**Faculdade de Tecnologia de Americana "Ministro Ralph Biasi"**

Raul Beserra de Brito

# IMPACTOS DO CONTROLE DE ACESSO NA SEGURANÇA DE AMBIENTES DE CONTEINERIZAÇÃO COMPARTILHADOS

Trabalho de graduação apresentado como exigência parcial para obtenção do título de Tecnólogo em Segurança da Informação pelo Centro Paula Souza – Faculdade de Tecnologia de Americana – Ministro Ralph Biasi.
Área de concentração: Segurança da Informação.

Americana, 04 de dezembro de 2024

**Banca Examinadora:**

Maria Cristina Aranda
Doutora
Fatec Americana "Ministro Ralph Biasi"

Edson Roberto Gaseta
Mestre
Fatec Americana "Ministro Ralph Biasi"

Henri Alves de Godoy
Doutor
Fatec Americana "Ministro Ralph Biasi"

# IMPACTOS DO CONTROLE DE ACESSO NA SEGURANÇA DE AMBIENTES DE CONTEINERIZAÇÃO COMPARTILHADOS

# IMPACTS OF ACCESS CONTROL ON THE SECURITY OF SHARED CONTAINERIZATION ENVIRONMENTS

Raul Beserra de Brito, Faculdade de Tecnologia de Americana – Ministro Ralph Biasi, raul.brito@fatec.sp.gov.br

Dra. Maria Cristina Aranda, Faculdade de Tecnologia de Americana – Ministro Ralph Biasi, mcris.aranda@fatec.sp.gov.br

**Resumo**

O objetivo deste trabalho foi analisar a eficácia de controle de acesso baseados em funções dentro de ambientes Kubernetes, com uso de contêineres Docker compartilhados. Para isso, aprofundou-se em temas, como virtualizações, controles de acessos e em diversas tecnologias presentes em grandes projetos comerciais. A metodologia utilizada foi de pesquisa bibliográfica, com abordagem qualitativa, para trazer base teórica de sustentação ao tema, com as mais diversas informações e contexto de aplicabilidade, além de recriar, na prática, em laboratórios, alguns casos de usos destes tipos de controle de acesso. Desta forma, o presente trabalho pode contribuir para o impulso na área de segurança dentro destes ambientes, trazendo com base os três principais pilares da segurança da informação: confidencialidade, integridade e disponibilidade. Concluiu-se que, apesar do controle de acesso baseado em funções ser uma das principais soluções de gerenciamento de permissões, com inúmeras formas de personalização, cada projeto possui sua demanda específica que deve ser tratada objetivamente com seu modelo mais adequado.

**Palavras-chave**: Virtualização; controle de acesso; conteinerização.

***Abstract***

*The aim of this work was to analyze the effectiveness of role-based access control within Kubernetes environments, using shared Docker containers. To do this, it delved into topics such as virtualization, access control and various technologies present in large commercial projects. The methodology used was bibliographical research, with a qualitative approach, to provide a theoretical basis to support the topic, with the most diverse information and context of applicability, in addition to recreating, in practice, in laboratories, some use cases of these types of access control. In this way, this work can contribute to boosting security in these environments, based on the three main*

*pillars of information security: confidentiality, integrity and availability. It was concluded that, although role-based access control is one of the main permissions management solutions, with countless ways of customizing it, each project has its own specific demands which must be dealt with objectively using the most appropriate model.*

***Keywords**: Virtualization; access control; containerization.*

## 1. Introdução

Com a evolução da tecnologia e o aumento na diversificação de ambientes operacionais, surgiu a necessidade de criar aplicações isoladas. Por conseguinte, as primeiras soluções desenvolvidas com este objetivo foram as Máquinas Virtuais (MVs, do inglês, *Virtual Machines*, VMs), que tiveram um papel importante nas primeiras etapas da virtualização. Todavia, com as limitações das VMs, geradas pelo grande consumo de recursos e pela ineficiência no desempenho, foi necessário desenvolver uma solução simplificada deste ambiente. Como resposta a esta necessidade, surgiu o modelo de conteinerização, no qual tornou a portabilidade de aplicações mais eficientes, além de melhorar o isolamento completo de um ambiente lógico dentro de um mesmo *kernel*.

