Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Ciência da Computação

Estratégias para o Suporte a Ambientes de Execução Confiável em Sistemas de Computação na Nuvem

Lília Rodrigues Sampaio

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande - Campus I como parte dos requisitos necessários para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Sistemas Distribuídos e Computação em Nuvem

Andrey Elísio Monteiro Brito
(Orientador)
Francisco Vilar Brasileiro
(Orientador)

Campina Grande, Paraíba, Brasil

FICHA CATALOGRÁFICA ELABORADA PELO BIBLIOTECÁRIO GUSTAVO DINIZ DO NASCIMENTO CRB - 15/515

S192e Sampaio, Lília Rodrigues.
Estratégias para o suporte a ambientes de execução confiável em sistemas de computação na nuvem / Lília Rodrigues Sampaio. ï Campina Grande, 2018.
91 f. : il. color.

Dissertação (Mestrado em Ciência da Computação) ï Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática, 2018.
"Orientação: Prof. Dr. Andrey Elísio Monteiro Brito, Prof. Dr. Francisco Vilar Brasileiro".
Referências.

1. Computação na Nuvem. 2. Segurança da Informação. 3. Intel SGX. I. Brito, Andrey Elísio Monteiro. II. Brasileiro, Francisco Vilar. III. Título.

CDU 004.771(043)

"ESTRATÉGIAS PARA O SUPORTE A AMBIENTES DE EXECUÇÃO CONFIÁVEL EM SISTEMAS DE COMPUTAÇÃO NA NUVEM"

LILIA RODRIGUES SAMPAIO

DISSERTAÇÃO APROVADA EM 15/02/2018

FRANCISCO VILAR BRASILEIRO, Ph.D, UFCG
Orientador(a)

ANDREY ELÍSIO MONTEIRO BRITO, Dr., UFCG
Orientador(a)

LEANDRO BALBY MARINHO, Dr., UFCG
Examinador(a)

KEIKO VERÔNICA ONO FONSECA, Dra., UTFPR
Examinador(a)

CAMPINA GRANDE - PB

Resumo

O alto poder computacional oferecido por provedores de nuvem, aliado às suas características de flexibilidade, eficiência e custo reduzido, têm levado cada vez mais usuários a utilizar recursos em nuvem para implantação de aplicações. Como consequência, uma grande quantidade de dados de diversas aplicações críticas, e de alta demanda de processamento, passam a trafegar livremente pela nuvem. Considerando isso, especialmente para aplicações que lidam com dados sensíveis, como sistemas bancários, medidores inteligentes de energia, entre outros, é de grande importância assegurar a integridade e confidencialidade de tais dados. Assim, é cada vez mais frequente que usuários de recursos de uma nuvem exijam garantias de que suas aplicações estão executando em um ambiente de execução confiável.

Abordagens tradicionais, como criptografia de dados em repouso e em comunicação, combinadas com políticas rígidas de controle de acesso têm sido usadas com algum êxito, mas ainda têm permitido sérios ataques. No entanto, mais recentemente, *Trusted Execution Environments* (TEE) têm prometido garantias de integridade e confidencialidade de dados e códigos ao carregá-los e executá-los em áreas seguras e isoladas do processador da máquina. Assim, para dar suporte a implementações de TEE, tecnologias de segurança em hardware podem ser utilizadas, como ARM Trustzone e Intel SGX. No contexto deste trabalho, usamos Intel SGX, que se propõe a garantir confidencialidade e integridade de dados e aplicações executadas dentro de áreas protegidas de memória, denominadas enclaves. Assim, o código é protegido até de software com acesso privilegiado, como o próprio sistema operacional, hipervisores, entre outros.

Diversos recursos da nuvem podem fazer uso de tais tecnologias de segurança, entre eles máquinas virtuais e contêineres. Neste estudo, propomos estratégias para o suporte de TEE em ambientes de nuvem, integrando Intel SGX e OpenStack, uma plataforma de nuvem de código aberto amplamente conhecida e utilizada por grandes empresas. Apresentamos uma nova abordagem no processo de provisionamento e escalonamento de máquinas virtuais e contêineres numa nuvem OpenStack segura, que considera aspectos essenciais para o SGX, como a quantidade de memória segura sendo utilizada, o *Enclave Page Cache* (EPC). Por fim, validamos esta nuvem com uma aplicação que exige processamento de dados sensíveis.

Abstract

The high level of computing power offered by cloud providers, together with the flexibility, efficiency and reduced cost they also offer, have increased the number of users wanting to deploy their applications on the cloud. As a consequence, a big amount of data from many critical and high performance applications start to traffic on the cloud. Considering this, specially for applications that deal with sensitive data, such as bank transactions, smart metering, and others, is very important to assure data integrity and confidentiality. Thus, it is increasingly common that users of cloud resources demand guarantees that their applications are running on trusted execution environments.

Traditional approaches, such as data cryptography, combined with strict access control policies have been used with a level of success, but still allowing serious attacks. However, more recently, Trusted Execution Environments (TEE) are promising guarantees of data and code integrity and confidentiality by loading and executing them in isolated secure areas of the machine's processor. This way, to support TEE implementations, hardware technologies such as ARM TrustZone and Intel SGX can be used. In the context of this research, we use Intel SGX, which proposes integrity and confidentiality guarantees for data and applications executed inside protected memory areas called enclaves. Thus, the code is protected even from high privileged software, such as the operational system, hypervisors and others.

Many cloud resources can use such security technologies, like virtual machines and containers. In this research, we propose strategies to support TEE in cloud environments, integrating Intel SGX and OpenStack, an open-source cloud platform widely known and used by many big companies. We present a new approach in the provisioning and scheduling of instances in a secure OpenStack cloud, considering essential aspects to SGX such as the amount of secure memory being used, the Enclave Page Cache (EPC). Finally, we validated this cloud by deploying an application that requires processing of sensitive data.

Agradecimentos

À Deus, por me mostrar os caminhos que sempre me fizeram perseverar.

Aos meus pais, palavras não conseguem expressar. Sem eles nada, por eles tudo.

Aos meus irmãos, por me ensinarem no companheirismo de sempre o poder da determinação e da dedicação. Por mais sonhos nossos realizados, juntos, sempre.

À toda minha família, que me enchem de amor e muito mimo, como a caçulinha que sempre serei.

Aos meus amigos, por serem comigo, companheiros e verdadeiros. Em especial às minhas Bests, meus Chimbas, minhas Chaminhas, meu Femme e ao eterno Amizade. Grata por ter vocês sempre por perto.

Ao meu orientador, Andrey, por sempre acreditar no meu melhor e me inspirar todos os dias a ser segura, motivada e ambiciosa. Obrigada por todos os trinta segundinhos de conversas cheias de sabedoria, conhecimento e amizade, por me compreender e ajudar quando precisei, e sem dúvida por ser peça chave na profissional que sou hoje.

Aos meus companheiros de SecureCloud, em especial Fábio Silva, Rodolfo Marinho, Marcus Tenório, Gabriel Fernandez, Clenimar Filemon e Amanda Souza, por todo empenho e afinco que colocam em tudo que fazem, e assim me enriquecerem todos os dias com conhecimento e discussões proveitosas que fizeram de mim uma profissional melhor. Além disso, pelo ombro amigo para chorar os medos e angústias, e comemorar as (muitas!) alegrias e sucessos. Vocês são peso, e deu certo! (e claro, não poderia faltar, #timecloud :)

Ao Laboratório de Sistemas Distribuídos, por todo amparo profissional e tecnológico de qualidade, mas em especial por sempre me fazer sentir em casa. Quando graduanda, ainda no terceiro período, ser LSD foi sem dúvidas a melhor escolha que eu poderia ter feito.

Por fim, agradeço à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) e ao projeto SecureCloud (3ª Chamada Coordenada Brasil-Europa, MCTIC/RNP e Comissão Europeia) pelo financiamento desta pesquisa.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Códigos Fonte

Capítulo 1

Introdução

1.1 Motivação e contextualização

A grande capacidade de poder computacional oferecido por provedores de nuvem, em conjunto com suas características de escalabilidade, eficiência, flexibilidade e baixo custo, têm atraído cada vez mais clientes desejando utilizar tais serviços. O relatório produzido pela RightScale [34] sobre o estado da arte da computação em nuvem no ano de 2017 mostra que companhias agora executam 79% de sua carga na nuvem, sendo 41% em nuvens públicas e 38% em privadas.

Nessa mesma pesquisa, como em anos anteriores, participantes confirmam crescimento nos benefícios que suas organizações adquirem a partir da utilização de ambientes em nuvem, ao passo em que se tornam mais experientes em seu uso. Ainda assim, como consequência da grande quantidade de dados de diversas aplicações robustas, e de alto nível de processamento, que passam a trafegar livremente pela nuvem, segurança é apontada como uma das maiores preocupações na escolha desses ambientes.

Considerando isso, especialmente para aplicações que lidam com dados críticos, como sistemas governamentais, bancários, medidores inteligentes de energia, entre outros, é de grande importância garantir a integridade e confidencialidade de tais dados [13; 23; 38]. Fica claro que é cada vez mais valorizado, por parte dos usuários de recursos de uma nuvem, as garantias de que suas aplicações estejam executando em um ambiente de execução confiável.

Abordagens tradicionais, como *software anti-malware*, podem não identificar ameaças antes delas se espalharem por um ambiente de nuvem [18]. Métodos que envolvem criptografia de dados em repouso e em comunicação, combinadas com políticas de controle de acesso são usadas com algum sucesso, mas ainda têm permitido sérios ataques. Por exemplo, recentemente surgiram casos de

vazamento de dados pessoais de usuários em grandes sistemas implantados em ambientes de nuvem [21].

Em novembro de 2017, o serviço de compartilhamento de caronas Uber[1] revelou ter conhecimento sobre uma violação em seus dados que expôs as informações pessoais de potencialmente 57 milhões de usuários e motoristas. Hackers tiveram acesso ao código do Uber armazenado no GitHub, que incluía nome, endereços de email e números de telefone de usuários ao redor do mundo. Em seguida, em dezembro do mesmo ano, o eBay[2] anunciou que, em virtude de um vazamento de dados, informações pessoais de muitos de seus usuários, incluindo nomes e histórico de compras, foram disponibilizados online para acesso público. No histórico era possível ver a compra de produtos mais sensíveis como testes de HIV, gravidez e drogas, o que pode ser comprometedor para os usuários.

Assim, de forma a garantir que aplicações sensíveis estejam executando em ambientes protegidos e confiáveis, algumas técnicas e tecnologias de segurança podem ser utilizadas. Surge o conceito de *Trusted Execution Environments* (TEE) [36], que têm prometido garantias de integridade e confidencialidade de dados e códigos ao carregá-los e executá-los em áreas seguras e isoladas do processador da máquina.

Para dar suporte a implementações de TEE, tecnologias de segurança em hardware podem ser utilizadas, como AMD SEV/SME [29] e ARM TrustZone [7]. O primeiro comprovadamente não oferece todas as medidas de segurança prometidas [26], e o segundo está disponível apenas para aparelhos móveis. Nesse contexto, está a tecnologia Intel SGX [19], que se propõe a garantir confidencialidade e integridade de dados e aplicações ao executá-las dentro de áreas protegidas de memória, denominadas enclaves.

Intel SGX é uma tecnologia consideravelmente recente, disponível em processadores desde 2015 [31]. Funcionalidades de segurança e disponibilidade oferecidas pelo SGX o tornam adequado para suportar aplicações que lidam com dados sensíveis na nuvem. Ele provê proteção a dados e código contra sua divulgação e/ou modificação por terceiros não autorizados. Tal tecnologia permite que as aplicações executando em enclaves sejam protegidas até de software com níveis altos de privilégio como o próprio sistema operacional e hipervisores.

Em ambientes de nuvem, diversos recursos podem fazer uso de tais tecnologias e mecanismos de segurança, entre eles máquinas virtuais e contêineres. Criar máquinas virtuais com capacidade SGX ainda é considerado em processo de desenvolvimento pela Intel. Entretanto, os esforços para dar suporte à isso são crescentes e já atingiram versões de teste disponíveis em [3; 4]. Esse *kernel*

[1] Uber - https://www.uber.com/ [Online; Último acesso: 22 de janeiro, 2018]

[2] eBay - https://www.ebay.com/ [Online; Último acesso: 22 de janeiro, 2018]

modificado, chamado *kvm-sgx* [3], funciona como um KVM que disponibiliza o recurso SGX da máquina hospedeira para as aplicações rodando nas máquinas virtuais.

Em outra frente estão contêineres e seus respectivos orquestradores, em linhas gerais, ferramentas que gerenciam contêineres instanciados em um *cluster* de máquinas como um único componente. Uma das características chave de um modelo de orquestração é a sua capacidade de automação na realização das mais diversas tarefas, como inicialização, escalonamento e implantação de contêineres servindo a aplicações.

Nesta pesquisa propomos estratégias para o suporte de TEE em ambientes de nuvem, integrando Intel SGX e OpenStack, uma plataforma de nuvem de código aberto amplamente conhecida e utilizada por grandes empresas. Apresentamos uma nova abordagem no processo de provisionamento e escalonamento de máquinas virtuais e contêineres numa nuvem OpenStack segura, que considera aspectos essenciais para o SGX, como a quantidade de memória segura sendo utilizada, o *Enclave Page Cache* (EPC). Por fim, validamos esta nuvem com uma aplicação que exige processamento de dados sensíveis.

Este estudo é parte do projeto *SecureCloud: Secure Big Data Processing in Untrusted Clouds*, que tem por objetivo principal criar novas aplicações de *big-data* capazes de usar dados sensíveis em plataformas de computação em nuvem, sem comprometer a privacidade e segurança de tais dados. Essa é uma parceria Brasil-Europa, que conta com contribuições de universidades como a Universidade Federal de Campina Grande (UFCG), Universidade Federal de Itajubá (UNIFEI), Technische Universität Dresden (TUD), Imperial College (IMP), University of Neuchâtel (UNINE), entre outras.

1.2 Objetivos

1.2.1 Objetivo geral

Este trabalho tem por objetivo propor estratégias para o suporte a ambientes de execução confiável em sistemas de computação na nuvem, integrando a tecnologia de segurança em hardware Intel SGX à plataforma OpenStack. Os recursos de nuvem considerados serão máquinas virtuais e contêineres, junto à orquestradores de contêineres como Kubernetes. Serão considerados aspectos de provisionamento, escalonamento e contabilidade de recursos, de forma a fazer com que a nuvem entenda de aspectos fundamentais relativos à alocação de recursos SGX.

Além disso, após integração com os componentes de segurança, o desempenho de recursos e componentes da nuvem será avaliado, de forma a determinar o custo de implantação de tais estraté-

gias de segurança. Por fim, validamos a nuvem com suporte a ambientes de execução confiável ao implantar uma aplicação que requer processamento de dados sensíveis, no nosso contexto, medições de energia obtidas a partir de medidores inteligentes.

1.2.2 Objetivos específicos

Objetivos específicos deste trabalho são como segue:

- Integrar componentes de virtualização do SGX ao OpenStack de forma a ser possível prover instâncias seguras usando o serviço Nova;
- Prover instâncias com suporte à SGX no Openstack de forma a ser possível criar clusters de contêineres seguros usando o serviço Magnum;
- Desenvolver um modelo de provisionamento e escalonamento para a nuvem OpenStack que considere aspectos relativos à alocação de recursos SGX;
- Implementação e avaliação de uma aplicação robusta que lide com dados sensíveis para validação da nuvem segura;

1.3 Estrutura do documento

Este trabalho é organizado da seguinte forma. O Capítulo 2 detalha conceitos relacionados à solução proposta nesta pesquisa e, portanto, importantes para o entendimento do restante do documento. Nele é apresentada a tecnologia de segurança em hardware SGX, usada para implementação do ambiente de execução confiável, além do OpenStack, nuvem escolhida para base da integração com SGX. Em seguida, o Capítulo 3 apresenta trabalhos relacionados a essa pesquisa e como a nossa solução se compara ao que já existe na literatura.

No Capítulo 4, é definida uma arquitetura para integração OpenStack-SGX. Nele apresentamos as estratégias de provisionamento, escalonamento e contabilidade de recursos, que levam em consideração aspectos de alocação do SGX. Ainda é avaliado o desempenho dos componentes e recursos da nuvem após implementação das modificações. Similarmente, o Capítulo 5 mostra a integração OpenStack-SGX para contêineres do tipo LXD, além da junção à orquestradores de contêineres usando o serviço Magnum.

Depois de apresentada a nuvem segura usando SGX, o Capítulo 6 descreve a validação dessa nuvem a partir da implantação de uma aplicação que exige processamento de dados sensíveis, no

nosso caso, medições de energia obtidas a partir de um medidor inteligente. Por fim, o Capítulo 7 discute as contribuições e conclusões finais deste trabalho.

Capítulo 2

Fundamentação teórica

2.1 Computação em nuvem

2.1.1 OpenStack

No contexto de computação em nuvem, a plataforma open-source OpenStack [5] tem crescido e se destacado nos últimos anos. Frequentemente oferecida como Infraestrutura como um Serviço (IaaS), ela é utilizada para implantação de nuvens públicas, privadas ou híbridas e mantida por uma comunidade ativa e financiada por mais de 200 organizações respeitadas no mercado, como Red Hat, Cisco, Rackspace, entre outras.

O OpenStack possui componentes que se inter-relacionam e controlam conjuntos de recursos físicos e virtuais, usados para processamento e armazenamento de dados, comunicando-se via redes internas e externas pré-definidas. Usuários podem gerenciar uma nuvem OpenStack através de um painel de controle disponível como um serviço web, com ferramentas de linha de comando, ou via uma API REST. Um dos grandes atrativos dessa plataforma de nuvem é a sua capacidade de criar uma nuvem IaaS usando hardware simples, e em pequena quantidade, ao mesmo tempo que possibilita o uso de serviços de forma sofisticada.

Dentre esses serviços disponíveis no OpenStack, existem os considerados essenciais, bem como serviços para casos de uso específicos, mas que não necessariamente atingem a maioria dos usuários. Aqui vamos citar os indispensáveis para o funcionamento regular de uma nuvem OpenStack:

Nova. Serviço primário de computação e provisionamento de recursos do OpenStack. É utilizado para implantar e gerenciar de pequenas à grandes quantidades de máquinas virtuais, físicas e até mesmo contêineres, sob demanda, oferecendo mecanismos de escalabilidade vertical e horizontal.

Nova interage com o Keystone para autenticação; Glance para imagens a serem implantadas nas máquinas; e Horizon para interface com o usuário.

Swift. Sistema de armazenamento para objetos e arquivos, de alta disponibilidade, distribuído e consistente. Desenvolvedores podem usar esse serviço para armazenar grandes quantidades de dados de maneira eficiente, segura e barata. Swift também é capaz de lidar com concorrência no acesso aos arquivos em todo o data set. Ideal para armazenamento de dados não estruturados e que podem crescer sem limitação específica.

Cinder. Outro sistema de armazenamento para objetos e arquivos, que usa o conceito de Block Storage em sua implementação. Isso quer dizer que, ao contrário da ideia tradicional de referenciar arquivos baseado na sua localização em disco, desenvolvedores podem referenciar um identificador único de um arquivo e deixar o OpenStack decidir onde essa informação é armazenada. Dessa forma, ao invés do desenvolvedor, o sistema é que se preocupa com a melhor forma de garantir que existe um backup seguro dos dados, em caso de falha de uma máquina ou da conexão de rede.

Glance. Provê serviços relacionados ao registro e recuperação de imagens de disco para máquinas virtuais e físicas no OpenStack. Imagens de VMs disponíveis através do Glance podem ser armazenadas em diferentes lugares, desde sistemas de arquivo simples até sistemas de armazenamento de objetos como o próprio serviço Swift.

Neutron. Serviço que gerencia conexões de rede entre os componentes do OpenStack, bem como com o mundo externo. Neutron garante que os componentes do sistema serão capazes de se comunicar com sucesso, e trocar dados e informações entre si. Mecanismos de segurança nessas conexões podem ser utilizados, como o uso de TLS.

Keystone. Provê serviços de autenticação para os usuários do sistema. É responsável por garantir que os projetos definidos na nuvem sejam acessados apenas por usuários com as permissões necessárias, bem como os recursos existentes nos dados projetos. Suporta LDAP, OAuth, OpenID Connect, SAML e SQL.

Ceilometer. Tem por objetivo coletar, normalizar e transformar dados da nuvem, de forma eficiente. De posse desses dados, é possivel ter uma visão do uso de recursos de cada usuário e projeto, ajudando provedores de nuvem a planejar melhor a implantação de aplicações baseado no uso de tais recursos, além de definir modelos de negócio para as respectivas nuvens.

Heat. Orquestra recursos para aplicações de nuvem, baseado em templates. Tais templates são escritos em forma de texto, tratados como código, e definem qual infraestrutura é necessária para que a aplicação esteja disponível. O Heat ainda provê serviços de escalonamento automatizado, integrado com o serviço Ceilometer, de forma a aumentar ou diminuir um cluster de máquinas de um usuário baseado nas necessidades da aplicação definidas no template.

Horizon. Painel de controle disponibilizado como um serviço web para os usuários do OpenStack. Essa interface gráfica é geralmente o primeiro contato com o sistema por parte dos usuários finais da nuvem. Dados de uso de recursos também são melhor visualizados através dessa interface, sendo de grande ajuda para administradores do sistema.

2.1.2 Tipos de recurso em nuvem

Provedores de nuvem podem oferecer seu poder computacional de variadas formas, entre elas, através do provisionamento de recursos como máquinas virtuais, contêineres e, mais recentemente, até máquinas físicas. Usuários devem escolher qual recurso utilizar baseado nas necessidades de suas aplicações e serviços a serem implantados. No contexto desse trabalho, abordamos especialmente máquinas virtuais e contêineres como recursos de nuvem, e portanto, discutimos a seguir os benefícios e pontos negativos de cada um deles.

Quando falamos em virtualização [41], pensamos em um método de alocação de recursos que abstrai os detalhes relacionados à máquina física em si, e provê esses mesmos recursos, de forma virtualizada, a partir do que a máquina hospedeira é capaz de oferecer. Tal conceito é fundamental para ambientes de nuvem, que provisionam recursos de acordo com a necessidade de seus usuários, e dessa forma, podem utilizar seus servidores ao máximo de sua capacidade. Considerando isso, surge o conceito de máquinas virtuais, que emulam as funcionalidades de uma máquina física, e funcionam como um sistema independente em cima de um dado servidor. Desse ponto de vista, virtualização é, fundamentalmente, uma forma de alocar vários usuários diferentes, com ambientes independentes, em uma mesma infraestrutura física.

Máquinas virtuais possuem um sistema operacional independente. Dessa forma, uma mesma máquina física é capaz de rodar um sistema operacional para cada VM instanciada nela. Essa abordagem, portanto, tem por objetivo compartilhar os recursos de uma máquina física entre vários ambientes isolados, que por sua vez, se enxergam como máquinas independente, com processador, sistema operacional, memória, disco, etc, próprios, alocados através da abstração provida pelo conceito de virtualização, como podemos ver na Figura 2.1. É importante destacar que o sistema operacional

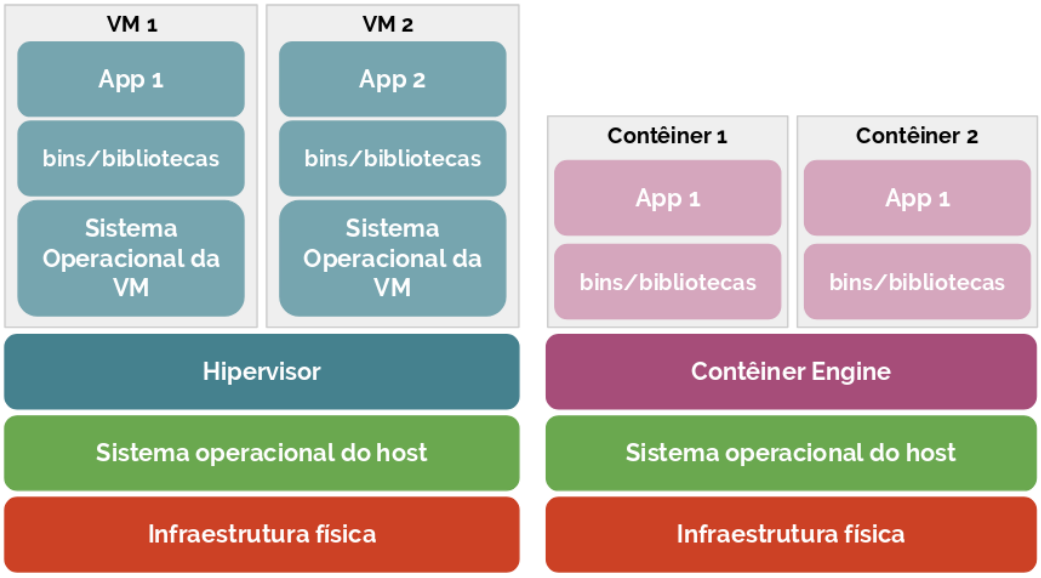


Figura 2.1: Infraestrutura para máquinas virtuais e contêineres.

de uma VM interage com o hardware virtual, provido por hipervisores, e não diretamente com o hardware real.

Outro método de virtualização é o de conteinerização. Diferentemente de máquinas virtuais, contêineres não possuem sistema operacional inteiramente independente, ou seja, um contêiner se enxerga como uma máquina inteira, mas compartilha um mesmo kernel com todos os contêineres instanciados num mesmo hospedeiro, como mostra a Figura 2.1. Isso significa que, desse ponto de vista, eles são mais leves que máquinas virtuais, já que compartilham o mesmo kernel, iniciam mais rápido, e usam uma fração menor de memória comparada com o que necessário para iniciar um sistema operacional inteiro.

Contêineres oferecem um mecanismo lógico de empacotamento que favorece aplicações divididas em microsserviços, unidades menores de uma aplicação completa, que vão executar inteiramente dentro de *clusters* de contêineres, com todos os seus requisitos e dependências. Esse modelo permite que as aplicações sejam abstraídas do ambiente no qual estão rodando, garantindo que desenvolvedores se preocupem apenas com a lógica e dependências de seus sistemas, enquanto operadores focam na gerência e implantação do recurso em si. Considerando isso, usuários podem criar imagens de ambientes propriamente configurados para suas aplicações, e tais imagens podem, então, ser implantadas em um contêiner sem maior esforço, além de replicadas quantas vezes preciso.

O sucesso comercial de contêineres teve início com o surgimento do Docker, um método de conteinerização popular, suportado por grandes plataformas como a *Google Cloud Plataform* e or-

questradores de contêineres como *Kubernetes*. Docker faz uso de estruturas de isolamento do Linux, como *kernel namespaces* para prover isolamento de redes, sistema de arquivos, e recursos similares, e *cgroups* para limitar o uso de recursos como memória, CPU e largura de banda de rede. Docker ainda provê acesso a um repositório a partir do qual imagens podem ser recuperadas e armazenadas.

Dessa forma, vemos que ambos os tipos de recurso, amplamente comparados não só na literatura como também no mercado, sejam eles máquinas virtuais ou contêineres, possuem seus benefícios e limitações. É importante, então, entender cada caso de uso e avaliar qual a melhor opção considerando suas características de isolamento, desempenho, segurança, uso de recursos, entre outros.

2.2 Orquestradores de contêineres

Por definição, orquestradores de contêineres são sistemas que gerenciam *clusters* de máquinas, que por sua vez, servem aplicações implantadas em um conjunto de contêineres espalhados pelo *cluster* [46]. Uma característica chave quando pensamos sobre serviços de orquestração é a automação que eles proveem para tarefas como inicialização, provisionamento e implantação de recursos e aplicações, além de monitoramento do comportamento e estado dos *clusters*, estratégias de escalonamento e tolerância a falhas. Em linhas gerais, a ferramenta de orquestração vai selecionar um hospedeiro apropriado para o contêiner sendo inicializado baseado em regras específicas definidas pelo usuário, e assim, operará-los através das funcionalidades citadas anteriormente.

O uso de contêineres para orquestrar aplicações está no seu auge atualmente, inclusive quando falamos no seu uso em produção. Duas pesquisas recentes produzidas pelo OpenStack [2] e Kubernetes [17], um dos grandes orquestradores de contêineres no mercado, confirmam essa informação. A pesquisa do OpenStack afirma que 50% dos seus usuários que precisam de contêineres para orquestração usam Kubernetes junto ao OpenStack. A pesquisa do Kubernetes, por sua vez, confirma que 75% dos usuários participantes utilizam o orquestrador para sistemas em produção. Por exemplo, OpenStack e Kubernetes tem sido usados em produção no processamento de grandes quantidades de dados pelo *European Organization for Nuclear Research* (CERN), atingindo cerca de 25 petabytes de dados analisados [32].

Dessa forma, vemos a força de serviços de orquestração nos dias atuais, especialmente Kubernetes, que hoje possui um dos repositórios mais ativos no GitHub [24]. Entre as características mais relevantes desse serviço está sua habilidade de rodar aplicações em ambientes de nuvem, dada sua integração com diversos provedores. Kubernetes ainda permite que os usuários possam customizar a implantação de seus serviços baseado nos recursos disponíveis pelo provedor da nuvem, como SSD,

rede, etc. Muitas outras funcionalidades são disponibilizadas como automação de reinicialização, escalonamento e replicação, incluindo monitoramento, balanceamento de carga e tolerância a falhas para nuvens distribuídas federadas.

Outro exemplo de serviço de orquestração conhecido no mercado é o *Docker Swarm*[1]. Ele usa sua capacidade de gerenciamento de *clusters* de forma a lidar com um conjunto de contêineres Docker como um só serviço, através de sua plataforma de conteinerização, a *Docker Engine*. Uma das grandes vantagens desse orquestrador é sua compatibilidade com o Docker[2], um serviço de provisionamento de contêineres já bastante utilizado. Assim, qualquer ferramenta que já faz uso de Docker pode usar o Docker Swarm para escalar, de maneira transparente, em hospedeiros variados.

