PRINCIPAIS DESAFIOS DA TECNOLOGIA DE CONTÊINERES PARA A COMPUTAÇÃO FORENSE[1]

Gilnei Ferraz

Resumo: A tecnologia de contêineres trouxe diversas facilidades para o desenvolvimento de *software* e gerenciamento da infraestrutura de tecnologia da informação (TI). Facilitou a execução de cargas de trabalho, permitindo um gerenciamento facilitado e flexível se comparado a virtualização tradicional. Os contêineres têm natureza efêmera e, juntamente com os orquestradores, possuem diversos componentes arquiteturais complexos que interagem entre si e com o sistema operacional do hospedeiro provendo a infraestrutura necessária para execução de contêineres com alta disponibilidade e demais benefícios dessa tecnologia. Este artigo buscou, por intermédio de revisão bibliográfica, relatar os principais desafios trazidos pela tecnologia de contêineres para a computação forense. Pode-se concluir que o sucesso da computação forense nesse ecossistema depende muito de planejamento e configurações prévias do ambiente, objetivando armazenamento e gerenciamento de *logs* e eventos que possam registrar comportamentos criminosos ou indevidos para análise forense em tempo real ou *post-mortem*.

Palavras-chave: *Container Forensics. Computação Forense. Contêineres.*

1 INTRODUÇÃO

As modernas tecnologias de contêineres surgiram em grande parte junto com a adoção de práticas *DevOps*, que buscam aumentar a integração estreita entre as equipes de desenvolvimento e operação. A natureza portátil e declarativa dos contêineres é particularmente adequada a essas práticas, pois eles permitem que a organização tenha elevada consistência entre os ambientes de desenvolvimento, teste e produção (SOUPPAYA, 2017).

Para Spiekermann et al. (2019), a principal tecnologia de contêineres utilizada é o Docker, mas existem outras como: containerd, CoreOS rkt, Mesos Containerizer, LXC Linux Containers e OpenVZ. Segundo Winkel (2017), o Docker causou um imenso

[1] Artigo apresentado como Trabalho de Conclusão do Curso de Especialização em Gestão de Segurança da Informação, da Universidade do Sul de Santa Catarina, como requisito parcial para a obtenção do título de Especialista em Gestão de Segurança da Informação.

impacto na forma que o *software* é desenvolvido e implantado nos ambientes atuais de TI.

Os contêineres têm natureza imutável e efêmera e isso tem implicações diretas na persistência dos dados. Em vez de misturar a aplicação com os dados que ele usa, os contêineres enfatizam o conceito de isolamento. A persistência de dados pode ser alcançada com a configuração prévia de volumes externos persistentes ou bancos de dados (SOUPPAYA, 2017).

Conforme VMware (2018), as *Virtual Machines* (VMs) e contêineres podem coexistir no ambiente tecnológico das organizações. A combinação de contêineres e VMs aproveita os benefícios de cada tecnologia, criando um resultado maior que a soma de suas partes, por isso que é uma das razões pelas quais os principais provedores de nuvem, como Google e Amazon, usam VMs para executar contêineres.

Segundo Souppaya et al. (2017), embora os contêineres sejam vistos como a próxima fase da virtualização, superando a virtualização de hardware, na realidade a virtualização de contêineres e hardware frequentemente coexistem e aprimoram os recursos individuais. As VMs oferecem muitos benefícios, como forte isolamento, automação de SO e um amplo e profundo ecossistema de soluções. As organizações não precisam escolher entre contêineres e VMs, podem continuar usando VMs para implantar, particionar e gerenciar seu hardware, enquanto usam contêineres para empacotar seus aplicativos e utilizar cada VM com mais eficiência.

Para Red Hat (2019), o Docker é excelente para gerenciar contêineres únicos. Mas com uma quantidade crescente de contêineres e aplicações em contêineres segregados em centenas de partes, o gerenciamento e a orquestração se tornou um grande desafio. Para isso surgiu a orquestração de contêineres. Existem alguns orquestradores disponíveis no mercado, como o Docker Swarm e Kubernetes, esse último inicialmente desenvolvido pela Google é amplamente utilizado.

O número de ataques cibernéticos cresce anualmente, segundo o *Internet Security Threat Report* publicado em 2019 pela empresa Symantec, ocorreu um aumento de 56 % no número de *web* ataques em relação ao ano de 2018 (SYMANTEC, 2019).

Segundo o Jesus et al. (2016), um dos maiores desafios na computação forense é a identificação exata dos locais abrangidos em determinado cenário criminoso, sendo a informação digital a principal evidência. Torna-se imprescindível a correta identificação,

isolamento, coleta e preservação dos vestígios para a apuração da autoria e materialidade do crime.

A *Digital Forensics and Incident Response* (DFIR) é uma parte basilar do processo de segurança de contêineres para qualquer organização. A forense de contêineres e a análise *post-mortem* aprimoram a segurança, auxiliando no entendimento e contenção dos impactos de violações de segurança que podem afetar negativamente a receita e a reputação da organização (SYSDIG, 2019).

De acordo com Riardi et al. (2019), embora muitos sistemas estejam migrando para a nuvem, poucas pesquisas foram realizadas sobre as ferramentas, processos e metodologias necessárias para obter evidências forenses legalmente defensáveis nesse ambiente.

A metodologia de pesquisa utilizada por esse trabalho é aplicada, com revisão teórica para identificar os principais desafios trazidos pela tecnologia de contêineres para a computação forense. Os principais meios de estudo foram artigos científicos, teses de doutorado, sites de fornecedores, *sites* especializados e livros técnicos da área de TI.

