Proposta de análise de desempenho no uso de políticas segurança de rede nativas Kubernetes vs. SPIFFE

Nikolas Jensen[1], **Charles C. Miers**[1]

[1] Departamento de Ciência da Computação (DCC)
Universidade do Estado de Santa Catarina (UDESC)

nikolas.j@edu.udesc.br, charles.miers@udesc.br

Resumo. *A preocupação com a segurança em ambientes de nuvem é constante, mas também há a necessidade de um bom desempenho. Sendo assim, o Kubernetes oferece a solução de políticas de rede para tratar estes aspectos, a qual possui dificuldades ao atender diversos endereços IP. Com isso, o SPIFFE busca oferecer mais segurança às aplicações em nuvem e com mais eficiência, oferecendo um serviço de autenticação para componentes de software. Este trabalho a presenta a proposta inicial de uma análise comparativa de desempenho entre políticas nativas do Kubernetes e a nova proposta do SPIFFE.*

1. Introdução

Os orquestradores de contêineres podem realizar diversos serviços, sendo possível oferecer diversas possibilidades aos desenvolvedores. O Kubernetes é um dos orquestradores mais utilizados atualmente em diversas empresas. Em entrevista com 500 profissionais de tecnologia, 88% destes afirma que utiliza o utilizam em sua empresa. O Kubernetes vem sendo abordado em pesquisas na academia, não somente em relação a segurança, mas como também desempenho. Além disso, o Kubernetes apresentou melhores resultados em eficiência em trabalhos analisados [Mercl and Pavlik, 2019]. O *Secure Production Identity Framework for Everyone* (SPIFFE) apresenta uma solução para aprimorar a segurança de ambientes de nuvem, utilizando identidades transitivas para autenticar componentes de *software*. Sendo assim, este trabalho visa comparar as políticas de rede do Kubernetes com o SPIFFE, em relação a eficiência deste no ambiente de nuvem.

Este trabalho está organizado como segue. A Seção 2 aborda o funcionamento do Kubernetes com uma visão geral dos seus principais componentes. A Seção 3 cobre a funcionalidade básica do SPIFFE. Por fim, a Seção 4 expõe a proposta de experimento.

2. Kubernetes

Cada componente do Kubernetes possui uma função, e juntos, realizam o objetivo final de oferecer mais eficiência e usabilidade de recursos para os sistemas. O Kubernetes possui *pods*, os quais são a menor unidade de desenvolvimento em um *cluster* que é possível criar e gerenciar [Kubernetes, 2021]. Estes isolam os contêineres e cada um contém um *namespace* próprio. Além disso, possui o nó controlador o qual possui diversos componentes dentro (Figura 1), dentre estes o servidor de *Application Programming Interface* (API), *etcd* para armazenamento dos dados do *cluster*, escalonador e um controlador para nós, processos, *endpoints* e dos dados de contas [Kubernetes, 2021]. O servidor da API permite expor o *cluster* para alguma rede, podendo ser interna ou externa. Esta será

responsável por orquestrar todas as operações do *cluster*, e.g., execução de comandos num contêiner especifico do *cluster*.

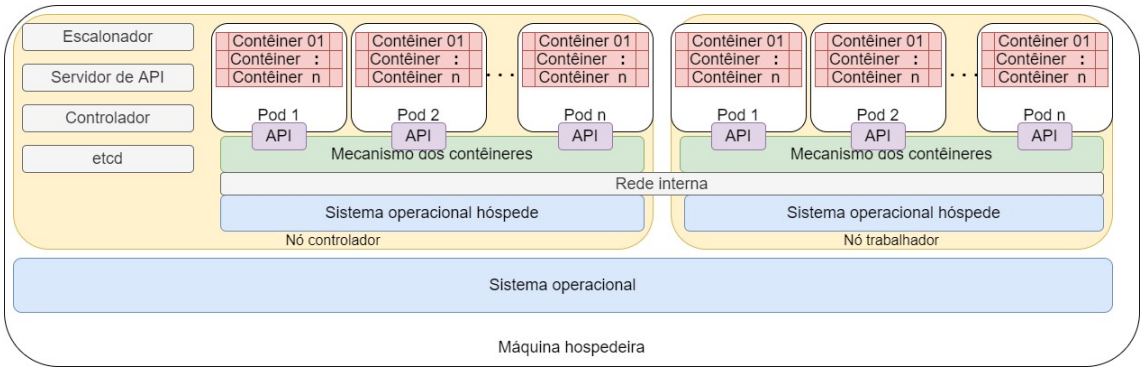


Figura 1. Diagrama da arquitetura de um *cluster* Kubernetes

Na Figura 1 descreve-se como um *cluster* Kubernetes pode ser definido, possuindo uma única máquina física e duas máquinas virtuais servindo como nó controlador e trabalhador. É possível notar que os nós são unidos por uma rede interna do *cluster*, isto serve para que cada nó se conheça mutuamente e possam trocar informações. O trabalho dos balanceadores de carga nesta rede interna é direcionar o tráfego de trabalho para os diversos *pods* e contêineres. Assim, a rota de comunicação sempre é a mesma e sempre passa por alguns componentes específicos.