Finalmente, quando falando sobre ambientes seguros de execução em nuvem, foco desse trabalho, o compartilhamento de kernel base para sistemas contêinerizados, pode ameaçar a confidencialidade e integridade dos dados sendo processados por tais contêineres, o que pode comprometer o sistema. Essa preocupação é confirmada pela pesquisa produzida pelo Kubernetes, previamente citado, que tem a segurança apontada por 43% dos usuários como um fator importante na escolha do ambiente de implantação de suas aplicações.

2.3 Intel SGX

Garantias de privacidade e integridade de dados é algo altamente desejado por usuários que buscam proteger informação sensível de ataques maliciosos. Algumas abordagens que tentam prover tal nível de segurança, especialmente em ambientes de nuvem, não possuem a habilidade de proteger as aplicações contra software com acesso privilegiado, como hipervisores e o próprio sistema operacional [45; 43]. Nesse contexto, surge a Intel ***S**oftware **G**uard e**X**tensions* (SGX) [19; 20], uma tecnologia de hardware que garante privacidade e integridade dos dados, protegendo o código de aplicações até contra componentes do sistema com acesso privilegiado, mas não-confiáveis.

Para isso, Intel SGX provê um conjunto de instruções que permitem acesso à memória, criando áreas protegidas chamas enclaves [31]. O *Enclave Page Cache* (EPC) é uma memória protegida usada para armazenar páginas de enclaves e estruturas SGX. O EPC é dividido em pedaços de 4KB chamados páginas de EPC. Uma página de EPC válida contém uma página de enclave ou um estrutura SGX. Os enclaves, portanto, são as áreas protegidas de memória onde as aplicações e seus dados residem, gerenciados por políticas de controle de acesso, as quais previnem ataques contra seu con-

[1]Docker Swarm. https://www.docker.com/products/docker-swarm/ [Último acesso: 22 de janeiro, 2018]

[2]Docker. https://www.docker.com/ [Online; Último acesso: 22 de janeiro, 2018]

teúdo. Código fora de um enclave não pode acessar a memória do mesmo. Entretanto, código dentro do enclave pode acessar a memória não confiável fora do EPC, sendo responsável por verificar a integridade desses dados.

Intel SGX ainda oferece métodos de atestação local e remota [6], que pode ser realizada por terceiros para assegurar que uma parte conhecida de software está rodando de maneira segura dentro de um enclave, em uma plataforma SGX. Atestação remota, requer o uso de criptografia assimétrica, já que a verificação vem de fora do sistema, além de um componente especial, o *Quoting Enclave* (QE). Esse enclave é responsável por criar a *Intel Enhanced Privacy ID* (EPID), chave usada para assinar atestações a serem certificadas por uma infraestrutura EPID de *backend*. Apenas o QE conhece essa chave EPID, a qual é baseada na versão de firmware do processador.

Possíveis usos do SGX [10] incluem tecnologias de autenticação, transações financeiras online, registro de atividades e informações pessoais de usuários, vídeo conferência, e muitos outros. Hoekstra et al. define três exemplos de soluções seguras que foram desenvolvidas com base nos benefícios do Intel SGX [27]:

One-time Password (OTP): OTP é uma tecnologia frequentemente usada em autenticação de usuários de dois fatores. Como o nome sugere, a senha é válida apenas para uma autenticação e é comumente usada para autorizar transações bancárias online, por exemplo. Seus componentes incluem um cliente e servidor OTP. O software do cliente possui algoritmos dentro de enclaves que interagem diretamente com segredos da aplicação OTP. O servidor pode então utilizar atestação remota para verificar o enclave rodando no cliente e estabelecer um canal de comunicação seguro, que será usado para enviar uma chave compartilhada para ser usada como senha pela aplicação.

Video Conferência Segura: Com a disponibilidade cada vez maior de banda de alta velocidade e hardware de baixo custo para captura de imagens, aplicações para conferência via vídeo tem se tornado mais populares. Isso abre uma brecha para atacantes maliciosos se apoderarem de tais dados, e distribuírem ilegalmente imagens em vídeo que podem conter informações sensíveis a respeito de usuários e companhias.

Atualmente, soluções para prover segurança à conferências via vídeo oferecem a proteção desses dados através do uso de métodos de criptografia. Entretanto, com a migração de ameaças para a plataforma de computação, além da rede em si, tais estratégias de segurança não são mais suficientes, já que o *stream* é processado nela. Dessa forma, o uso de SGX nas plataformas permite que a aplicação proteja o *stream* contra ataques externos.

Capítulo 3

Trabalhos relacionados

Nesta seção são apresentadas pesquisas no contexto de ambientes de execução confiável em nuvem e quais as soluções oferecidas considerando proteção e privacidade de dados, além da integridade de recursos virtualizados. Abordagens que consideram outras tecnologias em hardware também são incluídas, comparando os benefícios e pontos negativos ao virtualizar tais tecnologias e a Intel SGX. Ainda é realizada um incursão em trabalhos que propõem a criação de uma cadeia de confiança que inclui o próprio provedor da nuvem. Por fim, relacionamos pesquisas que apresentam outros tipos de soluções a nível de segurança para uma nuvem OpenStack, além de analisar possíveis ataques à recursos virtualizados com acesso à SGX.

Diversos trabalhos no contexto de segurança em ambientes de nuvem são voltados para o oferecimento de proteção e privacidade de dados e código de seus usuários [42; 15; 14]. Alguns apresentam soluções para proteção da integridade de máquinas virtuais contra outras VMs instanciadas no mesmo hospedeiro, ou contra o próprio provedor de nuvem que tem total controle sobre seu hipervisor e componentes de hardware [47; 35]. Em linhas gerais, algumas abordagens como a de Butt et al. [16] propõe uma arquitetura que define um novo modelo de hipervisor, oferecendo aos usuários da nuvem mais controle sobre a gerência de suas máquinas virtuais. Schiffman et al. [39], por outro lado, apresenta o sistema *Cloud Verifier*, que tem seu foco em garantias de integridade de máquinas virtuais, obtidas através do monitoramento das mesmas com base em critérios definidos pelos próprios usuários.

Outras soluções, como a de Brown e Chase em [15], propõe que a plataforma de nuvem em si seja confiável, de forma a oferecer suporte para serviços confiáveis em nuvem. Nesse contexto, um provedor de nuvem confiável pode agir como a base de uma cadeia de confiança que atesta serviços hospedados em nuvem para seus clientes, em outras palavras, uma plataforma de nuvem confiável.

O trabalho apresenta um protótipo dessa abordagem, no qual o provedor sela instâncias servindo à aplicações e atestam seu código fonte Python para usuários externos. Uma vez criadas e atestadas, instâncias desses serviços executam com uma identidade independente protegida de manipulação. A nossa solução independe de um provedor confiável, buscando oferecer recursos de nuvem que sejam considerados seus próprios ambientes de execução com garantias de segurança, a qual é provida através do uso da tecnologia SGX. Dessa forma, usuários de uma nuvem não precisam confiar em seus provedores, se beneficiando dos aspectos de segurança advindos dos recursos oferecidos pelo SGX, como a execução de código e processamento de dados dentro de enclaves.

Nesse contexto, ambientes de execução confiáveis podem ser implementados a partir do uso de tecnologias como a provida pela Intel SGX, proposta neste trabalho, assim como por outros tipos de recurso, a exemplo do *Trusted Platform Module* (TPM). Desenvolvido pelo *Trusted Computing Group* (TCG)[1], essa tecnologia provê uma solução de segurança baseada em hardware, tipicamente um chip embutido na placa mãe de uma plataforma, e controlado via software. Tal mecanismo pode ser estendido para ambientes de computação em nuvem através da integração dessa tecnologia à arquitetura da referida nuvem. Funcionalidades como autenticação e comunicação segura, controle de acesso e proteção de dados, podem ser oferecidas pelo TPM.

Outras abordagens relacionadas, como a de Berger et al. [12] e Hosseinzadeh et al. [28], apresentam implementações de software para virtualização do TPM, buscando habilitá-lo para as máquinas virtuais gerenciadas por um hipervisor. Vários benefícios surgem disso, como o oferecimento de armazenamento seguro para as chaves e segredos de uma plataforma e atestação remota para os hipervisores. Berger usa uma estratégia baseada em software, com o TPM virtual (vTPM) alocado em uma VM dedicada, que disponibiliza funções de criptografia e armazenamento seguro providos pelo TPM para as aplicações e sistemas operacionais executando nas máquinas virtuais provisionadas. Nessa mesma linha, Hosseinzadeh aborda a virtualização do TPM em outro recurso de nuvem bastante popular, os contêineres. Virtualização baseada em contêineres tem a vantagem de ser mais rápida e consumir menos memória que em VMs, ao passo que apresenta limitações como o compartilhamento de kernel com os demais contêineres em um mesmo hospedeiro. Aqui, o vTPM é alocado em um contêiner diferente, que tem acesso ao TPM do hardware e expõe a interface do vTPM para os demais contêineres. Para ambos os trabalhos, um ponto negativo da abordagem em software é que ela não é capaz de oferecer todos os benefícios que a baseada em hardware consegue. Na nossa solução, o recurso virtualizado em VMs e contêineres é o SGX. Isso é feito através de um kernel modificado, o *kvm-sgx*, que disponibiliza espaços seguros de memória baseado em uma abordagem de hardware.

[1] www.trustedcomputinggroup.org; [Online; Último acesso: 04 de janeiro, 2018]

Uma vantagem é que, diferentemente de TPM, SGX protege o sistema até de software privilegiado como sistemas operacionais e hipervisores.

Quando pensamos em um ambiente de nuvem como o OpenStack, utilizado em nossa solução, dado o grande número de nós de computação que podem estar associados à ele, e o fato de estarem hospedando VMs de usuários com chances de serem maliciosos, é possível que alguns desses nós sejam comprometidos. Tais máquinas virtuais podem ser controladas por atacantes e, por consequência, se tornarem capazes de explorar vulnerabilidades em hipervisores. Tais vulnerabilidades têm sido uma preocupação do grupo de segurança do OpenStack, afetando inclusive suas decisões de design. Sze et al. [44] avalia o impacto de nós comprometidos em uma nuvem OpenStack e mostra que um ataque realizado com sucesso em um único nó de computação podem comprometer toda a infraestrutura de nuvem. Ainda é proposto um novo sistema chamado SOS, Secure OpenStack, que limita o nível de confiança em nós de computação, antes considerados completamente confiáveis pelo sistema. SOS consiste em um *framework* capaz de impor um grande leque de politicas de segurança, em outras palavras, aplicando controle de acesso obrigatório para reger interações entre componentes diferentes. Ao contrário da nossa solução, SOS não requer modificações no OpenStack, mas não oferece recursos seguros a nível de hardware como proposto neste trabalho. Na nossa abordagem, independente do nível de confiança atribuído a um nó de computação, a segurança que ele oferece aos dados e código de VMs instanciadas neles, é advinda do SGX e do tamanho de memória protegida que ele é capaz de oferecer para a criação de enclaves, os quais irão executar aplicações seguras.

Por fim, Schwarz et al. [40] demonstra um ataque do tipo *side channel*, baseado em software, de um enclave SGX malicioso contra enclaves numa mesma máquina. Tal ataque é o primeiro *malware* a executar em um hardware SGX real, que faz o enclave malicioso abusar das funcionalidades de proteção do SGX para beneficiar a si mesmo. O ataque é capaz de extrair 96% do conteúdo de uma chave privada, e é demonstrado em um ambiente nativo, além de um que considera vários contêineres Docker. Aplicações executando dentro de contêineres não devem notar nenhuma diferença para cenários nativos, e dessa forma, o *malware* também funciona dentro de contêires. Schwarz relata que foi capaz de reproduzir o ataque até entre dois contêineres no mesmo hospedeiro, e que o ataque foi possível graças ao compartilhamento do mesmo *driver* SGX na máquina hospedeira, um só para todos os contêineres. Na nossa abordagem, uma vantagem clara é o isolamento proporcionado por máquinas virtuais, que, por consequência, permitem que cada VM provisionada seja capaz de ter o seu próprio driver SGX, tendo mais chances contra o ataque explorado em [40]. Schwarz não avaliou cenários considerando máquinas virtuais.

Capítulo 4

Máquinas virtuais e SGX em ambientes de nuvem

4.1 Virtualização do SGX com KVM

De forma a fazer com que usuários OpenStack possam implantar suas aplicações SGX em nuvem, é necessário que o OpenStack seja capaz de provisionar recursos com suporte à SGX. Do ponto de vista da nuvem, seja a aplicação implantada numa máquina virtual ou em um *cluster* de contêineres, o recurso base é uma máquina virtual. Assim, é preciso que VMs tenham um dispositivo SGX habilitado para seus usuários.

Criar máquinas virtuais com capacidade SGX ainda é considerado em processo de desenvolvimento pela Intel. Entretanto, os esforços para dar suporte à isso são crescentes e já atingiram versões de teste disponíveis em [3; 4]. Esse *kernel* modificado, chamado *kvm-sgx* [3], funciona como um KVM que disponibiliza o EPC da máquina hospedeira para as aplicações rodando nas máquinas virtuais. É importante destacar que o *kvm-sgx* precisa ser instalado em um hospedeiro com SGX disponível e habilitado.

Considerando isso, para expor o SGX da máquina física para VMs hospedadas nela, é necessário pré-definir uma quantidade específica de EPC a ser alocada às máquinas virtuais no seu processo de criação. Apenas o particionamento estático de EPC é suportado atualmente, o que significa que as páginas de EPC são estaticamente alocadas para as VMs e apenas liberadas quando a dada instância é destruída. Para alocar EPC, algumas mudanças no QEMU também são necessárias. Para isso, *qemu-sgx* [4] provê novos parâmetros a serem passados na definição do XML referente à máquina virtual a ser criada, contendo informação à respeito de quanto EPC deve ser alocado. Esses parâmetros

são `-sgx epc=$quantidade_de_epc` e `-cpu host`, os quais, combinados, comunicam ao hipervisor como criar uma VM com funcionalidades SGX.

Dessa forma, nós integramos *kvm-sgx* e *qemu-sgx* com uma nuvem OpenStack fazendo com que os nós de computação sejam capazes de expor seu SGX para as VMs hospedadas nele. Assim, nós de computação inseridos na nuvem são servidores SGX, e tem *kvm-sgx* instalado como hipervisor. Além disso, para dar ao usuário a possibilidade de definir quanto de EPC deve ser alocado, e fazer com que as instâncias SGX sejam propriamente escalonadas baseada na quantidade de EPC disponível no nó de computação, modificações foram feitas no OpenStack. Essas modificações são discutidas com detalhes na próxima seção.

4.2 KVM-SGX na nuvem

4.2.1 Provisionando uma instância no OpenStack

O serviço Nova é responsável por provisionar recursos computacionais numa nuvem OpenStack, oferecendo acesso sob demanda à grandes conjuntos de máquinas virtuais, físicas e contêineres. Para isso, tecnologias de virtualização são utilizadas, como os hipervisores KVM, QEMU, VMware, Xen, Hyper-V, e ainda Linux LXC e LXD para contêineres.

Para acesso aos recursos oferecidos, existe um sistema organizacional que considera projetos aos quais um usuário pode pertencer e papéis que ele pode exercer. Um projeto consiste em um conjunto de recursos que podem ser gerenciados por um conjunto de usuários, como volumes, instâncias, imagens e chaves, por exemplo. Cada projeto, por sua vez, terá acesso a uma parte dos recursos disponíveis na nuvem, limitado por cotas. Já os papéis que um usuário pode exercer, definem as ações que ele pode executar enquanto membro de um projeto. Nesse caso, regras são criadas para especificar essas limitações, quando existentes.

De forma a prover o serviço, o Nova conta com alguns componentes que desempenham variados papéis, desde a armazenamento de dados, alocação de instâncias, gerenciamento de hosts, entre outros. Abaixo temos os principais componentes e suas funções:

Controlador. O nó controlador é responsável pelo gerencimento de componentes importantes da nuvem OpenStack, que auxiliam no provisionamento de instâncias. Além do próprio Nova, outros serviços rodam nesse nó, como o Keystone para autenticação, Glance para gerenciamento de imagens, parte gerencial do Neutron para definição de redes, entre outros.

nova boot <options> <instance-name>	
Paramêtro	**Descrição**
–flavor <flavor>	Define configurações de hardware para uma instância.
–image <image>	Nome ou identificador da imagem base no Glance a ser implantada em uma instância.
–nic net-id='net-id' <net-id>	Identificador da rede de provisionamento a ser usada.

Tabela 4.1: Parâmetros necessários para criação de uma instância no serviço Nova.

Compute. Esse nó roda a parte referente aos hipervisores que operam as intâncias construídas. Um compute pode ser host de várias dessas instâncias, dependendo do conjunto de recursos que ele tem disponível, como CPU, RAM, disco entre outras especificidades. Além disso, mais de um compute pode ser associado a um mesmo controlador, o que permite que a nuvem seja capaz de prover um conjunto maior de instâncias, por exemplo.

Considerando esses componentes e como eles estão organizados, para provisionar uma máquina é preciso, por fim, compreender os objetos necessários para sua criação. A Tabela 4.1 mostra os parâmetros essenciais para criação de uma dada instância.

No contexto desse trabalho, é importante entender como funciona um *flavor*. Como definido na Tabela 4.1, esse objeto pode guardar as mais diversas especificações a respeito dos recursos de hardware sendo requisitados por uma instância. Tais recursos podem ser memória, CPU, capacidade de armazenamento, entre outros, como mostra a Tabela 4.2 abaixo.

Caso seja necessário definir características bastante específicas para um tipo de instância ou hardware, como informações que influenciam o escalonamento, ou mapeamento entre CPUs virtuais e físicas, é usado o parâmetro `-property <extra-specs>`. Por exemplo, se quisermos que uma instância seja escalonada só em nós de computação que tenham hardware GPU, é possivel criar um flavor que defina isso no seu <*extra-specs*>. Note que, essa noção é importante para entender o processo de integração do SGX à uma nuvem OpenStack, já que usamos essa propriedade para passar a quantidade de EPC a ser alocada para as instâncias requisitadas.

Quando uma requisição para criar uma instância é recebida, ela é direcionada ao componente do OpenStack responsável por filtrar e escolher qual nó de computação associado àquela nuvem possui recursos suficientes para receber a dada instância sendo requisitada. Durante o escalonamento, o *Filter Scheduler* itera sobre todos os nós de computação encontrados, avaliando cada um com base

Flavor	
Paramêtro	**Descrição**
–ram <ram>	Quantidade de memória RAM a ser usada (em megabytes).
–disk <disk>	Quantidade de disco a ser usada (em gigabytes) para a partição raiz.
–vcpus <vcpus>	Número de cpus virtuais a ser usado.
–is-public	Define se o flavor é disponível para todos os usuários ou um projeto específico. Valores padrão é *True*.
–property <extra-specs>	Pares chave e valor que definem em quais nós de computação uma instância pode rodar.

Tabela 4.2: Parâmetros necessários para criação de um *flavor*.

nos filtros selecionados pelo administrador na nuvem. Tais filtros definem métricas e condições que um nó de computação deve atender de forma a ser considerado apto para receber uma nova instância.

Todos os nós que passam pelos filtros selecionados são então ordenados de acordo com pesos também definidos pelo administrador, e o escalonador decide pelo melhor avaliado após usar o critério definido pelos pesos. Se o escalonador não conseguir encontrar nós aptos a receber uma instância, significa que não existem nós de computação com recursos suficientes para atender a dada requisição, e o processo falha.

Considerando filtros já existentes, existe uma variedade de estratégias de filtragem que podem ser selecionadas para escalonamento, algumas estão listadas abaixo:

AvailabilityZoneFilter. Filtra nós de computação a partir de zonas de disponibilidade. Nós compatíveis com a zona especificada pelo flavor passam no filtro. Assim como para provedores de nuvens públicas, diferentes zonas de disponibilidade têm diferentes domínios de falha, como diferentes linhas de distribuição de energia ou conexões de Internet. Usuários podem influenciar como esse parâmetro é configurado.

AggregateInstanceExtraSpecsFilter. Filtra nós baseado nos metadados de um agregado de nós de computação. Se tais metadados satisfazem alguma das configurações definidas no *extra_specs* de um *flavor* associado a uma instância, os nós pertencentes àquele agregado passam no filtro. Um *flavor*

definido pelo administrador da nuvem pode ou não ter metadados que restrinjam como o escalonamento é feito baseado em agregados de nós.

RamFilter. Filtragem é feita com base na quantidade de memória RAM disponível nos nós de computação. Se existe uma quantidade suficiente de RAM para atender ao requisitado pela instância, o nó passa no filtro.

DiskFilter. De maneira similar ao RamFilter, a filtragem é baseada na quantidade de disco disponível nos nós de computação. Apenas nós com espaço em disco suficiente para atender ao requisitado pela instância passam no filtro.

Existe uma grande variedade de filtros disponíveis no OpenStack que suportam várias estratégias de filtragem e definição de pesos. Através dessa variedade de filtros, o escalonador se torna flexível. Ainda assim, se essa flexibilidade não é suficiente para um caso de uso específico, novos filtros podem ser implementados usando algoritmos de filtragem customizados, o que é o caso da nossa solução, a ser discutida nas seções seguintes desse documento.

4.2.2 Arquitetura proposta

Considerando os componentes envolvidos no processo de provisionamento de uma instância Nova, detalhados na Seção 4.2.1, propomos modificações no fluxo arquitetural existente de forma a integrar Intel SGX e OpenStack, possibilitando a criação de recursos através da virtualização provida pelo *kvm-sgx*, apresentado na Seção 4.1.

Dito isso, queremos tornar o processo o mais transparente possível para os serviços envolvidos, mantendo a base utilizada por outros modelos de provisionamento implementados no Nova. A Figura 4.1 a seguir mostra os componentes principais envolvidos no processo de instanciação: o componente que recebe as requisições do cliente Nova, na figura chamado de API, o Escalonador que decide qual nó irá receber a instância requisitada, o Nó de Computação propriamente dito e um componente chamado Condutor, responsável por conduzir informação e requisições ao banco e outros serviços no processo de provisionamento.

Cada componente está associado a um serviço no Nova, são eles o `nova-api`, `nova-scheduler`, `nova-compute` e `nova-conductor`. Juntos, esses serviços regem o processo de provisionamento de uma instância OpenStack. Propomos então, modificações nos componentes e serviços em verde na Figura 4.1, o Escalonador e o Nó de Computação. Vamos olhar para o processo como um todo considerando uma requisição de instância chegando.

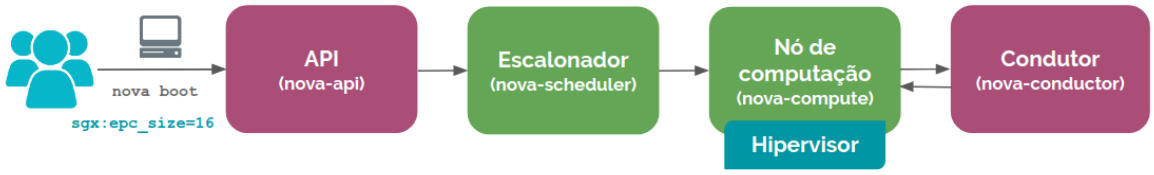


Figura 4.1: Componentes do serviço Nova envolvidos no provisionamento de uma instância.

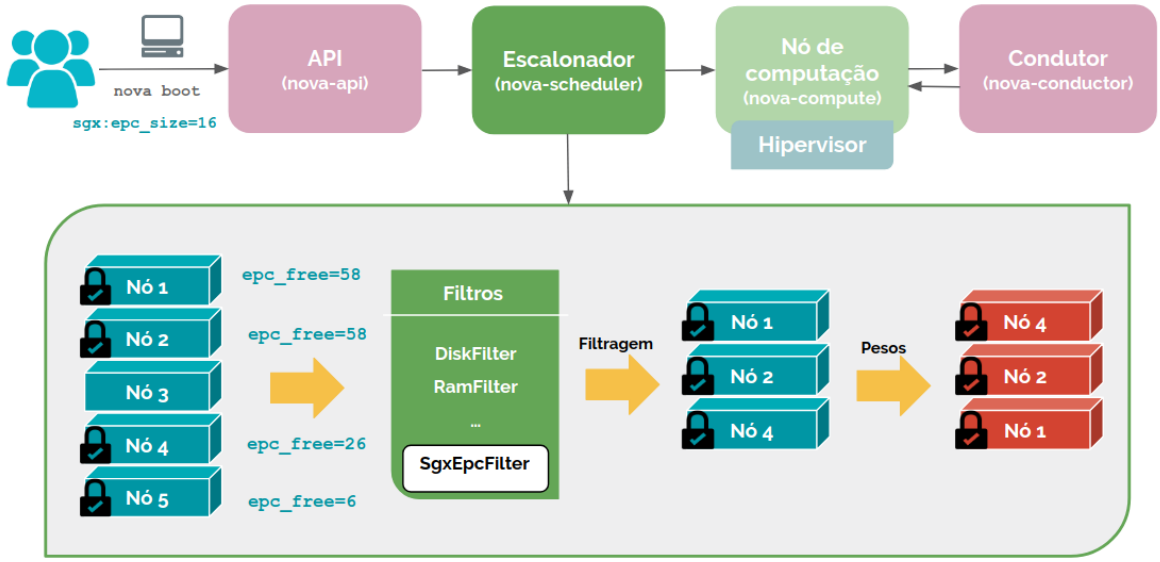


Figura 4.2: Filtragem de nós durante processo de escalonamento.

Inicialmente a requisição vem junto ao *flavor*, que contém as configurações referentes aos recursos que a máquina precisa para ser instanciada. Nesse ponto, é necessário dizer ao flavor quanto de EPC o hipervisor deve alocar na máquina SGX hospedeira. Isso é necessário pois o processo de virtualização oferecido pelo *kvm-sgx* exige que a alocação desse recurso seja definida no momento de criação da máquina, de maneira antecipada, como apresentado na Seção 4.1. Assim, fazemos isso através do campo `extra_specs` do *flavor*, definido na Tabela 4.2.

Em seguida, a Figura 4.2 apresenta o que acontece no componente Escalonador. Quando a requisição chega, o escalonador é responsável por decidir qual nó de computação é capaz de hospedar a instância dados os recursos sendo requisitados. Assim, todos os nós passam por um processo de filtragem, detalhado na Seção 4.2.1. Na figura, consideramos um ambiente com cinco nós, onde os Nós 1, 2, 4 e 5 são máquinas com SGX disponível e habilitado. Para filtragem, o administrador da nuvem pode selecionar uma quantidade de filtros pelos quais os nós devem passar. Aqui entra a segunda modificação proposta. Criamos um filtro especial, o `SgxEpcFilter`, que sabe calcular quanto de

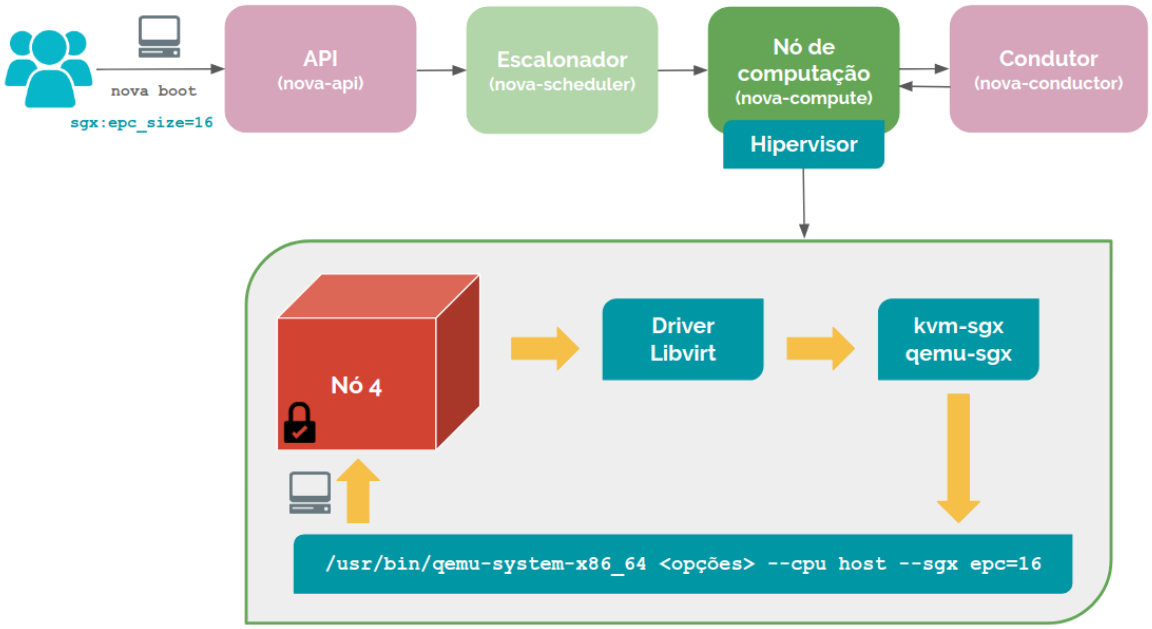


Figura 4.3: Parâmetros usados pelo kvm/qemu-sgx para criação de instância SGX.

recurso SGX ainda está disponível no nó hospedeiro, de forma a decidir quais máquinas ainda podem oferecer EPC.

Na Figura 4.2, os Nós 1 e 2 possuem 58 MB de EPC disponível, o Nó 4 possui 26 MB e o Nó 5 apenas 6 MB. O Nó 3 não é uma máquina SGX. Dessa forma, considerando os 16 MB de EPC sendo requisitados, o `SgxEpcFilter` irá selecionar os nós 1, 2 e 4 como aptos, já que o Nó 5 não possui quantidade suficiente de EPC disponível e o Nó 3 não é uma máquina SGX. Em seguida os nós aptos passam por um processo de ordenação, baseada em pesos definidos pelo administrador, que avalia o que vale mais em relação aos recursos disponíveis na máquina, e seleciona a melhor avaliada como sendo o hospedeiro escolhido para a instância requisitada.

