UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

GUILHERME ALMEIDA LOPES

DESENVOLVIMENTO DE UM AGENTE LEVE PARA MONITORAMENTO DE PROCESSOS EM LINUX EM UMA ARQUITETURA BIG DATA VOLTADA À CIBERSEGURANÇA

CAMPO MOURÃO, PR, BRASIL

2025

versão 11.0 (abr. 25)

GUILHERME ALMEIDA LOPES

DESENVOLVIMENTO DE UM AGENTE LEVE PARA MONITORAMENTO DE PROCESSOS EM LINUX EM UMA ARQUITETURA BIG DATA VOLTADA À CIBERSEGURANÇA

DEVELOPMENT OF A LIGHTWEIGHT AGENT FOR MONITORING PROCESSES IN LINUX IN A BIG DATA ARCHITECTURE FOCUSED ON CYBERSECURITY

Trabalho de Conclusão de Curso de Graduação apresentado (a) como requisito para obtenção do título de Bacharel em do Bacharelado em Ciência da Computação da Universidade Tecnológica Federal do Paraná.
Orientador(a): Rodrigo Campiolo

CAMPO MOURÃO, PR, BRASIL

2025

AGRADECIMENTOS

É inevitável que estes parágrafos não consigam abranger e nominar todas as pessoas que foram cruciais nesta importante fase da minha vida. Por essa razão, peço desculpas àquelas que não estão presentes nestas palavras, mas que saibam que são parte do meu pensamento e da minha imensa gratidão.

Contudo, o meu agradecimento principal é dedicado aos meus pais, Eduardo Capriotti Lopes e Elaine De Deus Almeida, e às minhas irmãs, Lívia de Almeida Lopes e Laura Almeida Lopes, por serem meu pilar.

Um agradecimento especial ao meu orientador, Prof. Dr. Rodrigo Campiolo, pela sabedoria, confiança e precisão com que me guiou durante toda esta jornada.

Aos meus colegas de sala, Felipe Gomes, Gabriel Viera de Souza, Felipe Luiz Corumbá e Caio Rangel, o meu reconhecimento pela colaboração e companheirismo.

RESUMO

O trabalho a seguir apresenta a proposta de um agente de monitoramento de processos local, projetado para ser o mais leve possível, escalável em ambientes reais e capaz de evitar a inundação da rede com dados desnecessários, além de ser minimamente intrusivo. Integrado a uma arquitetura de Big Data, o agente permitirá coletar diversas métricas de processos, e enviar para a arquitetura para reconhecer anomalias de funcionamento e visualizar as informações coletadas de forma eficiente. Como resultado preliminar, foi possível desenvolver um protótipo do agente de monitoramento e implementar a visualização em tempo real dos dados obtidos.

Palavras-chave: sistema operacional; linux; detecção de anomalias; visualização em tempo real.

LISTA DE ILUSTRAÇÕES

Figura 1 – Arquitetura Prometheus **9**
Figura 2 – Arquitetura Zabbix **10**
Figura 3 – Arquitetura Nagios **12**
Figura 4 – Arquitetura Big Data **14**
Figura 5 – Diagrama de Pesquisa **21**
Figura 6 – Arquitetura do Protótipo **27**
Figura 7 – Grafico de CPU vs. Intervalo fixo entre coletas (segundos) **28**
Figura 8 – Visualização **29**

SUMÁRIO

1 INTRODUÇÃO

Diante da crescente expansão da computação na sociedade e do acesso cada vez mais amplo à Internet e a dispositivos eletrônicos, torna-se fundamental monitorar esses equipamentos para identificar anomalias em seu funcionamento, como possíveis tentativas de ataques cibernéticos. Tais ataques, perpetrados por terceiros, visam obter benefícios indevidos ou causar prejuízos significativos. Um exemplo notório dessa ameaça, como pode ser visto na matéria **??**), no qual descreve que aconteceu no Hospital da Criança de Brasília em 2023, quando a instituição sofreu um ataque de *ransomware*, resultando em um prejuízo estimado em 5,5 milhões reais.

Para mitigar esses riscos, diversas ferramentas são utilizadas no diagnóstico do funcionamento de sistemas computacionais. O SNMP como apresentado no estudo (**??**), permite obter estatísticas detalhadas sobre a rede e acompanhar a comunicação entre os dispositivos.

Sistemas de monitoramento são amplamente utilizados no dia a dia como apresentado na matéria escrita **??**) para verificar o funcionamento de sistemas computacionais, obtendo métricas que indicam o desempenho e a saúde do sistema. Para isso, geralmente existe um agente local que coleta essas métricas e as envia a um agente externo que as armazena e as exibe para terceiros.

O principal problema que pode ser causado pelo monitoramento é o gasto excessivo de recursos como CPU, como é descutido no artigo apresentado proposto por **??**). Isso ocorre, principalmente, se o agente local não possuir políticas que garantam que seu funcionamento seja o minimamente intrusivo, pode causar atrasos no tempo de resposta. A falta dessas políticas pode comprometer o desempenho geral do sistema, prejudicando a justiça na alocação de recursos computacionais, trazendo também vieses em analise nas métricas, devido ao alto consumo dos recursos pelo agente local.

Além disso, softwares de monitoramento como o Nagios (**??**) e o Prometheus (**??**) não apresentam a capacidade de monitorar, métricas específicas de processos, focando em estatísticas gerais de funcionamento dos nós. Conforme observado em suas documentações, há a necessidade de informar manualmente o processo a ser monitorado, o que dificulta a obtenção de insights detalhados sobre processos rodando em um determinado host. Essa limitação compromete a realização de auditorias, uma vez que impede o acesso a dados mais granulares e específicos do ambiente.

Softwares de monitoramento como o Prometheus e o Nagios não possuem a capacidade de monitorar aspectos muito específicos como pode ser visto **??**) e na documentação do prometheus **??**), em grande parte devido à falta de mecanismos que garantam uma ingestão massiva de dados. Isso ocorre porque suas arquiteturas não foram modeladas para essa finalidade, devido a falta de tecnologias . Atualmente, com a crescente necessidade de monitorar métricas granulares de recursos computacionais e um número cada vez maior de hosts, torna-se imprescindível o desenvolvimento de arquiteturas projetadas especificamente para essa propriedade, conforme apresentado nos trabalhos de **??**) e , além disto arquitetura big data explorando mé-

tricas de processos não tem se mostrado muito explorada na litetura, focando mais em log e métricas de rede.

1.1 Objetivos

Objetiva-se neste trabalho implementar um agente de monitoramente local, que seja capaz de enviar métricas para uma arquitetura já implementada no trabalho de **??**), que consiga ajudar administração da infraestrutura distribuídas monitoramento processos.

Têm-se como objetivos específicos:

- desenvolver um agente leve que consuma menos recursos locais e reduza o volume de dados transmitidos pela rede;
- implementar um algoritmo eficiente que seja capaz de analisar uma grande quantidade de dados em curto período de tempos, para detecção de padrões atípicos relacionados a processos, no qual será implementado na arquitetura *big data*;
- avaliar a funcionalidade e a escalabilidade da solução em um ambiente real, no qual será analisado a latências e escalabilidade da solução.

1.2 Estrutura do trabalho

O Capítulo 2 descreve a fundamentação teórica. No Capítulo 3, é detalhada a proposta do trabalho. No Capítulo 4, são detalhados os resultados preliminares. Por fim, o Capítulo 5 apresenta as considerações finais.