A posteriori, foi desenvolvida a principal ferramenta de isolamento e gerenciamento de contêineres, conhecida como Docker. Segundo Susnjara e Smalley (2024), o Docker é uma plataforma de código aberto (qualquer pessoa pode contribuir em seu desenvolvimento) que permite aos desenvolvedores construir, implementar, executar, atualizar e gerenciar os contêineres.

Esta evolução desencadeou inúmeros ambientes de projetos compartilhados, onde equipes utilizam da mesma infraestrutura de contêineres, com intuito de economizar o máximo de recurso disponível. Segundo Myrbakken e Colomo-Palacios (2017), o uso de boas práticas de segurança nestes tipos de ambientes, apesar de ser um desafio, podem gerar grandes benefícios, como melhorarias na eficiência entre desenvolvedores e nas entregas.

No entanto, com o aumento de ambientes compartilhados, surgiram novos desafios relacionados à segurança, principalmente na área de gerenciamento de acesso, que se relaciona com os três principais pilares da segurança da informação: a confidencialidade, que divide permissões sobre ações; a integridade, no qual as aplicações não podem ser alteradas indevidamente; e a disponibilidade, garantindo entrega contínua dos serviços, com uso de políticas de segurança. Neste cenário, a Segurança da Informação desempenha um papel essencial, garantindo a proteção dos dados, com uso de políticas e normas.

Uma das abordagens existentes para mitigação de riscos é o controle de acesso baseado em função (CABF, do inglês *Role-Based Access Control*, RBAC), que gerencia as permissões específicas de acesso a recursos e ações. Ou seja, cada usuário é atribuído a uma ou mais funções específicas, nas quais cada uma difere em permissões. Apesar de ser um modelo facilitador em organizações, tem limitações, enquanto a sensibilidade do atributo do usuário, impossibilitando diferenciar quaisquer anomalias dentro de cada função (IBM Technology, 2024).

E, para facilitar a orquestração de grandes quantidades de contêineres e implementação de controles de acessos, surgiu o Kubernetes, uma ferramenta que se destaca no gerenciamento eficaz de recurso distribuído, além de proporcionar controle detalhado sobre permissões de acesso. Mustyala e Tatineni (2021) destaca que, em ambientes que demandam políticas avançadas de controle de acesso, granularidade em autorização, tornam-no essencial. Desta forma, ao combinar a robustez do Kubernetes junto à portabilidade do Docker, espera-se gerar uma solução segura, escalável e flexível.

A partir dessa explanação, este artigo aborda a seguinte questão: o RBAC supre a demanda de segurança entre os diferentes tipos de acesso de controles em ambientes compartilhados de contêineres?

Com base neste questionamento, levanta-se a seguinte hipótese: a

implementação do RBAC, embora eficaz, necessita de personalização para maior segurança.

Sendo assim, o objetivo geral deste trabalho é verificar a eficácia de controles de acesso, utilizando o modelo de RBAC em ambientes Kubernetes, com contêineres Docker compartilhados, com finalidade de identificar qual abordagem oferece maior segurança. Com esta proposta, foram definidos os seguintes objetivos específicos:

- analisar a eficácia do RBAC em ambientes Docker e Kubernetes compartilhados;
- comparar os modelos de controle de acesso;
- avaliar a implementação do modelo de RBAC.

A justificativa deste estudo é devido ao crescente uso de ambientes de contêineres com o uso de Docker e de Kubernetes, que demandam soluções de segurança eficientes. Assim, a presente pesquisa busca compreender, avaliar e contribuir para a proteção de dados e serviços nestes tipos de ambientes.

## 2. Referencial Teórico

A partir do que foi apresentado na introdução, pode-se observar a necessidade de compreender melhor os desafios no contexto de conteinerização e de orquestração, portanto é necessário explorar os termos e as tecnologias profundamente. Estas necessidades serão supridas nas secções abaixo.

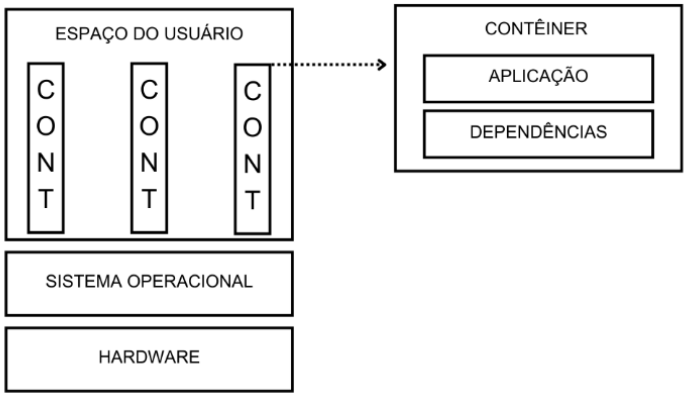
### 2.1. Virtualização

No geral, a virtualização pode ser dívidas em dois tipos: baseada em contêiner e baseada em hipervisores. Desta forma, uma provê à nível de sistema operacional, enquanto a outra à nível de *hardware*, nas quais cada uma possui suas próprias vantagens e desvantagens.

