Impacto do Uso da Plataforma de Segurança Prisma Cloud no Desempenho de Contentores do Docker

(Versão Definitiva Após Defesa Pública)

Kavadiambuko Maleka Lutandila

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática
(2º ciclo de estudos)

Orientador: Prof. Doutor Mário Marques Freire

Covilhã, Agosto de 2020

Dissertação elaborada no Instituto de Telecomunicações - Delegação da Covilhã e no Departamento de Informática da Universidade da Beira Interior e submetida à Universidade da Beira Interior para discussão em provas públicas.

Este trabalho foi financiado pela FCT/MCTES através de fundos nacionais e quando aplicável cofinanciado por fundos comunitários no âmbito do projeto UIDB/EEA/50008/2020 e foi suportado pela operação Centro-01-0145-FEDER-000019 - C4 - Centro de Competências em Cloud Computing, cofinanciada pelo Fundo Europeu de Desenvolvimento Regional (FEDER) através do Programa Operacional Regional do Centro (Centro 2020), no âmbito do Sistema de Apoio à Investigação Científica e Tecnológica - Programas Integrados de IC&DT.

Cofinanciado por:

Dedicatória

Dedico esta dissertação às minhas filhas, Carmina Nunes Maleka e Alina Maria Nunes Maleka, especialmente aos meus pais Armando Maleka e Alina Luntadila Nkoxi por todo apoio e força que me deram durante a minha formação.

Agradecimentos

Primordialmente a Allah, por ser essencial na minha vida, autor do meu destino, o meu guia, socorro presente na hora da angústia. Ao meu tio Guy Laredo Mayasi e a sua esposa Nicole Ngadi Kisibu, Liliana Makangilu Fulama, Nsamba Landu Vambanu, a todos os meus colegas e amigos Luzizila Salambiaku, Conceição João Apolinário, Luzayamo Makiese Mavakala, Joaquim Mpaka Nkudi, Nanouche Mafuala Adrienne, Nsiamfumu Kunzayila e toda a minha família pela ajuda e apoio nos meus projetos inclusive entendendo a minha ausência.

Agradecimento especial, ao meu orientador, Professor Dr. Mário Marques Freire pelo apoio, competência, interesse e contribuições no processo de orientação que se desenrolou em clima de profundo respeito e sem esquecer a Direção da Universidade Kimpa Vita 7ª Região Académica particularmente a Escola Superior Politécnica do Uíge/Angola, representada pela PhD. Maria de Fátima e a todos os Docentes do Departamento de Informática da Faculdade de Engenharia da UBI.

Resumo

A cada dia a popularização da tecnologia Docker torna-se mais evidente, em conjunto com todas as suas dependências juntas na forma de contentores. Esta abordagem permite incorporar, executar ou distribuir várias aplicações ou objetos (contentores, imagens, *hosts*), com menor overhead em qualquer ambiente de produção em nuvem. A virtualização de contentores do Docker mostra a necessidades de várias técnicas apropriadas para análise automatica de vulnerabilidades para garantir a segurança da infraestrutura, mesmo que atualmente os contentores sejam um dos recursos com melhor desempenho da tecnologia Docker. Por outro lado, o Twistlock, que foi integrado no Prisma Cloud em 2019, sendo uma plataforma de segurança nativa na nuvem, fornece as ferramentas necessárias para plataformas em nuvem com descrição das políticas de segurança bem definidas como o controlo de acessos a cargas de trabalho em contentores do Docker, permitindo apresentar auditorias em tempo real de todas atividades do sistema num único painel de controlo. O Twistlock utiliza gestão de vulnerabilidades para prevenir vulnerabilidades em todo o ciclo de vida das aplicações, de modo a garantir a conformidade em todo processo e possibilitar aprendizagem automática da topologia da rede com uma firewall nas camadas 4 e 7 para inspecionar todas as conexões autorizados e não autorizados da rede.

Nesta dissertação pretende-se avaliar o impacto no desempenho de contentores do Docker resultante do uso da plataforma Twistlock/Prisma Cloud, utilizando a plataforma Sumo Logic para monitorar o desempenho dos contentores do Docker em termos de consumo de memória, CPU e rede. O Sumo Logic foi instalado e configurado em duas máquinas físicas e numa máquina virtual (*Cluster*) com dois nós do *Swarm*.

Para analisar o desempenho foram recolhidos os valores das métricas de desempenho do Sumo Logic para cada teste. Os resultados obtidos mostram que os contentores do Docker sem Twistlock alcançaram uma taxa de transmissão de 14,8 Mb/s para tráfego enviado e 12,5 Mb/s para tráfego de recebido da rede, enquanto para contentores do Docker com Twistlock foram alcançadas taxas de 15 Mb/s para tráfego enviado e 13 Mb/s para tráfego recebido. Em relação ao consumo total de CPU ao longo do tempo em memória RSS (Resident Set Size), por contentor, observou-se que os valores desta métrica são semelhante com e sem Twistlock para contentores com cargas de trabalho idênticas, variando entre 1,0% e 8,32% para os contentores considerados nas experiências. Contudo, o uso de CPU por contentor durante a execução de uma carga de trabalho é superior para contentores com Twistlock, variando entre 1,20% e 3,85% para os contentores com Twistlock considerados nas experiências e entre 0,10% e 2,83% para os mesmos contentores mas sem Twistlock. Relativamente ao uso de CPU do *kernel* por contentor, observou-se uma maior homogeneidade para contentores sem Twistlock, variando entre 5,80% e 5,89% em relação a contentores com Twistlock, em que o uso de CPU do *kernel* por contentor variou entre 1,0% e 6,85%. Sobre o consumo de memória, observou-se que os contentores sem Twistlock considerados nos testes usaram, em média, 270 Kb/s de memória RSS, en-

quanto os contentores com Twistlock considerados nos testes usaram, em média, 40 Kb/s de memória RSS.

Os resultados apresentados mostram que o Twistlock, embora tenha impacto, não degrada significativamente o desempenho de contentores do Docker, mesmo quando é realizada uma análise profunda de vulnerabilidades e conformidades.

Palavras-chave

Twistlock, Prisma Cloud, Docker, Desempenho, contentores, Virtualização ao nível do sistema operativo.

Abstract

Every day the popularity of Docker technology becomes more relevant, together with all its dependencies in the form of containers. This approach allows to incorporate, run and distribute multiple applications or objects (containers, images, hosts), with less overhead in any cloud production environment. Container virtualization in Docker, shows the need for several suitable techniques for automated vulnerability analysis to ensure the security of the infrastructure, even though today containers are one of the best performing resources of Docker technology. On the other hand, being a native cloud security framework, Twistlock, integrated in Prisma Cloud in 2019, provides the necessary tools for cloud platforms with well-defined security policies such as access control or for workloads in Docker containers, allowing the presentation of a timely audit of all system activities in one dashboard. Twistlock uses vulnerability management to prevent vulnerabilities throughout the application lifecyle, to ensure compliance with the entire process and to allow automated learning of network topology with a firewall in layers 4 and 7 to inspect all authorized and unauthorized network connections.

This dissertation intends to evaluate the impact on the performance of Docker containers resulting from the use of the Twistlock/Prisma Cloud platform using the Sumo Logic platform to monitor the performance of Docker containers regarding memory, CPU and network usage. Sumo Logic was installed and configured in two physical machines and in a virtual machine (Cluster) with two Swarm nodes.

To analyze the performance, the values of the Sumo Logic performance metrics were collected for each test. The obtained results show that Docker containers without Twistlock reached a transmission rate of 14.8 Mb/s for outgoing traffic and 12.5 Mb/s for incoming traffic, while for Docker containers with Twistlock, the transmission rate achieved 15 Mb/s for outgoing traffic and 13 Mb/s for incoming traffic. Regarding the total CPU consumption over time in RSS (Resident Set Size) memory, per container, it was observed that the values of this metric are similar with and without Twistlock for containers with identical workloads, varying between 1.0 % and 8.32 % for the containers considered in the experiments. However, the CPU usage per container during the execution of a workload is higher for Twistlock containers, varying between 1.20 % and 3.85 % for Twistlock containers considered in the experiments and between 0.10 % and 2.83 % for the same containers but without Twistlock. Regarding the use of the *kernel* CPU by container, a larger homogeneity was observed for containers without Twistlock, varying between 5.80 % and 5.89 %, in relation to containers with Twistlock, in which the use of CPU of the *kernel* per container ranged from 1.0 % to 6.85 %. Regarding memory consumption, it was observed that the containers without Twistlock considered in the tests used, on average, 270 Kb/s of RSS memory, while the containers with Twistlock considered in the tests used, on average, 40 Kb/s of RSS memory.

The presented results show that Twistlock, although having an impact, does not significantly degrade the performance of Docker containers, even when a thorough analysis of

vulnerabilities and compliance is carried out.

Keywords

Twistlock, Prisma Cloud, Docker, Performance, Containers, Operating system level virtualization.

Índice

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos

API Application Programming Interface

AUFS Another Union File System

AWS Amazon Web Services

BS British Standard

BSD Berkeley Software Distribution

BTRFS B-tree File System

CD Continous Delivery

CGROUPS Control Groups

CIA Confidenttality Integrity and Availability

CIS Center for Internet Security

CLI Comand-Line Interface

CMS Conversational Monitor System

CNAF Cloud Native Application Firewall

CNNF Cloud Native Network Firewall

CPU Center Processing Unit

CVE Common Vulnerabilities and Exposures

CVSS Common Vulnerability Scoring System

DeVOps Development Operations

DoS Denial of Service

E/S Entra e Saída

GUI Graphical User Interface

IAAS Infrastruture as a Service

IAM identity and Access Management

IC Integration Continuous

ID Identity

IEC International Electrotechnical Commission

IP Internet Protocol Address

ISO International Organization for Standardization

KB Kilo Bytes

LMCTFY Let Me Contain That For You

LTS Long Term Support

LXC Linux Containers

MAC Medium Access Control

MB Mega Bytes

NAT Network Address Translation

NSX Network Virtualization

OS Operating System

OSI Open Systems Interconnection

PAAS Platform as a Service

RAM Random Access Memory

RBAC Role-Based Access Control

REST Representational State Transfer

RSS Resident Set Size

SecOps Security and Operations

SIEM Security Information and Event Management

SOC Security Operating Center

TI Information Technology

UID User Identification

VE Virtal Execution Environment

VLAN Virtual Local Areal Network

VMS Virtual Machines

VoIP Voice over Internet Protocol

VPS Virtual Private System

Capítulo 1

Introdução

.

1.1 Âmbito e Foco da Dissertação

A tecnologia de contentores do Docker [Doc20d] é atualmente muito popular. De acordo com o relatório S*tate of the Cloud Report* de 2020 da *Flexera*, o uso do Docker nas organizações cresceu de 57% em 2019 para 65% em 2020 [Fle20] e o kubernetes [Kub20], uma ferramenta de orquestração de contentores que utiliza o Docker, cresceu de 48% em 2020 para 58% em 2020. Para além disso, as ofertas de serviços do tipo contentores-como-um-Serviço na plataforma dos fornecedores líderes de serviços em nuvem públicos, *Amazon Web Services* (AWS), *Google Cloud Platform* e *Microoft Azure*, têm também tido um crescimento significativo: o conjunto de serviço *Amazon Elastic containers Service* (Amazon ECS) [Ser20b] *Amazon Elastic Kubernetes Service* (Amazon EKS) [Ser20c] cresceu de 44% em 2019 para 54% em 2020, o Serviço de contentores do Azure [Mic20] cresceu de 28% em 2019 para 46% em 2020, enquanto o Google *container Engine* [Goo20] aumentou de 15% em 2019 para 24% em 2020.

Um exemplo ilustrativo sobre o interesse pela tecnologia de contentores é o caso da *Netflix* [Net20], que em 2008 adotou uma estratégia de migração total para a nuvem e migrou toda a sua infraestrutura de computação interna para a AWS (*Amazon Web Services*), tendo quase todos os serviços da Netflix sido executados em máquinas virtuais (VMs) na AWS. A migração total para a nuvem constituíu um enorme sucesso para a *Netflix*, tendo inovado e contribuído para criação de normas nativas na nuvem, tais como, os microserviços fracamente acoplados (*loosely coupled microservices*) e a infraestrutura imutável (*immutable infrastructure*), que se tornaram em melhores práticas na industria da computação virtualizada ao nível do *hardware* da AWS. A *Netflix* está a investir na tecnologia de contentores, abordando a adoção de contentores para infraestrutura nativa existente na nuvem [LSB18] [AB17].

1.2 Definição do Problema a Resolver e Objetivos da Investigação

O presente trabalho consiste em apresentar uma avaliação e análise comparativa de desempenho de contentores em Docker com e sem a plataforma de segurança Twistlock para

contentores em Docker, durante o tempo de execução do sistema, aplicações, consumo de memória e de todo processo de atividades realizadas no sistema.

O objetivo da investigação consiste em implementar uma solução baseada em contentores do Docker, configurar, testar e avaliar o desempenho das diferentes ferramentas de segurança oferecida pelo Twistlock usando contentores do Docker com a finalidade de garantir a proteção das aplicações, *pipeline* de contentores e dos ambientes de implementação, definir as políticas de segurança e de conformidade.

1.3 Abordagem para Resolver o Problema

A abordagem para resolver o problema consiste em investigação orientada à resolução do problema envolvendo a implementação e configuração de contentores do Docker e a plataforma de segurança Twistlock com teste de desempenho ao nível do sistema operativo, num ambiente constituído por duas máquinas físicas. Depois da realização dos testes de desempenho das mesmas, será efetuado um estudo comparativo do desempenho, tendo em conta o tempo de execução do processador, a carga de trabalho e as políticas de segurança aplicadas pelo Twistlock e as interpretações do mesmo.

1.4 Principais Contribuições

Existem estudos feitos acerca da segurança de contentores em Docker envolvendo análise de algumas vulnerabilidades e ameaças críticas à segurança do Docker em nuvens e da virtualização ao nível de sistema operativo, incluindo a medição do desempenho de contentores do Docker. No entanto, no que se refere a segurança dos contentores em Docker utilizando a plataforma Twistlock ou às ferramentas Twistlock ainda existem muito poucos estudos realizados que abordem a integração das duas tecnologias, Twistlock e Docker. Por isto, a principal contribuição desta dissertação consiste em avaliar e comparar o desempenho de contentores do Docker com e sem a plataforma Twistlock.

1.5 Limitações do Trabalho Desenvolvido

Tendo em consideração o número de máquinas físicas disponibilizadas para a elaboração da investigação, este estudo limita-se a uma análise e avaliação do desempenho de contentores do Docker explorando algumas ferramentas e configurações relevantes que permitirão manter a segurança face às ameaças e vulnerabilidades expostas na fase do ciclo de vida e durante a produção de contentores.

1.6 Organização da Dissertação

Para facilitar uma melhor compreensão estruturamos a dissertação da seguinte forma:

Capítulo 1: apresenta uma breve introdução sobre a adoção da tecnologia de contentores do Docker, apresenta uma definição do problema a resolver e objetivos da investigação, abordagem para resolver o problema, principais pontribuições, limitações do trabalho desenvolvido e a organização da dissertação.

Capítulo 2: explica os conceitos básicos, aborda uma introdução sobre alguns conceitos da virtualização ao nível de sistema operativo, resenha histórica da virtualização ao nível de sistema operativo, contentores incluindo os seus recursos, conceitos de cibersegurança (a tríade CIA e vulnerabilidades) e os trabalhos relacionados.

Capítulo 3: aborda um conceito geral do Docker, Contentorização com Docker, a arquitetura Docker, os componentes relacionados do Docker e algumas vantagens do Docker, Docker *swarm* e descreve conceitos gerais da plataforma Twistlock, a sua função, as principais características apresentadas pelo Twistlock, arquitetura Twistlock e principais mecanismos de segurança oferecida pelo Twistlock.

Capítulo 4: descreve o ambiente experimental, as plataformas utilizadas e apresenta uma avaliação do desempenho de contentores do Docker com Twistlock e sem Twistlocks.

Capítulo 5: apresenta as principais conclusões e trabalho futuro.

Capítulo 2

Conceitos Básicos

2.1 Introdução

A virtualização é um conceito que permite partilhar os recursos físicos de uma máquina incluindo processador, memória, armazenamento entre várias maquinas virtuais. Com a virtualização é possível usar a capacidade total de uma máquina física, distribuindo os seus recursos por vários utilizadores ou ambientes, fornecer a possibilidade de otimizar os recursos de utilização e flexibidade da mesma [Red18].

A tecnologia de virtualização teve a sua origem na década de 1960. Contudo, ela passou a ser adotada amplamente somente no início dos anos 2000. As tecnologias que tornaram a virtualização uma realidade como os hipervisores, foram desenvolvidas há décadas para dar a vários utilizadores acesso simultâneo a computadores que realizavam processamento em lote [Red18]. Hoje com a virtualização podemos executar vários sistemas operativos em simultâneo numa única máquina, com o propósito de reduzir o custo de vários servidores solicitando serviços numa única máquina.

2.2 Resenha Histórica da Virtualização ao Nível de Sistema Operativo

O primeiro sistema operativo a suportar virtualização total para Vms foi o *Conversational Monitor system* (CMS). O CMS suportava tanto a virtualização completa quanto a paravirtualização. No início da decada de 60, a IBM apresentou a família VM de sistemas, que executava vários sistemas operativos de utilizador único sobre a base do VM *Control Program*, um dos primeiros hypervisors de tipo 1. A área da virtualização que a IBM popularizou na década de 60 é conhecida como virtualização de plataforma (ou sistema). Nesta forma de virtualização, a plataforma de *hardware* sudjacente é virtualizada para partilhá-la com diversos sistemas operativos e diferentes utilizadores [DJ11].

A virtualização ao nível do sistema operativo com ajuda da tecnologia de máquinas virtuais (VM), conduziu a um novo modo de computação conhecido como computação em nuvem. A computação em nuvem está a transformar o cenário da computação, deslocando os custos de *hardware* e pessoal da administração de um centro de computacão para terceiros. No entanto, a computação em nuvem tem pelo menos dois desafios. O primeiro é a capacidade de utilizar um número variável de máquinas físicas e instâncias de VM, dependendo das necessidades de um problema. O segundo desafio diz respeito à operação lenta de instânciar novas VMs [KGJ11].

Atualmente, as novas VMs originam-se como inicializações recentes ou réplicas de uma VM de modelo, sem conhecer o estado atual da aplicação. Portanto, para melhorar o suporte à computação em nuvem, uma grande quantidade de investigação e desenvolvimento deve ser feita. Como mencionado anteriormente, é lento inicializar uma VM no nível do *hardware* para cada VM cria a sua própria imagem do zero. Em ambiente de computação em nuvem, talvez milhares de VMs precisam ser inicializadas simultaneamente. Além da operação lenta, armazenar as imagens da VM também se torna um problema. Por uma questão de facto, há um considerável conteudo repetido entre as imagens da VM. Além disso, a virtualização completa ao nível de hardware também apresenta as desvantagens do desempenho lento e da baixa densidade e, portanto, é necessário que a para-virtualização modifique o SO convidado, podendo este aspeto ser mitigado através da virtualização assistida por hardware. Para reduzir a sobrecarga de desempenho da virtualização no nível do *hardware*, pode ser necessário até a modificação do *hardware*. A virtualização ao nível do sistema operativo fornece uma solução viável para estes problemas de virtualização no nível de *hardaware*. A virtualização do sistema operativo insere uma camada de virtualização ao nível do sistema operativo para iniciar os recursos físicos da máquina. Ele permite várias VMs isoladas num único *kernel* do sistema operativo. Este tipo de VM é geralmente chamado de ambiente de execução virtual (VE), Virtual Private System (VPS) ou simplesmente *contentor*. De ponto de vista do utilizador, os contentores parecem verdadeiros servidores. Isto significa que um contentor possui o seu próprio conjunto de processos, sistema de ficheiros, contas de utilizador, interfaces de rede com endereços IP, tabela de encaminhamento, regras de *firewall* e outras configurações pessoais. Embora os contentores possam ser personalizados para pessoas diferentes, eles partilham o mesmo *kernel* do sistema operativo. Portanto, a virtualização ao nível do SO também é designada por virtualização de imagem de SO [KGJ11] [Wan18].

2.3 Virtualização ao Nível do Sistema Operativo

A virtualização ao nível de sistema operativo é uma técnica obtida recorrendo através de sistema anfitrião (*host*), com um único *kernel* e de maneira a controlar a funcionalidade do sistema hóspede no qual a sua arquitetura possui um baixo *overhead*, maximizando a utilização de recursos de um servidor e com objetivo de fornecer um isolamento de processos. Essa forma de virtualização tem como vantagem o desempenho por já fazer parte do sistema opertivo. A desvantagem é que o utilizador não pode fazer uso de outro sistema operativo (outro *kernel*) no ambiente virtual [Sil07]. A virtualização ao nível de sistema operativo cria contentores isolados num único servidor físico e as instâncias de sistema operativo para utilizar o *hardware* e *software* em centros de dados e os contentores comportam-se como servidores reais. A virtualização de nível de OS é frequentemente usada na criação de ambientes de *hosting* virtuais para alocar recursos de *hardware* entre um grande número de utilizadores mutualmente desconfiados. É também usado, em menor grau, na consolidação de *hardware* do servidor, movendo serviços em *hosts* separados para contentores ou VMs num servidor [HDF12], [Fre18].

O *Linux containers* é o componente de *software* de código aberto e com uma alta dependência, permitindo que todas as aplicações sejam executadas de forma rápida, confiável num ambiente de computação e impõe muito pouca ou nenhuma sobrecarga ao hospedar instâncias virtuais [Kou17].

A tecnologia de contentores nasceu em 1979 com a *Unix* versão 7 e o sistema *chroot*. O sistema *chroot* isola um processo restringindo o acesso de uma aplicação a uma diretoria específica, onde esta é composto por diretorias raiz e filha. Este sistema foi a primeira mostra de um processo isolado, logo foi adotado e adicionado ao *BSD* OS em 1982. Nos anos 2000 a tecnologia ganhou interesse com a introdução das cadeias *Free BSD* nas quais pode haver várias partições no mesmo sistema. Esta versão foi aperfeiçõada em 2001 com o *Linux Vserver*, que incluía a partição de recursos, que posteriormente foi fortemente ligado ao *kernel* do *Linux* com *OpenVZ* em 2005, e as cadeias foram combinadas com a separação de limites para criar contentores no *Solaris* em 2004. Após as cadeias, a tecnologia de contentores prosseguiu com a introdução de grupos de controlo em 2006. Os grupos de controlo ou *cgroups* foram implementados para contabilizar e isolar o uso de recurso como CPU e memória. Eles, mais tarde, foram utilizados e construídos no LXC (*LinuX containers*) em 2008, e esta era a versão mais completa e estável da tecnologia de contentores da época, visto que funcionava no *kernel Linux* sem nehuma correção. Devido à fiabilidade e estabilidade do *LXC*, muitas outras tecnologias foram contruídas no *LXC* como *Warden* em 2011 e a Docker em 2013, etc... [Bas19].

O principal objetivo para ativar um contentor consiste em iniciar um conjunto limitado de aplicações ou serviços (geralmente referidos como micros serviços) e executá-los dentro do seu próprio ambiente numa área restrita. Esse isolamento impede que o processo em execução num determinado contentor monitorize ou afete outro processo em execução noutro contentor. Além disso, estes serviços em contentores não influenciam ou perturbar os *hots*. A ideia de ser capaz de consolidar muitos serviços espalhados por vários servidores físicos num só, é um dos vários motivos pelos quais as centros de dados optaram por adotar a tecnologia de contentores [Kou17].

Os recursos dos contentores incluem:

- **Segurança**: os serviços de rede podem ser executados num contentor, limitando assim os danos causados por uma violação de segurança. Um invasor que explora com êxito uma falha de segurança numa aplicação em execução neste contentor é restrito ao conjunto de ações possíveis dentro deste contentor [Kou17].
- **Isolamento**: os contentores permitem a implementação de uma ou mais aplicações na mesma máquina física, mesmo se essas aplicações devam operar sob diferentes domínios, cada um exigindo acesso exclusivo aos seus respetivos recursos. Por exemplo, várias aplicações executadas em contentores diferentes podem vincular-se ao mesmo interface de rede física usando um endereço IP distinto associado a cada contentor [Kou17].
- **Virtualização e transparência**: os contentores fornecem o sistema com um am-

biente virtualizado que pode ocultar ou limitar a visibilidade dos dispositivos físicos ou do sistema de configuração abaixo dele. O princípio geral por trás de um contentor é evitar alterar o ambiente em as aplicações estão a ser executadas, com exceção de abordar problemas de segurança ou isolamento [Kou17].

2.4 Conceitos de Cibersegurança

2.4.1 Segurança da Informação versus Cibersegurança

A segurança da informação está relacionada com proteção de um conjunto de dados, no sentido de preservar o valor que possuem para um individuo ou uma organização. São características básicas da segurança da informação os atributos de confidencialidade, integridade e disponibilidade, não estando esta segurança restrita somente a sistemas computacionais, informações eletrónicas ou sistemas de armazenamento digitais [Dig08].

A cibersegurança visa proteger os dados armazenados em formato eletrónico (por exemplo, em computadores, servidores, redes, dispositivos móveis etc.) contra comprometimento ou ataque. A cibersegurança consiste na prática de proteger sistemas, redes e programas contra ataques digitais, vulgarmente designados por ciberataques. Esses ciberataques visam geralmente aceder, alterar ou destruir informações confidenciais, extorquir dinheiro dos utilizadores, ou interromper processos comerciais normais [Cis20]. A implementação de medidas eficazes de cibersegurança é particularmente desafiadora hoje em dia, porque há mais dispositivos do que pessoas e os atacantes estão-se a tornar cada vez mais inovadores [Cis20]. Nesta dissertação estamos focados na segurança em ambientes contentorizados, a qual consiste na proteção da integridade dos contentores. Isto inclui a segurança desde as aplicações até a infraestrutura em que são executadas. É essencial que esta segurança seja integrada e contínua. A segurança no uso de contentores constitui uma preocupação, dado que as tecnologias de virtualização ao nível do sistema operativo desenvolveram-se muito recentemente, devido ao projeto Docker, podendo trazer vulnerabilidades de segurança nas imagens, superfícies de ataques ou ataque de negação de serviço [Sil18], [Dig08], [Red19].

Atualmente o conceito de segurança da informação é definido pela norma ISO/IEC 17799:2005, influenciada pelo norma inglesa (*British Standard*) BS 7799. A série de normas ISO/IEC 27000 foram reservadas para tratar de normas de segurança, incluindo a complementação ao trabalho original da norma inglesa [Dig08].

