Universidade da Beira Interior
Departamento de Informática

From Virtual Machines to Containers and Unikernels

Elaborado por:

Nelson Miguel de Sousa Matias

Orientador:

Professor Doutor Paul Andrew Crocker

22 de Junho de 2018

Agradecimentos

A feitura deste projecto, no que concerne ao foro pessoal, representa o término de uma etapa da minha vida, que deveria ter sido concluída há imensos anos! O significado deste momento, considerando todas as vicissitudes passadas, acompanhadas de muita incúria e insensatez, é por tal, biguano! No que concerne ao foro profissional, este momento, representa uma excelente oportunidade que me foi concedida, de estudar, experimentar e testar, uma série de tecnologias actuais, que envolvem a área respeitante à actividade que quero exercer nos meus dias vindouros. Por todas estas razões, quero deixar os meus mais sinceros agradecimentos, a todas as pessoas que, directa ou indirectamente, influenciaram de forma positiva, o meu percurso, na completação desta aventura académica.

Começo por declarar os meus maiores agradecimentos ao meu orientador: o Professor Doutor Paul Crocker, que para além da sua anuência relativamente à minha participação neste projecto, pude também contar com o seu total apoio e dedicação, na disponibilização dos meios e ferramentas necessárias, para a execução do mesmo. Quero também agradecer-lhe, pela sua postura firme e ao mesmo tempo indulgente, em todos os momentos marasmáticos e menos produtivos que atravessei.

Agradeço imensamente, a disponibilidade, prestabilidade e afabilidade, demonstradas pelos meus colegas de laboratório: o Rafael Couto, o José Manteigueiro, o Fábio Pereira, o André Rodrigues e o Bento Martins, que sempre se mostraram prontos para dar uma resposta a todas as minhas solicitações. Solicitações essas, por vezes inoportunas, mas cujas respostas sempre assertivas, me permitiram ultrapassar muitos dos obstáculos com que me deparei neste repto.

Aproveito também para agradecer profundamente e com veneração, à minha mãe: Isabel Salvado, a mirífica postura de apoio incondicional e absoluto, em todas as etapas da minha vida, enfatizando claro, a presente. Sendo para mim uma verdadeira referência, ainda assim, consegue demonstrar-me numa base diária, o verdadeiro significado de altruísmo para com a sua prole, atendendo a todas as minhas solicitações de uma forma sempre desinteressada e inspiradora!

Por fim, mais que um agradecimento, dedico este trabalho às pessoas mais importantes da minha vida, os meus descendentes: o Tito Matias e o Isaac Matias, pelo alento diário que me transmitem de forma natural e inconsciente, por darem um significado especial a todos os meus dias e, por serem a principal razão deste meu empreendimento.

Conteúdo

Lista de Figuras

Lista de Tabelas

Listagens

Acrónimos

API Application Programming Interface

AUFS Another Union File System

BRIC Bunch of Redundant Independent Clouds

CE Community Edition

CLI Command-line Interface

CoW Copy on Write

CPU Central Processing Unit

EPOS European Plate Observing System

ESFRI European Strategy Forum on Research Infrastructures

GNSS Global Navigation Satellite System

HTTP HyperText Transfer Protocol

HVM Hardware Virtual Machine

ICT Information & Communication Technology

IP Internet Protocol

ISA Instruction Set Architecture

JSON JavaScript Object Notation

LVM Logical Volume Manager

QoS Quality of Service

PVHVM Para-virtualisation on HVM

SDN Software Defined Networking

SGBD Sistema de Gestão de Bases de Dados

TCB Trusted Code Base

TI Tecnologias da informação

VM Virtual Machine

VMM Virtual Machine Monitor

Glossário

Cloud É um termo generalizado que pressupõe a prestação de serviços computacionais remotos, acedidos pela internet.. 5

Cluster É um grupo de servidores e outros recursos, que ajem como um sistema único e, permitem a disponibilização dos seus serviços de uma forma altamente disponível e equilibrada.. 23, 38

Compilador É um programa especial, que processa declarações escritas numa linguagem de programação específica, interpreta-as e converte-as em código máquina.. 13, 21

Container É um tipo de virtualização ao nível do sistema operativo, com recursos dedicados em termos de processador, memória, armazenamento e rede, partilhando no entanto, o núcleo do sistema operativo hospedeiro.. 9, 13, 17, 20, 26, 27, 31, 37, 38

Daemon É uma aplicação que está sempre em execução, com o propósito de tratar os pedidos respeitantes ao serviço que disponibiliza. Esses pedidos podem ser reencaminhados para outro serviço, ou podem ser devidamente processados.. ix, 19, 21, 26, 33, 34