2 REFERENCIAL TEÓRICO

Este capítulo apresenta os principais conceitos para o pleno entendimento deste trabalho, incluindo tópicos como: processos, segurança cibernética, detecção de anomalias e arquitetura de monitoramento.

2.1 Segurança cibernética

A Segurança Cibernética é a área da computação que visa garantir a proteção e o acesso restrito (somente a pessoas autorizadas) aos recursos computacionais. Essa proteção é fundamentada em três pilares, também conhecidos como Tríade CID (Confidencialidade, Integridade e Disponibilidade) como pode ser visto no livro **??**):

- Integridade: assegura que os recursos e dados computacionais permaneçam corretos, consistentes e protegidos contra alterações indevidas ao longo de todo o seu ciclo de vida.
- Confidencialidade: garante que o acesso aos recursos e às informações computacionais seja restrito apenas a usuários, processos ou entidades devidamente autorizados.
- Disponibilidade: assegura que os recursos computacionais estejam acessíveis e operacionais sempre que forem requisitados por usuários ou sistemas autorizados.

No entanto, a crescente complexidade dos sistemas computacionais atuais faz com que a garantia dessas propriedades se torne mais difícil. Por essa razão, a aplicação de algoritmos inteligentes que possuem a capacidade de reconhecer padrões àtipicos, tem crescido consideravelmente na área de cibersegurança, conforme demonstrado no livro **??**).

Essa abordagem é justificada pelo panorama atual da computação distribuída. Com a proliferação de sistemas executando em múltiplos nós e ambientes virtualizados, a garantia das propriedades de segurança cibernética se torna significativamente mais complexa. Por essa razão, a aplicação de algoritmos de inteligência artificial em sistemas computacionais, visando a detecção proativa de anomalias, tem se tornado uma prática cada vez mais necessária.

2.2 Monitoramento Tradicional

O monitoramento tradicional de sistemas baseia-se em ferramentas que coletam métricas de desempenho, como uso de CPU, memória e tráfego de rede. Entre os principais exemplos, destacam-se soluções como *Nagios* (**??**), *Zabbix* (**??**) e *Prometheus* (**??**), amplamente utilizadas em ambientes corporativos para acompanhamento de disponibilidade e alertas. No entanto, essas abordagens dependem de limiares estáticos e regras fixas, o que reduz a eficácia diante de comportamentos dinâmicos e ambientes complexos. Trabalho como (**??**) exploram

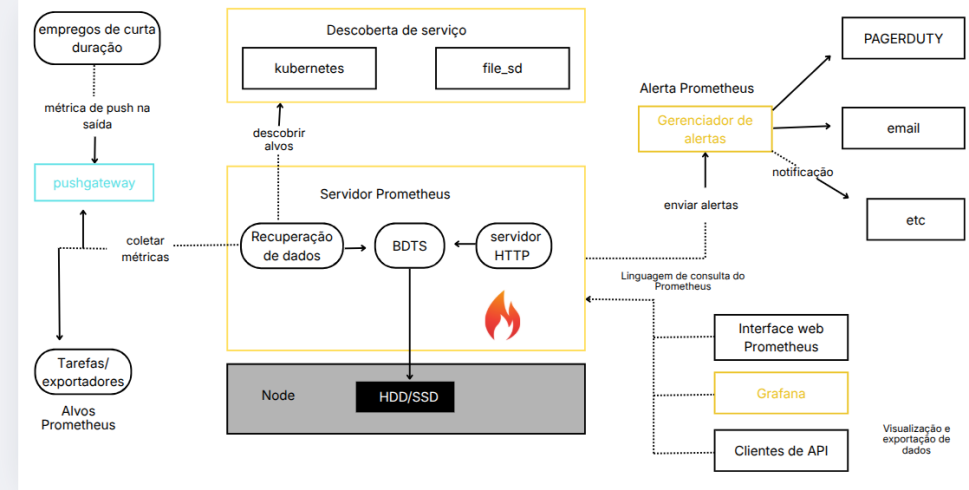
a coleta estruturada de métricas do sistema operacional, ressaltando a importância de análises temporais e históricas para a detecção de anomalias.

A seguir, são apresentadas e discutidas algumas das arquiteturas de monitoramento tradicionalmente utilizadas, na indústria:

2.2.1 Prometheus: Arquitetura e Componentes

A arquitetura prometheus é software, que permite monitorar recursos computacionais como: memória, CPU, aramazenamento,processos.Permitindo monitorar saúde dos sistemas computacionais, sendo sua arquitetura divida em 6 componentes (Figura **??**).

Figura 1 – Arquitetura de monitoramento Prometheus



Fonte: Adptada da documentação ??).

- Servidor Prometheus: módulo central da arquitetura, responsável por coletar, armazenar e processar as métricas de monitoramento.
- Alvos de monitoramento: *endpoints* ou serviços a serem monitorados, a partir dos quais o Servidor Prometheus realiza o *scraping* das métricas.
- Pushgteway: componente utilizado para monitorar alvos de curta duração, como *jobs* batch. As métricas são enviadas para o *Pushgteway*, que as disponibiliza para a coleta posterior pelo Servidor Prometheus.
- Descoberta de serviços: mecanismo que identifica e registra automaticamente novos alvos, facilitando a gestão em ambientes dinâmicos.
- Gerenciador de alertas: módulo responsável pelo tratamento e envio de notificações de alerta ao usuário sobre anomalias ou condições predefinidas nos serviços monitorados.
- Grafana: ferramenta de visualização utilizada para apresentar os dados coletados em painéis interativos e dashboards.

A arquitetura Prometheus permite então o monitoramento eficientes de recursos computacionais, como também o armazenamento de séries temporais. Permitindo o monitoramento de qualquer sistemas computacional, sendo extremamente flexivel e moderno.Ao combinar coleta contínua, descoberta automática de serviços e mecanismos robustos de notificação, o Prometheus se destaca como uma solução amplamente adotada tanto na indústria quanto na academia, servindo como referência para o desenvolvimento de abordagens mais avançadas de monitoramento e detecção de anomalias.

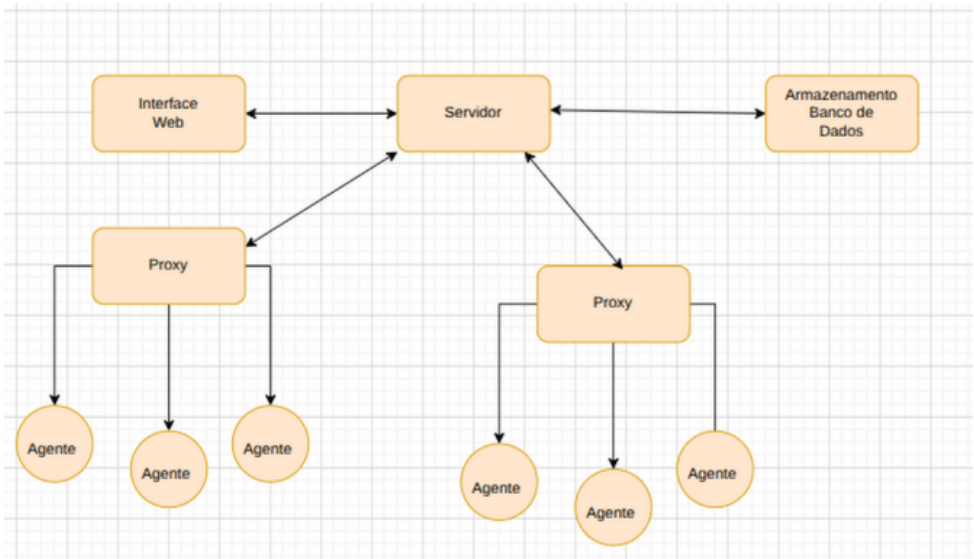
O Prometheus diferencia das demais arquiteturas de monitoramento tradicionais , por possuir um *Pushgteway*, no qual permite monitorar processos de curta duração, diferenciando das Arquitetura como Zabbix (**??**) e Nagios (**??**), permitindo monitoramento de atividades com período de vida curto, permitindo assim analisar scripts.Além disto, o prometheus possui a capacidade de monitoramento de processos usando o /proc

2.2.2 Zabbix: arquitetura e componentes

O Zabbix apresenta uma arquitetura robusta dividida em cinco componentes principais. Diferentemente de outras ferramentas de monitoramento como o Prometheus, o Zabbix permite uma comunicação *full-duplex* (bidirecional), onde tanto o agente quanto o servidor podem iniciar a solicitação de dados, resultando em maior flexibilidade e controle.