Por fim, a Figura 4.3 mostra o que acontece no Nó de Computação. Esse componente trabalha em conjunto com o hipervisor escolhido pelo administrador para realizar o processo final de criação da máquina virtual. No nosso cenário, o Nó 4 escolhido após processo de filtragem, deve possuir o kernel *kvm-sgx* instalado para assegurar que a máquina hospedeira é capaz de alocar EPC e disponibilizar SGX para suas instâncias. Aqui acontece a terceira modificação proposta. Implementamos no *driver* do Libvirt dentro do OpenStack um trecho de código que adiciona ao XML da máquina a ser criada pelo hipervisor os parâmetros necessários para que a máquina virtual tenha acesso ao SGX do hospedeiro. Tais parâmetros são `-cpu host` e `-sgx epc=16`, no nosso caso. O valor de EPC pode mudar dependendo da quantidade requisitada a partir do *flavor*.

Assim, finalmente, cobrimos todo o processo de provisionamento de uma instância no OpenStack, modificando o fluxo arquitetural no que é necessário para prover integração entre Intel SGX e a dada nuvem. Note que, considerando essa mesma arquitetura, o OpenStack é capaz de prover *clusters* de máquinas virtuais SGX, que podem ser utilizadas pelo seu serviço de orquestração de contêineres, o Magnum. Detalhes da nossa abordagem e implementação são apresentadas nas seções a seguir e no Capítulo 5 sobre contêineres SGX.

4.2.3 Provisionando uma instância segura no OpenStack usando SGX

O processo regular de provisionamento de uma máquina virtual usando o serviço Nova é descrito na Seção 4.2.1. Nós modificamos esse processo ao adicionar entradas no XML usado pelo hipervisor para criar uma instância, na criação de um novo filtro de escalonamento que entende as limitações do tamanho de EPC e apenas seleciona nós de computação com esse recurso disponível, e ainda contabilizamos o uso desse EPC, disponibilizando essa informação para o administrador da nuvem através da linha de comando do Nova. A visão geral dessa abordagem foi vista na Seção 4.2.2 e será detalhada nos tópicos a seguir.

A. Provisionamento

De forma a provisionar uma instância segura que tenha acesso ao SGX do nó de computação no qual ela está hospedada, é necessário fazer com que o hipervisor, no momento da criação da máquina, saiba quanto de EPC deve ser alocado para a instância sendo provisionada. Para isso acontecer, como explicado na Seção 4.1, é preciso que uma versão do kernel especial *kvm-sgx* esteja instalada na máquina física responsável por hospedar VMs. Nesse caso, considerando a nuvem OpenStack, tais máquinas são nós de computação, com *kvm-sgx* e *qemu-sgx* instalados e funcionando como hipervisor da nuvem.

O próximo passo consiste em adicionar entradas ao XML usado para criar as máquinas virtuais, passando a informação referente à quantidade de EPC que deve ser alocada, fazendo com o que hipervisor saiba como criar a instância SGX. No OpenStack, quando usando KVM como hipervisor, isso é gerenciado por um *driver* chamado *Libvirt*, implantado nos nós de computação. Assim, as modificações para adicionar as entradas necessárias ao XML foram feitas na classe que implementa tal *driver* e cria o XML.

A informação referente à quantidade de EPC a ser alocada vem do *flavor* utilizado na requisição da VM. Isso é definido internamente no seu `extra_specs`, como citado anteriormente para casos

específicos similares, no formato `sgx:epc_size=$quantidade_de_epc`. É importante perceber que *flavors* são criados pelo administrador da nuvem, e portanto, a quantidade de EPC definida para cada *flavor* depende das necessidades do administrador. Para este trabalho, foi decido que o tamanho do EPC pode variar de $8\ MB$, $16\ MB$, $32\ MB$ até $64\ MB$.

Finalmente, é necessário modificar o arquivo de configuração do Nova, *nova.conf*, de forma a permitir que o kernel modificado *kvm-sgx* possa coexistir com os demais serviços de provisionamento do OpenStack, já que tal mudança de hipervisor afeta diretamente o processo de criar uma instância. De maneira geral, nós mudamos o tipo de virtualização para KVM, que por padrão é QEMU[1], e habilitamos o console *Virtual Network Computing* (VNC) ao invés do SPICE, que é a opção padrão. Até o momento, apenas VNC é suportado pelo *kvm-sgx*. Mais detalhes na implementação do código utilizado para essas mudanças podem ser encontrados no Apêndice A.

B. Escalonamento

Numa nuvem OpenStack, o processo de criar instâncias é baseado num mecanismo de escalonamento que decide qual nó de computação é o mais apto a receber a máquina virtual sendo requisitada. Como mencionado na Seção 4.2.1, existe uma variedade de filtros disponíveis no Nova que podem ser selecionados pelo administrador da nuvem. Esses filtros são geralmente relacionados ao uso de recursos dos nós de computação e, consequentemente, sua capacidade de receber uma nova instância. Entretanto, podem existir casos de uso específicos que não são cobertos pelos filtros padrão oferecidos pelo Nova. Nesse caso, o administrador da nuvem pode criar um filtro customizado que atenda às suas necessidades.

Considerando o nosso cenário, é importante garantir que instâncias requisitando uma certa quantidade de EPC sejam implantadas em nós de computação apropriados, ou seja, os que tem capacidade de oferecer o tamanho requisitado de EPC. Dito isso, não existe um filtro padrão no OpenStack que entenda e implemente essa lógica, então nós criamos um, chamado `SgxEpcFilter`, com código detalhado no Apêndice A.

Todos os filtros do OpenStack contém um método base, chamado pelo escalonador durante o processo de criação de uma instância, que determina se o nó de computação está apto para receber um novo recurso baseado nas condições declaradas nesse método. Para o `SgxEpcFilter`, é checado se a quantidade de EPC sendo requisitada através do flavor é menor do que a quantidade disponível

[1]KVM e QEMU trabalham juntos no provisionamento de instâncias no OpenStack. A parametrização em `virt_type` no *nova.conf* indica o uso de KVM como gerenciador dos módulos de *kernel* durante a virtualização, o que é necessário para o uso do *kvm-sgx*.

no nó de computação. Se sim, então o métoda retorna `True` e o nó passa com sucesso pelo filtro. Se não existe EPC disponível, o nó é considerado inapto a receber uma nova instância, e portanto, não passa no filtro.

O cálculo da quantidade livre de EPC é baseado em experimentos anteriores que estressaram o sistema de forma a detectar quanto de EPC uma máquina física pode realmente prover. Além disso, usamos um *driver* Linux SGX modificado que possui um script para recuperar a quantidade de EPC livre da máquina. Ambos os testes retornaram um resultado próximo a $90\ MB$. Ao rodar o script[2] em VMs criadas na nuvem OpenStack, vemos que ele retorna exatamente a quantidade de EPC requisitada pelo *flavor*, o que também contribui para acreditarmos no valor real de $90\ MB$. Assim, sabemos o máximo de EPC que pode ser alocado a uma máquina física, e portanto, a um nó de computação.

Dessa forma, usamos essa informação para o cálculo final da quantidade livre de EPC, junto ao cálculo da quantidade já em uso pelo nó de computação. Isso é importante pois o escalonador precisa desse valor para poder determinar se o dado nó ainda é capaz de receber novas instâncias. Para isso, quando uma instância é criada com sucesso, o valor de EPC alocado para ela é incrementado no objeto `HostState`, responsável por guardar as informações referentes ao estado de um *host* e o uso de seus recursos, como CPU, memória, disco, entre outros. O Apêndice A mostra trechos de código que adicionam o atributo `epc_used` ao objeto `HostState`, e em seguida o momento onde essa variável é atualizada a partir do *flavor* escolhido.

Finalmente, de posse desses dois valores, o máximo de EPC que pode ser alocado e a quantidade já em uso pelo nó de computação, o filtro é capaz de calcular o EPC livre, e determinar se o nó está apto a receber mais uma instância. Para habilitar o filtro `SgxEpcFilter`, é necessário incluí-lo junto aos demais filtros que o administrador da nuvem seleciona, definidos no arquivo de configuração do Nova.

C. Contabilidade de Recursos

Dentro de um ambiente de nuvem é de grande importância contabilizar a quantidade de recursos sendo utilizada, seja para embasar modelos de cobrança ou estar preparado para melhor atender as necessidades dos usuários, escalando o ambiente de acordo. Considerando o OpenStack, o serviço Nova oferece por padrão mecanismos para contabilidade do uso de recursos como memória, disco e CPU. Administradores da nuvem podem, então, visualizar quanto desses recursos está em uso em

[2]O *driver* modificado que contém esse script foi desenvolvido pela TU-Dresden e UNINE, parceiros no projeto SecureCloud.

cada projeto, e baseado nisso, por exemplo, cobrar os usuários apropriadamente.

Dessa forma, considerando o nosso cenário, é importante que seja possível contabilizar o uso de EPC para que o administrador da nuvem possa se planejar melhor, oferecer *flavors* que atendam às expectativas dos usuários, e até mesmo balancear os recursos oferecidos por cada *flavor* de forma a fazer um melhor uso da máquina SGX como um todo. Por exemplo, se um projeto usa $90\ MB$ de EPC de uma mesma máquina mas apenas 30% de memória RAM disponível é utilizada, 70% desse recurso fica inutilizado já que $90\ MB$ é o máximo de EPC que um nó de computação pode oferecer. Da mesma forma, se 100% do disco é utilizado, mas só $50\ MB$ de EPC, $40\ MB$ desse recurso fica parado, já que nenhuma outra instância pode ser alocada àquele nó que não tem mais disco disponível.

O Código Fonte A.6 no Anexo A detalha as mudanças feitas no código que calcula o uso de recursos baseado na configuração do *flavor* utilizado nas instâncias. Tal utilização pode ser calculada para um espaço específico de tempo, determinado por dias, ou, para todas as instâncias que já foram parte de um projeto desde a inicialização do sistema.

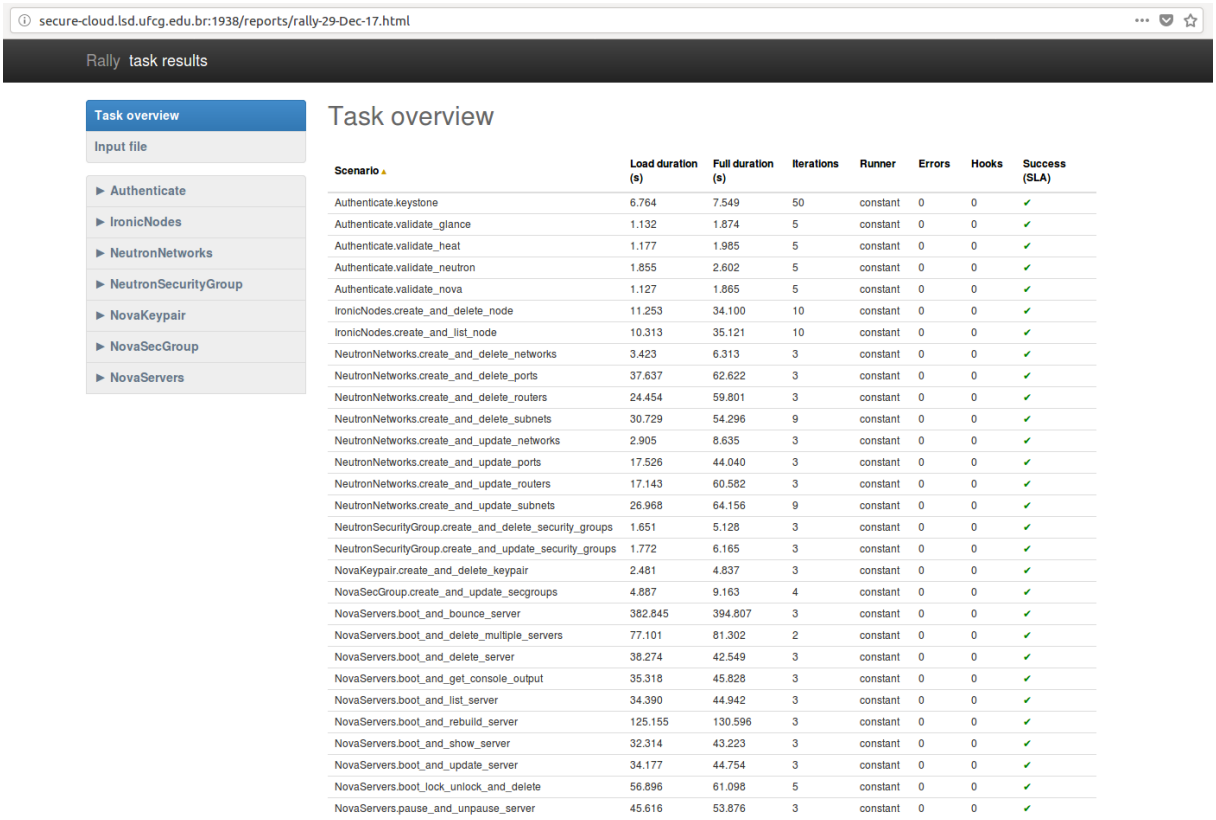
Para o administrador do sistema, a visualização desses dados contabilizados pode ser feita através do Horizon ou via linha de comando ao utilizar o cliente Python do serviço Nova. Na nossa abordagem, modificamos o cliente, que faz uso do comando `nova usage-list`, para exibir o uso de recursos como memória, disco e CPU para todos os projetos de uma dada nuvem. Note que, para ter acesso a esses dados, é necessário ser administrador da nuvem. O Código Fonte A.7 mostra as modificações feitas no código do cliente Python do Nova para adicionar uma nova coluna ao resultado do comando `nova usage-list` já existente.

4.2.4 Validação da implementação

Para validação dessa implementação, nós utilizamos uma ferramenta de *benchmarking* do OpenStack chamada Rally. Tal ferramenta foi utilizada para estressar o sistema e verificar se ele funciona bem quando submetido a cargas mais pesadas de processamento e utilização de recursos, após as modificações feitas para integrar o SGX. Por exemplo, o sistema é submetido a testes de criação, remoção, atualização e reconstrução de uma dada quantidade de instâncias por um número de vezes. Assim como para o Nova, também são realizados testes para outros serviços essenciais da nuvem como o de gerência de redes e autenticação de usuários. Dessa forma, vemos também se as modificações influenciaram outros serviços.

Por fim, é importante destacar ainda que os testes executados pelo Rally são configurados para rodar diariamente, de forma a monitorar o comportamento da nuvem e identificar possíveis falhas o mais rápido possível. Os resultados para o nosso ambiente seguro podem ser vistos na figura 4.4

abaixo. Vemos que todos os testes referentes à gerência de redes, autenticação de usuários e as mais diversas operações realizadas sobre instâncias foram executados com sucesso.



Scenario	Load duration (s)	Full duration (s)	Iterations	Runner	Errors	Hooks	Success (SLA)
Authenticate.keystone	6.764	7.549	50	constant	0	0	✔
Authenticate.validate_glance	1.132	1.874	5	constant	0	0	✔
Authenticate.validate_heat	1.177	1.985	5	constant	0	0	✔
Authenticate.validate_neutron	1.855	2.602	5	constant	0	0	✔
Authenticate.validate_nova	1.127	1.865	5	constant	0	0	✔
IronicNodes.create_and_delete_node	11.253	34.100	10	constant	0	0	✔
IronicNodes.create_and_list_node	10.313	35.121	10	constant	0	0	✔
NeutronNetworks.create_and_delete_networks	3.423	6.313	3	constant	0	0	✔
NeutronNetworks.create_and_delete_ports	37.637	62.622	3	constant	0	0	✔
NeutronNetworks.create_and_delete_routers	24.454	59.801	3	constant	0	0	✔
NeutronNetworks.create_and_delete_subnets	30.729	54.296	9	constant	0	0	✔
NeutronNetworks.create_and_update_networks	2.905	8.635	3	constant	0	0	✔
NeutronNetworks.create_and_update_ports	17.526	44.040	3	constant	0	0	✔
NeutronNetworks.create_and_update_routers	17.143	60.582	3	constant	0	0	✔
NeutronNetworks.create_and_update_subnets	26.968	64.156	9	constant	0	0	✔
NeutronSecurityGroup.create_and_delete_security_groups	1.651	5.128	3	constant	0	0	✔
NeutronSecurityGroup.create_and_update_security_groups	1.772	6.165	3	constant	0	0	✔
NovaKeypair.create_and_delete_keypair	2.481	4.837	3	constant	0	0	✔
NovaSecGroup.create_and_update_secgroups	4.887	9.163	4	constant	0	0	✔
NovaServers.boot_and_bounce_server	382.845	394.807	3	constant	0	0	✔
NovaServers.boot_and_delete_multiple_servers	77.101	81.302	2	constant	0	0	✔
NovaServers.boot_and_delete_server	38.274	42.549	3	constant	0	0	✔
NovaServers.boot_and_get_console_output	35.318	45.828	3	constant	0	0	✔
NovaServers.boot_and_list_server	34.390	44.942	3	constant	0	0	✔
NovaServers.boot_and_rebuild_server	125.155	130.596	3	constant	0	0	✔
NovaServers.boot_and_show_server	32.314	43.223	3	constant	0	0	✔
NovaServers.boot_and_update_server	34.177	44.754	3	constant	0	0	✔
NovaServers.boot_lock_unlock_and_delete	56.896	61.098	5	constant	0	0	✔
NovaServers.pause_and_unpause_server	45.616	53.876	3	constant	0	0	✔

Figura 4.4: Resultado do Rally após integração do OpenStack com SGX.

4.3 Avaliação de desempenho

Nesta seção queremos avaliar o desempenho da nuvem segura proposta por este trabalho, considerando aspectos que envolvem a instalação do kernel modificado *kvm-sgx* nos nós de computação associados, e seu uso nas máquinas virtuais que são gerenciadas pelo hipervisor especial. Os cenários serão avaliados considerando métricas como taxa de transferência de dados e tempo de execução de um conjunto de tarefas em um *benchmark* que usa altos índices de CPU.

A seguir apresentamos o *benchmark* escolhido, o experimento e cenários de execução pensados considerando ambientes que usam *kvm-sgx*, as hipóteses de pesquisa estudadas aqui e por fim a análise propriamente dita dos resultados desse experimento.

4.3.1 O benchmark

O *benchmark* escolhido para essa avaliação de desempenho foi o `Sysbench` [30], uma ferramenta de *benchmarking* disponível para diversos sistemas operacionais, que combinada à plataformas como o MySQL, avalia fatores relacionados ao desempenho do sistema quando executando um banco de dados submetido à uma carga intensa de operações.

`Sysbench` faz uso de testes simples que medem o desempenho de um sistema em cenários diferentes de uso intenso, dando ao usuário uma visão geral de seu comportamento sem a necessidade de configurar *benchmarks* complexos, e até mesmo sem instalar um banco de dados propriamente dito. Os testes disponíveis atualmente permitem avaliar um sistema quanto à taxa de transferência de dados em arquivos, escalonamento de CPU, alocação de memória e velocidade de transferência, desempenho do banco de dados, entre outros.

Na execução desse *benchmark*, um número específico de *threads* é criado e todas executam requisições em paralelo. A carga exata produzida por essas requisições depende do tipo de teste escolhido. Além disso, o usuário pode limitar o número de requisições enviadas e o tempo total desejado para execução do *benchmark*.

Os tipos de teste em foco neste estudo são:

- **CPU:** Nesse modo, cada requisição consiste no cálculo de números primos existentes até um valor especificado pelo usuário através do parâmetro `--cpu-max-primes`. Todos os cálculos são feitos usando inteiros de 64-bit. Cada *thread* criada executa as requisições concorrentemente até que o número total de requisições ou o tempo total de execução do *benchmark* exceda os limites especificados nas opções padrão do comando que inicia o teste.

- **Operações em disco:** Esse modo pode ser usado para produzir vários tipos de carga executadas em arquivos em disco. Para isso, três estágios devem ser seguidos. O primeiro consiste na preparação do ambiente para execução do *benchmark* através da criação de um número específico de arquivos, com um tamanho total pré-definido. Em seguida, cada *thread* executa operações de, por exemplo, escrita e leitura, nos arquivos criados no estágio anterior. Outras operações possíveis incluem `seqwr` (escrita sequencial), `seqrewr` (reescrita sequencial), `seqrd` (leitura sequencial), `rndrd` (leitura aleatória), `rndwr` (escrita aleatória), `rndrw` (leitura/escrita aleatória).

4.3.2 O experimento

O experimento proposto visa avaliar o desempenho do sistema em cenários que utilizam o kernel *kvm-sgx* modificado para criação de máquinas virtuais SGX, comparando esses resultados com outros cenários que não fazem uso desse hipervisor especial. Dessa forma, as hipóteses consideradas nesse estudo assumem que não existe diferença de desempenho quando considerando essas configurações, e avalia sua veracidade com base nos resultados dos testes de CPU e operações em disco propostos pelo *benchmark* `Sysbench`.

Para os cenários que consideram o uso de *kvm-sgx* em nós de computação na nuvem segura, são avaliadas as seguintes hipóteses:

- H0.1: Não existe diferença no desempenho de CPU para nós de computação com e sem *kvm-sgx*.
- H0.2: Não existe diferença no desempenho de operações em disco para nós de computação com e sem *kvm-sgx*.

Já para os cenários que consideram o acesso ao SGX em máquinas virtuais na nuvem segura, as seguintes hipóteses são propostas:

- H1.1: Não existe diferença no desempenho de CPU para máquinas virtuais com e sem acesso SGX.
- H1.2: Não existe diferença no desempenho de operações em disco para máquinas virtuais com e sem acesso SGX.

Na execução do experimento, foram utilizados nós de computação associados à nuvem segura, bem como máquinas virtuais instanciadas nessa infraestrutura. A Tabela 4.3 a seguir detalha a configuração dos ambientes que executaram os testes de *benchmarking*.

Foram executados dois tipos de testes, um para medir o desempenho de CPU e outro que coleta dados sobre a taxa de transferência de dados em operações em disco. O comando no Código Fonte 4.1 mostra o teste para CPU, que define um limite de 30000 números primos (parâmetro `--cpu-max-prime` explicado anteriormente) a serem calculados pelo *benchmark*.

[3]Detalhes da configuração do servidor SuperServer 5019S-MR podem ser encontrados em `https://www.supermicro.com/products/system/1U/5019/SYS-5019S-MR.cfm`

ID	Ambiente	Modelo	Kernel
A1.1	Nó de computação SGX	SuperServer 5019S-MR[3]	kvm-sgx-4.11.0
A1.2	Nó de computação regular	SuperServer 5019S-MR	Linux 4.4.0-96
A2.1	Máquina virtual SGX	Hospedeiro SuperServer 5019S-MR	kvm-sgx-4.11.0
A2.2	Máquina virtual regular	Hospedeiro SuperServer 5019S-MR	Linux 4.4.0-96

Tabela 4.3: Ambientes de execução dos testes de *benchmarking* para KVM.

```
1 $ sysbench --test=cpu --cpu-max-prime=30000 run
```

Código Fonte 4.1: Comando para execução do benchmark de CPU do Sysbench.

```
1 $ sysbench --test=fileio --file-total-size=2G --file-num=64 prepare
2 $ sysbench --test=fileio --file-total-size=2G --file-test-mode=rndrw --
    max-time=120 --max-requests=0 --file-block-size=4K --file-num=64 run
```

Código Fonte 4.2: Comando para execução do benchmark de operações em disco do Sysbench.

Os comandos no Código Fonte 4.2 mostram os passos de execução do teste para operações em disco. Primeiro são criados 64 arquivos, de tamanho total 2 GB, através do comando `prepare`. Em seguida, o teste é executado considerando um limite de tempo de 120 segundos para cada execução, que vai gerar operações do tipo `rndrw`, ou seja, leituras e escritas não sequenciais.

Cada teste de CPU e disco foi executado 30 vezes para todos os cenários propostos. Trechos de exemplos de saída para os comandos no Código Fonte 4.1 e 4.2 são exibidos abaixo. No exemplo no Código 4.3, estamos interessados na informação em *"total time:"*, que representa o tempo de execução das tarefas executadas pelo *benchmark* de CPU em uma rodada do experimento. No exemplo em 4.4, o dado de interesse é o valor entre parênteses após a informação *"Total transferred"*, que representa a taxa de transferência para as operações em disco consideradas pelo *benchmark*.

```
sysbench 0.4.12:  multi-threaded system evaluation benchmark
Maximum prime number checked in CPU test: 30000

Test execution summary:
    total time:                          33.2892s
    total number of events:              10000
    total time taken by event execution: 33.2885
    per-request statistics:
         min:                                  3.32ms
         avg:                                  3.33ms
         max:                                  8.61ms
         approx.  95 percentile:               3.33ms
```

Código Fonte 4.3: Saída do comando que executa o benchmark de CPU.

```
sysbench 0.4.12:  multi-threaded system evaluation benchmark
64 files, 32Mb each
2Gb total file size
Doing random r/w test

Operations performed:  25440 Read, 16960 Write, 27074 Other = 69474 Total
Read 99.375Mb Written 66.25Mb Total transferred 165.62Mb (1.3801Mb/sec)
  353.30 Requests/sec executed

Test execution summary:
    total time:                          120.0116s
    total number of events:              42400
    total time taken by event execution: 0.2693
```

Código Fonte 4.4: Saída do comando que executa o benchmark de operações em disco.

4.3.3 Análise dos dados

A Figura 4.5 mostra os resultados da execução do *benchmark* de CPU para todos os cenários propostos neste estudo, inclusive os referentes à contêineres LXD que serão detalhados na Seção 5.1.3. Em vermelho estão os *boxplots* que representam cenários onde SGX não estava habilitado, e em azul os ambientes com SGX habilitado e kernel *kvm-sgx* instalado.

Podemos ver que os resultados variaram pouco dada a pequena amplitude interquartílica para as caixas em cada cenário. Além disso, a ausência de semirretas longas o suficiente para se destacarem significa que os limites superiores e inferiores determinados pelo conjunto de dados também foram bastante próximos. Vemos ainda que, para todos os cenários, as caixas se encontram entre 30 e aproximadamente 33 segundos, mostrando que entre eles há uma diferença pequena nos tempos de execução do *benchmark* de CPU considerado.

Para corroborar essa análise, apresentamos um sumário dos dados representados pela variável `execution_time_sec` coletados para os cenários A1.1, A1.2, A2.1 e A2.3, respectivamente definidos na Tabela 4.3. Além disso, como vemos na Tabela 4.5, a variância dos dados pode ser confirmada como sendo um valor pequeno o suficiente para dizermos que as diferenças entre os cenários propostos são insignificantes quando considerando desempenho de CPU.

A Figura 4.6 mostra os resultados da execução do *benchmark* de operações em disco para os cenários propostos na Tabela 4.3, inclusive os referentes à contêineres LXD que também serão detalhados na Seção 5.1.3. Assim como para o gráfico de CPU, em vermelho estão os *boxplots* que representam cenários onde o uso de SGX não estava habilitado, e em azul os ambientes com SGX habilitado e kernel *kvm-sgx* instalado.

De modo geral, como foi observado para o gráfico 4.5, também vemos que os resultados variaram pouco dada a pequena amplitude interquartílica para cada cenário, mesmo sendo um pouco maiores. Além disso, percebemos da mesma forma a ausência de semirretas longas para cada caixa, significando que os limites superiores e inferiores determinados pelos dados também foram bastante próximos. Todas as caixas se encontram entre 1.0 e aproximadamente 1.5 Mb/s, mostrando que entre eles há uma diferença pequena na taxa de transferência coletada para o *benchmark* de operações em disco. Ainda vemos que, para cenários com SGX, essa taxa é um pouco menor, levando a crer que os dados precisam de mais tempo para serem transferidos.

De forma a confirmar o que observamos na Figura 4.6, apresentamos a seguir o sumário dos dados representados pela variável `transfer_rate_mbsec` coletados para os cenários A1.1, A1.2, A2.1 e A2.3, respectivamente definidos na Tabela 4.3.

Além disso, como vemos na Tabela 4.7, a variância dos dados pode ser confirmada como sendo um valor pequeno o suficiente para dizermos que as diferenças entre os cenários propostos são insignificantes quando considerando operações em disco.

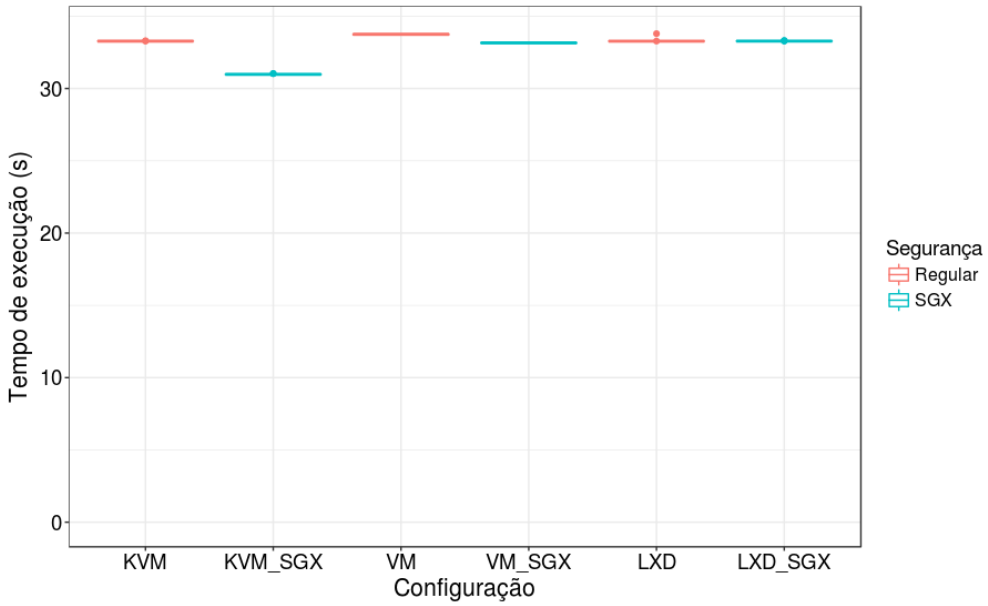


Figura 4.5: Tempo de execução dos testes de CPU para cenários KVM.

ID	Ambiente	Min.	1st Qu.	Mediana	Média	3rd Qu.	Max.
A1.1	Nó de computação SGX	30.95	30.97	30.98	30.98	30.99	31.03
A1.2	Nó de computação regular	33.27	33.27	33.28	33.28	33.28	33.30
A2.1	Máquina virtual SGX	33.10	33.14	33.16	33.15	33.17	33.20
A2.2	Máquina virtual regular	33.67	33.71	33.75	33.74	33.78	33.80

Tabela 4.4: Sumário dos dados baseado na variável `execution_time_sec` para os testes de desempenho de CPU em cenários KVM.