Datapath É um conjunto de unidades funcionais constituintes da Central Processing Unit (CPU), que executam operações de processamento de dados.. 19

Driver É uma pequena aplicação, que instrui o sistema operativo e outras aplicações, a comunicar com um determinado dispositivo de *hardware*.. 10, 11, 13, 18, 20

Exokernel É a imagem gerada pelo unikernel, para ser instanciada no hypervisor.. 14, 17, 26, 33, 34

Firewall É um *software* ou *firmware*, que impõe um conjunto de regras, relativamente aos pacotes que circulam numa determinada rede. São usados para filtrar o tráfego e baixar o risco da entrada de pacotes maliciosos que possam ter algum impacto negativo na segurança de uma rede privada.. 3, 13

Gateway É um dispositivo de rede que funciona como um nodo, com a capacidade de interligar duas redes que podem ter topologias diferentes.. 5

Hypercall É uma chamada de sistema feita de uma máquina virtual, directamente ao *hypervisor*. As máquinas virtuais, fazem pedidos de instruções privilegiadas através deste processo.. 10, 13

Hypervisor É uma camada de *software* existente entre o *hardware* e o sistema operativo anfitrião, responsável por fornecer aos sistemas operativos convidados, a abstração da máquina virtual.. 9–15, 17, 18, 21, 23, 24, 26, 35

Malware É um *software* malicioso que tem por objectivo, causar algum dano a um utilizador informático. Este *software*, pode desempenhar funções como: roubar; encriptar; apagar; alterar ou sequestrar informação e até mesmo, monitorizar a utilização de um sistema sem o consentimento do utilizador.. 3

Open Source É um tipo de licença de *software*, cuja distribuição se baseia nos seguintes termos: livre redistribuição, código fonte, *software* derivado, integridade do código fonte do autor, não descriminação de pessoas ou grupos, não descriminação de temas ou empreendimentos, distribuição da licença, a licença não pode ser específica a um produto, a licença não pode restringir outro software e a licença tem que ser tecnologicamente neutra.. 17–19

Plugin É um programa que é usado em conjunto com um programa principal, que disponibiliza a este, funcionalidade extra.. 34

Routing É o caminho, definido por um conjunto de protocolos de comunicação, que os pacotes de dados percorrem, através de multiplas redes, desde a sua origem até ao seu destino.. 13

Script É uma sequência de instruções que são interpretadas e/ou executadas pelo programa a que se destinam.. ix, 7, 27, 31, 32

Servidor É um computador, devidamente equipado para atender pedidos de informação, processar esses pedidos e, envia-los para outro computador pela internet, ou pela rede local.. vii, ix, 18, 23–25

Switch É um dispositivo que encaminha pacotes de rede de alguma porta de entrada, para uma porta específica de saída, que transportará esses pacotes até ao seu destino.. 3

Sysadmin Um administrador de sistemas, é um profissional especializado, que controla, monitoriza e optimiza, um ambiente computacional de dimensão variável, permitindo o seu bom uso e excelente desempenho.. 4, 5

Thin Provisioning É uma tecnologia de virtualização, associada a qualquer recurso computacional, como a memória ou o espaço em disco, onde é criado um nível de abstracção entre a quantidade (maior) do recurso disponibilizado e, a quantidade (menor) do recurso existente na realidade.. 3

Unikernel É uma máquina virtual que implementa um sistema operativo com o mínimo absoluto de funcionalidades e bibliotecas, suficiente para executar a única aplicação que suporta.. 9, 14, 15, 21, 26, 33, 35–37

Wrapper É um programa que é usado para encapsular outro programa, de funcionalidade ou natureza diferentes, de forma a executá-lo noutro tipo de ambiente.. 38

Capítulo 1

Introdução

O European Plate Observing System (EPOS), é um projecto europeu, que representa a infraestrutura integrada de investigação das ciências da Terra sólida, com aprovação do European Strategy Forum on Research Infrastructures (ESFRI). O seu propósito, é criar sinergias através de cientistas, instituições de investigação, peritos em Information & Communication Technology (ICT), executivos e o público em geral, para promover, abordagens inovadoras que permitam um melhor entendimento dos processos físicos que geram: os tremores de terra; as erupções vulcânicas e os maremotos, incluindo todos os fenómenos causados pelo movimento das placas tectónicas e pela própria dinâmica da superfície terrestre.[on Solid Earth, 2018]