2 Tecnologia de Contêineres

Segundo Albuquerque Filho (2016), os contêineres proveem isolamento da aplicação, dependências e recursos de maneira similar a uma máquina virtual, porém de forma mais leve por compartilhar o *kernel* com o SO hospedeiro.

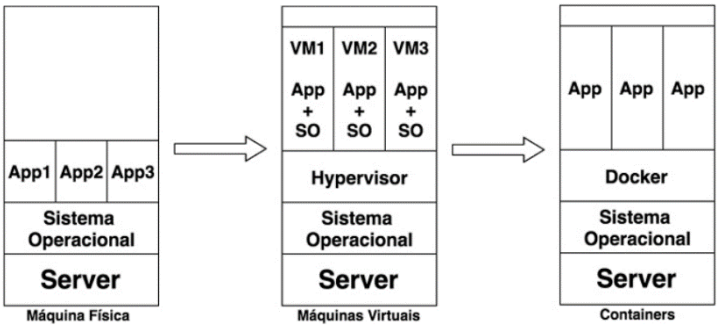
Segundo Vitalino e Castro (2018), contêiner é o agrupamento de uma aplicação junto com suas dependências, que compartilham o *kernel* do SO do servidor onde é executado, são interoperáveis, podendo o mesmo contêiner ser executado em diferentes sistemas operacionais. Os contêineres são compostos por imagens portáteis, normalmente pequenas, contendo apenas o essencial para o funcionamento da aplicação. Contêineres possuem similaridade com VMs, entretanto são mais leves e mais integrados ao SO da máquina hospedeira. Essa maior integração proporciona um melhor desempenho devido ao gerenciamento único de recursos.

Os contêineres e VMs têm benefícios semelhantes de isolamento e alocação de recursos, mas funcionam de maneira diferente porque os contêineres virtualizam o SO em vez do *hardware*. Vários contêineres podem ser executados na mesma máquina e compartilhar o *kernel* do SO com outros contêineres, cada um executando como processos isolados no espaço do usuário. Os contêineres ocupam menos espaço que as VMs, podem

lidar com mais aplicativos e requerem menos recursos. As VMs são uma abstração de *hardware* físico que transforma um servidor físico em muitos servidores virtuais. O *hypervisor* permite que várias VMs sejam executadas em uma única máquina física. Cada VM inclui uma cópia completa de um SO, aplicativo, binários e bibliotecas necessários, ocupando bastante espaço. Os contêineres e VMs usados juntos oferecem uma grande flexibilidade na implantação e gerenciamento de aplicativos (DOCKER, 2019).

Para Vitalino e Castro (2018), a execução de contêineres apresenta um pequeno *overhead* quando comparada à mesma aplicação sendo executada de forma nativa diretamente sobre o SO, isso se deve principalmente ao fato do compartilhamento dos recursos com a máquina hospedeira. Na execução de VMs, um novo SO completo e o seu *hardware* é emulado, consumindo assim muito mais recursos computacionais do servidor hospedeiro. Isso não ocorre com contêineres, pois os recursos serão compartilhados, possibilitando assim a execução de uma quantidade maior de contêineres em um único hospedeiro, se comparada a quantidade que seria possível com VMs. Na Figura 1.0 são apresentadas as diferentes camadas presentes na execução de aplicações nativamente, em *virtual machines* e em contêineres.

Figura 1.0 - Diferenças da execução de aplicações nativamente, em *virtual machines* e em contêineres.



Fonte: Descomplicando o Docker, 2018

A utilização inicial de conceitos de contêineres não é tão recente e começou com o comando *chroot* nos sistemas *Unix*, comando esse que era uma forma de isolar sistemas de arquivos. Na sequência surgiram as *jails* do FreeBSD que permitiam, além do isolamento de sistema de arquivo, o isolamento de processos. Em seguida a empresa Sun desenvolveu mais uma solução baseadas em contêineres apenas para sistemas Solaris, o Solaris Zones. Mas foi a criação do Virtuozzo pela empresa Parallels que permitiu uma ampla difusão dessa tecnologia. O Virtuozzo trouxe facilidades de gerenciamento de

contêineres e o seu código fonte *core* foi disponibilizado como *open source* com o nome de OpenVZ, ganhando assim grande destaque no gerenciamento de contêineres e contribuiu para a popularização de *Virtual Private Server* (VPS), mas o OpenVZ requeria a aplicação de um *patch* no *kernel* Linux (VITALINO e CASTRO, 2018).

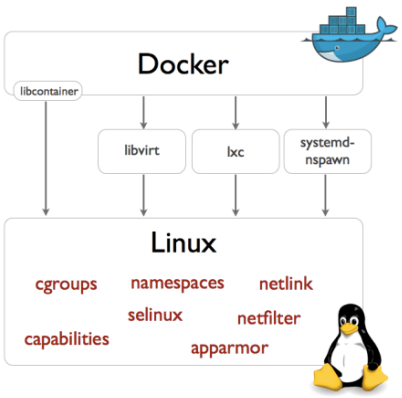
Conforme Vitalino e Castro (2018), após o surgimento do OpenVZ, a empresa Google começou o desenvolvimento do *CGroups* para o *kernel* Linux, adotando também contêineres em seus *datacenters*. No ano de 2008, juntamente com empresas como Virtuozzo e IBM, começaram o desenvolvimento do projeto Linux Containers (LXC), que incluía o *CGroups*, *namespaces* e *chroot* para compor uma solução completa e estável para a criação e gerenciamento de contêineres. Mas foi em 2013 que a tecnologia de contêineres ganhou grande popularidade com a disponibilização *open source* do *core* do Docker. O Docker foi criado pela empresa dotCloud que oferecia um *Plataform-as-a-Service* (PaaS) independente de linguagem de programação específica. As versões iniciais do Docker eram um *wrapper* do Linux Containers integrado ao sistema de arquivos *Union Filesystem*, mas ganhou muita popularidade e pouco tempo depois ganhou versões mais estáveis, sendo utilizado por empresas como Spotify, Amazon Web Services, Google e Red Hat. Mas desde a versão 0.9, o Docker substituiu o LXC pelo *libcontainer*, seu próprio formato de virtualização como o ambiente de contêiner padrão (BUI, 2015).