De acordo com Bui (2015), a virtualização baseada em contêineres é a mais leve, na qual se compartilha o *kernel* do host para rodar inúmeras aplicações, baseadas nas seguintes tecnologias: Linux-VServer, Linux Container (LXC) e o OpenVZ. Esses contêineres funcionam de maneira independente, com processos e recursos isolados, tornando-os mais eficiente.

Observa-se o funcionamento da arquitetura de virtualização baseada em contêiner na figura abaixo:

Figura 1 – Diagrama do modelo de virtualização baseado em contêineres

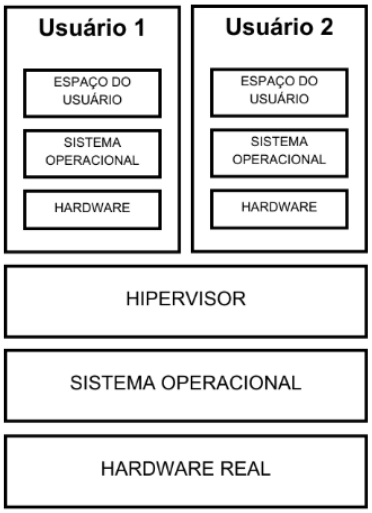



Fonte: Baseado em Bui (2015)

Já a visão baseada em hipervisores simula o *hardware* completo, desde componentes, sistemas operacionais e aplicações, tornando-a mais pesada, conhecidos principalmente como Máquinas Virtuais. Entretanto, possui o objetivo comum de qualquer virtualização: ser particular. Os hipervisores como KVM, VMWare e VirtualBox são alguns exemplos para esta abordagem comumente vistas em centros de dados e ambientes corporativos, onde há a necessidade de rodar ambientes operacionais, mantendo um alto isolamento computacional (Groesbrink *et al.*, 2014).

Esta arquitetura pode ser observada na Figura 2:

Figura 2 – Diagrama do modelo de virtualização baseado em hipervisor




Fonte: Baseado em Eder (2016)

Em contexto de projetos de ambientes distribuídos, que demandam maior portabilidade, eficiência e rapidez, o ambiente baseado em contêineres se destacou, já que pode manter uma aplicação rodando na menor quantidade de recursos possíveis, de maneira rápida e gerenciável (Merkel, 2014).

Segundo Burns *et al.* (2016), com este tipo de tecnologia, obteve-se maior facilidade no ciclo, na produção e nas entregas de serviços, uma vez que os contêineres possuem um tempo de vida (TDV, do inglês *Time to Live*, TTL), ou seja, são programados para inicializarem, rodarem as aplicações e, caso ocorra algum problema, serem finalizados, mantendo uma redundância. Ao contrário dos ambientes baseados em hipervisores, que tem como principal objetivo manter a maior disponibilidade possível.

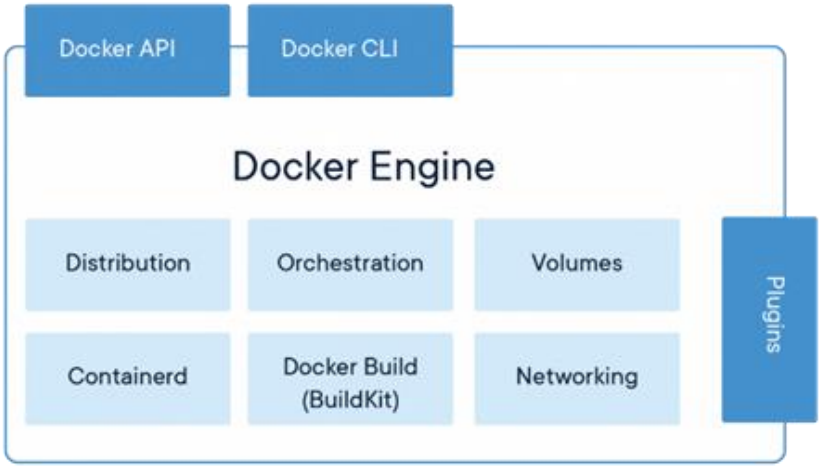
Este modelo não só permite a escalabilidade e rápida inicialização, mas também múltiplas execuções de serviços sem interferência mútua.

