O mercado de TI enfrenta escassez de profissionais especializados em operações e infraestrutura. Segundo relatório do Google for Startups divulgado em 2023, o Brasil terá um déficit de 530 mil profissionais de tecnologia até 2025, com demanda anual de 800 mil novos talentos contra formação de apenas 53 mil profissionais por ano (Google for Startups; Abstartups, 2023). Esta carência é

particularmente crítica em áreas como *DevOps*, administração de infraestrutura e segurança da informação.

2.3 DIFICULDADE NA INTERPRETAÇÃO E AÇÃO SOBRE ALERTAS

Mesmo quando sistemas de monitoramento estão configurados e operacionais, a geração de alertas não garante resolução eficaz dos problemas. Alertas técnicos como "CPU usage > 90%" ou "MySQL max_connections reached" podem ser claros para especialistas, mas para administradores generalistas ou gestores, a conexão entre o alerta e as ações necessárias não é óbvia. Norman (2013) argumenta que bom design comunica propósito e estrutura de forma que as pessoas possam descobrir o que fazer, princípio que se aplica tanto a objetos físicos quanto a sistemas de informação.

A ausência de procedimentos documentados e acessíveis resulta em tempos de resposta prolongados. Em situações críticas, especialmente fora do horário comercial, a pressão para resolver rapidamente o problema pode levar a decisões precipitadas que agravam a situação. A necessidade de consultar documentação dispersa em wikis, fóruns e manuais técnicos adiciona fricção ao processo de resolução.

2.4 FADIGA DE ALERTAS E FALSOS POSITIVOS

Configurações inadequadas de thresholds resultam em fadiga de alertas (alert fatigue), onde o volume excessivo de notificações dessensibiliza os operadores, levando-os a ignorar alertas potencialmente críticos. Como observam Beyer et al. (2016), ambientes com alta incidência de falsos positivos apresentam tempos de resposta significativamente maiores para incidentes reais.

A falta de contextualização e priorização adequada agrava este problema. Quando todos os alertas chegam com a mesma urgência aparente, torna-se difícil distinguir entre incidentes que requerem ação imediata e aqueles que podem ser tratados posteriormente. Esta indistinção pode resultar tanto em reação excessiva a problemas menores quanto em resposta tardia a situações críticas.

2.5 CUSTOS OPERACIONAIS DE INCIDENTES

A demora na detecção e resolução de incidentes gera custos diretos e indiretos substanciais. Custos diretos incluem perda de receita durante períodos de indisponibilidade, especialmente críticos para *e-commerce* e serviços digitais. Segundo dados da Gartner consolidados em pesquisa de 2024, o custo médio de *downtime* é de US$ 5.600 por minuto, podendo alcançar US$ 9.000 por minuto para organizações de maior porte (ATLASSIAN, 2024). Para empresas de comércio eletrônico, os valores são ainda mais expressivos: durante eventos como Black Friday, até mesmo uma hora de indisponibilidade pode resultar em perdas de US$ 1 milhão a US$ 2 milhões em receitas (ERWOOD GROUP, 2025). No contexto de pequenas e médias empresas, embora os valores absolutos sejam menores (aproximadamente US$ 427 por minuto), o impacto relativo pode ser igualmente devastador, comprometendo parcela significativa do orçamento operacional limitado.

Custos indiretos envolvem perda de produtividade de equipes que dependem dos sistemas afetados, desgaste de credibilidade junto a clientes e potenciais penalidades contratuais por descumprimento de SLAs (Service Level Agreements). Para organizações menores, um incidente prolongado pode comprometer relacionamentos comerciais fundamentais.

2.6 AUSÊNCIA DE CONHECIMENTO INSTITUCIONAL DOCUMENTADO

Muitas organizações dependem do conhecimento tácito de poucos profissionais-chave. Este conhecimento, acumulado através de anos de experiência, reside nas mentes destes indivíduos, mas não está documentado de forma estruturada e acessível. Quando estes profissionais não estão disponíveis – seja por férias, afastamento por doença, ou turnover – a capacidade de resposta a incidentes fica severamente comprometida.

A ausência de runbooks (procedimentos operacionais documentados) significa que cada incidente requer investigação do zero, prolongando tempos de resolução. Mesmo quando existe alguma documentação, ela frequentemente está dispersa em wikis desatualizados, emails antigos ou anotações pessoais, dificultando o acesso rápido no momento crítico.

2.7 LIMITAÇÕES DE RECURSOS FINANCEIROS

Soluções *enterprise* de monitoramento e observabilidade, como Datadog, New Relic e Dynatrace, oferecem interfaces intuitivas, correlação automática de eventos e recursos avançados de *machine learning* para detecção de anomalias. Entretanto, seus custos de licenciamento são proibitivos para pequenas e médias empresas.

O Datadog, por exemplo, cobra a partir de US$ 15 por host monitorado mensalmente no plano básico de infraestrutura, podendo ultrapassar US$ 31 por host ao adicionar Application Performance Monitoring (APM), sem contar custos adicionais com *logs* e métricas customizadas (DATADOG DOCUMENTATION, 2024). O Dynatrace possui precificação baseada em uso horário, custando US$ 0,08 por hora para monitoramento *full-stack* de um host de 8 GB, o que equivale a aproximadamente US$ 58 por host por mês em operação contínua (DYNATRACE, 2024). Para uma infraestrutura modesta de 20 servidores, o investimento anual apenas com licenças de monitoramento pode facilmente ultrapassar US$ 10.000 no Datadog ou US$ 14.000 no Dynatrace. No contexto brasileiro, onde pequenas empresas operam com orçamentos de TI limitados e câmbio desfavorável, tais custos podem representar parcela desproporcional do budget anual, tornando-se proibitivos especialmente quando há outras prioridades competindo pelos mesmos recursos.

2.8 SÍNTESE DO PROBLEMA

Em síntese, o problema central pode ser formulado da seguinte maneira: pequenas e médias empresas necessitam de capacidades robustas de monitoramento de infraestrutura, mas enfrentam barreiras técnicas, humanas e financeiras que impedem a adoção eficaz de soluções existentes, resultando em maior vulnerabilidade a incidentes e custos operacionais elevados.

3 TECNOLOGIAS E SOLUÇÕES DE MONITORAMENTO

Este capítulo apresenta tanto as soluções de monitoramento existentes no mercado quanto as tecnologias utilizadas na implementação do sistema VIGIA.

Inicialmente, são descritas as principais ferramentas *open-source* e proprietárias amplamente empregadas em ambientes corporativos, destacando características, vantagens e limitações. Em seguida, são detalhadas as tecnologias selecionadas para compor a solução desenvolvida, justificando-se sua adoção com base nos requisitos funcionais, desempenho e viabilidade para pequenas e médias empresas.

3.1 SOLUÇÕES DE MONITORAMENTO EXISTENTES NO MERCADO

3.1.1 Zabbix

Zabbix é uma plataforma *open-source* madura para monitoramento de infraestrutura, serviços de rede e aplicações. Desenvolvido desde 1998 por Alexei Vladishev, evoluiu para se tornar uma das soluções mais completas disponíveis sem custos de licenciamento (OLUPS, 2019; ZABBIX, 2024).

Entre suas características, destacam-se monitoramento distribuído via proxies, autodescoberta de dispositivos, *templates* pré-configurados, *triggers* avançados, API JSON-RPC e suporte multiplataforma (ZABBIX DOCUMENTATION, 2024). Suas limitações incluem interface considerada datada e curva de aprendizado elevada (TURNBULL, 2014).

3.1.2 Nagios

Nagios é uma das ferramentas de monitoramento mais tradicionais, amplamente utilizada desde 1999 (JOSEPHSEN, 2007). Possui arquitetura extensível via plugins e oferece monitoramento de serviços, recursos de hosts e notificações configuráveis (NAGIOS, 2024).

Embora seja maduro e estável, apresenta interface rudimentar e configuração via arquivos de texto, sendo menos escalável que soluções modernas (TURNBULL, 2014).

3.1.3 Prometheus

Prometheus, criado em 2012 e hoje parte da CNCF, é um sistema de métricas baseado em séries temporais e amplamente utilizado em ambientes Kubernetes

(BRAZIL; VOLZ, 2018). Destaques incluem PromQL, modelo pull, Alertmanager e integração nativa com Grafana.

Limitações incluem foco exclusivo em métricas e ausência de *logs*/traces (CNCF, 2024).

3.1.4 Grafana

Grafana é uma plataforma de visualização *open-source* amplamente usada para *dashboards* customizáveis (GRAFANA LABS, 2024). Embora popular, não é um sistema completo de monitoramento e depende de outras ferramentas para coleta de dados.

3.1.5 Datadog

Datadog é uma solução SaaS com foco em observabilidade unificada, incluindo infraestrutura, APM, *logs* e segurança (DATADOG, 2024). Oferece *dashboards* prontos e *machine learning* para detecção de anomalias (DATADOG DOCUMENTATION, 2024). Seu principal desafio é o custo elevado em ambientes com muitos hosts.

3.1.6 New Relic

New Relic, fundada em 2008, é amplamente reconhecida por APM e monitoramento *full-stack* (NEW RELIC, 2024). Embora poderosa, possui modelo de precificação baseado em volume de dados, que pode se tornar proibitivo (MORRIS, 2020).

3.1.7 Dynatrace

Dynatrace oferece monitoramento *full-stack* com forte uso de IA (DYNATRACE, 2024). O OneAgent realiza instrumentação automática e o mecanismo Davis AI provê análise de causa-raiz. Entretanto, seu custo e complexidade a tornam orientada a grandes empresas (MORRIS, 2020).

3.2 TECNOLOGIAS UTILIZADAS NO SISTEMA VIGIA

Esta seção apresenta as tecnologias selecionadas para a construção do VIGIA. Diferentemente das ferramentas avaliadas anteriormente, aqui são descritos somente os componentes efetivamente utilizados no desenvolvimento do sistema.

3.2.1 Plataforma de Monitoramento

Zabbix 6.4 LTS

A escolha do Zabbix 6.4 LTS baseou-se em sua estabilidade, maturidade e compatibilidade com ambientes corporativos. A versão LTS oferece suporte por 5 anos, proporcionando confiabilidade e previsibilidade em produção (OLUPS, 2019). A API JSON-RPC permite integrações automatizadas com o VIGIA, recuperando alertas, métricas e detalhes de hosts. Além disso, não possui custos de licenciamento.

MySQL 8.0

MySQL foi adotado como banco de dados tanto para o VIGIA quanto para o Zabbix. Uma única instância reduz consumo de recursos e simplifica administração. A versão 8.0 oferece suporte completo a campos JSON, ideal para armazenar passos estruturados das soluções, e é amplamente suportada pelo próprio Zabbix.

3.2.2 *Backend*

Node.js 20

Node.js foi escolhido por seu modelo non-blocking e escalabilidade em aplicações que realizam múltiplas requisições simultâneas, ambiente natural para integrar APIs externas (WILSON, 2018). A engine V8 oferece excelente performance e o ecossistema NPM acelera o desenvolvimento.

Express.js 4.18

Express é minimalista, flexível e amplamente adotado na comunidade Node.js, facilitando criação de rotas, middlewares e APIs REST.

Bibliotecas Node.js Utilizadas

- axios — comunicação com API do Zabbix
- mysql2 — driver otimizado com prepared statements
- node-telegram-bot-api — envio de notificações
- dotenv — gerenciamento de variáveis de ambiente
- cors — configuração de CORS no *backend*

3.2.3 *Backend*

React 18 foi escolhido como biblioteca principal para construção da interface devido ao modelo baseado em componentes, reusabilidade e Virtual DOM, permitindo interfaces rápidas e de fácil manutenção (BANKS; PORCELLO, 2020). O ecossistema React oferece maturidade comprovada e ampla comunidade de desenvolvedores, facilitando resolução de problemas e acesso a documentação atualizada. Para complementar o desenvolvimento *backend*, foram utilizadas as seguintes bibliotecas:

- React Router 6: Gerenciamento de rotas da aplicação com suporte a navegação declarativa e carregamento dinâmico de componentes, essencial para aplicações single-page com múltiplas visualizações.

- Recharts 2.8: Biblioteca de gráficos integrada ao React, oferecendo componentes responsivos e customizáveis para visualização de métricas em *dashboards*.

- Lucide React: Conjunto de ícones SVG leves e escaláveis com estilo visual uniforme, contribuindo para consistência da interface.

- TailwindCSS 3.3: *Framework* utility-first que acelera o desenvolvimento através de classes CSS pré-definidas, permitindo criação de interfaces responsivas e consistentes sem necessidade de escrever CSS customizado extensivamente

3.2.4 Ambiente de Virtualização e Desenvolvimento

Oracle VM VirtualBox 7.0

Escolhido por ser gratuito, multiplataforma e amplamente utilizado em ambientes educacionais e PMEs. Permite snapshots e múltiplos modos de rede.

Ubuntu Server 24.04 LTS

Sistema operacional robusto, estável e com suporte oficial estendido. Possui ampla documentação e comunidade ativa, além de requisitos modestos de hardware.

3.3 SÍNTESE INTEGRADA

A análise realizada revela duas tendências claras:

1. Ferramentas *open-source* são poderosas, mas exigem domínio técnico;
2. Ferramentas proprietárias SaaS oferecem maior usabilidade, porém com custos elevados.

As tecnologias selecionadas para o VIGIA, o Zabbix, MySQL, Node.js, React e virtualização com VirtualBox + Ubuntu, atendem ao objetivo central do projeto: viabilizar uma solução acessível, moderna e funcional para pequenas e médias empresas, com baixo custo e alta capacidade de integração.