Para Bachiega et al. (2018), Docker é a ferramenta mais popular no mundo dos contêineres. Isso se deve principalmente ao fato dele ter iniciado o movimento que popularizou o uso facilitado dos contêineres, revolucionando os mecanismos para administração de contêineres de maneira rápida, automatizada e simplificada.

De acordo com Vitalino e Castro (2018), o Docker utiliza o conceito de camadas, que são montadas utilizando técnicas *Copy-On-Write* (COW). Um contêiner é composto por uma pilha de camadas, compostas por diversas camadas somente leitura e uma camadas superior que permite leitura e escrita. Apesar dessa última camada permitir a escrita, devido a natureza efêmera dos contêineres, normalmente os dados que não podem ser apagados são persistidos em volumes externos apresentados ao contêiner. Mas existem situações que requerem a escrita no sistema de arquivos do contêiner, para isso o Docker utiliza o mecanismo *Storage Drivers*. Os principais *storage drivers* utilizados pelo Docker são: *AUFS (Another Union File System)*, *Device Mapper*, *OverlayFS*, *OverlayFS2* e BTRFS.

Para Vitalino e Castro (2018), para o seu funcionamento o Docker utiliza funcionalidades básicas do *kerner* Linux. Na figura 2.0 é apresentado o compartilhamento de recursos entre o Docker e o *kernel* do SO.

Figura 2.0 - Compartilhamento de recurso entre o Docker e *Kernel* Linux.



Fonte: Docker 0.9: introducing execution drivers and libcontainer, 2014

Segundo Vitalino e Castro (2018), os *namespaces* são responsáveis pelo isolamento dos contêineres. Permitem que cada contêiner tenha seu ambiente, não interferindo em outros contêineres. O Docker utiliza os PID *Namespace* para processos, *Net Namespace* para interfaces de rede, *Mnt Namespace* para pontos de montagem, IPC *Namespace* para S*ystemV* IPC isolado, UTS *Namespace* para isolamento de *hostname*, domínio, dentre outros e o *User Namespace* para manter o mapa de identificação de usuários de cada contêiner. O *CGroups* prove a limitação de utilização de recursos do hospedeiro pelos contêineres. O *Netfilter* permite a comunicação dos contêineres por meio de regras de roteamento via *iptables*.

Para Spiekermann et al. (2019), o Docker fornece várias formas de rede, cada uma dependendo do número de contêineres em execução e de sua interação. Um único contêiner em execução é conectado através da interface *docker0*, criada pelo Docker no hospedeiro. A *docker0* é uma ponte *Ethernet* virtual, conectada de um lado ao hospedeiro, do outro lado, com uma placa de interface de rede virtual (vNIC) com um nome exclusivo conectado ao contêiner. Cada contêiner possui uma vNIC atribuída, conectada a um *namespace* isolado, portanto, possui seu próprio endereço IP, tabelas de roteamento, tabelas ARP e dispositivos de rede. Os contêineres conectados aos mesmos *namespaces* se comportam de maneira semelhante às redes baseadas em *hardware*, podendo se comunicarem dentro desse ambiente virtual. O Docker permite que a criação de redes

próprias dentro, aumentando ainda mais a flexibilidade dos ambientes de contêiner. Por padrão, os contêineres do Docker não podem ser acessados externamente, mas é possível criar um encaminhamento de portas do hospedeiro para portas dos contêineres, facilitando o acesso externo ao contêiner. Para isso o Docker usa o *netfilter/iptables* para encaminhar os pacotes de rede às portas atribuídas.

Segundo Turnbull (2016), o ecossistema Docker tem os seguintes componentes: Docker *Engine*: é um *daemon* que executa os contêineres, interagindo com as imagens, redes e armazenamento; *Images*: são os blocos de construção portáveis do Docker, são criadas em camadas e são versionadas; *Registry*: são os repositórios de imagens, podendo serem públicos ou privados; Contêineres Docker: são as imagens em execução, com todos os requisitos e dependências descritos da imagem.

Para Souza (2018), a disseminação da tecnologia de contêineres, juntamente com novos paradigmas de desenvolvimento elevou exponencialmente a quantidade de contêineres em execução. O gerenciamento dessa arquitetura é complexo, pois diversos fatores devem ser considerados, os principais são: os registros, a rede, o armazenamento, a segurança e a telemetria. Segundo BURNS et al (2016), para o correto tratamento dessa complexidade foi necessário a criação de ferramentas que pudessem orquestrar os contêineres e tratar dessa complexidade, ferramentas como o Kubernetes.

Segundo Kubernetes (2019), o Kubernetes é uma plataforma portátil, extensível e de código aberto para gerenciar cargas de trabalho e serviços em contêiner, facilitando a configuração declarativa e a automação. O nome Kubernetes tem origem grega e significa timoneiro ou piloto. Foi originalmente desenvolvido pela empresa Google e teve seu código fonte aberto em 2014. No ano de 2015, na versão 1.0, o Google forneceu o controle do Kubernetes para a Cloud Native Computing Foundation (CNCF), administrada pela The Linux Foundation.

