

Mitigating Attacks on Persistent Volume Claims: Leveraging Audit Logs for Security in Kubernetes

João Barbosa - 42528

João Amendoeira - 39860

Trabalho realizado sob a orientação de

Rui Alves

Tiago Pedrosa

Licenciatura em Engenharia Informática

2024-2025

Mitigating Attacks on Persistent Volume Claims: Leveraging Audit Logs for Security in Kubernetes

Relatório da UC de Projeto
Licenciatura em Engenharia Informática
Escola Superior de Tecnologia e Gestão

João Barbosa - 42528
João Amendoeira - 39860

2024-2025

A Escola Superior de Tecnologia e de Gestão não se responsabiliza pelas opiniões expressas neste relatório.

Declaro que o trabalho descrito neste relatório é da minha autoria e é da minha vontade que o mesmo seja submetido a avaliação.

João Barbosa - 42528

João Amendoeira - 39860

Dedicatória

Dedicamos este trabalho às nossas famílias, aos nossos amigos e a toda a gente que nele participou, pelo apoio, incentivo e compreensão que sempre nos deram, tornando possível a concretização desta etapa nas nossas vidas.

Agradecimentos

Agradecemos ao nosso orientador Rui Alves, por toda a cooperação, disponibilidade e persistência prestada, não esquecendo todo o tempo e conhecimento que partilhou conosco.

Resumo

Este trabalho aborda a mitigação de ataques a Persistent Volume Claims (PVC) em ambientes Kubernetes, com o foco na distribuição K3s. A motivação surge devido aos riscos associados ao armazenamento persistente, como acessos não autorizados e padrões de Input/Output (I/O) anómalos, que podem comprometer a integridade dos dados. O objetivo central foi investigar de que forma os *logs* e métricas podem ser correlacionados para detetar e responder a anomalias, reduzindo falsos positivos e preservando evidências para análises futuras.

A metodologia consistiu na construção de um *cluster* K3s e na integração de uma *stack* de observabilidade e segurança: Prometheus para métricas, Loki para *logs*, Falco para deteção em *runtime* e Falco Talon para ações automáticas. Adicionalmente, foi desenvolvido um microserviço (*am-to-talon-bridge*) que traduziu alertas do Alertmanager, garantindo a resolução correta de pods alvo e permitindo aplicar políticas graduadas: rotulagem na primeira ocorrência e terminação em reincidências.

Os resultados evidenciaram que a combinação de *audit logs*, regras personalizadas e *pipelines* de métricas/*runtime* permitem identificar e mitigar comportamentos potencialmente maliciosos, reduzindo o tempo médio de deteção e resposta. A solução mostrou ser reproduzível e observável ponta a ponta, cumprindo o objetivo de reforçar a segurança em *clusters* Kubernetes.

Conclui-se que a abordagem proposta não só aumenta a eficácia da proteção de PVC, como estabelece uma base sólida para evolução futura, nomeadamente em cenários de maior resiliência, quarentena de *workloads* e integração sistemática da via de auditoria.

Palavras-chave: Kubernetes, Persistent Volume Claims, Audit Logs, Segurança

Abstract

This work addresses the mitigation of attacks on persistent volumes in Kubernetes environments, with a focus on the K3s distribution. The motivation arises from the risks associated with persistent storage, such as unauthorized access and anomalous I/O patterns, which can compromise data integrity. The main objective was to investigate how logs and metrics can be correlated to detect and respond to anomalies, reducing false positives and preserving evidence for future analysis.

The methodology consisted of building a K3s cluster and integrating an observability and security stack: Prometheus for metrics, Loki for logs, Falco for runtime detection, and Falco Talon for automatic actions. Additionally, a microservice (am-to-talon-bridge) was developed that translated alerts from Alertmanager, ensuring the correct resolution of target pods and allowing the application of graduated policies: labeling on first occurrence and termination on recurrence.

The results showed that the combination of audit logs, custom rules, and metrics/-runtime pipelines allows for the identification and mitigation of potentially malicious behavior, reducing the average detection and response time. The solution proved to be reproducible and observable from end to end, fulfilling the objective of strengthening security in Kubernetes clusters.

It can be concluded that the proposed approach not only increases the effectiveness of PVC protection, but also establishes a solid foundation for future evolution, particularly in scenarios of greater resilience, workload quarantine, and systematic integration of the audit trail.

Keywords: Kubernetes, Persistent Volume Claims, Audit Logs, Security

Conteúdo

1	Introdução	1
1.1	Enquadramento	1
1.2	Objetivos	2
1.3	Estrutura do Documento	2
2	Fundamentos	5
2.1	Terminologia e convenções	5
2.2	Persistência (PV/PVC/SC/CSI)	6
2.3	Riscos em PVC	7
3	Análise do Problema	9
3.1	Formulação do problema	9
3.1.1	Questões orientadores para o contexto	9
3.2	Contexto e pressupostos	10
3.2.1	Contexto	10
3.2.2	Pressupostos	10
3.3	Perfis funcionais	10
3.4	Superfícies de ataque e eventos de interesse	11
3.4.1	Configuração de volumes (riscos estruturais)	11
3.4.2	Operações administrativas na API (sinais de alarme)	11
3.4.3	Comportamentos em runtime (dentro do pod)	11
3.4.4	Métricas e sintomas de I/O no PVC	11

3.4.5	Eventos de interesse para correlação (exemplos práticos)	12
4	Arquitetura da Solução	13
4.1	Visão geral da arquitetura	13
4.2	Ambiente de testes	14
4.2.1	Namespaces	15
4.2.2	Componentes implementados(via Helm/manifestos)	15
4.2.3	Princípios operacionais	15
4.2.4	Parâmetros críticos por componente	16
4.3	Fluxos end-to-end	17
4.3.1	Via de métricas	17
4.3.2	Via de <i>runtime</i>	18
4.3.3	Auditoria	18
4.4	Lógica de detecção e correlação	19
4.4.1	Regras	19
4.4.2	Consultas	20
4.4.2.1	Falco - logs “open-for-write”	20
4.4.2.2	Talon — confirmação de ações	20
4.5	Síntese da arquitetura	20
5	Discussão	23
6	Conclusões	27
A	Proposta Original do Projeto	A1
B	Instalações	B1

Lista de Tabelas

4.1	Recursos das Máquinas Virtuais	14
-----	--	----

Lista de Figuras

4.1	Diagrama da Arquitetura da Solução.	14
4.2	Logs do Falco.	20
4.3	Ações do Talon.	20
A.1	Proposta original do projeto.	A1
B.1	Nós do cluster com o status <i>Ready</i>	B2
B.2	Serviços do Kube-Prometheus-Stack.	B3
B.3	<i>Pods</i> do Falco.	B7
B.4	Regra montada nos <i>Pods</i>	B7
B.5	<i>Pods</i> do FalcoSidekick.	B8
B.6	<i>Logs</i> do FalcoSidekick.	B8
B.7	<i>Pods</i> do Talon.	B8
B.8	ConfigMap do Falco Talon.	B9
B.9	<i>Pods</i> do Loki.	B11
B.10	<i>Services</i> do Loki.	B11
B.11	Logs do Loki.	B11
B.12	Loki a receber e a enviar para o SideKick.	B12
B.13	Loki a funcionar no Grafana.	B13
B.14	Regra do Loki-Ruler no ConfigMap.	B14
B.15	Logs do Loki-Ruler.	B14
B.16	Loki Rules no Grafana.	B17
B.17	Pod do AM-TO-TALON-BRIDGE.	B29

Siglas

API Application Programming Interface. 1, 5, 7, 9, 10

CNI Container Network Interface. 14

CSI Container Storage Interface. 5, 6, 10

I/O Input/Output. ix, 1, 7, 9–11, 13, 17, 24, 28

PSA Pod Security Admission. 10

PSS Pod Security Standards. 10

PV Persistent Volume. 1, 5–7, 9–11, 18

PVC Persistent Volume Claims. ix, x, 1, 2, 5–7, 9–13, 17–19, 23, 24, 27, 28

RBAC Role-Based Access Control. 28

ROX ReadOnlyMany. 5, 6, 10

RWO ReadWriteOnce. 5–7, 10

RWX ReadWriteMany. 5–7, 10, 11

SC Storage Class. 5–7, 9–11, 14, 18

VM Virtual Machine. 14, 16

Capítulo 1

Introdução

1.1 Enquadramento

A computação em *cloud* e a adoção de *containers* transformaram a forma como as aplicações são desenvolvidas e operadas. Entre as várias plataformas de orquestração de *containers*, o Kubernetes destaca-se pela excelência do conjunto de ferramentas, integrações e pela sua portabilidade [1]. Em cenários de pequena escala, como por exemplo, *edge computing*, laboratórios ou ambientes com recursos limitados é frequente recorrer ao K3s, pois, é uma distribuição leve e certificada de Kubernetes, compatível com a API dos Kubernetes [2].

A persistência de dados em Kubernetes assenta sobre Persistent Volume (PV) e PVC [3]. Apesar das suas vantagens, existem riscos, tais como, acessos não autorizados, padrões anómalos de I/O e operações administrativas fora do esperado que podem comprometer a disponibilidade e integridade do sistema. Por isso, a deteção atempada e a resposta rápida são extremamente necessárias, também em clusters leves como o K3s.

Este trabalho investiga de que modo os *logs* e métricas podem ajudar na deteção e mitigação de anomalias associadas a PVC, combinando registos de auditoria da API com telemetria em *runtime* e monitorização integrada [4], [5]. Pretende-se reduzir falsos positivos por correlação entre registos de auditoria distinguindo operações legítimas de padrões

anômalos, acelerar o tempo de reação e preservar a evidência para análises posteriores.

1.2 Objetivos

O objetivo deste trabalho é investigar como os *logs* podem ser utilizados para detectar atividades suspeitas e mitigar os riscos associados aos PVC, de forma a garantir a proteção e a integridade dos dados armazenados.

Na proposta original (ver apêndice A), o projeto foca-se na análise de *audit logs* como a principal fonte de detecção e na integração de ferramentas capazes de recolher, armazenar e associar métricas e eventos, o que reforça a capacidade de monitorização e resposta a incidentes.

Objetivos:

- Delimitar o problema, ameaças relevantes aos PVC, atores envolvidos e restrições do ambiente K3s.
- Explorar ferramentas e tecnologias adequadas ao domínio do problema.
- Desenvolver e integrar soluções de monitorização e detecção de anomalias relacionadas com os PVC.
- Definir e implementar várias soluções/*pipelines* para correlação e alertas.
- Modelar a resposta graduada: primeira ocorrência, rotular; reincidência dentro da janela, conter/terminar.
- Validar em K3s e consolidar uma arquitetura de referência reproduzível, com recomendações práticas e identificação de limitações.
- Documentar o processo e os resultados obtidos.

1.3 Estrutura do Documento

Este relatório está organizado da seguinte forma:

1. **Capítulo 1** — Introdução ao projeto.

2. **Capítulo 2** — Fundamentos (Background).
3. **Capítulo 3** — Análise do problema.
4. **Capítulo 4** — Arquitetura da Solução.
5. **Capítulo 5** — Discussão dos resultados.
6. **Capítulo 6** — Conclusões.
7. **Apêndices** — Proposta inicial do projeto e outros elementos complementares relevantes.