ID	Ambiente	Variância para execution_time_sec
A1.1	Nó de computação SGX	0.0004066278
A1.2	Nó de computação regular	0.0000315
A2.1	Máquina virtual SGX	0.0008121768
A2.2	Máquina virtual regular	0.001797208

Tabela 4.5: Variância da variável `execution_time_sec` nos testes de desempenho de CPU para cenários KVM.

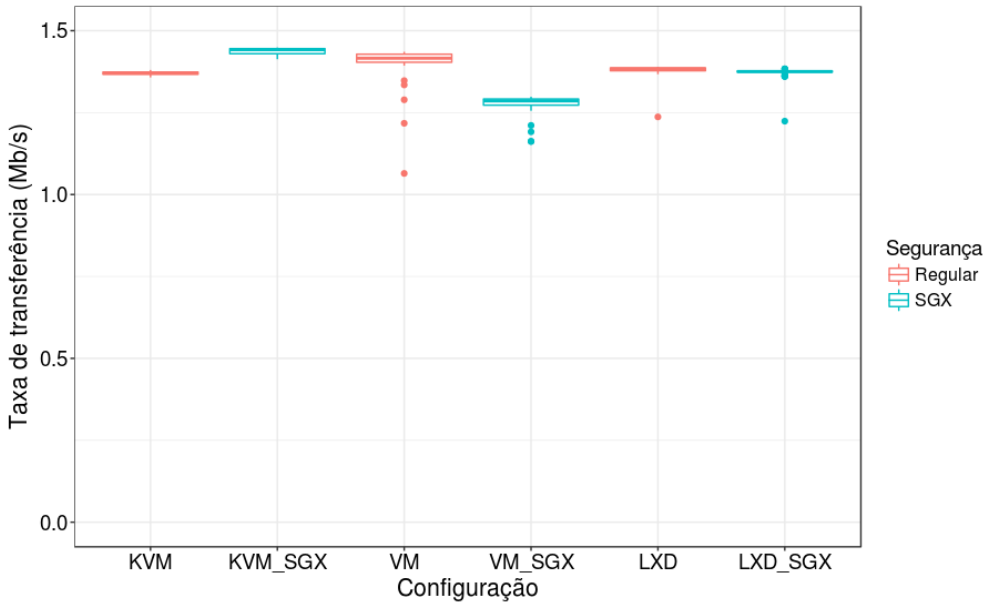


Figura 4.6: Taxa de transferência dos testes em arquivos em disco.

ID	Ambiente	Min.	1st Qu.	Mediana	Média	3rd Qu.	Max.
A1.1	Nó de computação SGX	1.413	1.430	1.442	1.436	1.445	1.448
A1.2	Nó de computação regular	1.357	1.367	1.370	1.370	1.374	1.380
A2.1	Máquina virtual SGX	1.162	1.273	1.286	1.270	1.291	1.298
A2.2	Máquina virtual regular	1.064	1.404	1.416	1.391	1.428	1.435

Tabela 4.6: Sumário dos dados baseado na variável `transfer_rate_mbsec` nos testes de operações em disco em cenários KVM.

ID	Ambiente	Variância para transfer_rate_mbsec
A1.1	Nó de computação SGX	0.0001253494
A1.2	Nó de computação regular	0.0000334474
A2.1	Máquina virtual SGX	0.001411568
A2.2	Máquina virtual regular	0.006040507

Tabela 4.7: Variância para a variável `transfer_rate_mbsec` nos testes de operações em disco em cenários KVM.

Capítulo 5

Contêineres e SGX em ambientes de nuvem

5.1 Contêineres seguros no OpenStack usando SGX

5.1.1 Contêineres LXD

Entre os serviços de conteinerização disponíveis atualmente está o Linux LXD. Seguindo os passos do *Linux Containers* (LXC)[1], LXD[2] é apresentado como uma nova geração desse serviço, fundado e, atualmente conduzido, pela Canonical Ltd com contribuições de algumas outras companhias e desenvolvedores. LXD é escrito em Go e desenvolvido com licenciamento Apache 2.

Entretanto, é importante destacar que LXD não é um LXC reescrito de maneira diferente, mas sim uma abordagem projetada e construída em cima do LXC, utilizando-o através de bibliotecas como a `liblxc` para criar e gerenciar contêineres. Dessa forma, LXD pode ser visto como uma alternativa ao LXC, melhorado com funcionalidades como a capacidade de ser controlado através da rede. Sua base é um *daemon* privilegiado que expõe uma API REST através de um socket ou da própria rede, se habilitado. Clientes podem então acessar aquele ambiente e realizar todas as operações que o LXD oferece.

Considerando isso, como outros tipos de contêineres, LXD oferece aos usuários uma experiência similar ao uso de máquinas virtuais, mas na verdade, usando contêineres Linux. Algumas de suas principais características são mecanismos de segurança como o uso de contêineres sem privilégios e

[1]Linux Containers. https://linuxcontainers.org/ [Online; Último acesso: 22 de janeiro, 2018]

[2]LXD Containers. https://linuxcontainers.org/lxd/ [Online; Último acesso: 22 de janeiro, 2018]

restrição no uso de recursos, facilidade de gerência dos contêineres através do uso da API REST citada anteriormente, escalabilidade que assegura a capacidade do LXD criar e gerenciar ambientes com poucos contêineres em uma máquina própria e milhões em servidores robustos, controle avançado no uso de recursos como CPU, memória, rede, disco e kernel, possibilidade de habilitação de dispositivos localizados na máquina física como USB, GPU, e, de nosso interesse, o SGX, além de muitas outras como gerência de rede e armazenamento.

Finalmente, uma das grandes vantagens LXD, e de interesse nessa pesquisa, é sua integração com ambientes de nuvem como o OpenStack, utilizada aqui também para integração com SGX. O projeto Nova oferece um *driver* especial usado para criar contêineres LXD de maneira transparente ao usuário, que requisita um contêiner da mesma forma que requisitaria uma máquina virtual. É necessário apenas que seja usada uma imagem específica LXD e escolhido um *flavor* que aponte para um nó de computação com o driver LXD habilitado. Mais detalhes na seção a seguir que apresenta um LXD seguro, integrado ao OpenStack SGX.

5.1.2 Contêineres LXD seguros no Nova

A nuvem OpenStack oferece a possibilidade de criar contêineres LXD que funcionam como recursos oferecidos pelo serviço Nova, responsável por virtualizar máquinas em nós de computação associados à dada nuvem. Assim, o usuário requisita um contêiner da mesma forma que requisitaria uma máquina virtual, escolhendo um *flavor* que direcione a requisição para um nó que provisione recursos via um hipervisor LXD especial. Como resultado, o usuário irá receber um contêiner Linux, de forma transparente, e hospedado em nuvem.

Queremos que esses contêineres sejam capazes de acessar o dispositivo *isgx* da máquina na qual eles estão hospedados, ou seja, o dispositivo que é habilitado no hospedeiro quando o *driver* SGX é instalado. Para isso, a única dependência é que tal *driver* esteja disponível no nó, e sua instalação feita como descrito no repositório disponibilizado pela Intel. Em um ambiente fora da nuvem, para que contêineres LXD sejam criados com acesso ao dispositivo *isgx* de seus respectivos hospedeiros, os comandos detalhados no Apêndice A são executados após a criação dos recursos. Basicamente, como para qualquer contêiner hospedado em uma máquina SGX, e que deseje acessar as suas funcionalidades, é preciso fazer uma ligação entre o dispositivo *isgx* criado após instalação do *driver* e o contêiner em si.

Dessa forma, considerando um ambiente OpenStack, o código do hipervisor LXD precisa ser modificado para que os contêineres sejam criados já com acesso ao dispositivo *isgx* da máquina hospedeira. De maneira similar ao que é feito para máquinas virtuais, modificamos o código do *driver*

LXD, e adicionamos os comandos detalhados no Apêndice A ao método `spawn()`, responsável por direcionar o fluxo de criação de um dado recurso LXD na nuvem.

Nesse método, os passos para criação do contêiner incluem recuperar a imagem armazenada a partir do serviço Glance, conectar as redes necessárias para inicializar o recurso, criar um *profile* junto ao LXD da máquina e por fim instanciar e iniciar o contêiner. Ao fim do processo, habilitamos o dispositivo *isgx* e modificamos a sua permissão de acesso, de forma a provisionar um contêiner LXD que consegue fazer uso do SGX disponível em sua máquina hospedeira.

Aqui observamos que, diferentemente do processo para criação de máquinas virtuais, não é necessário alocar previamente a quantidade de EPC a ser utilizada, e dessa forma, a integração é facilitada por não precisar de filtros especiais, alterações no *flavor* nem as demais particularidades especificadas para VMs. Além disso, para contêineres LXD, não precisamos da instalação do kernel especial *kvm-sgx* para virtualização do recurso EPC, que é provido diretamente ao contêiner hospedado na máquina SGX através dos mecanismos próprios dos serviços de conteinerização. Em contrapartida, há menos isolamento entre contêineres, ou seja, todos os contêineres em um nó estarão utilizando o mesmo dispositivo *isgx*, o que pode ser um ponto de vulnerabilidade à certos ataques.

5.1.3 Avaliação de desempenho

Nesta seção queremos avaliar o desempenho da nuvem segura proposta por este trabalho, considerando aspectos que envolvem a habilitação do acesso ao SGX da máquina hospedeira aos contêineres LXD hospedados nela. Os cenários avaliados consideram métricas como taxa de transferência de dados e tempo de execução de um conjunto de tarefas em um *benchmark* que usa altos índices de CPU, assim como para os cenários com *kvm-sgx*.

O *benchmark* escolhido aqui é o mesmo para os demais testes executados anteriormente nesta pesquisa, o `Sysbench`. A seguir apresentamos o experimento realizado junto à cenários de execução pensados considerando ambientes que usam LXD e SGX, as hipóteses de pesquisa estudadas aqui e por fim a análise propriamente dita dos resultados desse experimento.

A. O experimento

O experimento proposto visa avaliar o desempenho do sistema em cenários que utilizam contêineres LXD com acesso à SGX, comparando esses resultados com outros cenários que consideram contêineres LXD sem esse acesso. Assim, as hipóteses consideradas nesse estudo assumem que não existe diferença de desempenho quando considerando essas configurações, e avalia sua veracidade com base

ID	Ambiente	Modelo	SGX
A3.1	Contêiner LXD SGX	Hospedeiro SuperServer 5019S-MR	Habilitado
A3.2	Contêiner LXD regular	Hospedeiro SuperServer 5019S-MR	Desabilitado

Tabela 5.1: Ambientes de execução dos testes de *benchmarking* para LXD.

ID	Ambiente	Variância para execution_time_sec
A3.1	Contêiner LXD SGX	0.00008683895
A3.2	Contêiner LXD regular	0.009043391

Tabela 5.2: Variância da variável `execution_time_sec` nos testes de desempenho de CPU para cenários LXD.

nos resultados dos testes de CPU e operações em disco propostos pelo *benchmark* `Sysbench`.

Para os cenários que consideram o acesso ao SGX em contêineres LXD na nuvem segura, são avaliadas as seguintes hipóteses:

- H2.1: Não existe diferença no desempenho de CPU para contêineres LXD com e sem acesso ao SGX da máquina hospedeira.
- H2.2: Não existe diferença no desempenho de operações em disco para contêineres LXD com e sem acesso ao SGX da máquina hospedeira.

Na execução do experimento, foram utilizados nós de computação associados à nuvem segura que possuíam um *driver* SGX instalado, como é requerido por padrão para permitir o vínculo dos contêineres ao SGX da máquina hospedeira. Contêineres LXD com e sem acesso à esse SGX foram instanciados nessa infraestrutura. A Tabela 5.1 a seguir detalha a configuração dos ambientes que executaram os testes de *benchmarking* para cenários com LXD.

Dois tipos de testes foram executados, um para medir o desempenho de CPU e outro que coleta dados sobre a taxa de transferência de dados em operações em disco. Detalhes sobre como eles são executados e os parâmetros utilizados estão definidos na Seção 4.3.2.

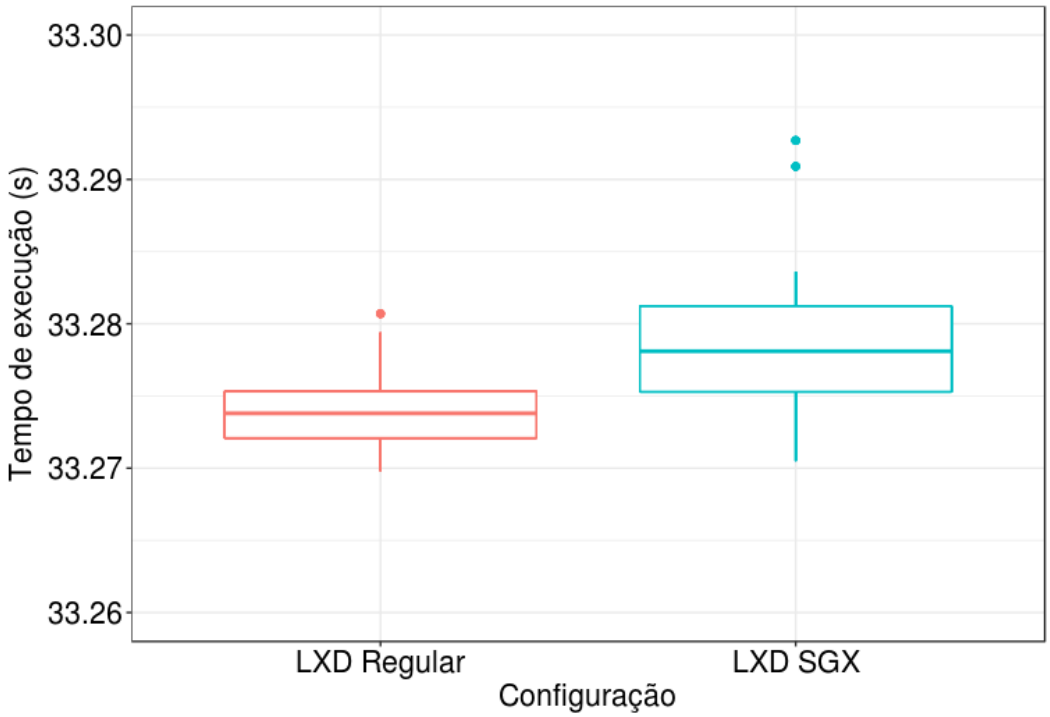


Figura 5.1: Tempo de execução dos testes de CPU para cenários LXD.

B. Análise dos dados

A Figura 5.1 mostra os resultados da execução do *benchmark* de CPU para os cenários que consideram contêineres LXD. Em vermelho estão os *boxplots* que representam contêineres onde acesso ao SGX não estava habilitado, e em azul os ambientes com acesso ao SGX de seus hospedeiros.

Para melhor visualização da amplitude interquartílica para cada cenário, o intervalo de tempo no eixo Y foi limitado a uma precisão de milésimos de segundo. Ainda assim, vemos que ambas as caixas se encontram na casa dos 33s, o que mostra o quanto a variação dessa variável no conjunto de dados é pequena, considerando o *benchmark* de CPU utilizado. A mesma análise se estende para as semirretas representativas dos limites superiores e inferiores das execuções para cada cenário. Vemos que elas não são suficientemente longas para se destacarem, significando que tais limites também foram bastante próximos.

Além disso, como vemos na Tabela 5.2, a variância dos dados pode ser confirmada como sendo um valor pequeno o suficiente para dizermos que as diferenças entre os cenários propostos são insignificantes quando considerando desempenho de CPU.

Em seguida, temos a Figura 5.2, que mostra os resultados da execução do *benchmark* de operações em disco para os cenários definidos na Tabela 5.1. Assim como para o gráfico de CPU, em vermelho

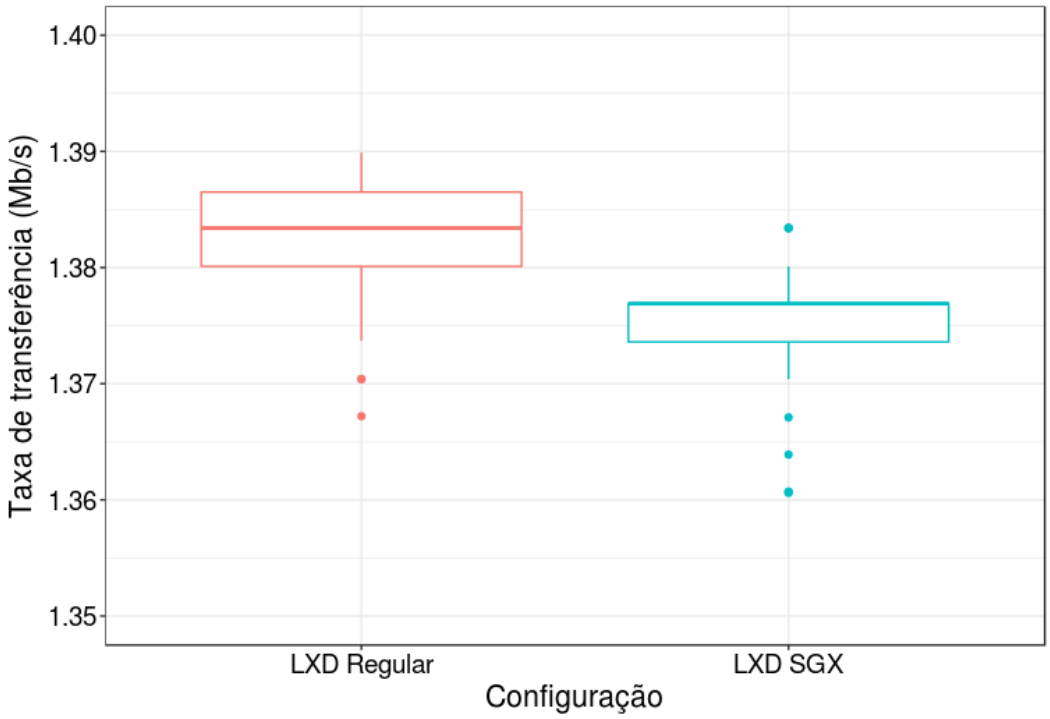


Figura 5.2: Taxa de transferência dos testes de operações em disco para cenários LXD.

ID	Ambiente	Variância para transfer_rate_mbsec
A3.1	Contêiner LXD SGX	0.000795683
A3.2	Contêiner LXD regular	0.0007294827

Tabela 5.3: Variância para a variável `transfer_rate_mbsec` nos testes de operações em disco para cenários LXD.

estão as caixas que representam contêineres onde acesso ao SGX não existia, e em azul os ambientes com SGX habilitado.

Além disso, como vemos na Tabela 5.3, a variância dos dados pode ser confirmada como sendo um valor pequeno o suficiente para dizermos que as diferenças entre os cenários LXD são insignificantes quando considerando operações em disco.

Os resultados também consideram um intervalo de taxas de transferência limitado a uma precisão de poucos bytes por segundo, e mostram uma variação pequena da amplitude interquartílica para cada cenário, sendo a caixa do terceiro quartil tão pequena que, mesmo com uma precisão grande, mal aparece. Além disso, ambas as caixas se encontram entre 1.36 e aproximadamente 1.39 MB/s,

mostrando que entre eles há uma diferença pequena na taxa de transferência coletada para o *benchmark* de operações em disco. Ainda vemos que, para cenários com SGX, essa taxa é um pouco menor, levando a crer que os dados precisam de mais tempo para serem transferidos.

5.2 Clusters de contêineres seguros no OpenStack usando SGX

5.2.1 O serviço OpenStack Magnum

O serviço Magnum [1], oferecido pela plataforma de nuvem OpenStack, disponibiliza um mecanismo de gerenciamento de orquestradores como Docker Swarm, Kubernetes e Apache Mesos, provisionando e orquestrando contêineres diretamente como recursos na nuvem. Magnum ainda utiliza o serviço Heat para orquestrar uma imagem que contém Docker e Kubernetes, e implanta a mesma em clusters de máquinas virtuais que hospedam contêineres provisionados.

Esse componente apresenta duas entidades principais: *clusters* e *cluster templates*, como podemos ver na Figura 5.3. Um *cluster template* é uma coleção de paramêtros que descreve como um *cluster* será construído. Alguns parâmetros são relevantes para a infraestrutura do *cluster*, enquanto outros são específicos do orquestrador de contêiner escolhido para uso. Assim, considerando um *workflow* típico, um usuário cria um *cluster template*, e a partir dele cria um ou mais *clusters*. Um provedor de nuvem pode ainda definir um conjunto de *templates* e oferecê-los à seus usuários.

Um *cluster*, por sua vez, é a instância de um *template*. No processo de criação de um *cluster* Swarm, um número definido de nós e um ou mais *masters*, identificados como gerentes do Swarm, são instanciados em máquinas virtuais como parte do *cluster*. Tais máquinas virtuais precisam de imagens específicas, compatíveis com cada tipo de orquestrador, sendo Fedora Atomic a mais normalmente suportada. É importante perceber que aqui precisamos também utilizar o serviço Glance do OpenStack para armazenamento dessas imagens de *cluster*.

Em linhas gerais, Magnum cria toda a infraestrutura necessária para a integração de um *cluster* com o orquestrador escolhido, de forma que o mesmo esteja habilitado a receber contêineres. Quando a integração é concluída, operações direcionadas ao *cluster* devem ser executadas de maneira transparente, utilizando a própria API do orquestrador escolhido.

Do ponto de vista operacional, para criar os principais componentes do serviço Magnum, é necessário a utilização dos parâmetros listados na Tabela 5.4.

Muitas outras opções podem ser definidas, como o *flavor* a ser utilizado no nó master do cluster,

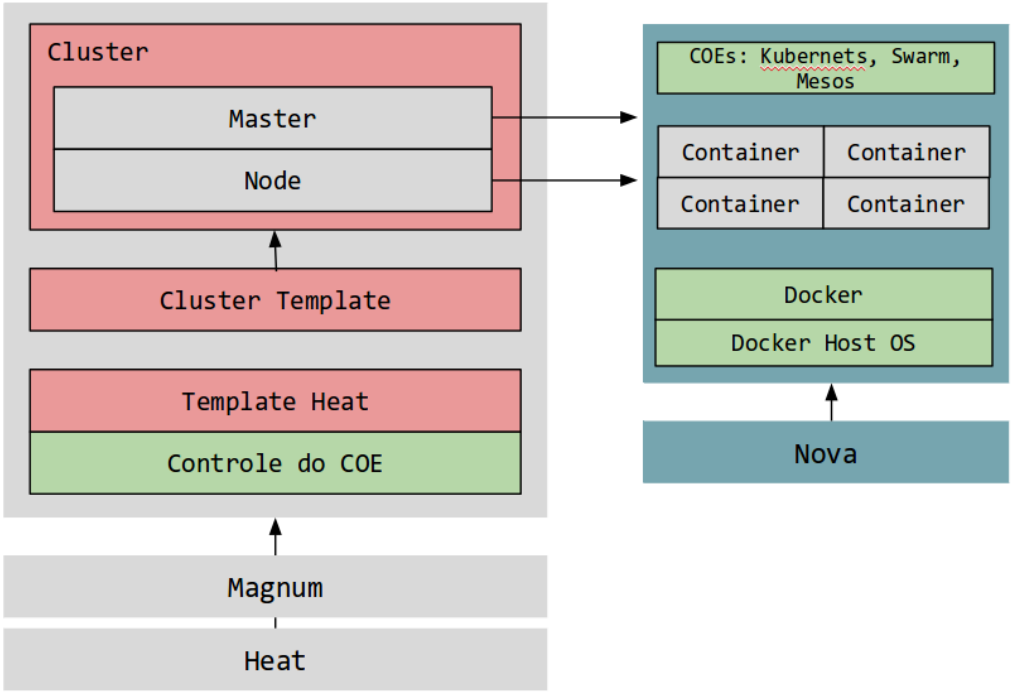


Figura 5.3: Visão geral da arquitetura do Magnum.

bem como nos demais nós, o número de nós a serem criados em um cluster, o tamanho do volume Docker associado, entre outros. Tais parâmetros, porém, não são obrigatórios, e existem valores padronizados para os que ainda são necessários durante a criação de um cluster.

Se um cluster template não está sendo utilizado por nenhum outro cluster, ele pode ser deletado normalmente. Por sua vez, para deletar um cluster, o único parâmetro necessário é o seu nome ou identificador.

5.2.2 Clusters seguros no Magnum

Um *cluster* no Magnum é essencialmente um conjunto de máquinas virtuais provisionadas pelo Nova. Considerando isso, para criar um *cluster* seguro que é capaz de receber contêineres com acesso ao SGX disponível na VM, é preciso apenas assegurar que as instâncias parte do cluster são VMs SGX. Nada muda no processo padrão de criação dos componentes do Magnum, como o *cluster template* e o próprio *cluster*, como explicado na seção anterior. O usuário apenas deve escolher um *flavor* apropriado que defina a quantidade de EPC a ser alocada para cada instância do *cluster* criado.

Após a criação ser concluída, todas as máquinas do *cluster* têm acesso ao SGX através da virtualização provida pelo *kvm-sgx*, assim como acontece para qualquer instância do Nova. No caso do

Cluster Template	
Paramêtro	**Descrição**
–coe <coe>	COE significa "Container Orchestration Engine". COEs suportados e que podem ser usados para criar um cluster são ‘kubernetes’, ‘swarm’, ‘mesos’.
–image <image>	Nome ou identificador da imagem base no Glance a ser implantada nos nós do cluster.
–keypair <keypair>	Nome de uma chave para acesso ssh criada previamente.
–external-network <network>	O nome ou identificador de uma rede Neutron para prover conectividade externa (internet) ao cluster.
Cluster	
Paramêtro	**Descrição**
–cluster-template <template>	Identificador ou nome do ClusterTemplate a ser utilizado.

Tabela 5.4: Parâmetros necessários para criação de componentes base do serviço Magnum.

cluster Magnum, a função de cada máquina é ser capaz de receber contêineres de uma dada aplicação. Para o nosso contexto, essas aplicações são SGX, e para funcionar corretamente, devem ter acesso ao SGX de seus *hosts*. Assim, precisamos instalar o *driver* SGX na máquina virtual, de forma a fazer com o que os contêineres possam acessar o dispositivo `isgx` habilitado pela instalação.

Tal processo de instalação seria simples e direto se a imagem implantada nas máquinas do *cluster* fosse uma das atualmente suportadas pelo *driver* SGX. Entretanto, para clusters do tipo Kubernetes, o orquestrador mais utilizado no ambiente OpenStack [2], um dos requisitos é que a imagem seja do tipo Fedora Atomic, a qual não é suportada por padrão pelo *driver* SGX. Nesse caso, precisamos preparar o ambiente de forma a ser possível a instalação correta do *driver*.

O primeiro passo, como mostra o Código Fonte B.1 no Apêndice B, é a atualização da imagem Fedora Atomic para sua versão mais recente, como requisitado pelo *driver* SGX. Para que a instalação tenha sucesso, é necessário que os módulos mais recentes do *kernel* estejam instalados, e, para Fedora Atomic, só conseguimos obtê-los através da atualização da imagem.

A seguir, é feita a instalação das dependências do driver, incluindo os módulos de kernel atualizados. Para que esse passo, assim como o restante da instalação, seja de sucesso, é necessário ainda

configurar o ambiente para permitir que as operações no diretório raiz do sistema persistam entre reinicializações da instância, o que não é um comportamento padrão para imagens Fedora Atomic. Para isso, é utilizado o comando `sudo ostree admin unlock –hotfix`, como detalhado no Código Fonte B.1 no Apêndice B. O restante do script de instalação do driver segue o processo padrão indicado pelo *driver* SGX. Recapitulando, primeiro a instalação dos módulos de kernel e dependências como mostrado no Código Fonte B.1 e, após isso, o script em B.2 é executado.

Uma abordagem diferente para habilitar o SGX em imagens do tipo Fedora Atomic seria criar as dadas imagens já com o *driver* instalado. Dessa forma, usuários não teriam que instalar o *driver* na mão, o que seria um ponto positivo para prover a segurança sem influenciar diretamente no processo de preparação do ambiente para implantar aplicações. Entretanto, não é possível implementar essa solução no momento devido à uma falha[3] reportada no projeto do *Disk Image Builder* (DIB), um *framework* usado pelo OpenStack para gerar imagens apropriadas para ambientes de nuvem. Assim, não podemos construir uma imagem Fedora Atomic já com o *driver* SGX instalado dadas as própria limitações do DIB.

Para mais detalhes nas modificações necessárias no processo de instalação do driver SGX em imagens do tipo Fedora Atomic 26, leia o Apêndice B que mostra os scripts de instalação utilizados, e ainda apresenta o passo a passo para a instalação também em imagens do tipo Fedora 26.

[3]DIB bug. https://bugs.launchpad.net/diskimage-builder/+bug/1719102 [Online; Último acesso: 22 de janeiro, 2018]

Capítulo 6

Avaliação de aplicações SGX na nuvem segura

6.1 Disseminação de dados de medição de energia em nuvem

Para validação da nuvem segura desenvolvida nesta pesquisa, consideramos a implantação de uma aplicação[1] que requer processamento de dados sensíveis e faz uso da tecnologia de hardware Intel SGX, usada aqui para prover aspectos de segurança à uma nuvem OpenStack.

A aplicação considera o uso de medidores inteligentes para obtenção de medições de energia de casas, prédios, entre outros ambientes, e cálculo do consumo dos mesmos através de mecanismos de agregação. Aplicações desse tipo são motivadas pelos benefícios advindos da análise dos dados de consumo detalhados obtidos através dos medidores inteligentes, que podem detectar anomalias e configurações indesejadas, e a partir disso, gerar recomendações que resultem em um uso mais eficiente de energia.