## 2.2. Contêineres e o papel do Docker

Os contêineres são abstrações na camada de aplicação que empacotam o *software*, junto a todas suas dependências básicas, com o objetivo de ser executado de forma rápida e confiável, independente do ambiente computacional. Logo estes ambientes ocupam menor espaço, ao comparar com as VMs. Uma das suas principais funções é o uso do *kernel* compartilhado, no qual impulsionou os conceitos primitivos do Linux, como o cgroups — responsável em limitar recursos em processos — e os namespaces — encarregado por isolar grupos de processo em nível lógico — (Docker, [s.d.]).

Com as inúmeras formas de isolamento de aplicação, não se tinha padronização no mercado. Posto isso, o Docker respondeu a esta necessidade desenvolvendo o Docker Engine, iniciando um movimento de padronização de contêineres, no qual trouxe um empacotamento universal que agrupa todas as dependências, nas quais são executadas dentro de seu próprio ambiente, como é demonstrado na Figura 3.

Figura 3 – Diagrama do funcionamento do Docker Engine




Fonte: Docker (2024)

Com este tipo de empacotamento, um contêiner Docker oferece suporte a qualquer tipo de aplicativo: monolítico, nativo à nuvem, legado etc., além de ser validado para funcionar com a Interface de Tempo de Execução de Contêiner do Kubernetes (TEC, do inglês *Container Runtime Interface,* CRI), uma ferramenta que padroniza a comunicação entre o Kubernetes e diferentes *runtimes* de contêineres (Docker, 2024).

## 2.3. Kubernetes e a Orquestração de Contêineres

O Kubernetes (K8S ou KUBE), criado pela Google, é uma plataforma de orquestração de contêineres que tem como função automatizar o *deploy*, a escalabilidade, a aplicação e gerenciar estes ambientes, com o uso de *clusters*, que podem ser divididos em duas partes fundamentais: o plano de controle, olhando para comunicação entre serviços, e as de instâncias (ou nós) de computação, que garante a resposta definida às ordens recebidas. Estes nós, por sua vez, possuem seus próprios ambientes Linux, podendo ser tanto físicos, quanto virtuais, nos quais possibilita executar seus contêineres de forma isolada e eficiente (Red Hat, 2020).

Segundo Albuquerque Filho e Gama ([s.d.]), o KUBE trabalha sem uma hierarquia fixa. Desta forma, cada instância executa trabalhos individuai, possibilitando um ambiente distribuído. Outro aspecto importante deste ecossistema é o *Node Controller*, cujos objetivos são monitorá-los, além de os deixarem em sincronia com as máquinas. Ademais existe o menor e mais simples objeto dentro do ecossistema do K8S, conhecido como *pod*, que é referenciado por ser um grupo de um ou mais contêineres que compartilham opções de gerenciamento. Com isso, é possível criar um grupo que compartilha a mesma faixa Protocolos de Internet (PI, do inglês *Internet Protocols*, IPs) e espaços de portas únicos, facilitando na gestão de aplicações.

Ao entender a necessidade do K8S para o gerenciamento de ambientes já

retratados, torna-se explícito a importância da orquestração de contêineres. Além disso, sua outra grande contribuição é na proteção e na integridade dos recursos utilizados, seja na infraestrutura ou na cadeia de suprimento. Ou seja, segundo Red Hat (2020), o Kubernetes permite aprimorar a segurança de TI, ao oferecer controle amplo sobre as segmentações, as políticas de segurança e as permissões de proteção do ambiente geral.

Logo, este conjunto de recursos contribuem diretamente para a integridade dos serviços, diminuindo as vulnerabilidades e assegurando maior confiabilidade nos processos.

## 2.4. Regras de Acesso e Controle de Segurança

As regras de acesso são uma das práticas fundamentais da segurança da informação, independentemente de onde sejam aplicadas. Seu propósito concentra-se em restringir ações de pessoas sem autorização. É um dos meios mais comuns para proteger sistemas ou infraestruturas físicas contra acessos indevidos, garantindo integridade do que restringe acesso.