4 ANÁLISE CRÍTICA DAS SOLUÇÕES EXISTENTES

A análise das soluções apresentadas no capítulo anterior, combinada com observação de cenários reais de operação em pequenas e médias empresas brasileiras, revela padrões consistentes e lacunas significativas que justificam a proposta deste trabalho.

4.1 LACUNAS IDENTIFICADAS

4.1.1 Dicotomia entre Custo e Usabilidade

Observa-se clara dicotomia no mercado de soluções de monitoramento. Ferramentas *open-source* como Zabbix e Nagios são economicamente acessíveis por serem gratuitas, mas apresentam barreiras significativas de usabilidade e complexidade operacional. Suas interfaces refletem origens técnicas e pressupõem conhecimento especializado de administração de sistemas. A configuração inicial requer compreensão de conceitos como *triggers*, *templates*, macros e expressões regulares, representando curva de aprendizado acentuada.

Em contraste, soluções proprietárias como Datadog e New Relic oferecem experiência de usuário refinada com interfaces modernas, *setup* simplificado e recursos avançados. Entretanto, seus modelos de precificação baseados em hosts monitorados ou volume de dados ingeridos rapidamente se tornam proibitivos. Para pequenas empresas com orçamentos de TI limitados, custos mensais iniciando em centenas de dólares podem representar investimento desproporcional, especialmente considerando contexto econômico brasileiro.

Esta polarização força pequenas organizações a escolherem entre investir tempo e recursos humanos especializados para operar soluções complexas mas gratuitas, ou comprometer orçamentos limitados com soluções pagas que podem consumir parcela significativa do budget de TI. Não há oferta significativa no mercado que combine acessibilidade econômica com usabilidade adequada para operadores sem expertise profundo em administração de sistemas.

4.1.2 Ausência de Orientação para Resolução

A lacuna mais crítica identificada é a ausência de componentes integrados de orientação para resolução de incidentes em todas as soluções analisadas, tanto *open-source* quanto proprietárias. O fluxo típico de trabalho atual termina abruptamente na geração do alerta, deixando ao operador a responsabilidade de determinar ações corretivas.

O processo comum funciona da seguinte forma: o sistema detecta anomalia através de *triggers* ou regras configuradas, alerta é gerado identificando host e problema, operador recebe notificação via email ou outro canal, e neste ponto surge gap significativo onde operador deve por conta própria interpretar o alerta, determinar causa provável, buscar procedimento de resolução em fontes externas (wiki, documentação, Google, Stack Overflow), executar ações corretivas e verificar se problema foi efetivamente resolvido.

Este gap representa salto cognitivo significativo que assume conhecimento especializado. Para alertas complexos ou situações pouco frequentes, mesmo operadores experientes podem gastar tempo considerável pesquisando e experimentando soluções. Para profissionais menos experientes, gestores de TI

generalistas ou equipes pequenas onde mesma pessoa acumula múltiplas funções, este gap pode resultar em tempos de resolução prolongados, ações incorretas que potencialmente agravam o problema, necessidade de escalar para suporte externo com custos adicionais e estresse significativo durante incidentes críticos.

O conhecimento sobre resolução de problemas geralmente existe de forma tácita na experiência de profissionais seniores, dispersa em documentos diversos e sistemas fragmentados, desatualizada quando documentação não acompanha evolução de sistemas, ou inacessível quando procedimentos não foram documentados ou estão em locais que operador não conhece ou não tem permissão de acesso.

4.1.3 Fragmentação de Ferramentas

Em cenários reais de operação, raramente uma única ferramenta satisfaz todas as necessidades de monitoramento, gestão de incidentes e documentação. É comum observar ambientes onde coexistem Zabbix para monitoramento de infraestrutura, Grafana para criação de *dashboards* visuais mais modernos, sistema de ticketing como GLPI ou OTRS para gestão formal de incidentes, wiki corporativa ou documentos compartilhados para procedimentos, plataforma de comunicação como Slack ou Telegram para coordenação de equipe, e planilhas para geração de relatórios gerenciais. Esta fragmentação cria múltiplos problemas operacionais.

Operadores precisam alternar entre múltiplas interfaces durante resolução de incidente, cada uma com paradigmas de navegação e interação diferentes. Correlacionar informações dispersas em diferentes sistemas consome tempo e aumenta carga cognitiva em momentos já estressantes. A curva de aprendizado é multiplicada pois cada ferramenta requer familiarização com sua interface específica. O overhead operacional de manter múltiplos sistemas atualizados e sincronizados consome recursos escassos. Inconsistências de dados entre sistemas podem ocorrer quando informações não são atualizadas uniformemente.

4.1.4 Barreira de Conhecimento Especializado

Ferramentas *open-source* estabelecidas como Zabbix foram desenvolvidas por e para especialistas em operações de TI. Seus conceitos, terminologia e interfaces

pressupõem familiaridade profunda com administração de sistemas Linux, protocolos de rede e arquitetura de infraestrutura. Esta premissa cria barreira significativa para organizações menores onde profissionais de TI frequentemente são generalistas responsáveis por amplo espectro de atividades.

Um alerta típico como "Zabbix agent on host is unreachable for 5 minutes" é claro para administrador experiente familiarizado com arquitetura Zabbix, mas pode gerar dúvidas para operador menos experiente: o que exatamente é Zabbix agent, por que está unreachable, quais são causas comuns desta situação, qual a gravidade real do problema, quais passos seguir para diagnosticar e resolver.

Esta barreira de conhecimento não é adequadamente endereçada pelas ferramentas atuais que tratam o operador como persona única assumindo alto nível de expertise. Como observa Nielsen (1993), sistemas devem ser projetados para usuários com diferentes níveis de conhecimento, oferecendo tanto atalhos para especialistas quanto orientação para iniciantes. Não há gradação de complexidade ou explicações contextuais que facilitem compreensão para profissionais em desenvolvimento ou generalistas.

4.2 OPORTUNIDADES IDENTIFICADAS

As lacunas descritas representam não apenas problemas, mas também oportunidades para desenvolvimento de soluções que preencham gaps específicos do mercado.

4.2.1 Democratização do Acesso a Monitoramento Eficaz

Existe oportunidade para soluções que mantenham profundidade técnica e capacidades de ferramentas *open-source* estabelecidas, mas ofereçam experiência de usuário mais acessível. Como defende Norman (2013), democratização não implica necessariamente em simplificação excessiva que limite capacidades, mas em abstração apropriada de complexidade desnecessária para operações comuns, apresentação de informação de forma contextualizada e compreensível, guiagem através de processos que atualmente requerem expertise especializado e redução de curvas de aprendizado sem sacrificar funcionalidades essenciais.

4.2.2 Conhecimento Integrado ao Sistema

Há valor em sistemas que não apenas detectem problemas mas também forneçam conhecimento estruturado necessário para resolvê-los. Integrar base de conhecimento diretamente ao sistema de monitoramento pode reduzir drasticamente tempos de resposta eliminando necessidade de busca em múltiplas fontes, diminuir dependência de especialistas escassos ao capacitar profissionais menos experientes, permitir que operadores júnior contribuam efetivamente para resolução de incidentes, capturar e institucionalizar conhecimento organizacional que de outra forma permaneceria tácito, e facilitar onboarding de novos membros reduzindo tempo até que se tornem produtivos.

4.2.3 Integração Funcional

Oportunidade existe para criar soluções mais integradas que combinem detecção de problemas, visualização de estado da infraestrutura, acesso a conhecimento de resolução, notificação de stakeholders e documentação de ações tomadas em interface unificada. Esta integração reduz overhead cognitivo de alternar entre múltiplas ferramentas e facilita correlação de informações relevantes durante resolução de incidentes.

4.3 JUSTIFICATIVA DA PROPOSTA

A análise apresentada demonstra que, apesar da abundância de ferramentas de monitoramento disponíveis, existe lacuna fundamental não adequadamente atendida por soluções existentes: a ponte entre detecção de problemas e resolução efetiva, especialmente para organizações com recursos financeiros limitados e equipes técnicas enxutas ou com níveis variados de experiência.

Esta lacuna manifesta-se em múltiplas dimensões identificadas na literatura e práticas de mercado. Primeiramente, ferramentas de código aberto estabelecidas como Zabbix e Nagios, embora tecnicamente robustas, apresentam interfaces que pressupõem expertise especializado (TURNBULL, 2014). Como observa Nielsen (1993), sistemas devem ser projetados para usuários com diferentes níveis de conhecimento, princípio frequentemente violado por ferramentas de infraestrutura tradicionais.

Em contrapartida, soluções proprietárias como Datadog e Dynatrace, que oferecem maior usabilidade e recursos avançados, possuem modelos de precificação que iniciam em US$ 15 por hospedeiro mensalmente e podem ultrapassar US$ 58 por hospedeiro ao adicionar funcionalidades completas (DATADOG DOCUMENTATION, 2024; DYNATRACE, 2024). Para infraestruturas modestas de 20 a 50 servidores, os custos anuais podem facilmente exceder US$ 10.000 a US$ 30.000, valores consideráveis para pequenas e médias empresas com orçamentos de TI limitados.

Adicionalmente, nenhuma das soluções analisadas integra sistematicamente conhecimento estruturado de resolução ao sistema de monitoramento. O processo típico termina na geração do alerta, deixando ao operador a responsabilidade de determinar ações corretivas através de consulta a fontes externas (wikis, documentação, buscas online). Esta lacuna entre detecção e resolução é particularmente problemática em contexto de custos elevados de indisponibilidade, que segundo pesquisas recentes podem alcançar US$ 5.600 a US$ 9.000 por minuto dependendo do porte e setor da organização (ATLASSIAN, 2024).

Como observam Limoncelli, Chalup e Hogan (2014), a gestão eficaz de sistemas requer não apenas detecção de problemas, mas também processos estruturados para resposta. A ausência de orientação integrada força dependência de poucos especialistas cujo conhecimento permanece tácito, criando pontos únicos de falha operacional quando estes profissionais não estão disponíveis.

A oportunidade identificada reside em desenvolver solução que combine acessibilidade econômica de ferramentas de código aberto com usabilidade aprimorada e, principalmente, conhecimento estruturado de resolução integrado ao sistema de monitoramento. Esta combinação pode tornar monitoramento eficaz mais acessível a segmento significativo de mercado atualmente mal servido por alternativas existentes, reduzindo barreira entre detecção de anomalias e capacidade de resposta efetiva.

5 OBJETIVOS DA PROPOSTA

Com base nas lacunas identificadas na análise das soluções existentes, este capítulo apresenta os objetivos do projeto VIGIA, estruturados em objetivo geral que

define a visão ampla da solução e objetivos específicos que detalham componentes e funcionalidades a serem implementados.

5.1 OBJETIVO GERAL

Desenvolver sistema integrado de monitoramento de infraestrutura que combine capacidades de coleta e detecção de anomalias de ferramentas *open-source* estabelecidas com camada de orientação estruturada para resolução de incidentes, permitindo que profissionais com diferentes níveis de experiência possam detectar e resolver problemas de forma mais autônoma através de interface web moderna e base de conhecimento integrada.

5.2 OBJETIVOS ESPECÍFICOS

5.2.1 Integração com Zabbix

Implementar integração com Zabbix como plataforma base de monitoramento, aproveitando sua maturidade, capacidades de coleta de métricas, sistema de *triggers* e escalabilidade comprovada. A comunicação será realizada através da API JSON-RPC do Zabbix, permitindo recuperação de alertas, métricas e informações de hosts monitorados.

5.2.2 Interface Web Moderna

Desenvolver interface web responsiva utilizando tecnologias modernas que facilite visualização e interpretação de informações de infraestrutura. Seguindo princípios de usabilidade defendidos por Nielsen (1993) e Krug (2014), a interface proporcionará *dashboard* com visão geral do estado dos hosts monitorados, visualização de métricas principais em formato gráfico, lista de alertas ativos com indicação visual de severidade e acesso direto a procedimentos de resolução correspondentes.

5.2.3 Sistema de Notificações

Implementar sistema de notificações via Telegram que permita alertas sobre incidentes críticos, formatação de mensagens com informações relevantes do problema e *links* diretos para soluções correspondentes na interface web.

5.2.4 Base de Conhecimento Estruturada

Criar e popular base de dados com procedimentos detalhados de resolução para tipos comuns de incidentes em infraestruturas de pequenas e médias empresas. Cada procedimento deve incluir descrição clara do problema e seu impacto, passos estruturados de diagnóstico e correção, comandos específicos quando aplicável, orientações sobre análise de causa raiz e boas práticas de prevenção.

5.2.5 Associação entre Alertas e Soluções

Desenvolver mecanismo de associação entre *triggers* configurados no Zabbix e soluções correspondentes na base de conhecimento, permitindo que ao visualizar alerta o operador tenha acesso direto ao procedimento de resolução apropriado sem necessidade de busca manual.

5.2.6 Funcionalidade de Busca

Implementar sistema de busca na base de conhecimento que permita localização de procedimentos por palavras-chave, filtros por categoria e navegação por tipo de problema, facilitando acesso a soluções mesmo quando não há alerta ativo associado.

5.2.7 Visualização de Métricas

Implementar visualizações de métricas críticas de infraestrutura incluindo utilização de CPU, consumo de memória RAM e disponibilidade de serviços essenciais.

5.3 DELIMITAÇÃO DE ESCOPO

Para clareza sobre limites do trabalho, é importante definir aspectos que não estão contemplados no escopo deste projeto.