Capítulo 2

Fundamentos

Este capítulo estabelece a base conceptual necessária para o restante trabalho. São definidas as principais terminologias e convenções, revistos os conceitos de armazenamento em Kubernetes (PV, PVC, Storage Class (SC), Container Storage Interface (CSI)) e enquadrados os mecanismos de observabilidade (*logs* e métricas), auditoria da API e segurança em *runtime*. O foco deste capítulo é clarificar o vocabulário e os mecanismos usados estabelecendo a base conceptual para os capítulos seguintes.

2.1 Terminologia e convenções

- ***Cluster* / nó / namespace / pod / *container*.** '*Cluster*' é o conjunto de nós (máquinas). 'Namespace' segmenta recursos lógicos. 'Pod' é a unidade mínima de execução que pode conter um ou mais '*containers*' [6].
- **PV / PVC / SC / CSI.** PV é a unidade de armazenamento que é disponibilizada ao *cluster*. PVC é o pedido da aplicação por armazenamento. SC define como os volumes são provisionados. CSI é o padrão que permite drivers de armazenamento. **Modos de acesso:** ReadWriteOnce (RWO), ReadWriteMany (RWX), ReadOnlyMany (ROX) [3], [7], [8].
- ***Audit logs*.** Registos emitidos pelo kube-apiserver com o “quem/quê/quando/onde”

das operações [9].

- **Resposta graduada.** Política em dois passos: a primeira ocorrência aplica a *label* informativa, numa reincidência dentro da janela faz uma ação de contenção, por exemplo, terminar o pod.
- **Segurança em *runtime* (Falco).** O Falco vigia eventos em tempo de execução(*runtime*) através do kernel/*syscalls* e a auditoria do Kubernetes e aplica regras. Quando uma regra dispara, o Falco emite um evento que pode ser enviado para Loki/Alertmanager/Sidekick para depois o Talon fazer as ações [10]–[12].
- **Observabilidade e *pipeline* de alertas (Loki / Prometheus / Grafana / Alertmanager / Sidekick / Talon).** Loki armazena e permite consultar *logs*; o Loki Ruler cria alertas a partir de consultas. O Prometheus recolhe métricas e o Grafana serve para visualização. O Alertmanager recebe alertas e encaminha-os por HTTP/webhook. O FalcoSidekick recebe eventos do Falco e entrega-os a Loki e/ou ao Alertmanager. O Falco Talon aplica ações no Kubernetes por resposta graduada com base em regras. Fluxo típico: Falco → Sidekick → Loki(Loki Ruler) → Alertmanager → Talon → Ação. [13]–[15]

2.2 Persistência (PV/PVC/SC/CSI)

Em Kubernetes, a persistência separa o ciclo de vida do armazenamento dos pods. Um PV é capacidade disponível de armazenamento dentro do *cluster* e um PVC é o pedido de armazenamento da aplicação, a associação pode ser estática ou dinâmica através da SC. Os modos RWO/ROX/RWX condicionam a partilha e isolamento o RWO isola melhor, RWX facilita partilha e eleva o risco [3], [8].

A CSI padroniza os drivers de armazenamento. A SC escolhe o *driver* e parâmetros, por exemplo, a política de *reclaim Retain*(reter os dados do PV)/*Delete*(eliminar os dados do PV) e binding *WaitForFirstConsumer*(adia o provisionamento até existir um

pod que consome o PVC), pode suportar expansão e *snapshots* ponto-no-tempo, quando disponíveis [7], [8].

Implicações: RWX e montagens partilhadas podem aumentar a superfície de ataque, operações sobre PV/PVC/SC devem ser auditadas, *snapshots* e *reclaim* ajudam na resiliência e na investigação pós-incidente.

2.3 Riscos em PVC

Os principais riscos em Kubernetes são montagens *hostPath*, uso inseguro de *subPath* (CVE-2021-25741)[16] e RWX combinado com privilégios excessivos, além disso, operações administrativas atípicas, tais como, criar/alterar/apagar um PV/PVC/SC são sinal de alarme. Nesses casos, deve-se correlacionar os eventos de API com sinais de *runtime* e métricas de I/O (por exemplo, contagens de “open-for-write” e picos de utilização do PVC), janelas temporais e *allowlists* para jobs legítimos (por exemplo, *backups*) ajudam a reduzir falsos positivos. Mitigar com menor privilégio, evitar *hostPath*(diretórios do *host* dentro do contentor), validar *subPath*, preferir RWO quando possível e garantir *audit logs* para o caminho “quem/quê/quando/onde” [9], [16].

Com estes fundamentos, no capítulo seguinte procede-se à Análise do Problema, caracterizando o contexto K3s/PVC, as ameaças e pressupostos, e fixando requisitos e métricas que orientarão as abordagens/soluções a considerar.

Capítulo 3

Análise do Problema

Neste capítulo será analisado o problema de proteger os PVC em ambientes K3s com recursos limitados. Delimitamos o escopo e os pressupostos, identificamos atores e superfícies de ataque, e avaliamos as fontes de dados (auditoria, *runtime*, métricas) e as suas limitações. A partir daí, derivamos requisitos e critérios de avaliação.

3.1 Formulação do problema

Em *clusters* K3s, os PVC suportam dados críticos mas estão sujeitos a configurações arriscadas, como por exemplo de `hostPath` [17] e/ou de `subPath` [16], a padrões anómalos de I/O e a operações administrativas inesperadas como criar, alterar, apagar, expandir PVC/PV/SC. O desafio é detetar cedo, correlacionar sinais de auditoria, *runtime* e métricas para reduzir falsos positivos e atuar com baixo impacto num *cluster* leve [10], [15], [18].

3.1.1 Questões orientadores para o contexto

1. Que eventos da API sobre PVC/PV/SC devem compor o trilha de “quem/quê/-quando/onde”?
2. Que sinais de *runtime* e métricas de I/O ajudam a distinguir ruído de anomalia?

3. Que políticas de resposta graduada (notificar \rightarrow *labelling* \rightarrow ação) equilibram risco e disponibilidade?

3.2 Contexto e pressupostos

3.2.1 Contexto

Ambiente de K3s (cluster leve com recursos limitados), compatível com a API Kubernetes. Persistência através de PVC/PV com SC/CSI; coexistem modos RWO/ROX/RWX. Existem *audit logs* do kube-apiserver com sinais de *runtime* e métricas de PVC disponíveis para correlação.

3.2.2 Pressupostos

- Rotulagem consistente por *namespace*/pod/*container*/PVC.
- Pod Security Admission (PSA)/Pod Security Standards (PSS) aplicados por *namespace* (princípio do menor privilégio), hostPath evitado e subPath apenas quando justificado.
- Existem cargas legítimas com picos de I/O (backups), tratadas com janelas temporais e *allowlists*
- Configuração declarativa e reproduzível através de *manifests*/Helm.

3.3 Perfis funcionais

- DevOps — define SC/quotas, ativa auditoria e observabilidade, gere exceções.
- SecOps — define políticas de detecção/resposta, triagem e forense.
- Dev — ajusta *workloads* face a *labels*/avisos e planeia janelas para *jobs* ruidosos.

3.4 Superfícies de ataque e eventos de interesse

Esta secção mapeia as principais superfícies de ataque ligadas ao armazenamento persistente e os eventos a monitorizar. Organizando os sinais por configuração, API, runtime e métricas de I/O.

3.4.1 Configuração de volumes (riscos estruturais)

- `hostPath` expõe diretórios do host ao contentor.
- `subPath` mal validado permite *path* traversal e fuga do volume, expondo ficheiros do host.
- RWX mais permissões excessivas ampliam a superfície de ataque.

3.4.2 Operações administrativas na API (sinais de alarme)

- **PVC/PV com SC:** *create/patch/delete/expand*, alterações de *reclaimPolicy*, *accessModes*, *allowVolumeExpansion*.
- **Bindings de segurança:** *Role/ClusterRole/Binding* que concedem *update/delete* sobre PVC/PV com SC.

3.4.3 Comportamentos em runtime (dentro do pod)

- Padrões de I/O elevados e/ou fora do normal.
- Acesso a ficheiros sensíveis dentro do pod ('open-for-write')

3.4.4 Métricas e sintomas de I/O no PVC

- Ocupação persistente acima de um limite
- Taxa de I/O alta fora do esperado e/ou *spikes* curtos repetidos.

3.4.5 Eventos de interesse para correlação (exemplos práticos)

- **Runtime:** $\geq n$ eventos de 'open-for-write' dentro de um período de tempo (exemplo, 15 minutos)
- **Métricas:** pico de utilização do PVC coincidente com os eventos de *runtime*.

Com esta análise, o Capítulo 4 apresentará a Arquitetura de Solução, materializando os princípios em *pipelines* alternativos de detecção/correlação/ação. Serão detalhados componentes, fluxos e modelos de dados e justificadas as escolhas face a requisitos e métricas. Fecha com a resposta graduada e a preparação dos cenários de validação.

Capítulo 4

Arquitetura da Solução

Aqui é apresentada a Arquitetura da Solução definida a partir da análise do problema. A arquitetura procura detetar e mitigar comportamentos anómalos de I/O em PVC num *cluster* K3s, privilegiando a simplicidade, separação de responsabilidades, reprodutibilidade através do Helm e correlação de fontes (*audit logs*, sinais de *runtime* e métricas).

4.1 Visão geral da arquitetura

A arquitetura combina duas vias principais orientadas aos PVC num *cluster* K3s, métricas e *runtime* e, em complemento, auditoria: eventos de *Audit Log* encaminhados para o Loki e correlacionados pelo Loki Ruler, opcionalmente, podem gerar alertas para o Alertmanager.

Na via de métricas, o Prometheus gera alertas que o Alertmanager encaminha para o 'am-to-talon-bridge', que resolve o pod alvo (por PVC) e remete para o Falco Talon. O Talon executa resposta graduada: à primeira ocorrência aplica uma *label* informativa, na reincidência dentro da janela temporal definida, termina o pod. Em paralelo, na via de *runtime*, o Falco envia eventos via FalcoSidekick para o Loki (usado para análise e correlação, o Loki Ruler pode agregar janelas/contagens, mas não está no caminho de mitigação). O Grafana centraliza a visualização (*dashboards/Explore*) para futura análise.