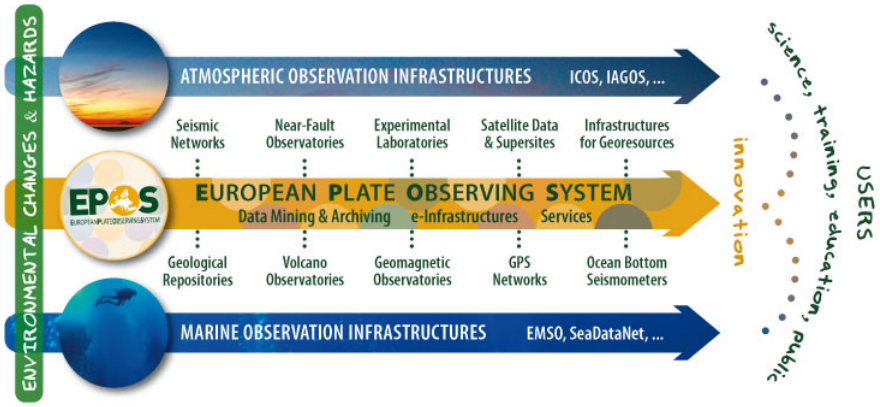

Figura 1.1: Estrutura do sistema EPOS

O EPOS, visa a criação de uma rede pan-europeia, em égide de uma sociedade sustentável e mais segura. Através da integração das infraestruturas de in-

vestigação e dados, este intento, permitirá aos cientistas, efectuar uma mudança marcante, no desenvolvimento de novos conceitos relativos a geoperigos e georrecursos e, no desenvolvimento de aplicações concernentes às ciências da Terra, para que consigamos responder de forma precisa e sustentada, às questões sociais respeitantes ao meio ambiente e à nossa qualidade de vida.[on Solid Earth, 2018]

A visão deste projecto, é a possibilidade do aumento do acesso e do uso, de dados multidisciplinares, armazenados pelas redes de monitorização da Terra sólida, que foram obtidos em experimentos laboratoriais ou, produzidos por simulações computacionais. Depois de estabelecido, o EPOS irá promover uma interoperabilidade destes serviços, a uma grande comunidade de utilizadores a nível mundial.[on Solid Earth, 2018]

1.1 Enquadramento

O trabalho proposto, tem bastante interesse, não só na unidade curricular em que se enquadra, como também no ambiente tecnológico empresarial e institucional, vigente no nosso quotidiano. A virtualização dos recursos informáticos, pode ter um papel vital, no compromisso financeiro e ambiental, existente nas empresas e nas instituições. O principal objectivo da virtualização, é possibilitar a execução, de múltiplas instâncias de sistemas operativos e respectivas aplicações num único sistema físico, para haver um aumento de eficiência, com particular ênfase, nos ambientes empresariais de economias de escala.

A virtualização, proporciona mais capacidade computacional com os mesmos recursos, simplifica os métodos operacionais e dá ao tecido empresarial, um acréscimo de competitividade nos seus modelos de negócio. Para concretizar este cenário, descrevem-se de seguida, alguns dos grandes benefícios da virtualização neste contexto:

- **Redundância e Tolerância a Falhas** são talvez as duas características mais relevantes em qualquer implementação de Tecnologias da informação (TI), actualmente. Falhas de sistema, podem causar perdas insuperáveis em qualquer entidade colectiva. Por este motivo, a virtualização permite que as instâncias em execução de um qualquer sistema numa determinada VM, sejam realocadas noutra VM, para recuperação do serviço em caso de falha permanente. Sem a redundância, muitas empresas e instituições, incorreriam facilmente em situações de perda de informação. Noutra perspectiva, a tolerância a falhas torna a resolução deste tipo de problemas ainda mais eficiente, porque permite a ligação de dois ou mais servidores entre si e, no

caso de haver alguma ocorrência em algum deles, os restantes poderão continuar a disponibilizar o serviço prestado, como se nada de anormal tivesse sucedido. A redundância e a tolerância a falhas dos sistemas, são excelentes aliados, quando existem procedimentos de recuperação de dados, nos casos em que a informação é apagada ou destruída, porque garantem um desempenho óptimo durante essas operações.[Hillen, 2016]

- **Migração de recursos**, é uma funcionalidade de uma conveniência inigualável na virtualização. Pelo facto de ser fácil, passar da infraestrutura física para a infraestrutura virtual, à medida que as necessidades o exigem, pode referir-se o caso do armazenamento em disco. Neste caso, várias tecnologias permitem a alocação de volumes de armazenamento virtuais, onde rapidamente se pode migrar a informação para volumes físicos e vice-versa. Também é possível migrar dados de um volume virtual ocupado completamente, para outro volume virtual com a tecnologia *Thin Provisioning*. Com a migração, os administradores de sistemas, podem facilmente prevenir o desperdício de recursos, aprovisionando só o que é estritamente necessário para um determinado ambiente e de uma forma organizada.[Hillen, 2016]