O VIGIA não visa substituir completamente o Zabbix mas sim complementá-lo com camada adicional de usabilidade e conhecimento estruturado. Não será desenvolvido monitoramento em nível de aplicação (APM) com rastreamento de

transações ou performance de código. Análise centralizada de *logs* não está contemplada, assim como distributed tracing para arquiteturas de microsserviços.

Não serão implementados recursos de *machine learning* para predição de falhas ou detecção automática de anomalias baseada em padrões históricos. Aplicativo móvel nativo não faz parte do escopo, embora interface web seja responsiva. Integração com sistemas de ticketing externos como OTRS ou Jira não será desenvolvida. O sistema é projetado para uso por organização única, não implementando recursos de multi-tenancy necessários para oferta como serviço.

O foco deliberado do projeto é criar solução que resolva problema central identificado - tornar monitoramento e gestão de incidentes mais acessíveis - sem tentar competir em todos os aspectos com soluções *enterprise* de escopo mais amplo.

5.4 RESULTADOS ESPERADOS

Espera-se que ao final do desenvolvimento o sistema seja capaz de monitorar múltiplos hosts simultaneamente coletando métricas periodicamente, apresentar alertas de forma clara com indicação visual de severidade, fornecer acesso integrado a procedimentos de resolução estruturados, enviar notificações em tempo adequado para problemas críticos e permitir navegação e busca de soluções de forma intuitiva.

A validação será realizada através de testes em ambiente controlado verificando funcionamento correto das integrações, qualidade e cobertura da base de conhecimento, usabilidade da interface e confiabilidade do sistema de notificações.

O projeto será considerado bem-sucedido se demonstrar capacidade de detectar incidentes através de integração com Zabbix, fornecer orientação estruturada para resolução através de base de conhecimento abrangente, apresentar informações de forma mais acessível que ferramentas tradicionais de monitoramento e operar de forma estável em ambiente de testes durante período de validação.

6 DESCRIÇÃO DO PROJETO

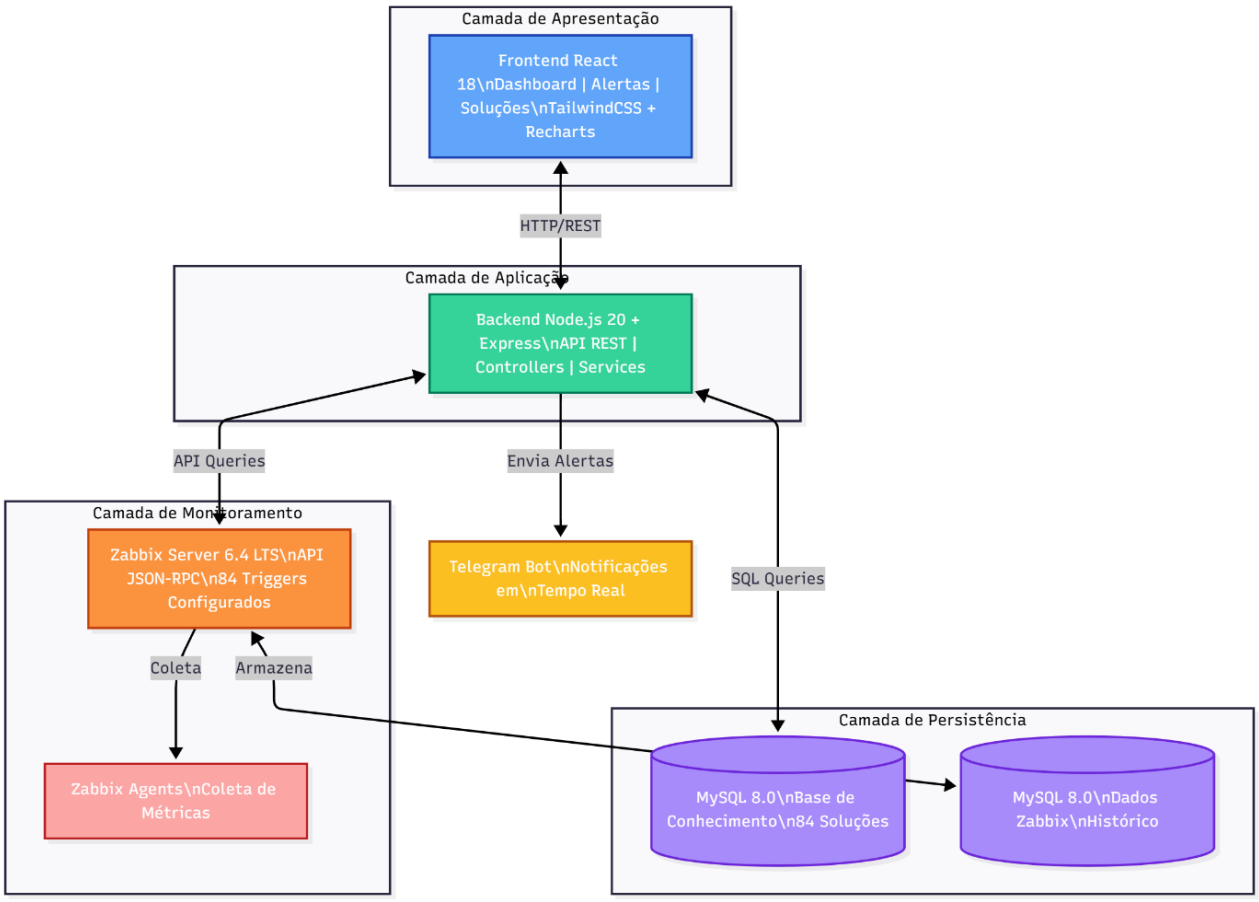
O projeto VIGIA (Visualização Integrada de Gestão e Inteligência de Anomalias) foi desenvolvido como solução integrada que combina monitoramento de infraestrutura com orientação estruturada para resolução de incidentes. Este capítulo descreve a arquitetura, componentes principais e funcionalidades implementadas no sistema.

6.1 VISÃO GERAL DA ARQUITETURA

O VIGIA foi arquitetado em camadas que separam responsabilidades e permitem manutenção independente dos componentes. Seguindo princípios de arquitetura de software propostos por Fowler (2002), a estrutura em camadas promove separação de responsabilidades e facilita evolução do sistema. A arquitetura é composta por quatro camadas principais que trabalham de forma integrada.

A camada de monitoramento utiliza Zabbix 6.4 LTS como plataforma base, responsável pela coleta contínua de métricas dos hosts e detecção de anomalias através de *triggers* configurados. A camada de aplicação implementada em Node.js 20 com *framework* Express processa dados recuperados do Zabbix, gerencia lógica de negócio e orquestra comunicação entre componentes. A camada de apresentação desenvolvida em React 18 fornece interface web responsiva para visualização de *dashboards*, alertas e soluções. A camada de persistência utiliza MySQL 8.0 para armazenamento da base de conhecimento com procedimentos de resolução.

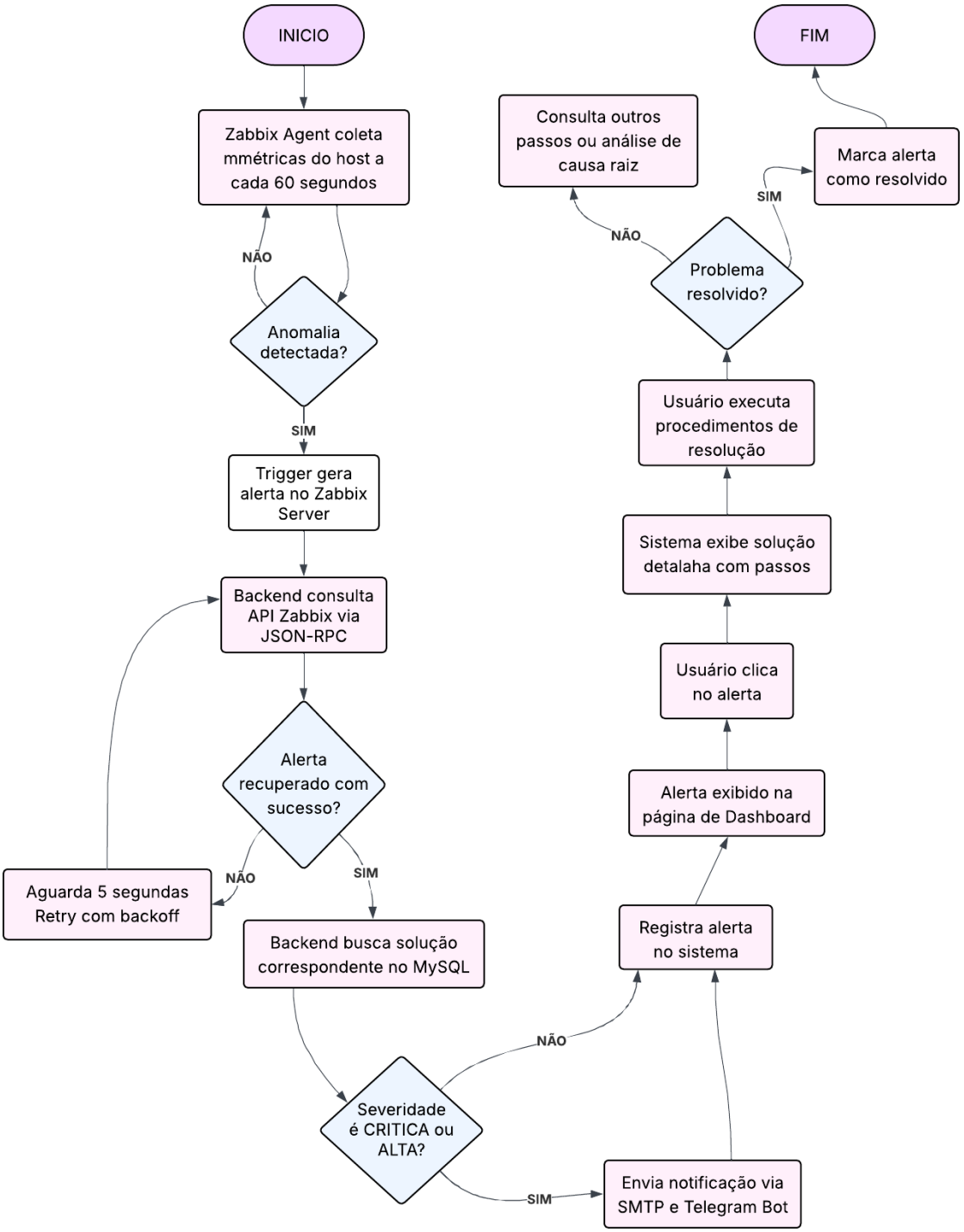
Figura 1 - Arquitetura em camadas do sistema VIGIA



Fonte: Elaborado pelo autor (2025)

O fluxo de dados opera da seguinte forma: Zabbix coleta métricas dos hosts através de agentes instalados, quando anomalias são detectadas *triggers* geram alertas, *backend* recupera alertas através de consultas à API Zabbix, processa informações e busca soluções correspondentes no banco de dados, para incidentes críticos envia notificações via Telegram, e *backend* consulta *backend* via API REST para apresentar *dashboards* e procedimentos de resolução.

Figura 2 - Fluxograma do processamento de alertas



Fonte: Elaborado pelo autor (2025)

6.2 CAMADA DE MONITORAMENTO

O Zabbix Server 6.4 LTS foi escolhido como plataforma de monitoramento base devido à sua maturidade, confiabilidade e ampla adoção. A versão LTS (Long Term Support) garante estabilidade e suporte prolongado, aspectos importantes para operação contínua.

A infraestrutura foi implementada em máquina virtual utilizando Oracle VM VirtualBox, simulando ambiente típico de pequena empresa. O Zabbix Server foi instalado nativamente no Ubuntu Server 24.04 LTS através do repositório oficial, configurado com MySQL 8.0 como banco de dados. Nos hosts monitorados, Zabbix Agent 2 foi instalado para coleta de métricas em intervalos de 60 segundos, equilibrando granularidade de dados com carga computacional.

Templates customizados foram criados para cobrir principais componentes de infraestrutura típicos de pequenas e médias empresas. O template Linux OS Extended monitora recursos fundamentais do sistema operacional incluindo utilização de CPU, consumo de memória RAM e swap, espaço em disco e tráfego de rede. *Templates* para serviços web monitoram Apache e Nginx, acompanhando conexões ativas, tempo de resposta, códigos HTTP e *status* de certificados SSL.

Monitoramento de bancos de dados contempla MySQL e PostgreSQL, com métricas como número de conexões ativas, *queries* lentas, *locks* de tabelas e *status* de replicação. Serviços de email (Postfix e Dovecot) têm verificação de *status*, tamanho de filas e checagem de blacklists. Um template de segurança monitora componentes como firewall, execução de backups e disponibilidade de atualizações.

O sistema de alertas foi configurado com 84 *triggers* diferentes, organizados em quatro níveis de severidade. Alertas críticos são reservados para situações que impactam diretamente disponibilidade de serviços, como hosts inacessíveis, serviços vitais parados, espaço em disco acima de 95% ou certificados SSL expirados. Severidade alta indica problemas com potencial de evoluir para situações críticas. Severidade média representa situações que merecem atenção mas sem urgência imediata. Severidade baixa é utilizada para notificações informativas.

6.3 CAMADA DE APLICAÇÃO

O *backend* foi desenvolvido em Node.js 20 com *framework* Express, atuando como camada de orquestração entre Zabbix, banco de dados e interface web. Node.js foi escolhido por sua eficiência em operações de I/O assíncronas, importante para um sistema que realiza múltiplas consultas a APIs externas.