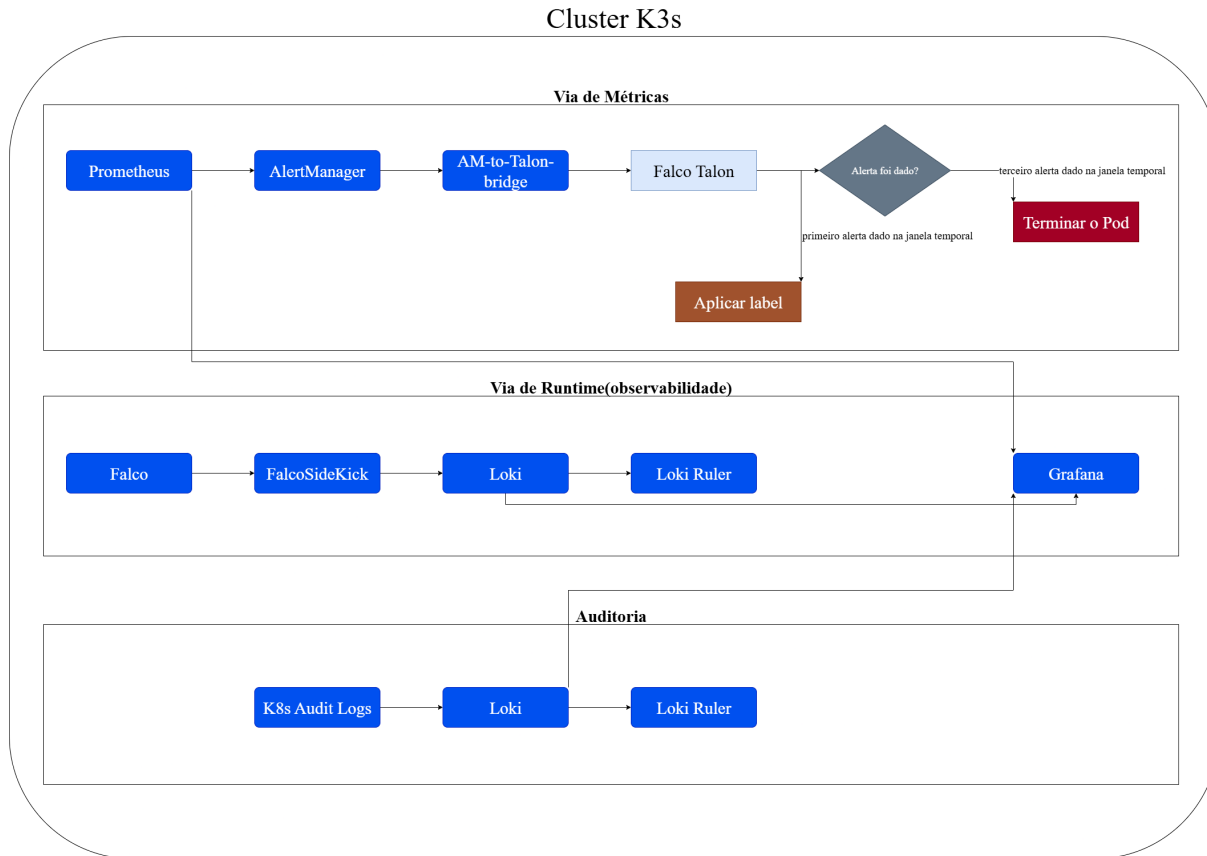


Figura 4.1: Diagrama da Arquitetura da Solução.

4.2 Ambiente de testes

Topologia e recursos. Três Virtual Machine (VM) numa rede privada (IP's internos), com K3s:

VM	SO	vCPU	RAM	Disco	Papel
projectServer	Debian(Ubuntu)	4	4GiB	30GiB	server/control-plane
projectAgent	Debian(Ubuntu)	2	2GiB	30GiB	worker
projectMetric D	Debian(Ubuntu)	2	2GiB	30GiB	<i>stack</i> de observabilidade

Tabela 4.1: Recursos das Máquinas Virtuais

Sistema base Debian. Container Network Interface (CNI) flannel(ligação dos *pods* à rede) com SC por omissão em *local-path*.

4.2.1 Namespaces

- ***monitoring*** (Prometheus/Alertmanager/Grafana)
- ***loki*** (Loki e LokiRuler)
- ***falco*** (Falco/Sidekick/Talon)
- ***observability*** (am-to-talon-bridge)
- ***default*** usado para cargas de teste.

4.2.2 Componentes implementados(via Helm/manifestos)

- kube-prometheus-stack (Prometheus, Alertmanager, Grafana).
- loki (modo single-binary) mais regras no Loki Ruler.
- falco, falcosidekick e falco-talon.
- am-to-talon-bridge(webhook “tradutor” entre Alertmanager e Talon)

4.2.3 Princípios operacionais

- Reprodutibilidade através de values.yaml.
- Exposição de serviços apenas para testes via NodePort(acesso aos serviços através do Windows).
- RBAC mínimos(princípio do menor privilégio), política faseada no Talon (*label* na primeira ocorrência, *terminate* em reincidência).
- *Logs* e métricas centralizados (Loki/Prometheus) para correlação e auditoria.

4.2.4 Parâmetros críticos por componente

- **Falco** - driver: ebpf

Foi usado eBPF para evitar dependências do kernel nas VM e garantir um arranque estável

```
1 driver:
2   enabled: true
3   kind: ebpf
```

- **Falco** - json_output: true

Saída estruturada para integração limpa com Sidekick/Loki, correlação confiável e *queries* no Grafana.

```
1 falco:
2   json_output: true
```

- **FalcoSidekick para Loki**

Canal único de eventos de runtime para armazenamento/consulta.

```
1 loki: { hostport: "http://loki.loki.svc.cluster.local:3100" }
```

- **Alertmanager para am-to-talon-bridge**

Definido apenas alertas relevantes para o “tradutor” que resolve o pod alvo, reduzindo falhas de “missing pod”.

```
1 receivers:
2   - name: talon-webhook
```

```
3   webhook_configs: [{ url: "http://am-to-talon-  
4   bridge.observability.svc:8080/alert" }]
```

- **Falco Talon**

Política não disruptiva primeiro (*label*), contenção na reincidência, implica menos falsos positivos com ação proporcional.

```
1   watchRules: true  
2   config:  
3     rulesOverride: |  
4       - action: Label Suspicious  
5       - action: Terminate Pod
```

Nota: Passos completos de instalação (K3s/Helm), values.yaml e *prints* de validação encontram-se no **Apêndice B**.

4.3 Fluxos end-to-end

Esta secção descreve, ao nível lógico, como um evento percorre a arquitetura desde a deteção até à ação/observabilidade.

4.3.1 Via de métricas

Passos

1. Prometheus avalia regras de alerta sobre indicadores ligados a PVC (por exemplo, utilização anómala, picos de I/O).
2. Quando uma regra entra em *firing*, o Alertmanager agrupa o alerta e envia um *webhook* para o 'am-to-talon-bridge'.

3. O *bridge* normaliza o alerta e resolve o alvo: Quando o alerta chega apenas com o PVC, o *bridge* enumera os pods do *namespace* e escolhe o que monta esse *claim*, priorizando os que estão em *Running* e o mais recentes. Se o pod referenciado já não existir, aponta para o seu sucessor.
4. O *bridge* envia o evento ao Falco Talon.
5. O Talon aplica a resposta graduada.
6. O Kubernetes recicla o pod (via ReplicaSet/Deployment).

4.3.2 Via de *runtime*

Passos

- O Falco observa *syscalls*/eventos K8s e emite eventos (por exemplo, a regra “open-for-write”).
- O FalcoSidekick entrega esses eventos ao Loki..
- O Loki armazena e permite correlação temporal e o Loki Ruler pode calcular contagens/janelas para alerting de observabilidade.
- O Grafana apresenta o *Explore* e as *dashboards* para futuras análises.

Nota: Esta via não desencadeia a contenção automática, serve apenas para contexto, validação e redução de falsos positivos por correlação.

4.3.3 Auditoria

Passos

1. O *kube-apiserver* emite *Audit Logs* (por exemplo de criação/alteração/eliminação de um PVC/PV/SC).
2. Estes registos são recebidos pelo Loki e podem ser consultados.

Nota: fornece rasto para investigações e pode reforçar a decisão (elevar severidade quando há coincidência entre *runtime* e a auditoria).

4.4 Lógica de detecção e correlação

Detetar duas classes de eventos ligados a PVC:

1. pressão de capacidade (uso do volume elevado/prolongado)
2. padrões de escrita anómalos (picos de open-for-write).

É comparado sempre por *namespace* e pod, quando existe, também por PVC.

4.4.1 Regras

1. **PVC_IO_Spike_1x (aviso):** primeira ocorrência da regra Falco Storage open-for-write.

```
1 - rule: PVC IO spike (label)
2     match: { source: prometheus, rules: [PVC_IO_Spike_1x] }
3     actions: [ { action: Label Suspicious } ]
```

2. **PVC_IO_Spike_3x (crítico):** na terceira ocorrência na janela temporal

```
1 - rule: PVC IO spike (kill)
2     match: { source: prometheus, rules: [PVC_IO_Spike_3x] }
3     actions: [ { action: Terminate Pod } ]
```


Loki, preservando o trilha “quem/quê/quando/onde” para análise forense. A instalação é reprodutível via Helm/*Manifests*, tem resposta em dois passos (*label* depois *terminate*), *bridge* para ajudar na ligação AlertManager com o Talon. Com isto obtém-se menor tempo médio de detecção e tempo médio para recuperar mais baixo, mantendo um rasto de auditoria.

Capítulo 5

Discussão

Este capítulo discute, ponta-a-ponta, a evolução do sistema desde a detecção em Falco até à contenção automática, evidenciando alternativas testadas, falhas diagnosticadas e a solução que mais se aproximou do objetivo final.

Partimos de uma via *runtime*: Falco \rightarrow FalcoSidekick \rightarrow Loki. O objetivo desta via foi sempre observabilidade e análise forense. Experimentámos colocar o Loki Ruler a agregar janelas/contagens por *namespace*/pod/PVC, isto funcionou para correlação, mas não é o caminho mais adequado para mitigação (o Ruler foi desenhado para alertar, não para acionar). Em paralelo, desenhámos a via de métricas com Prometheus/Alertmanager, a partir da qual foi possível orquestrar a resposta graduada no *cluster*.

A primeira abordagem de mitigação foi enviar alertas diretamente para o *webhook* do Falco Talon. Os *matches* apareciam, mas a ação falhava frequentemente com “missing pod”. Vimos ainda que o *chart* do Talon injetava uma ação padrão “Terminate Pod” que, por vezes, sombreava a nossa ação pretendida (parâmetros diferentes), criando inconsistências.

A segunda abordagem isolou o problema de endereçamento: criámos o ‘am-to-talon-bridge’, um *micro-webhook* entre o Alertmanager e o Talon. O *bridge* recebe alertas, escolhe os *labels* relevantes (*namespace*, pod, PVC) e resolve o alvo de forma robusta:

PVC para pod: lista pods do *namespace*, filtra os que montam o PVC e ordena *Running* $>$ *Pending* $>$ restantes (e, dentro de cada grupo, o mais recente). Antes de enviar ao

Talon, o *bridge* faz uma espera curta para garantir que o *pod* existe e não está no *status* de terminar. Esta estratégia estabilizou a execução. Nos *logs* do Talon é possível verificar que esta a terminar o pod correto. A terceira abordagem consolidou a orquestração. Definiram-se regras de Prometheus para PVC I/O (por exemplo, `PVC_IO_Spike_1x` e `PVC_IO_Spike_3x`) e configurou-se o Alertmanager a enviar apenas estas para o *bridge*. Para testes ponta-a-ponta usámos:

- `curl` direto ao Talon.
- `curl` ao *bridge*.
- `POST /api/v2/alerts` no Alertmanager (valida a rota real).
- uma PrometheusRule sintética com vector para simular reincidência.

Este último revelou um efeito: terminações a cada 15 minutos, foi diagnosticado que era o *repeat_interval* do Alertmanager a reenviar um alerta sempre *firing*. Em produção, tal não ocorre com regras reais (condição volta a *ok*), mas nos testes aplicámos *silences* ou ajustámos temporariamente o *repeat_interval*.

No Falco Talon, materializámos a resposta graduada:

- 'Label Suspicious' na primeira ocorrência (`PVC_IO_Spike_1x`);
- 'Terminate Pod' na reincidência (`PVC_IO_Spike_3x`), com 'grace_period_seconds: 0', ignorando *DaemonSets/StatefulSets*.

As tentativas iniciais falharam onde a realidade dinâmica do Kubernetes é mais sensível: nomes de pods efémeros e ausência de pods em alguns alertas. O que funcionou foi a separação clara de papéis: Prometheus/Alertmanager deteta e decide enviar, o *bridge* resolve o alvo e garante condições mínimas, o Talon executa a ação certa com regras explícitas. A via *runtime* (Falco→Loki) permanece essencial para observabilidade e perícia, mas a mitigação operacional vive na *pipeline* de métricas.