2.4.2 A Tríade CIA: Confidencialidade, Integridade e Disponibilidade

A Segurança da informação refere-se à proteção existente sobre a informação de uma determinada organização ou pessoa, isto é, aplica-se tanto à informação corporativa quanto

às pessoais. Entende-se por informação todo e qualquer conteúdo ou dado que tenha valor para alguma empresa ou pessoa. Ela pode estar guardada para uso restrito ou exposto ao público para consulta [Dig08]. A tríade CIA (*Confidentiality*, *Integrity and Availability*), Confidencialidade, Integridade e Disponibilidade representa os princípais atributos que, atualmente orientam a análise, o planeamento e a implementação da segurança para um determinado conjunto de dados que se deseja proteger. Estes e outros atributos também importantes podem ser definidos da seguinte forma [CH13]:

- **Confidencialidade**: Propriedade que limita o acesso a informação somente às entidades legítimas, ou seja, um sistema deve garantir que apenas utilizadores autorizados acedem à informação.
- **Integridade**: Propriedade que garante que a informação manipulada mantenha todas as características originais estabelecidas pelo proprietário da informação, incluindo controlo de alteração e garantia do seu ciclo de vida, ou seja, um sistema deve garantir a integridade, a precisão e a ausência de modificações não autorizadas em todos os seus componentes.
- **Disponibilidade**: Um sistema deve garantir que todos os componentes do sistema estejam disponíveis e operacionais quando necessários por utilizadores autorizados.
- **Prestação de contas (*Accountability*)**: Capacidade de um sistema de responsabilizar os utilizadores pelas suas ações (por exemplo, uso indevido de informação).
- **Auditabilidade**: Capacidade de um sistema de realizar uma monitorização persistente e não ignorável de todas as ações executadas por seres humanos ou máquinas no sistema.
- **Autenticidade / Confiabilidade**: Capacidade de um sistema de verificar a identidade e estabelecer confiança em terceiros e na informação por estes fornecida.
- **Não-repúdio**: Capacidade de um sistema de provar (com validade legal) a ocorrência ou não ocorrência de um evento ou participação ou não participação de uma parte num evento.
- **Privacidade**: Um sistema deve obedecer à legislação de privacidade e deve permitir que indivíduos controlem, sempre que possível, as suas informações pessoais (envolvimento do utilizador)

2.4.3 Vulnerabilidades

Um conceito fundamental no domínio da segurança de *software* é o conceito de vulnerabilidade. Uma vulnerabilidade é um defeito do sistema (de *software* ou outro) que pode ser explorado por um atacante com o objetivo de violar a política de segurança estabelecida [CS17]. As vulnerabilidades podem ser classificadas com base em três categorias:

- **Vulnerabilidade de Projeto**: é introduzido durante a fase de projeto de *software* (obtenção de requisitos e desenho) [CS17].
- **Vulnerabilidade de Codificação** (ou implementação): é introduzido durante a programação do *software*, ou seja, um *bug* com implicação de segurança [CS17].
- **Vulnerabilidade Operacional**: este é causada pelo ambiente no qual o *software* é executado ou pela sua configuração [CS17].

A lista das vulnerabilidades e exposições comuns (CVE - Common Vulnerabilities and Exposures) encontra-se disponível em [Cve20].

2.4.4 Defesa de Vulnerabilidades

A defesa consiste no conjunto de políticas e mecanismos desenhados, concretizados e implementados para diminuir as vulnerabilidades de um sistema, detetar e contrariar ou anular ataques passados ou atuais e minimizar os riscos decorrentes de ataques bem-sucedidos [And10]. A seguir aborda-se a defesa para faltas e falhas previsíveis e a defesa contra atividades não não autorizadas [And10]:

- **Defesa para Faltas e Falhas Previsíveis**: A defesa contra faltas e falhas previsíveis visa sobretudo minimizar o impacto de problemas que ocorrem com maior frequência, mas cujo impacto global é normalmente menor, por exemplo [And10]:
  - Bloqueio na execução de aplicações ou no sistema operativo;
  - Falhas temporárias de conectividade em troços da rede.
- **Defesa contra Atividade não Autorizadas**: A defesa contra atividade não autorizadas é um problema completamente diferente, isto é, da defesa de sistemas computacionais contra iniciativas tomadas por indivíduos e contra o funcionamento normal do sistema. As atividades não autorizadas podem ter originem em dois universos computacionais disjuntos: nos sujeitos que pertencem a organização dona do sistema computacionais que se quer proteger e os sujeitos que a ela não pertencem [And10]. Neste caso, pretende-se defender o sistema contra [And10]:
  - **Acesso a Informação**: neste caso inclui-se todos os tipos de acesso as informações reservadas ou confidenciais, logo não explicitamente tornadas públicas, guardados em um sistemas computacionais ou em trânsito em redes.
  - **Negação de serviço** (*Denial of Service*, DoS): o impedimento de prestação de serviço é um caso extremo de uso excessivo ou abusivo de recurso computacionais. E é extremo porque o objetivo principal da atividade ilícita é apenas impedir que terceiros tenham acesso aos recursos afetados sem qualquer usufruto próprio dos mesmos.

– **Utilização Exagerada** ou **abusiva de recurso computacionais**: Os recursos computacionais podem ser diversos tipos como o tempo de processamento, memória primário ou secundário, ocupação de redes de comunicação, etc. Mas são sempre limitado. Consequentemente, o uso exagerado ou abusiva de recurso pode ter como efeito a sua não disponibilidade para o terceiro.

2.5 Trabalhos Relacionados

No artigo [Mic17] são apresentadas considerações e recomendações para proteger os contentores do Docker instalados no Serviço de contentores do Azure. Várias dessas considerações aplicam-se, em geral, aos contentores do Docker, implementados no Azure ou noutros ambientes.

Chelladhurai *et al.* [CCK16] analisaram algumas das vulnerabilidades e ameaças críticas à segurança da tecnologia Docker. Os grupos de controlo (*Cgroups*) e a capacidade proeminente do *kernel Linux* ajudam a garantir que cada *contentor* tenha a sua partilha justa de memória, CPU, E/S de disco. Um único contentor não pode comprometer o sistema esgotando um destes recursos. As análises revelam que as questões de seguranças podem ser resolvidas através de vários métodos, incluindo a proposta de uma solução pragmática para os ataques de negação de serviço, sendo necessário inovar as soluções existentes para fortalecer os requisitos da segurança para carga de trabalhos e ambientes em contentores.

Casalicchio, Perciballi e Kumar [CP17] abordaram a supervisão de sistemas baseados em contentores que está no centro de soluções de gestão de recursos deste tipo de virtualização. Eles Exploraram as ferramentas disponíveis para medir o desempenho do Docker na perspetiva do sistema operativo *host* e do ambiente de virtualização e forneceram uma caraterização da sobrecarga de E/S da CPU (Unidade Central de Processamento – do inglês *Central Processing Unit*) e do disco introduzida pelos contentores. Os autores chegaram à conclusão que medir o desempenho do contentores com o objetivo de caraterizar a sobrecarga e carga de trabalho não é uma tarefa fácil, também porque não existem ferramentas estáveis e dedicadas que cubram uma ampla variedade de métricas de desempenho. Segundo estes autores, as ferramentas de monitorização de contentores disponíveis oferecem resultados corretos, mas estes devem ser devidamente interpretados. Além disso, a criação de uma infraestrutura de supervisão para contentores requer a interconexão de pelo menos três ferramentas: O *cAdvisor* e *mpstat* permitiram medir a carga de trabalho efetiva gerado pelo contentor na CPU, ou seja, % CPU; o Docker *stats* medem a quantidade de CPU solicitada (CPUreq) pelas *threads* em execução no contentor; existe uma correlação entre a cota de CPU definida para o contentor e o *overhead* de CPU (CPUovh). Caso nenhuma cota seja definida, quando o CPUreq estiver entre 65% e 75%, a sobrecarga do contentor é de cerca de 10% em relação à carga da CPU nativa. Quando CPUreq terminar os 80%, a sobrecarga é menor que 5%; não há ferramentas dedicadas à supervisão de disco E/S para ambientes baseados em Docker; a sobrecarga de E/S do disco varia entre 10% e 30%, não tendo encontrado nenhuma correlação entre a sobrecarga e o tamanho

da entrada ou a composição da carga de trabalho do disco.

Oliveira *et al.* [dOPP+19] analisaram a segurança do uso de contentores do Docker num ambiente de computação em nuvem real baseado em serviços de Infraestrutura como um Serviço (IaaS) do OpenStack. O ambiente de nuvem analisado tem como gestor de recursos o *OpenStack* e contentorização através de Docker. A principal contribuição deste trabalho consiste na análise de vulnerabilidades, riscos e ameaças no ambiente acima referido. O trabalho descreveu os impactos gerados com a exploração das vulnerabilidades encontradas, mas, também documentou as técnicas e ferramentas para solucionar os problemas. No contexto específico de contentores, este trabalho reforça a necessidade da auditoria das políticas de segurança de forma sistemática e, se possível, automática, com o auxílio de ferramentas específicas por critério. Observou-se ainda que devem ser seguidas as restrições sugeridas no perfil padrão do Docker (*docker-default*) e, caso seja necessário alguma adequação, a disponibilidade deve ser analisada cuidadosamente num contexto global com o auxílio de ferramentas de auditoria. No entanto, de acordo com aqueles autores o uso do perfil padrão não dispensa a auditoria de segurança através de ferramentas, pois o trabalho revelou que um contentor iniciado com o perfil padrão do Docker é capaz de fazer um ataque DoS (*Denial of Service*) através de um *forkbomb*.

Duarte e Antunes [DA19] procuraram analisar as vulnerabilidades de segurança do Docker e as respetivas medidas a serem tomadas. Para isso, efetuam uma análise detalhada dos relatórios de segurança e respetivas vulnerabilidades, sistematizando-os de acordo com causas, efeitos e consequências. Em seguida, analisaram a aplicabilidade dos analisadores de código estático na base de código do Docker, tentando entender, em retrospetiva, a utilidade dos relatórios das ferramentas. Os resultados mostram uma prevalência de desvio e ganho de privilégios e as ferramentas utilizadas foram ineficazes, não ajudando a identificar as vulnerabilidades analisadas. Também foi observado que seria fácil encontrar algumas vulnerabilidades usando testes de robustez ou de penetração, enquanto outras seriam realmente desafiadoras. Embora estas ferramentas tenham reportado alguns problemas que podem ser importantes para a qualidade e a capacidade de manutenção do código, os resultados obtidos com estas ferramentas dificilmente podem impedir diretamente a exploração das vulnerabilidades. Por outro lado, observou-se também que as técnicas tradicionalmente usadas para testar componentes ou sistemas de uma perspetiva externa podem revelar facilmente algumas das vulnerabilidades se aplicadas a funções críticas.

O trabalho de Madhumathi [Mad18] concentrou-se nas aplicações multicontentor, pelo que considera não apenas os contentores do Docker mas também precisa de ter grupos de contentores para aplicações multicontentor. O estudo revela que o desempenho do Docker é melhor (mais rápido) que o das máquinas virtuais, pois não possui sistema operativo convidado e menor sobrecarga de recursos. Além disso, a segurança de contentores do Docker também pode ser implementada através da ferramenta de supervisão *SYSDIG*, onde os contentores do Docker são implementados de maneira segura.

Levin, Stopel e Yanay [LSY18] investigaram um sistema e um método para execução se-

gura de contentores de software usando perfis de segurança. O método inclui i) receber um evento indicando que a imagem de um contentor requer a crição de perfil, em que a imagem do contentor inclui os recursos utilizados para executar o correspondente contentor da aplicação; ii) gerar um perfil de segurança para a imagem do contentor quando o evento é recebido, em que o perfil de segurança gerado indica pelo menos os portos de rede que são permitidos para, pelo menos, um dos seguintes: acesso ao contentor da aplicação, e acesso pelo contentor da aplicação; iii) monitorizar uma operação de execução do contentor da aplicação; iv) e detetar a violação do perfil de segurança com base na operação monitorizada.

A análise realizada por Tak *et al.* [Byu17] revela evidências de que imagens de linhagem não resolvida introduziram vulnerabilidades em contentores pertencentes a terceiros, que as atualizações para contentores públicos em serviço são comuns e que o scan de contentores ou imagens é insuficiente para erradicar vulnerabilidades de contentores públicos. Este autores defenderam um melhor suporte aos sistemas para rastrear a proveniência da imagem e resolver alterações disruptivas nos contentores e propor práticas que os utilizadores devem adotar para limitar a vulnerabilidade dos contentores.

Abbott [Abb17], na sua dissertação de mestrado aborda uma análise comparativa de contentores e máquinas virtuais em ambiente de produção, em que os contentores empregam uma série de ferramentas *Linux* e recursos do *kernel* para isolar parcialmente o conteúdo de contentores do restante sistema *host*. Em ambientes de produção, as máquinas virtuais (*VMs*) têm sido a melhor prática atual para isolamento, segregação de processos e aplicações. Os contentores poderão tornar-se comuns em ambientes de produção, mas eles não foram testados em termos de segurança como as máquinas virtuais. Como resultado, este trabalho seria mais aplicável a um grupo de desenvolvimento e/ou operações e teria sido mais adequado focar a investigação nas vulnerabilidades de contentores em execução.

Manu *et al.* [MPA+16] investigaram a segurança do *Cloud PaaS* (Plataforma como Serviço) em termos da segurança de contentores do Docker, fornecendo uma análise e comparação do Docker com outras tecnologias de contentores e com máquinas virtuais.

Henrique na sua dissertação [Hen17] reportou uma avaliação de desempenho de contentores Docker para aplicações do Supremo Tribunal Federal. A dissertação explorou o desempenho de contentore Docker para reduzir custos e melhorar a eficiência dos centros de dados corporativos. Os responsáveis pela área de infraestrutura de *TI* (Tecnologia de Informação) fizeram grandes investimentos, na última década, na consolidação de servidores. A consolidação de servidores permite que vários servidores virtuais sejam executadas em um mesmo *host*, o que conduz a uma melhor utilização da capacidade do *hardware*, reduzindo o espaço e o consumo de energia nos centros de dados. Os resultados obtidos nas avaliações mostram que a sobrecarga resultante do uso de contentore Docker é inferior a do ambiente análogo utilizando máquinas virtuais com hipervisor.

Tung *et al.* [TTSS14] apresentam uma abordagem económica para avaliar o desempenho do *scanner* de vulnerabilidades da *web*, a qual consistiu em aplicar a matriz de confusão para estimar o desempenho dos *scanners* de vulnerabilidades da *Web* de forma a que

as aplicações *Web* estão expostos a várias ameaças e ataques. Concluíram que a relação entre as vulnerabilidades detetadas e as reais é mais complexa do que se conhecia anteriormente.

Stopel *et al.* [SLD+19] abordaram a proteção de aplicações contra exposição não segura em redes informáticas, tendo proposto um método para i) identificar pelo menos uma porta através do qual a aplicacão é acessível quando não está configurado correctamente, em que a aplicação é executada num *host* conectado a pelo menos uma rede, tendo o *host* pelo menos uma porta; ii) enviar, para uma fonte externa, enviando, para um recurso externo, os dados de conexão para estabelecer a conexão à aplicação atráves de pelo menos uma porta, em que o recurso externo está configurado para tentar se conectar à aplicação com base nos dados da conexão e retornar resultados da tentativa da conexão; iii) determinar, com base nos resultados da tentativa de conexão, se existe uma vulnerbilidade exposta; iv) e realizar pelo menos uma ação de mitigação quando existir uma vulnerabilidade exposta.

2.6 Conclusão

Neste capítulo foram apresentados conceitos básicos sobre a tecnologia de virtualização ao nível de sistema operativo, incluindo os conceitos de contentores com seus recursos, a segurança e suas propriedades particularemente para os contentores do Docker, as vulnerabilidades de *software* e a defesa de vulnerabilidades e os trabalhos relacionados. Os trabalhos relacionados constituíram uma base importante para a eleboração do problema que conduziu à realização do trabalho técnico que suporta esta dissertação.

Capítulo 3

Tecnologias Docker e Twistlock

3.1 Introdução

Este capítulo é dedicado a duas tecnologias centrais para o trabalho conducente a esta dissertação: o Docker e o Twistlock.

O Docker [Doc20a] é uma plataforma de contentorização usada para instalar uma dada aplicação juntamente com todas as suas dependências na forma de contentores (*containers*), para garantir que a aplicação corre em qualquer ambiente de computação, podendo ser um ambiente de desenvolvimento, de teste ou de produção. O Docker é uma ferramenta de de software livre, projetada para facilitar a criação, implementação e execução de aplicações através do uso de contentores [Roh19]. O Docker foi distribuído inicialmente a 13 de março de 2013, tendo sido desenvolvido na linguagem de programação Go. A última versão estável do Docker (19.03.11), à data de escrita desta dissertação, data de 1 de junho de 2020 [Wik20a].

Apesar de os contentores em Linux (LXC - Linux *containers*) [Con20] terem contribuído para o interesse e a disseminação da tecnologia de contentores, a introdução do Docker conduziu a uma utilização crescente e generalizada da tecnologia de contentores. O Docker usou os LXC, antes da versão Docker 0.9, como ambientes de execução por omissão, tendo substituído os LXC por uma *libcontentore library* (atualmente *opencontainers/runc* [Ope14] aquando da distribuição da versão 0.9 do Docker a 13 de março de 2014 [Doc14]. Os *Libcontainers* foram fornecidos pela LMCTFY (*Let Me Contain That For You*) [Lmc20] que era uma pilha de contentores de código aberto onde as aplicações criavam e geriam os seus próprios subcontentores. Para além de construir sobre *software* anterior, o Docker tinha uma interface gráfica (GUI - *graphical user interface*) fácil de utilizar e que era capaz de ter várias aplicações com diferentes requisitos de sistema operativo executadas sobre um único sistema operativo. Devido a estas características, a adoção do Docker cresceu significativamente, atingindo 100 milhões de descargas (*downloads*) num ano e, após este sucesso na utilização do Docker, emergiu uma outra tecnologia, o rkt [Rkt20], o qual tentou corrigir alguns dos problemas do Docker através da incorporação de requisitos de segurança e de produção mais rigorosos [Bas19]. O impacto e a popularidade destas duas tecnolgias (Docker e rkt) aumentaram à medida que interagiram com a *Cloud Native Computing Fundation* (CNCF) [Cnf20], uma fundação que incluía, e incluí, um *hub* global de programadores, fundações e fabricantes de tecnologias para a nuvem [Bas19]. Esta cooperação estimulou o envolvimento e a colaboração da comunidade, que continuou a crescer, tendo posteriormente a *Microsoft* contribuído para esse crescimento quando permitiu em 2016 que contentores em Linux, incluindo

o Docker e o *rkt*, corressem nativamente em computadores com sistema operativo Windows, incluindo o *Windows* 10 [Wik20a] e incluindo contentores em *Windows* como uma nova *feature* no *Windows Server* 2016 [Wik20b]. Atualmente existem versões do Docker para diversos ambientes de computação, incluindo Linux, *Windows* e *Mac* [DCo20], sendo também oferecido como um serviço pelos mais importantes fornecedores de serviços em nuvem, incluindo *Amazon Elastic container Registry* [Ser20a], *container Registry* da *Google Cloud Platform* [Pla20], *container Registry* [Azu20b] e *container Instances* [Azu20a] do *Microsoft Azure*, IBM *Cloud containers Registry* [Clo20b] e *container Registry* [Clo20a] da Alibaba *Cloud*.

O Twistlock foi integrado na Prisma Cloud em 2019, sendo atualmente oferecido como Prisma Cloud. A Prisma *Cloud* é uma plataforma para gestão do estado de segurança da nuvem e uma plataforma de proteção de cargas de trabalho na nuvem que permite às equipas de infraestrutura, operações e segurança trabalharem em conjunto. A plataforma Prisma Cloud utiliza as APIs dos fornecedores de serviços em nuvem para acesso apenas de leitura para tráfego de rede, atividades do utilizador, configuração de sistemas e serviços e correlaciona estes conjuntos de dados diferentes para ajudar as equipas de conformidade com a nuvem e análise de segurança a prioritizar riscos e a responder rapidamente a problemas. A plataforma Prisma *Cloud* também usa uma abordagem baseada em agentes para proteger hosts e contentores contra vulnerabilidades, malware e violações de conformidade [Tec19].

Este capítulo é dedicado ao Docker e ao Twistlock/Prisma Cloud, sendo abordados os principais aspetos destas duas tecnologias.

3.2 Docker

3.2.1 Contentorização Usando o Docker

A contentorização resulta da virtualização ao nível do sistema operativo (SO) que cria várias unidades virtuais no espaço do utilizador, designadas por contentores. Os contentores partilham o mesmo kernel do host, mas são isolados entre si por espaços de nomes privados e mecanismos de controlo de recursos ao nível do SO. A Virtualização baseada em contentores fornece um nível diferente de abstração em termos de virtualização e isolamento em relação aos hypervisors usados na virtualização ao nível do hardware. Os hypervisors usam muito *hardware*, o que resulta em *overhead* em termos de virtualização de *hardware* e *drivers* de dispositivos virtuais, sendo executado um sistema operativo completo (por exemplo, Linux, *Windows*) sobre esse *hardware* virtualizado em cada instância de máquina virtual. Em contraste, os contentores implementam o isolamento dos processos ao nível do sistema operativo, evitando esse *overhead*. Os contentores são executados sobre o mesmo *kernel* partilhado do sistema operativo da máquina host subjacente, podendo ser executados um ou mais processos em cada contentor. Nos contentores não é

necessário pré-alocar RAM (*Random Access Memory*), uma vez que ela é alocada dinamicamente durante a criação de contentores, enquanto nas máquinas virtuais é necessário primeiro pré-alocar a memória e depois criar a máquina virtual. A contentorização conduz a uma melhor utilização de recursos e tem um processo de inicialização (*boot-up*) curto em relação às máquinas virtuais [Roh19].

Os contentores podem correr virtualmente em qualquer ambiente, com desenvolvimento e implementação muito fáceis nos sistemas operativos Linux, *Windows* e *Mac*, em máquinas virtuais, em centros de dados ou em plataformas de nuvens públicas. Os contentores virtualizam os recursos de CPU (*Central Processing Unit*), memória, armazenamento e rede ao nível do sistema operativo, fornecendo aos programadores uma visão do sistema operativo isolado logicamente de outras aplicações. O Docker é o formato de contentor de código aberto mais popular disponível e é suportado pelas plataformas de nuvens públicas mais relevantes [Roh19].

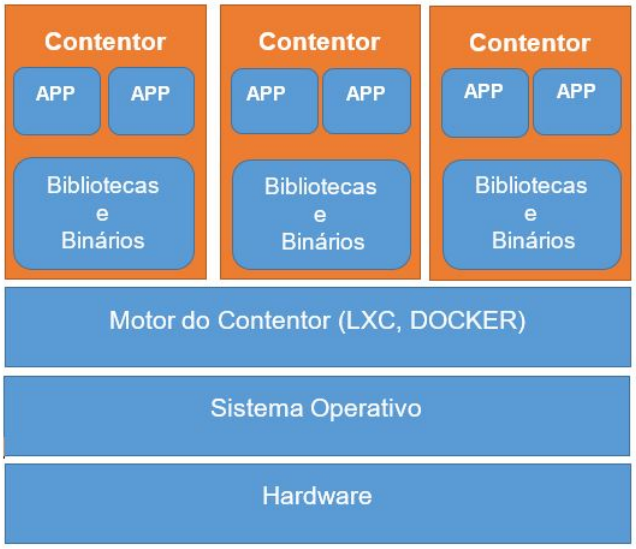

Figura 3.1: Arquitetura de virtualização baseada em contentores (figura redesenhada a partir de [Mor17]).

3.2.2 Arquitetura do Docker

A figura 3.1. representa a arquitetura da virtualização baseada em contentores, a qual ilustra o funcionamento de contentores sobre o motor do contentor (*Container Engine*), o qual corre sobre o sistema operativo do computador hospedeiro (*host*). No caso do Docker, o motor do contentor é designado por motor do Docker (Docker *Engine*), que atua como uma aplicação cliente-servidor [Ion19]]. A arquitetura básica do motor do Docker pode ser dividida em três componentes principais [Ion19], [Doc20b]:

- **O Docker *daemon***: um servidor que corre um processo daemon (comando *dockerd*), executado em segundo plano no sistema hospedeiro (*host*) e usado para con-

trolar o motor do Docker centralmente. O Docker *daemon* cria e gere objetos do Docker tais como imagens, contentores, redes e volumes.

- **APIs REST**: APIs que especificam um conjunto de interfaces que os programas podem usar para comunicar e fornecer instruções ao Docker *daemon*, usando o paradigma REST (*Representational state Transfer*).
- **Uma interface de linha de comandos (CLI - *Command Line Interface*) do cliente Docke**r: a CLI usa APIs do Docker para controlar ou interagir com o *daemon* do Docker através de *scripts* ou comandos diretos da CLI. Muitas outras aplicações do Docker usam a API e a CLI subjacentes.

O Docker permite que os utilizadores iniciem, parem e giram os contentores diretamente do dispositivo. O *daemon* é endereçado usando o comando Docker e instruções tais como *build* (criar), *pull* (baixar ou fazer o *download*) ou *run* (executar). O cliente e o servidor podem estar no mesmo sistema. Em alternativa, os utilizadores têm a opção de aceder a um *daemon* do Docker noutro sistema. Dependendo do tipo de conexão a ser estabelecida, a comunicação entre o cliente e o servidor é feita através da API REST, de um *socket* ou uma interface de rede [Ion19]. A figura 3.2 ilustra a interação dos diferentes componentes do Docker usando os comandos Docker *build*, Docker *pull* e Docker *run*.

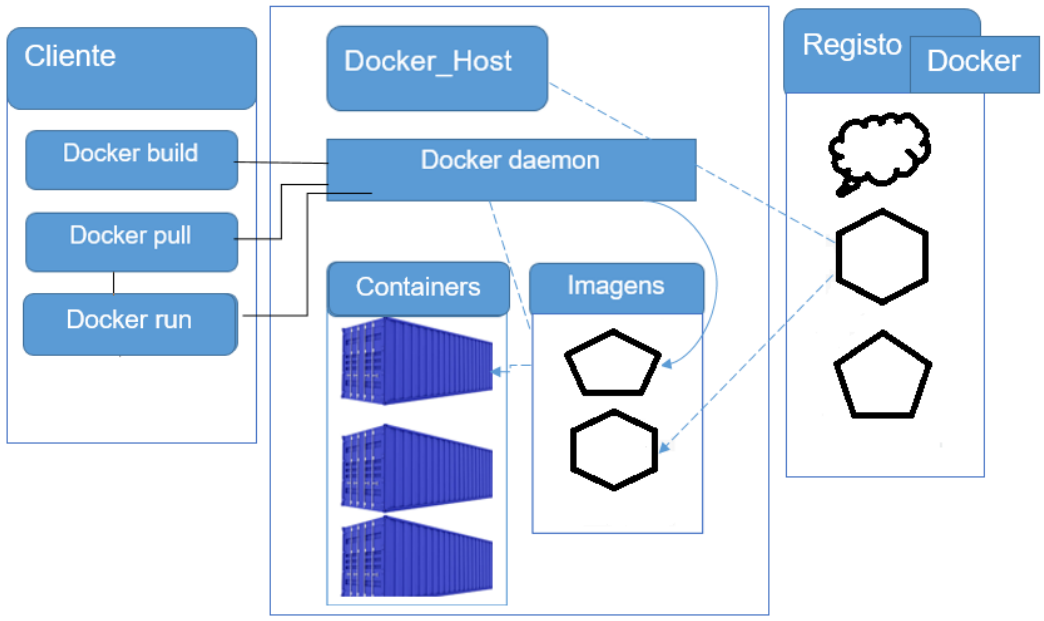

Figura 3.2: Representação esquemática da arquitectura do Docker (figura redesenhada a partir de [Doc20b]).

3.2.3 Componentes do Docker

Os principais componentes do Docker incluem clientes e servidores do Docker, imagens

do Docker, Ficheiros do Docker (*Dockerfile*), Registos do Docker, contentores do Docker, entre outros. A seguir é apresentada uma breve descrição destes componentes.

3.2.3.1 Docker *Daemon*

O Docker *daemon* fica à espera de solicitações da API do Docker, sendo também responsável por gerir imagens de contentores, redes, e volumes, incluindo baixar ou enviar imagens para registo remoto (com o *hub* Docker), criar imagens a partir do *Dockerfile* e assinar imagens. Um *daemon* também pode comunicar com outros *daemons* para gerir serviços do Docker. O *daemon* é executado como *root* no sistema hospedeiro e é controlado remotamente através de um *socket* do *Unix*. O Docker *Daemon*/*server* recebe solicitações do cliente Docker através de APIs REST ou CLI, procedendo conforme a solicitação [Mor17], [CMD16].