Por exemplo, a coleta de medições a cada segundo pode facilitar a identificação de aparelhos eletrodomésticos em uma residência. Isso é conhecido como *Non-Intrusive Load Monitoring* (NIALM) [11]. Com essa informação, recomendações customizadas podem levar consumidores a economizar quantidades consideráveis de energia [8]. Por outro lado, informações detalhadas sobre o consumo de energia podem revelar muito mais do que parece. Pesquisas anteriores mostram que até

[1] A aplicação descrita a seguir foi implementada como caso de uso no artigo *Secure and Privacy-Aware Data Dissemination for Cloud-Based Applications* [37], produzido e publicado no contexto desse trabalho.

particularidades a respeito do que um morador assiste em sua TV podem ser detectados através do detalhamento dos dados de energia [25]. Dessa forma, fica claro que tais dados podem revelar muito sobre os hábitos dos indivíduos em uma residência.

Em resumo, podemos dizer que o consumo detalhado de energia é útil. Por um lado, provedores podem usar esses dados para planejar melhor a geração dessa energia, além de influenciar os consumidores quanto ao seu uso. Por outro, consumidores se beneficiam das análises sendo feitas a partir desses dados. Entretanto, apesar dos benefícios, os dados não podem ser confiados a qualquer aplicação e nem todo consumidor vai querer compartilhar seus dados. Consequentemente, um sistema que possibilita usuários a ter um melhor controle acerca de quem acessa seus dados e reduz o risco de vazamento, pode ser a base para outras aplicações sofisticadas que requerem privacidade, não apenas infraestruturas de medidores inteligentes.

Esse problema é abordado pelo sistema descrito aqui através do uso de ferramentas que usam a capacidade do SGX para comunicação entre sistemas e processamento de dados seguros. O sistema proposto combina e estende ferramentas como o próprio SGX, *Secure CONtainer Environment* (SCONE) [9] e *Secure Content-Based Routing* (SCBR) [33], de forma a fazer com que provedores de dados tenham controle sobre quem os acessa e até restrinjam o nível de informação que essas entidades podem consumir. A partir dessa combinação, produzimos um ambiente no qual medidores inteligentes de energia enviam dados sensíveis para serem agregados por entidades confiáveis e, então, consumidos por aplicações que necessitem de um nível menor de confiabilidade ou até que sejam inseguras.

Nesta seção descrevemos as ferramentas utilizadas para construção do sistema, seus componentes e fluxo de envio e agregação de um conjunto de medições. Por fim, validamos sua implantação na nuvem segura e avaliamos o comportamento da aplicação quanto à latência no envio e recebimento de dados pelas partes envolvidas e no consumo de CPU da instância SGX para variados cenários de execução.

6.1.1 Ferramentas SGX utilizadas

A. SCONE

SCONE [9] é um mecanismo de criação de contêineres Docker seguros, que usa SGX para proteger processos dos referidos contêineres através do uso de enclaves. Tal mecanismo oferece contêineres seguros junto à sistemas operacionais inseguros, e faz isso de maneira transparente à ambientes Docker já existentes. Para funcionamento do SCONE é necessário que o hospedeiro possua um *driver*

Linux SGX e um módulo *kernel* do próprio SCONE.

Uma imagem segura a ser utilizada na criação de um contêiner SCONE deve ser construída de maneira diferenciada. É necessário construir um executável da aplicação, e compilá-lo junto com suas dependências e o módulo SCONE. Um cliente SCONE seguro é então usado para criação e configuração de arquivos e meta-dados necessários para proteção do sistema via *file system* (FS), que contém códigos de autenticação de mensagens e chaves usadas para criptografia, por exemplo. O arquivo de proteção FS é então encriptado e adicionado à imagem. SCONE também suporta encriptação transparente e autenticação de dados via *shields*.

O cliente SCONE é também utilizado para instanciar e se comunicar com contêineres seguros. Cada um deles requer um *start-up configuration file* (SCF) para ser inicializado. Esse arquivo contém dados relevantes à segurança do sistema, como chaves de encriptação e *hash* do arquivo FS de proteção. Tal arquivo SCF só pode ser acessado por um enclave verificado. Considerando isso, durante a inicialização de um enclave, o SCF é recebido através de uma conexão segura TLS. Como alternativa, atestação remota provida pelo SGX poderia atestar o enclave para verificar sua identidade.

Resultados do uso da abordagem SCONE apontam para a preservação da confidencialidade e integridade dos serviços em contêineres usando Intel SGX. Um contêiner SCONE possui um *Trusted Computing Base* (TCB) de até no máximo duas vezes o tamanho do código da aplicação e são compatíveis com Docker. Ainda foi percebido que o uso de *system calls* assíncronas reduz efetivamente a sobrecarga das transições entre o enclave e o exterior impostas pelo modelo do SGX. Por fim, SCONE não requer mudanças na aplicação, ou no *kernel* do Linux além da recompilação da aplicação pelo cliente SCONE e o carregamento do seu próprio módulo de *kernel*.

B. Python SGX

Intel SGX SDK é um conjunto de ferramentas para desenvolvimento de aplicações SGX disponível para as linguagens C e C++. Isso significa que apenas aplicações escritas nessas linguagens podem ser adaptadas para executar e se comunicar com enclaves. Isso é uma limitação para a tecnologia SGX já que, considerando aplicações escritas em outras linguagens, portar código pode ser um obstáculo e levar ao surgimento de falhas adicionais.

Entre linguagens de programação populares, Python merece uma atenção especial. Ano passado, ela foi considerada a linguagem de progração Top 1 no ranking *2017 Programming Languages* promovido pela IEEE[2]. Muitos softwares populares também são escritos em Python, como o próprio OpenStack usado neste trabalho, YouTube, DropBox, Instagram e muitos outros.

[2]http://spectrum.ieee.org/computing/software/the-2017-top-programming-languages

Usar o Intel SDK para implementar aplicações SGX pode gerar um esforço extra na portabilidade de código já existente, ou até na criação de novos pedaços de software. Para satisfazer essa demanda, SCONE provê um compilador C modificado, baseado na biblioteca *libmusl*[3]. Esse compilador, chamado *sgxmusl-gcc*[4], automaticamente gera o código a ser executado dentro dos enclaves SGX, tornando mais simples o processo de implementação de aplicações prontas para serem executadas em hardware protegido. Entretanto, o compilador *sgmusl-gcc* é obviamente restrito à código C, e possivelmente linguagens com suporte a GCC, como Fortran.

Considerando isso, e o aumento do uso da linguagem Python mencionada anteriormente, possibilitar a execução de código Python dentro de enclaves SGX se torna atrativo. Assim, nós potencializamos o compilador *sgxmusl-gcc* para produzir um interpretador Python modificado. Nosso interpretador é compilado com *sgxmusl-gcc* e estendido para interpretar e atestar código Python dentro de enclaves SGX. Dessa forma, essa abordagem aumenta o conjunto de aplicações que podem ser executadas usando SGX assim como o número de desenvolvedores capazes de se beneficiar dessa tecnologia de segurança.

Entretanto, o compilador *sgxmusl-gcc* tem algumas limitações. Uma das maiores é o fato de que a vinculação dinâmica de bibliotecas não é permitido. Todas as bibliotecas de sistema, como *openssl*[5] e *ncurses*[6], junto com módulos nativos de Python requisitados pelas aplicações dos usuários devem ser vinculadas estaticamente antes da construção do Python-SGX. Essas limitações não são necessariamente problemáticas para o interpretador, já que mesmo para aplicações mais complexas, é improvável o uso de uma quantidade extremamente grande de bibliotecas.

Neste trabalho usamos o Python-SGX para interpretar código de alguns dos componentes detalhados na seção a seguir, e os atestamos de forma a garantir que o código enviado pelo desenvolvedor é o mesmo durante todo o processo de sua implantação, ou seja, não houveram modificações de terceiros que possam comprometer o sistema. Isso é feito através de checagens introduzidas no código do interpretador, que comparam o *hash SHA-256* da aplicação com o *hash* provido pelo desenvolvedor durante o processo de atestação do Python-SGX pelo SCONE. Numa visão geral, o Python-SGX é considerado seguro por ser previamente atestado pelo SCONE e essa confiabilidade é estendida para o código posteriormente interpretado por ele.

[3]https://www.musl-libc.org/

[4]https://sconedocs.github.io

[5]https://www.openssl.org/

[6]https://en.wikipedia.org/wiki/Ncurses

6.1.2 Arquitetura e componentes

Medidor Inteligente

Medidores inteligentes são componentes chave em *smart grids*. Tais dispositivos são responsáveis por coletar dados de energia de casas, prédios e outros ambientes, fazendo com que clientes reduzam seus gastos com eletricidade através do monitoramento de seu consumo de energia. Um medidor inteligente pode ler essas medições puras em intervalos de tempo específicos, e comunicar essa informação para um provedor de energia.

No nosso cenário, nós consideramos o componente Medidor Inteligente como sendo um dispositivo capaz de direta ou indiretamente enviar dados para um servidor remoto. Na prática, por conta de restrições de custo ou regulamentais, isso é tipicamente feito indiretamente. Medidores enviam dados para *gateways* e esses os enviam para o sistema que vai processá-los do lado do provedor de energia. Na nossa aplicação, nós consideramos diferentes medidores que podem ser acessados através de redes cabeadas ou sem fio. Por fim, consideramos que dados de consumo de energia são coletados por um componente chamado *Metering Data Collector* (MDC), e então as medições são enviadas para outros sistemas.

Metering Data Collector

Esse componente é responsável por coletar dados de consumo de energia vindos de um medidor inteligente. A aplicação MDC se conecta ao dispositivo através de uma rede TCP/IP e recupera novos dados a cada segundo. Encriptação é necessária já que o componente que irá recebê-los, chamado *Dispatcher*, não é considerado confiável. Tal encriptação é atualmente implementada usando o método AES-CTR.

Existem duas maneiras de criar e gerenciar a chave usada para encriptação: (*i*) a chave é negociada com o SCBR durante o processo de atestação (descrito abaixo); (*ii*) a chave será gerada pelo provedor dos dados e compartilhada com todos os participantes confiáveis do sistema. Como iremos detalhar a seguir, a primeira abordagem introduz uma carga maior sobre o SCBR, reduzindo sua escalabilidade. Em contraste, a segunda opção requer rotação periódica das chaves.

Por outro lado, já que recebe os dados crus vindos do medidor, o MDC precisa ser confiável. Essa confiança pode ser estabelecida através de certificados, por exemplo, ou do uso de ambientes de execução confiável. No nosso caso, consideramos a abordagem (*ii*). Assim, a aplicação MDC é executada pelo interpretador Python-SGX gerado pelo SCONE, como detalhado na seção anterior. Através da execução do MDC dentro de um enclave SGX, podemos validar seu código antes da

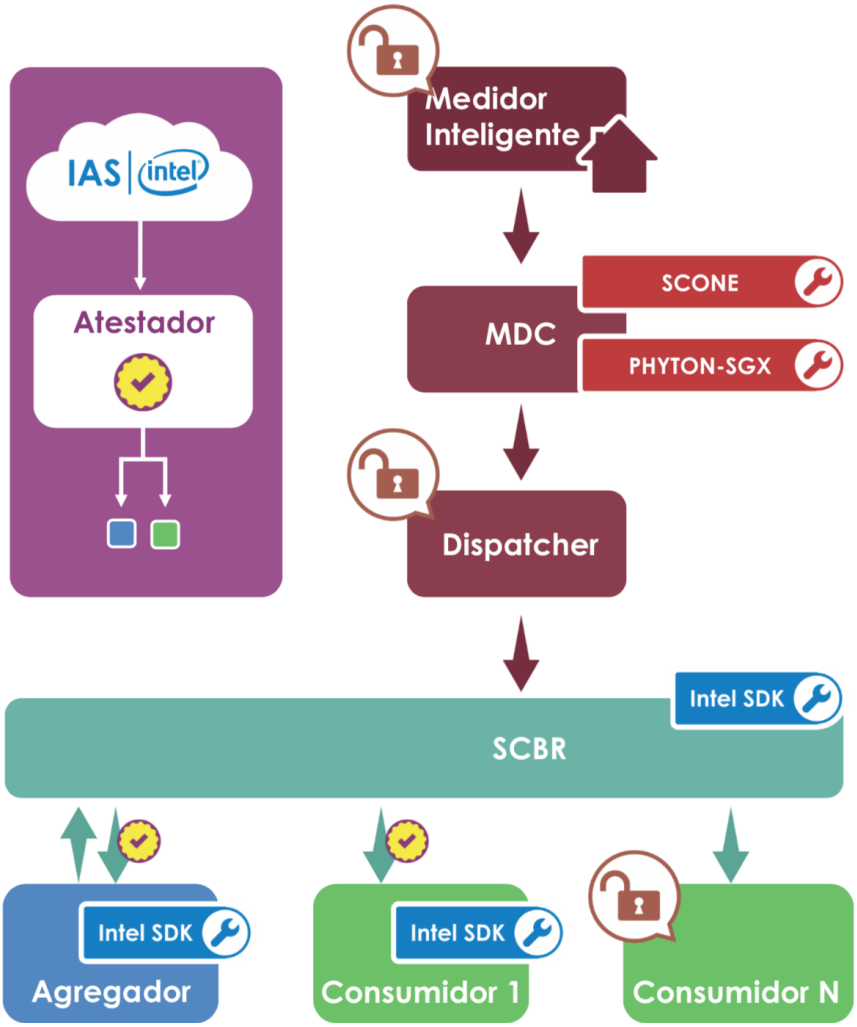


Figura 6.1: Arquitetura da plataforma de disseminação de dados para infraestruturas de medidores inteligentes. (Adaptada de [37].)

execução, garantindo que apenas versões com as assinaturas esperadas serão executadas.

Dispatcher

No nosso cenário, o *Dispatcher* funciona como um *gateway*, passando adiante as medições recebidas da aplicação MDC para o barramento SCBR. Porque o MDC pode ser limitado em funcionalidade, o uso do *dispatcher* possibilita maior flexibilidade na configuração do restante do sistema. Além disso, dado que o *dispatcher* não precisa ser confiável, ele possui mais opções de implementação. Como exemplo, com um *dispatcher* não confiável, é fácil mudar o barramento escolhido se as garantias oferecidas pelo SCBR não forem mais de interesse.

Na nossa implementação específica, a comunicação com o barramento requer conexões via Ze-

roMQ[7]. Por outro lado, o interpretador Python-SGX, utilizado junto ao MDC, possui limitações na importação desse módulo. Como comumente ocorreria na prática, o uso de um nível de indireção, no nosso caso a adição do *Dispatcher*, faz com que o componente MDC seja dissociado do barramento, e consequentemente, do protocolo de comunicação escolhido em sua implementação, o ZeroMQ. Essa dissociação elimina a necessidade de reimplementação do protocolo no barramento, o que poderia possivelmente adicionar mais complexidade à plataforma proposta.

Assim, o *Dispatcher* simplesmente implementa uma camada de comunicação com o barramento via ZeroMQ e se comunica com o MDC através de um *socket* simples. Finalmente, as medições encriptadas recebidas são enviadas para o componente SCBR.

Secure Content-Based Routing

O componente *Secure Content-Based Routing* segue o paradigma de publicação-subscrição [22], no qual remetentes de mensagens, chamados publicadores, não endereçam mensagens explicitamente, mas categorizam as mesmas independente dos destinatários, os inscritos, que irão recebê-las. Do ponto de vista dos inscritos, inscritos expressam interesse em um ou mais tipos de mensagens e recebem as que eles se interessam, independente do seu publicador.

Diferentemente de outras ferramentas do tipo publicação-subscrição, SCBR tem um modo no qual apenas o publicador pode submeter inscrições para suas publicações. Nós usamos esse modo para fazer com que inscritos tenham de, inicialmente, se comunicar com os publicadores. Durante essa comunicação inicial, o publicador vai atestar o candidato consumidor, e se obter sucesso, pode trocar chaves de encriptação.

O barramento SCBR roteia mensagens de maneira segura entre publicadores e inscritos. Suas garantias de segurança e privacidade são consequência do fato de que as decisões de roteamento são tomadas dentro do enclave SGX. No nosso caso de uso, consideramos que a informação sensível vai carregar seu nível de sensibilidade no tópico da publicação.

Dependendo da escolha na abordagem de encriptação do MDC, como discutido anteriormente, existem duas escolhas: (*i*) se a chave de encriptação do MDC é negociada com o SCBR, o barramento desencriptaria o dado e esse dado seria disseminado sem encriptação; (*ii*) se a chave de encriptação é negociada com os componentes confiáveis, o dado permanece encriptado até dentro do enclave do SCBR.

Mensagens enviadas seguem um padrão específico em seus cabeçalhos, que contém o tipo da mensagem, ou nível de privacidade, e seu modo de encriptação, que pode ser *plain-text*. SCBR

[7]http://zeromq.org/

permite que seu nível de segurança seja configurado, sendo possível habilitar o uso de SGX ou não. O barramento usa ferramentas do Intel SDK na sua implementação e, ao habilitar o SGX, executa seu mecanismo de roteamento dentro de um enclave.

Atestador

Na plataforma proposta, para que consumidores recebam dados do SCBR, além de serem registrados no barramento, eles precisam ser considerados confiáveis, e portanto, atestados. Através dessa abordagem, o consumidor sabe como descriptografar as medições criptografadas, e pode ter acesso às informações publicadas.

O processo de atestação segue o protocolo de atestação remota especificado pelo SGX, e explicado na Seção 2.3, fazendo uso do serviço provido pela Intel, o *Intel Attestation Service* (IAS). Dessa forma, o Atestador é responsável por mediar o processo de atestação dos consumidores pelo IAS, e então, o processo de troca de chaves durante atestação.

Agregador

Agregar medições individuais para produzir relatórios de consumo de energia e seus respectivos modelos de cobrança é uma funcionalidade importante, bastante desejada por provedores em um cenário considerando medidores inteligentes de energia. Aqui, o Agregador serve a esse propósito e agrega as medições puras gerando dados de energia agregados. Intervalos de tempo podem variar entre minutos, horas ou meses, mas cada mensagem recebida por um Agregador é uma medição individual, como publicado previamente pelo produtor.

Durante sua inicialização, o Agregador é atestado pelo IAS através do Atestador, e então, capaz de desencriptar mensagens recebidas do barramento. Após a troca de chaves, esse componente também é capaz de encriptar os dados agregados e publicá-los novamente no SCBR para ser consumido pelos consumidores finais. Na nossa implementação, esse código foi escrito usando o Intel SDK.

Consumidores Finais

Podem existir vários consumidores finais, os quais são capazes de se registrar no barramento SCBR e expressar interesse em um certo tipo de mensagem. Para o registro, eles precisam contactar o referente publicador e então serão atestados pelo IAS através do Atestador. Como consequência, poderão receber a chave para desencriptar mensagens publicadas. Esse pedaço de código também foi escrito usando Intel SDK.

Uma alternativa é que o consumidor requisite dados considerados públicos. Nesse caso, o publicador apenas iria registrá-lo sem que seja necessário o processo de atestação.

6.1.3 Fluxo de execução

Como visto na Figura 6.1, o fluxo é iniciado com as medições sendo gravadas pelos Medidores Inteligentes. Esses dados são então enviados para a aplicação MDC, a qual é interpretada pelo Python-SGX. O Python-SGX é atestado pelo SCONE, e em seguida, capaz de atestar o componente MDC, garantindo que o *hash SHA-256* da aplicação é compativel com o *hash* provido durante o processo de atestação do SCONE.

O MDC, então, coleta as medições vindas do Medidor a cada segundo, e as encripta usando o método AES-CTR. A chave usada para encriptação é gerada por um Vetor de Inicialização (VI) e a respectiva chave de desencriptação é entregue durante o processo de atestação do consumidor. O MDC, então, envia as medições encriptadas para o Dispatcher via sockets TCP. Para completar o fluxo de publicação, o Dispatcher se comunica com o barramento SCBR através de conexões ZeroMQ, e publica a medição encriptada de acordo com seu nivel de privacidade.

Na mesma figura podemos ver que, a metade de baixo a partir do componente SCBR, representa os consumidores interessados nas mensagens publicadas pelo Dispatcher. A figura ilustra dois tipos de consumidores, Agregadores e Consumidores Finais, como descrito na seção anterior. No nosso cenário, temos um agregador e um número de consumidores finais. Todos devem primeiro se registrar no barramento através dos produtores, e declarar qual tipo de mensagem estão interessados. Durante o registro, os consumidores são atestados pelo IAS através do componente Atestador. Esse processo é indicado na Figura 6.1 pelos símbolos em amarelo acima das setas conectando os consumidores ao barramento. Quando o processo de atestação é completado, os consumidores recebem a chave capaz de desencriptar as mensagens publicadas. Essa chave compartilhada é enviada encriptada por uma chave simétrica também negociada durante o processo de atestação.

A partir desse ponto, consumidores são capazes de desencriptar mensagens recebidas através do barramento SCBR. Por definição, o agregador recebe as medições encriptadas e as agrega de acordo com intervalos de tempo previamente definidos. Esses intervalos podem variar entre segundos, horas, meses e assim por diante. Após os dados serem agregados, eles podem ser encriptados novamente, e publicados de volta ao SCBR pelo próprio agregador, que nesse cenário também funciona como publicador. Na publicação, o nível de privacidade da informação é definido, e os consumidores finais irão recebê-las de acordo com o tipo de mensagem pela qual demonstraram interesse.

6.1.4 Resultados

Nesta seção discutimos os experimentos que validam a arquitetura de disseminação de dados proposta. Todos os experimentos foram executados em máquinas virtuais SGX na nuvem OpenStack segura, resultado e objeto de estudo desta pesquisa. Assim, validamos também sua implementação, mostrando que a dada nuvem é capaz de executar uma aplicação que faz processamento de dados críticos usando ferramentas de segurança que executam em enclaves SGX. As máquinas virtuais possuíam 4 vcpus, 4 GB de RAM, 60 GB de disco, 32 MB de EPC alocado e uma imagem Ubuntu Linux 16.04 Xenial.

Os medidores inteligentes foram simulados como processos que se conectam ao componente MDC (ver Seção 6.1.2 para mais detalhes). Os experimentos processam até 1 milhão de medições. Tipicamente, duas curvas ou *boxplots* são mostrados: um, descrito como *SGX*, ilustra execuções onde o uso do SGX está habilitado; o outro, descrito como *regular* considera execuções onde o SGX está desabilitado nos componentes confiáveis. Em ambas as execuções, SGX e regular, os componentes usam o método de encriptação AES-CTR para os dados de medição enviados ao barramento. Para cada cenário que considera uma taxa específica de envio de dados, o experimento rodou por 10 segundos. As medições de CPU consideram o uso para os processos do SCBR.

A seguir está o detalhamento dos resultados do experimento considerando sua execução na nuvem segura. Aqui queremos fundamentalmente comprovar que a aplicação consegue ser executada e avaliada no ambiente de execução confiável proposto por este trabalho. Para fins informativos, segue a avaliação dos dados coletados.

A. Latência para uma publicação por segundo

No experimento retratado na Figura 6.2 o sistema estava submetido à uma carga bastante leve. Uma única medição foi publicada a cada segundo. Nesse cenário, é possível ver que a latência para publicar uma única medição, passando pelo barramento SCBR na mesma máquina virtual, não é estatisticamente diferente entre os dois cenários. Também podemos ver que o valor mais alto de latência coletado é parecido para ambas as configurações, o que significa que o pior caso quando SGX é utilizado acontece similarmente em cenários regulares.

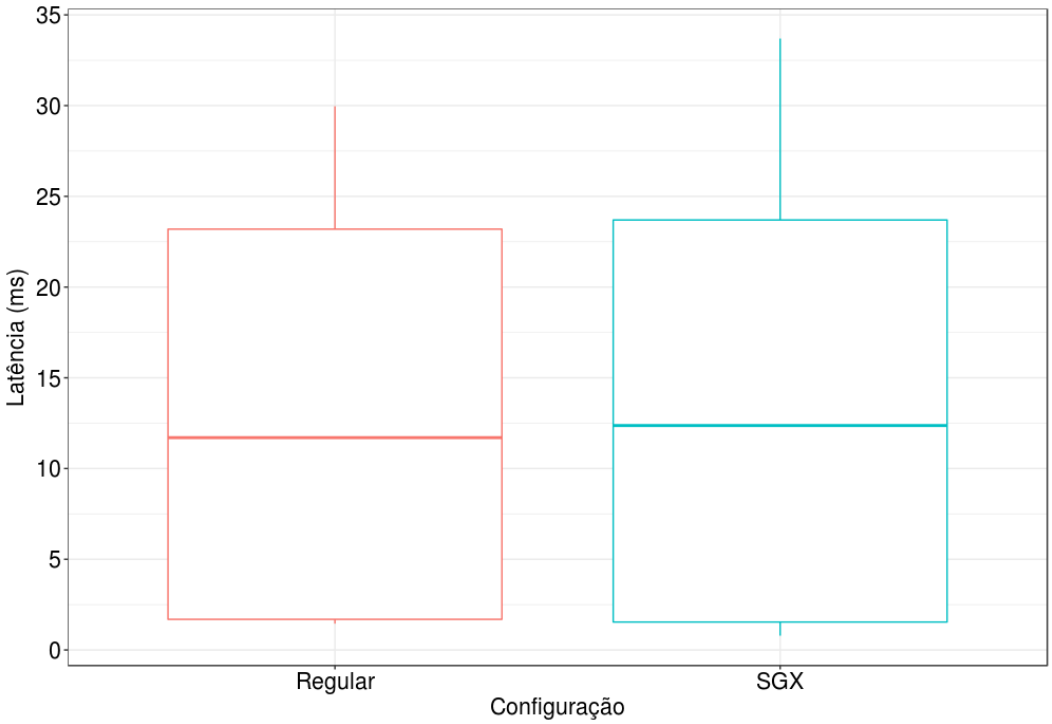


Figura 6.2: Latência para uma publicação isolada.

B. Latência e CPU para 1 milhão de publicações

Em seguida, dado que experimentos considerando uma publicação isolada podem não ser representativos para cenários mais complexos de execução, o comportamento do sistema foi analisado ao ser submetido à uma carga de processamento mais pesada, totalizando 1 milhão de publicações vindas de um só produtor. A Figura 6.3 detalha os 15 primeiros segundos de execução. A latência mostra um comportamento crescente no início da execução, o que significa que o barramento recebe tantas publicações quanto for possível até que suas filas internas estejam cheias, e, a partir desse momento, vemos apenas uma pequena variação na latência. Isso pode ser explicado como um tipo de mecanismo de *back-pressure*, o que, no nosso caso, significa que o barramento faz com que o dispositivo enviando dados aguarde um pouco para executar novos envios até que o gargalo tenha sido eliminado. Podemos ver também que tal condição acontece em ambas as configurações, apenas apresentando uma latência um pouco maior para cenários SGX. Ainda considerando o mesmo cenário de processamento de 1 milhão de publicações, a Figura 6.4 mostra que o uso de CPU mantém uma média regular para ambos os cenários, levando em torno de 60s para executar o total de medições.

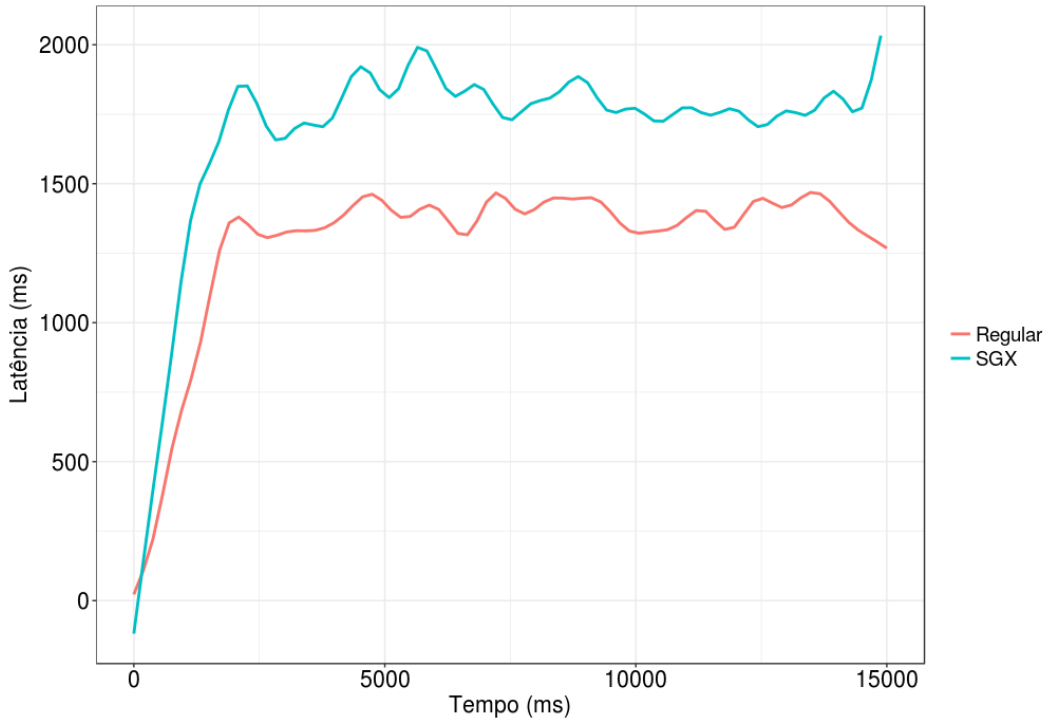


Figura 6.3: Latência para um 1 milhão de publicações.