Em suma, a solução atual cumpre o objetivo: temos um ciclo deteção/contenção reproduzível, observável e testável 'end-to-end'. Como passos seguintes, propomos melhorar

a operação, ativar a via de auditoria no Loki Ruler para enriquecer correlação e, funcionalmente, evoluir a resposta para quarentena/anotação antes de *terminate* em *workloads* mais sensíveis.

Capítulo 6

Conclusões

Este projeto demonstrou, que é possível reduzir o tempo entre detecção e contenção de comportamentos anómalos relacionados com PVC em Kubernetes, combinando vários componentes de observabilidade e resposta. A arquitetura final integrou: (i) Falco para sinais de *runtime* e visibilidade forense (via FalcoSidekick→Loki), (ii) Prometheus/Alertmanager para alertas de métricas, (iii) um *bridge* (am-to-talon-bridge) que normaliza e resolve o alvo e (iv) Falco Talon para executar políticas de resposta graduada. O Grafana consolidou a análise e a auditoria. A prova de conceito alcançou uma cadeia fiável: de um *alertFiring* até ao *log* “status=success” do Talon e à substituição automática do pod, com *traceability* entre Alertmanager, *Bridge* e Talon.

Do ponto de vista crítico, as principais dificuldades foram motivadas por características do sistema Kubernetes, volatilidade de nomes de pods e a necessidade de uma resolução robusta, as condições entre a chegada do alerta e o ciclo de vida do pod, mitigadas com verificação de existência/eliminação e pequenas esperas e a derivação de configuração no Talon, em que ações *default* do *chart* podiam sobrepor as regras pretendidas. O *bridge* revelou-se o elemento diferenciador, desacoplou o formato dos alertas da lógica de ação e introduziu salvaguardas que aumentaram a precisão do alvo. Em contrapartida, introduz mais um serviço a operar e a manter, que deve ser endurecido. Outro ponto importante foi a separação clara de papéis, correlação/análise (Loki/Loki Ruler) não deve bloquear a mitigação, que deve permanecer simples e precisa.

Em termos de valor, o projeto entregou uma arquitetura replicável com artefactos declarativos (*manifests*/Helm), observabilidade alinhada com o ciclo de resposta (*logs* coerentes em Bridge e Talon, com *trace_id*), princípio do menor acoplamento, o Talon atua sempre sobre recursos Kubernetes, enquanto a escolha do alvo é resolvida a cargo do *Bridge*. As limitações residem sobretudo em cenários de carga elevada, bem como em *workloads stateful* que podem exigir respostas mais conservadoras (*quarentena/eviction* em vez de *terminate* imediato).

Como trabalho futuro, destacam-se quatro frentes:

- Endurecimento e segurança: imagem otimizada do Bridge, Alertmanager→Bridge→Talon, *NetworkPolicies* e Pod Security, Role-Based Access Control (RBAC) estrito, *rate-limiting* e *health metrics* e *dashboards* para o próprio Bridge/Talon.
- Alta disponibilidade e resiliência: réplicas, testes de caos, *backpressure* para suavizar picos de alertas e garantir reentrega.
- Políticas de resposta: modos de quarentena (*labels* + *Admission/NetworkPolicy*), *evictions* controladas e exceções por classe de *workload*.
- Via de auditoria: ingestão sistemática dos *Audit Logs* em Loki, correlações no Ruler e, quando apropriado, geração de alertas para o mesmo caminho de mitigação.

Em síntese, o projeto atingiu o objetivo central, detetar e conter de forma automática e auditável anomalias de I/O em PVC, com uma solução modular que respeita boas práticas de observabilidade e de operação em Kubernetes. Ficou claro que a eficácia não depende apenas da qualidade do sinal, mas sobretudo da capacidade de identificar o alvo correto no momento certo e de manter a configuração sob controlo. A base técnica construída é sólida para evolução, a maturação para produção passará por segurança, resiliência e enriquecimento das políticas, sem abdicar da simplicidade e da auditabilidade que tornaram a solução eficaz.


Bibliografia

- [1] K3s. «Documentação.» (), URL: <https://k3s.io/>.
- [2] Kubernetes. «Documentação.» (), URL: <https://kubernetes.io/docs/concepts/overview/#why-you-need-kubernetes-and-what-can-it-do>.
- [3] Kubernetes. «Persistent Volumes (PV/PVC).» (), URL: <https://kubernetes.io/docs/concepts/storage/persistent-volumes/>.
- [4] NSA/CISA. «Kubernetes Hardening Guidance v1.2 (2022).» (), URL: https://media.defense.gov/2022/Aug/29/2003066362/-1/-1/0/CTR_KUBERNETES_HARDENING_GUIDANCE_1.2_20220829.PDF.
- [5] CNCF. «Observability Micro-Survey (2022).» (), URL: https://www.cncf.io/wp-content/uploads/2022/03/CNCF_Observability_MicroSurvey_030222.pdf.
- [6] Kubernetes. «Conceitos.» (), URL: <https://kubernetes.io/docs/concepts/>.
- [7] CSI. «Documentação.» (), URL: <https://kubernetes-csi.github.io/docs/>.
- [8] Kubernetes. «Documentação.» (), URL: <https://kubernetes.io/docs/concepts/storage/storage-classes/>.
- [9] Kubernetes. «Documentação.» (), URL: <https://kubernetes.io/docs/tasks/debug/debug-cluster/audit/>.
- [10] Falco. «Documentação.» (), URL: <https://falco.org/docs/>.
- [11] Falcosidekick. «GitHub README.» (), URL: <https://github.com/falcosecurity/falcosidekick?tab=readme-ov-file#description>.

- [12] F. Talon. «Documentação.» (), URL: <https://docs.falco-talon.org/docs/overview/>.
- [13] Grafana. «Documentação.» (), URL: <https://grafana.com/docs/loki/latest/>.
- [14] Grafana. «Documentação.» (), URL: <https://grafana.com/docs/grafana-cloud/visualizations/>.
- [15] Prometheus. «Documentação.» (), URL: <https://prometheus.io/docs/introduction/overview/>.
- [16] CVE.org. «CVE-2021-25741.» (), URL: <https://www.cve.org/CVERecord?id=CVE-2021-25741>.
- [17] Kubernetes. «Documentação.» (), URL: <https://kubernetes.io/docs/concepts/storage/volumes/#hostpath>.
- [18] Kubernetes. «Auditing (docs).» (), URL: <https://kubernetes.io/docs/tasks/debug/debug-cluster/audit/>.

Apêndice A

Proposta Original do Projeto



INSTITUTO POLITÉCNICO DE BRAGANÇA
Escola Superior de Tecnologia e Gestão

Curso de Licenciatura em Engenharia Informática
Projeto 3º Ano - Ano letivo de 2024/2025

**Mitigating Attacks on Persistent Volume Claims: Leveraging Audit
Logs for Security in Kubernetes**

Inteligência artificial	Multimédia/Realidade aumentada/...
Aplicações móveis	Redes e gestão de sistemas
Plataformas web	X * Cibersegurança

* - A especificar pelo proponente

Orientador: Rui Alves
Coorientador: Tiago Pedrosa

1 **Objetivo**
Este estudo visa investigar como os logs podem ser utilizados para detectar atividades suspeitas e mitigar os riscos associados nos PVCs por forma a garantir a proteção e a integridade dos dados armazenados.

2 **Detalhes**
A adoção de plataformas de orquestração de containers é cada vez mais comum na maioria das empresas permitindo a implantação e a gestão eficiente de aplicações em ambientes distribuídos. Dentro desse contexto, os Persistent Volume Claims (PVCs) desempenham um papel crucial, fornecendo armazenamento persistente e confiável para dados essenciais das aplicações. No entanto, a preocupação com a segurança, especialmente no que diz respeito à vulnerabilidade dos PVCs a diferentes tipos de ataques (ex: injeção de malware, acesso não autorizado a dados sensíveis, entre outros) é um risco que deve ser considerado dado a sua natureza complexa de detetar e mitigar.

3 **Metodologia de trabalho**
1 - Levantamento do estado da arte.
2 – Exploração de ferramentas e tecnologias associadas ao domínio do problema.
3 – Desenvolvimento da camada de código para implementação dos objetivos descritos.
4 – Escrita do Relatório.

Dimensão da equipa:	2 Alunos
Recursos necessários:	Computador Pessoal

Figura A.1: Proposta original do projeto.

Apêndice B

Instalações

Instalação do servidor - *projectServer*

```
1 curl -sL https://get.k3s.io | sh -
2 # Verificar se o serviço está a correr
3 sudo systemctl status k3s
4 sudo k3s kubectl get nodes -o wide
5 # Ver o node-token para futura instalação nas máquinas agent e metrics
6 sudo cat /var/lib/rancher/k3s/server/node-token
```

Instalação do agent e metrics

```
1 # Em cada VM de agent e metrics
2 curl -sL https://get.k3s.io | K3S_URL=https://192.168.1.84:6443
   ↪ K3S_TOKEN=<TOKEN> sh -
3 # Objetivo dos comandos seguintes é usar `kubectl` localmente nas máquinas,
   ↪ apontando ao IP da API do servidor (192.168.1.84).
4 # 1) Copiar o kubeconfig do servidor para a VM atual
5 mkdir -p ~/.kube
6 scp projectServer@192.168.1.84:/etc/rancher/k3s/k3s.yaml ~/.kube/config
7 #2) Ajustar o endpoint para o IP do servidor (e não o localhost: 127.0.0.1)
```

```
8 sed -i 's/127\.0\.0\.1/192.168.1.84/' ~/.kube/config
9 #3) Proteger o kubeconfig
10 chmod 600 ~/.kube/config
11 #4) Verificar se o serviço K3S está a funcionar
12 kubectl get nodes -o wide
```

```
projeto@projeto:~$ kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
projeto-agent	Ready	<none>	69d	v1.32.4+k3s1
projeto-metric	Ready	<none>	67d	v1.32.4+k3s1
projeto-server	Ready	control-plane,master	235d	v1.32.4+k3s1

Figura B.1: Nós do cluster com o status *Ready*.

Instalação do Helm nas três máquinas

```
1 #1) Instalar a partir do script oficial em cada VM
2 curl -fsSL
   ↪ https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 |
   ↪ bash
3 #2) Confirmar a instalação e a sua versão
4 helm version
```

Instalação do Kube-Prometheus-Stack

```
1 # Criar o namespace de monitorização
2 kubectl create namespace monitoring
3 # Adicionar o repositório prometheus-community e atualizar índices
4 helm repo add prometheus-community
   ↪ https://prometheus-community.github.io/helm-charts
5 helm repo update
6 #Instalar o chart kube-prometheus-stack
7 helm upgrade --install kube-prometheus-stack
   ↪ prometheus-community/kube-prometheus-stack --namespace monitoring
```

```

8 #Verificar os pods e serviços que foram criados
9 kubectl -n monitoring get pods
10 kubectl -n monitoring get svc

```

Acesso aos serviços partir do Windows

```

1 #Verificar o nome dos serviços
2 kubectl -n monitoring get svc
3 #Patch simples - o cluster escolhe automaticamente uma porta livre
4 kubectl -n monitoring patch svc prometheus-stack-grafana -p
  ↪ '{"spec":{"type":"NodePort"}}'
5 kubectl -n monitoring patch svc prometheus-stack-kube-prom-prometheus -p
  ↪ '{"spec":{"type":"NodePort"}}'
6 #Verificar as portas atribuídas
7 kubectl -n monitoring get svc

```