- ***Firewalls* virtuais e Segurança** representam um tópico que envolve um desafio permanente e considerável para as empresas e para as instituições. Existe um benefício por parte dos *Firewall* virtuais, para mitigar o problema da segurança na protecção e no acesso aos dados, a custos muito inferiores, comparativamente aos métodos tradicionais. É de referir que a segurança virtual, envolve o uso de módulos avançados, como os *Switch* virtuais, que ajudam ao bloqueio de ataques maliciosos provenientes de fontes desconhecidas. Existe um isolamento das aplicações, para que não possam ser afectadas por *Malware*. Os *Firewall* virtuais, aplicam as suas funcionalidades na base das redes virtualizadas, para gerar segmentação da informação. Para a quantidade de informação triada por estes sistemas, tal não seria possível fora do contexto da virtualização, sem haver um comprometimento entre a segurança e o desempenho.[Hillen, 2016]

- **Reforço das equipas de TI** é obrigatório, quando as empresas e as instituições decidem avançar para uma implementação cujo funcionamento se baseia na virtualização. Neste cenário, existe uma maior complexidade tecnológica e, por conseguinte, é necessário um maior controlo operacional, onde a fluência dos dados tem que ser irrepreensível.[Hillen, 2016]

- **Redução de custos e protecção do ambiente** será o aspecto mais importante, do ponto de vista social e económico de qualquer empresa ou instituição. O modelo tradicional em que existe um servidor para uma aplica-

> ção, actualmente, é sinónimo de custos elevados e subutilização de recursos. Com a virtualização, é possível uma abordagem antagonista ao modelo tradicional, no sentido em que passa a existir uma consolidação dos recursos computacionais, permitindo a redução de problemas de compatibilidade e, a redução drástica do custo pecuniário envolvido. Com esta abordagem, existe a alocação precisa de recursos necessários à execução das aplicações, o que se traduz num menor consumo energético e, por conseguinte, numa menor pegada ecológica.[Hillen, 2016]

Todo o cenário operacional do EPOS, baseia-se num conjunto de aplicações, que possibilitam o armazenamento, o tratamento e a disponibilização dos dados recolhidos, por toda a infraestrutura integrada de investigação das ciências da Terra sólida. E, sendo o EPOS um projecto europeu, constituído por equipas multidisciplinares europeias, as aplicações supracitadas, serão supostamente acedidas concorrentemente, de uma forma, que pode chegar a ser intensiva. Tendo em conta esta possibilidade, faz todo o sentido proceder à virtualização das instâncias destas aplicações, por forma a haver um melhor aproveitamento dos recursos existentes, já por si limitados.

1.2 Motivação

A especialização profissional que quero exercer e aprofundar, é indubitavelmente a administração de sistemas. É uma área que me fascina, talvez por se encaixar perfeitamente com muitos aspectos da minha personalidade e, seguramente, por ser uma profissão onde cada dia pode ser uma aventura. Atrai-me a parte do controlo, da optimização e da organização, inerentes às incumbências diárias de um *Sysadmin*, onde a resiliência e a perseverança são sentidas com normalidade.

Figura 1.2: Postura profissional de um *Sysadmin*

De acordo com algumas estimativas de mercado, a grande maioria do tecido empresarial que constitui os grandes grupos económicos, à excepção da banca, já virtualizou grande parte da sua actividade computacional. A virtualização tornou-se numa tecnologia de eleição e, a sua aceitação não está longe de ser total! Como consequência directa, a virtualização tornou-se numa área onde um *Sysadmin* já tem que ter algumas competências adquiridas.

Uma das competências a enfatizar, é a gestão de redes. Grandes intervenientes no mercado, como a ©*Microsoft* e a ©*VMWare*, estão a apostar em arquitecturas como o Software Defined Networking (SDN), que permitem aos *Sysadmin* um nível de controlo e flexibilidade sobre as redes virtuais, sem precedentes. Por outro lado, a criação de camadas lógicas e camadas virtuais sobre a rede física, aumenta em muito, a sua complexidade. A criação e manutenibilidade deste tipo de redes, requer de um *Sysadmin*, um conhecimento sólido das redes IP.

Outra competência muito importante é a gestão de utilizadores e a sua autenticação. Já começa a ser comum a existência de ambientes computacionais mistos, com servidores físicos e servidores virtuais a operar remotamente na *Cloud*. E apesar do processo de autenticação ser idêntico para os dois tipos, as dependências do próprio processo podem organizar-se de forma diferente, o que muda certamente a sua gestão.