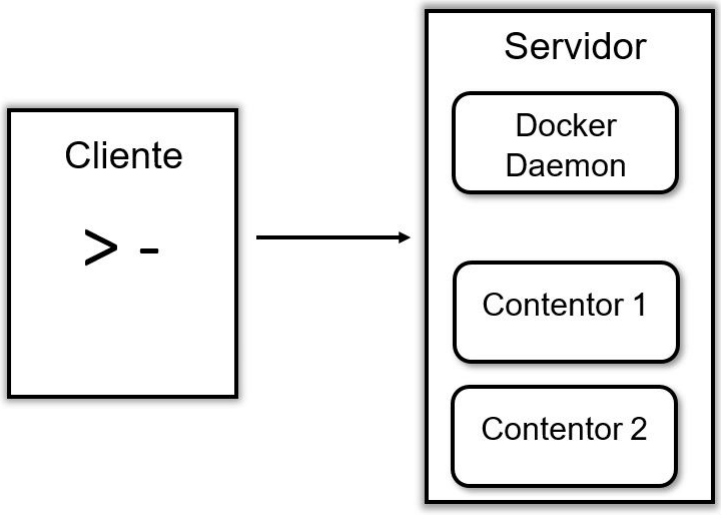

Figura 3.3: Arquitetura cliente/servidor do Docker (figura redesenhada a partir de [Roh19].)

3.2.3.2 Cliente Docker

O cliente Docker representa a principal forma de muitos utilizadores do Docker interagirem com o mesmo. Quando se utilisa o docker *run*, o cliente envia o comando para o *dockerd*, que o executa. Os clientes Docker utilizam comandos na API do Docker para poderem comunicar com mais do que um *daemon* [Doc20a].

3.2.3.3 Imagens do Docker

As imagens do Docker são usadas para criar contentores do Docker usando um modelo somente de leitura, o que significa que uma vez criada, uma imagem nunca é modificada. A base de cada imagem do Docker é uma imagem de base como, por exemplo, a do Ubuntu 20.04 LTS ou a do Fedora 32. As imagens de base também podem ser criadas a partir do zero, podendo ser adicionadas as aplicações necessárias à imagem de base, modificando-a, pelo que este processo de criação de uma imagem é conhecido como “compremetendo

a alteração" [Roh19]. A figura 3.4 representa um exemplo de uma imagem de um contentore, a qual apresenta uma imagem do Ubuntu com uma instalação do Apache. Esta imagem é constituida por três camadas básicas do Ubuntu, uma camada de atualização, uma camada *Apache* e uma camada de ficheiro personalizado na parte superior [Twi19a].

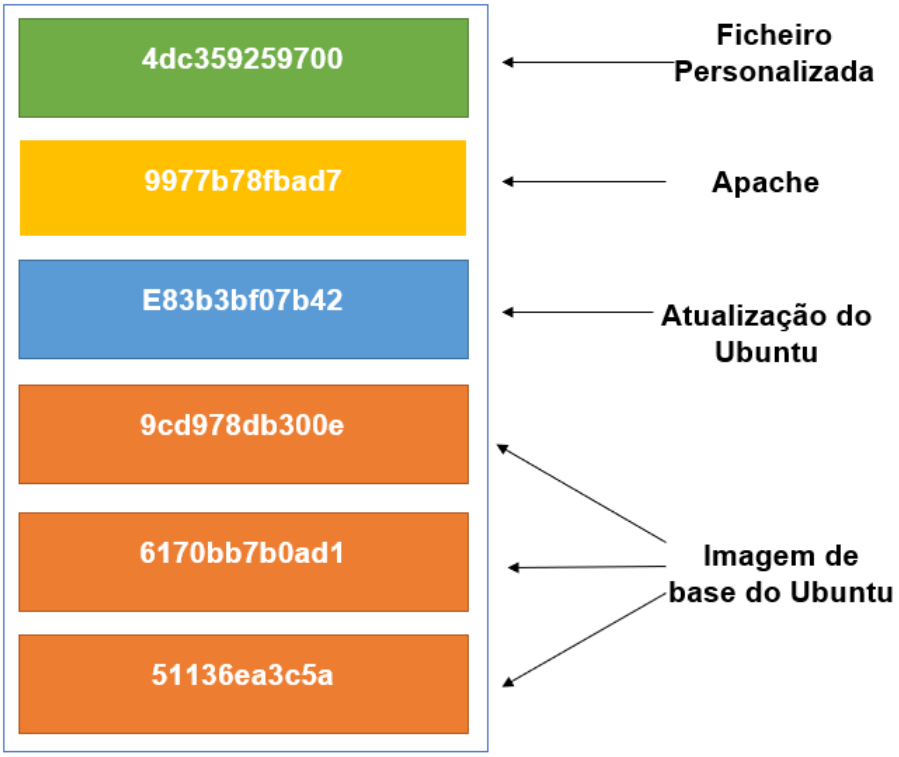

Figura 3.4: Visualização de uma imagem Docker (figura redesenhada a partir de [Twi19b].)

3.2.3.4 Ficheiro Docker

O ficheiro Docker (*Dockerfile*) é um ficheiro de texto que contém numa série de instruções sobre como criar sua imagem do Docker. Esta imagem contém todo o código do projeto e suas depedências. A mesma imagem do Docker pode ser usada para correr vários contentores, cada um com modificações na imagem subjacente. A imagem final pode ser carregada no Docker *Hub* e partilhada entre vários colaboradores para teste e implementação [Roh19].

3.2.3.5 Registos do Docker

Os registos do Docker (*Docker registry*) representam o componente de armazenamento para imagens do Docker para posterior partilha. É possível armazenar as imagens em repositórios públicos ou privados, para que vários utilizadores possam colaborar na criação de uma aplicação. O Docker *Hub* é o próprio repositório da nuvem do Docker. O Docker *hub* é designado por registo público, onde todos podem obter imagens disponíveis e enviar as suas próprias imagens [Roh19].

3.2.3.6 Contentores do Docker

Os contentores do Dockers são instâncias da imagem do Docker. Os contentores contêm todas as ferramentas necessárias para um ficheiro, de modo a que ele possa ser executado de forma isolada conforme se ilustra na figura 3.5 [Roh19].

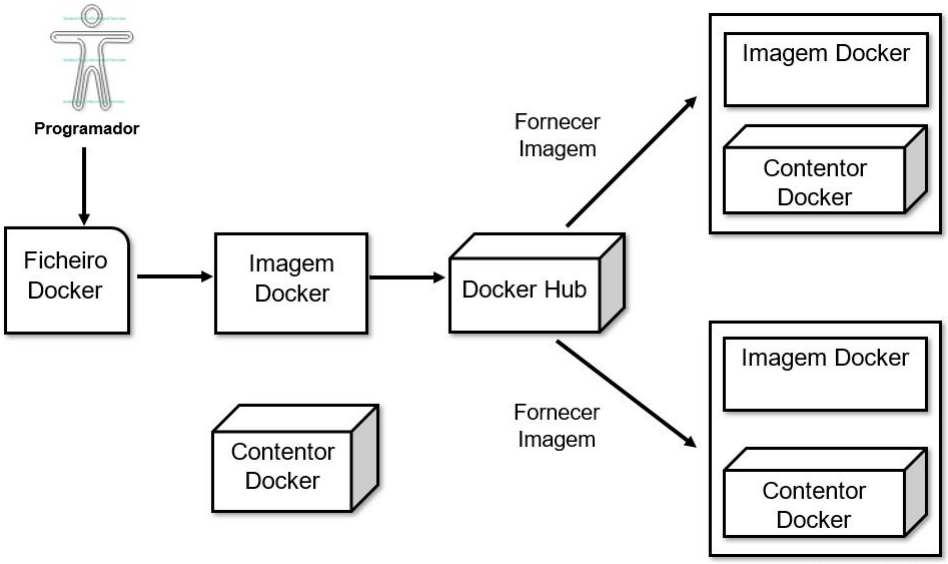

Figura 3.5: Funcionamento de um contentor do Docker (figura redesenhada a partir de [Roh19].)

3.2.3.7 Docker-*Hub*

O Docker *Hub* é um registo baseado em nuvem para repositórios de *software*, ou seja, um tipo de biblioteca para imagens do Docker. O serviço *online* é dividido num espaço público e numa área privada. O espaço público oferece aos utilizadores a capacidade de disponibilizar as suas próprias imagens e partilhá-las com a comunidade [Ion19]. Além disso, o Docker *Hub* dispõe um conjunto de imagens oficiais certificadas pelo Docker. Os utilizadores podem usar imagens oficiais como base para criar suas próprias imagens ou aplicações [Twi19a].

Os principais recursos do Docker *Hub* incluem [Doc20e]:

- **Repositórios particulares**: Permitem carregar (*upload*) e baixar (*download*) imagens de containers;
- **Construções automatizadas**: Permitem criar automaticamente imagens de contentores do *GithHub*, *Bitbucker* e envia-as para o Docker *Hub*;
- **Equipas e Organizações**: Administram o acesso a repositórios privados;
- **Imagens Oficiais**: Imagens de contentores de alta qualidade fornecidas pelo Docker que podem ser baixadas (*download*) e usadas;

- **Imagens do editor**: Imagens de contentores de alta qualidade carregadas por fornecedores externos. As imagens certificadas também incluem suporte e garantia de compatibilidade com o Docker *Entreprise*;
- ***Webhooks***: Desencadeiam ações após um envio bem sucedido (*successful push*) para um repositório para integrar o Docker *Hub* com outros serviços.

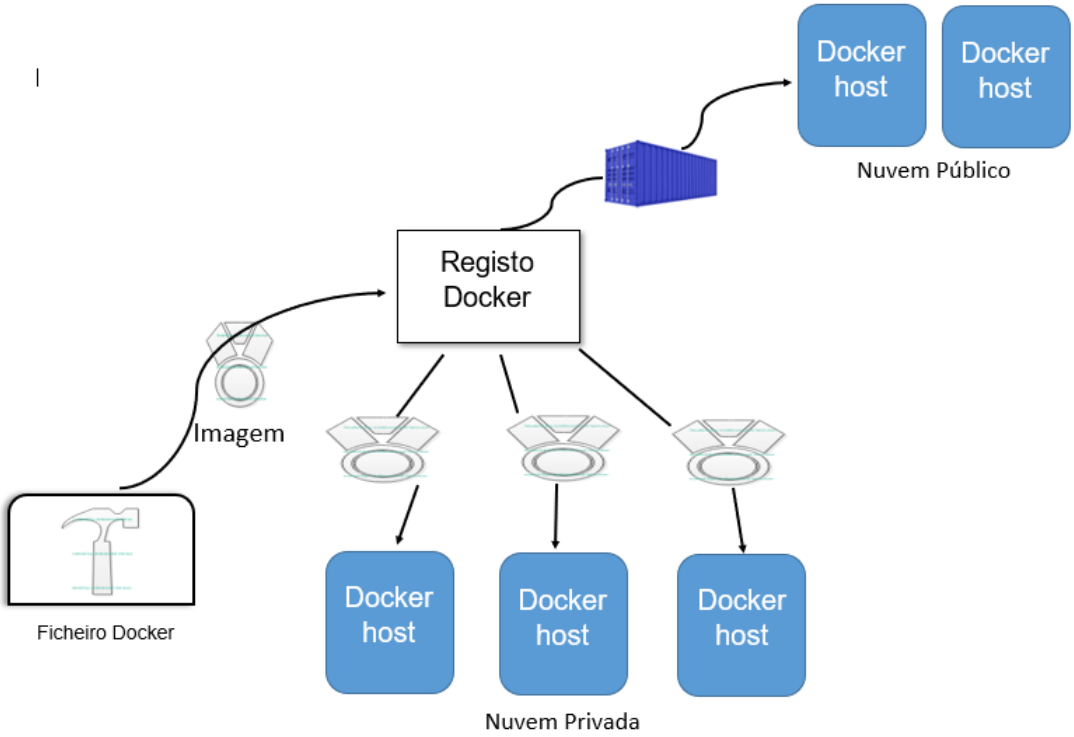

Figura 3.6: Distribuição do interior do Docker através de registros (figura redesinhada a partir de [Twi19a].)

3.2.3.8 Redes Docker

Quando criamos e colocamos a funcionar um contentor, o Docker atribui-lhe um endereço IP por omissão. Na maioria das vezes, é necessário criar e implementar redes do Docker conforme a necessidade, pelo que o Docker permite projetar a rede de acordo com os requisitos das aplicações [Roh19]. O subsistema de rede do Docker é *pluggable*, usando *drivers*. Existem vários *drivers* por omissão que fornecem as principais funcionalidades de rede [Doc20c]:

- ***bridge***: É o driver de rede por omissão (*default*). Se não for especificado um *driver*, este é o tipo de rede que será criado. As redes em *bridge* são usadas geralmente quando as aplicações são executadas em contentores autónomos que precisam de comunicar entre si.
- ***host***: Para contentores isolados (*standalone*), este driver remove o isolamento da rede entre o contentor e o *host* do Docker e usa a rede do *host* diretamente. O *host*

está disponível apenas para serviços em modo *swarm* (o modo *swarm* permite configurar vários *hosts* de modo a criar um *cluster* de elevada disponibilidade) a partir da versão do Docker 17.06.

- ***overlay***: As redes sobrepostas (*overlay networks*) conectam vários *daemons* do Docker e permitem a comunicação entre serviços *swarm*. As redes sobrepostas permitem também a comunicação entre um *cluster* de *hosts* com serviços em modo *swarm* e um contentor isolado ou entre dois contentores isolados em diferentes *daemons* do Docker. Esta estratégia elimina a necessidade de encaminhamento (*routing*) no nível do sistema operativo entre esses contentores.

- ***macvlan***: As redes **macvlan** permitem atribuir um endereço MAC (*Medium Access Control*) a um contentor, fazendo com que ele apareça como um dispositivo físico na rede. O *daemon* do Docker encaminha o tráfego para os contentores através dos respetivos endereços MAC. O *driver macvlan* pode ser a melhor opção para aplicações legadas que têm de ser conetadas diretamente à rede física, em vez de usarem encaminhamento através da pilha de rede do *host* do Docker.

- ***none***: Para um dado contentor, este *driver* desativa todas as redes, não estando disponível para serviços em modo *swarm*. Este *driver* é geralmente usado em conjunto com um driver de rede personalizado.

- ***Plug-ins*** de rede: Permite instalar e usar *plugins* de rede de terceiros com o Docker. Esses *plugins* estão disponíveis no Docker *Hub* ou através de fornecedores de equipamentos ou de serviços.

3.2.3.9 *Namespaces*

O Docker usa uma tecnologia designada por *namespaces* para fornecer um ambiente de trabalho isolado designado por contentor. Quando corremos um contentor, o Docker cria um conjunto de *namespaces* para esse contentor, os quais fornecem uma camada de isolamento. Cada aspecto de um contentor corre num *namespace* separado e o respetivo acesso é limitado a esse *namespace* [Doc20a].

3.2.3.10 Grupos de Controlo

O Docker *Engine* no Linux também se baseia numa outra tecnologia chamada *control groups* (*cgroups*). Um *cgroups* limita uma aplicação a um conjunto especifico de recursos. Os *control groups* permitem que o motor do Docker (Docker *engine*) partilhe recursos de hardware disponiveis com contentores e que, opcionalmente, lhes imponha limites e restrições. Por exemplo, o motor do Docker pode limitar a memória disponível para um determinado contentor [Doc20a].

3.2.3.11 Sistemas de Ficheiros *Union*

s sistemas de ficheiros *Union*, ou *UnionFS*, operam criando camadas e tornando-as muito leves (*lightweight*) e rápidas. O Docker *Engine* usa o *UnionFS* para fornecer os blocos de construção para os contentores. O Docker *Engine* pode usar várias variantes do *UnionFS*, incluindo *AUFS*, *btrfs*, *vfs* e *DeviceMapper* [Doc20a].

3.2.3.12 Formato de Contentor

O Docker *Engine* combina *namespaces*, *control groups* e os sistemas de ficheiros *Union* num *wrapper* designado por formato contentor, cujo formato de contentor por omissão (*default*) é o *libcontainer*. No futuro, o Docker poderá suportar outros formatos de contentor através da integração de tecnologias tais como *BSD Jails* ou *Solaris Zones* [Doc20a].

3.2.3.13 *Netfilter*

O módulo *Netfilter* do *kernel* do sistema operativo integra o *iptables*. O Docker utiliza o *iptables* para garantir o isolamento da rede dos contentores e também para que os contentores consigam comunicar entre si através de regras de encaminhamento [And18].

3.2.3.14 Vantagens do Docker

Atualmente, o Docker tornou-se popular devido aos benefícios proporcionados pelos contentores do Docker. As principais vantagens do Docker são as seguintes [Roh19].

- **Velocidade**: Os contentores do Docker, em comparação com as maquinas virtuais, são muito mais rápidos. O tempo necessário para criar um contentor é muito menor do que o de uma máquina virtual, porque os contentores são pequenos e leves. Por outro lado, o desenvolvimento, teste e implementação podem ser feitos mais rapidamente, devido ao facto de os contentores serem pequenos. Os contentores podem ser enviados para testes depois de ser criados e dos testes para o ambiente de produção.
- **Portabilidade**: As aplicações criadas dentro dos contentores do Docker são extremamente portáteis. Essas aplicações portáteis podem ser facilmente migradas para outros ambientes como um único elemento e o respetivo desempenho é semelhante.
- **Escalabilidade**: O Docker tem a capacidade de ser implementado em vários servidores físicos, servidores de dados e plataformas em nuvem. Também pode ser executado em todas as máquinas Linux. Os contentores podem ser facilmente migrados de um ambiente em nuvem para um *host* local e deste novamente para a nuvem em ritmo acelerado.

- **Densidade**:O docker usa os recursos disponíveis com mais eficiência porque não usa um hipervisor. Este é o motivo pelo qual mais contentores podem ser executados num único *host* em comparação com as máquinas virtuais. Os contentores do Docker apresentam um desempenho mais elevado devido à sua alta densidade e ao menor desperdício de recursos.

3.3 Docker *Swarm*

Um Docker Swarm é um grupo de máquinas físicas ou virtuais que correm a plataforma Docker e que foram configuradas para se juntarem num cluster. As atividades do cluster são controladas por um gestor do swarm e as máquinas que integram o cluster são chamadas de nós. Num Docker Swarm há tipicamente vários nós de trabalho (worker nodes) e pelo menos um nó de gestão (manager node), que é responsável por gerir os recursos dos nós de trabalho com eficiência e garantir que o cluster funciona com eficiência [Sum20].

3.4 Twistlock

3.4.1 Introdução

O Twistlock é uma plataforma de segurança cibernética nativa da nuvem que foi criada especificamente para garantir segurança e conformidade para hosts e contentores fornecendo proteção para as carga de trabalho (*workloads*) em qualquer ambiente de computação. O Twistlock oferece uma política de segurança do ciclo de vida em cada fase de desenvolvimento para contentores de modo a proporcionar fiabilidade, conformidade e eficiência em contentores, nomeadamente em contentores do Docker. O Twistlock preconiza regras de políticas de gestão que permitem aos utilizadores do sistema criar ou iniciar contentores, podendo o utilizador modificar essas regras mas não eliminá-las [Oli15] [Gir19].

O Twistlock também examina (*scanning*) imagens de contentores nos próprios registos. Em ambientes de tempo de execução, o Twistlock apresenta um *proxy* Docker em execução no mesmo servidor com os outros contentores de aplicação. Isso é essencialmente uma filtragem de tráfego, pelo que o contentor de aplicação que chama de *daemon* do Docker é redirecionado pelo Twistlock. Esta abordagem reforça o controlo de acesso, permitindo uma configuração mais segura onde nenhum contentor está definido para ser executado como raiz, ele facilita proteção de ambientes da *cloud* e as instâncias criado de servidor uníco [Oli15] [Gir19].

3.4.2 Função de Twistlock

O Twistlock tem como função principal fornecer uma garantia de segurança para as aplicações em contentores ou em plataformas em nuvem desde o início do ciclo de vida do contentor e implementação até à respetiva execução. O Twistlock prima em tornar todo o processo gerível, realizando uma série de testes sobre vulnerabilidades e reforçando as medidas de segurança para evitar ameaças, de modo a que seja possível garantir a conformidade dos requisitos com padrões bem definidos sobre a violação das políticas que cobrem todo o processo [Kou17] [Gir19].

3.4.3 Principais Características

A plataforma Twistlock fornece gestão de vulnerabilidades e conformidade em todo o ciclo de vida das aplicações, examinando (*scanning*) imagens e funções sem servidor (*serverless functions*) para impedir que vulnerabilides ou ameaças se transformem em problemas no *pipeline* de desenvolvimento e para monitorizar continuamente todos os registos em ambientes de produção.

Além disso, o Twistlock integra uma ferramenta de avaliação, verificação de dependências e defesa em profundidade, com controlo de acessos, autoproteção de aplicações em execução automatizada e *firewalls* nativas na nuvem para proteger aplicações contra ameaças. Apesar de a gestão da *firewall* não ser uma missão fácil para as plataformas em nuvem, o Twistlock previne a exploração de vulnerabilidades em contentores, *hosts*, e imagens de contentores [Oli15].

3.4.4 Arquitetura Twistlock

A arquitetura do Twistlock é constituida por vários componentes, incluindo o *defender*, o *intelligence stream*, a consola e aplicações em execução em contentores do Docker. O Twistlock faz a gestão de cada instância criada a partir da consola do Twistlock, a qual permite criar e remover políticas de segurança, para que seja possível estabelecer a conformidade de imagens e monitorizar o estado de segurança de cada contentor incluindo o estado do sistema de ficheiros [Eli17]. A figura 3.7 representa uma perspetiva geral da arquitetura do Twistlock.

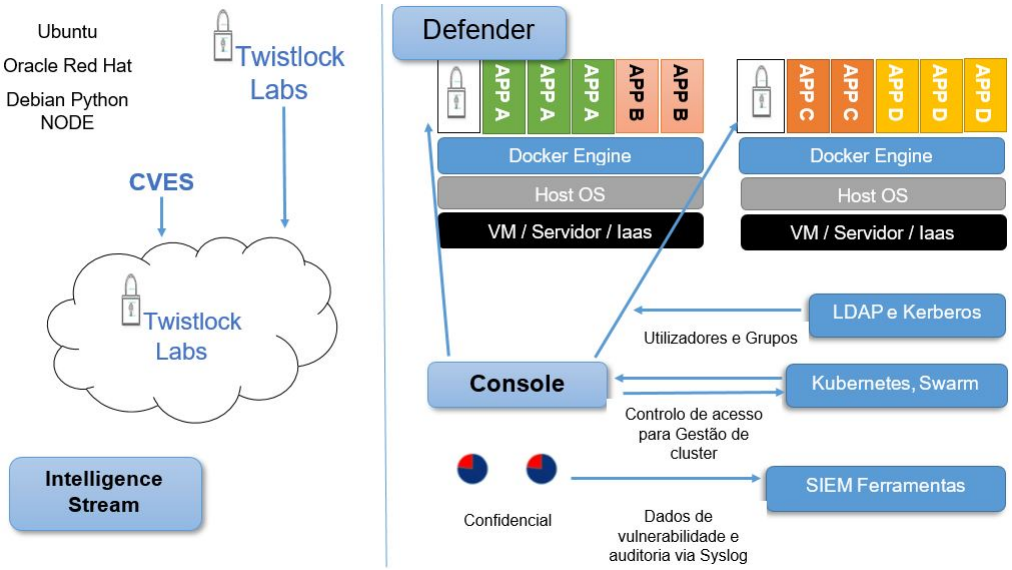

Figura 3.7: Arquitetura Twistlock (figura redesinhada a partir de [Eli17].)

3.4.5 Principais Mecanismos de Segurança do Twistlock

3.4.5.1 Defesa Durante a Execução

A defesa durante a execução proporciona uma proteção em profundidade através da automação, avaliações de rotinas e aprendizagem automática (*machine learning*) sem intervenção humana para proteger cargas de trabalho e aplicações nativas na nuvem [Rou19]. De acordo com Jim Routh [Rou19], o Twistlock é a única plataforma de segurança necessária para proteger ambientes nativos na nuvem, incluindo *hosts,* contentores, orquestrador de aplicações em contentores e aplicações sem servidor contra vulnerabilidades e ameaças ativas.

A proteção durante a execução do Twistlock usa a aprendizagem automática para criar automaticamente um modelo 4D de cada aplicação no respetivo ambiente. Estes modelos definem todos os comportamentos válidos conhecidos dos hosts e contentores, através de sensores de processo, rede, sistema de ficheiros e chamada de sistema. Estes modelos estão correlacionados com os *IDs* das imagens, pelo que quando um programador desenvolve uma aplicação, obtém um modelo calculado e personalizado exclusivamente para esse desenvolvimento especifico [Rou19].

3.4.5.2 Segurança Nativa na Nuvem

Apesar de as ferramentas tradicionais se basearem em listas estáticas que tentam prever todos os eventos maliciosos possíveis que possam acontecer, o Twistlock representa uma nova abordagem para a segurança nativa da nuvem, permitindo passar para um modo de permissão explicita, no qual apenas as atividades e os recursos específicos exigidos pela aplicação estão no modelo e os demais são tratados como anómalos e preveni-

dos. O Twistlock utiliza a aprendizagem automática para criar esses modelos automaticamente [Rou19].

3.4.5.3 Prevenção de Ameaças e Anomalias e Resposta a Incidentes em Tempo real Usando Análise Forense Nativa na Nuvem

O Twislock *Incident Explorer* identifica automaticamente padrões de ataque no ambiente e exibe-os através de uma interface clara e formatada. Ao correlacionar e analisar eventos que abrangem várias ações e sensores, o SOC (*Security Operations Center*) pode visualizar uma cadeia de eventos de forma coesa, destacando os princípais indicadores ao longo do caminho e permitindo uma resposta a incidentes mais rápida e eficaz [Rou19].

O Twistlock recolhe de forma eficiente dados forenses em todas as cargas de trabalho nativas na nuvem, à medida que são executadas antes de um incidente ocorrer, associa esses dados à defesa durante a execução e à identificação de incidentes, e armazena e partilha esses dados para análise automatica. O Twistlock integra esses dados com o *Incident Explorer* para oferecer uma visibilidade profunda do estado das aplicações e microsserviços [Rou19].

3.4.5.4 Conformidade de Contentores

O Twistlock ajuda a monitorizar e reforçar a conformidade com as melhores práticas para *hosts*, contentores e ambientes sem servidor. A utilização do Twistlock *Compliance Explorer*, que é o sistema de gestão de conformidade para impor configurações padrões, melhores práticas de segurança, uso de registos confiáveis e templates de instalação recomendados, permite aumentar a garantia de os ambientes nativos da nuvem permaneçam em conformidade com as políticas da industria e da empresa [Rou20].

À medida que as equipas de programadores desenvolvem novos recursos na nuvem, as equipas de segurança podem desconhecer ou serem incapazes de identificar repositórios não protegidos ou aplicações sem servidor. Para gestão automatica de ativos nativos em nuvens públicas, o Twistlock permite aos responsáveis pela segurança informática ou pela conformidade ver e serem alertados sobre qualquer entidade previamente desconhecida [Rou20].

O Twistlock incorpora regras explicitamente permitidas ou negadas para registos e repositórios e reaprende automaticamente o conjunto confiável de fontes com um simples *click* ou uma chamada à API. O Twistlock com o *Compliance Explorer* fornece uma visão integrada de todas as políticas do ambiente de computação, do estado atual de conformidade e dos dados históricos de conformidade [Rou20].

3.4.5.5 Integração Contínua

O Twistlock integra a segurança nos fluxos de trabalho de integração contínua (*continuous integration workflows*), de modo a permitir encontrar e corrigir problemas antes que

estes se revelem durante a produção. Os recursos de integração contínua (IC) permitem aos programadores verem o estado de vulnerabilidades cada vez que compilam o código de uma aplicação, sem terem de executar uma ferramenta separada ou usar uma interface diferente. As equipas de segurança podem definir políticas que atuam como portas (*gates*) de controlo de qualidade para garantir que apenas imagens corrigidas avançam no pipeline [Twi20a].