```

projeto@projeto:~$ kubectl get svc -n monitoring

```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
alertmanager-operated	ClusterIP	None	<none>	9093/TCP, 9094/TCP, 9094/UDP	59d
prometheus-operated	ClusterIP	None	<none>	9090/TCP	59d
prometheus-stack-grafana	NodePort	10.43.170.241	<none>	80:30449/TCP	59d
prometheus-stack-kube-prom-alertmanager	ClusterIP	10.43.1.50	<none>	9093/TCP, 8080/TCP	59d
prometheus-stack-kube-prom-operator	NodePort	10.43.54.28	<none>	443:32589/TCP	59d
prometheus-stack-kube-prom-prometheus	NodePort	10.43.227.227	<none>	9090:31040/TCP, 8080:32527/TCP	59d
prometheus-stack-kube-state-metrics	ClusterIP	10.43.39.107	<none>	8080/TCP	59d
prometheus-stack-prometheus-node-exporter	ClusterIP	10.43.175.161	<none>	9100/TCP	59d

Figura B.2: Serviços do Kube-Prometheus-Stack.

Falco

values-falco.yaml:

```

1 falco:
2   json_output: true
3   priority: notice
4
5   # Apenas o plugin "container"
6   load_plugins: [container]

```

```

7   plugins:
8     - name: container
9       library_path: libcontainer.so
10
11  # envia eventos para o Falcosidekick
12  http_output:
13    enabled: true
14    url: "http://falcosidekick.falco:2801/"
15    keep_alive: true
16
17  # Webserver pra/ health/metrics
18  webserver:
19    enabled: true
20    listen_port: 8765
21    prometheus_metrics_enabled: true
22
23  # Regras base + diretório de extras
24  rules_files:
25    - /etc/falco/falco_rules.yaml
26    - /etc/falco/rules.d
27
28  driver:
29    enabled: true
30    kind: ebpf
31
32  customRules:
33    simple-open-for-write.yaml: |-
34      - macro: any_open_for_write
35        condition: (evt.type in (open, openat, openat2) and
36          ↪ evt.is_open_write=true and fd.type=file and evt.dir=<)

```

```

36
37 - rule: Storage open-for-write
38   desc: "Qualquer abertura de ficheiro para escrita (agregação/alerta no
        ↳ backend)."
39   condition: any_open_for_write and proc.name != falco
40   enabled: true
41   output: >-
42     Storage write open | user=%user.name file=%fd.name proc=%proc.name
        ↳ pid=%proc.pid cmd=%proc.cmdline container=%container.id
43   priority: NOTICE
44   source: syscall
45   tags: [filesystem, storage]

```

FalcoSideKick

values-falcoSidekick.yaml:

```

1 config:
2   loki:
3     hostport: http://loki.loki.svc.cluster.local:3100
4     endpoint: /loki/api/v1/push
5     tenant: fake
6     minimumpriority: debug
7     customfields: ["app=falco"]
8
9   webhook:
10     address: http://falco-talon.falco:2803/
11     minimumpriority: debug

```

FalcoTalon

values-talon.yaml:

```
1 watchRules: true
2
3 service:
4   type: ClusterIP
5   port: 2803
6
7 rbac:
8   create: true
9
10 config:
11   rulesOverride: |
12     - action: Label Suspicious
13       actionner: kubernetes:label
14       parameters:
15         level: pod
16         labels: { suspicious: "true" }
17
18     - action: Terminate Pod
19       actionner: kubernetes:terminate
20       parameters:
21         grace_period_seconds: 5
22         ignore_statefulsets: true
23         ignore_daemonsets: true
24
25     - rule: PVC IO spike (label)
26       match: { source: prometheus, rules: [PVC_IO_Spike_1x] }
27       actions: [ { action: Label Suspicious } ]
28
29     - rule: PVC IO spike (kill)
30       match: { source: prometheus, rules: [PVC_IO_Spike_3x] }
```

```

31     actions: [ { action: Terminate Pod } ]
32
33     replicaCount: 2
34     logLevel: debug

```

Implementação do Falco, Sidekick e Talon via Helm

```

1  helm repo add falcosecurity https://falcosecurity.github.io/charts
2  helm repo update
3  helm upgrade --install falco falcosecurity/falco \
4    -n falco --create-namespace \
5    -f values-falco.yaml
6  kubectl -n falco get pods -o wide

```

```

projectmetric@projectmetric:~$ kubectl -n falco get pods -l app.kubernetes.io/name=falco
NAME          READY   STATUS    RESTARTS   AGE
falco-hg5tg   1/1     Running   9 (120m ago)  10d
falco-jcv88   1/1     Running   9 (119m ago)  10d
falco-vmcdd   1/1     Running   9 (120m ago)  10d
projectmetric@projectmetric:~$

```

Figura B.3: *Pods* do Falco.

Confirmar que a regra personalizada está montada no *pod*:

```

1  kubectl -n falco exec -ti ds/falco -- sh -lc "grep -n 'Storage
↪   open-for-write' /etc/falco/rules.d/*.yaml || true"

```

```

projectmetric@projectmetric:~$ kubectl -n falco exec -ti ds/falco -- \
sh -lc "grep -n 'Storage open-for-write' /etc/falco/rules.d/*.yaml || true"
Defaulted container "falco" out of: falco, falco-driver-loader (init)
4:- rule: Storage open-for-write
projectmetric@projectmetric:~$

```

Figura B.4: Regra montada nos *Pods*.


```

1 helm upgrade --install falcosidekick falcosecurity/falcosidekick \
2   -n falco \
3   -f values-falcosidekick.yaml
4 kubectl -n falco get pods -l app.kubernetes.io/name=falcosidekick -o wide

```

```

projectmetric@projectmetric:~$ kubectl -n falco get pods -l app.kubernetes.io/name=falcosidekick -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP              NODE               NOMINATED NODE   READINESS GATES
falcosidekick-dfd55b8fc-4m5sv       1/1     Running   7 (4m1s ago)  7d18h  10.42.1.94      projectagent        <none>            <none>
falcosidekick-dfd55b8fc-df5ll       1/1     Running   7 (4m7s ago)  7d18h  10.42.2.64      projectmetric        <none>            <none>
projectmetric@projectmetric:~$

```

Figura B.5: *Pods* do FalcoSidekick.

Consultar os logs:

```

1 kubectl -n falco logs -l app.kubernetes.io/name=falcosidekick --since=1m

```

```

projectmetric@projectmetric:~$ kubectl -n falco logs -l app.kubernetes.io/name=falcosidekick --since=1m
2025/08/31 15:12:48 [INFO] : Loki - POST OK (284)
2025/08/31 15:12:48 [INFO] : Webhook - POST OK (280)
2025/08/31 15:12:50 [INFO] : Loki - POST OK (284)
2025/08/31 15:12:50 [INFO] : Loki - POST OK (284)
2025/08/31 15:12:50 [INFO] : Webhook - POST OK (280)
2025/08/31 15:12:50 [INFO] : Webhook - POST OK (280)
2025/08/31 15:12:52 [INFO] : Webhook - POST OK (280)
2025/08/31 15:12:52 [INFO] : Webhook - POST OK (280)
2025/08/31 15:12:52 [INFO] : Loki - POST OK (284)
2025/08/31 15:12:52 [INFO] : Loki - POST OK (284)
projectmetric@projectmetric:~$

```

Figura B.6: *Logs* do FalcoSidekick.

Instalar/atualizar Talon:

```

1 helm upgrade --install falco-talon falcosecurity/falco-talon \
2   -n falco \
3   -f values-talon.yaml

```

```

projectmetric@projectmetric:~$ kubectl -n falco get pods -l app.kubernetes.io/name=falco-talon
NAME                                READY   STATUS    RESTARTS   AGE   IP              NODE               NOMINATED NODE   READINESS GATES
falco-talon-69db47db7c-nx9j4       1/1     Running   7 (120m ago)  7d21h  10.42.1.94      projectagent        <none>            <none>
falco-talon-69db47db7c-s4v2m       1/1     Running   7 (120m ago)  7d21h  10.42.2.64      projectmetric        <none>            <none>
projectmetric@projectmetric:~$

```

Figura B.7: *Pods* do Talon.

```

1  kubectl -n falco create configmap falco-talon-rules --from-file=rules.yaml
   ↪ -o yaml --dry-run=client | kubectl -n falco apply -f -
2  kubectl -n falco get configmap falco-talon-rules -o yaml | sed -n '1,120p'

```

```

projectmetric@projectmetric:~$ kubectl -n falco get configmap falco-talon-rules -o yaml | sed -n '1,120p'
apiVersion: v1
data:
  rules.yaml: |2-
    - action: Terminate Pod
      actionner: kubernetes:terminate

    - action: Label Pod as Suspicious
      actionner: kubernetes:label
      parameters:
        labels:
          analysis/status: "suspicious"

    # --- Ações reutilizáveis ---
    - action: Label Suspicious
      actionner: kubernetes:label
      parameters:
        level: pod
        labels:
          suspicious: "true"

    - action: Terminate Pod
      actionner: kubernetes:terminate
      parameters:
        grace_period_seconds: 5
        ignore_statefulsets: true
        ignore_daemonsets: true

    # --- Regra 1: primeira ocorrência => Label ---
    - rule: PVC IO spike (Label)
      match:
        source: loki
        rules:
          - PVC_IO_Spike_1x
      actions:
        - action: Label Suspicious

    # --- Regra 2: terceira ocorrência => terminar ---
    - rule: PVC IO spike (kill)
      match:
        source: loki
        rules:
          - PVC_IO_Spike_3x
      actions:
        - action: Terminate Pod
kind: ConfigMap
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      [{"apiVersion":"v1","data":{"rules.yaml":""},"kind":"ConfigMap","metadata":{"annotations":{},"creationTimestamp":null,"name":"falco-talon-rules","namespace":"falco"}}]
    meta.helm.sh/release-name: falco-talon
    meta.helm.sh/release-namespace: falco
    creationTimestamp: "2025-08-14T16:52:57Z"
  labels:
    app.kubernetes.io/instance: falco-talon
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/name: falco-talon
    app.kubernetes.io/part-of: falco-talon
    app.kubernetes.io/version: 0.3.0
    helm.sh/chart: falco-talon-0.3.0
  name: falco-talon-rules
  namespace: falco
  resourceVersion: "478426"
  uid: e45bc3ad-d4a9-4cc8-a2bc-df6318acd84f
projectmetric@projectmetric:~$

```

Figura B.8: ConfigMap do Falco Talon.

Instalação do Loki

```

1  helm repo add grafana https://grafana.github.io/helm-charts
2  helm repo update
3  kubectl create ns loki
4  helm upgrade --install loki grafana/loki -n loki -f values-loki.yaml
5  kubectl -n loki get pods -o wide

```

values-loki.yaml:

```
1 deploymentMode: SingleBinary
2 singleBinary:
3   replicas: 1
4   # garantir que o modo SimpleScalable fica a 0
5 read:
6   replicas: 0
7 write:
8   replicas: 0
9 backend:
10  replicas: 0
11 loki:
12   auth_enabled: false
13   commonConfig:
14     replication_factor: 1
15   # por defeito o chart usa S3 - muda para filesystem
16   storage:
17     type: filesystem
18     filesystem:
19       chunks_directory: /var/loki/chunks
20       rules_directory: /var/loki/rules
21       admin_api_directory: /var/loki/admin
22   # schema mínima para tsdb + filesystem
23   schemaConfig:
24     configs:
25       - from: "2024-01-01"
26         store: tsdb
27         object_store: filesystem
28         schema: v13
29         index:
```