Por último, há que referir o armazenamento. Qualquer *Sysadmin* que já administrou um serviço de cópias de segurança, está familiarizado com as várias topologias de armazenamento, bem como os problemas de conectividade e Quality of Service (QoS). Muitos operadores dos serviços *Cloud*, usam armazenamento por objectos, comparativamente ao armazenamento por blocos, o que implica o uso de um *Gateway* para efeitos de conectividade. Outra tendência que se verifica, é a utilização de técnicas como as Bunch of Redundant Independent Clouds (BRIC), por parte de uma só entidade colectiva, para prevenir a eventualidade de falhas. Não havendo qualquer incerteza relativamente à importância deste tópico, torna-se premente a compreensão aprofundada do armazenamento remoto.

Para concluir, a possibilidade de pôr em prática algumas técnicas de virtualização, que se adequam ao ambiente operacional do EPOS, é para mim uma oportunidade académica sem precedentes!

1.3 Objetivos

Este trabalho, tem o propósito de virtualizar o conjunto de aplicações usadas pelo ambiente EPOS. Por conseguinte, consoante o tipo e a quantidade de recursos que

cada aplicação necessita, é aplicada a técnica de virtualização mais apropriada, por forma a conseguir uma optimização do sistema que as suporta e, o melhor tempo de execução possível.

O ambiente operacional do EPOS, é constituído pelas seguintes aplicações:

- **database** é o Sistema de Gestão de Bases de Dados (SGBD) que utiliza a tecnologia *PostgreSQL* e, que possibilita o armazenamento da informação relativa às estações geodésicas, aos dados recolhidos e à qualidade da informação do sistema, numa base de dados chamada *GNSS-Europe*.

- **Products_Portal** é uma aplicação que utiliza a tecnologia *PHP* e é executada no servidor *Apache*. Serve de ponto de entrada para o acesso aos produtos associados às estações Global Navigation Satellite System (GNSS), que fazem parte da rede EPOS. Esta aplicação, incorpora dados e metadados disponibilizados pela Application Programming Interface (API) de outra aplicação chamada *EPOS_GLASS_Framework* e, por consequência, à base de dados *GNSS-Europe*.

- **EPOS_GLASS_Framework** é uma *Framework* que utiliza a tecnologia *Java* e o servidor de aplicações *GlassFish*. Esta ferramenta, trata de todos os pedidos que fornecem os dados e os metadados, ao *Data Gateway* e ao *Products Gateway*, usando a base de dados *GNSS-Europe*.

- **FWSS - Flask Web Service Server** é um serviço *Web* que utiliza a tecnologia *Python* e um servidor *Web*, o *Flask*. E tem como função, servir de protótipo/ferramenta de teste para os serviços do GNSS.

- **EPOS_Sync_System** é uma ferramenta que utiliza a tecnologia *Java* e, que trata da sincronização das tarefas realizadas entre os nodos da rede EPOS.

- **GLASS-web-client** é uma interface que utiliza a tecnologia *Java* e, é executada no servidor de aplicações *GlassFish*. Funciona como cliente *Web*, para a API da *EPOS_GLASS_Framework*.

- **NodeManager** é uma ferramenta que utiliza a tecnologia *NodeJS* e, é executada no servidor de aplicações *GlassFish*. Esta ferramenta, gere nodos, conexões entre nodos e centros de dados.

- **indexGD** é uma aplicação que utiliza a tecnologia *Python* e, tem a função de indexar dados geodésicos, que estejam contidos em ficheiros presentes numa determinada localização (directoria), enviando depois o resultado sob a forma de objectos JavaScript Object Notation (JSON), para o *FWSS - Flask Web Service Server*.

- **RunQC** é um Script que utiliza a tecnologia *Perl* e, que gera metadados sobre a qualidade dos dados provenientes do GNSS, no formato *XML*. Este Script recebe mensagens completas de navegação fidedignas, que são descarregadas automaticamente de várias fontes consolidadas. Estas mensagens são depois descompactadas, para depois configurar, iniciar e controlar o *G-Nut/Anubis*. Se necessário, o ficheiro *XML* que contém o resultado, é comunicado a um serviço presente num servidor *Web*, para, por sua vez, fazer o registo na base de dados.

Tendo em conta, para além dos requisitos de memória, todas as dependências que têm que existir no ambiente de execução de cada aplicação, em termos de *software*, é indicada na tabela 1.1, o tipo de virtualização a usar em cada caso.