Os componentes que compõem a arquitetura do Zabbix são, apresentados conforme a documentação oficial (**??**):

Figura 2 – Arquitetura de monitoramento Zabbix



Fonte: Adptada da documentação ??).

- Servidor: é o componente central do sistema. Ele recebe informações dos agentes e proxies, realiza o processamento dos dados coletados, gerencia as configurações e armazena as métricas no banco de dados.

- Armazenamento de Dados (Banco de Dados): Este módulo realiza a persistência de dados. Toda a informação coletada (métricas), a configuração do Zabbix e os dados de *status* são armazenados em um sistema de gerenciamento de banco de dados relacional.

- Interface Web (*Frontend*): responsável por fornecer a visualização dos dados coletados (gráficos, *dashboards*) e a configuração de todos os módulos do Zabbix, sendo o principal ponto de interação do usuário com o sistema.

- *Proxy*: atua como um gerenciador de carga e coletor de dados intermediário. É responsável por coletar informações de agentes em um local remoto ou em uma rede segmentada e enviá-las ao servidor. Seu uso é descentralizar o processamento e evita a sobrecarga do servidor central.

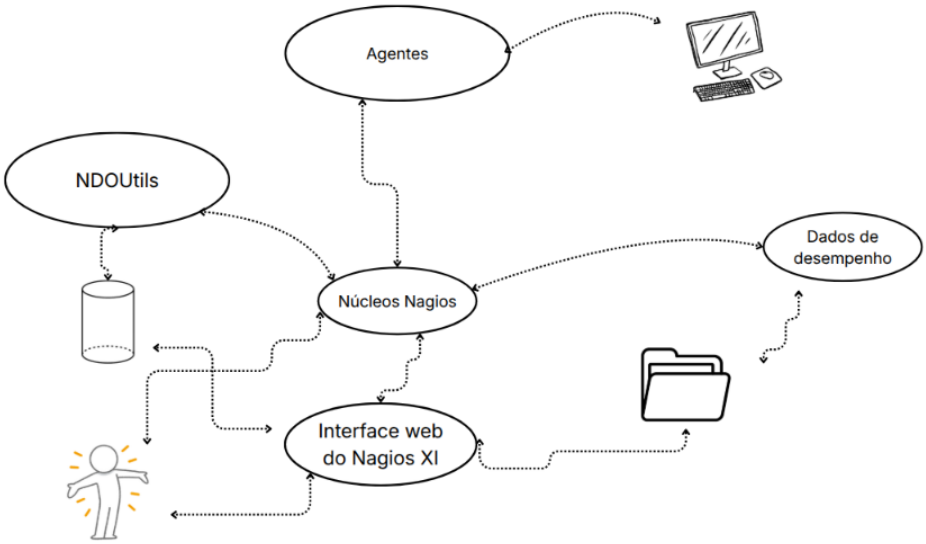
- Agente: é o sensor/coletor de métricas que é instalado diretamente na máquina ou serviço que se deseja monitorar. Ele coleta dados de desempenho e disponibilidade e os reporta ao Server ou a um *Proxy*.

A arquiteutra do Zabbix oferece uma solução altamente estruturada para monitoramento de sistemas, combinando uma coleta ativa e passiva, oferecendo uma solução completa e altamente escálavel. A presença de componentes como o Proxy e a comunicação bidirecional entre agentes e servidor permite ao Zabbix operar eficientemente em ambientes distribuídos, redes segmentadas e cenários de grande escala, trazendo um diferencial do prometheus.

2.2.3 Nagios: Arquitetura e Componentes

O Nagios é um serviço de monitoramento essencial que permite a supervisão contínua de diversos elementos críticos de infraestrutura, incluindo a rede, o uso de CPU e a memória, entre outros. Sua arquitetura modular é ilustrada na Figura **??** (**??**)

Figura 3 – Arquitetura Nagios



Fonte: Adptado da documentação ??).

- Agentes: funcionam como sensores responsáveis por coletar os dados de monitoramento dos dispositivos e serviços.
- Núcleos Nagios: é o servidor central que processa as informações recebidas, avalia o estado dos recursos e gera os alertas necessários em caso de anomalias.
- Dados de Desempenho: refere-se ao armazenamento das métricas de desempenho, frequentemente visualizadas por meio de gráficos para análise histórica.
- NDOUtils (*Nagios Data Output Utilities*): é a ferramenta que se encarrega de enviar e organizar os dados de monitoramento, geralmente direcionando-os para um banco de dados para persistência.
- Interface Web Nagios XI: consiste na interface gráfica utilizada para a visualização das informações monitoradas, a emissão de relatórios e a gestão do sistema.

A arquitettura do Nagios é considerado um modelo tradicional monitoramneto, baseado em agentes, processamento centralizado, com um modelo para armazenamento e visualização, mesmo com sua arquitetura rígida, o Nagios permanence amplamente utilizado devido a sua estabilidade e capacidade de adaptação a diferentes ambientes de infraestrutura. A combinação entre coleta distribuída, análise centralizada e suporte a integrações como o NDOUtils permite

que ele ofereça uma solução confiável para acompanhamento contínuo do estado dos sistemas, mantendo seu papel relevante no ecossistema de monitoramento corporativo.

2.3 Arquitetura de Monitoramento Big Data

A garantia da consistência e disponibilidade dos recursos computacionais constitui um requisito essencial para a continuidade operacional das organizações. A ocorrência de anomalias nesses ambientes pode comprometer diretamente a execução de processos críticos e impactar a estabilidade dos serviços. Nesse contexto, torna-se imprescindível a realização de análises contínuas e sistemáticas sobre o comportamento dos sistemas computacionais. Entretanto, tal atividade apresenta elevada complexidade, decorrente tanto do aumento expressivo na quantidade de dispositivos conectados quanto da complexidade topológica das interconexões entre eles.

Dessa forma, a coleta e o monitoramento de métricas e logs de desempenho assumem papel central na detecção de falhas, anomalias e degradações de serviço. Contudo, o volume massivo de dados gerados por esses sistemas exige a utilização de arquiteturas especializadas, capazes de processar, armazenar e correlacionar grandes quantidades de informações em tempo baixo de resposta.

Um exemplo representativo desse tipo de abordagem é apresentado por **??**), que propôs uma arquitetura voltada ao monitoramento de métricas de processos do sistema operacional, abrangendo parâmetros como uso de CPU, memória, operações de entrada/saída e tráfego de rede. A solução desenvolvida realiza a coleta e o armazenamento estruturado desses dados, preservando também a informação temporal de cada amostra, o que viabiliza consultas eficientes e análises históricas sobre o comportamento dos recursos monitorados.