```
30         prefix: loki_index_
31         period: 24h
```

```

projectjetic@projectjetic:~$ kubectl -n loki get pods --no-headers
NAME          READY   STATUS    RESTARTS   AGE   IP            NODE          NOMINATED NODE   READINESS GATES
curl          0/1     Completed 0           28d   <none>        projectagent   <none>            <none>
curl-push     0/1     Completed 0           28d   <none>        projectagent   <none>            <none>
promtail      2/2     Running   3 (121m ago)  3d22h  10.42.1.107   projectagent   <none>            <none>
loki-canary-w54znz  1/1     Running   22 (121m ago)  28d   10.42.2.78   projectjetic   <none>            <none>
loki-canary-vh2znz  1/1     Running   22 (122m ago)  28d   10.42.0.113  projectserver   <none>            <none>
loki-canary-w5shwz  1/1     Running   22 (121m ago)  28d   10.42.1.101  projectagent   <none>            <none>
promtail-6xjxj  1/1     Running   22 (121m ago)  28d   10.42.1.185  projectagent   <none>            <none>
promtail-fvfrt  1/1     Running   22 (121m ago)  28d   10.42.2.76   projectjetic   <none>            <none>
promtail-zfpzf  1/1     Running   22 (122m ago)  28d   10.42.0.114  projectserver   <none>            <none>
projectjetic@projectjetic:~$

```

Figura B.9: *Pods* do Loki.

Verificação do serviço e *logs* do Loki:

```
1 kubectl -n loki get svc
```

```
projectmetric@projectmetric:~$ kubectl -n loki get svc
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
loki	NodePort	10.43.36.73	<none>	3100:31300/TCP	20d
loki-canary	ClusterIP	10.43.213.60	<none>	3500/TCP	20d
loki-headless	ClusterIP	None	<none>	3100/TCP	20d
loki-memberlist	ClusterIP	None	<none>	7946/TCP	20d

Figura B.10: *Services* do Loki.

```
1 kubectl -n loki logs -l app.kubernetes.io/name=loki --tail=200 | egrep -i
   ↳ 'ingester|received|tenant|fake'
```

```

level=info src=2025-08-23T22:34:32.103Z kubectnl -n lokloggs -k app.kubernetes.io/name=lokkit -eval-200 | egrep -A 1 'ingester\[received|cancelled\]'
level=info src=2025-08-23T22:34:32.103Z caller=metrics.go:237 component=ruler evaluation_model=logical obj-id=fake traceID=3c7f7e982ed7f latency=fast query=(sum by (h8s_name,h8s_pod_name,persistentvolumeclaim) count over time(rules_storage_op_for_write["source"], h8s_name=~"V", h8s_pod_name=~"V", persistentvolumeclaim=~"V"] duration=1s) h8s_name=~"V" limit=0 returned_lines=0 throughput=0 total_bytes_structured_metadata=0 total_bytes_per_second=0 total_lines=0 post_filter_lines=0 total_entries=0 store_chunks_download_time=0 queue_time=0 splits=0 shards=0 query_referenced_structured_metadata=fake pipeline_wrapper.filtered_lines=0 chunk_refs_fetch_time=09 us cache_chunk_req=0 cache_chunk_hit=0 cache_chunk_miss=0 cache_chunk_not_found=0 cache_chunk_not_found_time=0 cache_chunk_not_found_time=0 cache_chunk_not_found_time=0 cache_chunk_not_found_time=0 cache_chunk_not_found_time=0 hit=0 cache_stats_result_download_time=0 cache_volume_result_req=0 cache_volume_result_download_time=0 cache_volume_result_req=0 cache_result_req=0 cache_result_hit=0 cache_result_download_time=0 cache_result_query_length_served=0 cardinality.estimate=0 ingester_chunk_refs=0 ingester_chunk_download=0 ingester_chunk_matches=0 ingester_request=1 ingester_chunk_head_bytes=0 ingester_chunk_compressed_bytes=0 ingester_chunk_decompressed_bytes=0 ingester_post_filter_lines=0 congestion_controller.latency=85 index_total_chunks=0 index_post_block_filter_lines=0 index_post_block_filter_lines=0 index_post_block_filter_lines=0 index_post_block_filter_lines=0 index_post_block_filter_lines=0)
level=info src=2025-08-23T22:34:32.103Z caller=comp_got.go:68 user-fake rule_name=PVC-10_Spike_3x NOPVC rule_type=alerting query=(sum by (h8s_name,h8s_pod_name) count over time(rules_storage_op_for_write["source"], h8s_name=~"V", h8s_pod_name=~"V", lokkit[monitoring.V5m]) 3s) query_hash=38262034088 ass=evaluating rule

```

Figura B.11: Logs do Loki.

Ligação Falcosidekick com o Loki

values-falcosidekick.yaml:

```
1 config:
2   loki:
3     hostport: "http://loki.loki.svc.cluster.local:3100"
4     format: "json"
5     extralabels: "app=falco"    # útil para queries
```

Aplicar configuração no Sidekick:

```
1 helm upgrade --install falcosidekick falcosecurity/falcosidekick -n falco \
2   --reuse-values -f values-falcosidekick.yaml
```

```
projectmetric@projectmetric:~$ kubectl -n falco logs -l app.kubernetes.io/name=falcosidekick --since=1m
2025/08/31 15:12:48 [INFO] : Loki - POST OK (204)
2025/08/31 15:12:48 [INFO] : Webhook - POST OK (200)
2025/08/31 15:12:50 [INFO] : Loki - POST OK (204)
2025/08/31 15:12:50 [INFO] : Loki - POST OK (204)
2025/08/31 15:12:50 [INFO] : Webhook - POST OK (200)
2025/08/31 15:12:50 [INFO] : Webhook - POST OK (200)
2025/08/31 15:12:52 [INFO] : Webhook - POST OK (200)
2025/08/31 15:12:52 [INFO] : Webhook - POST OK (200)
2025/08/31 15:12:52 [INFO] : Loki - POST OK (204)
2025/08/31 15:12:52 [INFO] : Loki - POST OK (204)
```

Figura B.12: Loki a receber e a enviar para o SideKick.

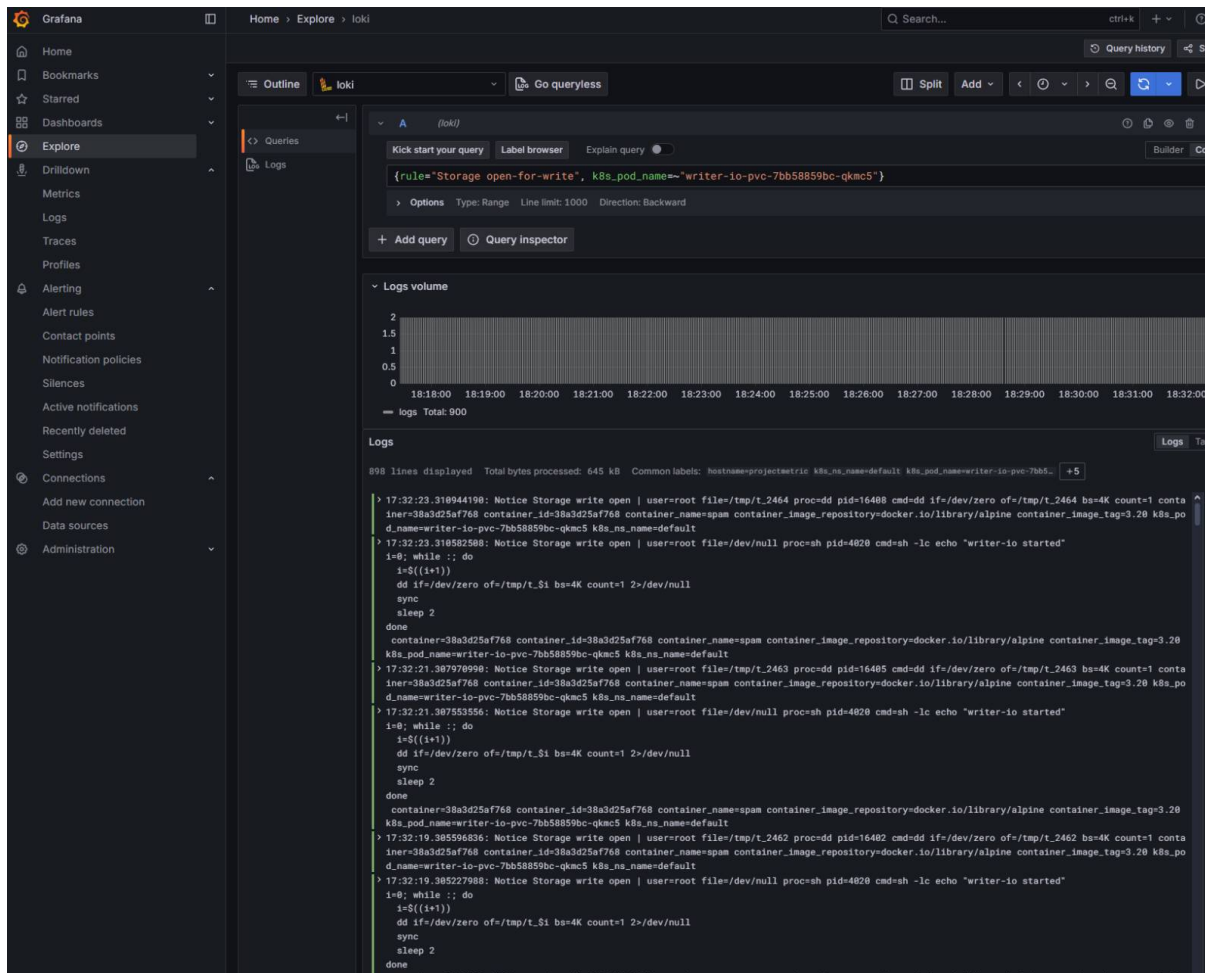


Figura B.13: Loki a funcionar no Grafana.