Segundo Red Hat (2019), o Kubernetes permite: Orquestração de contêineres em vários servidores; Melhor aproveitamento de *hardware* para maximizar os recursos necessários na execução das aplicações corporativas; Controle e automatização das implantações e atualizações de aplicações; Montar e adicionar armazenamento para executar aplicações com monitoração de estado; Escalar rapidamente as aplicações em contêineres e recursos relacionados; Gerenciamento de serviços de forma declarativa, garantindo que as aplicações sejam executadas sempre da mesma maneira como foram

implantadas; Verificação de integridade e autorrecuperação das aplicações com posicionamento, reinício, replicação e escalonamento automáticos.

Conforme Souza (2018), o Kubernetes é implantado em uma arquitetura de *cluster*, sendo composto de diversos componentes, subdivididos em componentes *Master* e *Node*. Exigindo-se no mínimo três servidores para a configurar um *cluster* Kubernetes de alta disponibilidade.

Segundo Kubernetes (2019), o conjunto dos componentes *master* fornece o plano de controle do *cluster* e faz toda a gestão das decisões globais. São componentes *master*:

- *kube-apiserver*: expõe a *Application Programming Interfaces* (API) *Representational State Transfer* (REST) do Kubernetes, sendo o *front end* do plano de controle do Kubernetes. Foi desenvolvido para ser dimensionado horizontalmente, ou seja, é dimensionado implementando mais instâncias;
- *etcd*: provê armazenamento consistente e de alta disponibilidade para todos os dados chave-valor do *cluster* Kubernetes;
- *kube-scheduler*: procura *pods* recém-criados sem nó designado e seleciona um nó para execução;
- *kube-controller-manager*: executa os controladores, cada controlador é um processo separado, mas para reduzir a complexidade, todos são compilados em um único binário e executados em um único processo. Esses controladores incluem *Node Controller*, *Replication Controller*, *Endpoints Controller*, *Service Account and Token Controller; e*
- *cloud-controller-manager*: executa os controladores que interagem com os provedores de nuvem subjacentes. Os controladores que têm dependências do provedor de nuvem são: *Node Controller*, *Route Controller*, *Service Controller e Volume Controller*.

Os componentes do tipo *node*, segundo Kubernetes (2019), são executados em todos os nós, mantendo os *pods* em execução e fornecendo o ambiente de tempo de execução Kubernetes. São eles:

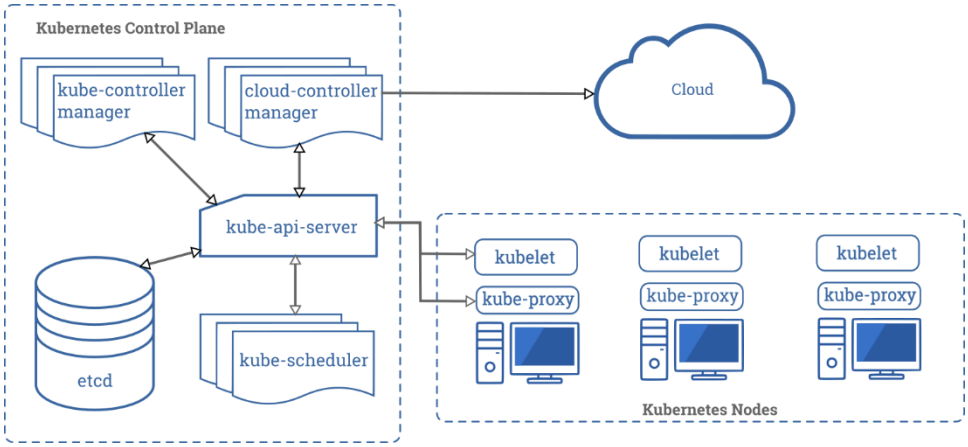
- *kubelet*: é o agente que garante que os contêineres estão sendo executados em um *pod*;
- *kube-proxy*: é um *proxy* de rede executado em cada nó do *cluster*, implementando parte do conceito de *Service*, mantendo regras de rede nos nós e permitindo a

comunicação em rede com seus *pods* a partir de sessões de rede dentro ou fora do *cluster*; e

- *Container Runtime*: é o *software* responsável pela execução de contêineres. O Kubernetes suporta vários *container runtimes*: Docker, containerd, cri-o, rktlet e qualquer implementação do Kubernetes *Container Runtime Interface* (CRI).

Na Figura 3.0 são apresentados os componentes arquiteturais que formam o *cluster* Kubernetes.

Figura 3.0 - Diagrama de um *cluster* Kubernetes com todos os componentes.



Fonte: Kubernetes Components, 2019.

Conforme Kubernetes (2019), a API REST é a estrutura fundamental do Kubernetes. A API é aberta e padronizada por um documento de convenção. Todas as operações e comunicações entre componentes e comandos do usuário externo são chamadas de API REST que o servidor de API manipula. Tudo na plataforma Kubernetes é tratado como um objeto de API e tem uma entrada correspondente na API. A maioria das operações pode ser executada através da interface da linha de comandos como kubectl ou o kubeadm e utilizam a API, também é possível acessar a API diretamente usando chamadas REST.