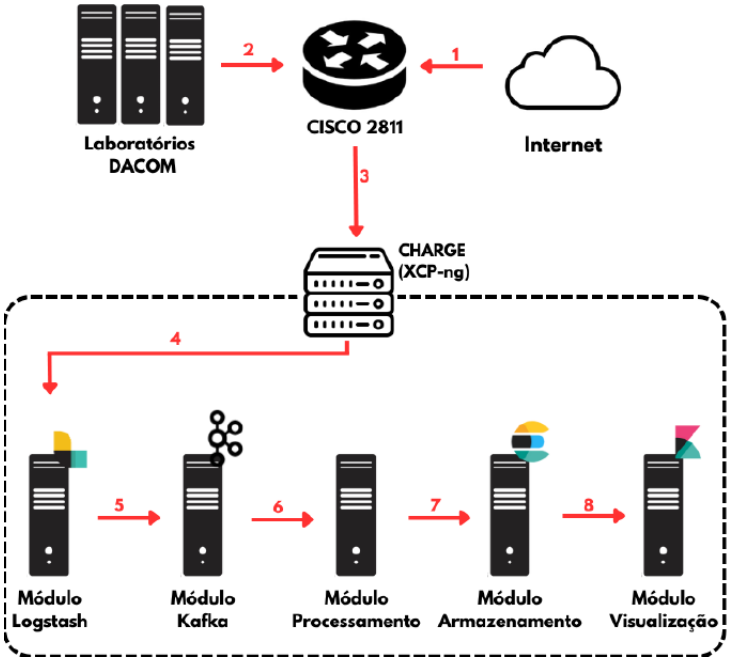
As arquiteturas de Big Data têm se tornado cada vez mais frequentes atualmente, devido à grande quantidade de dados gerados por diversas aplicações e ações empresariais. Um exemplo de arquitetura de Big Data aplicada ao monitoramento do protocolo *NetFlow* é apresentado no trabalho **??**).

Essa arquitetura possibilitou a obtenção de *insights* sobre as redes, dada a sua capacidade de processar uma grande variedade e volume de dados. Isso resultou em um aumento considerável na eficácia do monitoramento e no acesso aos laboratórios do DACOM

O objetivo deste trabalho é expandir a capacidade de monitoramento dessa arquitetura, tornando-a capaz de monitorar processos executando em diversas máquinas. Nesse sentido, trabalhos relacionados, como **??**) e **??**), demonstraram a capacidade de encontrar anomalias de sistema analisando múltiplas métricas de séries temporais, ideal para processos.

A arquitetura apresentada no trabalho de **??**) pode ser vista na Figura **??**.

Figura 4 – Arquitetura Big Data



Fonte: Monografia ??).

- Fluxo 1 (Entrada para a Internet): tráfego de pacotes destinado ao roteador externo (Internet).
- Fluxo 2 (Entrada Interna - DACOM): tráfego de pacotes destinado ao roteador da rede interna do DACOM.
- Fluxo 3 (Coleta Centralizada): o tráfego de rede capturado e roteado para o servidor **Charge**. Este servidor atua como o ponto de entrada e hospeda a arquitetura central de processamento de Big Data.
- Fluxo 4 (Processamento Inicial com Logstash): o fluxo de dados recebido pelo servidor **Charge** é encaminhado para a primeira Máquina Virtual (VM). Nesta VM, o serviço Logstash está em execução, responsável pelo processamento massivo, padronização, filtragem e enriquecimento inicial dos dados.
- Fluxo 5 (Fila de Mensagens - Apache Kafka): os dados pré-processados são enviados para o Apache Kafka. O Kafka atua como um message broker (agente de mensagens) distribuído, garantindo o desacoplamento e o processamento assíncrono dos dados, além de fornecer durabilidade e alta vazão para a ingestão subsequente na arquitetura.
- Fluxo 6 (Processamento Distribuído - Apache Spark): os dados são consumidos do Apache Kafka e enviados para o Apache Spark. O Spark realiza o processamento paralelo e massivo dos dados (análise, transformações e cálculos), extraindo insights e preparando-os para a etapa de armazenamento e visualização.
- Fluxo 7 (Indexação - Elasticsearch): os dados já processados pelo Apache Spark são enviados ao Elasticsearch. O Elasticsearch serve como o datastore (repositório de

dados) e motor de busca distribuído, responsável por indexar os dados de forma otimizada, formando o pipeline de entrega de dados em tempo real para o Kibana.

- Fluxo 8 (Visualização - Kibana): o Kibana acessa e consome os dados indexados no Elasticsearch. Ele é a interface utilizada para a criação de painéis (dashboards), gráficos e relatórios, permitindo a visualização interativa dos resultados obtidos a partir do processamento do Big Data.

O objetivo deste projeto é estender a arquitetura existente, habilitando-a para a ingestão e o processamento de métricas de desempenho provenientes de sistemas *Linux*. A coleta de dados será intermediada pelo *Apache Kafka*, que atuará como um *broker* para o *pipeline* de dados, e as métricas serão então persistidas e disponibilizadas para a plataforma de arquitetura para posterior análise e visualização.

2.4 Processos

O conceito de processos é muito importante para cybersegurança, sabendo que esta sendo executado em sistemas computacionais é fundamental para a garantia de confidencialidade, integridade e disponibilidade, por este motivo conceitos de processos são fundamentais para a segurança da informação, exisitindo diversas ferramentas para a garantia de segurança cybernética.

De acordo com Tanenbaum,(**??**), um processo é, fundamentalmente, a abstração de um programa em execução. Ele possui sua própria região de memória e executa até o término de sua ação.

O sistema operacional (SO) costuma executar diversos processos simultaneamente. O SO é responsável por implementar o conceito de justiça na distribuição dos recursos computacionais entre eles. Contudo, existem processos maliciosos que podem comprometer o funcionamento do SO e ameaçar a segurança do sistema.

Por esta razão, o SO possui ferramentas de monitoramento de processos por padrão, como os comandos ps e htop no Linux, entre outros recursos. No entanto, conforme apresentado em (**??**), atualmente se observa a necessidade de utilizar algoritmos inteligentes para a detecção de ameaças, devido à complexidade de sistemas computacionais modernos.

No contexto do Linux, cada processo é representado por uma estrutura denominada `task_struct`, que armazena informações essenciais como identificador do processo (PID), estado de execução, prioridade, uso de CPU (Central Processing Unit) e memória, ponteiros para arquivos abertos, entre outros metadados. O sistema de arquivos virtual `/proc` fornece uma interface rica para o monitoramento desses processos, permitindo que administradores e ferramentas acessem dados em tempo real, como consumo de recursos, threads ativas e contexto de execução. Essa arquitetura baseada em arquivos faz com que Linux altamente transparente e flexível para análise e auditoria de processos.

Além disso, diversos trabalhos mostram que é possível identificar anomalias por meio de análises dinâmicas de processos, observando características como uso de CPU(Central Processing Unit), acessos à memória e operações de entrada e saída. Esse tipo de análise fornece uma visão temporal e contextual do comportamento dos processos, permitindo distinguir atividades legítimas de possíveis ameaças.

Dessa forma, a integração de algoritmos inteligentes de monitoramento com os mecanismos tradicionais do SO representa um avanço crucial para a segurança e confiabilidade de sistemas modernos.