O Twistlock integra a segurança como parte da compilação, pelo que, com o Twistlock podem ser criadas políticas granulares que fornecem controlo preciso sobre cada trabalho de IC. O Twistlock pode ser usado para impor requisitos específicos, como por exemplo, bloquear compilações de código com vulnerabilidades com classificação CVSS (*Common Vulnerability Scoring System*) média ou elevada para as quais existe um *patch* do fabricante. O Twistlock permite ao programador certificar-se que apenas as imagens que satisfaçam os respetivos requisitos de segurança são assinadas e enviadas para o registo [Twi20a].

Como parte da examinação (*scanning*) de cada compilação através do *plugin Jenkins* ou *twistcli*, o Twistlock fornece resultados tanto nas ferramentas de desenvolvimento nativas como na interface do utilizador do Twistlock. O Twistlock segue as vulnerabilidades que tenham sido descobertas em cada compilação em cada camada, a partir de um local centralizado [Twi20a].

3.4.5.6 Firewalls Nativas na Nuvem

A plataforma Twistlock capacita as equipas de segurança para além da gestão manual de endereços IP (*Internet Protocol Address*) na lista de permissões, oferecendo firewalls criadas para ambientes nativos na nuvem. O Twistlock fornece *firewalls* nas camadas 4 e 7 do modelo OSI (*Open Systems Interconnection*) que aprendem automaticamente a topologia de rede das aplicações e fornece uma microsegmentação, desenhada à medida, para todos microsserviços [Twi20c].

Conforme referido em [Twi20c], a automação é o principal facilitador *(enabler*) das *firewalls* do Twistlock. A plataforma Twistlock mapeia, identifica e permite fluxos de tráfego válidos nos respetivos ambientes, de forma automática, com base no conhecimento acerca do comportamento desses fluxos de tráfego. O Twistlock cria dinamicamente filtros que, automaticamente, permitem conexões válidas e rejetitam conexões suspeitas, independentemente da localização do contentor no *cluster*.

O CNAF (*Twistlock Cloud Native Applicarion Firewall*) do Twistlock combina o seu conhecimento com a localização e visividade de um determinado ambiente para filtrar automaticamente o tráfego *web*. Com o CNAF, o Twistlock identifica o local onde as aplicações estão em execução e redireciona automaticamente o tráfego de entrada através do Twistlock Defender e aplica um filtro da camada 7 otimizado e específico para essa aplicação, enviando apenas o tráfego filtrado para o contentor onde está instalada a aplicação [Twi20c].

O CNNF (*Cloud Native Network Firewall*) é uma *firewall* da camada 4 para microsserviços baseada em aprendizagem automática (*machine learning*) que funciona em qualquer nuvem ou orquestrador de contentores. O CNNF protege os *hosts* e contentores e fornece um motor de políticas granular para a empresa com features de gestão copreensíveis [Twi20c].

O Twistlock Radar também fornece uma visão das conexões em tempo real do ambiente de computação, integrada com a informação sobre vulnerabilidades, conformidade e execução, de modo a que seja possível visualizar facilmente a topologia e o estado de segurança do ambiente de computação [Twi20c].

3.4.5.7 Controlo de Acessos

O Twistlock permite estabelecer e monitorizar as medidas de controlo de acessos para cargas de trabalho e aplicações nativas na nuvem. O Twistlock protege os respetivos *hosts* subjacentes, o Docker e o Kubernetes, integrando o IAM (*Identity and Access Management*) e as ferramentas de gestão de *passwords* e segredos, em conjunto com outras tecnologias nucleares [Twi20b].

O Twistlock permite realizar auditorias de elevada fidelidade em tempo real para operações sensíveis ou confidenciais em todos os seus hosts através de um único painel de controlo, incluindo acesso ao *sudo*, ao *sshd* e à API do Docker. O Twistlock também vigia tipos de acessos específicos a ficheiros e caminhos (*paths*) para cumprir o objetivo da conformidade [Twi20b].

O Twistlock pode receber ficheiros de log de segurança de qualquer aplicação ou serviço, incluindo *sshd*, *sudo* e *nginx*, analisá-los (*parse*) para eventos com interesse e enviar alertas para os fluxos de trabalho de segurança existentes [Twi20b].

O RBAC (*Role-based Access Control*) é um método de acesso baseado na definição das funções dos funcionários e dos privilégios correspondentes na organização [EKR19]. O Twistlock inclui sete funções separadas para fornecer privilégios de acesso diferenciados aos seus *devops* e equipas de segurança. O Twistlock usa coleções atribuídas para controlar com precisão o que as equipas podem ver ou para isolar de forma segura e multitenant unidades lógicas, como por exemplo unidades de negócio ou unidades geográficas [Twi20b], conforme se ilustra na figura 3.8.

Os princípais componentes de uma estratégia baseada em funções (*roles*) para controlo de acessos são [EKR19]:

- **Utilizador** :um indivíduo com UID (*User Identification*) com acesso a um sistema.
- **Função**: uma função nomeada (indica o nível de autoridade).
- **Permissão**: equivalente a direitos de acesso.
- **Sessão**: um mapeamento entre um utilizador e um conjunto de funções às quais o utilizador está atribuido no contexto do horário de trabalho (expediente).

- **Objecto**: um recurso do sistema que requer permissão para acesso.
- **Operação**: qualquer ação na rede protegida.

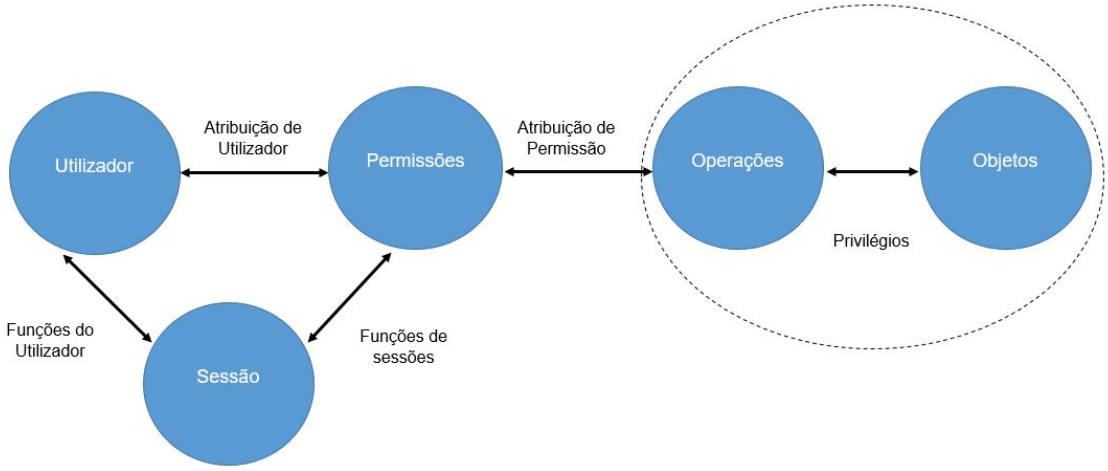

Figura 3.8: Esquema de Operação RBAC (figura redesenhada a partir de [EKR19].)

3.4.5.8 A Gestão de Vulnerabilidades

A gestão de vulnerabilidades identifica e evita vulnerabilidades em todo o ciclo de vida da aplicação e dá prioridade à mitigação de riscos para ambientes nativos na nuvem. A segurança desde o desenvolvimento até à produção incluiu a gestão de vulnerabilidades em qualquer processo de IC, enquanto continuamente monitoriza, identifica e previne riscos a *hosts*, imagens de contentores e funções no ambiente de computação. O Twistlock combina a detenção de vulnerabilidades com inteligência e conhecimento atualizados sobre as aplicações e instalações em execução para dar proridade a riscos especificos para o ambiente de computação [Twi20d].

O Twistlock apresenta-se como a única ferramenta de gestão de vulnerabilidades necessária para criar e implementar com segurança aplicações nativas na nuvem. A imposição automatizada e personalizada de políticas permite um controlo completo em todas as fases do ciclo de vida das aplicações. O Twistlock mostra as vulnerabilidades para cada compilação na interface do utilizador [Twi20d].

O Twistlock *Intelligence Stream* procura e agrega informação sobre vulnerabilidades para fornecer os dados mais atuais para todas as camadas da pilha da nuvem. O Twistlock combina esses dados com o conhecimento das implementações, tais como quais os contentores com ligação à Internet, quais os contentores que correm com previlégios elevados ou quais os contentores que têm outras restrições de segurança, de modo a permitir ver quais as vulnerabilidades que são mais críticas num ambiente específico [Twi20d].

3.5 Conclusões

Este capítulo foi dedicado a duas tecnologias: o Docker e o Twistlock. Em relação ao Docker, foi abordada a contentorização baseada na virtualização ao nível do sistema operativo, a arquitetura do motor do Docker (Docker *Engine*), os principais componentes do Docker, assim como as principais vantagens do Docker.

Foi também abordada a plataforma de segurança Twistlock, a qual oferece uma politíca de segurança desde o ciclo de vida em toda fase de desenvolvimento até à fase de produção e lida com a examinação (*scanning*) de imagens de contentores nos próprios registos, as funções (*roles*) desempenhadas pelos funcionários e as características dessas funções. Este capítulo apresentou uma perspetiva geral sobre a arquitetura do Twistlock e descreveu os príncipais mecanismos de segurança que o Twistlock oferece para proteção contra vulnerabilidades e ameaças ao ambiente de computação e para reforçar o cumprimento com as conformidades e com as melhores práticas para *hosts* e imagens de contentores.

Este capítulo, dedicado ao Docker e ao Twistlock, abordou as tecnologias centrais usadas no trabalho conducente a esta dissertação de mestrado.

Capítulo 4

Ambiente Experimental e Avaliação do Desempenho do Twistlock sobre Contentores Docker

4.1 Introdução

Este capítulo pretende apresentar uma descrição do ambiente experimental usado para realizar os testes para avaliar o impacto no desempenho do Twistlock sobre contentores do Docker. São apresentados e explicados alguns requisitos de instalação do Twistlock, as configurações de ambiente experimental realizado, as plataformas utilizadas. É apresentada uma comparação do desempenho de contentores do Docker com e sem Twislock, usando a plataforma Sumo Logic para coleta de dados referentes ao desempenho.

4.2 Implementação do Ambiente Experimental

4.2.1 Caracterização do Ambiente Experimental

O ambiente experimental é constituído por um conjunto de dois computadores físicos e uma maquina virtual, com as especificações indicadas na Tabela 4.1. Os dois computadores nos quais foram instalados e configurados o Twistlock, a partir da API *Token* da plataforma Prisma Cloud para análise de vulnerabilidades e de conformidade, e a plataforma Sumo Logic nos permitiu coletar dados para análise do desempenho de contentores com e sem Twislock. Os dois computadores possuem as mesmas características técnicas, mas com sistemas operativos diferentes, um com CentOS 7 Workstation e o outro com Ubuntu 18.04 Workstation. No computador com Linux centOS 7 (k8s-master) foi instalada uma máquina virtual na qual, após instalação do Docker, tendo o utilizador todas as permissões para executar comandos do Docker, configurámos o Docker swarm com os nós de *cluster*: ManagerO, Worker1 e Worker2.

Todos os computadores físicos estão ligados à rede informática do Laboratório 6.10 do Departamento de Informática da UBI (endereços IP 10.0.5.33 e 10.0.5.34) de modo a poderem comunicar entre si. As plataformas Prisma Cloud e Sumo Logic foram instaladas de forma paralela em cada computador. No entanto, o Sumo Logic foi o primeiro a ser instalado de modo a permitir recolher os dados relativos ao desempenho de contentores sem Twistlock, pelo que só a seguir é que foi instalado a plataforma Prisma Cloud.

De modo a analisar e ilustrar o funcionamento da plataforma Prisma Cloud/Twistlock, incluindo a análise de vulnerabilidades e a análise de conformidade de *hosts* e contentores de imagens no nível de *cluster* de *swarm*, foi considerado o ambiente experimental representado esquematicamente na figura 4.1, o qual inclui a plataforma Prisma Cloud/Twistlock sem a plataforma Sumo Logic. O ambiente experimental ilustrado na figura 4.2 inclui as plataformas Prisma Cloud/Twistlock e Sumo Logic, em que a plataforma Sumo Logic é usada para recolher dados sobre o impacto do uso da plataforma Prisma Cloud no desempenho contentores em Docker. O ambiente experimental ilustrado na figura 4.3 inclui a plataforma Sumo Logic sem o Prisma Cloud/Twistlock, de modo a recolher dados sobre o desempenho de contentores em Docker, sem usar a plataforma Prisma Cloud/Twistlock.

Tabela 4.1: Especificações de *hardware* e *software*

Tipo de hardware e software	Especificações
CPU	Intel (R) Core i7-8700 3.20 GHz (x2)
Memória RAM	RAM 32.0 GB (Giga Bytes) (x2)
Disco	HDD 1T (TeraBytes) (x2)
Máquina Virtual /Docker Swarm	Linux Centos 7.7.1908/x86_64 Docker Swarm instalado sobre host Linux Centos 7.7.1908
Sistemas Operativos e Máquina Virtual	Linux Ubuntu 18.04/x86_64 Linux Centos 7.7.1908/x86_64
Plataformas	Docker Engine : CE 19.03.5 Twistlock : Private:defender_19_11_480 Sumo Logic : SumoColletor_Linux_amd64_19_288-10.sh Prisma Cloud : Versão 19.11.480

O Docker *Swarm* permitiu criar, na máquina virtual, um *cluster* com três nós, em que dois nós do Swarm, designados por *Worker1* e *Worker2*, são controlados por um nó principal *Manager0*. A máquina virtual foi instalada sobre o Linux *centOS7* no computador k8s-master.

O outro computador com Linux Ubuntu foi usado para criar vários contentores que permitiram realizar diferentes testes, verificar o estado de conformidade e vulnerabilidades com Twistlock ao nível dos *clusters* de forma profunda através da plataforma Prisma Cloud. Para tal instalámos a API *Token* da Prisma Cloud que contém as configurações predefinidas do Twistlock *Defender*, que é fornecido pela Prisma Cloud para poder coletar todos os dados dos respetivos computadores afim de serem analisados e visualizados a partir da plataforma Prisma Cloud.

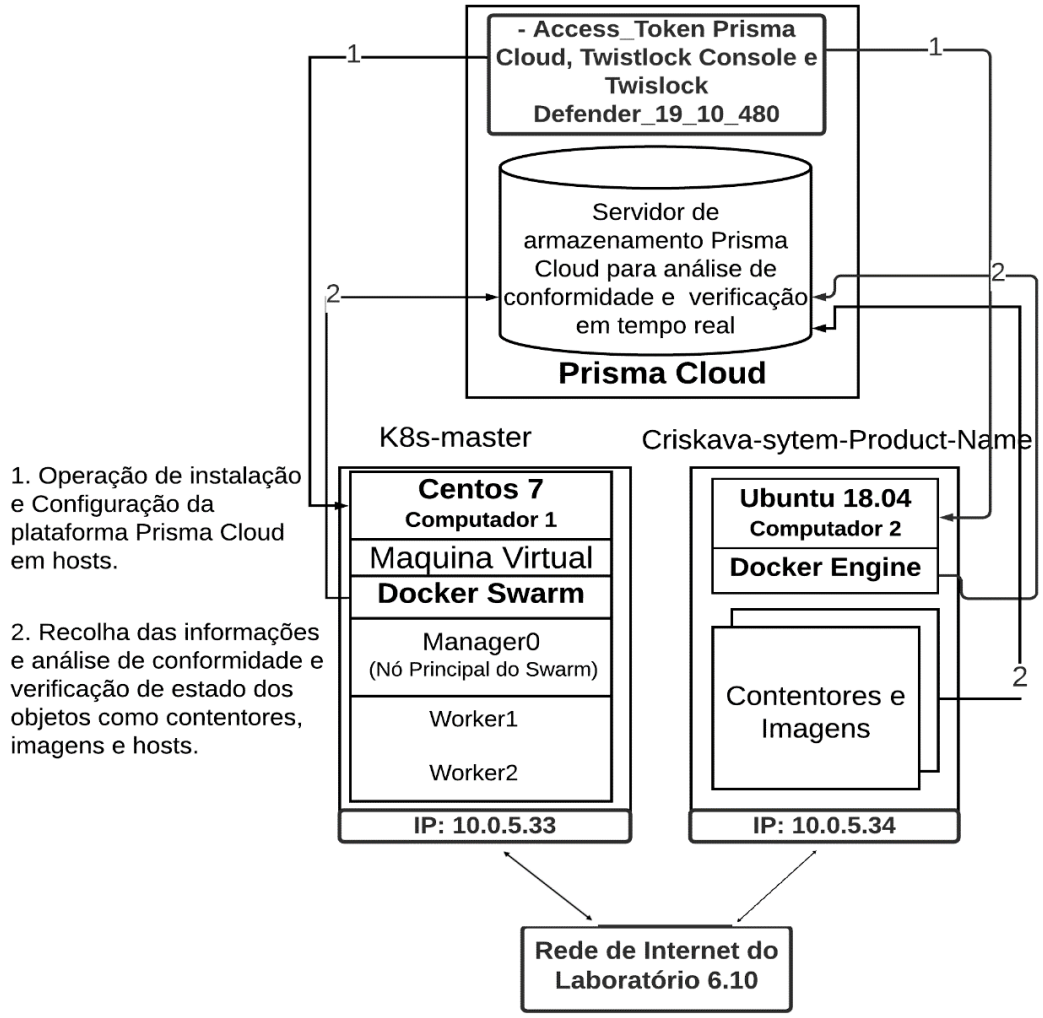

Figura 4.1: Representação esquemática do ambiente experimental com a plataforma Prisma Cloud e sem Sumo Logic.

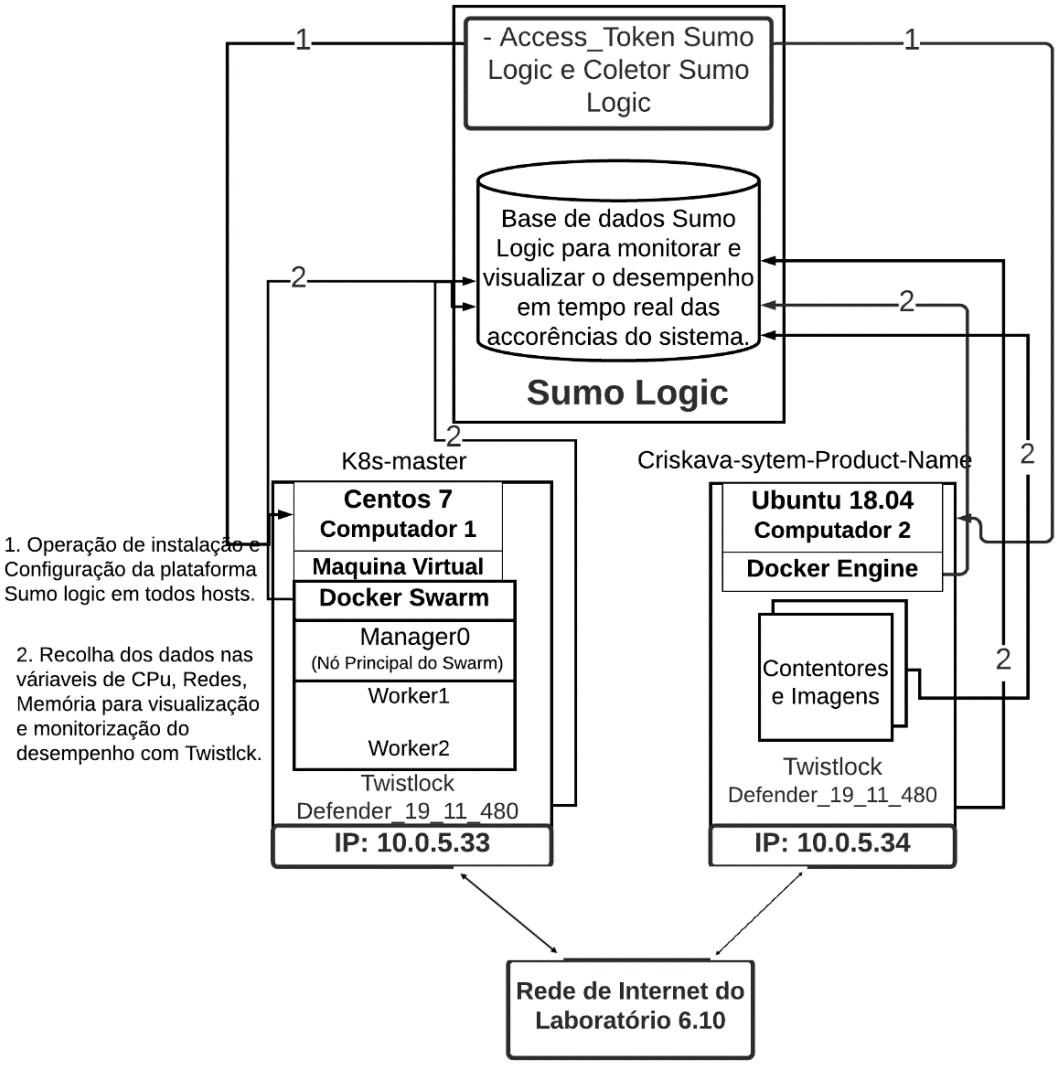

Figura 4.2: Representação esquemática do ambiente experimental com as plataformas Sumo Logic com Twistlock.

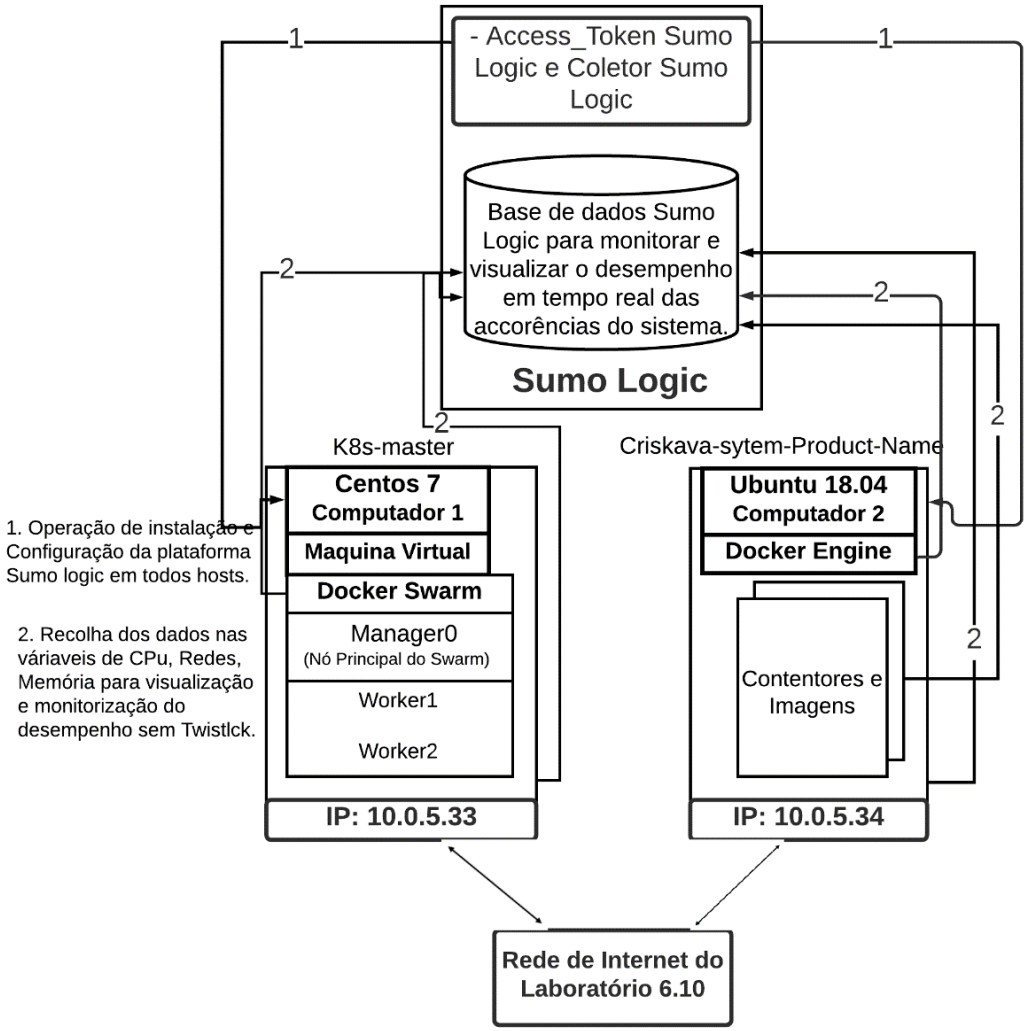

Figura 4.3: Representação esquemática do ambiente experimental com a plataforma Sumo Logic (sem Twistlock).

4.2.2 Instalação e Configuração do Twistlock

O *software* Prisma Cloud consiste em dois componentes: Twistlock *Console* e Twistlock *Defender*. A respetiva instalação é feita em dois passos, primeiro instala-se o *Console* e, de seguida, instala-se o *Defender*. O *Console* é a interface de gestão do Prisma Cloud que define as políticas e monitorização do seu ambiente, a qual é entregue como uma imagem de contentor eo Twistlock *Defender* conecta-se ao *Console* através de um *websocket* pela porta 8084 para recuperar políticas e enviar dados. O diagrama 4.4 da figura mostra as princípais conexões do Twistlock *Defender* e do Twistlock *Console* [Hos20].

Figura 4.4: Diagrama de Conexões (figura redesenhada a partir de [Hos20]).

O Twistlock *Defender* protege o seu ambiente de acordo com as políticas definidas no *Console*, o que depende de cada organização. O *Defender* é instalado em todos os *hosts* que executam contentores. A instalação do Prisma Cloud é simples e rápida num *host* único e independente, mas é necessário um registo na plataforma para poder controlar o seu ambiente através do Prisma Cloud [Hos20].

Para que a instalação seja feita de forma completa, o Prisma Cloud é configurado no *host* através da *API*, a qual fornece o certificado de instalação do utilizador através de um *script* de instalação incorporando uma chave privada de utilizador e o *Access_Token* para ser digitado na linha de comando de *Linux*. o Access_token contém o pacote de instalação do Twistlock/private:*defender*_19_11_480 com um prazo de validade. Isto estabelece a comunicação entres todos objectos e outros componentes com a plataforma de maneira a identificar e coletar dados do sistema e análisar em que ponto se encontram o estado do ambiente, afim de revelar o nivel de conformidade e de vulnerabilidades [Hos20].

Para instalar o Twistlock, conforme se indica na figura 4.5, deve ter o Access_Token (chave de acesso), que é fornecido pela Prisma *Cloud* como certificado de instalação do utilizador, e se conectar como root na máquina Linux. A seguir, deve executar os seguintes passos:

- Passo 1. Se estiver o Acess_Token Prisma *Cloud* em maiúscula deve converter todos caracteres em minúsculo porque o Cliente Docker exige que os repositórios estejam minúsculo [Hos20].

  # echo <ACCESS_TOKEN> | tr ' [:upper:] ' ' [:lower:] '

- Passo 2. Puxar a imagem do Console do registo do Prisma *cloud*

  # docker *pull registry-auth.twistlock.com*/tw_<ACCESS_TOKEN>/twistlock/ *console:console*_19_11_480

- Passo 3. Puxar imagem do Defender do registo do Prisma Cloud

  # docker *pull registry-auth.twistlock.com*/tw_<ACCESS_TOKEN>/twistlock /*defender:defender*_19_11_480

- Passo 4. Apresentar tela da confirmação de instalação de Twistlock *Defender* com versão do Twistlock instalado.