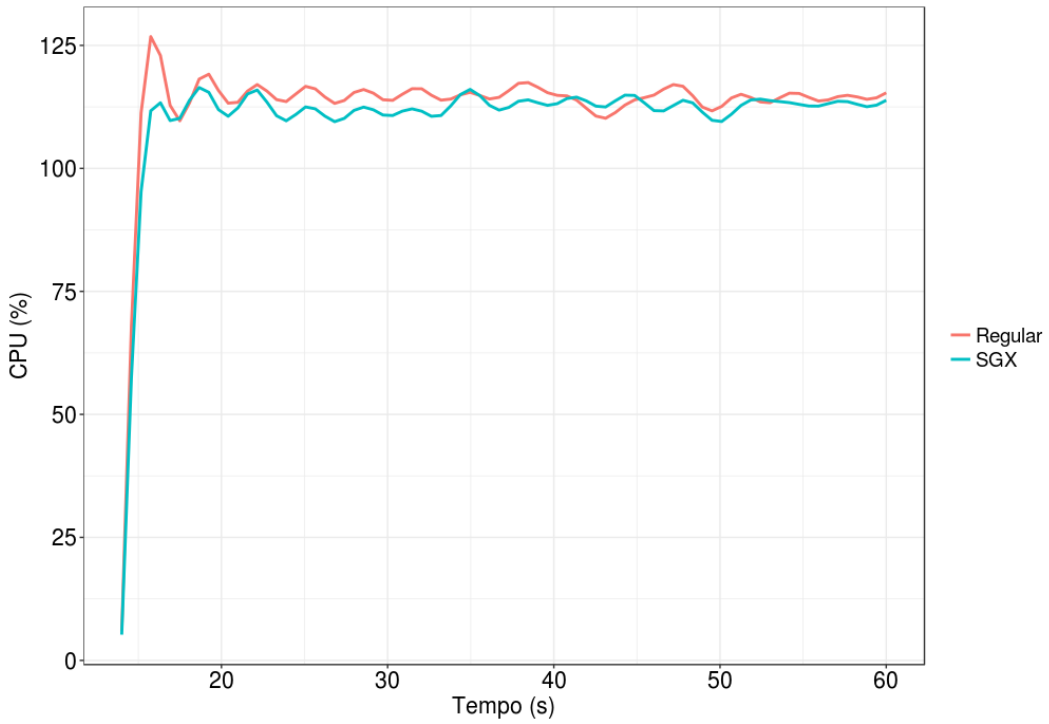


Figura 6.4: Uso de CPU para 1 milhão de publicações.

C. Latência e CPU para taxas específicas de publicação

Experimentos considerando uma variação de taxas de publicação podem ser vistas nas Figuras 6.5 e 6.6. As taxas consideradas foram 1000, 2500, 5000, 10000, 15000 e 20000 medições por segundo, sendo a última o número garantido de publicações processadas em um segundo sem atrasos acumulativos. Na Figura 6.5, nós vemos que apenas a partir de 15000 medições/s a latência média para ambas as configurações começam a diferir consideravelmente. Para taxas menores diferenças quase não são notadas.

A Figura 6.6 depura, então, o consumo de CPU do barramento de mensagens durante o experimento com variados cenários. Para ambas as configurações, o uso aumenta conforme a taxa de publicação. A partir dos resultado podemos ver que não existem grandes diferenças no consumo de CPU quando usando um cenário regular e outro com SGX.

O último experimento detalha o comportamento do sistema na presença de rajadas de eventos periódicas. Cada sub-figura da Figura 6.7 depura um passo na progressão de execução de experimentos com 1000 medições/s, até 20000 medições/s. Em cada passo podemos ver como o atraso real se afasta do ideal. Por exemplo, a curva ideal é demonstrada com a cor vermelha e uma linha reta. Para algumas taxas de publicação, a latência real se afasta desse comportamento perfeito.

A Figura 6.7 mostra as execuções num espaço de 2 segundo. Esse valor foi escolhido pois demonstra dois ciclos, indicando um comportamento recorrente, enquanto não deixa a figura difícil de ler. Podemos ver que para taxas acima de 5000 medições/s, não existe desvio visível de latência, as linhas quase se sobrepõem. A partir de 10000 medições/s, nós podemos identificar desvios mas claros para ambas as configurações, com e sem uso de SGX.

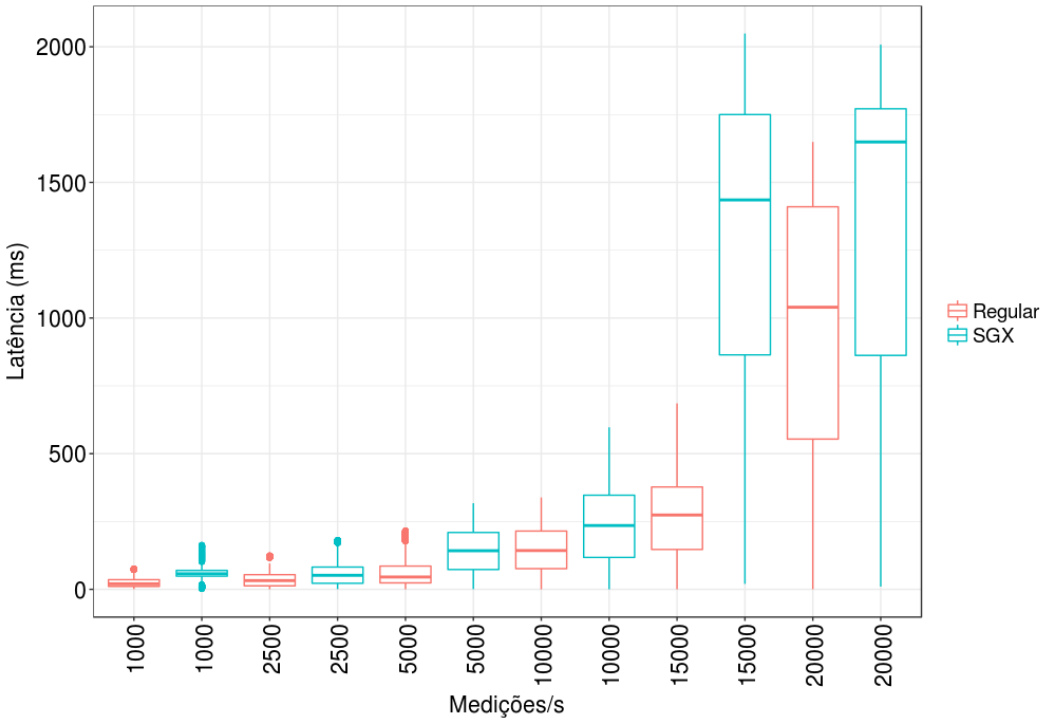


Figura 6.5: Latência considerando taxas específicas de publicação.

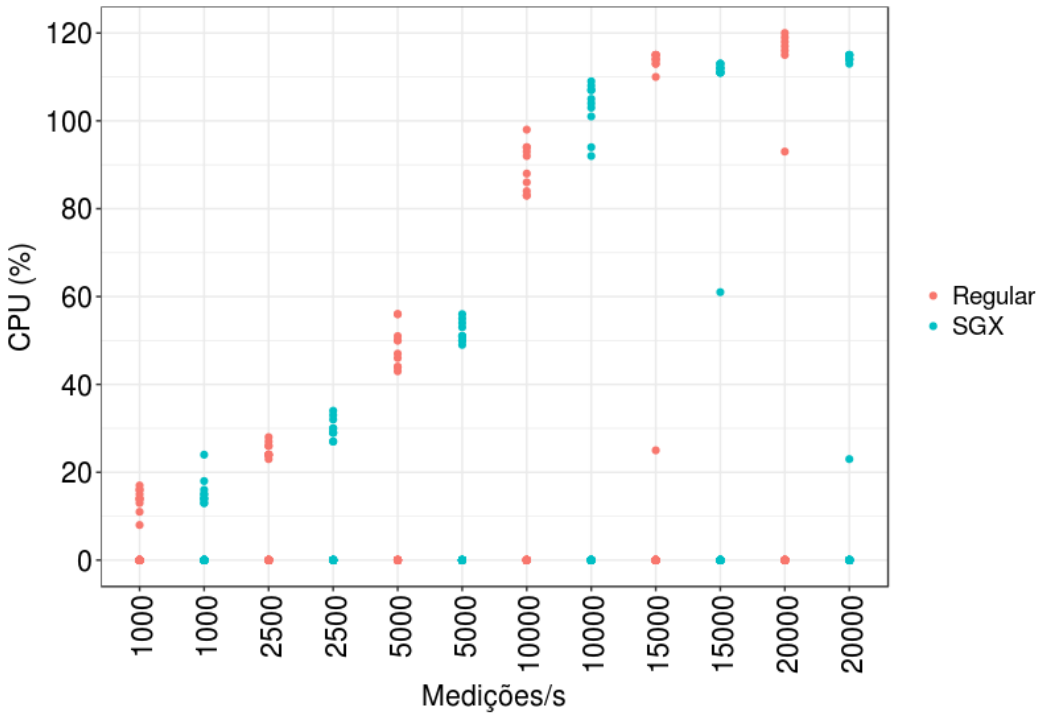


Figura 6.6: CPU considerando taxas específicas de publicação.

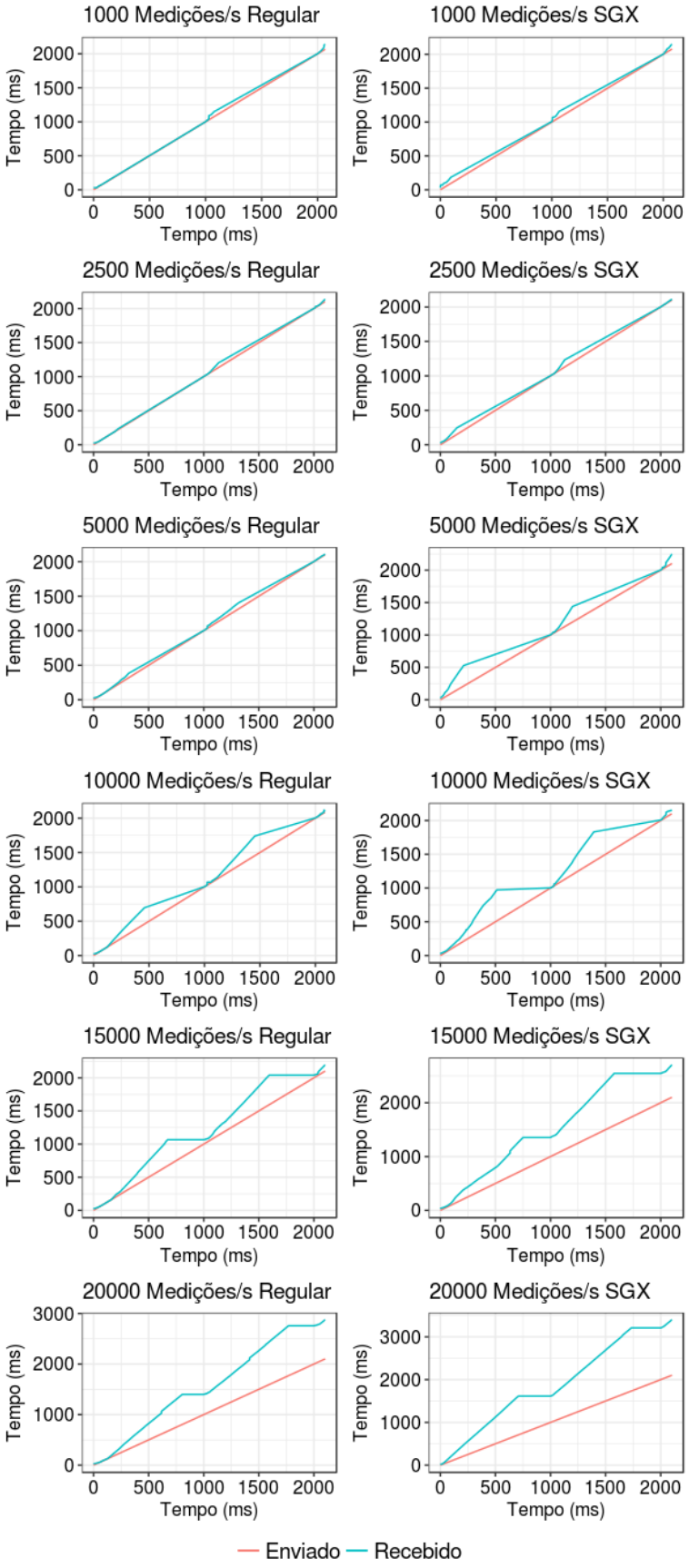


Figura 6.7: Latência detalhada considerando uma variação de taxas específicas de publicação.

Capítulo 7

Conclusões

7.1 Considerações Finais

Neste trabalho foram apresentadas estratégias para o suporte à ambientes de execução confiável em sistemas de computação na nuvem. A abordagem utilizada integra a tecnologia de segurança em hardware Intel SGX e a nuvem OpenStack, plataforma de código aberto amplamente conhecida e utilizada por grandes empresas. Dessa forma, recursos são provisionados com acesso ao SGX de suas máquinas hospedeiras, sejam tais recursos máquinas virtuais ou contêineres. O uso de um *kernel* modificado, chamado *kvm-sgx*, possibilitou a virtualização do SGX para máquinas virtuais. Assim, nossa abordagem é menos vulnerável à ataques como o exposto por Schwarz et al. [40], capaz de atingir diferentes contêineres em uma mesma máquina hospedeira dado o compartilhamento do mesmo *driver* SGX entre todos os contêineres hospedados nela. Isso não aconteceria em máquinas virtuais graças ao isolamento provido pelo uso de sistemas operacionais independentes, o que possibilita a instalação de um *driver* SGX para cada instância.

Considerando isso, propomos uma nova estratégia no processo de provisionamento e escalonamento de recursos numa nuvem OpenStack segura, que considera aspectos essenciais para o SGX, como a quantidade de EPC sendo utilizado. Para isso, adaptamos o serviço de provisionamento de instâncias no OpenStack, o Nova, de forma a viabilizar o cálculo do consumo de EPC em um nó de computação bem como o provisionamento de instâncias baseado nesse recurso. Foi criado um novo filtro de escalonamento, chamado `SgxEpcFilter`, que avalia os nós de computação associados à nuvem quanto à sua capacidade de oferecer recursos SGX à uma nova instância. Se o nó possui EPC suficiente para prover a quantidade sendo requisitada, ele passa no filtro. Caso contrário, não é considerado um nó apto para provisionamento. Aqui, um ponto essencial sobre essa abordagem

é a sua implementação seguir o padrão dos demais filtros de escalonamento disponibilizados pelo serviço Nova, como os baseados em consumo de memória e CPU. Dessa forma, a integração foi feita de forma compatível com a arquitetura existente da nuvem, o que favorece a aceitação tanto por parte da comunidade como dos usuários da mesma.

Ainda na frente de provisionamento e escalonamento, de forma a ser possível acessar o SGX das máquinas hospedeiras, foi preciso a instalação do *kernel* especial *kvm-sgx* como citado anteriormente. Cada nó de computação com capacidade SGX teve esse *kernel* instalado e comprovadamente funcional em conjunto com os demais serviços da nuvem. Por fim, o *driver* Libvirt associado ao hipervisor KVM no OpenStack teve sua estrutura modificada para que o XML das máquinas virtuais sendo criadas passassem a conter parâmetros referentes ao SGX, como o valor de EPC estaticamente alocado. Dessa forma, usuários passam na configuração da máquina o quanto de EPC é desejado, e essa informação chega até o *driver*, que a adiciona ao referido XML.

Além disso, a abordagem ainda cobre a computação do consumo de EPC utilizado pelas instâncias em um dado projeto, que já é oferecida pelo Nova para outros recursos como disco e memória, e que agora conta também com o EPC. Dessa forma, provedores podem monitorar seu uso para poder planejar melhor o oferecimento de tais recursos SGX e produzir modelos de cobrança em concordância com o consumo de EPC.

Quando falamos sobre contêineres, consideramos que eles podem se oferecidos na forma de contêineres LXD vinculados ao dispositivo *isgx* na máquina hospedeira, ou através de contêineres Docker gerenciados por orquestradores de contêineres provisionados sob demanda através do serviço Magnum. Para o primeiro caso, instalamos um *driver* SGX no nó de computação associado ao hipervisor de contêineres LXD, e, durante inicialização, vinculamos os contêineres sendo criados ao dispositivo SGX no nó hospedeiro. O processo é transparente do ponto de vista dos usuários, que requisitam uma instância normalmente através do serviço Nova e recebem um contêiner LXD SGX apenas ao escolher uma imagem e *flavor* compatível. Para o serviço Magnum, a criação de um *cluster* de VMs SGX já é suficiente para permitir que aplicações sejam implantadas em contêineres com acesso ao SGX de suas máquinas hospedeiras. Um detalhe importante, e contribuição desse trabalho, é a viabilidade de instalação do *driver* SGX em imagens do tipo Fedora Atomic, requisito para orquestradores de contêineres como Kubernetes e Docker Swarm, e não suportadas por padrão pelo *driver* SGX oferecido pela Intel.

A avaliação de desempenho para os cenários que utilizavam *kvm-sgx*, tanto para nós de computação como para máquinas virtuais, bem também os ambientes LXD, mostrou que as diferenças entre os tempos de execução e taxa de transferência dos *benchmarks* executados nos experimentos para cada

cenário foram insignificantes. O desempenho tão similar é um bom indicador de que, nesse aspecto, o impacto das mudanças feitas para integração com SGX não afetam fortemente a infraestrutura da nuvem e seus recursos.

Por fim, validamos esta nuvem com uma aplicação que exige processamento de dados sensíveis, no nosso contexto, um sistema que considera medidores inteligentes de energia e a disseminação desses dados de consumo com garantias de integridade e confidencialidade através do uso de ferramentas executando em enclaves SGX [37]. Tal aplicação foi implantada com sucesso na nuvem segura resultado deste trabalho, e experimentos foram realizados para avaliação de seu comportamento quanto à latência no atendimento às requisições e consumo de CPU ao serem submetidas à cargas mais altas de trabalho.

7.2 Trabalhos Futuros

A integração da tecnologia de segurança em hardware Intel SGX com a nuvem OpenStack pode ser estendida para outros serviços, e, fazer uso de benefícios do SGX como mecanismos de atestação local e remoto, citados anteriormente. Por exemplo, o serviço Keystone de autenticação de usuários pode ser aliado a tais mecanismos de atestação em ambientes de nuvens federadas. Outros serviços como o Swift para armazenamento de dados e objetos pode se utilizar do SGX para proteger tais dados de vazamentos e modificações indesejadas por terceiros não confiáveis.

Considerando os serviços já utilizados em nossa solução, Magnum e Nova, uma otimização é a geração de imagens já preparadas com um *driver* SGX. Hoje isso é inviável por limitações do serviço de criação de imagens para o OpenStack, o *Disk Image Builder* (DIB), mas é uma possibilidade a ser explorada, já que resultaria no provisionamento de ponta a ponta da instância segura, entregue pronta para receber aplicações SGX.

Outra melhoria é o armazenamento do consumo de EPC no banco de dados do OpenStack junto ao objeto *ComputeNode*, hoje calculado através do objeto *HostState* a cada vez que uma instância é requisitada. Dessa forma, considerando nuvens maiores, com mais nós de computação e máquinas virtuais implantadas, buscar o consumo de EPC no banco tende a ser menos custoso do que calculá-lo a cada requisição.

Bibliografia

[1] Magnum service. http://docs.openstack.org/developer/magnum/userguide.html. [Online; Último acesso: 04 de janeiro, 2018].

[2] Openstack user survey november 2017. https://www.openstack.org/assets/survey/OpenStack-User-Survey-Nov17.pdf. [Online; Último acesso: 22 de janeiro, 2018].

[3] Github kvm-sgx. https://github.com/intel/kvm-sgx/wiki, 2018. [Online; Último acesso: 04 de janeiro, 2018].

[4] Github qemu-sgx. https://github.com/intel/qemu-sgx/wiki, 2018. [Online; Último acesso: 04 de janeiro, 2018].

[5] Openstack. https://www.openstack.org/, 2018. [Online; Último acesso: 22 de janeiro, 2018].

[6] Ittai Anati, Shay Gueron, Simon Johnson, and Vincent Scarlata. Innovative technology for cpu based attestation and sealing. In *HASP '13: Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, volume 13. ACM, 2013.

[7] ARM Limited. Security technology building a secure system using trustzone technology (white paper). 2009.

[8] K. Carrie Armel, Abhay Gupta, Gireesh Shrimali, and Adrian Albert. Is disaggregation the holy grail of energy efficiency? the case of electricity. *Energy Policy*, 52:213 – 234, 2013. Special Section: Transition Pathways to a Low Carbon Economy.

[9] Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, Andre Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Dan O'Keeffe, Mark L. Stillwell, David Goltzsche, David Eyers, Rüdiger Kapitza, Peter Pietzuch, and Christof Fetzer. Scone: Secure linux containers with intel sgx. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, pages 689–703, Berkeley, CA, USA, 2016. USENIX Association.

[10] Manuel Barbosa, Bernardo Portela, Guillaume Scerri, and Bogdan Warinschi. Foundations of hardware-based attested computation and application to sgx. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 245–260. IEEE, 2016.

[11] Nipun Batra, Jack Kelly, Oliver Parson, Haimonti Dutta, William Knottenbelt, Alex Rogers, Amarjeet Singh, and Mani Srivastava. Nilmtk: An open source toolkit for non-intrusive load monitoring. In *Proceedings of the 5th International Conference on Future Energy Systems*, e-Energy '14, pages 265–276, New York, NY, USA, 2014. ACM.

[12] Stefan Berger, Ramón Cáceres, Kenneth A. Goldman, Ronald Perez, Reiner Sailer, and Leendert van Doorn. vtpm: Virtualizing the trusted platform module. In *Proceedings of the 15th Conference on USENIX Security Symposium - Volume 15*, USENIX-SS'06, Berkeley, CA, USA, 2006. USENIX Association.

[13] Stefan Berger, Kenneth Goldman, Dimitrios Pendarakis, David Safford, Enriquillo Valdez, and Mimi Zohar. Scalable attestation: A step toward secure and trusted clouds. In *Proceedings of the 2015 IEEE International Conference on Cloud Engineering*, IC2E '15, pages 185–194, Washington, DC, USA, 2015. IEEE Computer Society.

[14] Ideler H. Nürnberger S. Sadeghi AR. Bleikertz S., Bugiel S. Client-controlled cryptography-as-a-service in the cloud. In *Applied Cryptography and Network Security*, ACNS 2013.

[15] Andrew Brown and Jeffrey S. Chase. Trusted platform-as-a-service: A foundation for trustworthy cloud-hosted applications. In *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop*, CCSW '11, pages 15–20, New York, NY, USA, 2011. ACM.

[16] Shakeel Butt, H. Andrés Lagar-Cavilla, Abhinav Srivastava, and Vinod Ganapathy. Self-service cloud computing. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 253–264, New York, NY, USA, 2012. ACM.

[17] Cloud Native Computing Foundation. Cloud native technologies are scaling production applications. https://www.cncf.io/blog/2017/12/06/cloud-native-technologies-scaling-production-applications/, 2017. [Online; Publicado: 06 de dezembro, 2017; Último acesso: 22 de janeiro, 2018].

[18] Intel Corporation. Creating trust in the cloud. http://www.intel.com.tw/content/dam/www/public/us/en/documen papers/creating-trust-in-cloud-ubuntu-intel-white-paper.pdf, 2013. [Online; accessed 25-November-2016].

[19] Intel Corporation. Intel software guard extensions. Cryptology ePrint Archive, Report 2016/086, 2015. https://software.intel.com/sites/default/.

[20] Victor Costan and Srinivas Devadas. Intel sgx explained. Cryptology ePrint Archive, Report 2016/086, 2016. http://eprint.iacr.org/2016/086.

[21] Heidi Daitch. 2017 data breaches. https://www.identityforce.com/blog/2017-data-breaches, 2018. [Online; Último acesso: 22 de janeiro, 2018].

[22] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, June 2003.

[23] Diogo A. Fernandes, Liliana F. Soares, João V. Gomes, Mário M. Freire, and Pedro R. Inácio. Security issues in cloud environments: A survey. *Int. J. Inf. Secur.*, 13:113–170, 2014.

[24] GitHub Inc. The state of the octoverse 2017. https://octoverse.github.com/. [Online; Último acesso: 22 de janeiro, 2018].

[25] Ulrich Greveler, Peter Glösekötterz, Benjamin Justusy, and Dennis Loehr. Multimedia content identification through smart meter power usage profiles. In *Proceedings of the International Conference on Information and Knowledge Engineering (IKE)*, page 1. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2012.

[26] Felicitas Hetzelt and Robert Buhren. Security analysis of encrypted virtual machines. In *Proceedings of the 13th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, pages 129–142. ACM, 2017.

[27] Matthew Hoekstra, Reshma Lal, Pradeep Pappachan, Vinay Phegade, and Juan Del Cuvillo. Using innovative instructions to create trustworthy software solutions. In *HASP@ ISCA*, page 11, 2013.

[28] Shohreh Hosseinzadeh, Samuel Laurén, and Ville Leppänen. Security in container-based virtualization through vtpm. In *Proceedings of the 9th International Conference on Utility and Cloud Computing*, UCC '16, pages 214–219, New York, NY, USA, 2016. ACM.

[29] David Kaplan, Jeremy Powell, and Tom Woller. Amd memory encryption. *White paper, Apr*, 2016.

[30] Alexey Kopytov. Sysbench manual. http://imysql.com/wp-content/uploads/2014/10/sysbench-manual.pdf, 2009. [Online; Último acesso: 04 de janeiro, 2018].

[31] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R Savagaonkar. Innovative instructions and software model for isolated execution. In *HASP@ ISCA*, page 10, 2013.

[32] Lucas Mearian. The state of the octoverse 2017. Computerworld - https://www.computerworld.com/article/2960642/cloud-storage/cerns-data-stores-soar-to-530m-gigabytes.html. [Online; Publicado: 14 de agosto, 2015; Último acesso: 22 de janeiro, 2018].

[33] Rafael Pires, Marcelo Pasin, Pascal Felber, and Christof Fetzer. Secure content-based routing using intel software guard extensions. In *Proceedings of the 17th International Middleware Conference*, Middleware '16, pages 10:1–10:10, New York, NY, USA, 2016. ACM.

[34] RightScale. Rightscale 2017: State of the cloud report. https://www.rightscale.com/lp/2017-state-of-the-cloud-report, 2017. [Online; Último acesso: 22 de janeiro, 2018].

[35] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, pages 199–212, New York, NY, USA, 2009. ACM.

[36] M. Sabt, M. Achemlal, and A. Bouabdallah. Trusted execution environment: What it is, and what it is not. In *2015 IEEE Trustcom/BigDataSE/ISPA*, pages 57–64, Aug 2015.

[37] Lilia Sampaio, Fábio Silva, Amanda Souza, Andrey Brito, and Pascal Felber. Secure and privacy-aware data dissemination for cloud-based applications. In *Proceedings of the10th International Conference on Utility and Cloud Computing*, UCC '17, pages 47–56. ACM, 2017.

[38] Nuno Santos, Krishna P. Gummadi, and Rodrigo Rodrigues. Towards trusted cloud computing. In *Proceedings of the 2009 Conference on Hot Topics in Cloud Computing*, HotCloud'09, Berkeley, CA, USA, 2009. USENIX Association.

[39] J. Schiffman, Y. Sun, H. Vijayakumar, and T. Jaeger. Cloud verifier: Verifiable auditing service for iaas clouds. In *2013 IEEE Ninth World Congress on Services*, pages 239–246, June 2013.

[40] Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, and Stefan Mangard. Malware guard extension: Using SGX to conceal cache attacks. *CoRR*, abs/1702.08719, 2017.

[41] Prateek Sharma, Lucas Chaufournier, Prashant Shenoy, and Y. C. Tay. Containers and virtual machines at scale: A comparative study. In *Proceedings of the 17th International Middleware Conference*, Middleware '16, pages 1:1–1:13, New York, NY, USA, 2016. ACM.

[42] Abhinav Srivastava and Vinod Ganapathy. Towards a richer model of cloud app markets. In *Proceedings of the 2012 ACM Workshop on Cloud Computing Security Workshop*, CCSW '12, pages 25–30, New York, NY, USA, 2012. ACM.

[43] S. Subashini and V. Kavitha. Review: A survey on security issues in service delivery models of cloud computing. *J. Netw. Comput. Appl.*, 34(1):1–11, January 2011.

[44] Wai Kit Sze, Abhinav Srivastava, and R. Sekar. Hardening openstack cloud platforms against compute node compromises. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '16, pages 341–352, New York, NY, USA, 2016. ACM.

[45] Jakub Szefer, Eric Keller, Ruby B. Lee, and Jennifer Rexford. Eliminating the hypervisor attack surface for a more secure cloud. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, pages 401–412, New York, NY, USA, 2011. ACM.

[46] A. Tosatto, P. Ruiu, and A. Attanasio. Container-based orchestration in cloud: State of the art and challenges. In *2015 Ninth International Conference on Complex, Intelligent, and Software Intensive Systems*, pages 70–75, July 2015.

[47] Yinqian Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Cross-vm side channels and their use to extract private keys. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 305–316, New York, NY, USA, 2012. ACM.

Apêndice A

Portando a nuvem OpenStack para criação de recursos com suporte à SGX

A.1 Máquinas virtuais SGX

A.1.1 Provisionamento

Fundamentalmente, para o provisionamento de instâncias SGX é preciso que o hipervisor da nuvem seja capaz de criar recursos que conheçam o SGX da máquina hospedeira. Considerando o nosso caso, onde *kvm-sgx* e *qemu-sgx* se unem para criação de tais recursos, esse processo envolve modificar o XML usado pelo Libvirt na criação de uma máquina virtual para conter os parâmetros necessários à alocação de EPC.

O Código Fonte A.1 mostra trechos do método `_get_guest_xml()` onde as modificações foram inseridas. Primeiro a informação da quantidade de EPC a ser alocada é recuperada do *flavor*. Logo após os parâmetro necessários para criação da máquina virtual SGX são adicionados à *string* que representa o XML, são eles o `-sgx epc=$VALOR` e `-cpu host`.

Já o Código Fonte A.2 mostra as configurações necessárias no arquivo `nova.conf` para que o serviço Nova e o *kvm-sgx* possam coexistir no ambiente OpenStack. O tipo de virtualização é modificado para `kvm`, e a ferramenta gerenciadora do acesso via console à máquina virtual é configurada para `vnc`, desabilitando a padrão `spice`. As tags no arquivo de configuração devem parecer com o apresentado no Código A.2.

```
def _get_guest_xml(self, context, instance, network_info, disk_info,
                   image_meta, rescue=None, block_device_info=None):
    {...}
    #sgx
    extra_specs = instance.flavor.extra_specs
    if extra_specs:
        sgx_epc = extra_specs.get('sgx:epc_size')
        if sgx_epc:
            qemu_ns = " xmlns:qemu='http://libvirt.org/schemas/domain/
                qemu/1.0'"
            qemu_tags = '''
                <qemu:commandline>
                <qemu:arg value='-sgx'/>
                <qemu:arg value='epc=''' + sgx_epc + '''M'/>
                <qemu:arg value='-cpu'/>
                <qemu:arg value='host'/>
                </qemu:commandline>
            '''
            xml = xml[:18] + qemu_ns + xml[18:]
            xml = xml[:-10] + qemu_tags + xml[-10:]
    {...}

    return xml
```

Código Fonte A.1: Código do *driver* Libvirt no OpenStack para alocação de EPC.

```
[libvirt]
virt_type = kvm

[spice]
agent_enabled = False
enabled = False

[vnc]
enabled = True
```

Código Fonte A.2: Arquivo de configuração do Nova com as opções para o *kvm-sgx*.