2.5 Agentes de Monitoramento

O monitoramento é um tema amplamente explorado em arquiteturas distribuídas. Dada a sua complexidade e a extensa distribuição de nós em diversos computadores, a necessidade de monitoramento para prevenir e responder a falhas é crucial, constituindo um dos tipos de mecanismos de tolerância a falha.

O desenvolvimento de agentes de monitoramento, particularmente em contextos de microsserviços, é uma prática corrente, conforme demonstrado no estudo **??**). Adicionalmente, essa abordagem reforça os pilares de disponibilidade de recursos do sistema.

Segundo a definição proposta por **??**), um agente de monitoramento pode ser conceituado como um sensor responsável pela coleta de dados do ambiente, cuja componente inteligente processa essas informações para deliberar e executar uma ação apropriada.

Diversas ferramentas foram desenvolvidas para o monitoramento de servidores, conforme exemplificado no trabalho de referência **??**). Este estudo, especificamente, demonstra a criação de um agente dedicado a monitorar de Máquinas Virtuais (VMs) em ambientes Máquina Virtual Baseada em Kernel (KVM). Adicionalmente, softwares cruciais para a coleta e análise de métricas de CPU incluem o Prometheus (**??**) e o Zabbix (**??**), evidenciando a necessidade de desenvolvimento de agentes de monitoramento.

2.6 Detecção de anomalias

As técnicas de detecção representam uma evolução significativa na capacidade de processamento dos sistemas computacionais. Em contraste com as operações estritamente aritméticas executadas por máquinas tradicionais, esses modelos empregam algoritmos avançados (como ML,*Deep Learning* e *Unsupervised Machine Learning*),tais métodos são capazes de reconhecer padrões complexos e específicos a partir de grandes volumes de dados, permitindo, inclusive, o processamento de anomalias em tempo curto de resposta.

Essa capacidade preditiva e de reconhecimento de padrões tem impulsionado uma verdadeira revolução na área de segurança computacional. Conforme apresentado em **??**), observa-se uma aplicação crescente e inovadora de modelos inteligentes para o reconhecimento e a mitigação de vulnerabilidades.

A empregabilidade desses modelos manifesta-se em diversas frentes, como é evidenciado pelo trabalho de **??**). Os autores destacam o uso de redes neurais, voltada à detecção de *malware*, o que mostra a versatilidade e o poder preditivo dessa abordagem no campo da segurança digital.

A inovação reside na eficiência e velocidade com que os modelos inteligentes conseguem identificar anomalias,sejam elas softwares maliciosos, *exploits* ou falhas de hardware, superando a capacidade de análise humana em termos de volume e rapidez. Adicionalmente, **??**) demonstra que o reconhecimento de anomalias pode ser realizado por meio de aprendizagem não supervisionada, utilizando técnicas como a clusterização (com o algoritmo K-means, por exemplo) em combinação com o One-Class SVM. Os autores ressaltam que a baixa latência é um fator fundamental para a segurança cibernética.

Por esse motivo, grande parte dos estudos considerados inovadores na área de segurança tem se concentrado no desenvolvimento e na aplicação de algoritmos inteligentes para extrair insights valiosos a partir dos dados gerados continuamente pelos sistemas computacionais.

2.6.1 Tipos de anomalias

Como apresentado por **??**), tipos de anomalias pode ser divididas em três tipos:

- Anomalias Pontuais (Point Anomalies): são desvios dramáticos e globais (*outliers*) que se destacam muito dos dados ordinários, um exemplo disso é uma temperatura de 50°C em uma região com média de 25°C).
- Anomalias Contextuais (Contextual Anomalies): um ponto de dados que é incomum apenas em um contexto ou situação específica, mas que seria normal em outro contexto, um exemplo disto é 30°C é normal nos trópicos, mas excepcional no Ártico.
- Anomalias Coletivas (Collective Anomalies): um aglomerado ou sequência de pontos de dados conectados que, quando vistos em conjunto, desviam-se significativamente da tendência, mesmo que os pontos individuais não o façam um exemplo disso é uma série de tentativas de *logins* no mesmo local em pouco tempo que, juntos, podem indicar um ataque.

A análise dos diferentes tipos de anomalias é fundamental para o entendimento de potenciais problemas nos sistemas e de diferentes ataques computacionais. Um exemplo disso são os ataques de DDoS(Distributed Denial of Service), que frequentemente se caracterizam por anomalias pontuais. Por este motivos, é importante entender os diferentes tipos de anômalias para manter a segurança dos sistemas computacionais.

2.7 Trabalhos relacionados

Nesta seção, descrevem-se os trabalhos relacionados ao objetivo deste estudo.

2.7.1 Arquitetura de monitoramento de serviços

As arquiteturas de monitoramento de serviços são amplamente exploradas na literatura, dada a crítica necessidade de se monitorar serviços para o diagnóstico da saúde de servidores e sistemas. Essa prática é fundamental para garantir a estabilidade e o desempenho das aplicações.

Trabalhos como os de **??**) e **??**), demonstram a relevância do tema ao apresentar abordagens e soluções que visam aprimorar a capacidade de observação e resposta a eventos em ambientes de TI.

2.7.2 Detecção de anomalias com aprendizagem não supervisionada e aprendizagem supervisionada

A detecção de anomalias utilizando ML é um campo de pesquisa extremamente ativo e de alto impacto no monitoramento de serviços e sistemas complexos. O objetivo central é identificar desvios sutis ou significativos do comportamento considerado normal (baseline), que, quando detectados rapidamente, podem indicar falhas, vulnerabilidades de segurança ou ineficiências operacionais.

Os trabalhos com aprendizagem não supervisionada , segundo **??**), são capaz de serem utilizados nos grandes volumes de dados, tendo um maior destaque em questão devido a não necessidade de não rótular o dado, o que tem apresentado como desafio devido ao grande volume de dados gerados.

Também trabalhos como detecção de *malware*, usando métricas de processo, utilizando arquiteturas transformes pode ser visto no artigo (**??**), tendo uma taxa bem alta de acertos perante a detecção de *malwares*, mostrando que tem sido bem positivo a utilização de Inteligências computacional para área de segurança.

2.7.3 Arquiteturas Processamento Massivo de Dados

A arquiteturas de processmaneto massivo de dados tem se tornando bastante popula, devido a grande quantidade de dados ingestão de dados produzidas por ela, mas na cybersegurança tais arquiteturas tem sido exploradas bastante devido a grande de lidar com informações.

No trabalho de **??**) é apresentada uma arquitetura big data focada na cybersegurança, nela é utilizada campos obtidos pelo protocolo *NetFlow* de computadores ligados ao laboratórios DACOM (Departamento de Computação), permitindo assim obter dados sobre a rede. A

diferença entre meu trabalho é as métricas recolhidas será de processos linux e será desenvolvidos um agente de montitoramento.

3 PROPOSTA

Este capítulo apresenta os materiais e o método de pesquisa.