Figura 4.5: Instalação Twistlock *Defender* no *host Linux* Ubuntu.

4.2.3 Sumo Logic

O Sumo Logic [Sum20] é uma plataforma multiserviço nativa da nuvem que permite tomar decisões orientadas por dados, permitindo reduzir o tempo para investigar problemas operacionais e de segurança. A plataforma ajuda a coletar e centralizar a agregação de dados na pilha e no *pipeline* e permite monitorar e visualizar esses dados através dos paineis personalizaveis.

O Sumo Logic é instalado através do Coletor Sumo Logic (ver figura 4.6), executado a partir da linha de comando do *Linux*. O Sumo Logic atribui, de forma secreta para cada utilizador do sistema, ou seja logo na criação da conta, um <*access_id*>, um <*access_key*> e, para adicionar as maquinas na plataforma, ele fornece o *access_token* de forma automatica [Sum19]. Pode fazer-se o download do Coletor a partir da página **Gestão de dados** do Sumo Logic > **Coleção** > **Coleção** ou através dos seguintes passos:

- Passo 1. Fazer o download do compilador, escolher entre os URL estátiscos de 32 ou 64 bits para as compilações mais recentes do Coletor Linux e selecionar o *pod* apropriado do Sumo Logic.

- Passo 2. Alterar as permissões para permitir que o ficheiro seja executado:

  # *chmod* 774 SumoCollector_*linux*_amd64_19_288-10.sh

- Passo 3. Como root, executar a instalação usando as seguintes instruções:

  # ./SumoCollector_*linux*_amd64_19_288-10.sh –q–VskipRegistration=true

  –Vephemeral=true –Vsources=/path/to/*sources.json* –Vsumo.*accessid*=<*access_id*>-Vsumo.*accesskey* =<*access_key*>

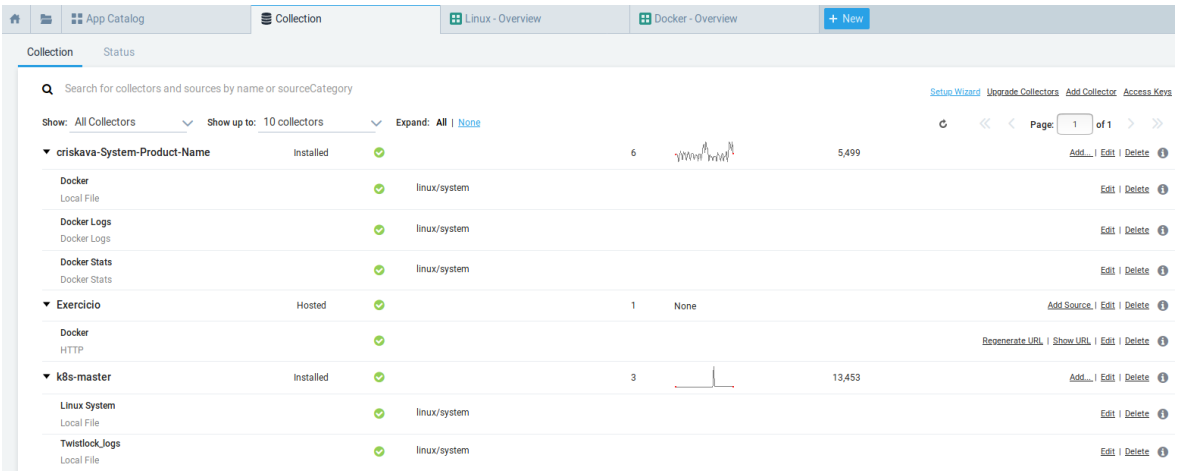

Figura 4.6: Colector Sumo Logic.

4.3 Exemplo Ilustrativo do Funcionamento da Plataforma Prisma Cloud Sobre o Docker

4.3.1 Radar

O Radar é um ambiente que utiliza o Twistlock para monitorar e visualizar a conectividade entre microsserviços, fornecendo aprendizagem automática da topologia de rede às aplicações, e criar, de forma dinâmica, um filtro que admite automaticamente ligações válidas e suprime as ligações duvidosas, independemente do local onde os contentores estão a ser executados no *cluster*, a fim de realizar uma analise de vulnerabilidades por camada e avaliar a conformidade [HD19], [Mel19], conforme se ilustra nas figuras 4.7 e 4.8.

Figura 4.7: O Radar permite visualizar em tempo real as conexões dos contentores.

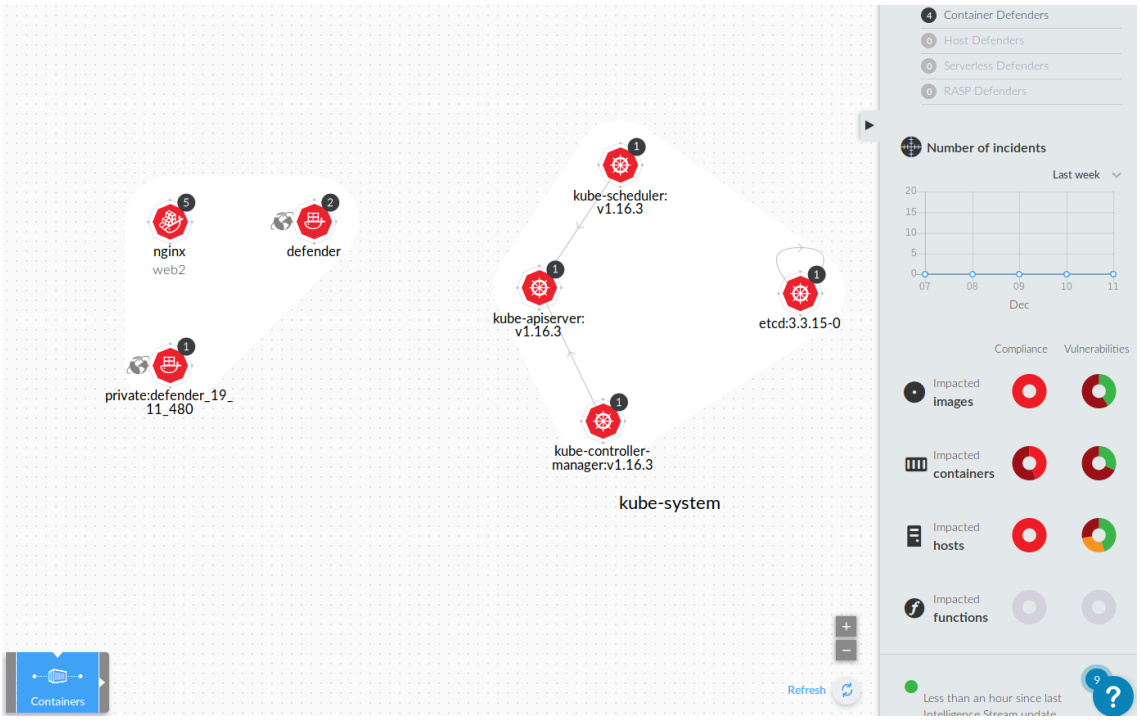

Figura 4.8: Visualização da conformidade de conexões com o Radar.

Importa sublinhar que cada nó é representado por uma cor com base na gravidade da vulnerabilidade ou do problema de conformidade, em conformidade com as políticas de segurança implementadas ou definidas para as vulnerabilidades e conformidade [Mel19]. O código de cores ajuda a identificar problemas, de forma rápida, conforme se ilustra na figura 4.9. A descrição do código de cores é resumida na tabela 4.2.

Tabela 4.2: Descrição de Código de cores no Radar(adaptado de [HD19])

Código de cores	Descrição de cores
Vermelho escuro	**Alto risco, uma ou mais vulnerabilidades de gravidade crítica**
Vermelho	**Vulnerabilidades de alta gravidade detectadas**
Laranja	**Vulnerabilidades médias detectadas**
Verde	**Nehuma vulnerabilidade detectada**

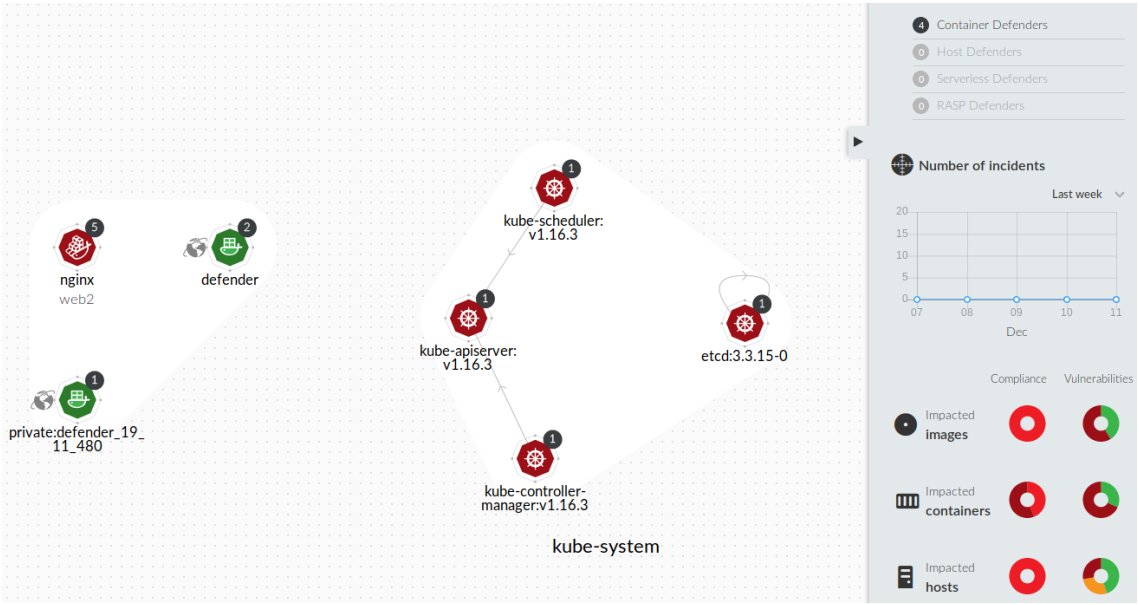

Figura 4.9: Ilustração da cor das vulnerabilidades no Radar.

O Radar permite a visualização dos contentores, dos *hosts* e das conexões. Na exibição do contentor, cada imagem com execução de contentores é representada como um nó no esquema. Na visualização do *host*, cada serviço *systemd* ou aplicação é caracterizado como um nó no esquema. Para obter a visão geral do ambiente é preciso instalar o Twislock *Defender* em todos os *hosts* [HD19], conforme se mostra na figura 4.10.

Figura 4.10: Cor das vulnerabiliades de Radar com as conexão no *Host*.

O estado da execução de cada contentor é representado por uma bola, no qual a bola azul simboliza que o contentor ainda está no estado de aprendizagem. A bola preta indica o modelo de contentor ativado e o simbolo do globo indica que um contentor pode aceder à Internet [HD19], conforme ilustrado nas figuras 4.9 e 4.10.

4.3.2 *Explorador de Vulnerabilidades*

O Explorador de Vulnerabilidades (*Vulnerability Explorer*) analisa, de forma profunda, os dados dentro do contexto do ecossistema, evitando vulnerabilidades em todo o ciclo de vida da aplicação, e exibe o resultado detalhado no formato do sistema CVE (*Common Vulnerabilities and Exposures*). Para cada tipo de objeto (contentor, imagem, *host*), o gráfico reporta uma contagem das vulnerabilidades com maior gravidade em cada classe de objetos em função do tempo [HD19], conforme se ilustra nas figuras 4.11 e 4.12.

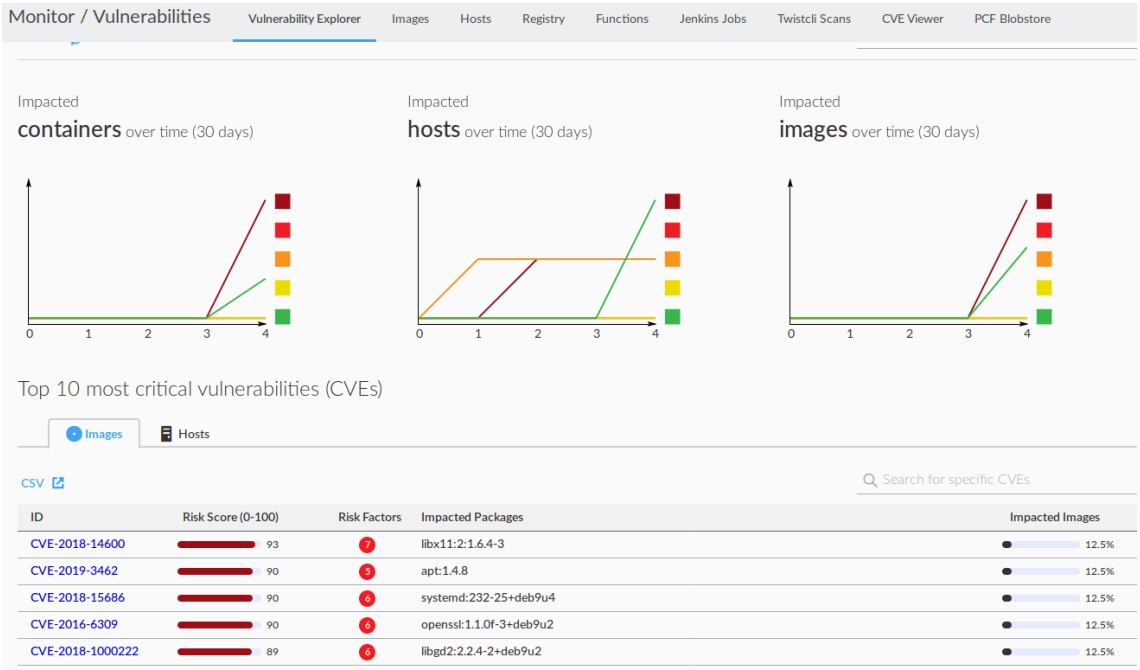

Figura 4.11: Ilustração dos resultados fornecidos pelo Explorador de Vulnerabilidades.

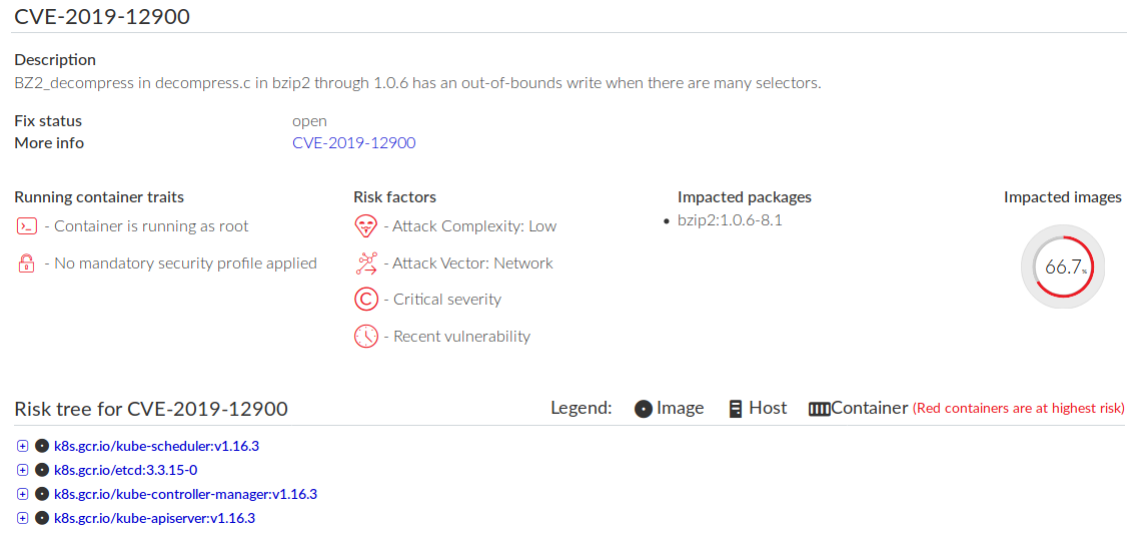

Figura 4.12: Descripção de ID do CVE.

O Explorador de Vulnerabilidade do Twistlock fornece a informação num único local, um painel, para que se percecione onde o sistema está mais vulnerável. Ele ajuda a compreender o risco no nível de contentor, imagem ou *host*. Os totais agregados dão uma ideia de como o risco é distribuído e onde existem vulnerabilidades. Os gráficos de análise de tendência contribuem para ver como estão a ser eliminadas as *CVEs* ao longo do tempo. O Twistlock mostra as dez princípais vulnerabilidades mais criticas da imagem ou *host*, classificadas de acordo com a pontuação de risco do Twistlock, incluindo o *Twistlock Risk Score* para ajudar a identificar e prioritizar a atenuação das *CVEs* mais criticas. O Explorador de Vulnerabilidade do Twistlock fornece informação detalhada sobre a quantidade de vulnerabilidades existentes no ambiente de contentores [Lew17].

4.3.3 Fatores de Risco

Os fatores de risco são programados para definir a pontuação de uma vulneranilidade, com o proposito de exibir as pontuações com maior ameaça ou risco usando as cores que definem o grau de vulnerabilidade examinado de maneira detalhada [HD19], tal como apresentam as figuras 4.13, 4.14 e 4.15.

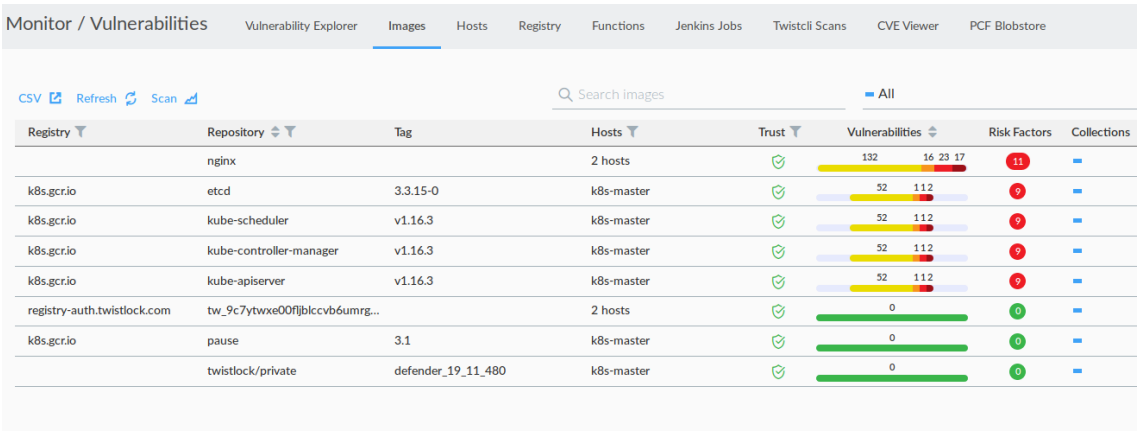

Figura 4.13: Factores de risco das imagens.

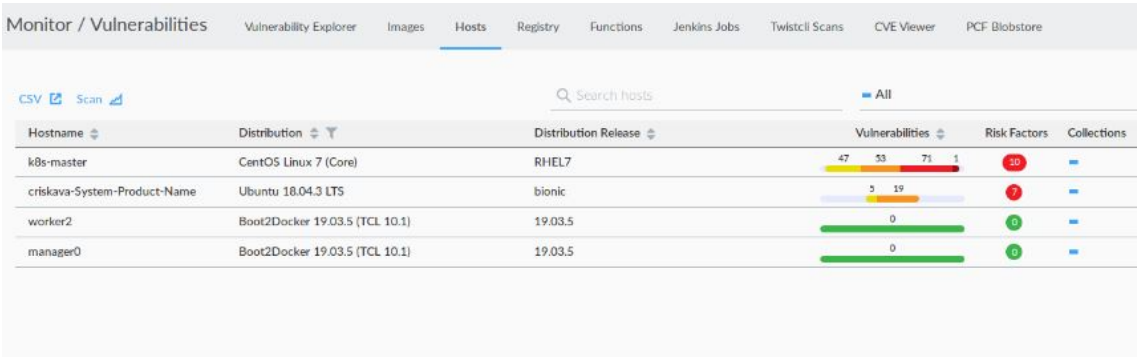

Figura 4.14: Factores de risco dos *hosts*.

Tabela 4.3: Código de cores para Vulnerabiliades e Conformidades (adaptado de [HD19]).

Código de cores	Descrição de cores
Vermelho escuro	Alto risco, uma ou mais vulnerabilidades de gravidade crítica
Vermelho	Vulnerabilidades e Conformidades de alta gravidade detectadas
Laranja	Vulnerabilidades e conformidades médias detectadas
Verde	Nehuma vulnerabilidade e conformidades detectada
Amarelo	Vulnerabilidades e Conformidades baixa

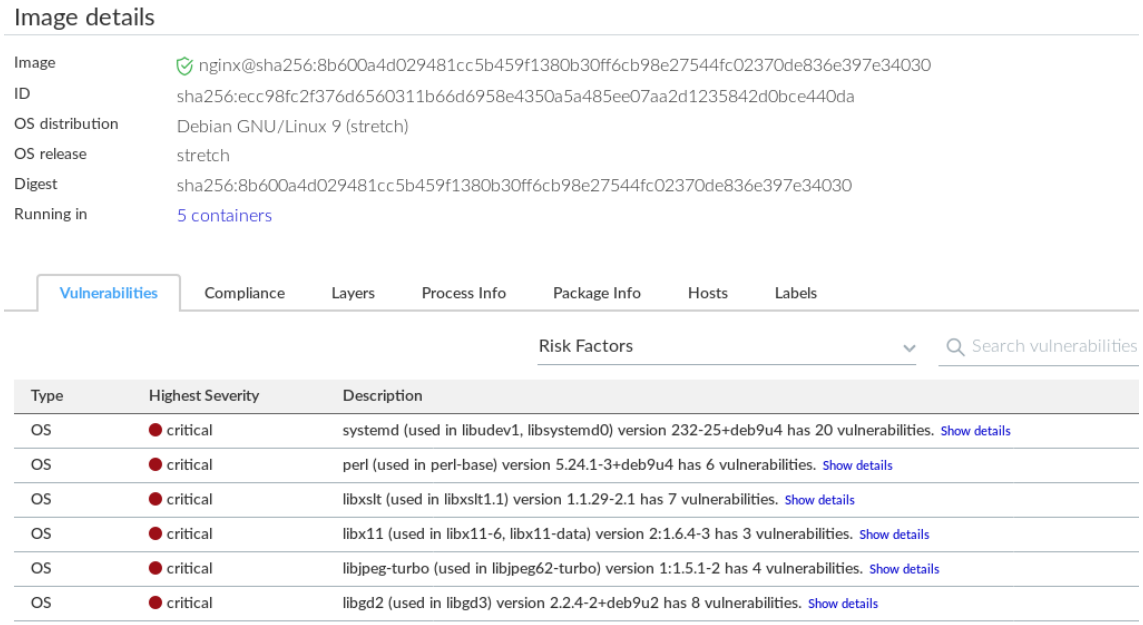

Figura 4.15: Fatores de risco com descrição de cada Vulnerabilidades.

4.3.4 Regras de Gestão de Vulnerabilidades

As politicas definidas de forma equilibrada ou discplinada descrevem as praticas a serem tomadas quando as vulnerabilidades são encontradas em recursos do ambiente de computação. Essas políticas incluem os relatórios de varrimento e visualizações de radar [HD19].

O Twistlock integra a verificação de vulnerabilidades a qualquer ferramenta de integração contínua e disponibiliza todos os dados em formatos abertos como *CSV*, *JSON* e até *syslog*. Ele automatiza todos os aspectos da digitalização com uma API abrangente que facilita a integração do Twistlock com as restantes ferramentas de integração contínua e disponibilização contínua. Isto permite que as equipas de segurança meçam os riscos ao longo do tempo com gráficos e linhas de tendência claras, à medida que os problemas são identificados e corrigidos [Rou19], conforme se ilustra nas figuras 4.16 e 4.17.

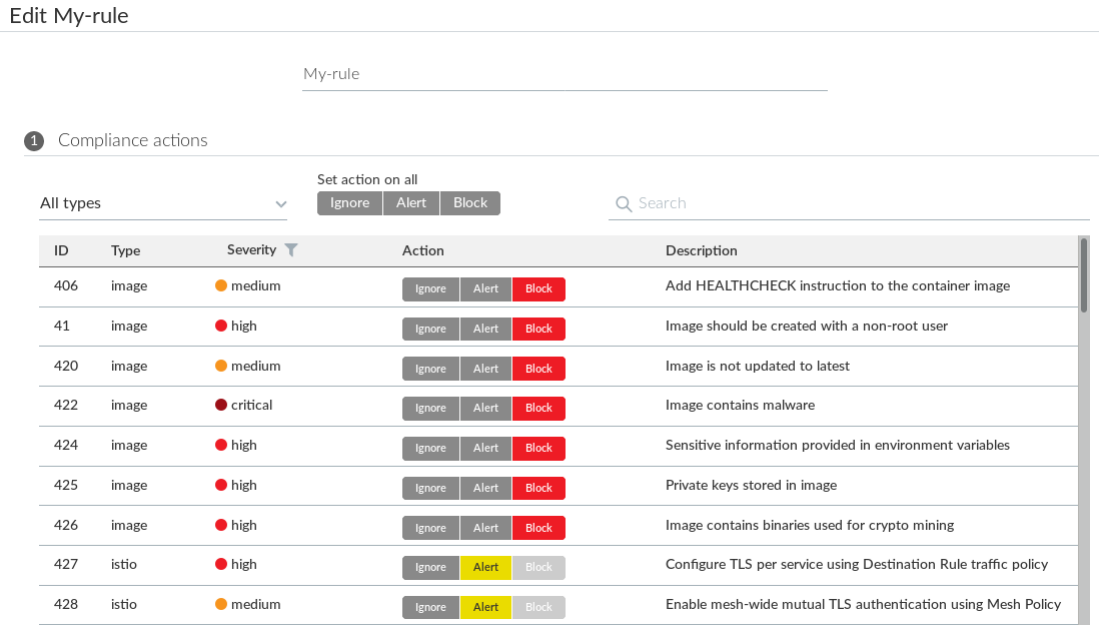

Figura 4.16: Aplicação das Regras de Gestão de Conformidade de ambiente.

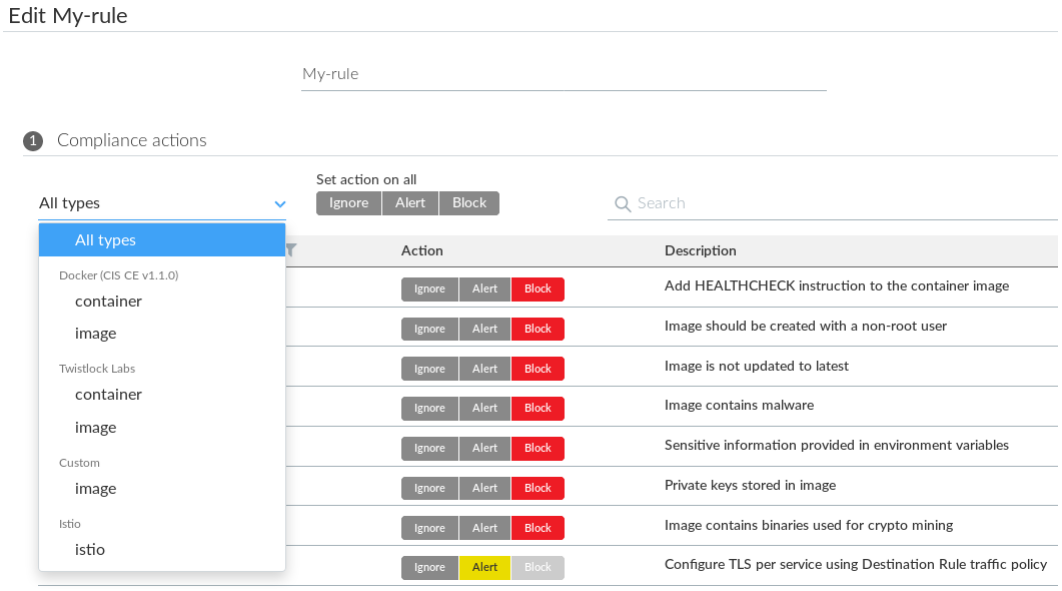

Figura 4.17: Tipos de Objectos para Gestão de Regras.