LokiRuler

```

1 kubectl -n loki apply -f loki-ruler-falco-io.yaml
2 kubectl -n loki get configmap loki-ruler-falco-io -o yaml | sed -n '1,120p'
3 kubectl -n loki logs -l app.kubernetes.io/name=loki --since=2m | egrep -i
   ↪ 'ruler|rule|load'

```

```

apiVersion: v1
kind: ConfigMap
metadata:
  name: loki-ruler-falco-io
  namespace: loki
data:
  alerts.yaml: |
    - alert: PVC_IO_Spike_1x
      expr: sum by (k8s_ns_name,k8s_pod_name,persistentvolumeclaim) (
        count_over_time([k8s_namespace="falco",source="system"]
          kube_io_volume_write{job="falco"}[15m])
      ) > 1
      labels:
        severity: warning
      annotations:
        summary: "PVC IO Spike (k8s_ns_name={{k8s_ns_name}}) com taxa de 1x"
        description: "Pod {{k8s_ns_name}}/{{k8s_pod_name}} escreveu no PVC {{k8s_pv_name}} com taxa de 1x."
    - alert: PVC_IO_Spike_3x
      expr: sum by (k8s_ns_name,k8s_pod_name,persistentvolumeclaim) (
        count_over_time([k8s_namespace="falco",source="system"]
          kube_io_volume_write{job="falco"}[15m])
      ) > 3
      labels:
        severity: critical
      annotations:
        summary: "PVC IO Spike (k8s_ns_name={{k8s_ns_name}}) com taxa de 3x"
        description: "Pod {{k8s_ns_name}}/{{k8s_pod_name}} escreveu 3x no PVC {{k8s_pv_name}} com taxa de 3x."
  kind: ConfigMap
  apiVersion: v1
  kind: ConfigMap
  metadata:
    name: loki-ruler-falco-io
    namespace: loki
  data:
    alerts.yaml: |
      - alert: PVC_IO_Spike_1x
        expr: sum by (k8s_ns_name,k8s_pod_name,persistentvolumeclaim) (
          count_over_time([k8s_namespace="falco",source="system"]
            kube_io_volume_write{job="falco"}[15m])
        ) > 1
        labels:
          severity: warning
        annotations:
          summary: "PVC IO Spike (k8s_ns_name={{k8s_ns_name}}) com taxa de 1x"
          description: "Pod {{k8s_ns_name}}/{{k8s_pod_name}} escreveu no PVC {{k8s_pv_name}} com taxa de 1x."
      - alert: PVC_IO_Spike_3x
        expr: sum by (k8s_ns_name,k8s_pod_name,persistentvolumeclaim) (
          count_over_time([k8s_namespace="falco",source="system"]
            kube_io_volume_write{job="falco"}[15m])
        ) > 3
        labels:
          severity: critical
        annotations:
          summary: "PVC IO Spike (k8s_ns_name={{k8s_ns_name}}) com taxa de 3x"
          description: "Pod {{k8s_ns_name}}/{{k8s_pod_name}} escreveu 3x no PVC {{k8s_pv_name}} com taxa de 3x."

```

Figura B.14: Regra do Loki-Ruler no ConfigMap.

```

# Loki-Ruler ConfigMap
apiVersion: v1
kind: ConfigMap
metadata:
  name: loki-ruler-falco-io
  namespace: loki
data:
  alerts.yaml: |
    - alert: PVC_IO_Spike_1x
      expr: sum by (k8s_ns_name,k8s_pod_name,persistentvolumeclaim) (
        count_over_time([k8s_namespace="falco",source="system"]
          kube_io_volume_write{job="falco"}[15m])
      ) > 1
      labels:
        severity: warning
      annotations:
        summary: "PVC IO Spike (k8s_ns_name={{k8s_ns_name}}) com taxa de 1x"
        description: "Pod {{k8s_ns_name}}/{{k8s_pod_name}} escreveu no PVC {{k8s_pv_name}} com taxa de 1x."
    - alert: PVC_IO_Spike_3x
      expr: sum by (k8s_ns_name,k8s_pod_name,persistentvolumeclaim) (
        count_over_time([k8s_namespace="falco",source="system"]
          kube_io_volume_write{job="falco"}[15m])
      ) > 3
      labels:
        severity: critical
      annotations:
        summary: "PVC IO Spike (k8s_ns_name={{k8s_ns_name}}) com taxa de 3x"
        description: "Pod {{k8s_ns_name}}/{{k8s_pod_name}} escreveu 3x no PVC {{k8s_pv_name}} com taxa de 3x."
  kind: ConfigMap
  apiVersion: v1
  kind: ConfigMap
  metadata:
    name: loki-ruler-falco-io
    namespace: loki
  data:
    alerts.yaml: |
      - alert: PVC_IO_Spike_1x
        expr: sum by (k8s_ns_name,k8s_pod_name,persistentvolumeclaim) (
          count_over_time([k8s_namespace="falco",source="system"]
            kube_io_volume_write{job="falco"}[15m])
          ) > 1
        labels:
          severity: warning
        annotations:
          summary: "PVC IO Spike (k8s_ns_name={{k8s_ns_name}}) com taxa de 1x"
          description: "Pod {{k8s_ns_name}}/{{k8s_pod_name}} escreveu no PVC {{k8s_pv_name}} com taxa de 1x."
      - alert: PVC_IO_Spike_3x
        expr: sum by (k8s_ns_name,k8s_pod_name,persistentvolumeclaim) (
          count_over_time([k8s_namespace="falco",source="system"]
            kube_io_volume_write{job="falco"}[15m])
          ) > 3
        labels:
          severity: critical
        annotations:
          summary: "PVC IO Spike (k8s_ns_name={{k8s_ns_name}}) com taxa de 3x"
          description: "Pod {{k8s_ns_name}}/{{k8s_pod_name}} escreveu 3x no PVC {{k8s_pv_name}} com taxa de 3x."

```

Figura B.15: Logs do Loki-Ruler.

loki-ruler-falco-io.yaml:

```

1  apiVersion: v1
2  kind: ConfigMap
3  metadata:
4    name: loki-ruler-falco-io
5    namespace: loki
6    labels:
7      ruler.loki.grafana.com/rule: "true" # <- muito importante
8  data:
9    alerts.yaml: |
10     groups:
11       - name: pvc-io
12         interval: 30s
13       rules:
14         - alert: PVC_IO_Spike_1x
15           expr: |
16             sum by (k8s_ns_name,k8s_pod_name,persistentvolumeclaim) (

```

```

17         count_over_time({rule="Storage open-for-write",
18             ↪ source="synthetic",
19             k8s_ns_name=~".+", k8s_pod_name=~".+",
20             ↪ persistentvolumeclaim=~".+"}[15m])
21     ) >= 1
22 for: 0m
23 labels:
24     severity: warning
25 annotations:
26     summary: "PVC {{ $labels.persistentvolumeclaim }} com escrita
27     ↪ detetada"
28     description: "Pod {{ $labels.k8s_ns_name }}/{{ $labels.k8s_pod_name
29     ↪ }} escreveu no PVC {{ $labels.persistentvolumeclaim }}
30     ↪ (>=1x/15m)."
```

- alert: PVC_IO_Spike_3x

```

28     expr: |
29         sum by (k8s_ns_name,k8s_pod_name,persistentvolumeclaim) (
30             count_over_time({rule="Storage open-for-write",
31                 ↪ source="synthetic",
32                 k8s_ns_name=~".+", k8s_pod_name=~".+",
33                 ↪ persistentvolumeclaim=~".+"}[15m])
34             ) >= 3
35 for: 0m
36 labels:
37     severity: critical
38 annotations:
39     summary: "PVC {{ $labels.persistentvolumeclaim }} com alta taxa de
40     ↪ IO"
41     description: "Pod {{ $labels.k8s_ns_name }}/{{ $labels.k8s_pod_name
42     ↪ }} excedeu 3 aberturas em 15m no PVC {{
43     ↪ $labels.persistentvolumeclaim }}."
```

Queries úteis

```
1 # Ocorrências por PVC/pod/namespace em 15m
2 sum by (k8s_ns_name, k8s_pod_name,
   ↪ persistentvolumeclaim)(count_over_time({rule="Storage
   ↪ open-for-write"}[15m]))
3 # Apenas pods/volumes com >= 3 ocorrências (deve alinhar com o alerta 3x)
4 sum by (k8s_ns_name, k8s_pod_name,
   ↪ persistentvolumeclaim)(count_over_time({rule="Storage
   ↪ open-for-write"}[15m]))>= 3
```

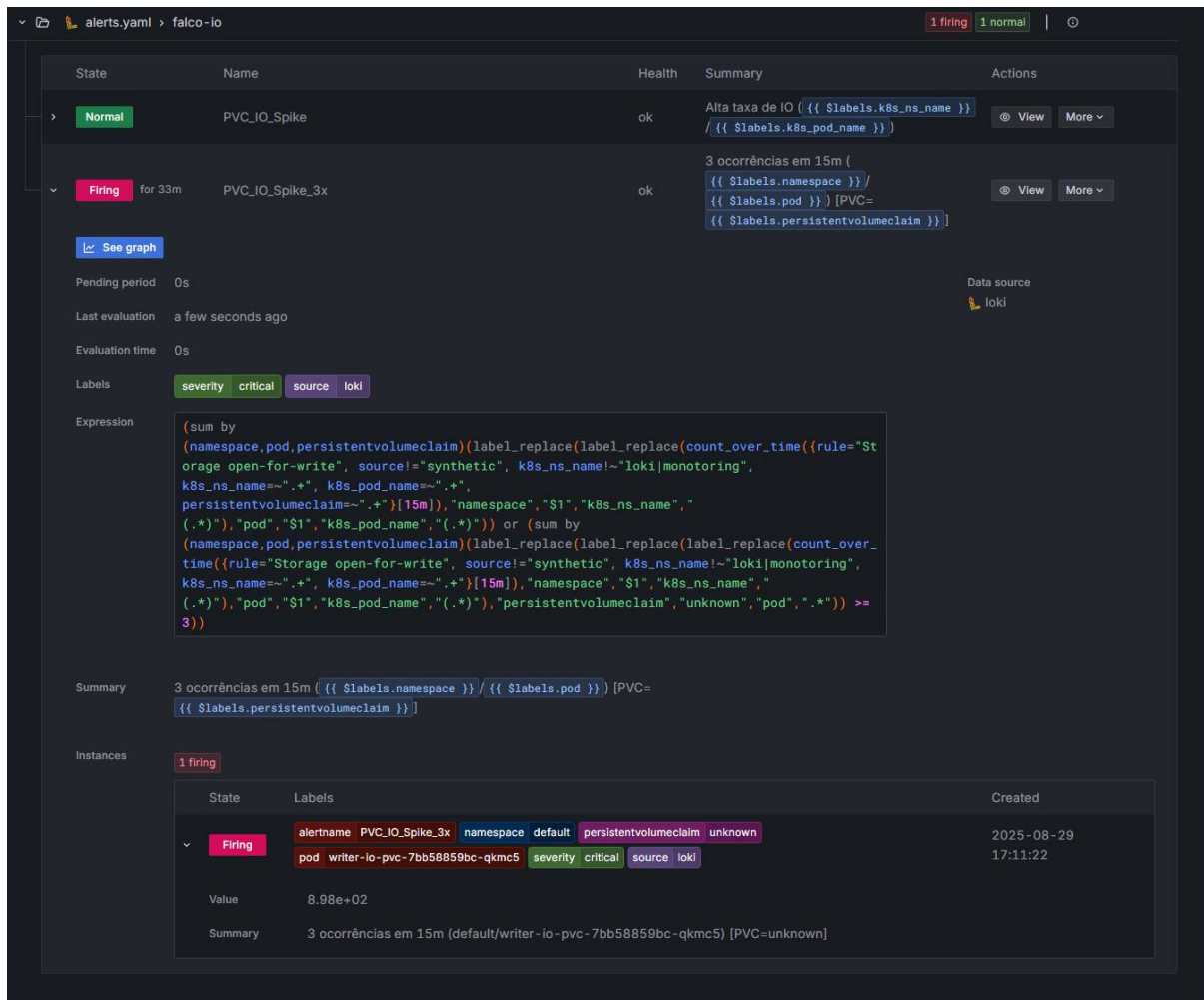


Figura B.16: Loki Rules no Grafana.