A.1.2 Escalonamento

Assim como para outros casos no OpenStack, para criação do novo filtro de escalonamento que considera o uso de EPC num nó de computação SGX, chamado aqui de `SgxEpcFilter`, é necessária a implementação do método `host_passes()`. O Código Fonte A.3 mostra as condições definidas para ele.

Primeiro é recuperado do *flavor* quanto de EPC está sendo requisitado. Depois é calculada a quantidade de EPC livre no hospedeiro considerando o limiar de $90\ MB$ como sendo o máximo que um nó pode oferecer. O EPC em uso é recuperado do objeto `host_state` que carrega informações sobre o uso de variados recursos em um nó. Por fim, se a quantidade de EPC livre é maior que o quanto está sendo requisitado, o métoda retorna `True`, caso contrário, `False`.

```
1  class SgxEpcFilter(filters.BaseHostFilter):
2
3      run_filter_once_per_request = True
4
5      def host_passes(self, host_state, spec_obj):
6          flavor_obj = spec_obj.flavor
7          extra_specs = (flavor_obj.extra_specs
8                         if 'extra_specs' in flavor_obj else {})
9          epc_requested = extra_specs.get('sgx:epc_size')
10         if epc_requested:
11             free_epc = 90 - host_state.epc_used
12
13             if free_epc > int(epc_requested):
14                 return True
15         return False
```

Código Fonte A.3: Filtro de escalonamento que seleciona nós baseado no consumo de EPC.

O Código Fonte A.4 mostra como é armazenado no objeto `host_manager` o consumo de EPC em um nó. No método `__init__()` que inicia o objeto é criada uma variável chamada `epc_used`, que guarda a quantidade de EPC em uso no nó. Em seguida, no método `_locked_consume_from_request()`, a variável iniciada anteriormente é atualizada com o valor de EPC sendo alocado para a instância requisitada.

Por fim, o Código Fonte A.5 mostra como deve ficar a configuração no arquivo `nova.conf` para habilitar o novo filtro.

```
class HostState(object):

    def __init__(self, host, node, cell_uuid):
        {...}
        self.total_usable_ram_mb = 0
        self.total_usable_disk_gb = 0
        self.disk_mb_used = 0
        self.free_ram_mb = 0
        self.free_disk_mb = 0
        self.vcpus_total = 0
        self.vcpus_used = 0
        self.pci_stats = None
        self.numa_topology = None

        #sgx
        self.epc_used = 0
        {...}

    def _locked_consume_from_request(self, spec_obj):
        {...}
        self.free_ram_mb -= ram_mb
        self.free_disk_mb -= disk_mb
        self.vcpus_used += vcpus

        #sgx
        epc_to_be_allocated = spec_obj.flavor.extra_specs.get('sgx:
            epc_size')
        if epc_to_be_allocated:
            self.epc_used += int(epc_to_be_allocated)
        {...}
```

Código Fonte A.4: Código do objeto *HostManager* para atualizar o consumo de EPC em um nó.

```
[filter_scheduler]
available_filters = nova.scheduler.filters.all_filters
available_filters = nova.scheduler.filters.sgx_epc_filter.SgxEpcFilter

enabled_filters = AggregateInstanceExtraSpecsFilter,SgxEpcFilter,
    RetryFilter,AvailabilityZoneFilter,RamFilter,ComputeFilter,
    ComputeCapabilitiesFilter,ImagePropertiesFilter,
    ServerGroupAntiAffinityFilter,ServerGroupAffinityFilter,
    AggregateCoreFilter,AggregateDiskFilter
```

Código Fonte A.5: Arquivo de configuração do Nova para habilitar o filtro *SgxEpcFilter*.

A.1.3 Contabilidade

Para contabilidade do uso de EPC é necessário calcular a quantidade de horas pela quantidade de recurso utilizada. Dentro do serviço Nova já existe um mecanismo que faz essa computação para outros tipos de recurso. Dessa forma, o Código Fonte A.6 mostra o método `_tenant_usages_for_period()`, que calcula o uso de recursos como memória, disco e CPU através dos valores nos respectivos *flavors* das instâncias.

Adicionamos então o EPC como sendo um dos consumos computados, e seguimos o mesmo padrão obtendo os valores a partir do dado *flavor*. O cálculo total é feito multiplicando as horas de uso pela quantidade de recurso utilizada.

Para que o administrador tenha acesso a esses dados, modificamos o serviço `nova-client` para exibir junto ao resultado do comando `nova usage-list` também o consumo de EPC. Esse mesmo comando já existe no cliente, e mostra os resultados para os recursos de memória, disco e CPU de um projeto. O comando permite exibir a utilização em intervalos de tempo definidos ou por toda a duração do sistema.

O Código Fonte A.7 mostra o método `do_usage_list()`, que adiciona uma coluna, *EPC MB-Hours*, ao resultado do comando `nova usage-list` contendo o consumo de EPC das instâncias em um projeto. Os valores para preenchimento da tabela vem do serviço Nova, através do método `_tenant_usages_for_period()` descrito anteriormente.

```
def _tenant_usages_for_period(self, context, period_start, period_stop,
                              tenant_id=None, detailed=True, limit=None,
                              marker=None):
    {...}
    for instance in instances:
        try:
            info['memory_mb'] = instance.flavor.memory_mb
            info['local_gb'] = (instance.flavor.root_gb +
                                instance.flavor.ephemeral_gb)
            info['vcpus'] = instance.flavor.vcpus

            #sgx
            epc_size = instance.flavor.extra_specs.get('sgx:epc_size')
            if epc_size:
                info['epc_mb'] = int(epc_size)
            else:
                info['epc_mb'] = 0
    {...}
    if info['tenant_id'] not in rval:
        summary = {}
        {...}
        summary['total_vcpus_usage'] = 0

        #sgx
        summary['total_epc_usage'] = 0
    {...}
    summary['total_vcpus_usage'] += info['vcpus'] * info['hours']

    #sgx
    summary['total_epc_usage'] += info['epc_mb'] * info['hours']

    {...}
    return list(rval.values()), all_server_usages
```

Código Fonte A.6: Código Nova que computa o uso de EPC.

```
def do_usage_list(cs, args):
    dateformat = "%Y-%m-%d"

    #sgx
    rows = ["Tenant ID", "Servers", "RAM MB-Hours", "CPU Hours",
            "Disk GB-Hours", "EPC MB-Hours"]
    {...}
    def simplify_usage(u):
        simplerows = [x.lower().replace(" ", "_") for x in rows]

        setattr(u, simplerows[0], u.tenant_id)
        setattr(u, simplerows[1], "%d" % len(u.server_usages))
        setattr(u, simplerows[2], "%.2f" % u.total_memory_mb_usage)
        setattr(u, simplerows[3], "%.2f" % u.total_vcpus_usage)
        setattr(u, simplerows[4], "%.2f" % u.total_local_gb_usage)

        #sgx
        setattr(u, simplerows[5], "%.2f" % u.total_epc_usage)

    {...}
    utils.print_list(usage_list, rows)
```

Código Fonte A.7: Código que exibe o consumo de EPC para instâncias em projetos.

A.2 Contêineres LXD SGX

Para criar contêineres LXD no Nova que se beneficiem da segurança provida pelo SGX é preciso que eles sejam capazes de acessar o dispositivo *isgx* da máquina na qual estão hospedados. Em um ambiente fora da nuvem, isso é feito através dos comandos listados no Código Fonte A.8, executados após a criação dos recursos.

Em um ambiente OpenStack, o código do *driver* LXD precisa ser modificado para adicionar esses comandos de forma a fazer com os contêineres sejam criados já com acesso ao dispositivo *isgx* da máquina hospedeira. O Código Fonte A.8 mostra o método `spawn()`, responsável por direcionar o fluxo de criação de um dado recurso LXD na nuvem, e lá adicionamos os comandos.

Nesse método, os passos para criação do contêiner incluem recuperar a imagem armazenada a

partir do serviço Glance, conectar as redes necessárias para inicializar o recurso, criar um *profile* junto ao LXD da máquina e por fim instanciar e iniciar o contêiner. Ao fim do processo, habilitamos o dispositivo *isgx* e modificamos a sua permissão de acesso, de forma a provisionar um contêiner LXD que consegue fazer uso do SGX disponível em sua máquina hospedeira.

```
1 $ lxc config device add sgx-test /dev/isgx unix-char path=/dev/isgx
2 $ lxc config device set sgx-test /dev/isgx  mode 666
```

Código Fonte A.8: Comandos para habilitar o uso do dispositivo *isgx* em um contêiner LXD.

```
1 def spawn(self, context, instance, image_meta, injected_files,
2           admin_password, network_info=None, block_device_info=None):
3
4     [...]
5
6     bind_sgx_cmd = 'lxc config device add {} /dev/isgx unix-char path=/
          dev/isgx'.format(instance.name)
7     os.system(bind_sgx_cmd)
8     permission_cmd = 'lxc config device set {} /dev/isgx mode 666'.format
          (instance.name)
9     os.system(permission_cmd)
```

Código Fonte A.9: Código do driver LXD no Nova que habilita o uso do dispositivo *isgx* nos contêineres LXD.

Apêndice B

Viabilização de suporte à SGX para imagens do tipo Fedora e Fedora Atomic

O procedimento detalhado a seguir descreve os passos para configuração e instalação do *Intel SGX Driver*, bem como *SGX Software Development Kit* (SDK) e *SGX Plataform Software* (PSW), em imagens Fedora 26 e Fedora Atomic 25, para as quais não há suporte oferecido por padrão. As imagens que possuem suporte são as Ubuntu 16.04 LTS, Red Hat Enterprise Linux Server 7.3 e CentOS 7.3.1611, todas para 64 bits. O processo de instalação realizado nessas imagens foi tomado como base para prover suporte à Fedora 26 e Fedora Atomic 25, e a partir disso, foram feitas as modificações necessárias para adaptar o processo e completar a nova instalação.

Para a imagem do tipo Fedora Atomic 25, inicialmente é feita uma atualização do *kernel* da máquina virtual, permitindo assim a instalação dos pacotes de *kernel* e demais dependências do SGX em suas versões adequadas. Nos comandos listados no Código Fonte B.1 abaixo, os pacotes necessários para a instalação do *driver* SGX e do SDK/PSW são listados. Além disso, para essa imagem, as mudanças só são incorporadas após a reinicialização da máquina virtual, como também vemos nos comandos do Código Fonte B.1, após cada atualização e instalação.

Por padrão, Fedora Atomic não permite que mudanças sejam executadas no diretório raiz da máquina virtual, e para instalação do driver SGX e seu SDK/PSW, é necessário ter acesso a esses diretórios. Por isso, precisamos permitir tais modificações e fazê-las persistentes quando a máquina por ventura for reiniciada. Utilizamos para isso o comando `unlock` acompanhado do modo `-hotfix`, como vemos no Código Fonte B.1 a seguir.

Outra configuração que precisa ser ajustada diz respeito às políticas de segurança providas pelo *Security-Enhanced* (SE) Linux sobre todos os processos e objetos do sistema. Em imagens Fedora Atomic, essa configuração vem por padrão com o modo `enforcing` ativado, significando que as políticas de segurança estão obrigatoriamente em vigor e a execução de determinadas ações não serão permitidas. Para que o serviço *aesmd* do SGX PSW seja inicializado, essa configuração precisa ser modificada para `permissive`, permitindo assim a inicialização desse serviço.

```
1  ## Atualiza kernel
2  $ sudo atomic host upgrade
3  $ sudo systemctl reboot
4
5  ## Instala pacotes dependentes
6  $ sudo atomic host install git gcc gcc-c++ kernel-devel kernel-headers
       elfutils-libelf-devel
7  $ sudo systemctl reboot
8  $ sudo atomic host install ocaml wget libcurl-devel protobuf-compiler
       protobuf-devel autoconf automake gettext-devel libtool openssl-devel
9  $ sudo systemctl reboot
10
11 ## Permite modificar diretorio raiz
12 $ sudo ostree admin unlock --hotfix
13
14 ## Arquivo editado do modo 'enforcing' para 'permissive'
15 # SELINUX=enforcing -> SELINUX=permissive
16 $ sudo vi /etc/selinux/config
17 $ sudo systemctl reboot
```

Código Fonte B.1: Comandos para preparação de ambiente para instalação do driver SGX e seu SDK/PSW numa máquina virtual usando imagem Fedora Atomic 25.

Por fim, com o ambiente preparado para a instalação do *driver* SGX e seu SDK/PSW, podemos executar os passos indicados nos guias da Intel. O Código Fonte B.2 abaixo mostra o processo para ambos os módulos. A exceção está na construção do módulo PSW. Uma modificação no arquivo que configura o serviço *aesmd* é necessária para indicar o diretório correto da *flag InaccessibleDirectories*.

```
## Instala Driver Linux SGX
$ git clone https://github.com/01org/linux-sgx-driver.git
$ cd linux-sgx-driver
$ make
$ sudo mkdir -p "/lib/modules/"`uname -r`"/kernel/drivers/intel/sgx"
$ sudo cp isgx.ko "/lib/modules/"`uname -r`"/kernel/drivers/intel/sgx"
$ sudo touch /etc/modules
$ sudo sh -c "cat /etc/modules | grep -Fxq isgx || echo isgx >> /etc/
    modules"
$ sudo /sbin/depmod
$ sudo /sbin/modprobe isgx

## Instala Linux SDK/PSW
$ git clone https://github.com/01org/linux-sgx.git
$ cd linux-sgx/
$ ./download_prebuilt.sh
$ make
$ make sdk_install_pkg

## Edita arquivo com diretorio correto
$ sudo vi psw/ae/aesm_service/config/aesmd_service/aesmd.service
## InaccessibleDirectories=/var/home

$ make psw_install_pkg
$ cd linux/installer/bin
$ sudo ./sgx_linux_x64_sdk_${version}.bin
$ sudo ./sgx_linux_x64_psw_${version}.bin
```

Código Fonte B.2: Comandos para instalação do driver SGX e seu SDK/PSW numa máquina virtual usando imagem Fedora Atomic 25.

Para a imagem do tipo Fedora 26, a instalação dos pacotes de *kernel* e demais dependências é feita utilizando o gerenciador de pacotes DNF, viabilizando portanto a instalação das versões corretas dos mesmos sem necessidade da atualização do *kernel* do sistema como para a Fedora Atomic 25. Tais passos são descritos no Código Fonte B.3 abaixo.

```
# Instala pacotes dependentes para o driver SGX
$ sudo dnf install git gcc kernel-devel-4.11.8-300.fc26.x86_64 kernel-
    headers-4.11.8-300.fc26.x86_64 elfutils-libelf-devel

# Instala pacotes dependentes para o Intel SDK/PWS
$ sudo dnf groupinstall 'Development Tools'
$ sudo dnf install ocaml wget python gcc-c++ libcurl-devel protobuf-
    compiler protobuf-devel automake autoconf gettext-devel libtool ocaml-
    ocamlbuild compat-openssl10-devel
```

Código Fonte B.3: Comandos para preparação de ambiente para instalação do driver SGX e seu SDK/PSW numa máquina virtual usando imagem Fedora 26.

Em seguida, a instalação do *driver* SGX segue como para a Fedora Atomic 25, e a instalação do Intel SDK/PSW sofre uma pequena alteração junto à realização do *make*, onde as *flags* `CFLAGS` e `CXXFLAGS` são passadas. Os comandos utilizados são detalhados no Código Fonte B.4 abaixo.

```
## Instala Linux SDK/PSW
$ git clone https://github.com/01org/linux-sgx.git
$ cd linux-sgx/
$ ./download_prebuilt.sh

# Configura flags junto ao make
$ CFLAGS="-Wno-deprecated -Wno-implicit-fallthrough" CXXFLAGS="-Wno-
    deprecated -Wno-implicit-fallthrough" make

$ make sdk_install_pkg
$ make psw_install_pkg
$ cd linux/installer/bin
$ sudo ./sgx_linux_x64_sdk_${version}.bin
$ sudo ./sgx_linux_x64_psw_${version}.bin
```

Código Fonte B.4: Comandos para instalação do Intel SGX SDK/PSW numa máquina virtual usando imagem Fedora 26.

Apêndice C

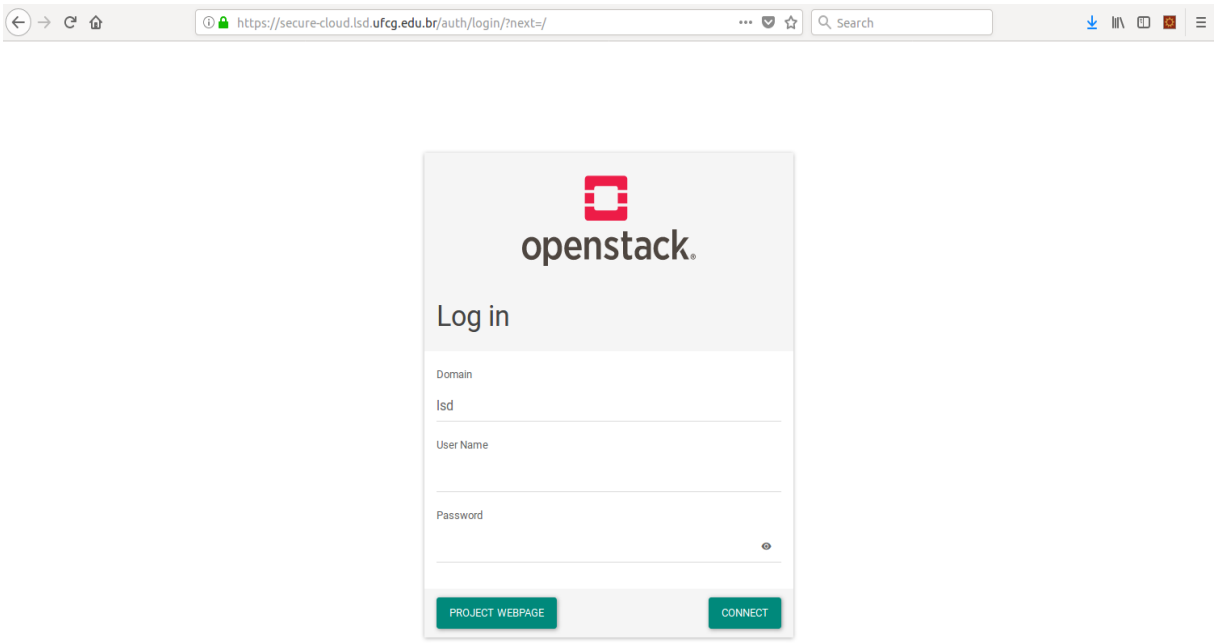
Usando a nuvem segura: criação de instâncias Nova

Neste Apêndice são apresentados os passos para criar instâncias no Nova usando a solução proposta nesse trabalho para prover, escalar e contabilizar o uso de recursos SGX em um ambiente OpenStack. Para isso, nós usamos a nossa nuvem, composta de servidores SGX e acessível através do Horizon, interface Web do OpenStack, disponível em `https://secure-cloud.lsd.ufcg.edu.br`.

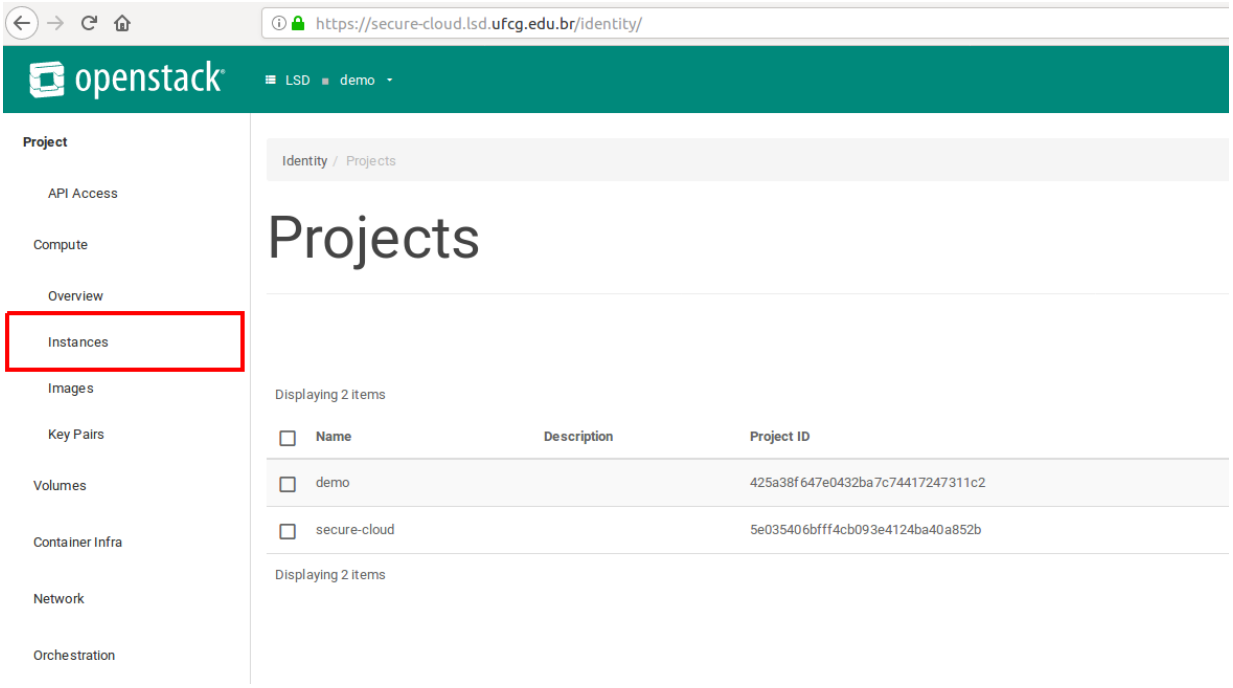
Cada captura de tela abaixo mostra um passo no processo de criação da instância. Depois de criada, executamos verificações a respeito do provisionamento da mesma, assegurando que a quantidade de EPC requisitada foi alocada com sucesso. A contabilidade do uso desse recurso é visível para os usuários das instâncias através de um script provido por um *driver* SGX modificado que acessa os dados a respeito do uso dos recursos SGX e exibe os resultados ao usuário. Da perspectiva da nuvem, o uso de EPC de um projeto no OpenStack é visível para o administrador através do comando `openstack usage list`.

A seguir, apresentamos os passos para criar uma instância Nova com acesso SGX. Capturas de tela e instruções de linha de comando são dadas para cada passo.

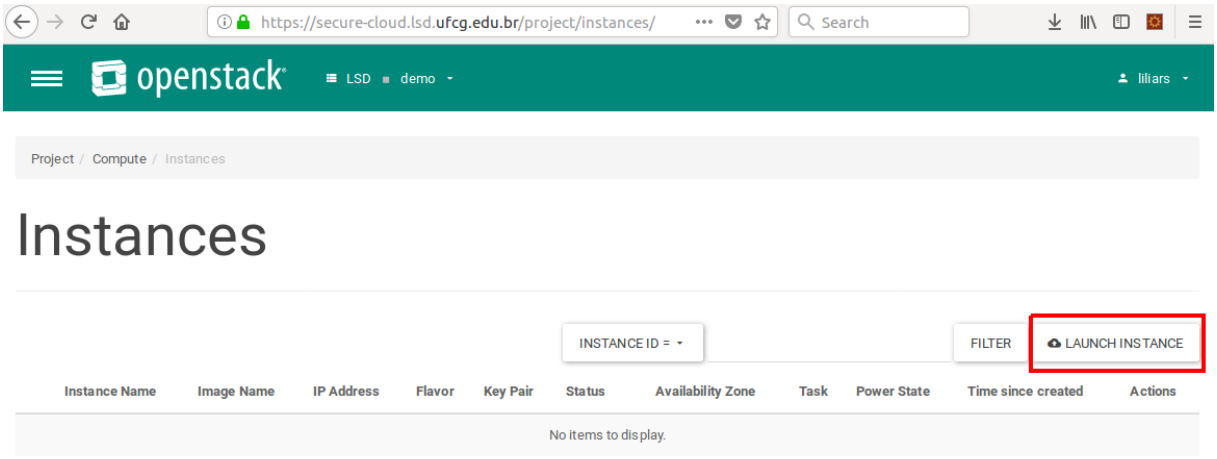
A. Acesse a nuvem



B. Selecione `Instances` no menu `Project – Compute`

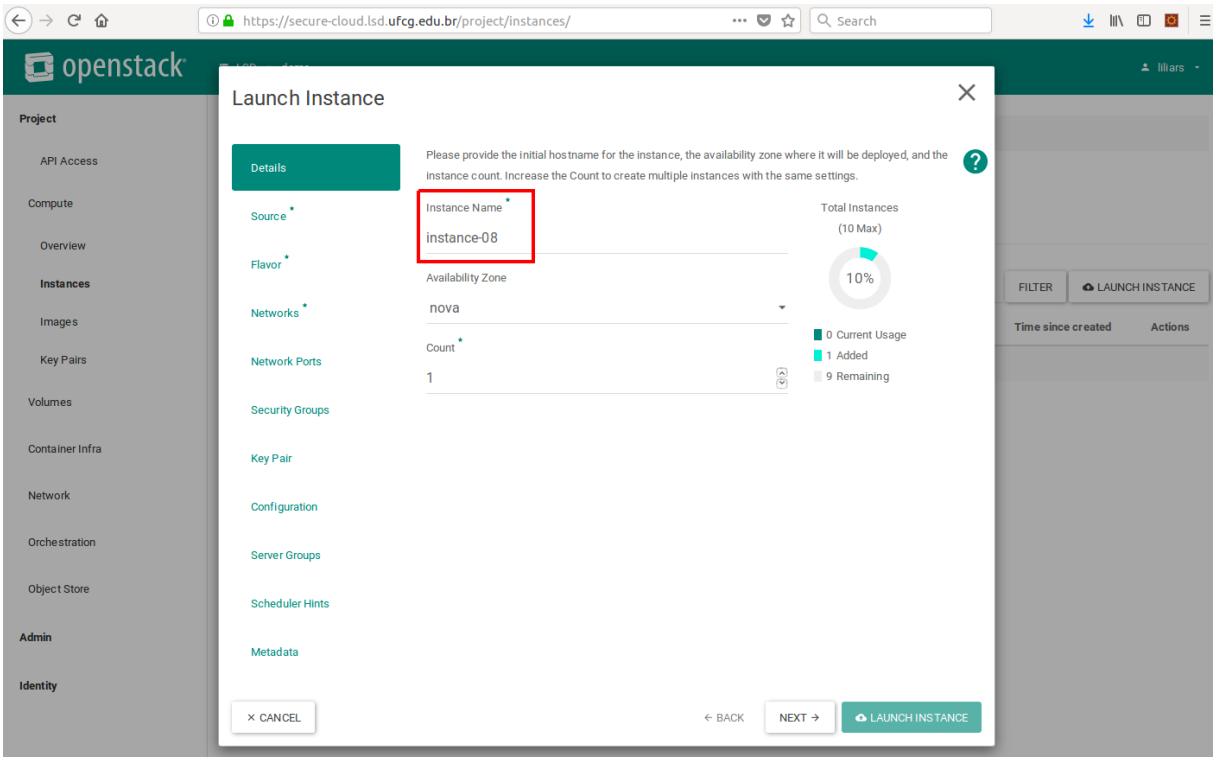


C. Pressione o botão `Launch Instance` na barra de ferramentas



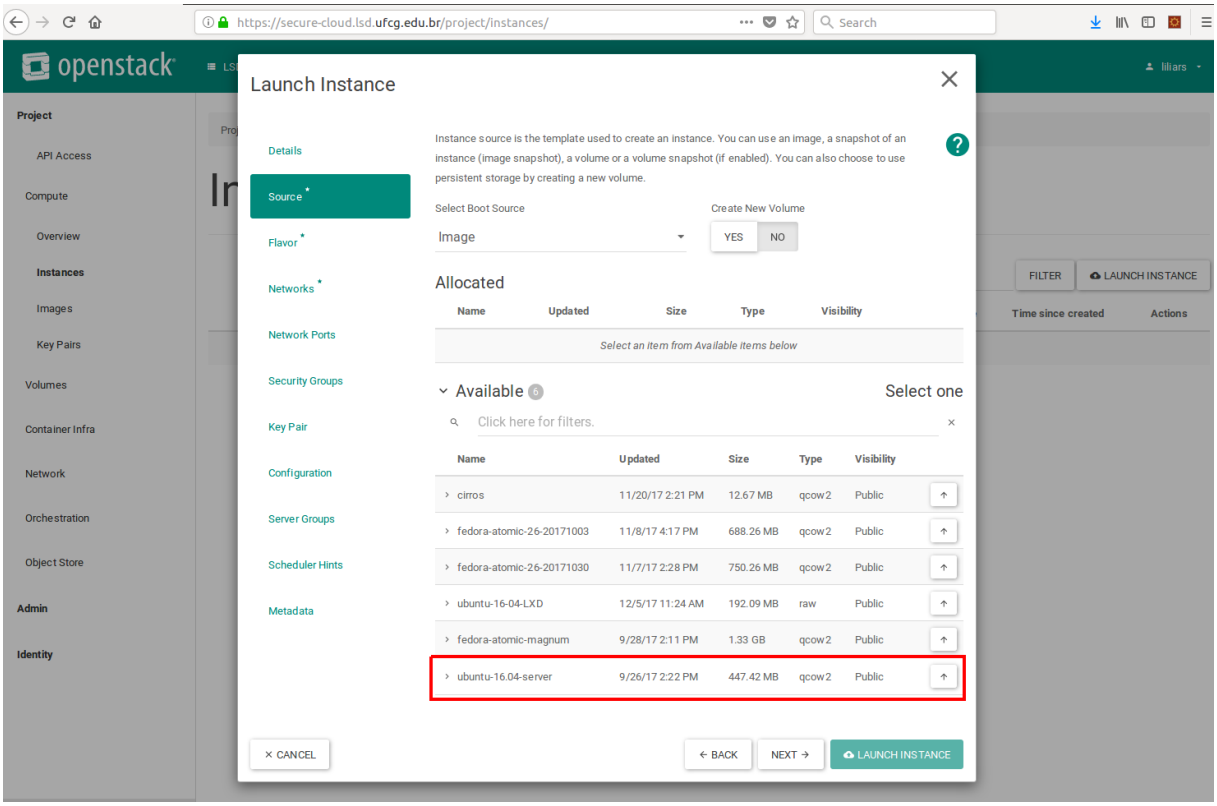
C.1 Configurando uma instância

A. Escolha um nome identificador



B. Selecione uma imagem Ubuntu 16.04

Usamos uma imagem Ubuntu 16.04, compatível com a instalação do *Linux SGX Driver*. Essa imagem pode ser obtida em qualquer repositório do Ubuntu, e foi disponibilizada ao OpenStack através do serviço Glance, mencionado anteriormente neste trabalho. Outras imagens poderiam ser utilizadas, respeitando sua compatibilidade com o *driver* SGX.