4.3.5 Análise de Vulnerabilidades Por Camada

A Análise de Vulnerabilidades Por Camada permite simplificar a percepção de como as imagens são estruturadas e quais os componentes que têm vulnerabilidades, na tentativa de identificar de forma automatizada e personalizada os pontos vulneráveis em todo o estágio do ciclo de vida da aplicação, junto com uma interface de utilizador para correção, conforme se ilustra na figura 4.18.

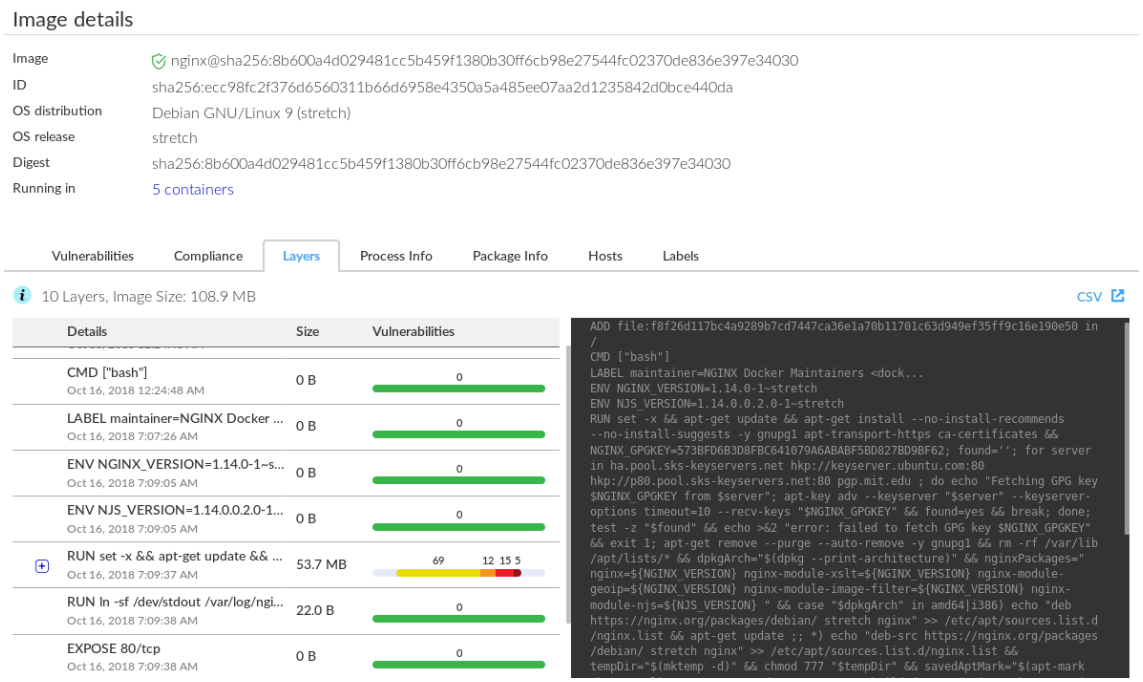

Figura 4.18: Análise de Vulnerabilidades por Camada.

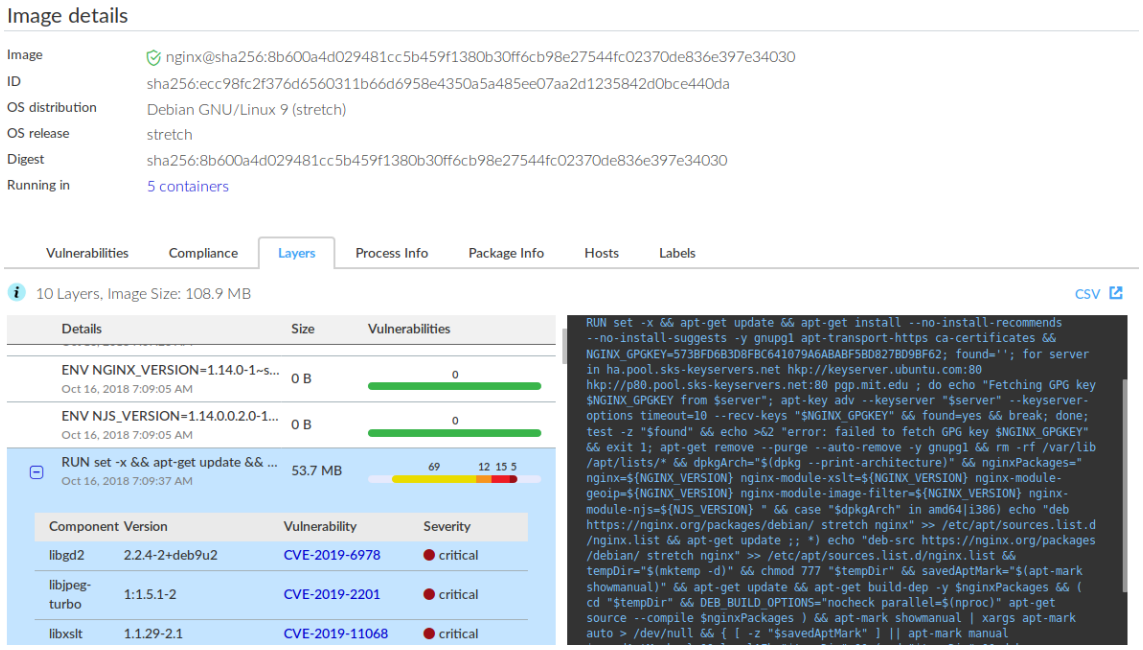

Figura 4.19: Instruções utilizadas em cada camada e descrição da imagem.

A figura 4.19, exibe a pesquisa de vulnerabilidades acionáveis e detalhadas para cada *build*, na mesma interface de utilizadores e permite digitalizar a imagens e funções em qualquer registo e em qualquer lugar [Tho17].

4.3.6 Explorador de Conformidade

O Explorador de Conformidade (*Compliance Explorer*) fornece uma compreensão única

de conformidade de todos os objetos no ambiente, incluindo o estado atual e os históricos da conformidade. O Explorador de Conformidade mostra a conformidade de todos objetos (imagens, *hosts* e contentores) em percentagem do total de verificações definidas e salienta todos os objectos não definidos de forma facilmente visualisável, apresentando os últimos 30 dias de estado de conformidade em formato gráfico, conforme ilustrado na figura 4.20 [HD19], [Tho17].

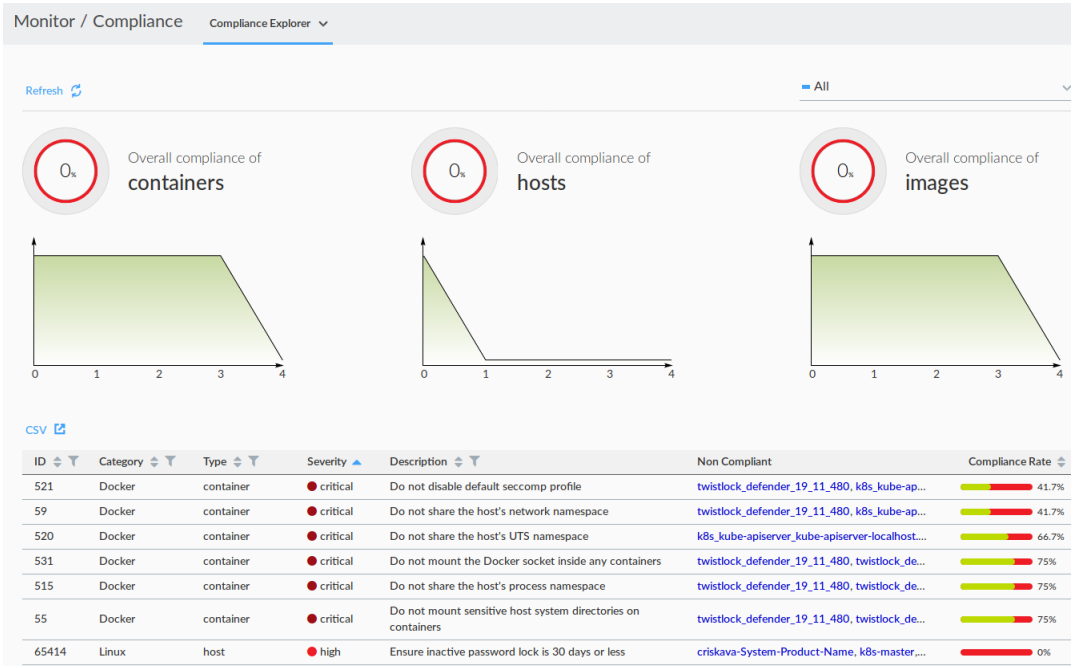

Figura 4.20: Visão única do estado de conformidade.

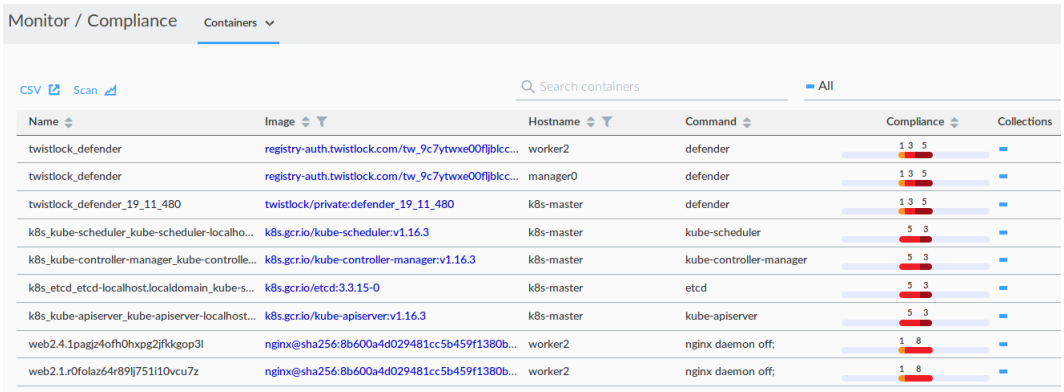

Figura 4.21: Estado de conformidade de contentores.

A figura 4.21 mostra o estado de conformidade de contentores instalados em todos os *hosts* e *cluster*, incluido o respetivo nível de conformidade.

A figura 4.22 ilustra os resultados do estado de conformidade das imagens junto dos seus repositórios, registos e os hosts de aplicação, com os valores de conformidade de cada imagem executada.

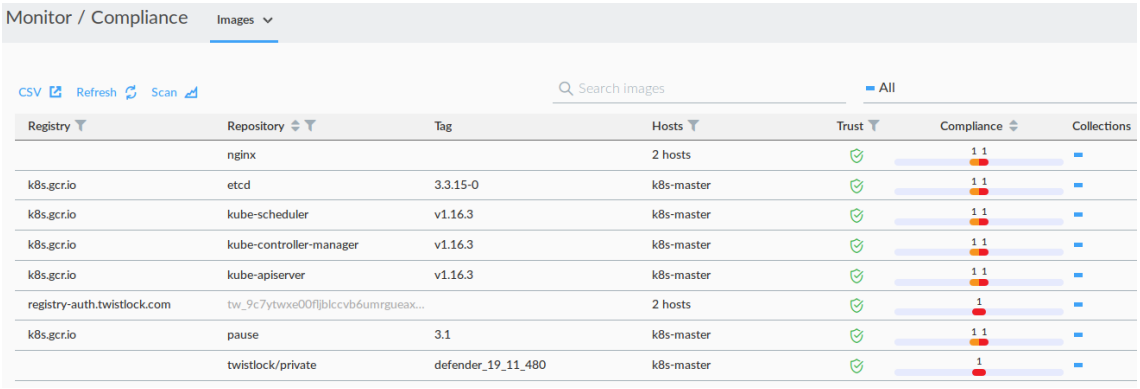

Figura 4.22: Resultado de Conformidade de Imagens.

O Twistlock faz uma análise de conformidade eficaz até ao nível de hosts e de *clusters*, conforme se ilustra na figura 4.23, para determinar a pontuação de conformidade necessária para tomar uma medida adequada para todas as conexões estabelecidas entre máquinas na rede.

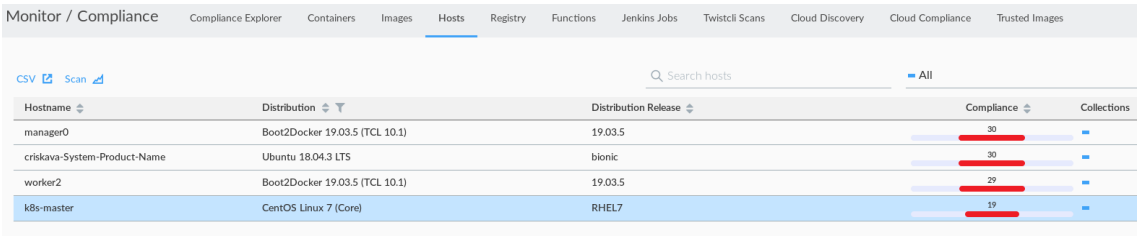

Figura 4.23: Análise de Conformidade de *Hosts* e *Clusters*.

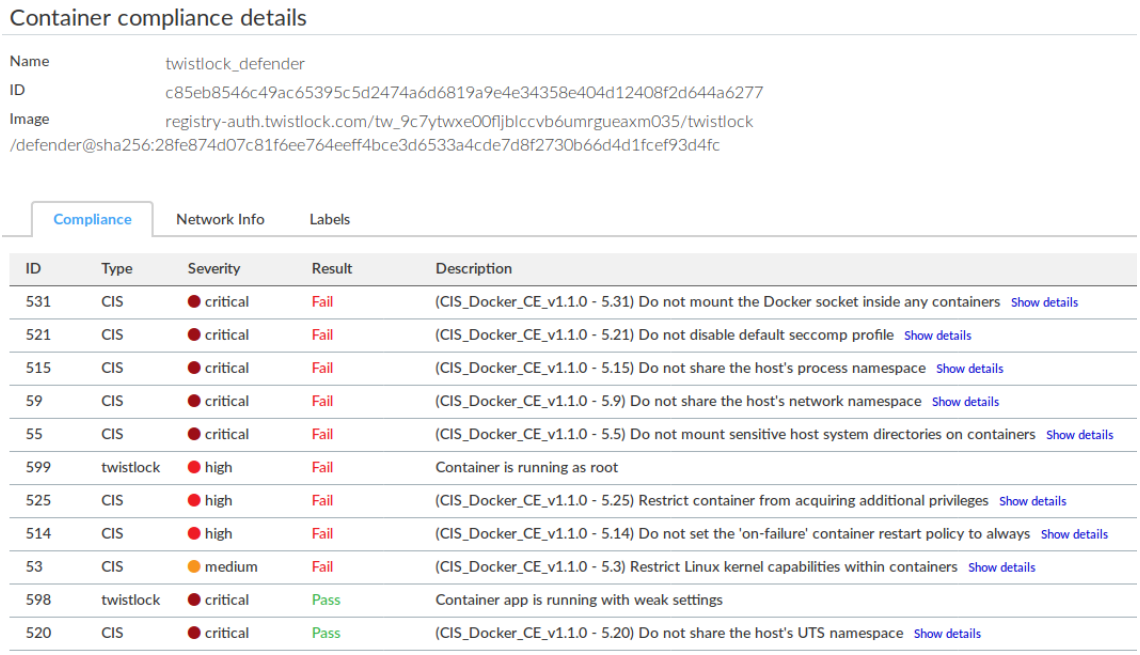

Figura 4.24: Descrição de conformidade dos contentores.

O Twistlock fornece detalhes de conformidade de contentor com *ID* e norma de segurança

para proteção do sistemas *CIS* (*Center for Internet Security*), conforme se ilustra na figura 4.24, para ajudar a garantir a segurança de dados, acesso não autorizado, de maneira a evitar a vulnerabilidade no sistema e limitar ataques cibernéticos sem esquecer o autorizado e não autorizado com descrição de cada tipo de *CIS*.

4.4 Impacto do Uso da Plataforma Twistlock no Desempenho de Contentores do Docker

4.4.1 Introdução

O desempenho de contentores é um fator fundamental em ambientes que utilizam a tecnologia de contentores do Docker e usam o Twistlock para proteger todas as suas carga de trabalho. Para analisar o desempenho de contentores do Docker, utilizamos o coletor de dados Sumo Logic para monitorizar o desempenho das actividades do sistema. O coletor foi instalado nas máquinas afim de extrair as informações do sistema para serem exportadas, avaliadas e supervisionadas a partir da plataforma Sumo Logic com o propósito de apresentar os resultados coletados em tempo real. Para coletar informações detalhadas sobre o desempenho de contentores e processos, incluindo a frequência de uso da CPU em contentores do Docker e a quantidade de memória utilizada em contentores do Docker, o tráfego da rede com Twistlock e sem Twistlock, é importante realizar certos ajustes no coletor Sumo logic. Para coletar dados dos processos executados, as máquinas devem estar em execução.

4.4.2 Experiências Realizadas e Cargas de Trabalho Utilizadas

Os testes realizados sobre as duas máquinas com sistemas operativos Linux Ubuntu e CentOS 7, envolveram uma carga de trabalho inicial com cinco contentores (Meu_*container*, *Cloud_discovery*, *Exciting_davinci*, *Keep_varaminhira*, *Nginx_app*) que foram configurados para testes, e uma máquina virtual sobre o sistema operativo Linux CentOS 7. Para poder medir o desempenho dos contentores foi necessário carregar e baixar no Docker Hub os mesmos contentores criados, o que permite depois avaliar o desempenho de cada um deles através da plataforma Sumo Logic para poder apresentar como os processos estão a ser executados, o uso da CPU na execução dos contentores e a eficiência para manter o CPU mais estável, o tráfego e os pacotes de rede dos contentores enviados e recebidos durante a fase de execução.

4.4.3 Desempenho de Contentores do Docker sem Twistlock e com Twistlock Utilizando a Plataforma Sumo Logic

Depois da instalação do coletor do Sumo Logic, esta plataforma controlou e forneceu uma

análise do comportamento de todos os recursos do sistema durante a realização dos testes, o que permitiu supervisionar o desempenho dos contentores nas variáveis de memória, rede, processador, incluindo número de *hosts* e número de contentores, determinando os consumos realmente utilizados pelos processos executados ao longo do tempo.

Durante a execução, os painéis são preenchidos automaticamente e é importante observar que cada painel é preenchido lentamente com dados correspondentes à consulta por intervalo de tempo. As figuras 4.25 e 4.26 mostram o número de *hosts* do Docker num único gráfico, fornecendo o número de contentores criados ou iniciados num único gráfico, o número de contentores parados ou pausados num gráfico com um só valor, exibindo o número de contentores detectados num gráfico de colunas numa linha do tempo, os principais contentores por consumo de CPU num gráfico de área numa linha de tempo, os principais contentores por consumo médio de memória, os dados sobre os principais contentores, pela quantidade de dados enviados e recebidos num gráfico de linhas, o número total de erros de rede num gráfico de colunas e o número de ações do contentor num gráfico de colunas numa linha do tempo. Os resultados apresentados foram produzidos a partir dos dados coletados nos paínéis da plataforma Sumo Logic que posteriormente irão permitir elaborar gráficos com base nos valores extraidos do ambiente Sumo Logic.

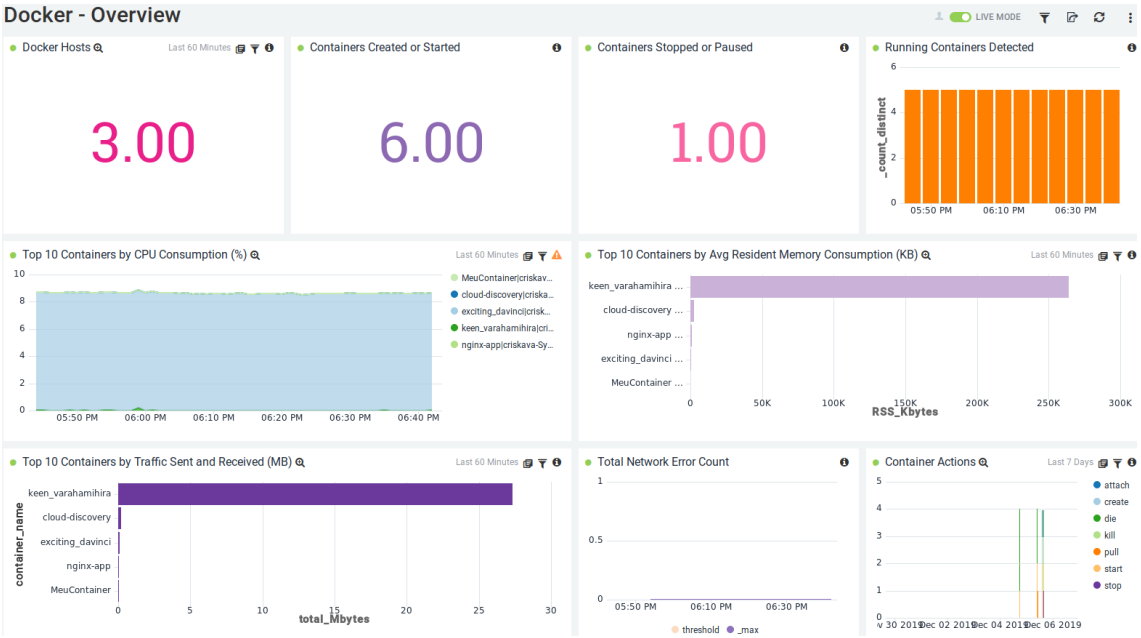

Figura 4.25: Visão Geral do Desempenho do Docker sem Twistlock.

O Sumo Logic apresenta uma visão geral do desempenho sem Twistlock e com Twistlock com número de máquinas Docker em funcionamento, contentor ligado e conectado ao sistema com desempenho geral de consumo de CPU, da memória e o tráfego da rede tal como ilustram as figuras 4.25 e 4.26.

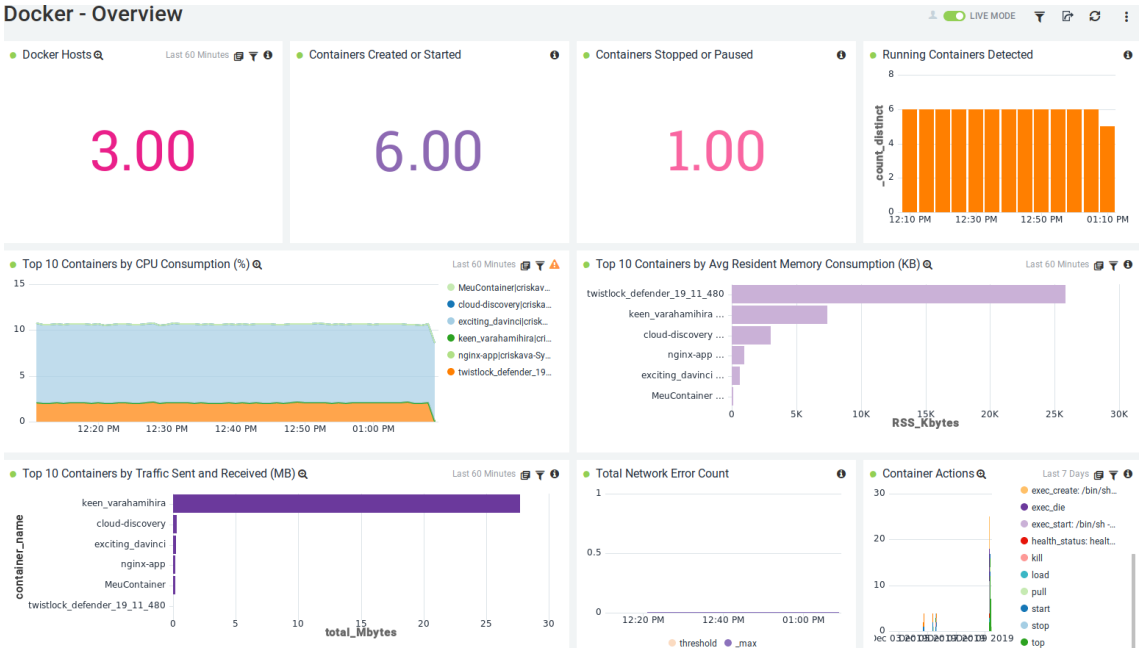

Figura 4.26: Visão Geral do Desempenho do Docker com Twistlock.

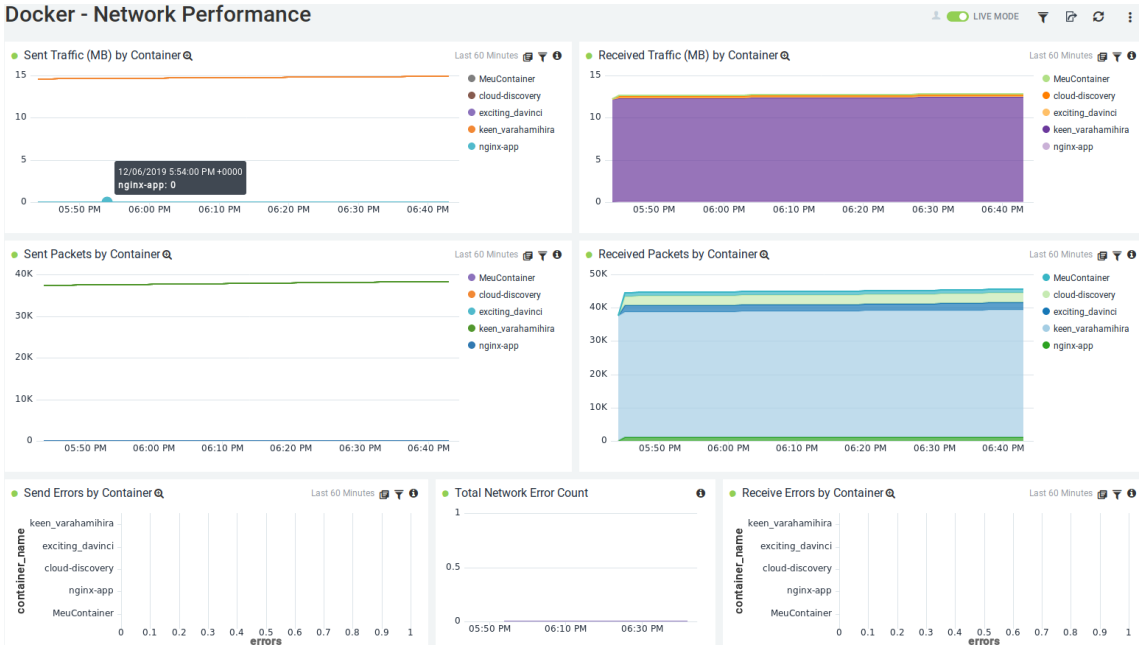

Figura 4.27: Desempenho de Rede do Docker sem Twistlock.