Ao levar em consideração o contexto do KUBE, existem dois controles de acesso amplamente discutido: o "Controle de Acesso Baseado em Função" (CABF, do inglês *Role-Based Access Control*, RBAC) e o "Controle de Acesso Baseado em Atributos" (ABA, do inglês *Attribute-Based Access Control*, ABAC).

No modelo RBAC, as permissões de acesso são atribuídas em grupos, permitindo um uma administração mais eficiente e centralizada, especialmente em ambientes complexo, no qual há grande rotação de pessoas físicas. As permissões são agrupadas em objetos chamados de *Roles* ou *ClusterRoles*, que podem ser vinculados aos grupos ou aos usuários por meio do *RoleBiding* ou *ClusterRoleBiding*, que é a forma em que o K8S vincula os usuários e as aplicações em suas devidas funções criadas dentro ambiente trabalhado (Kubernetes, [s.d.]).

Ferraiolo, Cugini e Kuhn (1995) destacam o RBAC como uma solução cuja finalidade é simplificar o gerenciamento de acesso ao organizar as permissões pelas ocupações dos usuários, trazendo tudo que é necessário para uma boa segurança em ambientes de contêineres.

Por outro lado, o ABAC adota uma abordagem mais granular e detalhada, utilizando atributos específicos, com localização, horário, identidade entre outros. No Kubernetes, este modelo de regra de acesso é configurado através de políticas escritas em arquivos JavaScript Object Notation (JSON), no qual é uma formatação utilizada em estruturas de dados, onde cada linha representa uma regra que combina atributos de permissões (Kubernetes, [s.d.]).

Ao levar em consideração a ferramenta Kubernetes, Rostami (2023) discute o uso do ABAC para autorização, retratando que, embora este modelo possua maior granularidade, apresenta alta complexidade em sua aplicação, principalmente na definição e atualização de políticas. Por outro lado, ao comparar com RBAC, mesmo sem sua dificuldade, sua hierarquia simplificada pode atender a inúmeros requisitos de ambientes e projetos.

Além disso, a aplicabilidade deste meio de segurança é um ponto forte, no qual ajuda tanto empresas pequenas, que não possuem mão de obra especializada, quanto as grandes, no quesito do uso desta solução com pouca curva de aprendizado.

No aspecto da segurança da informação, a aplicação deste conceito destaca-se em diversos pilares:

- confidencialidade: com uso de restrições mínimas para usuário ou aplicação.
- integridade: ao restringir acesso a modificações, com objetivo de resguardar qualquer informação presentes dentro de *pods* ou

contêineres.

- disponibilidade: ao prover, mesmo indiretamente, orquestrações substanciais para o uso de aplicações e microsserviços presentes nos ambientes operacionais.

## 3. Materiais e Métodos

Em complemento à revisão, foi desenvolvido um laboratório experimental para validar os conceitos estudados. Utilizando como ambiente principal o WSL2, no qual é um ambiente Linux interno do Windows, o Kubernetes Kind, para criação de *clusters*, *pods* e orquestração e, por fim, o Docker. O laboratório permitiu mensurar o impacto de diferentes configurações de RBAC, possibilitando o entendimento do desempenho na segurança.

### 3.1. Requisitos

A ambientação da infraestrutura simulada foi baseada em um ambiente virtualizado utilizando o WSL2, por conta da sua simplicidade de integração e desempenho. Neste caso, optou-se pelo o Ubuntu 22.04 LTS, devido a sua estabilidade e suporte a tecnologias recentes.

Como ferramentas para aplicar os modelos de RBAC, foi utilizado o Docker, na versão 27.3.1, e o Kubernetes Kind, na versão 0.25.0, no qual emula um ambiente completo dentro de um contêiner.

### 3.2. Tokens para Service Accounts

Criou-se uma *Service Accounts* (SA), um recurso do Kubernetes usado para identificar uma aplicação dentro do *cluster*. Estas aplicações normalmente utilizam autenticação com a interface de programação de aplicativos (IPA, do inglês *Application Programming Interface*, API) do Kubernetes, sem o uso de credenciais

genéricas ou de usuários.