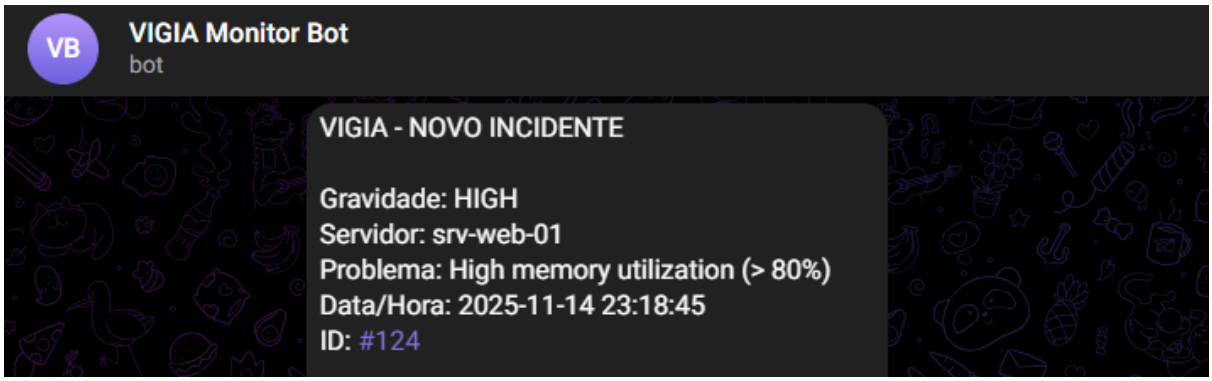
Ele foi estruturado seguindo o princípio de separação de responsabilidades, com organização modular que facilita manutenção e extensibilidade. A camada de configuração centraliza parâmetros de conexão e credenciais. *Controllers* implementam a lógica de negócio para manipulação de hosts, alertas e soluções. *Routes* definem os endpoints da API REST consumidos pelo *backend*. *Services* encapsulam integrações com sistemas externos, como Zabbix, Telegram e SMTP.

A comunicação com o Zabbix ocorre através de sua API JSON-RPC, que requer autenticação via token de sessão. O *backend* implementa um serviço dedicado que gerencia essa comunicação, incluindo autenticação, renovação automática de tokens e tratamento de erros. Um mecanismo de *polling* verifica novos alertas periodicamente, mantendo um estado local dos alertas já processados para evitar notificações duplicadas.

O módulo de notificações utiliza a API oficial do Telegram para envio de mensagens estruturadas, formatadas com identificação do host afetado, descrição do problema e nível de severidade. O sistema suporta envio para múltiplos destinatários simultaneamente.

Além disso, o *backend* também integra um serviço de envio de e-mails via SMTP, seguindo a mesma arquitetura de abstração utilizada no módulo do Telegram. O serviço SMTP encapsula o envio de mensagens, formata o conteúdo das notificações e permite o disparo para múltiplos endereços de e-mail, ampliando a redundância e a flexibilidade dos canais de alerta disponíveis no sistema.

Figura 3 - Notificação via Telegram com informações do incidente.



Fonte: Mensagem enviada pelo bot (VIGIA) do Telegram

Figura 4 - Notificação via SMTP com link para solução

Fonte: Notificação enviada via SMTP

6.4 BASE DE CONHECIMENTO

O componente central do VIGIA é sua base de conhecimento estruturada, composta por 84 procedimentos completos de resolução de incidentes. Essa base não apenas documenta rotinas operacionais, mas representa o núcleo da proposta de democratização do conhecimento técnico, permitindo que equipes menos experientes tenham acesso a orientações de nível profissional.

Cada solução foi projetada a partir de estrutura lógica que guia o usuário desde o diagnóstico inicial até a resolução definitiva e a prevenção de recorrências. As seções incluem: explicação do problema e seu impacto operacional; comandos de verificação; procedimentos de correção; análise de causa raiz; indicação de *logs* relevantes; recomendações de ajustes de configuração; e boas práticas de monitoramento contínuo. Os procedimentos são apresentados em linguagem clara, evitando jargões desnecessários. Cada comando é acompanhado de comentários explicativos contextualizando sua finalidade, garantindo que usuários não apenas executem ações, mas compreendam o motivo e o efeito esperado de cada etapa. Esse cuidado reduz riscos, aprimora a tomada de decisão e favorece o desenvolvimento de autonomia técnica.

Além da fundamentação teórica em documentações oficiais, os procedimentos foram desenvolvidos com forte base na experiência prática adquirida ao longo da minha atuação profissional como Administrador de Infraestrutura e Segurança, vivenciando diariamente incidentes reais em ambientes produtivos. Essa vivência permitiu identificar padrões recorrentes, erros comuns, falhas operacionais típicas e os métodos mais eficientes para resolução de problemas. Este conhecimento foi sistematizado e adaptado para linguagem acessível dentro do VIGIA. Dessa forma, o sistema não apenas replica passos genéricos, mas incorpora práticas comprovadas utilizadas no cotidiano de profissionais de infraestrutura.

As 84 soluções foram organizadas em seis categorias principais:

- Conectividade e Disponibilidade (15 soluções): hospedeiros inacessíveis, falhas de serviço, perda de conectividade.

- Desempenho e Recursos (20 soluções): alto uso de CPU, esgotamento de memória, falta de espaço em disco.

- Banco de Dados (18 soluções): MySQL e PostgreSQL, incluindo consultas lentas, bloqueios, corrupção de tabelas e replicação.

- Serviços Web (15 soluções): Apache, Nginx, certificados SSL e erros HTTP.

- Correio Eletrônico e Comunicações (8 soluções): Postfix, Dovecot e diagnósticos de falhas de entrega.

- Segurança e Sistema (8 soluções): atualizações, firewall, permissões, políticas de cópia de segurança e endurecimento do sistema.

A elaboração de cada uma dessas soluções seguiu processo metodológico estruturado que envolveu:

(1) identificação dos tipos de incidentes mais frequentes baseada em experiência profissional e análise de alertas típicos do Zabbix; (2) pesquisa em documentações oficiais dos sistemas monitorados para fundamentação técnica; (3) estruturação dos procedimentos em formato padronizado; (4) testes em ambiente de laboratório virtualizado com simulação deliberada de cenários de incidentes para validação dos comandos; (5) refinamento iterativo da linguagem para acessibilidade a diferentes níveis de experiência. Comandos destrutivos ou de alto risco foram evitados, priorizando abordagens conservadoras adequadas ao público-alvo.

O resultado é base de conhecimento com média de 52 passos por solução, totalizando mais de 4.300 passos documentados, oferecendo repositório robusto, validado e alinhado às necessidades operacionais de pequenas e médias empresas.

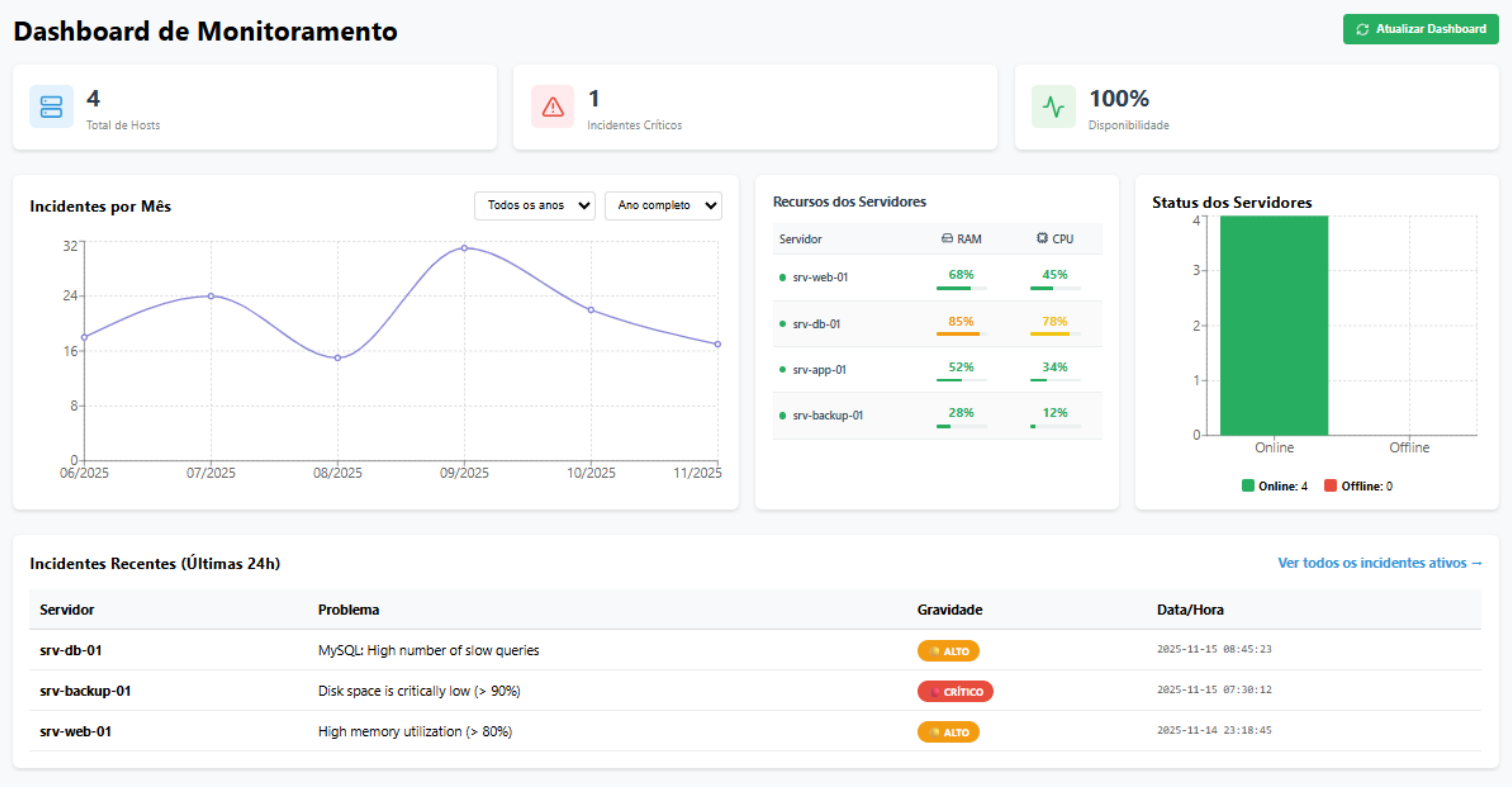
6.5 CAMADA DE APRESENTAÇÃO

O *backend* foi desenvolvido em React 18, biblioteca JavaScript amplamente adotada para construção de interfaces modernas. Como observam Banks e Porcello (2020), a arquitetura baseada em componentes do React promove reusabilidade e facilita manutenção de aplicações complexas.

O *dashboard* principal apresenta visão consolidada da saúde da infraestrutura. No topo, *cards* resumem métricas fundamentais: total de hosts monitorados, número de alertas ativos, hosts com problemas e estatísticas de incidentes. Lista exibe todos os hosts monitorados, cada um em card individual mostrando nome, endereço IP, *status* e métricas principais (CPU, memória, disco) com barras de progresso visuais. Hosts com alertas ativos são destacados visualmente.

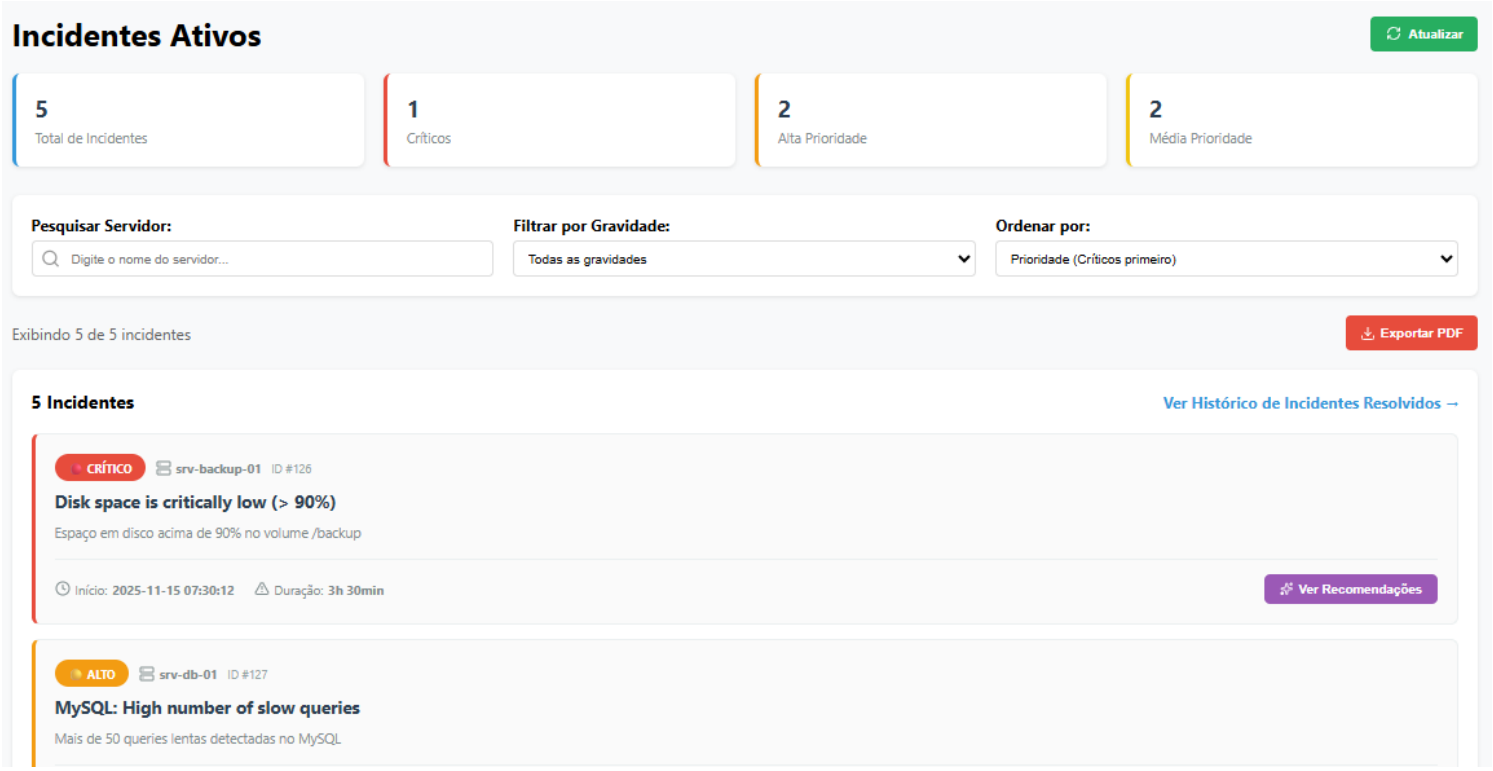
Gráficos de linha exibem evolução temporal das principais métricas, permitindo identificar tendências e padrões. A visualização gráfica é útil para detectar problemas intermitentes. Seção de alertas recentes lista últimos incidentes detectados, ordenados por timestamp. Cada alerta exibe severidade, host afetado, descrição e botão para acesso direto à solução correspondente.