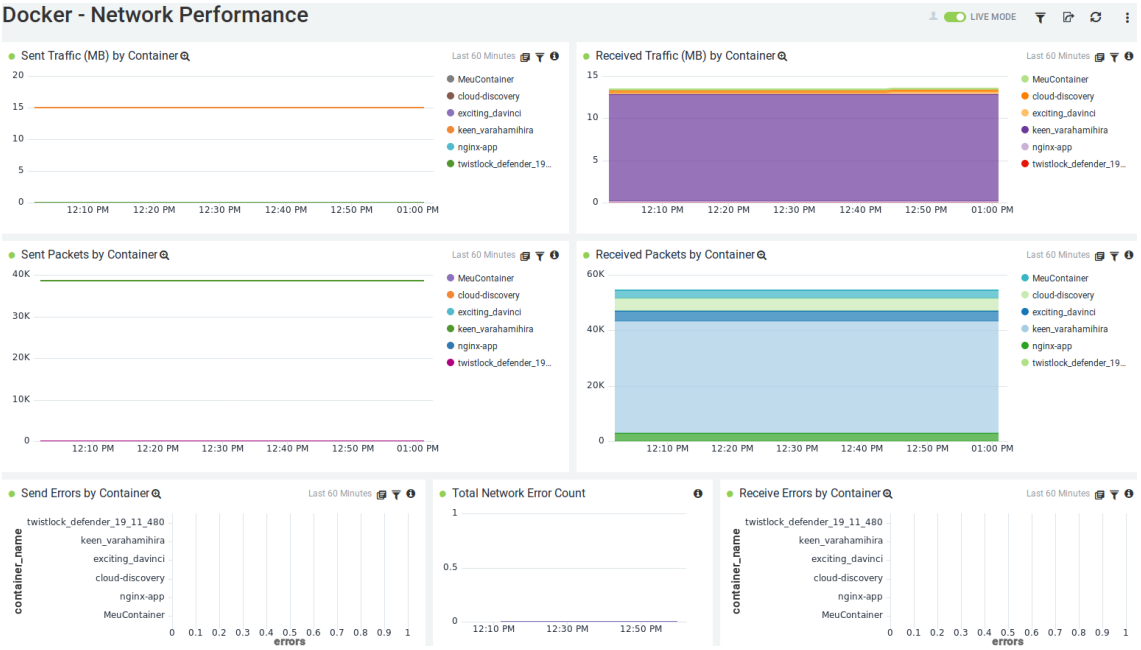

Figura 4.28: Desempenho de Rede do Docker com Twistlock.

As figuras 4.27 e 4.28 mostram o desempenho do tráfego da rede com Twistlock e sem Twistlock, com o tráfego de envio e receção de contentores tal como o envio de todo os pacotes da rede e receção dos pacotes por contentores.

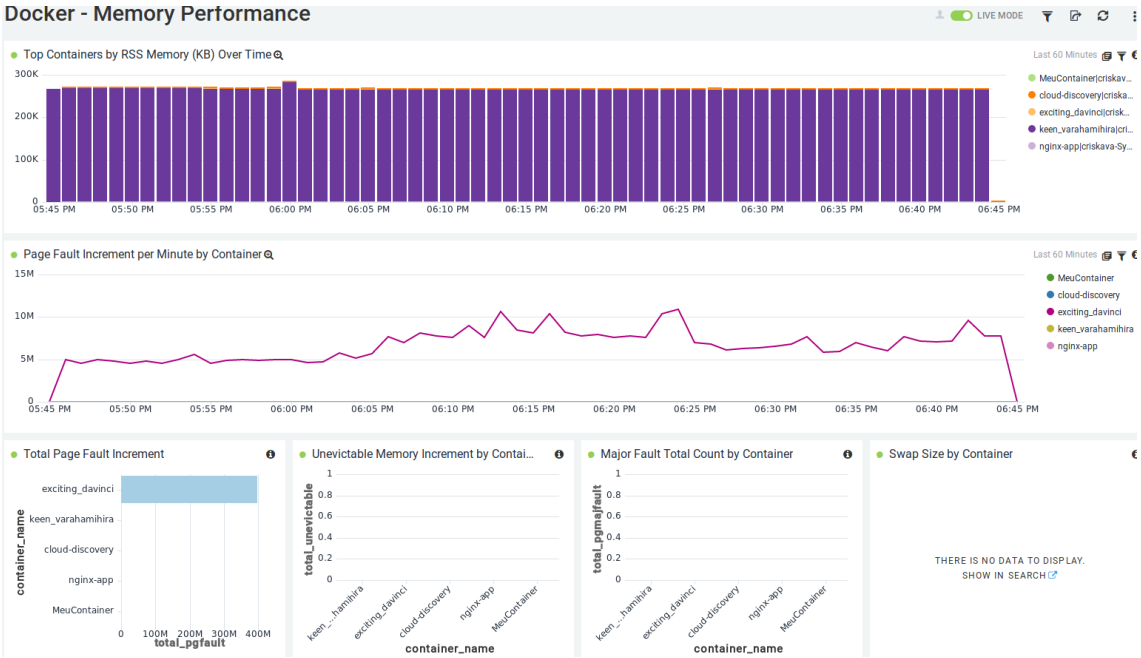

Figura 4.29: Desempenho da Memória sem Twistlock.

O desempenho da memória do Docker apresenta a quantidade de bytes utilizados pelo processo na memória RAM e incremento de página por minuto no contentor com Twistlock e sem Twistlock, como ilustram as figuras 4.29 e 4.30.

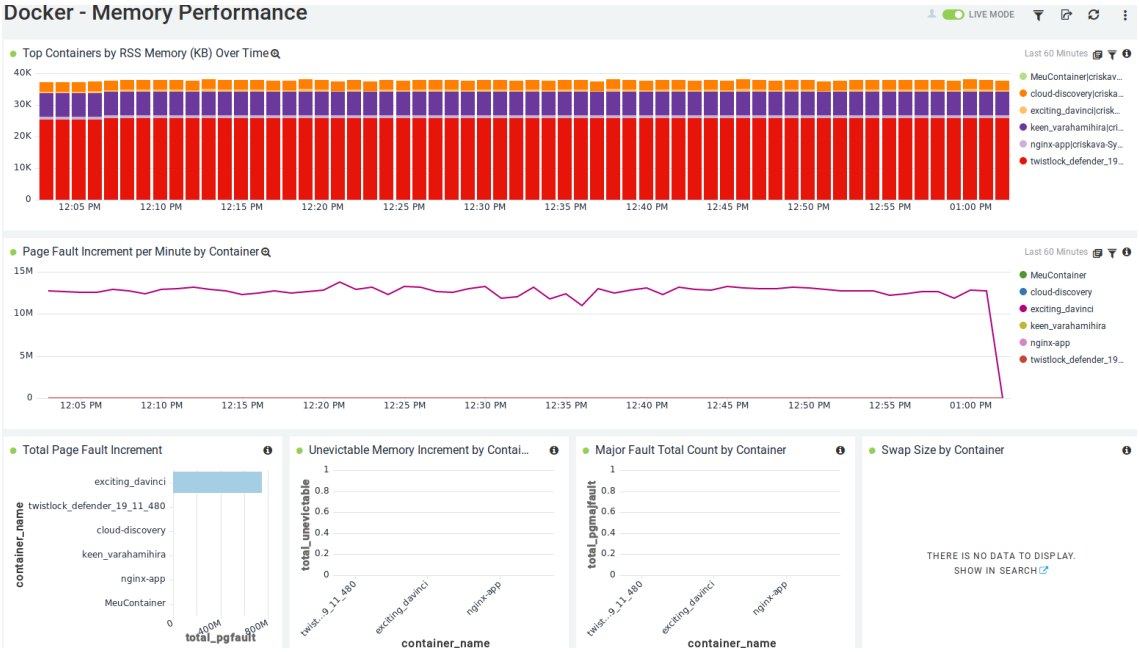

Figura 4.30: Desempenho da Memória com Twistlock.

O Sumo Logic Apresenta o desempenho da CPU em Docker sem Twistlock e com Twislock, com um total de consumo de CPU em memória por contentor, a utilização de CPU por contentor, a forma como os contentores usam o *Kernel* e os processos na CPU.

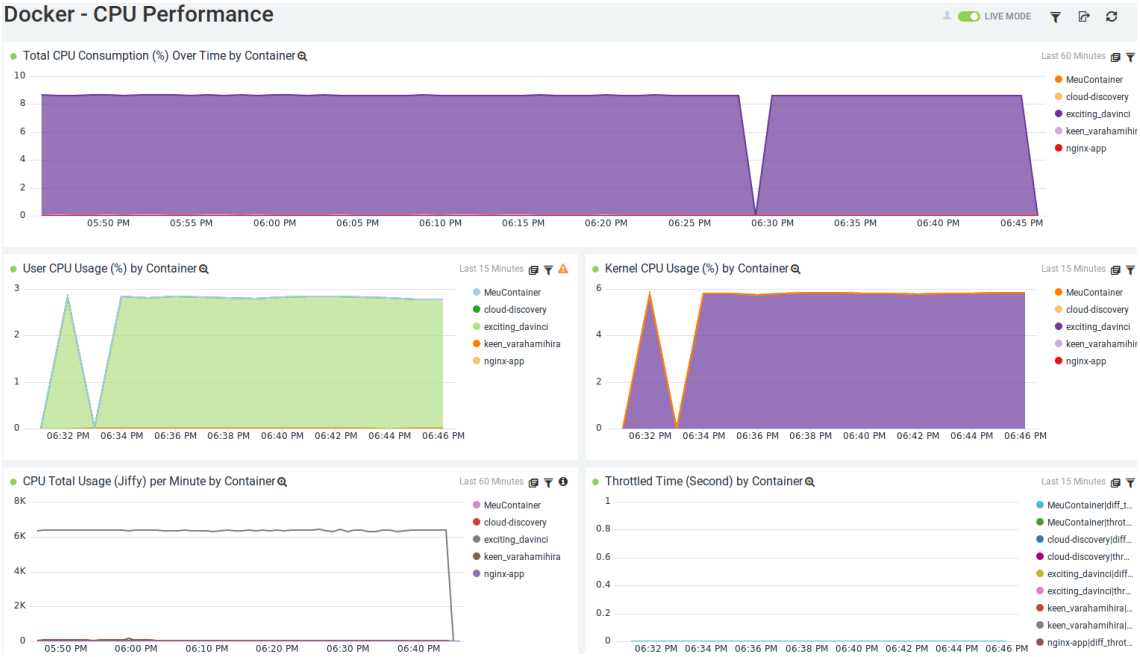

Figura 4.31: Desempenho da CPU em Docker sem Twistlock.

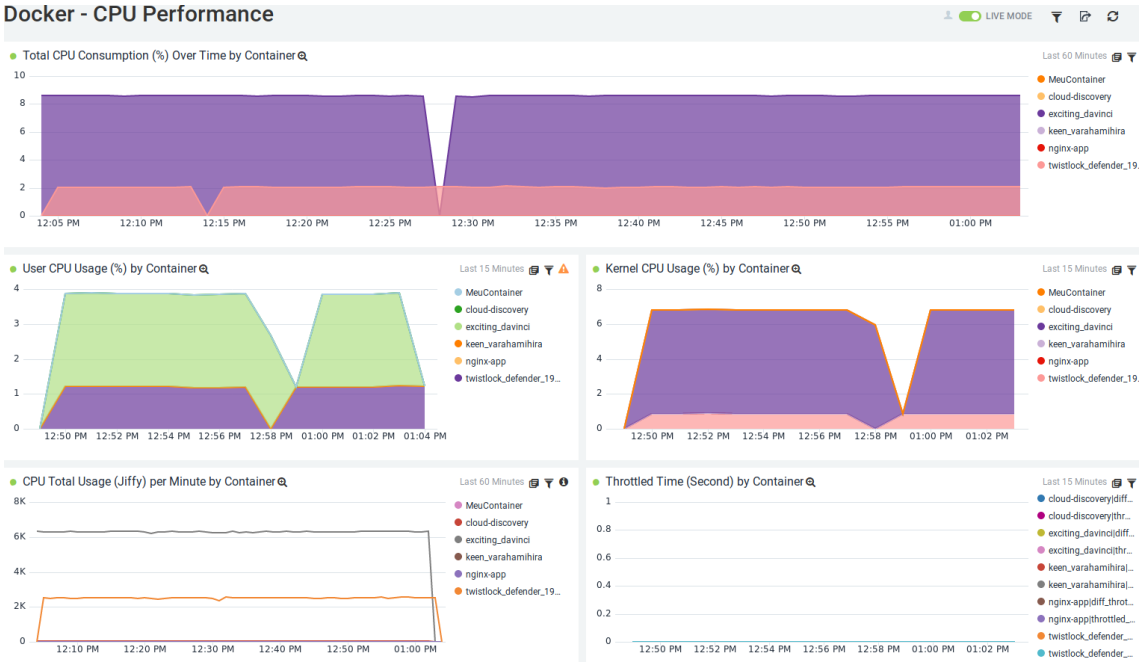

Figura 4.32: Desempenho da CPU em Docker com Twistlock.

4.4.4 Estudo Comparativo do Desempenho de Contentores do Docker sem Twistlock e com Twistlock

Esta subsecção apresenta os resultados obtidos sobre o desempenho dos contentores do Docker antes da instalação do Twistlock e depois da instalação do Twistlock, apontamos alguns indicadores mais relevantes que mostram o desempenho da carga de trabalho com Twistlock e sem Twistlock. As figuras, apresentadas ao longo desta secção, foram feitas com base nos resultados gerados pelo coletor de Sumo Logic, permitindo interpretar sobre a forma de gráficoos dados fornecidos através dos painéis do Sumo Lógica nas variáveis de rede, CPU e memória.

Desempenho de Rede em Docker:

O desempenho de rede em Docker indica que a taxa de tráfego enviado por contentor sem Twistlock é maior, aproximadamente 14.8 Mb/s, que a taxa de tráfego enviado por contentor com Twistlock, que atinge uma vez 15Mb/s, conforme se pode ver na figura 4.33.

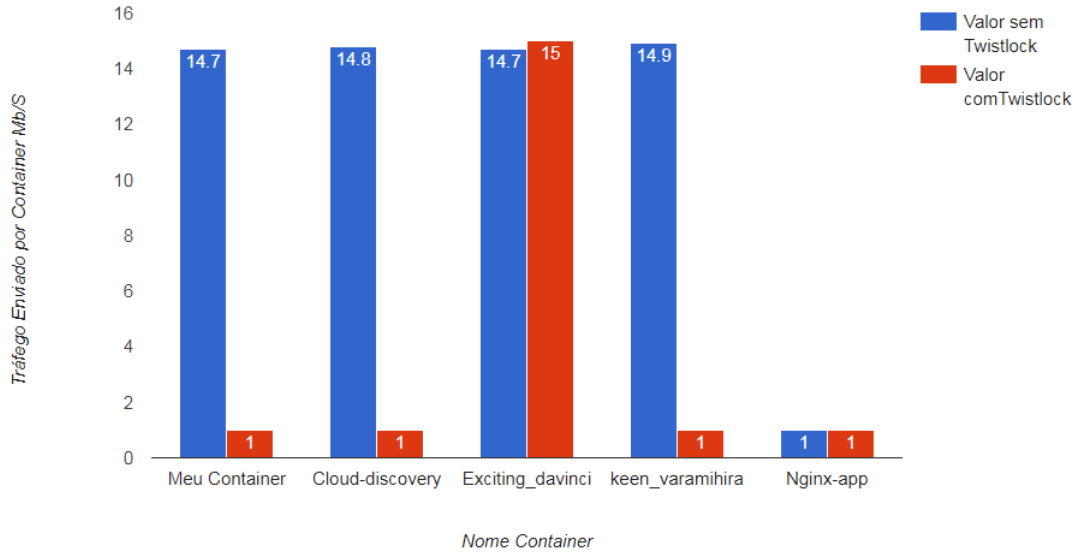

Figura 4.33: Tráfego enviado por contentor.

Observamos um equilíbrio entre o tráfego recebido por contentor sem Twistlock e com Twistlock, conforme se pode ver na figura 4.34, tendo uma taxa de transmissão na rede próxima de 12,8 Mb/s por contentor, mesmo com o Twistlock a aplicar o mecanismo de gestão do tráfego de rede e a controlar todas as ligações autorizadas e não autorizadas que possam ser suprimidas.

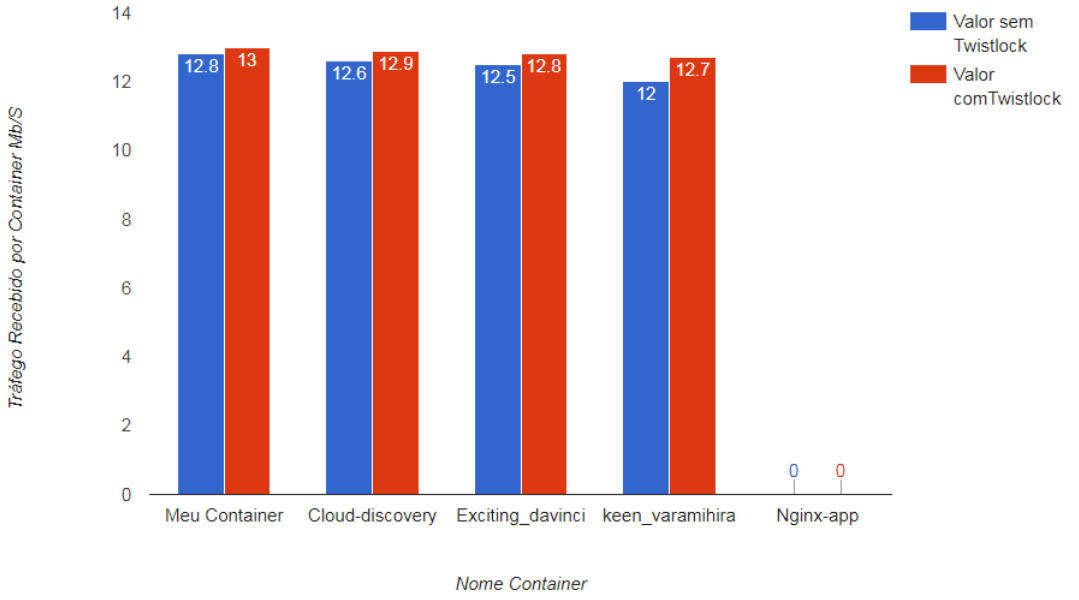

Figura 4.34: Tráfego recebido por contentor.

Apenas um contentor conseguiu enviar pacotes com Twistlock, com uma taxa de transmissão de 12,7 Mb/s, em comparação com os pacotes enviados por contentores sem Twistlock, conforme se pode ver na figura 4.35.

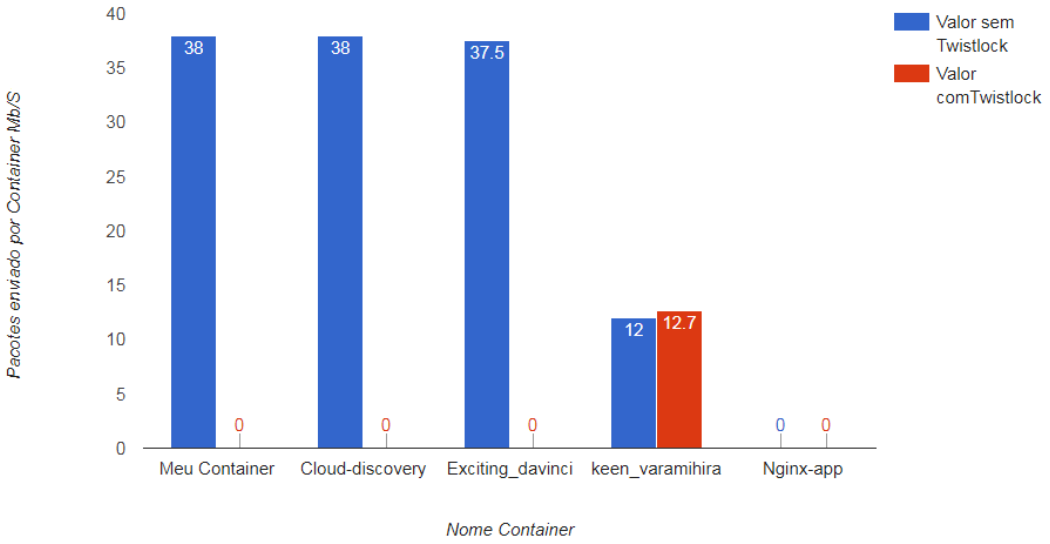

Figura 4.35: Pacotes enviados por contentor.

Apesar de uma filtragem de tráfego das camadas da rede pelo Twistlock *Defender* simplesmente evidênciar uma legeira variação na receção dos pacotes por contentor com Twistlock, constatamos um equílibrio na receção entre os pacotes com Twistlock e sem Twistlock, como se ilustra figura 4.36.

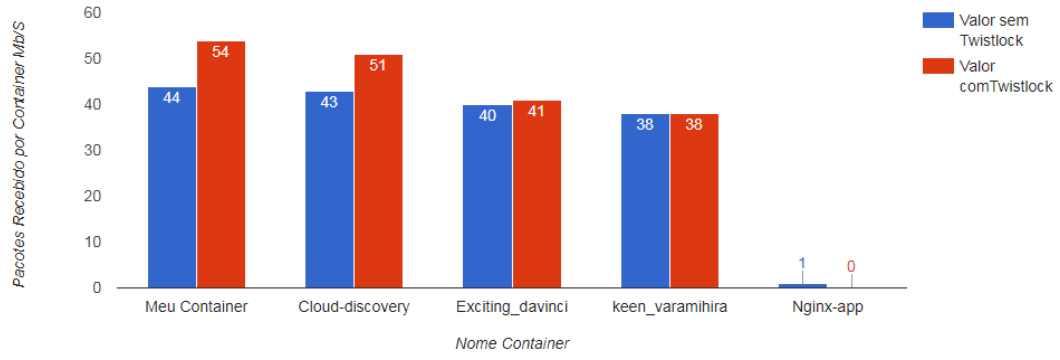

Figura 4.36: Pacotes recebidos por contentor.

Desempenho do CPU em Docker:

O consumo total de CPU ao longo do tempo em memória foi uma batalha nivelada na perspetiva da CPU, tanto com Twistlock com sem Twistlock, com uma média de 8,30% de consumo de memória no processador, conforme se ilustra na figura 4.37, embora o Twistlock garanta a proteção de todas as suas cargas de trabalho.

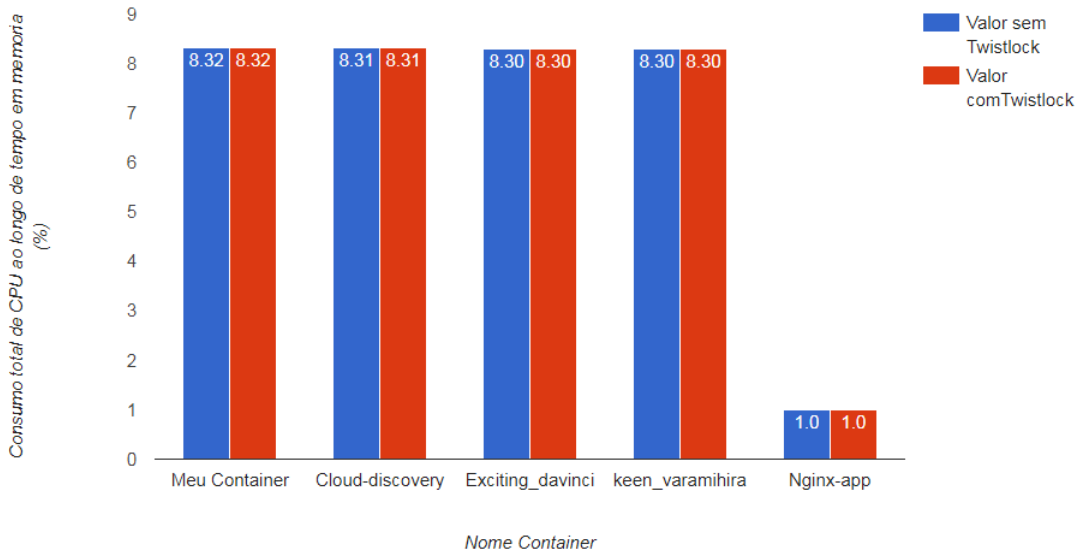

Figura 4.37: Consumo total de CPU ao longo do tempo em memória.

Observou-se um desiquílibrio entre o uso da CPU de utilizadores por contentor com e sem Twistlock, conforme se ilustra na figura 4.38. Os contentores com Twistlock precisaram de mais percentagem de uso de CPU para executar as suas tarefas, em relação aos contentores sem Twistlock para executarem as mesmas tarefas.

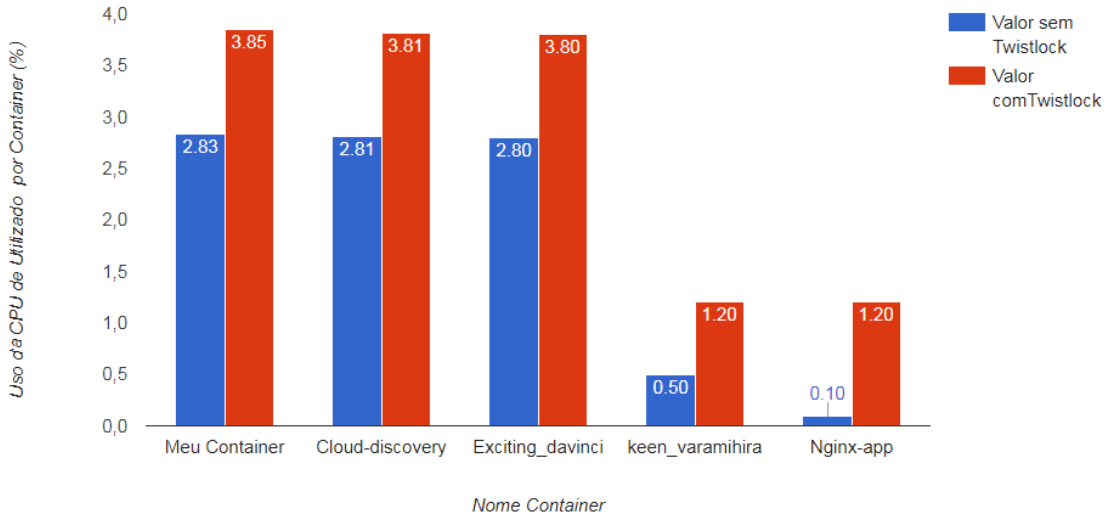

Figura 4.38: Uso do CPU de Utilizado por contentor.

O gráfico da figura 4.39 mostra o uso da CPU do *Kernel* por contentor. Conforme se pode ver, os contentores com Twistlock tiveram melhor desempenho em relação aos contentores sem Twistlock, devido ao facto de os dois ultimos contentores terem usarem muito pouco a CPU do *kernel* (1%).

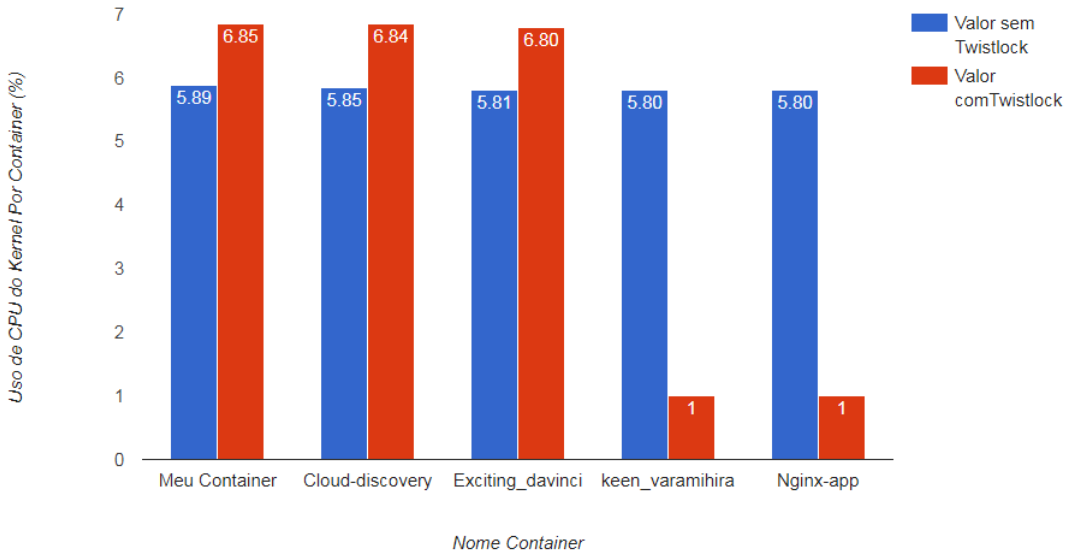

Figura 4.39: Uso de CPU do *Kernel* Por *contentor*.