Ambiente EPOS		
Aplicação	**Container**	**Unikernel**
database	✓	
Products_Portal	✓	
EPOS_Sync_System		✓
EPOS_GLASS_Framework	✓	
indexGD		✓
RunQC		✓
FWSS - Flask Web Service Server	✓	

Tabela 1.1: Atribuição das tecnologias de virtualização

1.4 Organização do Documento

De modo a refletir sucintamente todo o trabalho produzido neste projecto, este documento encontra-se estruturado em cinco capítulos, precedidos pela prestação de todos os agradecimentos e, sucedidos pelas referências bibliográficas usadas durante o seu desenvolvimento. O conteúdo de cada capítulo, pode ser resumido da seguinte forma:

1. O primeiro capítulo – **Introdução** – apresenta o projecto e faz uma breve descrição da instituição que o suporta. É também feito o enquadramento do tema, no contexto dessa mesma instituição, referindo a importância das tecnologias usadas. A motivação da escolha deste projecto, é baseada na contextualização da virtualização, na minha área de especialização profissional de eleição. São apresentados os objectivos propostos neste trabalho,

fazendo uma descrição das aplicações a virtualizar e dos respectivos ambientes de execução. Por último, é apresentada a organização do documento.

2. O segundo capítulo – **Estado de Arte** – descreve em detalhe, todas as tecnologias de virtualização usadas neste projecto, que estão disponíveis no mercado, actualmente.

3. O terceiro capítulo – **Tecnologias Utilizadas** – refere com especificidade, todas as ferramentas utilizadas, na aplicação das tecnologias escolhidas, para o cumprimento dos objectivos propostos. Este capítulo, refere também as razões dessas escolhas.

4. No quarto capítulo – **Implementação e Testes** – são mencionadas todas as características do equipamento utilizado ao nível do *hardware*. De seguida, é feita uma descrição detalhada de todas as implementações feitas para se chegar às soluções requeridas, bem como, a enunciação de todas as configurações utilizadas. Por fim, é também feita uma menção a todos os testes realizados, onde se relatam os obstáculos encontrados.

5. No quinto capítulo – **Conclusões e Trabalho Futuro** – é feita uma breve dissertação sobre a experiência proporcionada por este projecto e, é feita também referência, a alguma optimização que possa ser implementada no futuro.

Capítulo 2

Estado da Arte

2.1 Introdução

Nos últimos anos, a virtualização tem evoluído imenso, por parte não só dos fabricantes de *software*, que têm apresentado melhores soluções à custa de maior complexidade na sua implementação, como também por parte dos fabricantes de *hardware*, que têm permitido ultrapassar certos limites de desempenho existentes na própria área tecnológica.

Pela importância deste tópico e, porque todo o projecto se baseia nestas tecnologias, irão ser abordados com detalhe, todos os tipos de virtualização usados nos dias de hoje. Serão introduzidos na secção 2.2, todos os conceitos relativos às VMs, bem como os seus tipos e as suas características. Para respeitar a sequência cronológica do aparecimento destas soluções, na secção 2.3, será abordado o tema dos *Containers*. Por último, os *Unikernels* serão referidos na secção 2.4, visto serem o tópico mais recente.

2.2 Máquinas Virtuais

O mundo da virtualização assenta num conceito base que é o *Hypervisor* ou Virtual Machine Monitor (VMM). É o *Hypervisor* que comunica directamente com o *hardware* e que partilha os recursos do sistema, agendando tempo de processamento na CPU e alocando, entre outros recursos, memória para cada VM. É o *Hypervisor* que controla o acesso dos sistemas operativos convidados aos dispositivos de *hardware*. Actualmente, existem dois tipos de *Hypervisors*, por tal, irão ser descritos detalhadamente, para se perceber as diferenças conceptuais e funcionais que há entre eles.

2.2.1 Hypervisors Nativos

São referidos como *Hypervisors* do tipo 1 e foram desenvolvidos pela ©*IBM* na década de 60. Como são executados em modo privilegiado e, correm directamente sobre o *hardware*, este tipo de *Hypervisors* são muito eficientes e têm um excelente desempenho.

Arquitectura Micro-kernel

Esta arquitectura, está concebida por forma a minimizar o tamanho do *Hypervisor*. Só algumas funcionalidades como a gestão da memória e o escalonamento do processador, estão incluídas directamente no *Hypervisor*. Todas as outras funcionalidades como a gestão: dos *Drivers* dos dispositivos de *hardware*; da rede; dos dispositivos de armazenamento de dados; das *Hypercalls* e outras, são geridas numa partição de gestão separada, como se pode constatar na figura 2.1. Enquanto que o *Hypervisor* é executado em modo supervisor, esta partição é gerida por um sistema operativo independente, cujo ambiente de execução faz parte do modo utilizador. Ainda assim, esta partição é considerada como fazendo parte do Trusted Code Base (TCB), o que lhe confere o privilégio de aceder directamente ao núcleo e a recursos do sistema e, a incumbência de intermediar a comunicação entre as VMs e o *Hypervisor*.[Shropshire, 2014]