O estilo geral da API do Kubernetes é *RESTful*, onde os clientes criam, atualizam, excluem ou recuperam uma descrição de um objeto por meio dos verbos *Hypertext Transfer Protocol* (HTTP) padrão (*POST*, *PUT*, *DELETE* e *GET*). A API aceita e retorna *JavaScript Object Notation* (JSON), preferencialmente. O Kubernetes também expõe pontos de extremidade adicionais para verbos não padrão e permite tipos de conteúdo alternativos. Todo o JSON aceito e retornado pelo servidor possui um esquema, identificado pelos campos "*kind*" e "*apiVersion*". Onde existirem campos relevantes do

cabeçalho HTTP, eles devem refletir o conteúdo dos campos JSON, mas as informações não devem ser representadas apenas no cabeçalho HTTP (KUBERNETES, 2019).

A API é usada para representar os objetos do Kubernetes, os objetos básicos são: *Pods*: são as menores unidades computacionais implementáveis que podem ser criadas e gerenciadas no Kubernetes; *Services*: são uma maneira abstrata de expor um aplicativo em execução em um conjunto de *Pods* como um serviço de rede; *Ingress*: gerencia o acesso externo aos serviços em um *cluster*, geralmente HTTP, fornecendo balanceamento de carga; Volumes: são diretório acessíveis aos contêineres em um *pod*, resolve o problema da efemeridade; *Namespaces*: são *clusters* virtuais suportados por um mesmo *cluster* físico, são destinados para ambientes com muitos usuários espalhados por várias equipes ou projetos (KUBERNETES, 2019).

O Kubernetes tem outros objetos básicos denominados de *Controllers*, são eles: *ReplicaSet*: objetiva manter um número estável de *pods* execução a qualquer momento, é frequentemente usado para garantir a disponibilidade de um número especificado de *pods* idênticos; *Deployments*: fornece atualizações declarativas para *Pods* e *ReplicaSets* mantendo estado desejado; *StatefulSets*: é um objeto usado para gerenciar aplicativos com estado, gerenciando a implantação e o dimensionamento de um conjunto de *pods* e fornecendo garantias sobre a ordem e exclusividade desses *pods*; *DaemonSet*: garante que todos, ou alguns, nós executem uma cópia de um *Pod* (KUBERNETES, 2019).

3 Ferramentas e Desafios Forenses

Segundo Puccini (2018), nuvem, contêineres Linux e orquestração de contêineres são os tópicos mais discutidos atualmente. A maior parte das organizações de TI também está falando sobre *DevOps* e microsserviços. Conforme Winkel (2017), as organizações estão mais necessitadas de ferramentas e técnicas forenses para investigações de incidentes. O registro de eventos centralizado de contêineres Docker está se tornando crucial na resposta bem-sucedida a incidentes. Ferramentas como a pilha Elasticsearch, Logstash e Kibana podem ser parte de investigações de incidentes de imagens do Docker.

A resposta incidentes de segurança em um ambiente de contêiner, para Carpenter (2018), representa um desafio para as organizações de segurança tradicionais e para as equipes de segurança cibernética. As ferramentas e métodos tradicionais, como imagens de disco e memória, quando possível, não trazem resultados satisfatórios. Adiciona-se a isso, o fato de os contêineres não armazenarem estado e serem potencialmente efêmeros.

O contêiner que foi afetado por um incidente de segurança pode não existir mais no momento em que o problema for identificado, dificultando a coleta de artefatos dos recursos diretamente envolvidos.

Conforme Carpenter (2018), ferramentas como Twistlock possibilitam uma estrutura de forense distribuída, com gravação de dados do ambiente em execução em cada nó protegido pela ferramenta. Esse gravador mantém um registro contínuo de telemetria sobre a movimentação de contêineres e, quando um incidente em potencial é detectado, retém dados relevantes, permitindo que analistas de segurança cibernética identifiquem rapidamente um evento, investigando o risco para o contêiner, o *host* e para o ambiente.

Ao executar análise forense em contêineres, afirmam Wallace e Baer (2019), é preciso uma investigação estruturada e manter uma cadeia de evidências documentada para determinar exatamente o que aconteceu no ambiente e quem foi responsável. Nesse sentido, a análise forense e a resposta a incidentes são as mesmas para contêineres e para outros ambientes. A diferença dos contêineres é quais dados pode-se coletar e como reagir a um incidente. Mesmo antes de ocorrer um incidente, deve-se montar um plano de resposta a incidentes considerando os níveis de gravidade, ações planejadas e riscos específicos da organização.

A melhor forma para responder um incidente, para Wallace e Baer (2019), é com análise dos dados, artefatos como *logs*, discos e informações gravadas em tempo real revelam o que aconteceu como o ambiente, porém é necessário configurar com antecedência a persistência desses registros. Existem vários tipos de *logs* em um ambiente de contêineres que podem ser capturados. No caso da Google Cloud Plataform, é possível configurar *logs* de auditoria na nuvem para nós Google Kubernetes Engine e Compute Engine, incluindo *logs* de auditoria do Kubernetes, *logs* específicos do SO e os próprios *logs* da aplicação. No caso de incidente não se deve encerrar os *pods* ou reiniciar os nós, isso interromperá o problema em questão, mas também alerta um invasor em potencial de que sua ação é conhecida, limitando a capacidade de fazer perícia.

Para Kaczorowski e Wallace (2019), a aplicação de forense em contêineres requer os seguintes *logs*, que devem ser coletadas e armazenadas em uma infraestrutura de *logs*:

- *Logs* de infraestrutura: o que a infraestrutura faz e o que um ser humano faz com a infraestrutura;

- *Logs* do Kubernetes: o que o plano de controle faz, o que um contêiner faz com o plano de controle e o que o ser humano faz com o plano de controle;
- *Logs* do sistema operacional: o que um contêiner faz no nó; e
- *Logs* da aplicação: o que um aplicativo faz em um contêiner.