Os valores equílibrados demonstram a batalha no uso de CPU em minutos por contentor com Twistlock e sem Twistlock para executar os os processos tal como ilustra o gráfico da figura 4.40.

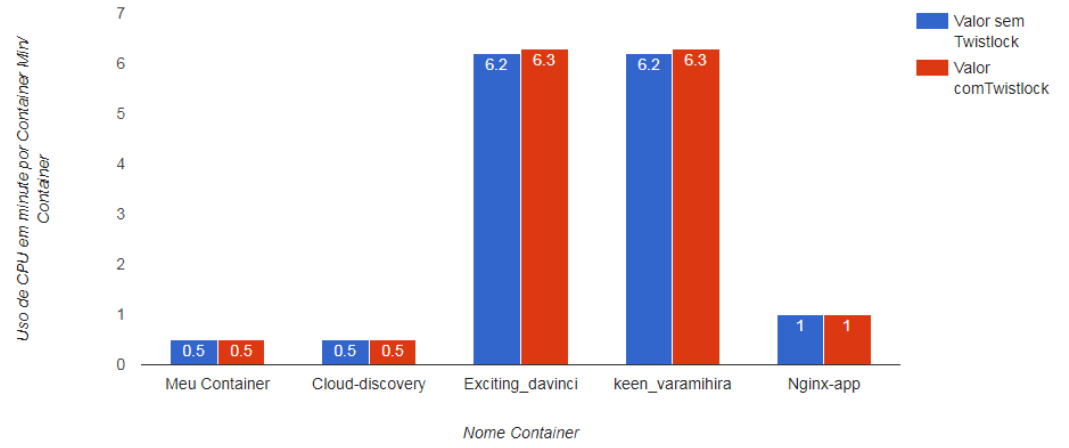

Figura 4.40: Uso de CPU em minutos por *contentore*.

Desempenho da Mémoria em Docker

A quantidade de bytes utilizados pelos processos na memória RAM para os contentores sem Twistlock, meu *container*, *cloud-discovery* e *keen-varamihira*, foi, em média de 270 Kb/s de memória RSS (*Resident Set Size*) ao longo do tempo, em comparação com os contentores com Twistlock que têm valores abaixo de 40 Kb/s, conforme se ilustra na figura 4.41.

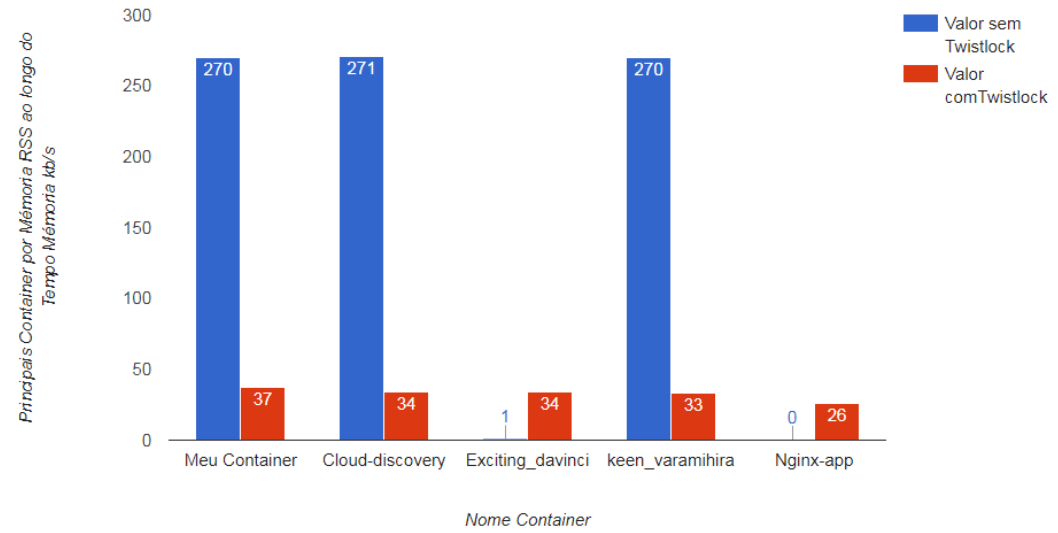

Figura 4.41: Os Principais contentores por memoria RSS ao longo do tempo.

Os contentores com Twistlock apresentam um incremento de falhas de páginas por minuto elevado, variando entre 1,0 e 12,5 página/minuto em relação aos contentores sem Twistlock que apresentam um incremento de falhas de páginas por minuto entre 1,0 à 7,0 página/minuto, conforme se pode ver na figura 4.42

.

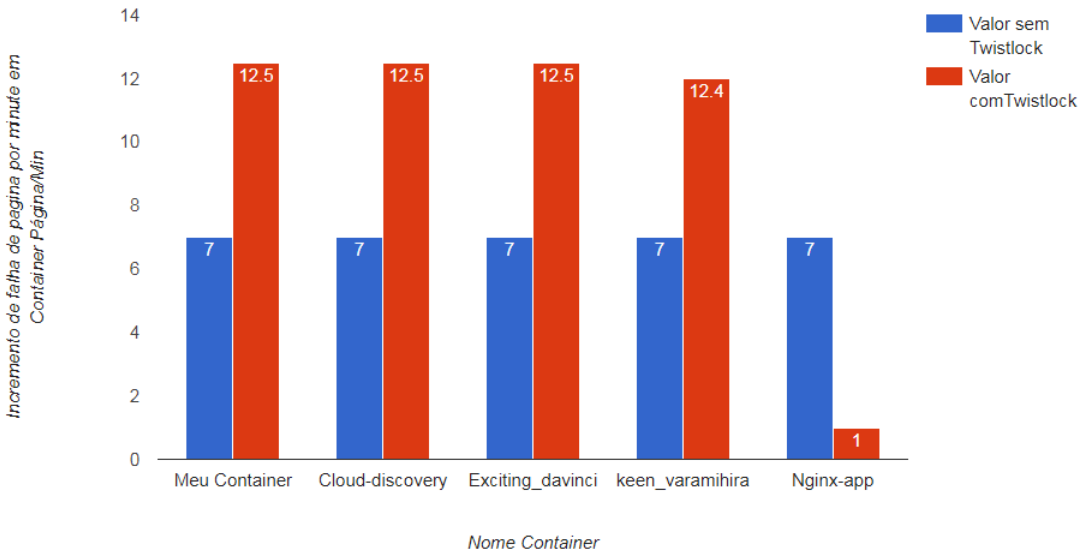

Figura 4.42: Incremento de falhas de página por minuto em contentores.

Em relação ao incremento total de falhas de páginas, observa-se na figura 4.43 a mesma tendência em relação ao gráfico da figura 4.42: os contentores com Twistlock têm um maior número de falhas de página por minuto na memória em relação aos contentores sem Twistlock. Constatamos que com Twistlock, o sistema operativo necessita de mais tempo para validar o endereço de memória.

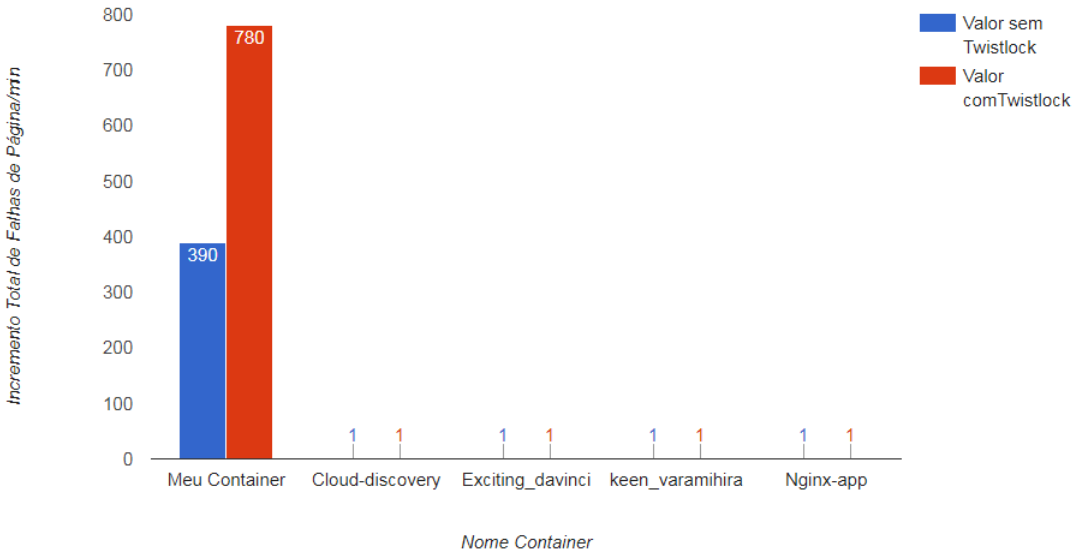

Figura 4.43: Incremento Total de Falhas de Página.

4.5 Conclusão

Este capítulo procurou descrever a implementação de um ambiente experimental envolvendo a instalação do Twistlock utilizando a plataforma Prisma Cloud para ilustrar o respetivo funcionamento e para apresentar uma avaliação e estudo comparativo de desempenho de contentores do Docker com Twistlock e sem Twistlock.

Os testes realizados a partir da plataforma Sumo Logic permitiram medir o desempenho de contentores do Docker com Twistlock e sem Twistlock. Os resultados indicam que o Twislock não altera significativamente o desempenho da carga de trabalho em ambiente em termos de desempenho nas variáveis de memória, redes e processador, apesar da garantia do Twistlock em termos de segurança.

Capítulo 5

Conclusões e Trabalho Futuro

5.1 Principais Conclusões

Actualmente a segurança em ambientes nativos da nuvem merece uma maior atenção atendendo às necessidades do mercado relativas ao uso de contentores em ambientes Docker e imagens construídas por terceiros podem causar várias vulnerabilidades que podem ser aproveitadas para atividades maliciosas. O Twistlock, sendo uma plataforma cibernética nativa da nuvem, fornece ferramentas adequadas com estratégia de proteção do ciclo de vida em todas as fases de desenvilvimento de *software* e a pilha completa, garantindo as suas políticas de segurança de forma a controlar todos os serviços da nuvem que estão a ser implementados e executados.

O Twistlock reforça o controlo do acesso, apresentando uma visão mais ampla e detalhada do sistema de modo a que todo o comportamento não autorizado seja detectado e exibida a respetiva gravidade em tempo de execução, sem intervenção humana, para proteger a carga de trabalho e as aplicações nativas da nuvem, incluindo todos os objetos, contentores, imagens e *hosts*), utilizando mecanismos de gestão de conformidade para garantir que a segurança do ambiente em execução permanece em conformidade com as políticas de configurações estabelecidas. Estas análises de defesa profunda de vulnerabilidades e ameaças garantem as medidas para identificar repositórios não autorizados e conexões não autorizadas, que podem passar despercebidas às equipas de seguranças, de modo a que estas sejam alertadas automaticamente sobre qualquer evento ou tentativa de ação não autorizada.

Observou-se durante a fase experimental, nomeadamente durante a implementação e realização de testes práticos, que o Twistlock permite que diferentes contentores, imagens, *hosts* e aplicações do Docker, na mesma rede, sejam analisados num único local para que consiga detetar onde está mais vulnerável sem que haja prejuízo no sistema. No que se refere ao desempenho de contentores do Docker com Twistlock e sem Twistlock, os recursos do sistema não foram significativamente comprometidos porque o Twistlock não altera significativamente o uso da CPU, memória, e tráfego da rede. Os recursos consumidos pelos contentores do Docker foram reduzidos, sendo este um dos fatores mais importantes na fase de desenvolvimento e da produção de contentores do Docker.

5.2 Sugestões para Trabalho Futuro

Como sugestão para trabalho futuro, fica a proposta de avaliar diretamente o desempenho de contentores do Docker sem recorrer ao Sumo Logic e utilizar uma ferramenta de orquestração de contentores como o Kubernetes.

Bibliografia

[AB17] A. Leung, A. Spyker and T. Bozarth. "Titus: Introducing containers to the netflix cloud". *ACM Queue*, vol. 15(no. 5):pp. 1–25, 2017. 1

[Abb17] Brendan Michael Abbott. A Security Evaluation Methodology for Container Images, 2017. 13

[And10] Zuquete André. *Segurança em Redes Informáticas*. Lisboa, 3ª ed. act edition, 2010. 10

[And18] Flavio Andrade. Os containers do Docker, 2018. Available from: `https://medium.com/@flaviochess/entendendo-os-containers-do-docker-a4a481007885`. 24

[Azu20a] Microsoft Azure. Container Instances, 2020. Available from: `https://azure.microsoft.com/pt-pt/services/container-instances/{%}0A`. 16

[Azu20b] Microsoft Azure. Container Registry, 2020. Available from: `https://azure.microsoft.com/pt-pt/services/container-registry/`. 16

[Bas19] Bora Basyildiz. A Brief History of Container Technology, 2019. Available from: `https://www.section.io/engineering-education/history-of-container-technology/`. 7, 15

[Byu17] IBM TJ Watson Research Center Byungchul Tak, Kyungpook National University; Canturk Isci, Sastry Duri, Nilton Bila, Shripad Nadgowda, and James Doran. Understanding Security Implications of Using Containers in the Cloud. *Open access to the Proceedings of the 2017 USENIX Annual Technical Conference is sponsored by USENIX*, 2017. Available from: `https://www.usenix.org/conference/atc17/technical-sessions/presentation/tak`. 13

[CCK16] Jeeva Chelladhurai, Pethuru Raj Chelliah, and Sathish Alampalayam Kumar. Securing docker containers from Denial of Service (DoS) attacks. *Proceedings - 2016 IEEE International Conference on Services Computing, SCC 2016*, pages 856–859, 2016. 11

[CH13] Yulia Cherdantseva and Jeremy Hilton. A Reference Model of Information Assurance & Security. Availability, Reliability and Security (ARES). *Proceeding of the 18th International Conference*, pages 1–11, 2013. 9

[Cis20] Cisco. What Is Cybersecurity?, 2020. Available from: `https://www.cisco.com/c/en/us/products/security/what-is-cybersecurity.htm`. 8

[Clo20a] Alibaba Cloud. Container Registry, 2020. Available from: https://www.alibabacloud.com/product/container-registry?spm=a3c0i.239195.3156523820.dnavproductelastic12.170912bdDXyOjM. 16

[Clo20b] IBM Cloud. IBM Cloud Container Registry, 2020. Available from: https://www.ibm.com/uk-en/cloud/container-registry. 16

[CMD16] Theo Combe, Antony Martin, and Roberto Di Pietro. To Docker or Not to Docker: A Security Perspective. *IEEE Cloud Computing*, 3(5):54–62, 2016. 19

[Cnf20] Cnfc. Cloud Native Computing Foundation (CNCF), 2020. Available from: https://www.cncf.io/. 15

[Con20] Linux Containers. Linux containers, 2020. Available from: https://linuxcontainers.org/. 15

[CP17] Emiliano Casalicchio and Vanessa Perciballi. Measuring docker performance: What a mess!!! In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion*, pages 11–16, 2017. 11

[CS17] Miguel Pupo Correia and Paulo Jorge Sousa. *Segurança no Software*. Lisboa, 2.ª edição edition, 2017. 9, 10

[Cve20] Cve. Common Vulnerabilities and Exposures (CVE) List and the associated references, 2020. Available from: https://cve.mitre.org/. 10

[DA19] Ana Duarte and Nuno Antunes. An Empirical Study of Docker Vulnerabilities and of Static Code Analysis Applicability. *Proceedings - 8th Latin-American Symposium on Dependable Computing, LADC 2018*, pages 27–36, 2019. 12

[Dig08] Segurança Digital. Segurança da informação, conceitos e mecanismos, 2008. Available from: https://www.oficinadanet.com.br/seguranca. 8, 9

[DJ11] Virtualização De and M Tim Jones. Uma introdução à virtualização de aplicativos. pages 1–8, 2011. 5

[Doc14] Docker. Docker 0.9: introducing execution drivers and libcontainer, 2014. Available from: https://www.docker.com/blog/docker-0-9-introducing-execution-drivers-and-libcontainer/. 15

[Doc20a] Docker. Docker, 2020. Available from: https://www.docker.com/. 15, 19, 23, 24

[Doc20b] Docker. Docker Engine overview, 2020. Available from: https://docs.docker.com/engine/. xv, 17, 18

[Doc20c] Docker. Networking overview, 2020. Available from: https://docs.docker.com/network/. 22

[Doc20d] Docker. tecnologia de container em Docker, 2020. Available from: https://www.docker.com/. 1

[Doc20e] Docker2. Private Repositories to Share Container Images, 2020. Available from: https://www.docker.com/products/docker-hub. 21

[dOPP+19] Kerolayne de S. V. de Oliveira, Guilherme Panizzon, Maurício A Pillon, Koslovski P. Guilherme, C. Charles Miers, and M. Nelson Gonzalez. Uma análise de segurança no uso de contêineres Docker em nuvens IaaS OpenStack. *Computer on the Beach*, pages 21–30, 2019. 12

[EKR19] EKRAN. Role-based Access Control vs Attribute-based Access Control: How to Choose, 2019. Available from: https://www.ekransystem.com/en/blog/rbac-vs-abac. xv, 30, 31

[Eli17] Elizabeth. Docker security, 2017. Available from: https://www.alibabacloud.com/forum/read-675. xv, 26, 27

[Fle20] Flexera. State of the Cloud Report, 2020. Available from: https://info.flexera.com/SLO-CM-REPORT-State-of-the-Cloud-2020. 1

[Fre18] Mário Freire. *Virtualization: Concepts, Implementation Levels, Structu-res / Tools and Mechanisms Tecnologias Cloud e Data Center. Slides de Apoio às aulas de Tecnologias Cloud e Data Centers.* PhD thesis, Universidade Beira Interior, 2018. 6

[Gir19] Juan Ignacio Giro. Spotlight on Twistlock: Twistlock's unique all-in-one approach to integrating security in DevOps makes it an option worth considering., 2019. Available from: https://caylent.com/spotlight-on-twistlock. 25, 26

[Goo20] Google. Google Container Engine, 2020. Available from: https://cloud.google.com/container-options. 1

[HD19] Secure Hosts and Serverless Deployments. Prisma Cloud Administrator's Guide (Compute). 2019. Available from: https://docs.paloaltonetworks.com/content/dam/techdocs/en{_}US/pdf/prisma/prisma-cloud/prisma-cloud-admin-compute/prisma-cloud-admin-compute.pdf. xvii, 40, 42, 43, 45, 46, 49

[HDF12] Kai Hwang, Jack Dongarra, and Geoffrey Fox. Virtual Machines and Virtualization of Clusters and Data Centers. *British Library*, pages 129–187, 2012. 6

[Hen17] Flávio Henrique. Universidade de Brasília para Aplicações do Supremo Tribunal Federal, 2017. 13

[Hos20] Prisma Cloud Compute Edition Administrator's Guide 19.11. *Palo Alto Networks*, 2020. Available from: www.paloaltonetworks.com/company/contact-support. xv, 37, 38

[Ion19] Ionos. Tutoriel Docker : installation et fonctionnement de la plateforme de conteneurs, 2019. Available from: https://www.ionos.fr/digitalguide/serveur/configuration/tutoriel-docker-installation-et-premiers-pas/. 17, 18, 21

[KGJ11] Hwang Kai, C. Fox Geoffrey, and Dongarra Jack. *distributed-and-cloud-computing-from-parallel-processing-to-the-internet-of-things*. San Francisco, CA, United States, 1st edition, 2011. 5, 6

[Kou17] Petros Koutoupis. Geek guide, Twistlock- Managing Container Security and Compliance in Docker. *LinuxJournal*, pages 2–4, 2017. 7, 8, 26

[Kub20] Kubernetes. Kubernetes, 2020. Available from: https://kubernetes.io/pt/. 1

[Lew17] Kevin Lewis. Vulnerability Explorer: Deep Dive, 2017. Available from: https://www.twistlock.com/2017/07/18/vulnerability-explorer-deep-dive/. 44

[Lmc20] Lmctfy. lmctfy - Let Me Contain That For You, 2020. Available from: https://github.com/google/lmctfy{%}0A. 15

[LSB18] Andrew Leung, Andrew Spyker, and Tim Bozarth. Titus: Introducing containers to the netflix cloud. *Communications of the ACM*, 61(2):38–45, 2018. 1

[LSY18] Liron Levin, Dima Stopel, and Eran Yanay. Networking-Based Profiling of Container and Security Enforcement. *Twistlock*, 2018. 12

[Mad18] R. Madhumathi. The Relevance of Container Monitoring Towards Container Intelligence. *2018 9th International Conference on Computing, Communication and Networking Technologies, ICCCNT 2018*, pages 1–5, 2018. 12

[Mel19] Mike Melanson. Twistlock 19.07 Builds on Automation, Visibility, Prevention, 2019. Available from: https://thenewstack.io/twistlock-19-07-builds-on-automation-visibility-prevention/. 40, 41

[Mic17] Microsoft. Proteger contêineres do Docker no Serviço de Contêiner do Azure, 2017. Available from: https://docs.microsoft.com/pt-br/azure/container-service/dcos-swarm/container-service-security?fbclid=IwAR1mOL8a6qoKJ9RD8zXnKGFEP3ZQ5NMEifirk--GEsSM1NC-u3G6R1a6n{_}E. 11

[Mic20] Microsoft. Serviço de Containers do Azure, 2020. Available from: https://azure.microsoft.com/pt-pt/product-categories/containers/. 1

[Mor17] Roberto Morabito. Virtualization on internet of things edge devices with container technologies: A performance evaluation. *IEEE Access*, 5:8835–8850, 2017. xv, 17, 19

[MPA+16] A. R. Manu, Jitendra Kumar Patel, Shakil Akhtar, V. K. Agrawal, and K. N.Bala Subramanya Murthy. A study, analysis and deep dive on cloud PAAS security in terms of Docker container security. *Proceedings of IEEE International Conference on Circuit, Power and Computing Technologies, ICCPCT 2016*, pages 1–13, 2016. 13

[Net20] Netflix. Netflix, 2020. Available from: https://www.netflix.com/pt/. 1

[Oli15] Kiran Oliver. Twistlock and the Future of Container Security, 2015. Available from: https://thenewstack.io/twistlock-future-container-security/. 25, 26

[Ope14] Opencontainers/runc. opencontainers/runc, 2014. Available from: https://github.com/opencontainers/runc/tree/master/libcontainer. 15

[Pla20] Google Cloud Platform. Container Registry, 2020. Available from: https://cloud.google.com/container-registry{%}0A. 16

[Red18] Redhat. Virtualização, 2018. Available from: https://www.redhat.com/pt-br/topics/virtualization/what-is-virtualization. 5

[Red19] Redhat. Segurança de containers, 2019. Available from: https://www.redhat.com/pt-br/topics/security/container-security. 8

[Rkt20] Rkt. A security-minded, standards-based container engine, 2020. Available from: https://coreos.com/rkt/. 15

[Roh19] Gupta Rohan. Containerization using Docker, 2019. Available from: https://www.geeksforgeeks.org/containerization-using-docker/. xv, 17, 19, 20, 21, 22, 24

[Rou19] Jim Routh. Runtime Defense, 2019. Available from: https://www.twistlock.com/platform/runtime-defense/. 27, 28, 46

[Rou20] Jim Routh. Twistlock, Monitor, Achieve, and Enforce Compliance for Containers and Hosts, 2020. Available from: https://www.twistlock.com/platform/container-compliance/. 28

[Ser20a] Amazon Web Services. Amazon Elastic Container Registry, 2020. Available from: https://aws.amazon.com/pt/ecr/?nc2=h{_}ql{_}prod{_}ct{_}ec2reg. 16

[Ser20b] Amazon Web Services. Amazon Elastic Container Service, 2020. Available from: https://aws.amazon.com/pt/ecs/?nc2=h{_}ql{_}prod{_}ct{_}ecs. 1

[Ser20c] Amazon Web Services. Amazon Elastic Kubernetes Service (Amazon EKS), 2020. Available from: https://aws.amazon.com/pt/eks/?nc2=h{_}ql{_}prod{_}ct{_}eks. 1

[Sil07] Rodrigo Ferreira Silva. Virtualização de Sistemas Operacionais. page 114, 2007. Available from: https://www.lncc.br/{~}borges/doc/VirtualizacaodeSistemasOperacionais.TCC.pdf. 6

[Sil18] Fernando Silva. Segurança em Ambientes Conteinerizados, 2018. Available from: https://medium.com/@FernandoDebrand/seguran{ç}a-em-ambientes-conteinerizados-3390db012878. 8

[SLD+19] Dima Stopel, Liron Levin, Shapira Daniel, Ben Num Nitshan, and Morello John. Techniques for Protecting Applications from Unsecure Network Exposure. *Twistlock*, 1, 2019. 14

[Sum19] SumoLogic. Sumo Logic knows cloud operations and analytics, 2019. Available from: https://www.sumologic.com/how-it-works/. 39

[Sum20] SumoLogic. Docker Swarm, 2020. Available from: https://www.sumologic.com/glossary/docker-swarm/. 25, 39

[Tec19] Techdoc. Prisma Cloud, 2019. Available from: https://docs.paloaltonetworks.com/prisma/prisma-cloud.html. 16

[Tho17] Josh Thorngren. Application Security The Road to Twistlock 2.0: Twistlock Compliance Explorer, 2017. Available from: https://www.twistlock.com/2017/04/12/compliance-management-system/. 48, 49

[TTSS14] Yuan Hsin Tung, Shian Shyong Tseng, Jen Feng Shih, and Hwai Ling Shan. W-VST: A testbed for evaluating web vulnerability scanner. *Proceedings - International Conference on Quality Software*, pages 228–233, 2014. 13

[Twi19a] Twistlock. Container Basics Whitepaper, 2019. Available from: https://www.twistlock.com/resources/container-basics-whitepaper-chapter-1/. xv, 20, 21, 22

[Twi19b] Twistlock. Docker Basics Whitepaper, 2019. Available from: https://www.twistlock.com/resources/docker-basics-whitepaper-chapter-2/. xv, 20

[Twi20a] Twistlock. Cloud Native Security Integrated with CI/CD Pipelines, 2020. Available from: https://www.twistlock.com/platform/continuous-integration-tools/. 29

[Twi20b] Twistlock. Complete Access Control for Cloud Workloads and Cloud Native Apps, 2020. Available from: https://www.twistlock.com/platform/access-control/. 30

[Twi20c] Twistlock. Securing Cloud Native Apps with Layer 4 and Layer 7 Firewalling, 2020. Available from: https://www.twistlock.com/platform/cloud-native-firewall/. 29, 30

[Twi20d] Twistlock. Security from Development to Production, 2020. Available from: https://www.twistlock.com/platform/vulnerability-management-tools/. 31

[Wan18] Shawn X. Wang. *Current Trends in Computer Science and Mechanical Automation Vol.1: Selected Papers from CSMA2016*, volume 1. De Gruyter Open Ltd, Warsaw/Berlin, shawn x. w edition, 2018. Available from: www.degruyter.com. 6

[Wik20a] Wikipedia. Docker (software), 2020. Available from: https://en.wikipedia.org/wiki/Docker{_}(software){%}0A. 15, 16

[Wik20b] Wikipedia. Hyper-V, 2020. Available from: https://en.wikipedia.org/wiki/Hyper-V. 16