3.1 Materiais

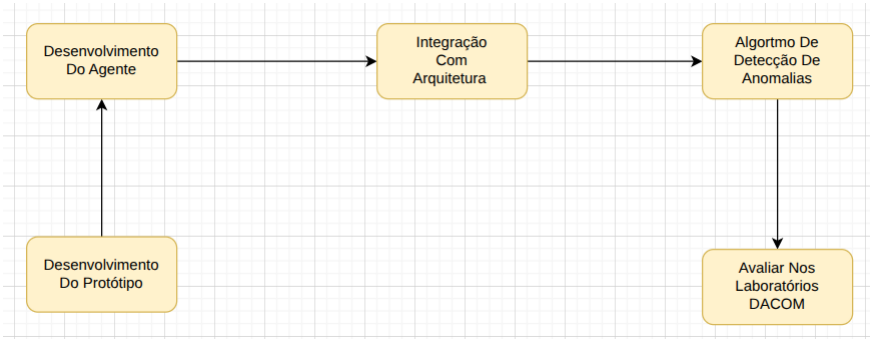
Para o desenvolvimento deste trabalho e da implementação da proposta de pesquisa, serão usados os seguintes componentes e ferramentas:

- Arquitetura Proposta (**??**): uma arquitetura de Big Data específica, fundamental para o processamento eficiente de grandes volumes de dados.
- Linguagem de Programação C: escolhida para o desenvolvimento do agente de monitoramento devido à sua leveza e eficiência, o que garante o consumo mínimo de recursos do sistema.
- Linguagem de Programação Python: será empregada no desenvolvimento do algoritmo de detecção.
- Biblioteca Scikit-learn (sklearn): uma biblioteca de *machine learning* (ML), usado para o desenvolvimento rápido e robusto do algoritmo de detecção.
- Laboratório do DACOM: para o desenvolvimento deste trabalho será necessário usar a infraestrutura dos laboratório do DACOM. O agente será executando na máquinas locais, eviando métricas para a arquitetura implementada no trabalho de **??**)

3.2 Método de pesquisa

Esta seção detalha a concepção do agente de monitoramento e a sua função como um módulo de expansão da arquitetura de Big Data proposta por **??**). Como pode ser observado na Figura **??** a seguir:

Figura 5 – Fluxo de desenvolvimento do método de pesquisa



Fonte: Autoria própria (2025).

A metodologia apresentada divide-se em cinco etapas sequenciais. Primeiramente, será realizado o desenvolvimento do agente de monitoramento em linguagem C. No passo seguinte, será realizada a integração desse agente com a arquitetura proposta (**??**). Posteriormente, será implementado o algoritmo de detecção de anomalias. Após a conclusão destas três etapas, serão iniciados os testes no ambiente do laboratório DACOM. Por fim, com base nos resultados dos testes, será analisada a viabilidade da solução proposta.

3.2.1 Desenvolvimento do Agente

O Agente será desenvolvido utilizando a linguagem C, conhecida por seu desempenho e eficiência no gerenciamento de recursos. A implementação explorará bibliotecas que garantem alto desempenho, como a *pthreads*, que possibilitará a programação paralela para otimizar a execução das tarefas.

Para conferir flexibilidade e portabilidade ao agente de monitoramento, será criado um arquivo de configuração. Este arquivo conterá variáveis de ambiente essenciais para o ajuste fino do sistema, incluindo:

- Tempo de coleta de métricas;
- Tamanho da memória *buffer*;
- Quantidade de registros enviados por vez.
- paralização de tarefas com *threads* independentes

O principal objetivo do projeto é otimizar ao máximo os recursos computacionais e, simultaneamente, evitar o congestionamento da rede causado pelo tráfego de dados. Para isso, o agente implementará uma lógica que enviará ao servidor somente os dados que mudaram de estado durante a janela de coleta. Essa estratégia visa garantir uma significativa otimização tanto no uso de recursos do agente quanto na largura de banda da rede.

Algumas funcionalidades adicionais serão implementadas, incluindo a coleta da versão do kernel e da distribuição de cada máquina. Esse procedimento tem como objetivo a verificação de possíveis vulnerabilidades inerentes à respectiva distribuição Linux. Além disto, essa

funcionalidade será executada em grandes intervalos, com baixa frequência e amplos intervalo de tempo.

Caso haja disponibilidade de tempo, será implementado um algoritmo para a verificação da versão de todos os aplicativos instalados na máquina, visando à identificação de vulnerabilidades introduzidas por software de terceiros.

3.2.2 Métricas Coletadas pelo Agente de Monitoramento

As métricas coletadas pelos agente serão aquelas apresentadas **??**). Essas métricas possibilitam analisar o comportamento dos processos e identificar padrões anômalos no consumo de recursos. A seguir, são descritas detalhadamente:

- *status_sleeping*: indica que o processo está inativo, aguardando algum evento ou recurso.
- *mem_rss*: quantidade de memória física residente utilizada pelo processo.
- *mem_uss*: quantidade de memória exclusiva do processo (única e não compartilhada).
- *ionice_value*: nível de prioridade de operações de I/O atribuída ao processo.
- *mem_shared*: quantidade de memória compartilhada entre processos.
- *mem_vms*: tamanho total da memória virtual alocada.
- *mem_text*: memória utilizada pelo código executável (segmento de texto).
- *nice*: valor de prioridade de CPU definido pelo usuário.
- *kb_received*: quantidade de dados de rede recebidos (em KB).
- *kb_sent*: quantidade de dados de rede enviados (em KB).
- *num_fds*: número total de descritores de arquivos abertos pelo processo.
- *mem_data*: memória utilizada pelos dados e variáveis do processo.
- *gid_effective*: id de grupo efetivo associado ao processo.
- *cpu_num*: número da CPU onde o processo está sendo executado.
- *num_threads*: quantidade de threads pertencentes ao processo.
- *cpu_percent*: percentual de utilização da CPU pelo processo.
- *io_read_chars*: total de caracteres lidos em operações de I/O.
- *ctx_switches_involuntary*: mudanças de contexto forçadas pelo sistema operacional.

- *ionice_ioclass*: classe de prioridade de I/O atribuída.
- *cpu_user*: tempo de CPU gasto em modo usuário.
- *cpu_sys*: tempo de CPU gasto em modo kernel.
- *io_read_count*: número total de operações de leitura realizadas.
- *ctx_switches_voluntary*: mudanças de contexto feitas voluntariamente pelo processo.
- *cpu_children_sys*: tempo de CPU em modo kernel consumido por processos filhos.
- *io_write_count*: número total de operações de escrita.
- *cpu_children_user*: tempo de CPU em modo usuário consumido por processos filhos.
- *io_read_bytes*: quantidade de bytes lidos em operações de I/O.
- *io_write_bytes*: quantidade de bytes escritos em operações de I/O.
- *io_write_chars*: total de caracteres escritos em operações de I/O.
- *status_running*: indica que o processo está atualmente em execução.
- *status_disk-sleep*: processo bloqueado aguardando operações de disco.
- *mem_dirty*: quantidade de memória marcada como suja (não escrita em disco).
- *mem_swap*: quantidade de uso de memória *swap* pelo processo.
- *mem_lib*: memória utilizada por bibliotecas carregadas dinamicamente.

3.2.3 Integração do Agente de Monitoramento ao Pipeline da Arquitetura

Na Figura **??** apresenta a arquitetura proposta, na qual o agente de monitoramento, desenvolvido em C, atua como o elemento inicial do fluxo de dados. Ele é responsável por enviar as métricas coletadas para o restante da arquitetura de processamento e análise.

O agente é executado diretamente nos equipamentos monitorados, coletando métricas de CPU, memória, rede e outros indicadores relevantes. Após a coleta, o agente envia as informações os dados diretamente para o *broker* Apache Kafka, dependendo da configuração do ambiente. A integração é realizada utilizando a biblioteca *librdkafka*, que fornece um cliente de alto desempenho para publicação de mensagens em C.

O pipeline completo ocorre da seguinte forma:

- Passo 1 - Coleta: O agente desenvolvido em C coleta as métricas em tempo real, organiza-as em um binário e aplica marcações de tempo para posterior análise.