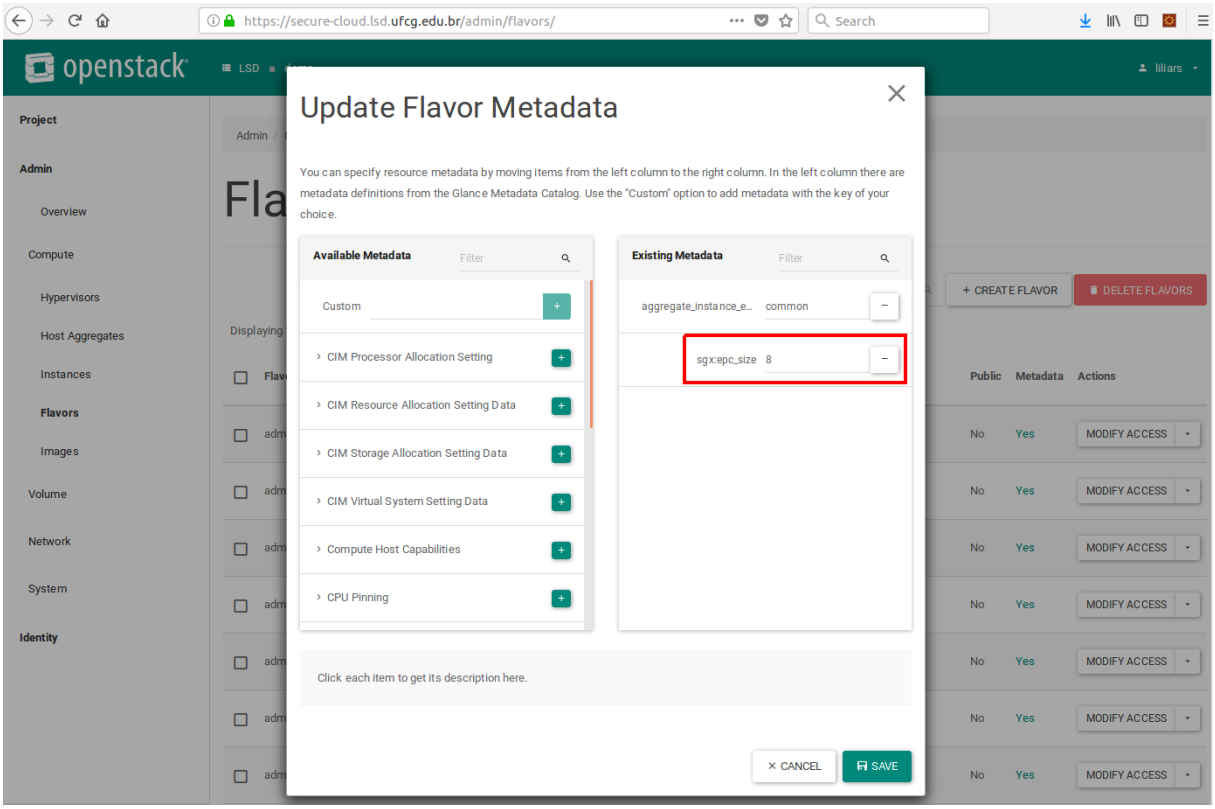


C. Selecione um dos *flavors* disponíveis

Aqui nós temos dois *flavors*, um capaz de criar instâncias com 8 MB de EPC, e outro com 16 MB. Como explicado anteriormente, o tamanho do EPC a ser alocado é definido internamente em um *flavor* através do seu `extra_specs`, nesse caso no formato `sgx:epc_size`. Esses parâmetros são destacados nas figuras abaixo, e exibidas via terminal usando o comando `nova flavor-show`.

É importante notar que *flavors* são criados pelos administradores da nuvem, e portanto, a quantidade de EPC requisitado através deles é definida de acordo. Para os fins deste trabalho, nós decidimos que tais valores podem variar a partir de 8 MB, 16 MB, 32 MB até 64 MB.

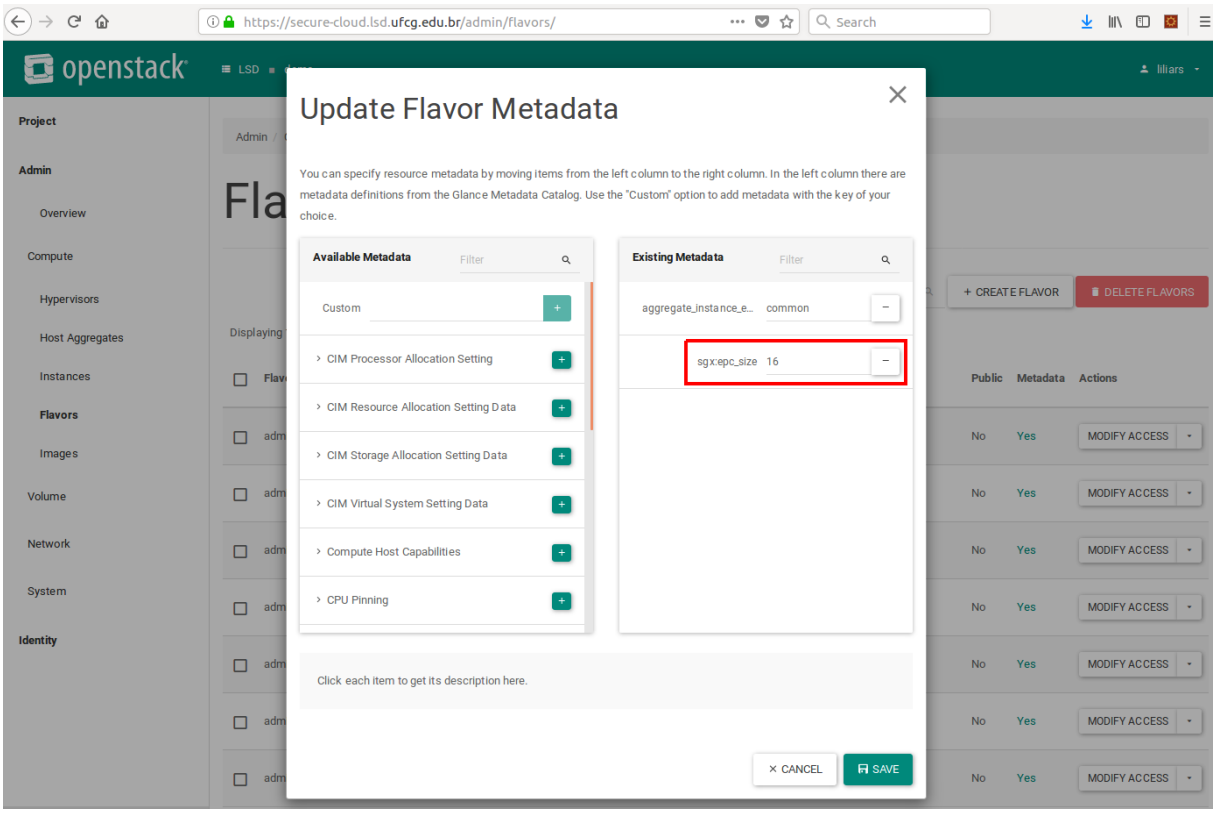
No final do processo descrito neste capítulo, ambos os *flavors* terão sido usados para criar duas instâncias, com tamanhos diferentes de EPC.



Flavor `admin.c.small` para instâncias com $8\ MB$ de EPC.

```
(novaclient) liliars@triunfo-pc-84:~$ nova flavor-show admin.c.small
+----------------------------+-------------------------------------------------------------------------------------+
| Property                   | Value                                                                               |
+----------------------------+-------------------------------------------------------------------------------------+
| OS-FLV-DISABLED:disabled   | False                                                                               |
| OS-FLV-EXT-DATA:ephemeral  | 0                                                                                   |
| disk                       | 60                                                                                  |
| extra_specs                | {"aggregate_instance_extra_specs:aggregate": "common", "sgx:epc_size": "8"}         |
| id                         | 17b312c0-5386-4725-a959-12b9bc9885b1                                                |
| name                       | admin.c.small                                                                       |
| os-flavor-access:is_public | False                                                                               |
| ram                        | 2048                                                                                |
| rxtx_factor                | 1.0                                                                                 |
| swap                       |                                                                                     |
| vcpus                      | 1                                                                                   |
+----------------------------+-------------------------------------------------------------------------------------+
```

Linha de comando mostrando o *flavor* `admin.c.small` para instâncias com $8\ MB$ de EPC.

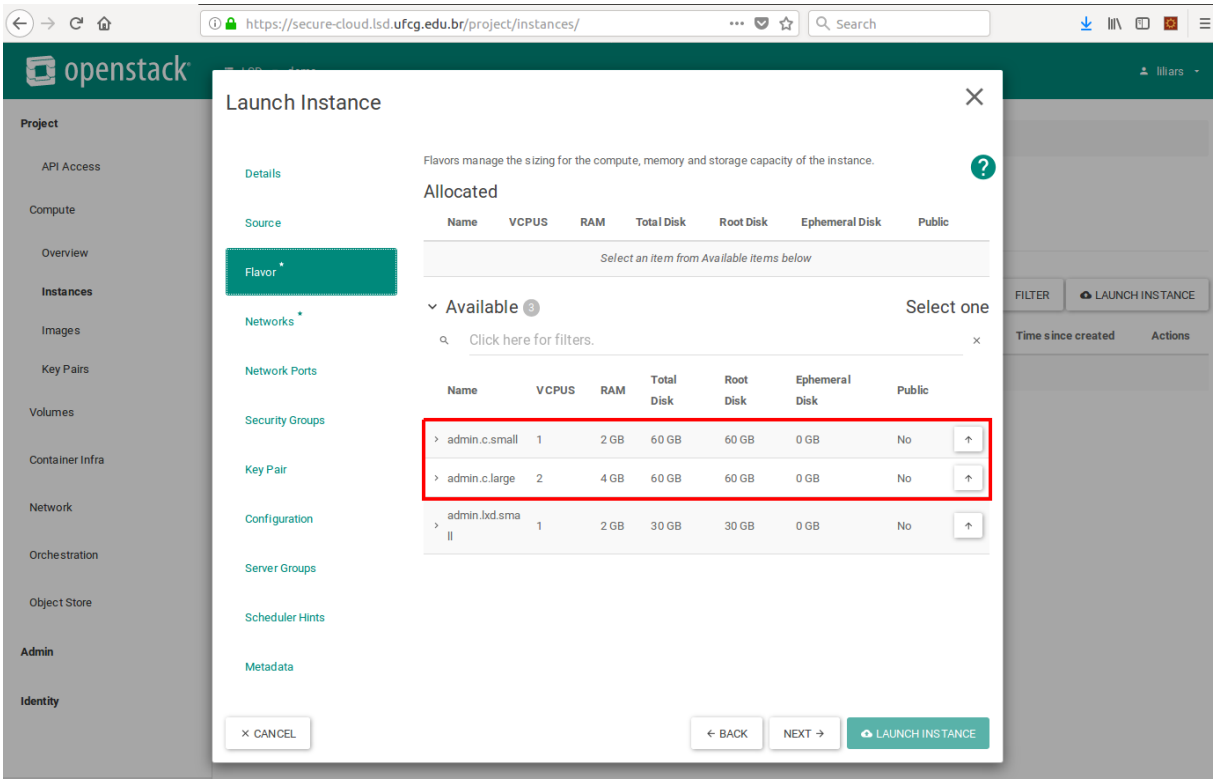


Flavor `admin.c.large` para instâncias com 16 MB de EPC.

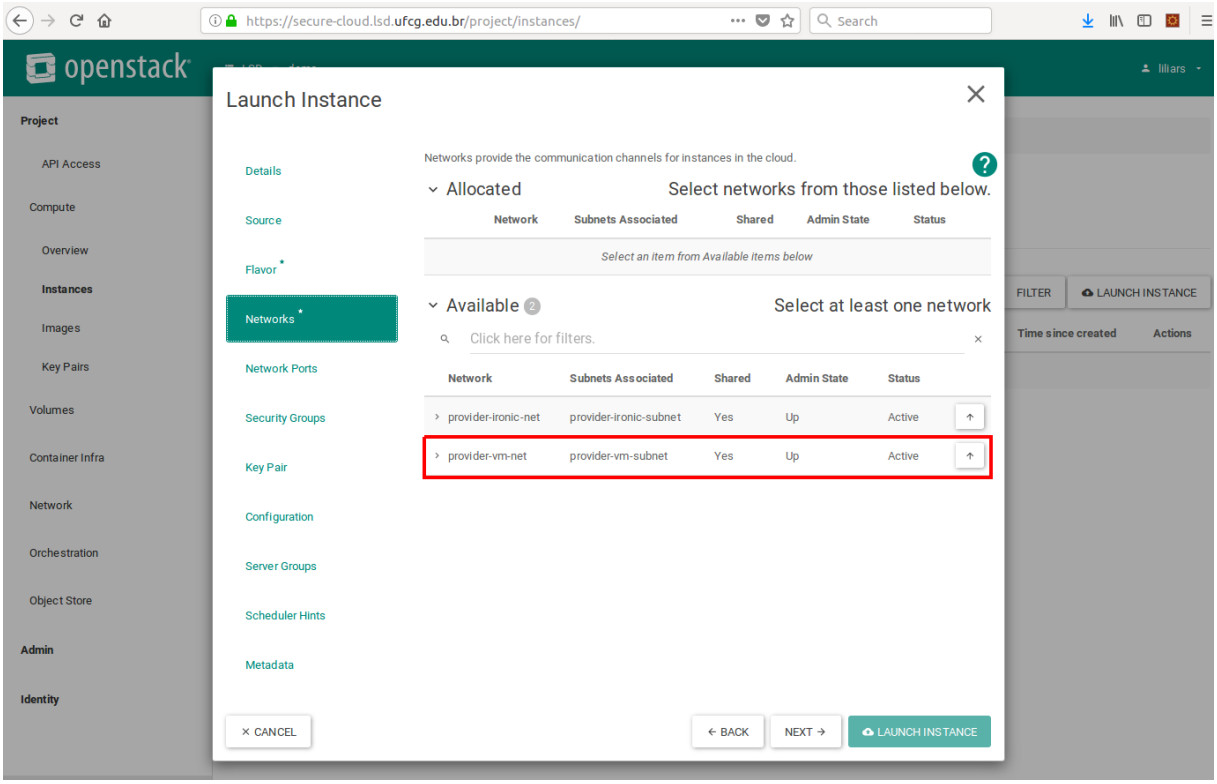
```
(novaclient) liliars@triunfo-pc-84:~$ nova flavor-show admin.c.large
+----------------------------+-----------------------------------------------------------------------------------+
| Property                   | Value                                                                             |
+----------------------------+-----------------------------------------------------------------------------------+
| OS-FLV-DISABLED:disabled   | False                                                                             |
| OS-FLV-EXT-DATA:ephemeral  | 0                                                                                 |
| disk                       | 60                                                                                |
| extra_specs                | {"aggregate_instance_extra_specs:aggregate": "common", "sgx:epc_size": "16"}      |
| id                         | 1f177f11-b6fc-4436-b0e0-414d1c2e4ea5                                              |
| name                       | admin.c.large                                                                     |
| os-flavor-access:is_public | False                                                                             |
| ram                        | 4096                                                                              |
| rxtx_factor                | 1.0                                                                               |
| swap                       |                                                                                   |
| vcpus                      | 2                                                                                 |
+----------------------------+-----------------------------------------------------------------------------------+
```

Linha de comando mostrando o *flavor* `admin.c.large` para instâncias com 16 MB de EPC.

Considerando esses *flavors*, para criar uma instância com 8 MB de EPC, nós selecionamos o *flavor* `admin.c.small`. Similarmente, para criar uma instância com 16 MB de EPC, nós selecionamos o *flavor* `admin.c.large`. Ambas as opções são vistas na figura abaixo.

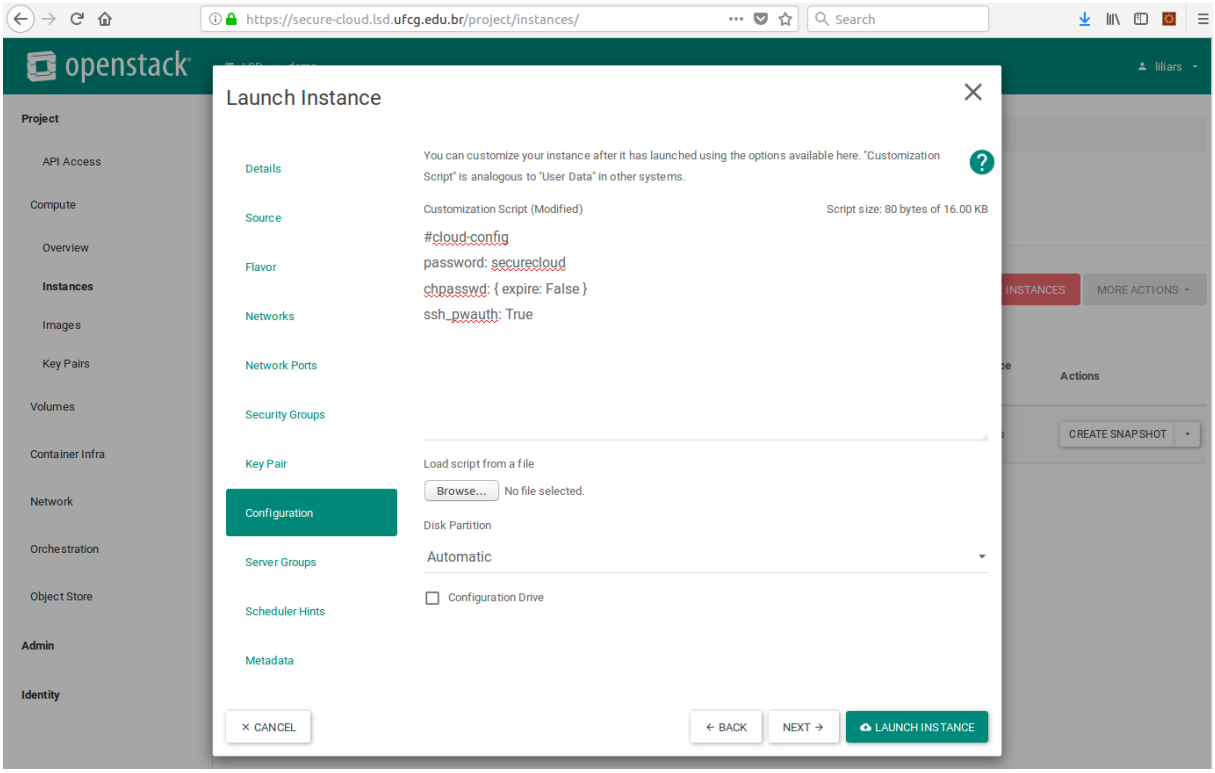


D. Selecione uma rede de provisionamento

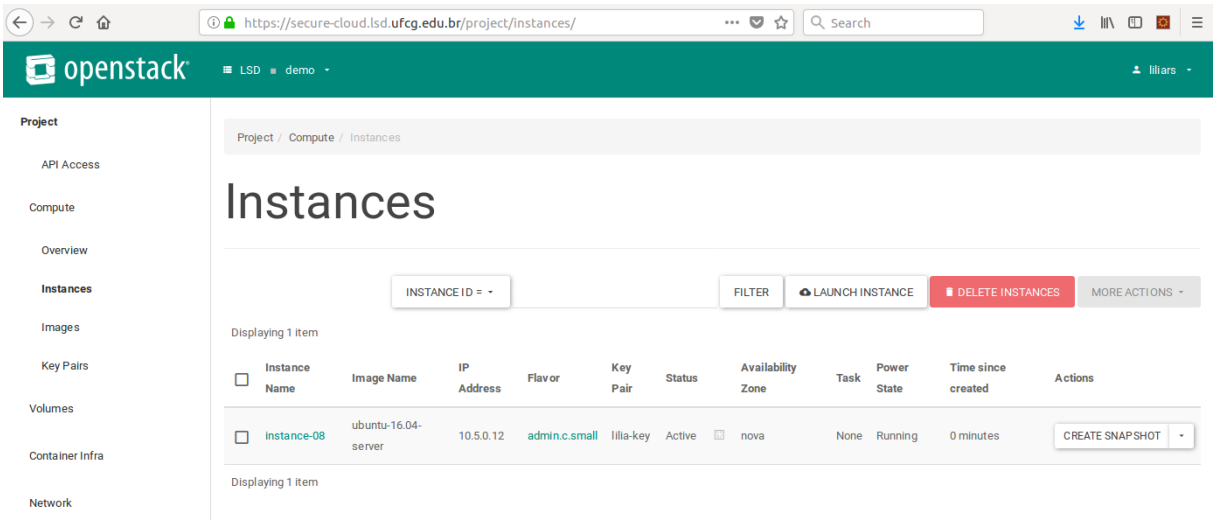


E. Insira configurações no `cloud-config` para acessar a instância

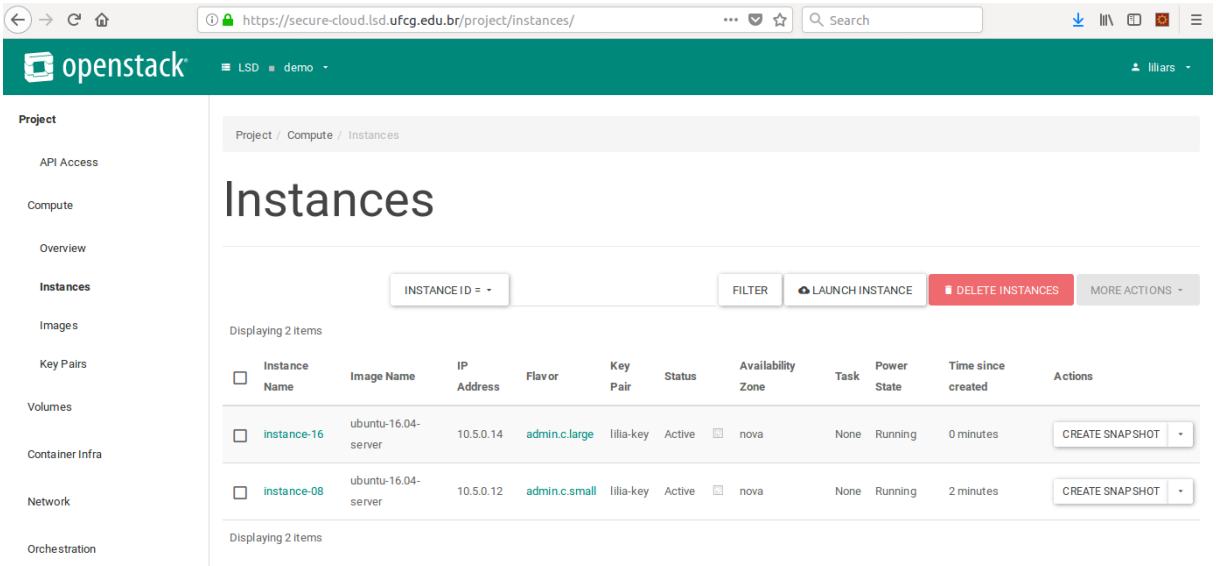
Para acessar uma instância, um usuário pode passar uma chave antes de sua criação, ou submeter suas credenciais através de um script no `cloud-config`. Aqui nós usamos a segunda opção. Depois disso, é necessário apenas criar no botão `Launch Instance`.



C.1.1 Iniciando uma instância



Instância com $8\ MB$ de EPC iniciada com sucesso.



Instância com $16\ MB$ de EPC iniciada com sucesso.

C.1.2 Instalando o *driver* SGX nas instâncias

Depois das instâncias terem sido iniciadas com sucesso, elas podem ser acessar via SSH a partir de um terminal ou pelo console próprio do OpenStack. Aqui nos usamos as credencias definidas no `cloud-init` para se conectar via SSH a partir de um terminal.

Em seguida, nós instalamos o *driver* SGX modificado, mencionado anteriormente, dentro da VM. Na seção seguinte, o provisionamento das instâncias é verificado com base no total de EPC alocada para cada uma.

A. Instale dependências

```
1 $ sudo su
2 $ apt-get install linux-headers-4.4.0-96-generic -y
```

Código Fonte C.1: Instalando dependências do driver SGX.

B. Clone o repositório

O repositório usado aqui é uma versão do código original do *Linux SGX Driver*, adicionando as modificações necessárias para habilitar a exibição de dados a respeito do uso de recursos SGX.

```
1 $ git clone https://git.lsd.ufcg.edu.br/secure-cloud/linux-sgx-driver.git
    -b sgx_epc
```

Código Fonte C.2: Clonando repositório do driver SGX customizado.

C. Instale o *driver* modificado

```
1 $ cd linux-sgx-driver
2 $ make
3 $ make install
4 $ depmod -a
5 $ modprobe isgx
```

Código Fonte C.3: Instalando o driver SGX.

C.1.3 Verificando o uso de EPC em instâncias e projetos

A. Como um usuário

A partir das instâncias criadas, usuários podem verificar que a quantidade de EPC requisitada foi corretamente alocada através de um script provido pelo *driver* modificado previamente instalado. As variáveis `sgx_nr_total_epc_pages` e `sgx_nr_free_pages` mostram o número de páginas de EPC, considerando o tamanho de cada página como $4\ KB$, o que deve ser compativel com o uso para instâncias com $8\ MB$ e $16\ MB$ de EPC, respectivamente mostrados nas Figuras C.1 e C.2.

```
root@instance-08:~/linux-sgx-driver# ifconfig ens3
ens3      Link encap:Ethernet  HWaddr fa:16:3e:3d:ef:66
          inet addr:10.5.0.12  Bcast:10.5.255.255  Mask:255.255.0.0
          inet6 addr: fe80::f816:3eff:fe3d:ef66/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:116088 errors:0 dropped:0 overruns:0 frame:0
          TX packets:11472 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:163396065 (163.3 MB)  TX bytes:857595 (857.5 KB)

root@instance-08:~/linux-sgx-driver# ./show_values.sh
sgx_nr_total_epc_pages= 2048
sgx_nr_free_pages= 2048
sgx_nr_low_pages= 32
sgx_nr_high_pages= 64
sgx_nr_marked_old= 0
sgx_nr_evicted= 0
sgx_nr_alloc_pages= 0
```

Figura C.1: Uso de EPC para instância com $8\ MB$ de EPC, endereço IP 10.5.0.12.

```
root@instance-16:~/linux-sgx-driver# ifconfig ens3
ens3      Link encap:Ethernet  HWaddr fa:16:3e:ab:f8:73
          inet addr:10.5.0.14  Bcast:10.5.255.255  Mask:255.255.0.0
          inet6 addr: fe80::f816:3eff:feab:f873/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:117359 errors:0 dropped:0 overruns:0 frame:0
          TX packets:13000 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:165035818 (165.0 MB)  TX bytes:952769 (952.7 KB)

root@instance-16:~/linux-sgx-driver# ./show_values.sh
sgx_nr_total_epc_pages= 4096
sgx_nr_free_pages= 4096
sgx_nr_low_pages= 32
sgx_nr_high_pages= 64
sgx_nr_marked_old= 0
sgx_nr_evicted= 0
sgx_nr_alloc_pages= 0
```

Figura C.2: Uso de EPC para instância com $16\ MB$ de EPC, endereço IP 10.5.0.14.

B. Como administrador da nuvem

O administrador da nuvem tem acesso ao uso de EPC de todos os *tenants* na nuvem. Na Figura C.3, a coluna `Tenant ID` presenta cada projeto existente. As demais colunas a respeito de servidores, memória, CPU e disco, são também parte do comando `nova usage-list`. Finalmente, a última coluna, `EPC MB-Hours`, representa o uso de EPC ao longo do tempo.

```
(novaclient) liliars@triunfo-pc-84:~$ nova usage-list
Usage from 2017-11-20 to 2017-12-19:
+----------------------------------+---------+--------------+-----------+---------------+--------------+
| Tenant ID                        | Servers | RAM MB-Hours | CPU Hours | Disk GB-Hours | EPC MB-Hours |
+----------------------------------+---------+--------------+-----------+---------------+--------------+
| 5e035406bfff4cb093e4124ba40a852b | 14      | 7306979.48   | 6520.21   | 90952.38      | 0.00         |
| 2898c119bc86410fbc27ae4a885d9fda | 199     | 35731737.37  | 10635.31  | 400412.59     | 2.75         |
| d3116a07f8f94c3d8dd442064429d44f | 2781    | 3995118.05   | 1950.74   | 97537.06      | 2.03         |
| 425a38f647e0432ba7c74417247311c2 | 7       | 1190339.27   | 581.22    | 23833.78      | 2866.81      |
+----------------------------------+---------+--------------+-----------+---------------+--------------+
```

Figura C.3: Uso de EPC para todos os projetos do OpenStack.