A forense computacional permite encontrar respostas para questionamentos essenciais (Onde? O que? Quando? Quem? Por quê?) após um incidente de segurança da informação. Porém, em ambientes que utilizam contêineres, como Docker e Kubernetes, essas perguntas se tornam bem mais específicas. As análises forenses e *post-mortem* são complexas em ambientes computacionais dinâmicos e distribuídos, isso se agrava ainda mais com a natureza efêmera dos contêineres (SYSDIG, 2019).

Os contêineres são criados e destruídos com frequência, os serviços são escalados automaticamente. Um contêiner afetado já pode ter sido destruído ou mesmo estar executando em um outro nó do *cluster*. As imagens de contêiner são imutáveis, mas seu tempo de execução não é, o sistema de arquivos e *logs* internos são efêmeros e são apagados assim que o contêiner for destruído. A comunicação dos contêineres se dá através de *Software Defined Networking* (SDN) gerenciada pelo *cluster*, seus endereços IP internos se modificam frequentemente. Reconstruir uma linha do tempo significativa de eventos e atividades do sistema é uma tarefa difícil e exige ferramentas forenses que suportem paradigma de microsserviços e contêineres (SYSDIG, 2019).

Outra ação para forense em contêineres, segundo Kaczorowski e Wallace (2019), é a confecção de um *snapshot* do nó afetado por incidente, para isso se deve identificar os nós afetados com todos os discos conectados, e então realizar um *snapshot* desses discos para análises. Com o *software* Docker Explorer é possível montar o *snapshot* e encontrar o identificador do contêiner, permitindo assim a montagem do seu sistema de arquivos, possibilitando uma análise forense dos arquivos nele presentes. O Docker Explorer é uma ferramenta forense que explora sistemas de arquivos Docker de forma *offline*, a visualização do sistema de arquivos de um contêiner do Docker comprometido pode ser útil análise forense (DOCKER EXPLORER, 2019).

Para Kaczorowski e Wallace (2019), em diversas situações é necessário realizar uma análise forense em tempo real e em informações gravadas, ferramentas como GRR Rapid Response (GRR) e Sysdig Inspect & Capture auxiliam nesses procedimentos,

ambos proveem diversas funcionalidades forenses que podem ajudar a descobrir problemas em todo o ambiente.

O GRR Rapid Response é um *framework* de resposta a incidentes focado em análise forense remota em tempo real. O objetivo do GRR é oferecer suporte a investigações forenses de forma rápida e escalável, para permitir que os analistas façam uma triagem rápida de ataques e realizem análises remotamente. O GRR funciona baseado em arquitetura cliente/servidor. O cliente GRR é implantado em sistemas que se pode querer investigar. Em cada sistema desse tipo, uma vez implantado, o cliente GRR consulta periodicamente os servidores *frontend* do GRR em busca de uma ação específica para executar. A infraestrutura do servidor GRR consiste em vários componentes (*frontends*, *works*, servidores de interface do usuário) e fornece interface gráfica do usuário baseada na *Web* e um terminal de API que permite aos analistas agendar ações nos clientes e visualizar e processar dados coletados (GRR, 2019).

Dependendo do tipo de dados do cliente a ser analisado, há vários locais em que podem ser examinados. O GRR fornece o sistema de arquivos virtual, que exibe os arquivos, diretórios e entradas do registro que já foram coletados de um cliente, possibilitando também a coleta de informações adicionais. Porém, quando se deseja coletar informações específicas sobre uma máquina, é necessário executar um fluxo direta ou indiretamente. O fluxo permite realizar uma ou mais operações em uma máquina cliente, para coletar ou verificar dados. Para executar um fluxo em vários clientes se faz necessário o mecanismo de *Hunt*. Outro conceito do GRR é o artefato, que é uma maneira de coletar e nomear um grupo de arquivos ou outros dados que um analista pode querer coletar como uma unidade (GRR, 2019).

Segundo Riadi et al. (2020), o GRR *Server* fornece uma interface fácil através do GRR *WebUI*, ajudando o profissional a obter informações apropriadas de todos os clientes GRR. No estudo de um ataque *Distributed Denial of Service* (DDoS) a uma aplicação *web* executando em um *cluster* Kubernetes foi possível obter evidências digitais (endereços IP, número da porta e registros de data e hora) em arquivos de *log* coletados pelo GRR. O *framework* GRR provê recursos completos e confiáveis para investigar evidências digitais de servidores da *Web* sendo executados dentro de *cluster* Kubernetes, com respostas rápidas e remotamente através de redes de computadores. Porém exigem configurações prévias no servidor GRR e no cliente GRR para a coleta de dados. Outro

fato a ser considerado é a possível modificação de endereços *Internet Protocol* (IP) de origem pelos componentes internos do Kubernetes. Se o *ServiceType* escolhido for *NodePort*, os pacotes enviados para o nó Kubernetes serão modificados, o IP de origem no pacote será alterado para o endereço IP do roteador. Portanto, o IP de origem visto no *Pod* de Kubernetes de destino não será o IP de origem verdadeiro do cliente. A tradução de endereços também foi percebida em um estudo analisando um ataque DDoS a uma aplicação *web* em execução no Docker Swarm (SUNARDI et al., 2019).

Para enfrentar os desafios da análise forense de contêineres e análise *post-mortem* o Sysdig registra as atividades pré e pós-ataque por meio de chamadas do sistema, permitindo analisar tudo o que ocorreu no contêiner, facilitando assim a compreensão de como ocorreu o incidente. Isso é possível através do *ContainerVision*, que executa no nível do *kernel*. O Sysdig tem integração com diversas ferramentas de orquestração do mercado. Esse conjunto de ferramentas forenses para contêiner é composto de diferentes produtos e licenças: Sysdig Falco, Sysdig Inspect e Sysdig Secure (SYSDIG, 2019).