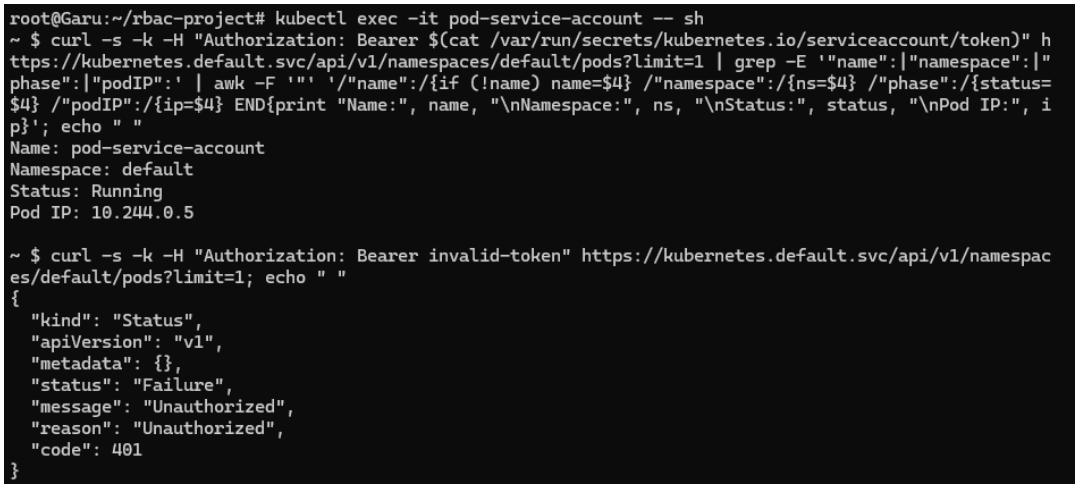
Após a criação da SA e a sua aplicação no *cluster*, foi gerado um *token* de autentificação, que foi armazenado em uma *secret* relacionada a SA, na qual é representada por um recurso de armazenamento seguro.

E, para restringir suas ações, foi configurada uma *role*, na qual é um conjunto de permissões dentro de um *namespace*. Neste caso, configurou-se apenas para ter acesso a listar e visualizar os *pods* no *namespace* padrão. Desta forma, assegurou-se que não pudesse criar, modificar ou excluir recursos alheios.

Para vincular esta *role* criada ao *namespace*, criou-se uma *RoleBinding*, que é um recurso de vínculo entre um usuário ou um grupo. Neste caso, utilizou para conectar a *role* ao SA, garantido os acessos configurados.

No exemplo demostrado na Figura 5, o *pod* criado foi acessado, e, internamente, duas execuções foram executadas com o objetivo de obter informação sobre o primeiro pod dentro do *namespace default*. A primeira teve sucesso, pois utilizou o *token* fornecido pelo *service account*, já na outra, sem utilizar um *token*, retornou erro de autorização.

Figura 5 – Gerenciamento de permissões na API do Kubernetes com o uso de *tokens*



Fonte: Elaborada pelos autores (2024)

### 3.3. Autenticação por certificado

Diferente do *token* para SA, é criado um certificado utilizando o OpenSSL, uma das principais bibliotecas de criptografia. Com o certificado gerado para o devido usuário, é enviado para o Kubernetes, onde será aprovado internamente no aplicativo.

Após aprovar o certificado do usuário, cria-se uma *role* junto a um *namespace*, que, basicamente, engloba o que os usuários poderão fazer dentro do *cluster*.

Em seguida, cria-se uma *RoleBinding* para conectar as *roles* com os usuários, no qual se se certificará de agregar as respectivas permissões pré-configuradas.

Por fim, deve adicionar o certificado do usuário no kubeconfig. Este arquivo inclui todas as informações necessárias para requisição, seja tempo de expiração, chave privada, endereço do servidor entre outros. Assegurando maior integridade na hora da autenticação na área de desenvolvimento.

Com isso, conseguimos verificar, pela Figura 6, o gerenciamento de permissão do contexto *developer* criado, no qual está limitado apenas aos *pods*, podendo visualizar, acompanhar, atualizar, listar entre outros. Como exemplo, criou-se e executou um pod chamado "nginx" dentro do *namespace "dev"*, contendo apenas um único contêiner baseado na imagem do nginx, um servidor *web* orientado a eventos e assíncrono, e, por fim, listado os pods em execução. Em outro aspecto, apenas a tentativa de listar os recursos de *deployments*, foi barrada, pois a *role* não possui acesso ao grupo de API *apps* no *namespace dev*.