Bridge:

```
1 kubectl apply -f am-to-talon-bridge.yaml
```

am-to-talon-bridge.yaml:

```
1 apiVersion: v1
2 kind: Namespace
3 metadata:
4   name: observability
```

```

5  ---
6  apiVersion: v1
7  kind: ConfigMap
8  metadata:
9      name: am-to-talon-bridge
10     namespace: observability
11 data:
12     app.py: |
13         import os, json, datetime, time, re
14         import requests
15         from flask import Flask, request, jsonify
16
17         # --- K8s client (para resolver pod por PVC/família) ---
18         try:
19             from kubernetes import client as k8s_client, config as k8s_config
20         except Exception:
21             k8s_client = None
22             k8s_config = None
23
24         TALON_URL = os.getenv("TALON_URL",
25                               ↪ "http://falco-talon.falco.svc.cluster.local:2803/alert")
26         RULE_FALLBACK = os.getenv("RULE_FALLBACK", "AlertmanagerEvent")
27         PRIORITY_DEFAULT = os.getenv("PRIORITY_DEFAULT", "Critical")
28
29         LABEL_KEYS_NS = os.getenv("LABEL_KEYS_NS",
30                                   ↪ "namespace,k8s_ns_name").split(",")
31         LABEL_KEYS_POD = os.getenv("LABEL_KEYS_POD",
32                                    ↪ "pod,k8s_pod_name").split(",")
33         LABEL_KEYS_PVC = os.getenv("LABEL_KEYS_PVC",
34                                    ↪ "persistentvolumeclaim").split(",")

```

```

31
32 app = Flask(__name__)
33
34 # ----- utils -----
35 def pick(d, keys):
36     for k in keys:
37         if k in d and d[k]:
38             return d[k]
39     return None
40
41 _K8S = None
42 def init_k8s():
43     """Inicializa cliente CoreV1 uma vez (in-cluster)."""
44     global _K8S
45     if _K8S is not None or not (k8s_client and k8s_config):
46         return
47     try:
48         k8s_config.load_incluster_config()
49         _K8S = k8s_client.CoreV1Api()
50         print("K8S client: ready", flush=True)
51     except Exception as e:
52         print(f"K8S client init failed: {e}", flush=True)
53         _K8S = None
54
55 def exists_and_not_terminating(ns, name):
56     if not _K8S or not name or name == "unknown":
57         return False
58     try:
59         p = _K8S.read_namespaced_pod(name=name, namespace=ns)
60         return getattr(p.metadata, "deletion_timestamp", None) is None

```

```

61     except Exception:
62         return False
63
64 def wait_ready_current(ns, name, timeout=3.0, interval=0.2):
65     """Espera curto para o pod existir e não estar a terminar (mitiga
66     ↪ race)."""
67     deadline = time.time() + timeout
68     while time.time() < deadline:
69         if exists_and_not_terminating(ns, name):
70             return True
71         time.sleep(interval)
72     return False
73
74 def _score_pod(p):
75     """Ordena Running > Pending > outros; depois mais recente."""
76     phase = (p.status.phase or "").lower()
77     pri = 0 if phase == "running" else (1 if phase == "pending" else 2)
78     ts = p.metadata.creation_timestamp or
79     ↪ datetime.datetime.min.replace(tzinfo=datetime.timezone.utc)
80     return (pri, -ts.timestamp())
81
82 def resolve_pod_by_pvc(ns, pvc):
83     """Devolve o melhor pod que monta o PVC (Running/Pending e mais
84     ↪ recente)."""
85     if not _K8S or not pvc or pvc == "unknown":
86         return None
87     try:
88         pods = _K8S.list_namespaced_pod(namespace=ns).items
89         cands = []
90         for p in pods:

```

```

88         for v in (p.spec.volumes or []):
89             if v.persistent_volume_claim and
           ↪ v.persistent_volume_claim.claim_name == pvc:
90                 cands.append(p); break
91         if not cands:
92             return None
93         cands.sort(key=_score_pod)
94         return cands[0].metadata.name
95     except Exception as e:
96         print(f"K8S lookup error (pvc): {e}", flush=True)
97         return None
98
99 def resolve_pod_by_family(ns, podname):
100     """Escolhe o pod atual da mesma 'família' (preferindo
           ↪ pod-template-hash)."""
101     if not _K8S or not podname or podname == "unknown":
102         return None
103     try:
104         # tenta extrair o hash do RS no nome: <base>-<rs-hash:10hex>-<suffix>
105         m = re.match(r"^(.+)-([0-9a-f]{10})-[-~]+$", podname)
106         label_selector = f"pod-template-hash={m.group(2)}" if m else None
107
108         pods = (_K8S.list_namespaced_pod(namespace=ns,
           ↪ label_selector=label_selector).items
109                 if label_selector else
           ↪ _K8S.list_namespaced_pod(namespace=ns).items)
110
111         if not label_selector:
112             # fallback: prefixo até ao penúltimo '-'
113             parts = podname.rsplit("-", 1)

```

```

114         if len(parts) >= 2:
115             prefix = parts[0] + "-"
116             pods = [p for p in pods if p.metadata.name.startswith(prefix)]
117
118         if not pods:
119             return None
120
121         pods.sort(key=_score_pod)
122         return pods[0].metadata.name
123     except Exception as e:
124         print(f"K8S lookup error (family): {e}", flush=True)
125         return None
126
127     # ----- endpoints -----
128     @app.route("/healthz")
129     def healthz():
130         return "ok", 200
131
132     @app.route("/alert", methods=["POST"])
133     def alert():
134         try:
135             body = request.get_json(force=True, silent=False)
136         except Exception as e:
137             return jsonify({"error": f"invalid json: {e}"}), 400
138
139         init_k8s() # prepara cliente k8s
140
141         alerts = body.get("alerts", [])
142         sent = 0
143         errors = []

```

```

144     for a in alerts:
145         if a.get("status") != "firing":
146             continue
147         labels = a.get("labels", {}) or {}
148         ann = a.get("annotations", {}) or {}
149
150         ns = pick(labels, LABEL_KEYS_NS) or pick(ann, LABEL_KEYS_NS) or
151             ↪ "default"
152         pod = pick(labels, LABEL_KEYS_POD) or pick(ann, LABEL_KEYS_POD) or
153             ↪ "unknown"
154         pvc = pick(labels, LABEL_KEYS_PVC) or pick(ann, LABEL_KEYS_PVC) or
155             ↪ "unknown"
156
157         # --- resolução do alvo (PVC -> família), garantindo não terminar ---
158         resolved = None
159
160         if pvc and pvc != "unknown":
161             cand = resolve_pod_by_pvc(ns, pvc)
162             if cand and wait_ready_current(ns, cand, timeout=3.0):
163                 if cand != pod:
164                     print(f"Resolved pod by PVC: {ns}/{pvc} -> {cand} (was {pod})",
165                         ↪ flush=True)
166                 else:
167                     print(f"Resolved pod by PVC: {ns}/{pvc} -> {cand}", flush=True)
168                 pod, resolved = cand, cand
169
170         if not resolved:
171             cand = resolve_pod_by_family(ns, pod)
172             if cand and wait_ready_current(ns, cand, timeout=3.0):
173                 if cand != pod:

```



```

170         print(f"Resolved pod by family: {ns}/{pod} -> {cand}",
               ↪ flush=True)
171     else:
172         print(f"Resolved pod by family: {ns}/{pod} -> {cand}
               ↪ (unchanged)", flush=True)
173     pod = cand
174
175     rule = labels.get("alertname") or RULE_FALLBACK
176     sev = labels.get("severity", PRIORITY_DEFAULT)
177     now = datetime.datetime.utcnow().isoformat() + "Z"
178
179     payload = {
180         "output": f"{rule} on {ns}/{pod} [PVC={pvc}]",
181         "priority": (sev or "").capitalize() or PRIORITY_DEFAULT,
182         "rule": rule,
183         "time": now,
184         "output_fields": {
185             "k8s.ns.name": ns,
186             "k8s.pod.name": pod,
187             "k8s.persistentvolumeclaim": pvc
188         },
189         "source": "alertmanager"
190     }
191
192     try:
193         r = requests.post(TALON_URL, json=payload, timeout=5)
194         print(f"TALON RESP {r.status_code}: {r.text}", flush=True)
195         if r.status_code >= 300:
196             errors.append(f"Talon {r.status_code}: {r.text}")
197         else:

```

```

198         sent += 1
199     except Exception as e:
200         errors.append(str(e))
201
202     status = 200 if sent and not errors else (207 if errors else 202)
203     return jsonify({"forwarded": sent, "errors": errors}), status
204
205     if __name__ == "__main__":
206         app.run(host="0.0.0.0", port=8080)
207
208 ---
209 apiVersion: v1
210 kind: ServiceAccount
211 metadata:
212     name: am-to-talon-bridge
213     namespace: observability
214 ---
215 apiVersion: rbac.authorization.k8s.io/v1
216 kind: ClusterRole
217 metadata:
218     name: am-to-talon-bridge-read-pods
219 rules:
220 - apiGroups: [""]
221   resources: ["pods"]
222   verbs: ["get", "list"]
223 ---
224 apiVersion: rbac.authorization.k8s.io/v1
225 kind: ClusterRoleBinding
226 metadata:
227     name: am-to-talon-bridge-read-pods
228 roleRef:

```

```
228   apiGroup: rbac.authorization.k8s.io
229   kind: ClusterRole
230   name: am-to-talon-bridge-read-pods
231 subjects:
232 - kind: ServiceAccount
233   name: am-to-talon-bridge
234   namespace: observability
235 ---
236 apiVersion: apps/v1
237 kind: Deployment
238 metadata:
239   name: am-to-talon-bridge
240   namespace: observability
241   labels:
242     app: am-to-talon-bridge
243 spec:
244   replicas: 1
245   selector:
246     matchLabels:
247       app: am-to-talon-bridge
248   template:
249     metadata:
250       labels:
251         app: am-to-talon-bridge
252     spec:
253       serviceAccountName: am-to-talon-bridge
254       containers:
255       - name: bridge
256         image: python:3.11-slim
257         workingDir: /app
```

```

258     command: ["/bin/sh", "-lc"]
259     args:
260     - |
261         pip install --no-cache-dir flask requests kubernetes && \
262         python app.py
263     env:
264     - name: TALON_URL
265       value: "http://falco-talon.falco.svc.cluster.local:2803/alert"
266     - name: LABEL_KEYS_NS
267       value: "namespace,k8s_ns_name"
268     - name: LABEL_KEYS_POD
269       value: "pod,k8s_pod_name"
270     - name: LABEL_KEYS_PVC
271       value: "persistentvolumeclaim"
272     ports:
273     - name: http
274       containerPort: 8080
275     startupProbe:
276       httpGet:
277         path: /healthz
278         port: http
279         failureThreshold: 24    # ~2min para o pip install
280         periodSeconds: 5
281     readinessProbe:
282       httpGet:
283         path: /healthz
284         port: http
285         periodSeconds: 10
286         timeoutSeconds: 2
287         failureThreshold: 3

```

```

288     livenessProbe:
289         httpGet:
290             path: /healthz
291             port: http
292             periodSeconds: 10
293             timeoutSeconds: 2
294             failureThreshold: 3
295         volumeMounts:
296             - name: app
297               mountPath: /app
298         volumes:
299             - name: app
300               configMap:
301                 name: am-to-talon-bridge
302                 items:
303                     - key: app.py
304                       path: app.py
305 ---
306 apiVersion: v1
307 kind: Service
308 metadata:
309     name: am-to-talon-bridge
310     namespace: observability
311 spec:
312     selector:
313         app: am-to-talon-bridge
314     ports:
315         - name: http
316           port: 8080
317           targetPort: http

```

```
projectmetric@projectmetric:~$ kubectl get pods -n observability
NAME                                READY   STATUS    RESTARTS   AGE
am-to-talon-bridge-65f544b58d-ccfc8 1/1     Running   0           99m
curltest                             0/1     Completed 0           25h
```

Figura B.17: Pod do AM-TO-TALON-BRIDGE.