A análise forense de contêineres e análise *post-mortem* em Kubernetes pode ser feita com o Sysdig Secure, que audita automaticamente todos os comandos e processos executados em toda a infraestrutura sem a necessidade de mudanças em aplicativos ou contêineres. Na ocorrência de um incidente o Sysdig Secure pode coletar mais informações, resultando em um arquivo que contém todas as chamadas do sistema em ordem cronológica. O Sysdig Inspect permite abrir arquivos de captura e para análises forenses de contêineres, possibilitando uma análise forense de diferentes cenários de ataques (SYSDIG, 2019).

No Quadro 1.0 são apresentadas as principais características das ferramentas normalmente utilizadas em computação forense em ambientes conteinerizados.

Quadro 1.0 - Ferramentas para computação forense em contêineres e suas principais características

Ferramenta	**Licenciamento**	**Funcionalidades**
Twistlock	Proprietário	Gravação contínua de dados de telemetria dos contêineres, dos *hosts* e do ambiente, registrando a movimentação de contêineres e dados relevantes de potenciais incidentes.

		Verificação automática de imagens, desde o *build*, registro e execução.
GRR Rapid Response	Apache License 2.0	Análise forense remota escalável em tempo real. Sistema de arquivos virtual, que exibe os arquivos, diretórios e entradas do registro que já foram coletados de um cliente, possibilitando também a coleta de informações adicionais. Execução de comandos e *scripts* e customizados em um ou mais clientes.
Docker Explorer	Apache License 2.0	Montagem e exploração de sistema e arquivos de contêineres Docker de forma *offline* a partir de *snapshots.*
Sysdig Inspect & Capture	GPL 2.0	Captura de chamadas de sistema. Leitura de arquivos de captura para análises forenses de contêineres em diferentes cenários de ataques.
Sysdig Secure	Proprietário	Auditoria automática de todos os comandos e processos executados em toda a infraestrutura. Coleta e armazenamento de informações em um arquivo que contém todas as chamadas do sistema em ordem cronológica permitindo análise forense em tempo real e análise *post-mortem*.

Fonte: Elaborado pelo autor.

4 CONCLUSÕES

Com esse artigo pode-se concluir que a tecnologia de contêineres trouxe diversos desafios para computação forense, dentre os principais estão: a natureza efêmera dos contêineres; as características arquiteturais dos componentes dos contêineres e a complexidade dos orquestradores de contêineres.

A natureza efêmera dos contêineres pode ser elencada como o maior desafio da computação forense. Os arquivos de *logs* e os sistemas de arquivos são apagados quando os contêineres ou *pods* são destruídos. Para que esses dados sejam persistidos é necessário

configurar previamente a persistência em volumes externos ou banco de dados. Nesse contexto o planejamento prévio de uma infraestrutura externa para coleta e armazenamento de logs da infraestrutura, do orquestrador, do contêiner, do SO e da aplicação, se mostra essencial para a preservação de evidências dos eventos anômalos ou criminosos para uma correta análise forense.

As características arquiteturais dos componentes dos contêineres também é um desafio para a computação forense. Os contêineres apresentam camadas de isolamento de rede, de processos e de sistemas de arquivos, para que determinado contêiner em execução não comprometa o *kernel* ou outros processos em execução no hospedeiro. O conhecimento desses isolamentos e dos componentes estruturantes da arquitetura do ambiente de contêiner, assim como sua interação com o *kernel* do SO se mostram imprescindíveis para uma correta e completa análise forense.

Além disso, a complexidade dos orquestradores torna a análise forense desse ecossistema um grande desafio. Ademais a necessidade de conhecer os componentes do ambiente de contêineres, os orquestradores surgem com arquiteturas distribuídas complexas, com diversos componentes que interagem entre si por meio de API REST. Os orquestradores executam em modo *cluster*, seus componentes mestres e nós normalmente são distribuídos em diferentes servidores físicos ou virtuais. Proporcionando alta disponibilidade, autoescalabilidade, monitoramento e verificação de integridade de contêineres e dos componentes do orquestrador. Assim como nos contêineres, a compreensão das responsabilidades e do funcionamento de cada componente é essencial para a completude da análise forense.

Algumas ferramentas, como o GRR Rapid Reponse e o Sysdig Secure surgem para auxiliar na análise forense dos contêineres e na análise *post-mortem*, alcançando sucesso na identificação e análise de evidências digitais em ambientes conteinerizados. Porém, percebe-se que muito do sucesso da coleta, preservação e análises das evidências exige planejamento e configurações prévias ao incidente cibernético, sendo altamente recomendável que cada organização tenha um plano de ação para resposta a incidentes de segurança em seu ambiente de contêineres.

Conclui-se com esse trabalho que a tecnologia de contêineres apresenta grandes desafios para computação forense e que muito do sucesso das análises forenses depende de planejamento e configurações prévias para coletar, preservar e assim permitir análises

forenses em tempo real ou *post-mortem*. Como sugestão para futuras pesquisas pode-se relacionar os seguintes estudos: análise comparativas entre as principais ferramentas de forense em contêineres; identificação dos principais ataques cibernéticos em ambientes conteinerizados; avaliação do *overhead* causados por sistemas de segurança em ambientes com orquestradores de contêineres.