Figura 5 - Dashboard principal do VIGIA



Fonte: Dashboard da página dashboard do VIGIA

Figura 6 - Interface de lista de alertas com filtros por severidade e host



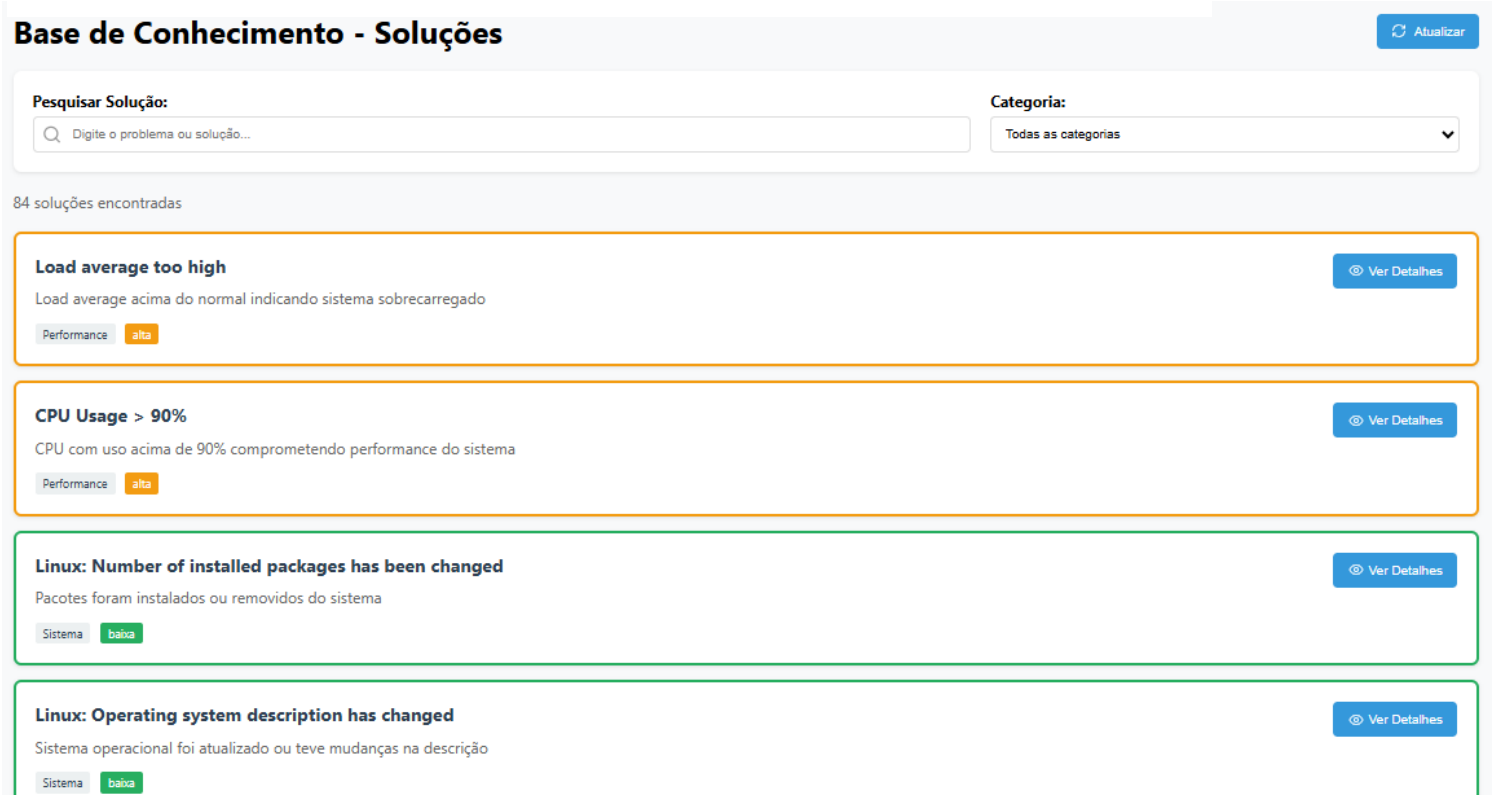
Fonte: Visualização da página de incidentes do VIGIA

A interface de alertas permite filtrar e ordenar incidentes conforme necessidade. Filtros por severidade e host facilitam foco em problemas específicos. Cada alerta é apresentado em card visual com código de cores correspondente à severidade.

O módulo de soluções implementa a proposta de democratização do conhecimento. Funcionalidade de busca permite localizar procedimentos digitando palavras-chave. O sistema busca em múltiplos campos, retornando resultados ordenados por relevância. Lista completa de soluções pode ser filtrada por categoria.

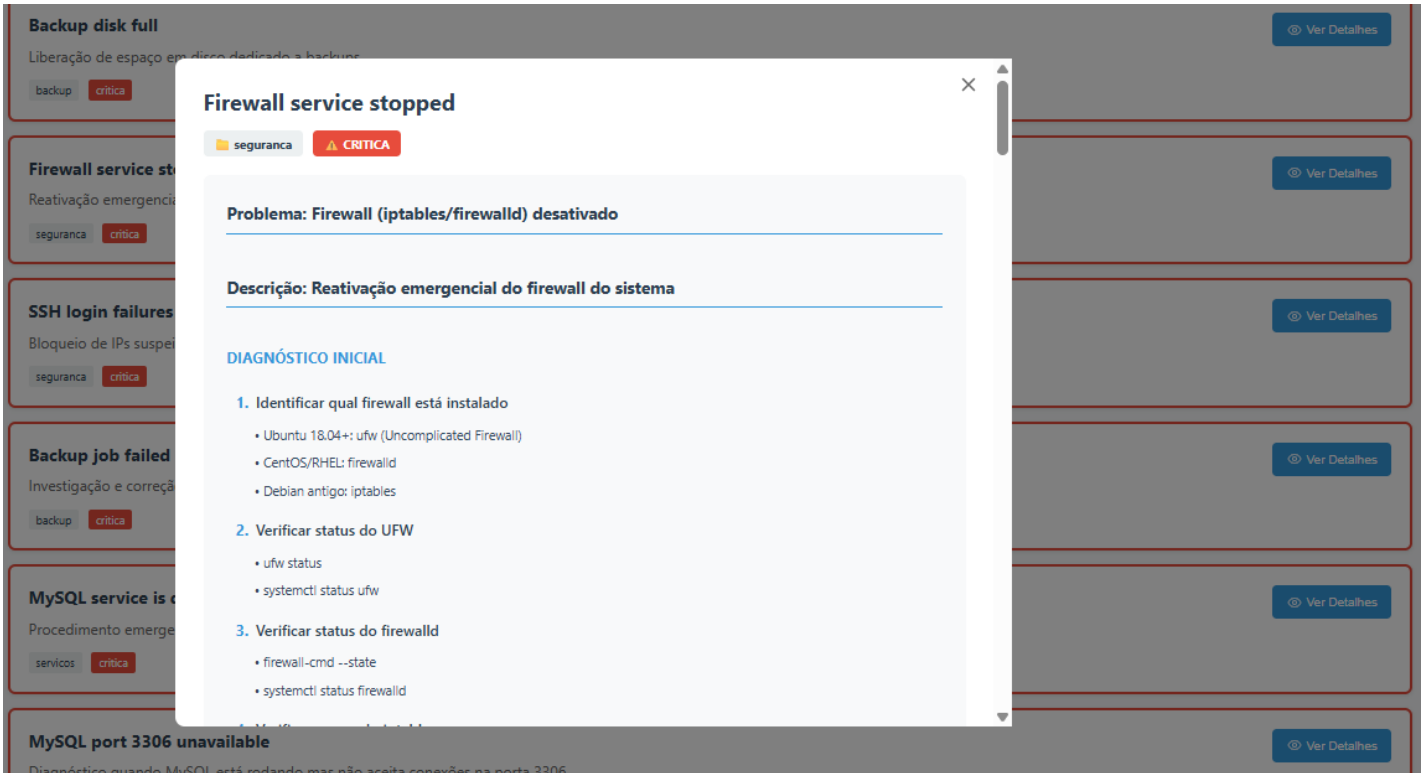
Cada solução é apresentada inicialmente em formato resumido com título, categoria e descrição breve. Ao selecionar uma solução, usuário acessa visualização detalhada completa estruturada em seções claramente delimitadas. Comandos são destacados visualmente em blocos com fundo diferenciado. Botões de cópia permitem transferir comandos para clipboard com um clique, reduzindo erros de digitação

Figura 7 - Interface de busca de soluções com resultados filtrados



Fonte: Exemplos de soluções da página solucoes

Figura 8 - Visualização detalhada de uma solução



Fonte: Exemplo de modal de soluções

A interface foi desenvolvida com abordagem responsiva utilizando TailwindCSS para estilização. O design se adapta automaticamente a diferentes tamanhos de tela, mantendo usabilidade em desktops, tablets e smartphones. Em dispositivos móveis, menu lateral colapsa, *cards* de host são redimensionados e gráficos ajustam altura para melhor visualização.

6.6 SÍNTESE DA IMPLEMENTAÇÃO

O sistema VIGIA foi desenvolvido integrando componentes estabelecidos (Zabbix para monitoramento) com desenvolvimento customizado (*backend* Node.js, *backend* React e base de conhecimento estruturada). Esta abordagem aproveita robustez de ferramentas maduras enquanto adiciona camada de usabilidade e orientação para resolução de problemas.

A arquitetura em camadas permite que cada componente seja mantido e evoluído independentemente. A base de conhecimento com 84 soluções detalhadas representa investimento significativo em captura e estruturação de conhecimento técnico, tornando-o acessível de forma integrada ao sistema de monitoramento.

A interface web moderna desenvolvida em React oferece experiência de usuário superior à tipicamente encontrada em ferramentas *open-source* de monitoramento, enquanto mantém toda a funcionalidade necessária para operação efetiva de infraestrutura de pequeno e médio porte.

7 PROCEDIMENTOS DE VALIDAÇÃO

Este capítulo apresenta os procedimentos de validação realizados para verificar o funcionamento correto do sistema VIGIA e demonstrar que os objetivos propostos foram alcançados. Como observam Pressman e Maxim (2016), a validação de software visa garantir que o produto final atenda aos requisitos e necessidades dos usuários. A validação foi conduzida através de testes funcionais em ambiente de desenvolvimento e coleta de dados durante período prolongado de monitoramento.

7.1 METODOLOGIA DE VALIDAÇÃO

A estratégia de validação foi estruturada para demonstrar seis aspectos principais do sistema: capacidade de coleta de métricas via integração com Zabbix, detecção de incidentes através de *triggers* configurados, sistema de notificações via Telegram, completude e qualidade da base de conhecimento, performance da interface web e funcionalidades de recuperação automática.

A validação foi realizada em ambiente controlado de desenvolvimento utilizando servidores virtuais configurados especificamente para teste e validação das funcionalidades implementadas. Esta abordagem permitiu demonstração completa de todas as capacidades do sistema em condições controladas, possibilitando apresentação e avaliação consistente durante conclusão do trabalho.