- Passo 2 - Envio para o Apache Kafka (via librdkafka): Utilizando a *librdkafka*, o agente envia os eventos para o tópico configurado no cluster Kafka. São aplicadas configurações para garantir eficiência e tolerância a falhas.
- Passo 3 - Processamento no Pipeline: O módulo de processamento consome os eventos do Kafka. Nesta etapa, são aplicados enriquecimentos filtragem e agregações por janelas temporais, permitindo reduzir ruído e preparar os dados para o modelo de detecção de anomalias.
- Passo 4 - Armazenamento Estruturado: As métricas tratadas são enviadas ao módulo de armazenamento, onde podem ser persistidas em um banco orientado a séries temporais ou indexadas. Essa etapa garante histórico, consultas rápidas e suporte aos módulos seguintes.
- Passo 5 - Visualização e Análise: Por fim, os dados são disponibilizados no módulo de visualização, permitindo a construção de dashboards, gráficos de tendência e identificação de anomalias.

Posteriormente, o sistema será capaz de visualizar possíveis anomalias no funcionamento dos sistemas computacionais presentes nos Laboratórios do DACOM. Isso facilitará o gerenciamento de vulnerabilidades em todos os laboratórios, fornecendo dados segregados e relevantes para cada ambiente.

3.2.4 Algortimos de detecção de anomalias

Para a detecção de anomalias, será implementado um algoritmo focado na identificação de processos não comuns ao sistema fora do horário de funcionamento padrão dos computadores. O objetivo principal dessa abordagem é precisamente detectar e sinalizar qualquer processo em execução em períodos incomuns ou não previstos.

Caso haja disponibilidade de tempo, será considerada a implementação de um algoritmo inteligente para a detecção abrangente de anomalias. Para tal, serão explorados modelos combinação do PCA com K-means como apresentado em no trabalho **??**).O objetivo é identificar diversas anomalias, incluindo: falhas de hardware, presença de malware em máquinas ou a execução de processos não autorizados.

Para a validação, serão realizados testes no Laboratório do DACOM. O processo de clusterização por meio do K-means será executado de forma isolada no ambiente do laboratório, o que significa que cada cluster será analisado localmente com base nos processos presentes naquele ambiente.

Para alcançar um alto desempenho na classificação, será utilizada uma implementação do algoritmo K-means integrada ao framework Apache Spark. Essa abordagem permitirá uma significativa otimização na velocidade e na eficiência do processo de detecção.

3.2.5 Teste Nos Laboratórios DACOM

Para a validação da proposta de pesquisa, serão realizados experimentos utilizando os computadores do laboratório DACOM. O principal objetivo é verificar a escalabilidade do agente de monitoramento e reconhecer anomalias nos equipamentos do laboratório, verificando a latência entre a resposta e o envio do agente.

Algumas anomalias serão geradas artificialmente nos computadores e, posteriormente, será verificado se o algoritmo de detecção foi capaz de identificá-las, um exemplo disto é o funcionamento de um computador do laboratório em horário indevido, como processos rodandos fora do horário de funcionamento.

3.3 Cronograma de Atividades

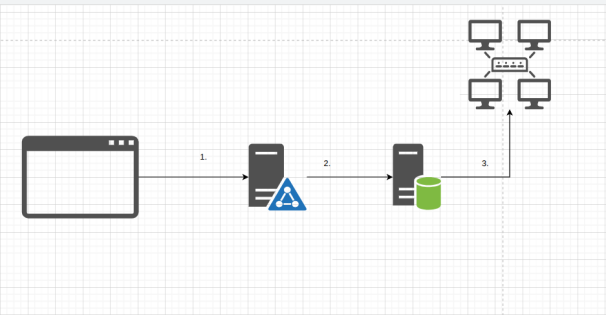
Tabela 1 – Cronograma do TCC (Jan–Jun)

Mês	Implementação do Agente	Integração com a Arquitetura	Algortimo de detecção de anomalias	Testes nos Laboratórios do DACOM	Escrita da Monografia
Jan	X				
Fev	X	X			
Mar		X	X		
Abr			X		
Mai				X	X
Jun			X	X	X

4 RESULTADO PRELIMINARES

Como resultado preliminar, foi desenvolvido um protótipo da arquitetura, utilizando um agente em Python que é capaz de coletar e visualizar métricas de processos no Kibana. A seguir, a Figura **??** ilustra o fluxo de dados deste protótipo.

Figura 6 – Arquitetura do protótipo



Fonte: Autoria própria (2025).

O fluxo de dados é dividido no seguinte caminho:

1. Agente Local → Logstash: o Agente Local envia as métricas de processo para o Logstash em um intervalo de tempo predefinido.
2. Logstash → Elasticsearch:o Logstash processa os dados e os envia ao Elasticsearch, que é responsável pelo armazenamento e indexação das métricas.
3. Visualização: o Kibana acessa o Elasticsearch e exibe as métricas de forma visual, permitindo a análise dos dados.

4.1 Métricas Recolhidas pelo Prótipo

Para o protótipo, as métricas recolhidas foram definidas conforme a lista abaixo. A fim de facilitar a coleta, foi utilizada a biblioteca psutil . As métricas coletadas são as seguintes:

- PID (*Process Identifier*): O número de identificação único do processo.
- Nome: o nome executável do processo.
- Usuário (*username*): O nome do usuário proprietário do processo.
- Timestamp: o registro de tempo da coleta da métrica.
- Status: o estado atual do processo (*e.g.*, running, sleeping).
- Tempo de Criação (*create_time*): o registro de tempo de inicialização do processo.
- Número de Threads (*num_threads*): a contagem de *threads* associadas ao processo.

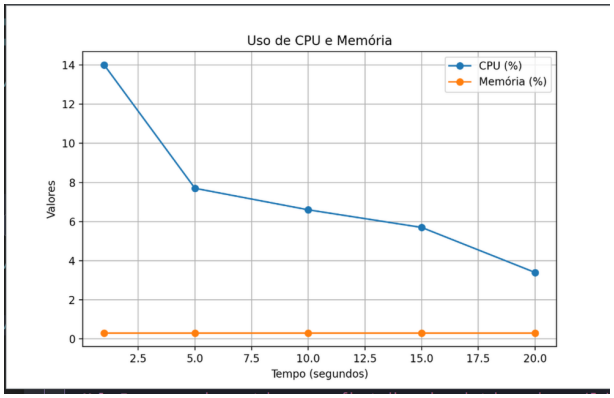
- Percentual de CPU (*cpu_percent*): o uso percentual de CPU pelo processo.
- Número de Processos Filhos (*num_child_processes*): a contagem de processos derivados.
- Percentual de Memória (*memory_percent*): O uso percentual de memória física pelo processo.
- Bytes de Leitura (*read_bytes*): o número de bytes lidos do disco (*I/O*).
- Bytes de Escrita (*write_bytes*): o número de bytes escritos no disco (*I/O*).
- Descritores de Arquivos (*num_fds*): o número de descritores de arquivos abertos (se disponível no sistema operacional).
- Prioridade (*nice*): o valor de prioridade *nice* do processo.

4.2 Desempenho do protótipo

O desempenho do protótipo revela que o consumo de recursos computacionais foi elevado para um agente local, atingindo aproximadamente 14% de uso de CPU com uma janela de 1 segundos de coleta de métricas. Essa ineficiência se deve, em grande parte, à linguagem de programação utilizada, visto que o Python é uma linguagem interpretada, o que introduz uma sobrecarga inerente ao processo.

O gráfico a seguir ilustra a porcentagem de CPU utilizada em função da janela de tempo definida para a coleta periódica de métricas.