REFERÊNCIAS

ALBUQUERQUE FILHO, Alexandre Cisneiros. **Estudo comparativo entre Docker Swarm e** Kubernetes **para orquestração de contêineres em arquiteturas de software com microsserviços**. 2016. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Universidade Federal de Pernambuco, Recife, 2016.

BACHIEGA, N. G. et al. Container-based performance evaluation: A survey and challenges. **IEEE International Conference on Cloud Engineering.** p. 398–403. Abril de 2018.

BETTINI, Anthony. **Vulnerability Exploitation in Docker Container Environments**. 2015. Disponível em: https://www.blackhat.com/docs/eu-15/materials/eu-15-Bettini-Vulnerability-Exploitation-In-Docker-Container-Environments-wp.pdf. Acesso em: 29 julho 2019.

BUI, Thanh. Analysis of Docker Security. **Aalto University T-110.5291 Seminar on Network Security.** Janeiro de 2015.

CARPENTER, Neil. **Cloud Native Forensics: Security Incident Response in Twistlock 2.5**. 2018. Disponível em: https://www.twistlock.com/2018/08/09/cloud-native-forensics-security-incident-response-twistlock-2-5. Acesso em: 08 nov. 2019.

DOCKER. **What is a Container? A standardized unit of software**. Disponível em: https://www.docker.com/resources/what-container. Acesso em: 15 out. 2019.

DOCKER-EXPLORER. **README.md**. 2019. Disponível em: https://github.com/google/docker-explorer. Acesso em: 10 dez. 2019.

GRR. **What is GRR?** 2019. Disponível em: https://grr-doc.readthedocs.io/en/latest/what-is-grr.html. Acesso em: 10 nov. 2019.

HYKES, Solomon. **Docker 0.9: introducing execution drivers and libcontainer**. 2014. Disponível em: https://www.docker.com/blog/docker-0-9-introducing-execution-drivers-and-libcontainer/. Acesso em: 16 nov. 2019.

JESUS, Antonio Velho et al (Org.). **Tratado de Computação Forense**. Campinas. São Paulo: Milennium, 2016.

KACZOROWSKI, Maya; WALLACE, Ann. **Container Forensics: When your cluster becomes a cluster**. 2019. Disponível em: https://sched.co/MPdK. Acesso em: 27 nov. 2019.

KUBERNETES. **Concepts**. Disponível em: https://kubernetes.io/docs/concepts. Acesso em: 20 dez. 2019.

PUCCINI, Renato. **10 dicas para usar containers na otimização do seu aplicativo**. 2018. Disponível em: https://imasters.com.br/devsecops/10-dicas-para-usar-containers-na-otimizacao-seu-aplicativo. Acesso em: 15 julho 2019.

RED HAT. **Kubernetes e a tecnologia de containers**. Disponível em: https://www.redhat.com/pt-br/topics/containers/what-is-kubernetes. Acesso em: 30 nov. 2019.

RIADI, I; UMAR, R; SUGANDI, A. Web Forensic On Kubernetes Cluster Services Using Grr Rapid Response Framework. **International Journal of Scientific & Technology Research**. v.9, p. 3484-3488. Janeiro de 2020.

SOUPPAYA, M; MORELLO, J; SCARFONE, K. **Application Container Security Guide**. NIST Special Publication 800-190. 2017. Disponível em: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-190.pdf. Acesso em: 10 jan. 2020.

SOUZA, Gabriel de. **Proposta de nuvem privada multicâmpus para o IFSC usando orquestração de contêineres**. 2018. Disponível em: https://wiki.sj.ifsc.edu.br/wiki/images/e/e3/Projeto_de_TCC_Gabriel_de_Souza_v1.0.pdf. Acessado em: 21 out. 2019.

SUNARDI, S; RIADI, I; SUGANDI, A. Forensic Analysis of Docker Swarm Cluster using Grr Rapid Response Framework. **International Journal of Advanced Computer Science and Applications**. v.10, n.2, p. 549-466. Mar. de 2019.

SPIEKERMANN, D; EGGENDORFER, T; KELLER, J. A Study of Network Forensic Investigation in Docker Environments. **ARES '19: Proceedings of the 14th International Conference on Availability, Reliability and Security.** n. 44. p. 1–7. Agosto de 2019.

SYMANTEC. **Internet Security Threat Report** - ISTR Volume 24 - February 2019. Disponível em: https://www.symantec.com/content/dam/symantec/docs/reports/istr-24-2019-en.pdf. Acesso em: 20 set. 2019.

SYSDIG. **Container Forensics**. Disponível em: https://sysdig.com/use-cases/container-forensics/. Acesso em: 10 set. 2019.

TURNBULL, J. The Docker Book Containerization is the new virtualization. [S.l.]: Amazon Digital Services LLC, 2016.

VITALINO, J. F. N; CASTRO, M. A. N. Descomplicando o Docker. 2. ed. Rio de Janeiro: Brasport Livros e Multimídia Ltda, 2018.

VMware. **Containers on Virtual Machines or Bare Metal? Deploying and Securely Managing Containerized Applications at Scale**. 2018. Disponível em: https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/whitepaper/vmw-wp-container-on-vms-a4-final-web.pdf. Acesso em: 10 jan. 2020.

WALLACE, A; BAER, C. **Exploring container security: Performing forensics on your GKE environment**. 2019. Disponível em: https://cloud.google.com/blog/products/containers-kubernetes/best-practices-for-performing-forensics-on-containers. Acesso em: 10 jan. 2020.

WINKEL, Stefan. **Forensicating Docker with ELK**. 2017. Disponível em: https://www.sans.org/reading-room/whitepapers/forensics/paper/37870. Acesso em: 15 julho 2019.