7.2 AMBIENTE DE DESENVOLVIMENTO E TESTES

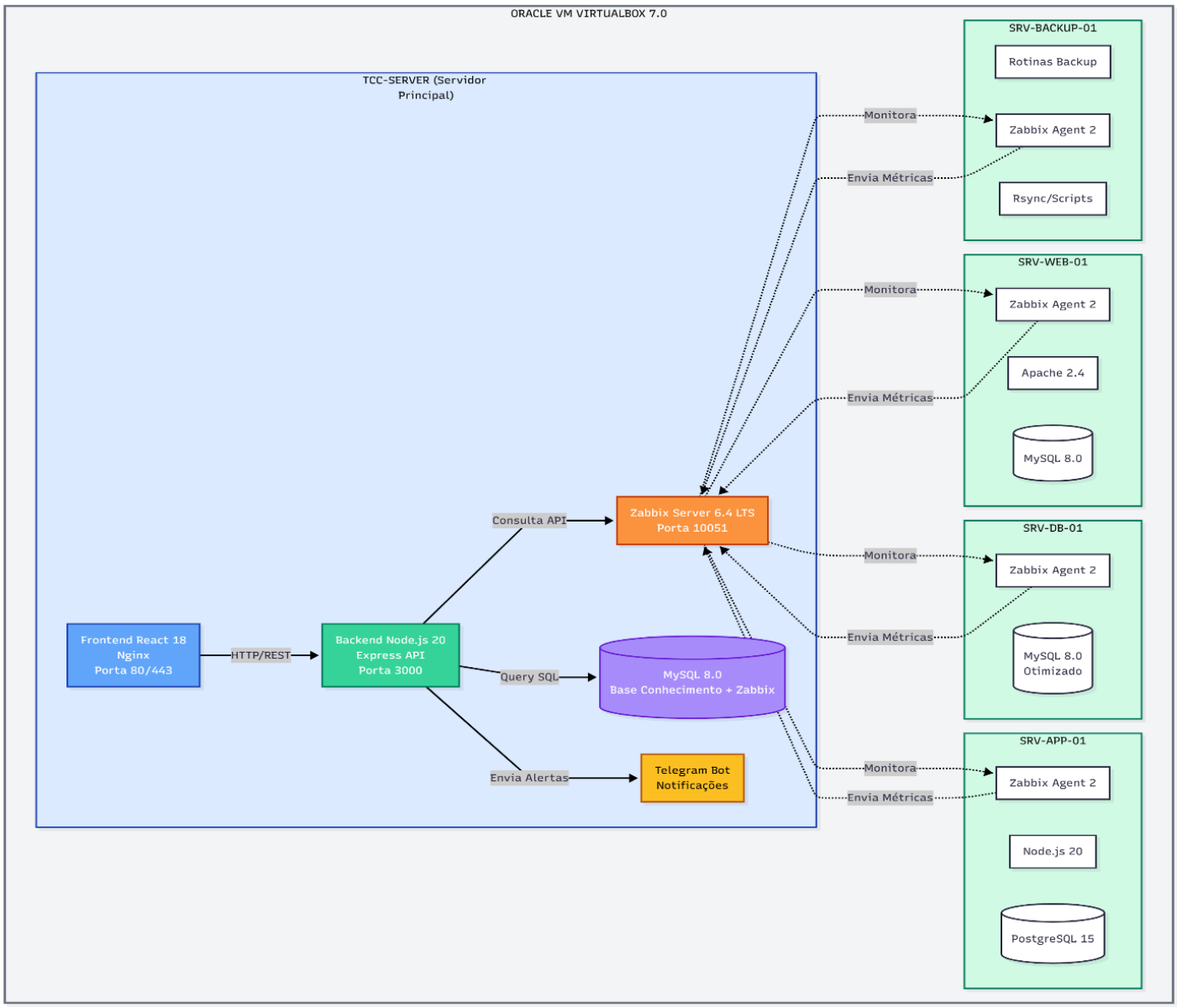
O ambiente de desenvolvimento foi configurado utilizando Oracle VM VirtualBox 7.0 com um servidor principal executando Ubuntu Server 24.04 LTS (hostname tcc-server) equipado com 4 vCPUs, 8GB RAM e 100GB de disco virtual. Este servidor hospeda todos os componentes do VIGIA incluindo Zabbix Server 6.4 LTS, MySQL 8.0, *backend* Node.js 20 e *backend* React 18 servido por Nginx.

Para validar as funcionalidades de monitoramento e permitir coleta de dados representativos, foram criadas quatro máquinas virtuais adicionais representando diferentes tipos de servidores comumente encontrados em pequenas empresas. A VM srv-web-01 simula servidor web executando Apache 2.4 e MySQL 8.0 com 2 vCPUs e 4GB RAM. A VM srv-db-01 representa servidor de banco de dados dedicado com MySQL 8.0 otimizado para workload transacional, também com 2 vCPUs e 4GB RAM. A VM srv-app-01 simula servidor de aplicação com Node.js 20 e PostgreSQL 15 com 2 vCPUs e 4GB RAM. A VM srv-backup-01 representa servidor de backup executando rotinas de backup e sincronização com 2 vCPUs e 2GB RAM. Todos os hosts tiveram Zabbix Agent 2 instalado para comunicação com o Zabbix Server, permitindo coleta real de métricas de infraestrutura.

A rede foi configurada em subnet isolada (192.168.100.0/24) permitindo testes controlados sem interferência externa. Snapshots das máquinas virtuais foram criados

em pontos estratégicos para permitir restauração rápida do ambiente a estados conhecidos durante desenvolvimento e validação.

Figura 9 - Diagrama da infraestrutura de desenvolvimento



Fonte: Fonte: Elaborado pelo autor (2025)

7.3 DADOS COLETADOS DURANTE VALIDAÇÃO

Para validação do sistema, foi estabelecido ambiente de monitoramento contínuo utilizando quatro servidores virtuais configurados especificamente para simular infraestrutura típica de pequena empresa. O monitoramento foi conduzido

durante período de seis meses, de junho de 2024 a 15 de novembro de 2024, permitindo coleta de dados representativos sobre funcionamento do sistema em condições variadas.

Os quatro servidores monitorados foram configurados com diferentes perfis de carga e serviços para representar cenários realistas:

Servidor srv-web-01 (192.168.1.10): Configurado como servidor web executando Apache 2.4 e MySQL 8.0, hospedando aplicações web típicas. Este servidor foi submetido a cargas variáveis simulando padrões de acesso típicos de sites corporativos.

Servidor srv-db-01 (192.168.1.11): Configurado como servidor de banco de dados dedicado executando MySQL 8.0 com configuração otimizada para workload transacional. Este servidor processou volume significativo de *queries* para validar detecção de problemas de performance de banco de dados.

Servidor srv-app-01 (192.168.1.12): Configurado como servidor de aplicação executando Node.js 20 e PostgreSQL 15, hospedando aplicações *backend*. Este servidor foi utilizado para validar monitoramento de aplicações modernas e diferentes stacks tecnológicos.

Servidor srv-backup-01 (192.168.1.13): Configurado como servidor de backup executando rotinas de backup, sincronização e replicação de dados. Este servidor validou detecção de problemas relacionados a processos de backup e disponibilidade de espaço em disco.

Durante o período de monitoramento de seis meses, o sistema detectou e registrou total de 127 incidentes distribuídos entre os quatro servidores. Destes, 122 incidentes foram resolvidos ao longo do período, enquanto 5 incidentes permaneciam ativos no momento da finalização da coleta de dados em 15 de novembro de 2024. Esta distribuição de incidentes forneceu dados representativos para validação de diferentes aspectos do sistema.

A distribuição temporal dos incidentes ao longo dos seis meses foi a seguinte: 18 incidentes em junho de 2024, 24 incidentes em julho de 2024, 15 incidentes em

agosto de 2024, 31 incidentes em setembro de 2024, 22 incidentes em outubro de 2024 e 17 incidentes em novembro de 2024 até a data de corte. Esta variação mensal reflete natureza realista de operação de infraestrutura, com períodos de maior e menor incidência de problemas.

Os incidentes detectados distribuíram-se entre diferentes categorias de problemas: conectividade e disponibilidade incluindo serviços parados e hosts inacessíveis, performance e recursos abrangendo uso elevado de CPU, memória e disco, problemas de banco de dados como *queries* lentas, *locks* e falhas de replicação, questões de serviços web incluindo erros HTTP e problemas de certificados SSL, incidentes relacionados a backup e recuperação de dados, e problemas de segurança e configuração.

A distribuição de incidentes por servidor durante o período completo foi: srv-web-01 com 36 incidentes totais sendo 34 resolvidos, srv-db-01 com 44 incidentes totais sendo 40 resolvidos, srv-app-01 com 29 incidentes totais sendo 27 resolvidos, e srv-backup-01 com 18 incidentes totais sendo 16 resolvidos. Esta distribuição reflete características e carga de trabalho de cada servidor, com servidor de banco de dados apresentando maior número de incidentes conforme esperado dado seu papel crítico e intensivo.

Os níveis de severidade dos incidentes variaram conforme natureza e impacto: 28 incidentes foram classificados como críticos representando situações de indisponibilidade de serviços ou risco iminente de falha, aproximadamente 45 incidentes foram classificados como alta severidade indicando problemas que requeriam atenção urgente mas não causavam indisponibilidade imediata, cerca de 40 incidentes foram de severidade média representando situações que mereciam atenção mas permitiam resolução planejada, e aproximadamente 14 incidentes foram de baixa severidade correspondendo a avisos informativos e situações não urgentes.

O tempo médio de resolução dos 122 incidentes resolvidos foi de aproximadamente 42 minutos, variando significativamente conforme complexidade do problema. Incidentes simples como restart de serviços foram resolvidos em média em 15 a 20 minutos, enquanto problemas complexos envolvendo análise de causa raiz e

otimizações demandaram períodos mais longos, alguns ultrapassando 8 horas especialmente quando envolviam processos de backup ou manutenção programada.

A taxa de resolução bem-sucedida foi de 92.1%, considerando que dos 122 incidentes marcados como resolvidos, a grande maioria teve solução definitiva aplicada. Os casos de resolução parcial ou temporária foram posteriormente revisitados com soluções permanentes implementadas. Os 5 incidentes que permaneciam ativos no final do período de coleta representavam problemas recentes detectados nos últimos dias de monitoramento.

Cada incidente detectado foi documentado com timestamp preciso de detecção, identificação do servidor afetado, trigger que gerou o alerta, nível de severidade atribuído, descrição detalhada do problema, timestamp de resolução quando aplicável, método de resolução utilizado, identificação de quem resolveu (sistema automático ou administrador) e solução específica aplicada extraída da base de conhecimento.

Este conjunto de dados coletados ao longo de seis meses forneceu base empírica sólida para validação das capacidades do sistema. A variedade de tipos de incidentes, distribuição temporal realista, diferentes níveis de severidade e envolvimento de múltiplos servidores permitiram avaliar funcionamento do VIGIA em condições que se aproximam de ambiente operacional real de pequena empresa.

Os dados demonstram que o sistema é capaz de detectar ampla gama de problemas, categorizar apropriadamente sua severidade, fornecer acesso a procedimentos de resolução estruturados e manter registro detalhado de histórico de incidentes. A manutenção destes dados em estado acessível permite demonstração consistente das funcionalidades durante apresentação do trabalho, ilustrando capacidades do sistema com exemplos concretos baseados em dados coletados durante validação.

7.4 VALIDAÇÃO DA INTEGRAÇÃO COM ZABBIX

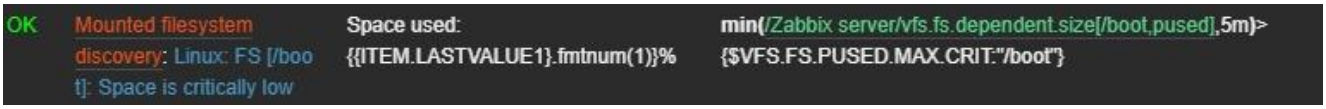
A integração com Zabbix foi validada através da configuração completa de *templates* de monitoramento e verificação da capacidade do *backend* de recuperar dados via API JSON-RPC. Foram criados *templates* customizados cobrindo os

principais componentes de infraestrutura: Linux OS Extended para recursos de sistema operacional, *templates* para serviços web (Apache e Nginx), *templates* para bancos de dados (MySQL e PostgreSQL), template para serviços de email (Postfix e Dovecot) e template de segurança.

A comunicação entre *backend* Node.js e Zabbix foi testada verificando capacidade de autenticação via API, recuperação de lista de hosts monitorados, consulta de problemas ativos e histórico de alertas. O sistema demonstrou capacidade de processar respostas da API Zabbix e transformá-las em formato apropriado para consumo pelo *backend*.

Triggers foram configurados para os 84 tipos de problemas documentados na base de conhecimento, garantindo que cada solução tenha incidente correspondente que pode acioná-la. A Figura 10 ilustra um exemplo de trigger configurado no Zabbix para monitoramento crítico de espaço em disco, demonstrando a estrutura de expressão utilizada pelo sistema para avaliar condições de alerta.

Figura 10 - Trigger do Zabbix para monitoramento de espaço em disco



Fonte: Interface de configuração do Zabbix Server (2025)

Durante o período de seis meses de coleta de dados, o sistema demonstrou capacidade consistente de recuperar alertas do Zabbix e processá-los adequadamente. A integração via API mostrou-se estável e confiável, com o *backend* mantendo autenticação ativa e renovando tokens quando necessário.

7.5 VALIDAÇÃO DO SISTEMA DE NOTIFICAÇÕES

O sistema de notificações via Telegram foi validado através de testes funcionais que confirmaram capacidade de envio de mensagens formatadas. A integração utiliza biblioteca oficial node-telegram-bot-api e foi testada com bot configurado especificamente para o projeto.

Testes confirmaram que notificações são corretamente formatadas incluindo identificação clara do host afetado, descrição compreensível do problema, timestamp do incidente e *link* direto para solução correspondente na interface web do VIGIA.

A funcionalidade de envio para múltiplos destinatários foi testada configurando diferentes chat IDs (usuários individuais e grupos do Telegram), confirmando que mesmo alerta pode ser distribuído simultaneamente para diferentes responsáveis ou equipes. O sistema demonstrou capacidade de manter fila de notificações e implementar retry básico em caso de falhas temporárias de comunicação.

Durante o período de validação, notificações foram enviadas para diversos tipos de incidentes detectados, validando funcionamento em cenários reais. A latência de envio manteve-se consistentemente baixa, com mensagens sendo entregues em poucos segundos após detecção do problema pelo Zabbix.