Figura 6 – Demonstração do contexto de *developer* criado

```
root@Garu:~/rbac# kubectl config use-context developer
Switched to context "developer".
root@Garu:~/rbac# kubectl run nginx --image=nginx -n dev
pod/nginx created
root@Garu:~/rbac# kubectl get pods
NAME    READY   STATUS    RESTARTS   AGE
nginx   1/1     Running   0          14s
root@Garu:~/rbac# kubectl get deployments
Error from server (Forbidden): deployments.apps is forbidden: User "developer" cannot
list resource "deployments" in API group "apps" in the namespace "dev"
```

Fonte: Elaborada pelos autores (2024)

## 4. Resultados e Discussões

A implementação do modelo RBAC em ambientes Kubernetes e Docker compartilhados mostrou-se eficaz. Garantiu controle centralizado e simplificado para gerenciamento de permissões de acesso. Com isso, possibilitou:

- isolamento de recurso;
- minimização de acesso;
- separação entre aplicação e usuário.

Ao analisar a base teórica, percebe-se que as políticas criadas dentro do escopo de RBAC são altamente escaláveis e auditáveis, o que simplifica o gerenciamento em ambientes com inúmeros usuários e aplicações, melhorando diretamente a resposta em qualquer tipo de incidente que demanda fazer alguma alteração.

Em consideração a outros modelos de acessos de controle, o principal tipo de controle de acesso é o RBAC, no qual foi confirmado, levando em consideração a padronização e centralização das permissões. Além da facilidade na aplicação e modificação de políticas de segurança.

Neste papel, provou-se resiliente a mudanças, mesmo em ambientes

complexo e dinâmicos.

Ao analisar ambientes que demanda maior segurança e granularidade, o RBAC pode não ser a melhor escolha, visto que não possui a elasticidade para anomalias dentro de um sistema.

Ou seja, em ambientes complexo e estruturados, pode ser a melhor escolha. Já em ambientes que não se tem uma base fixa, pode ser sujeito a falhas, já que está tudo acoplado a grupos.

Reforçando que, embora seja uma solução prática, pode não ser o ideal para ambientes com contextos em construção. Neste caso, escolhas como ABAC pode ser ideal, visto que oferece maior flexibilidade ao trabalha com políticas que engloba: horário, geolocalização, IP e, até mesmo, contexto de ambiente.

## 5. Considerações Finais

Este artigo teve como objetivo é verificar, por meio de uma metodologia explicativa, com abordagens qualitativa, junto a aplicação em laboratórios, a eficácia de controles de acessos, com base no RBAC, em ambientes Kubernetes, com contêineres Docker compartilhados.

Por meio deste estudo, pode-se observar os impactos positivos de controles de acessos em diversos ambientes, principalmente quando se há o uso de contêiners.

Ao validar os pontos estudados, entendeu-se que o uso do RBAC supre a grande demanda do mercado, mas não todas. O uso de outros tipos de controles de acessos, como ABAC, pode ter ganhos positivos em determinados contextos.

Cada ambiente possui seu contexto, no qual o desenvolvedor principal de infraestrutura lógica não se deve replicar um único conceito de acesso de controle. Desta forma, conclui-se que a hipótese "a implementação do RBAC, embora eficaz,

necessita de personalização para maior segurança" é falsa, pois o RBAC, em si, possui uma gama de configurações presentes, nas quais podem se encaixar perfeitamente em um determinado contexto. Todavia, quando se necessita de grandes mudanças das quais não fazem parte do escopo deste tipo controle, perde-se desempenho no projeto.

Embora o escopo do trabalho aborda apenas um tema específico da segurança, é importante lembrar que existem outros pontos cruciais nesta área.

Embora o RBAC atenda o que se propõem a fazer, é primordial reconhecer a necessidade de ter políticas organizacionais robustas, programas de conscientização, auditorias regulares e uma gestão proativa para proteção de informações. Apenas com o uso combinado destas práticas é possível minimizar, ao máximo, os riscos.

## Referências

ALBUQUERQUE FILHO, A.; GAMA, K. **Estudo comparativo entre Docker Swarm e Kubernetes para orquestração de contêineres em arquiteturas de software com microsserviços**. [S.l.: s.d]. Disponível em: https://www.cin.ufpe.br/~tg/2016-2/acaf.pdf. Acessado em: 27 out. 2024.

BUI, Thanh. **Analysis of Docker Security**. Aalto University School of Science, 2014. Disponível em: https://arxiv.org/abs/1501.02967. Acesso em: 1 nov. 2024.