Figura 7 – Gráfico de cpu vs. intervalo fixo entre coletas (segundos)



Fonte: Autoria própria (2025).

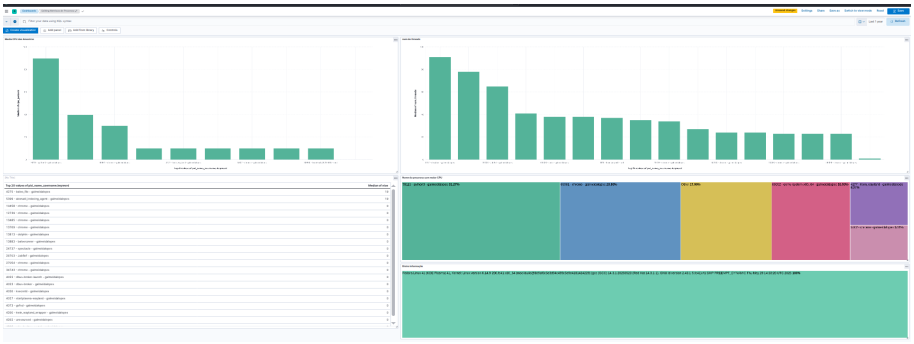
Acredita-se que, com a adoção de uma linguagem de programação mais eficiente em questão de gerenciamento de recursos, será possível otimizar o consumo de CPU. Outro ponto importante é que, ao realizar a visualização, o próprio agente apareceu consistentemente como um dos processos de maior consumo de CPU, demonstrando, assim, sua ineficiência e o alto custo de recursos inerente à implementação.

Outro ponto importante a ser analisado é que a implementação de políticas de caching garante uma redução no processamento gasto, especialmente quando adotada uma janela de coleta de métricas entre 5 a 10 segundos.

4.2.1 Visualização

Conforme ilustrado na Figura **??**, foi possível coletar com sucesso diversas métricas, incluindo dados de CPU, número de threads, versão do kernel e a identificação dos processos que consumiram mais memória.

Figura 8 – Gráficos do kibana



Fonte: Autoria própria (2025).

Mostrando assim a viabilidade da proposta deste trabalho, no qual os mesmo já conseguiu obter métricas de monitoramente, em uma arquitetura semelhante a prosta (**??**), evidenciando assim que com poucas alterações é possível desenvolver o agente e integrar pipeline da mesma.

5 CONSIDERAÇÕES FINAIS

Acreditamos que, por meio deste agente local e do algoritmo de detecção de anomalias, seja possível obter dados de anomalias em recursos computacionais críticos, auxiliando na segurança desses recursos.no próximos passos, será desenvolvido o agente em C e integração com a arquitetura.

A principal dificuldade e limitação apresentada reside na necessidade de uma grande quantidade de máquinas para a execução do agente e para a definição da escalabilidade da solução. Isso representa um grande desafio para a fase de experimentação da arquitetura proposta.

A principal ameaça à viabilidade do projeto é o desenvolvimento de um agente considerado muito pesado computacionalmente. Tal peso prejudicaria o objetivo de um monitoramento minimamente intrusivo, levando a não viabilidade em ambiente real, não consolidando a proposta do trabalho.

REFERÊNCIAS

ENES, J.; EXPóSITO, R. R.; TOURIñO, J. Bdwatchdog: Real-time monitoring and profiling of big data applications and frameworks. **Future Generation Computer Systems**, Elsevier BV v. 87„ p. 420–437, out. 2018. ISSN 0167-739X.

ENTERPRISES, N. **Nagios XI – Architecture Overview**, . 2015. https://support.nagios.com/kb/article/nagios-xi-architecture-overview-35.html. Acesso em: 1 nov. 2025.

FOROUZAN, B. A.; MOSHARRAF, F. **Redes de Computadores: Uma Abordagem Top-Down**. 1. ed. Porto Alegre, RS: Bookman, 2013. Acesso em: 31 mar. 2024. ISBN 9788580551693. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788580551693.

GUPTA, P.; TRIPATHY, P. Unsupervised learning for real-time data anomaly detection: A comprehensive approach. **International Journal of Computer Science and Engineering**, Seventh Sense Research Group Journals v. 11, n. 10, p. 1–11, out. 2024. ISSN 2348-8387.

HAMMAD, Y.; AHMAD, A. A.-S.; ANDRAS, P. An empirical study on the performance overhead of code instrumentation in containerised microservices. **Journal of Systems and Software**, Elsevier BV v. 230„ p. 112573, dez. 2025. ISSN 0164-1212.

LIU, J. *et al.* Practical anomaly detection over multivariate monitoring metrics for online services. **International Symposium on Software Reliability Engineering (ISSRE'2023)**, arXiv„, ago. 2023.

MATOS, G. L. D. Proposta e implantaÇÃo de uma arquitetura deprocessamento massivo de dados para monitoramento e detecÇÃo de ameaÇas cibernÉticas, „,. 2025.

NATSOS, A. L. S. D. Transformer-based malware detection using process resource utilization metrics. **Results in EngineeringVolume 25, March 2025, 104250**, „, 2025.

NOGUEIRA, M. **Minicursos do XXIV Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais**. [*S.l.*]: SBC, 2024. ISBN 9788576696018.

NYCZAK, J. Zero-effort service monitoring. **MSc System and Network Engineering**, „, August 17, 2015.

PINHEIRO, M.; CARONE, C. **PCDF investiga ataque hacker ao sistema do Hospital da Criança**, . 2024. Metrópoles. Acesso em: 10 dez. 2025. Disponível em: https://www.metropoles.com/distrito-federal/na-mira/pcdf-investiga-ataque-hacker-ao-sistema-do-hospital-da-crianca.

PROMETHEUS, . 2025. Acessado em: 31 out. 2025. Disponível em: https://prometheus.io/docs/introduction/overview/.

SILVA, G. **Prometheus: monitorando a saúde da sua aplicação**, . 2019. https://medium.com/tech-grupozap/prometheus-monitorando-a-saÃžde-da-sua-aplicaÃğÃčo-bd9b3b63e7b1. Acessado em 11-12-2025.

TANENBAUM, A. S.; WOODHULL, A. S. **Sistemas Operacionais: Projeto e Implementação**. 4. ed. São Paulo: Pearson Prentice Hall, 2016. ISBN 9788576052375.

WEI, H.; YAN, H. Abnormal linux process detection based on cpu usage. *In*: 2023 2ND INTERNATIONAL CONFERENCE ON BIG DATA, INFORMATION AND COMPUTER NETWORK (BDICN). 2023. **Anais [...]** [*S.l.*]: IEEE, 2023. p. 288–291.

YANHAONA, M. N.; PRODHAN, M. A. T.; GRIMSHAW, A. S. An agent-based distributed monitoring framework (extended abstract). *In*: 2015 INTERNATIONAL CONFERENCE ON NETWORKING SYSTEMS AND SECURITY (NSYSS). 2015. **Anais [...]** [*S.l.*]: IEEE, 2015. p. 1–10.

ZABBIX, . 2025. Acessado em: 1 nov. 2025. Disponível em: https://www.zabbix.com/documentation/6.0/pt/manual/introduction/overview.

APÊNDICE A – Abreviaturas dos meses do ano

Abreviaturas dos meses do ano

Janeiro – jan.

Fevereiro – fev.

Março – mar.

Abril – abr.

Maio – maio (único não abreviado).

Junho – jun.

Julho – jul.

Agosto – ago.

Setembro – set.

Outubro – out.

Novembro – nov.

Dezembro – dez.