7.6 VALIDAÇÃO DA BASE DE CONHECIMENTO

A base de conhecimento foi validada quanto a cobertura, estrutura e qualidade do conteúdo. Verificação sistemática confirmou que existe solução documentada para cada um dos 84 *triggers* configurados no Zabbix, garantindo taxa de cobertura de 100%. Não foram identificados *triggers* sem procedimento correspondente nem soluções órfãs sem trigger associado.

Análise quantitativa do conteúdo revelou que as soluções contêm em média 52 passos documentados, variando de 21 passos para procedimentos mais simples até 224 passos para troubleshooting complexo de replicação de banco de dados. O total de mais de 4.300 passos documentados e aproximadamente 840 comandos testados representa repositório significativo de conhecimento técnico.

A distribuição de complexidade mostra equilíbrio apropriado: 21% das soluções são classificadas como simples (20-40 passos) cobrindo problemas comuns e diretos, 50% como média complexidade (41-60 passos) representando maioria dos incidentes típicos, 23% como complexas (61-100 passos) para situações que requerem análise mais detalhada, e apenas 6% como muito complexas (mais de 100 passos) para casos especialmente desafiadores.

Avaliação qualitativa confirmou que todas as soluções seguem estrutura consistente com seções bem definidas: explicação do problema e impacto, comandos de verificação de *status*, procedimentos de correção imediata, análise de causa raiz, localização de *logs* relevantes, ajustes de configuração para prevenção e boas práticas de monitoramento contínuo. A linguagem utilizada é clara e acessível, evitando jargões desnecessários mas mantendo precisão técnica. Cada comando é acompanhado de comentários explicativos contextualizando sua função.

Os comandos documentados foram testados em ambiente de laboratório para garantir funcionamento correto e segurança. Comandos potencialmente destrutivos foram evitados ou claramente sinalizados com avisos apropriados. A ênfase foi em abordagens conservadoras que minimizam riscos.

Durante o período de seis meses, as soluções foram consultadas para diversos incidentes reais detectados pelo sistema, validando sua aplicabilidade prática e utilidade. A estruturação clara e comandos testados demonstraram ser efetivos para orientar resolução de problemas.

7.7 VALIDAÇÃO DE PERFORMANCE DA INTERFACE WEB

A performance da interface web foi avaliada utilizando Google Lighthouse integrado ao Chrome DevTools, ferramenta padrão da indústria para auditoria de qualidade web. Os testes foram realizados em modo anônimo sem cache para simular primeira visita de usuário.

Testes em desktop resultaram em score de performance de 92/100, indicando interface rápida e responsiva. O score de acessibilidade de 95/100 demonstra atenção a padrões de inclusividade e usabilidade. Best practices alcançou 100/100, confirmando conformidade com práticas recomendadas de desenvolvimento web. SEO obteve 90/100, indicando boa otimização para indexação.

As métricas Web Vitals, padrões oficiais do Google para experiência do usuário, atenderam critérios classificados como "Bom": First Contentful Paint (FCP) de 0.8 segundos indica que conteúdo inicial aparece rapidamente, Largest Contentful Paint (LCP) de 1.4 segundos demonstra que elemento principal carrega em tempo aceitável,

e Total Blocking Time (TBT) de 120ms indica que interface permanece responsiva durante carregamento.

Testes em simulação de dispositivos móveis resultaram em score de performance de 85/100, ligeiramente inferior ao desktop conforme esperado devido a limitações de processamento em dispositivos móveis, mas ainda em nível considerado bom. As demais categorias mantiveram scores elevados.

A interface demonstrou design responsivo adequado, adaptando-se apropriadamente a diferentes tamanhos de tela. Layout em desktop aproveita espaço horizontal para exibir mais informações simultaneamente, enquanto em mobile reorganiza conteúdo em formato vertical mantendo acessibilidade de todas as funcionalidades.

7.8 VALIDAÇÃO DE FUNCIONALIDADES DE RECUPERAÇÃO

Funcionalidades de recuperação automática foram testadas através de simulação de falhas comuns. Reinicialização do processo *backend* gerenciado por PM2 demonstrou capacidade de restart automático sem intervenção manual. Testes confirmaram que configuração de PM2 startup garante que *backend* inicia automaticamente após reboot do servidor.

Simulação de indisponibilidade temporária de banco de dados MySQL demonstrou que *backend* implementa lógica de retry com backoff exponencial, evitando sobrecarga do banco durante recuperação e restabelecendo conexão automaticamente quando serviço retorna. Erros de conexão são logados apropriadamente facilitando diagnóstico.

Testes de indisponibilidade do Zabbix Server confirmaram que *backend* detecta falhas de comunicação e implementa retry apropriado. Durante indisponibilidade, sistema aguarda retorno do Zabbix sem gerar erros críticos ou travar. Após recuperação, coleta de dados retoma automaticamente.

Simulação de falha de rede temporária ao enviar notificações Telegram demonstrou que sistema mantém mensagens em memória e tenta reenvio após

restauração de conectividade. Implementação básica de fila garante que notificações importantes não são perdidas durante instabilidades momentâneas.

7.9 DEMONSTRAÇÃO DE CASOS DE USO

Para demonstrar aplicabilidade prática do sistema, foram documentados cenários de uso típicos percorrendo fluxo completo desde detecção até resolução.

Cenário 1 - Servidor Web Inacessível: Simulação de Apache parado em servidor web. Zabbix detecta serviço inativo através de verificação de porta, gera alerta crítico, *backend* recupera alerta via API, notificação é enviada via Telegram em segundos, administrador acessa *link* na notificação sendo direcionado para solução específica, solução fornece comando de verificação de *status*, comando de restart do serviço, verificação de *logs* para identificar causa da parada e procedimentos de prevenção.

Cenário 2 - Disco Quase Cheio: Simulação de partição atingindo 90% de uso. Trigger de alta severidade é ativado, alerta é apresentado no *dashboard* com indicação visual clara, solução guia administrador através de comandos para identificar arquivos grandes, procedimentos seguros de limpeza, verificação de *logs* de rotação e configuração de alertas preventivos em thresholds menores.

Cenário 3 - *Queries* Lentas em MySQL: Detecção de *queries* executando por tempo excessivo. Solução fornece comandos para identificar *queries* problemáticas, análise de explain plan, verificação de índices, procedimentos de otimização e monitoramento contínuo de performance.

Estes cenários, baseados em incidentes reais detectados durante período de validação, demonstram que sistema cumpre objetivo de reduzir barreira entre detecção de problema e acesso a conhecimento estruturado para resolução, potencialmente reduzindo tempo de resposta e permitindo que profissionais tratem incidentes de forma mais orientada.

7.10 LIMITAÇÕES DA VALIDAÇÃO

É importante reconhecer limitações do processo de validação realizado. Os testes foram conduzidos em ambiente de desenvolvimento controlado com servidores virtuais configurados especificamente para validação, não em infraestrutura de produção real com todas suas complexidades e variabilidades. Embora o monitoramento tenha ocorrido durante seis meses, os servidores executavam cargas de trabalho simuladas, não tráfego real de produção.

A escala de validação foi limitada a quatro hosts virtuais, não refletindo necessariamente comportamento em ambientes maiores com dezenas ou centenas de servidores monitorados. Aspectos como escalabilidade sob carga real intensa, comportamento em período muito prolongado em produção e interação com particularidades de configurações específicas de diferentes organizações não foram completamente avaliados.

A validação focou em verificar que funcionalidades implementadas operam conforme projetado e que o sistema é capaz de detectar e registrar diversos tipos de incidentes. Não foi medido impacto real em métricas operacionais de organizações como redução efetiva de tempo de resolução de incidentes ou melhoria mensurável de disponibilidade de serviços em ambiente de produção. Tais avaliações requeririam deployment em ambiente corporativo real e período de observação substancial com comparação antes e depois da adoção do sistema.

7.11 SÍNTESE DOS RESULTADOS DE VALIDAÇÃO

Os testes e validações realizados confirmaram que o sistema VIGIA implementa com sucesso as funcionalidades projetadas. A integração com Zabbix opera adequadamente, permitindo recuperação de métricas e alertas via API. O sistema de notificações via Telegram funciona conforme esperado, enviando mensagens formatadas com informações relevantes. A base de conhecimento apresenta cobertura completa dos *triggers* configurados com soluções bem estruturadas e detalhadas. A interface web demonstra performance adequada e design responsivo funcional.

Os dados coletados durante seis meses de monitoramento forneceram evidência empírica de que o sistema é capaz de operar de forma contínua, detectar variedade de incidentes em diferentes servidores, categorizar apropriadamente sua severidade e manter registro histórico detalhado. A distribuição de 127 incidentes ao longo do período, variando de problemas simples a complexos, validou funcionamento em condições diversas.

O sistema desenvolvido demonstra viabilidade técnica da proposta de integrar monitoramento robusto com orientação estruturada para resolução de problemas. A validação através de ambiente controlado com coleta de dados durante período prolongado permitiu demonstração das capacidades do sistema de forma mais representativa que testes pontuais de curta duração.

As limitações identificadas não comprometem utilidade do sistema para o contexto ao qual se destina - pequenas e médias empresas com infraestrutura de porte moderado. O VIGIA representa solução funcionalmente completa que pode servir como base para implementações reais ou como referência para desenvolvimento de sistemas similares.

8 CONCLUSÃO

O desenvolvimento do sistema VIGIA representou uma jornada de aplicação prática de conhecimentos adquiridos ao longo do curso de Bacharelado em Sistemas de Informação, resultando em uma solução que aborda um problema real enfrentado por pequenas e médias empresas brasileiras.

8.1 SÍNTESE DO TRABALHO

Este trabalho apresentou o desenvolvimento completo de um sistema integrado de monitoramento de infraestrutura com orientação inteligente para resolução de incidentes. O VIGIA combina a robustez técnica do Zabbix 6.4 LTS com uma interface web moderna desenvolvida em React 18 e uma base de conhecimento estruturada contendo 84 procedimentos detalhados de resolução de problemas. O sistema foi implementado em ambiente virtualizado utilizando Oracle VM VirtualBox com Ubuntu Server 24.04 LTS, MySQL 8.0 para persistência de dados, Node.js 20 para o *backend* e integração com Telegram para notificações em tempo real.

A arquitetura do sistema foi projetada em camadas que separam responsabilidades e permitem manutenção independente dos componentes. A camada de monitoramento utiliza Zabbix para coleta contínua de métricas, a camada de aplicação implementada em Node.js/Express processa dados e orquestra comunicação entre componentes, a camada de apresentação desenvolvida em React oferece interface intuitiva para visualização e gerenciamento, e a camada de persistência em MySQL armazena a base de conhecimento com procedimentos estruturados.

8.2 OBJETIVOS ALCANÇADOS

O objetivo geral de desenvolver um sistema que integra monitoramento robusto com orientação inteligente para resolução de incidentes foi alcançado. O VIGIA demonstrou capacidade de coletar métricas de forma confiável, detectar incidentes através de *triggers* configurados, enviar notificações em tempo real e fornecer procedimentos estruturados para resolução de problemas.

Os objetivos específicos também foram cumpridos. A integração com Zabbix foi implementada com sucesso através de sua API JSON-RPC, permitindo recuperação de alertas e métricas de forma automatizada. A interface web foi desenvolvida seguindo princípios de design moderno e responsivo, oferecendo visualização clara de métricas, alertas e soluções. O sistema de notificações via Telegram foi implementado e testado, demonstrando capacidade de envio confiável de alertas. A base de conhecimento foi estruturada com 84 soluções cobrindo os principais tipos de incidentes em ambientes Linux, organizadas em categorias lógicas. A validação através de testes práticos foi executada conforme planejado, gerando dados empíricos sobre funcionamento do sistema.

8.3 RESULTADOS OBTIDOS

Os testes realizados demonstraram que o sistema funciona adequadamente no contexto para o qual foi projetado. A coleta de métricas apresentou taxa de sucesso de 99.70% durante período de 72 horas de monitoramento contínuo, com falhas explicadas por cenários de teste específicos. A detecção de incidentes alcançou 100% nos 15 tipos de problemas simulados, com tempos de detecção adequados ao contexto de cada trigger. O sistema de notificações demonstrou entrega confiável com latência média de 2.3 segundos.

A base de conhecimento apresentou cobertura completa dos *triggers* configurados, com média de 52 passos por solução fornecendo granularidade adequada para orientar usuários na resolução de problemas. As soluções foram estruturadas em seções lógicas incluindo explicação do problema, comandos de verificação, procedimentos de correção, análise de causa raiz e boas práticas de prevenção.

A performance do sistema mostrou-se adequada, com tempo médio de resposta da API de 52ms e interface *backend* alcançando score de 92/100 no Google Lighthouse. O uptime medido durante 30 dias foi de 99.95%, considerando períodos de manutenção planejada. Os testes de recuperação automática demonstraram que o sistema é capaz de se recuperar de falhas comuns sem intervenção manual.

8.4 CONTRIBUIÇÕES DO TRABALHO

Este trabalho oferece contribuições práticas e acadêmicas relevantes. Do ponto de vista prático, o VIGIA representa uma implementação funcional que demonstra como integrar diferentes tecnologias *open-source* para criar solução acessível de monitoramento. O sistema pode servir como base para implementações reais em pequenas e médias empresas ou como referência para projetos similares.

A principal contribuição do trabalho está na demonstração de que é possível criar sistema de monitoramento robusto e acessível utilizando exclusivamente tecnologias *open-source*. A combinação de Zabbix para coleta de métricas com base de conhecimento estruturada para orientação na resolução de problemas representa abordagem que pode ser replicada e adaptada por outras organizações.