BURNS, B. et al. Borg, omega, and kubernetes. **Communications of the ACM**, v. 59, n. 5, p. 50–57, 2016. Disponível em: https://dl.acm.org/doi/fullHtml/10.1145/2890784. Acesso em: 24 out. 2024.

DOCKER. **Container runtime**. Docker, 2021. Disponível em: https://www.docker.com/products/container-runtime/. Acesso em: 04 nov. 2024.

DOCKER. **What is a container?**. Docker, 2024. Disponível em: https://www.docker.com/resources/what-container/. Acesso em: 04 nov. 2024.

EDER, M. Hypervisor-vs. container-based virtualization. **Future Internet (FI) and Innovative Internet Technologies and Mobile Communications (IITM)**, v. 1, 2016. Disponível em: https://www.net.in.tum.de/fileadmin/TUM/NET/NET-2016-07-1/NET-2016-07-1_01.pdf. Acesso em: 1 nov. 2024.

FERRAIOLO, D. et al. Role-based access control (RBAC): features and motivations. In: **Proceedings of 11th annual computer security application conference**. 1995. p. 241-48. Disponível em: https://orbit-lab.org/raw-attachment/wiki/Internal/Rbac/RbacResources/ferraiolo-cugini-kuhn-95.pdf. Acesso em: 2 nov. 2024.

GROESBRINK, S. et al. **Towards certifiable adaptive reservations for hypervisor-based virtualization**. Porto: CISTER Research Unit, Instituto Politécnico do Porto, 2014. Technical Report CISTER-TR-140304. Disponível em: http://www.cister.isep.ipp.pt/docs/towards_certifiable_adaptive_reservations_for_hypervisor_based_virtualization/833/view.pdf. Acesso em: 1 nov. 2024.

IBM Technology. **Role-based access control (RBAC) vs. Attribute-based access control (ABAC)**. YouTube, 14 jun. 2023. 7min38s. Disponível em: https://www.youtube.com/watch?v=rvZ35YW4t5k. Acesso em: 24 out. 2024.

KUBERNETES. **Using RBAC authorization**. Disponível em: https://kubernetes.io/docs/reference/access-authn-authz/rbac/. Acesso em: 8 nov. 2024.

KUBERNETES. **Utilizando autorização ABAC**. Disponível em: https://kubernetes.io/pt-br/docs/reference/access-authn-authz/abac/. Acesso em: 8 nov. 2024.

MERKEL, D. Docker: lightweight Linux containers for consistent development and deployment. **Linux Journal**, v. 2014, n. 239, p. 2, 2014. Disponível em: https://www.seltzer.com/margo/teaching/CS508.19/papers/merkel14.pdf. Acesso em: 5 nov. 2024.

MUSTYALA, Anirudh; TATINENI, Sumanth. Advanced security mechanisms in Kubernetes: isolation and access control strategies. **ESP Journal of Engineering & Technology Advancements** (ESP JETA), v. 1, n. 2, p. 57-68, 202. Disponível em: https://www.espjeta.org/Volume1-Issue2/JETA-V1I2P109.pdf. Acesso em: 24 out. 2024.

MYRBAKKEN, H.; COLOMO-PALACIOS, R. DevSecOps: a multivocal literature

review. In: **Software Process Improvement and Capability Determination**: **17th International Conference,** SPICE 2017, Palma de Mallorca, Spain, October 4–5, 2017, Proceedings. Springer International Publishing, 2017. p. 17-29. Disponível em: http://rcolomo.com/papers/314.pdf. Acesso em: 24 out. 2024.

RED HAT. **Introduction to Kubernetes architecture**. Red Hat, 2020. Disponível em: https://www.redhat.com/en/topics/containers/kubernetes-architecture. Acesso em: 5 nov. 2024.

RED HAT. **What is Kubernetes?**. Red Hat, 2020. Disponível em: https://www.redhat.com/pt-br/topics/containers/what-is-kubernetes. Acesso em: 6 nov. 2024.

ROSTAMI, G. Role-based access control (RBAC): authorization in Kubernetes. **Journal of ICT Standardization**, v. 11, n. 3, p. 237-260, 2023. Disponível em: https://doi.org/10.13052/jicts2245-800X.1132. Acesso em: 24 out. 2024.

SUSNJARA, S.; SMALLEY, I. **Docker**, 2024. Disponível em: https://www.ibm.com/topics/docker. Acesso em: 24 out. 2024.