As políticas do Kubernetes servem para garantir mais segurança ao *cluster* impondo regras. Para aprimorar a segurança da rede do *cluster* é possível implementar políticas de segurança de rede, como restringir os *Internet Protocol*s (IPs) com os quais determinado nó, *pod* ou contêiner pode se comunicar [Shamim et al., 2020]. Uma política de segurança é importante para que limite-se a superfície de ataque do contêiner, prevenindo que ações indesejadas sejam realizas por atacantes. Por outro lado, diversas políticas de desempenho podem afetar o desempenho das operações sendo relevante entender esse impacto.

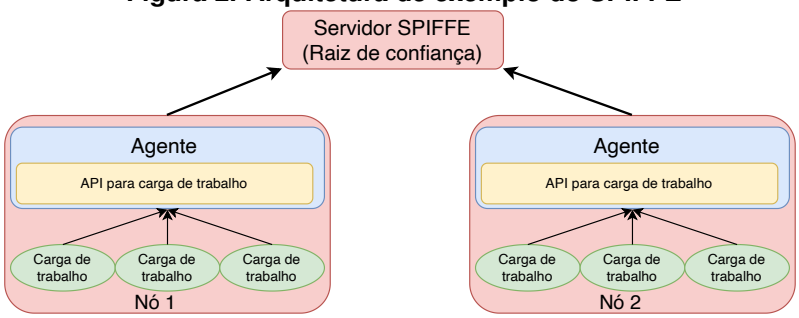
3. SPIFFE

Nuvens podem proporcionar diversos benefícios, e.g., escalabilidade, eficiência, replicabilidade, etc. Porém, existem problemas de segurança de vêm sendo estudados pela academia para tornar estes ambientes mais seguros aos usuários. A falta do correto uso da autenticação, identificação e autorização em ambientes de nuvem é um problema que gera ameaças. A partir de certo momento, o uso de políticas de rede para a segurança da aplicação se torna custoso com sua escalabilidade.

Tendo isto em vista, o SPIFFE é uma especificação que busca uma identificação segura para os componentes de um ambiente da nuvem, a especificação do SPIFFE [Feldman et al., 2020] documenta suas funcionalidades e componentes em detalhes. Na forma de certificados X.509, este oferece identidades aos componentes de software para que o sistema na nuvem se comunique com componentes legítimos. Para isto, utiliza alguns componentes, i.e., domínio de confiança, SPIFFE ID, Documento de identidade verificável SPIFFE (SVID), pacote de confiança SPIFFE e API para cargas de trabalho SPIFFE (Figura 2). O domínio de confiança pode emitir e validar identidades e deve estar num servidor totalmente seguro e confiável. Cada nó do ambiente possui um agente

que se comunica com o domínio de confiança, enquanto a API recebe uma requisição de uma carga de trabalho, e então fornece a identidade da determinada carga de trabalho solicitante. A SVID é o documento da identidade em forma de certificado, a qual possui diversas informações acerca do objeto identificado, incluindo seu domínio de confiança. As cargas de trabalho podem utilizar a sua SVID para autenticar-se com outra carga e então realizar uma comunicação segura entre si, ou então para assinar e verificar um *Json Web Token* (JWT). Sendo assim, um ambiente de nuvem utilizando SPIFFE identifica todo componente de software que pertencê-lo.

Figura 2. Arquitetura de exemplo do SPIFFE



Na Figura 2 é ilustrado um exemplo de arquitetura do SPIFFE. Neste caso, utiliza-se dois nós com agentes SPIFFE e um terceiro com um servidor SPIFFE para emitir e validar as identidades. As cargas de trabalho devem comunicar-se com a API de cada agente para receber sua identidade. Deste modo, o SPIFFE busca fazer que o fluxo de trabalho (*workload*) possa se autenticar sozinho, sem precisar de intermediação do Kubernetes. Com tal fato espera-se proporcionar um desempenho melhor do que este fluxo de trabalho precisar que externamente seja autorizado passar por todos os nodos envolvidos no fluxo do trabalho pelo Controlador do Kubernetes.