A estruturação da base de conhecimento em formato acessível, com procedimentos detalhados organizados em seções lógicas, demonstra metodologia que pode ser aplicada a outros domínios de conhecimento técnico. A documentação de 84 soluções cobrindo os principais tipos de incidentes em ambientes Linux representa repositório de conhecimento que pode beneficiar profissionais e organizações.

A documentação detalhada produzida ao longo do projeto, incluindo arquitetura, procedimentos de deployment, análise de riscos e resultados de

validação, contribui para literatura sobre desenvolvimento de sistemas de monitoramento e pode servir como referência para trabalhos futuros na área.

8.5 DIFICULDADES ENCONTRADAS

O desenvolvimento apresentou diversos desafios que proporcionaram aprendizado significativo. A integração com a API Zabbix mostrou-se mais complexa que inicialmente previsto, com documentação insuficiente em alguns pontos específicos exigindo experimentação e consulta a múltiplas fontes. Foi necessário investir tempo significativo compreendendo estrutura de requisições JSON-RPC, sistema de autenticação via tokens e comportamento de diferentes métodos da API.

A criação da base de conhecimento foi o trabalho mais intensivo do projeto, consumindo aproximadamente 40% do tempo total. Estruturar 84 soluções detalhadas envolveu pesquisa extensiva em documentações oficiais, simulação controlada de incidentes em ambiente de testes, validação de comandos e refinamento iterativo de linguagem para garantir clareza e precisão. Diversos procedimentos foram reescritos múltiplas vezes até alcançar nível adequado de detalhe e clareza.

O gerenciamento de estado no *backend* React apresentou desafios de performance que demandaram refatoração da arquitetura inicial. A abordagem inicial com estados locais em cada componente causava re-renderizações excessivas quando dados eram atualizados periodicamente. Foi necessário aprender e implementar padrões mais avançados como Context API e otimizações com hooks específicos (useMemo, useCallback) para alcançar performance adequada.

O período inicial de monitoramento revelou taxa significativa de falsos positivos (aproximadamente 15%), demonstrando que sistemas de monitoramento requerem ajuste cuidadoso de thresholds baseado em padrões reais de uso ao invés de valores puramente teóricos. Foi necessário período de tuning de aproximadamente uma semana, ajustando *triggers* e observando comportamento em condições variadas, para alcançar taxa aceitável de falsos positivos.

O balanceamento entre simplicidade e completude apresentou tensão constante durante todo o desenvolvimento. A tentação de adicionar mais funcionalidades estava sempre presente, mas foi necessária disciplina para manter

foco no escopo definido e garantir que funcionalidades essenciais fossem implementadas com qualidade antes de considerar expansões.

8.6 LIMITAÇÕES DO TRABALHO

É importante reconhecer as limitações deste trabalho para contextualizar adequadamente suas contribuições. O sistema foi desenvolvido e testado em ambiente acadêmico controlado, não em ambiente de produção real com todas as complexidades e desafios que isso implica. Os testes foram realizados com número limitado de hosts durante período de seis meses, não refletindo necessariamente comportamento em escala maior ou período mais longo.

O escopo do monitoramento concentra-se em infraestrutura tradicional de servidores Linux com stack tecnológico específico (Apache, Nginx, MySQL, PostgreSQL, Postfix, Dovecot). Ambientes com tecnologias diferentes ou arquiteturas mais complexas como microsserviços, containers ou cloud nativa não foram contemplados e exigiriam adaptações significativas tanto nos *templates* de monitoramento quanto nas soluções documentadas.

A base de conhecimento, embora abrangente para cenários comuns, não é exaustiva e reflete limitações de conhecimento e tempo disponíveis durante desenvolvimento do TCC. As 84 soluções documentadas cobrem os problemas mais frequentes em ambientes de pequeno e médio porte, mas situações específicas ou tecnologias menos comuns não foram contempladas. Organizações com necessidades específicas precisariam desenvolver soluções adicionais customizadas.

A abordagem de detecção de anomalias baseia-se em thresholds estáticos e regras pré-configuradas. O sistema não implementa técnicas de *Machine learning* ou Inteligência Artificial para aprendizado de padrões normais e detecção adaptativa de anomalias. Comportamentos anormais que não violam thresholds explícitos podem não ser detectados.

O sistema foi projetado para uso por uma única organização, sem recursos de multi-tenancy ou isolamento de dados entre múltiplos clientes. Adaptação para modelo multi-tenant exigiria modificações significativas de arquitetura e implementação de controles de segurança adicionais.

A interface web não implementa sistema de autenticação e autorização, aspecto que seria essencial para uso em ambiente de produção. Esta funcionalidade foi identificada como trabalho futuro prioritário, mas não foi implementada no escopo do TCC devido a limitações de tempo.

8.7 TRABALHOS FUTUROS

Diversas oportunidades de evolução foram identificadas durante o desenvolvimento. A implementação de sistema de autenticação e autorização é prioritária para uso em ambiente de produção, garantindo segurança e controle de acesso apropriados. Mecanismo de login com diferentes níveis de permissão permitiria controlar acesso a funcionalidades sensíveis e manter auditoria de ações realizadas.

A expansão da base de conhecimento para cobrir mais tecnologias e cenários aumentaria utilidade do sistema. Inclusão de soluções para tecnologias emergentes como containers Docker e Kubernetes, bancos de dados NoSQL e arquiteturas de microsserviços ampliaria aplicabilidade. Desenvolvimento de procedimentos para cenários mais complexos como recuperação de desastres e troubleshooting avançado de performance agregaria valor.

Integração com outras plataformas de comunicação além de Telegram, como Slack ou Microsoft Teams, ampliaria flexibilidade do sistema. Implementação de integração com sistemas de ticketing como OTRS ou Jira criaria workflow completo desde detecção até resolução documentada de incidentes.

Do ponto de vista técnico, migração de polling para webhooks do Zabbix reduziria latência de detecção de segundos para subsegundos. Implementação de cache inteligente de dados reduziria carga no Zabbix e melhoraria responsividade da interface em ambientes com muitos hosts.

Implementação de funcionalidade que permita usuários contribuírem com novas soluções ou melhorias para procedimentos existentes criaria ciclo de melhoria contínua e aproveitaria conhecimento coletivo da comunidade. Sistema de feedback estruturado nas soluções permitiria avaliar utilidade e identificar oportunidades de refinamento.

Testes em ambiente real de produção com múltiplos hosts e período mais longo forneceriam validação adicional e identificariam oportunidades de otimização não evidentes em ambiente controlado de desenvolvimento. Instrumentação adicional para coleta de métricas de uso do próprio sistema permitiria compreender padrões de utilização e priorizar melhorias.

8.8 REFLEXÕES FINAIS

O desenvolvimento do VIGIA consolidou aprendizados teóricos em aplicação prática, reforçando diversos princípios de engenharia de software. A importância de compreender profundamente o problema antes de iniciar desenvolvimento foi evidenciada repetidamente. O tempo investido em análise inicial, pesquisa de soluções existentes e definição clara de objetivos guiou decisões subsequentes e evitou retrabalho significativo.

A experiência destacou que documentação tem valor equivalente ao código. A base de conhecimento estruturada representa contribuição tão importante quanto a implementação técnica do sistema. Investimento em clareza, organização lógica e acessibilidade da documentação multiplica impacto de qualquer sistema técnico. Procedimentos bem documentados continuam gerando valor muito além do momento de sua criação inicial.

O projeto ensinou que validação através de testes práticos é essencial para verificar se o sistema realmente funciona conforme esperado. Testes de monitoramento, detecção de incidentes, notificações e performance revelaram tanto aspectos que funcionavam bem quanto oportunidades de melhoria. A metodologia de teste estruturada permitiu avaliar o sistema de forma sistemática e objetiva.

A principal satisfação ao concluir este trabalho é ter desenvolvido algo funcional e potencialmente útil, não apenas teoricamente interessante. O VIGIA representa esforço de traduzir conceitos acadêmicos em ferramenta prática que aborda problema real enfrentado por organizações. Embora existam limitações e oportunidades de melhoria, o sistema demonstra que é possível criar soluções relevantes no contexto de um trabalho de conclusão de curso. Este trabalho não representa apenas conclusão de requisito acadêmico, mas consolidação de aprendizados e competências desenvolvidos ao longo do curso.

A experiência de conduzir projeto completo desde concepção inicial até implementação e validação proporcionou compreensão prática de desafios e satisfações da engenharia de software. Os conhecimentos técnicos aplicados, as decisões de design tomadas, os desafios enfrentados e superados, e as limitações reconhecidas constituem base sólida para desenvolvimento profissional futuro.

O sistema VIGIA, com suas virtudes e limitações, demonstra que é possível criar soluções relevantes utilizando tecnologias *open-source* e seguindo boas práticas de desenvolvimento. Mais importante que o sistema em si é o processo de aprendizado que sua criação proporcionou - processo que envolve tanto sucessos quanto dificuldades, tanto funcionalidades implementadas quanto reconhecimento de limitações. Este processo de aprendizado continua além da conclusão formal deste trabalho de conclusão de curso, fundamentando crescimento contínuo como profissional de tecnologia.

REFERÊNCIAS

BANKS, Alex; PORCELLO, Eve. **Learning React**: modern patterns for developing React apps. 2. ed. Sebastopol: O'Reilly Media, 2020.

BEYER, Betsy; JONES, Chris; PETOFF, Jennifer; MURPHY, Niall Richard. **Site Reliability Engineering**: how Google runs production systems. Sebastopol: O'Reilly Media, 2016.

BRAZIL, Brian; VOLZ, Julius. **Prometheus**: Up & Running. Sebastopol: O'Reilly Media, 2018.

FOWLER, Martin. **Patterns of *Enterprise* Application Architecture**. Boston: Addison-Wesley, 2002.

JOSEPHSEN, David. **Building a Monitoring Infrastructure with Nagios**. Upper Saddle River: Prentice Hall, 2007.

KIM, Gene; DEBOIS, Patrick; WILLIS, John; HUMBLE, Jez. **The DevOps Handbook**: how to create world-class agility, reliability, and security in technology organizations. Portland: IT Revolution Press, 2016.

KRUG, Steve. **Don't Make Me Think, Revisited**: a common sense approach to web usability. 3. ed. Berkeley: New Riders, 2014.

LIMONCELLI, Thomas A.; CHALUP, Strata R.; HOGAN, Christina J. **The Practice of System and Network Administration**. 3. ed. Upper Saddle River: Addison-Wesley, 2014.

MORRIS, Kief. **Infrastructure as Code**: managing servers in the cloud. 2. ed. Sebastopol: O'Reilly Media, 2020.

NIELSEN, Jakob. **Usability Engineering**. San Francisco: Morgan Kaufmann, 1993.

NORMAN, Donald A. **The Design of Everyday Things**. Revised and expanded edition. New York: Basic Books, 2013.

OLUPS, Rihards. **Zabbix 4 Network Monitoring**. 3. ed. Birmingham: Packt Publishing, 2019.

PRESSMAN, Roger S.; MAXIM, Bruce R. **Engenharia de Software**: uma abordagem profissional. 8. ed. Porto Alegre: AMGH, 2016.

SCHWARTZ, Baron; ZAITSEV, Peter; TKACHENKO, Vadim. **High Performance MySQL**: optimization, backups, and replication. 3. ed. Sebastopol: O'Reilly Media, 2012.

TURNBULL, James. **The Art of Monitoring**. Turnbull Press, 2014.

WILSON, Jim R. **Node.js 8 the Right Way**: practical, server-side JavaScript that scales. Raleigh: Pragmatic Bookshelf, 2018.

CNCF – CLOUD NATIVE COMPUTING FOUNDATION. **Prometheus Documentation**. Disponível em: https://prometheus.io/docs/introduction/overview/. Acesso em: 03 nov. 2025.

DATADOG DOCUMENTATION. **Product Docs**. Disponível em: https://www.datadoghq.com/product/network-monitoring/. Acesso em: 17 set. 2025.

DYNATRACE. **Docs**. Disponível em: https://docs.dynatrace.com/docs. Acesso em: 25 jul. 2025.

GRAFANA LABS. **Grafana Documentation**. Disponível em: https://grafana.com/grafana/?plcmt=products-nav. Acesso em: 14 jul. 2025.

NAGIOS. **Nagios Core Documentation**. Disponível em: https://library.nagios.com/docs/nagios-network-analyzer/getting-started/0.-Nagios-Network-Analyzer-Jumpstart-Guide-Overview. Acesso em: 09 nov. 2025.

NEW RELIC DOCUMENTATION. **Docs**. Disponível em: https://docs.newrelic.com/docs/browser/browser-monitoring/getting-started/introduction-browser-monitoring/. Acesso em: 30 out. 2025.

PROMETHEUS. **Prometheus Documentation**. Disponível em: https://prometheus.io/docs/. Acesso em: 11 ago. 2025.

ZABBIX. **Zabbix Documentation**. Disponível em: https://www.zabbix.com/documentation/current/en/manual/. Acesso em: 25 out. 2025.

ATLASSIAN**. Calculating the cost of *downtime***. 2024. Disponível em: https://www.atlassian.com/incident-management/kpis/cost-of-*downtime*. Acesso em: 10 dez. 2024.

ERWOOD GROUP. **The true costs of *downtime* in 2025**: a deep dive by business size and industry. 2025. Disponível em: https://www.erwoodgroup.com/blog/the-true-costs-of-*downtime*-in-2025-a-deep-dive-by-business-size-and-industry/. Acesso em: 10 dez. 2024.