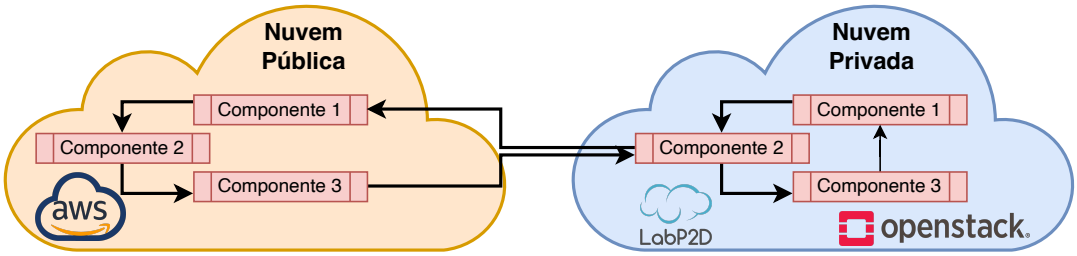
4. Proposta de experimento

Visto que o SPIFFE, possui como um dos objetivos diminuir o *overhead* causado por políticas de rede tradicionais, a proposta é testar o desempenho do SPIFFE em relação às políticas de rede [Feldman et al., 2020][1] do Kubernetes. Um exemplo de política de rede é somente permitir o acesso de endereços IP pertencentes a determinado alcance, ou então, permitir determinado tipo de tráfego entre *pods*.

A proposta é criar ambientes heterogêneos e dinâmicos, propícios para o uso do SPIFFE e similar a um ambiente real como descrito em [Feldman et al., 2020]. Para isto, é necessário dois ambientes de nuvem, uma pública e outra privada local, então realizar a comunicação entre estas, também, entre os componentes da própria nuvem. A comunicação deve ser realizada de forma que possua mensagens bloqueadas, devido a falta de autenticação ou permissão. Com isso, o tempo até o fim do experimento deve ser monitorado para que seja possível realizar a comparação entre o uso das políticas de rede e do SPIFFE (Figura 3). Está previsto ser utilizada a nuvem privada do LabP2D/UDESC baseada em OpenStack e a nuvem pública AWS.

[1] https://kubernetes.io/pt-br/docs/concepts/services-networking/network-policies/

Figura 3. Arquitetura do experimento a ser realizado.



A Figura 3 mostra a arquitetura do experimento a ser realizado. Devem ocorrer duas execuções, uma utilizando as políticas de segurança padrão do Kubernetes e outra com o SPIFFE. As mensagens devem seguir determinado fluxo para que sejam bloqueadas ou aceitas. Serão monitorados os seguintes aspectos: latência das requisições (decompostas em estágios), volume de tráfego por requisição, uso de processador e rede dos contêineres envolvidos e Controlador Kubernetes. O monitoramento será realizado utilizando as métricas providas pelo Kubernetes e pelos contêineres, e.g., uso da CPU.

5. Considerações e Próximos passos

O Kubernetes pode ser utilizado para diversos fins, com isso a atenção com a segurança do *cluster* deve ser constante, devido ao grande fluxo de informações. Sendo assim, as políticas de rede buscam barrar endereços IP não desejados, mas, devido à sua arquitetura de funcionamento, pode não escalar como o desejado em aplicações da nuvem heterogêneas e dinâmicas. Por outro lado, o SPIFFE apresenta outra solução para estes problemas com uma arquitetura e funcionalidades diferentes das políticas de rede. Utilizando identidades transitivas baseadas em confianças, os nós não precisam realizar verificações a todo momento se cada endereço IP que envia pacotes possui devida autorização. Sendo assim, é relevante verificar o desempenho do uso do SPIFFE em ambientes dinâmicos e heterogêneos em relação a opção mais utilizada que são os recurso nativos do Kubernetes. Para trabalhos futuros serão realizados experimentos utilizando outras tecnologias, como Docker Swarm, Apache Mesos, Openshift, etc.

Agradecimentos:
Os autores agradecem o apoio do LabP2D/UDESC e da FAPESC.
Este trabalho é parcialmente patrocinado pela Hewlett-Packard Enterprise.

Referências

Feldman, D., Fox, E., Gilman, E., Haken, I., Kautz, F., Khan, U., Lambrecht, M., Lum, B., Fayó, A. M., Nesterov, E., Vega, A., and Wardrop, M. (2020). *Solving the Bottom Turtle - a SPIFFE Way to Establish Trust in Your Infrastructure via Universal Identity*. Sprint Lab, Nova Zelândia.

Kubernetes (2021). Kubernetes documentation. https://kubernetes.io/docs.

Mercl, L. and Pavlik, J. (2019). The Comparison of Container Orchestrators. In Yang, X.-S., Sherratt, S., Dey, N., and Joshi, A., editors, *3rd ICICT*, Advances in Intelligent Systems and Computing, pages 677–685. Springer.

Shamim, M. S. I., Bhuiyan, F. A., and Rahman, A. (2020). Xi commandments of kubernetes security: A systematization of knowledge related to kubernetes security practices. *2020 IEEE Secure Development (SecDev)*, pages 58–64.