Para suportar a execução das VMs, o *Hypervisor*, captura os pedidos aos recursos de *hardware* e transfere-os para a partição de gestão. Esta, por sua vez, processa-os e encapsula-os através de uma camada virtualizada, antes de comunicar com os *Drivers* dos dispositivos de hardware.[Shropshire, 2014]

Arquitectura Monolítica

Neste modelo, todas as funcionalidades do sistema estão incluídas no *Hypervisor*. Tal inclui: o núcleo, todos os subsistemas, interfaces e *Drivers* dos dispositivos de *hardware*. Como consequência directa, o *Hypervisor* tem um tamanho maior, mas todo ele é executado em modo supervisor. O que implica que todo o seu TCB resida na área mais segura do sistema.[Shropshire, 2014]

Como referência hierárquica, as VMs são instanciadas logo acima do *Hypervisor* onde um VMM interage com cada uma das instâncias. O VMM está contido no *Hypervisor* e, aí recebe todos os pedidos aos recursos de *hardware*. Por último, é de referir que, na arquitectura monolítica, também é suportado o sistema de comunicação por *Hypercalls*.[Shropshire, 2014]

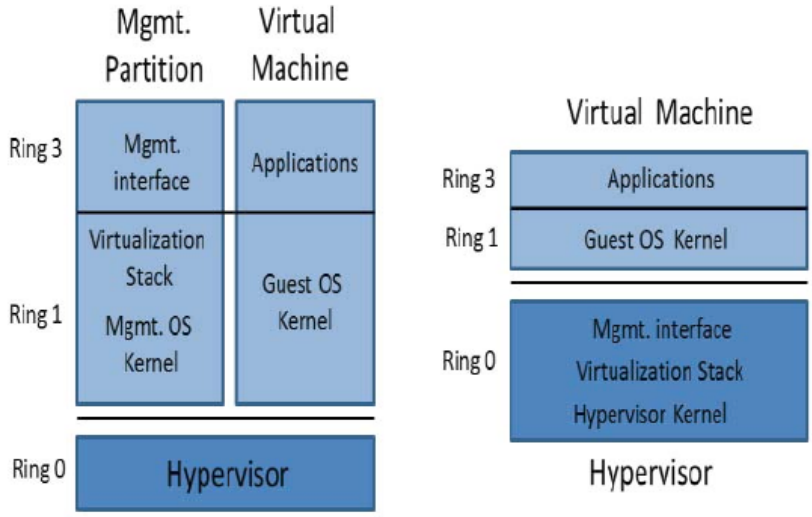

Figura 2.1: Arquitecturas: Micro-kernel (esquerda) e Monolítica (direita)

2.2.2 Hypervisors Alojados

Esta arquitectura, constitui os *Hypervisors* do tipo 2. Este *Hypervisor* é instalado como uma mera aplicação. Como todos os recursos são geridos pelo próprio sistema operativo anfitrião, o *Hypervisor*, partilha esses mesmos recursos com todas as outras aplicações existentes nesse sistema.

Um *Hypervisor* pertencente a esta arquitectura, pode ser instalado sem a necessidade da existência de qualquer alteração no sistema operativo anfitrião, aspecto este, que pode ser considerado uma vantagem na configuração de certos ambientes. Também devido a este facto, o *Hypervisor*, depende inteiramente do sistema operativo anfitrião para a facultação dos *Drivers* dos dispositivos de *hardware* e outros serviços de baixo nível.[Hwang et al., 2012]

Comparado com os *Hypervisors* nativos, o *Hypervisor* alojado, tem um desempenho notoriamente inferior. Tal acontece de forma expectável porque, quando uma aplicação virtualizada requer o acesso a algum dispositivo de *hardware*, é necessário fazer o mapeamento dos recursos através de quatro camadas distintas constituintes da própria arquitectura. Mais acentuada se torna a diferença de desempenho, quando a Instruction Set Architecture (ISA) do sistema operativo virtualizado, é diferente da ISA do *hardware* existente! O que obriga a uma tradução binária completa das instruções.[Hwang et al., 2012]

2.2.3 Virtualização Completa

Esta técnica de virtualização, também conhecida por Hardware Virtual Machine (HVM), não tem a necessidade de modificar o sistema operativo convidado porque, combina a tradução binária e a execução directa de instruções. O sistema operativo convidado, é completamente isolado do *hardware* do ambiente de execução. Consequentemente, este não tem a menor noção de que está a ser virtualizado. Tal feito, pode ser conseguido através de dois processos a seguir enunciados:

Tradução Binária/Emulação de Hardware

Neste caso, é feita a emulação da ISA que o sistema operativo convidado está à espera de encontrar, sobre a ISA presente no processador do ambiente anfitrião. Assim, é possível efectuar a captura de instruções priviligiadas, que não podem ser virtualizadas. Para tal, é usado um *software* que emula os dispositivos de *hardware* comuns, que são expostos a posteriori, ao sistema operativo convidado. Mais, os pedidos de entrada/saída de dados, são redireccionados para o *Hypervisor*, onde são traduzidos por pedidos de entrada/saída de dados, reais. Por outro lado, todas as instruções não críticas, são executadas directamente no *hardware*, não só promovendo a eficiência da virtualização, como também afirmando a segurança do sistema.[De Lucia, 2017]

Virtualização Assistida por Hardware

Para proporcionar um melhoramento adicional ao desempenho dos *Hypervisors* e, diminuir a complexidade da sua implementação, esta técnica foi desenvolvida, para disponibilizar aos métodos de virtualização, uma solução baseada no *hardware*. Esta solução, provoca um aumento no número de instruções disponíveis nos processadores afectados, o que torna possível as operações de virtualização, acontecerem directamente no hardware! O próprio *hardware*, está preparado para capturar certas chamadas ao sistema por parte das VMs em execução, o que permite ignorar os processos de tradução existentes na camada de virtualização e, enviar os pedidos de entrada/saída de dados directamente para o *Hypervisor*, que por sua vez, decide executá-los ou não.[De Lucia, 2017]

Em 2005, a ©*Intel*, foi pioneira na apresentação da sua solução de virtualização assistida por hardware nos seus processadores, com as tecnologias *VT-x* (*Virtual Machine Control Structures*) e *VT-d* (*Virtualization Technology for Directed I/O*). O novo conjunto de instruções destes processadores, permite a execução do *Hypervisor* no anel -1! O que significa que o sistema operativo convidado, pode continuar a ser executado em modo privilegiado, como se não existisse qualquer

processo de virtualização, ou seja, no anel 0. Este novo conjunto de instruções dedicadas à virtualização, permite ainda a gestão de memória por cada VM, de forma individualizada e, o acesso directo a um dispositivo de rede e outros.[De Lucia, 2017]

2.2.4 Para-virtualização

Com esta técnica de virtualização, o sistema operativo presente na VM é modificado e, tem a perfeita noção de que está a ser virtualizado. É instalado *software* adicional como: *Drivers* e ferramentas de sistema, que permitem a comunicação directa entre a VM e o *Hypervisor*. Os sistemas operativos convidados para-virtualizados, ficam munidos de um *Compilador* inteligente que, substitui as instruções criticas por *Hypercalls*, em tempo de compilação. Assim, com a para-virtualização, é reduzida substancialmente a carga computacional acrescida que deriva da própria virtualização e, como consequência, existe um ganho concernente ao desempenho do sistema.

2.3 Containers

Esta tecnologia, cria uma camada de virtualização dentro do sistema operativo, que permite particionar os recursos do próprio sistema físico. Assim, é possível a execução de múltiplas instâncias isoladas de um determinado sistema operativo, partilhando o núcleo desse mesmo sistema operativo. Uma instância desta natureza, é normalmente chamada de: *VEE* (*Virtual Execution Environment*); *VPS* (*Virtual Private System*); ou simplemente um *Container*. Na prática, temos um ambiente que se parece com o ambiente de uma VM, porque cada uma destas instâncias, tem: os seus próprios processos; sistema de ficheiros; contas de utilizadores; interfaces de rede com endereços Internet Protocol (IP) atribuídos; tabelas de *Routing*; regras de *Firewall* e outras configurações próprias, mas sem o peso nem o tempo de inicialização que caracterizam o uso da virtualização ao nível do *hardware*.[Hwang et al., 2012]

O conceito é simples e antigo, sendo o comando *chroot*, o precursor mais conhecido desta tecnologia. Com o *chroot*, é possível segregar os acessos de diretorias e evitar que o utilizador possa ter acesso à raiz do sistema ("/" ou *root*).

Para se ter um conhecimento aprofundado sobre este tema, é aconselhável abordar conceitos relativos ao sistema *Linux* como os *cgroups* e os *namespaces*. Tais conceitos, representam funcionalidades inerentes ao próprio núcleo do sistema, que permitem criar o isolamento necessário, entre o *Container* e os outros processos que são executados concorrentemente nesse mesmo sistema.

Os *namespaces*, desenvolvidos pela ©*IBM*, conglomeram um conjunto de recursos existentes no sistema, que são disponibilizados a um processo, como se fossem recursos dedicados única e exclusivamente a esse mesmo processo.

Os *cgroups*, desenvolvidos pela ©*Google*, criam o isolamento e limitam a utilização dos recursos presentes no sistema, como o processador e a memória, para grupos de processos que necessitam desses recursos.

Em resumo, os *namespaces* tratam do isolamento dos recursos existentes para um único processo, enquanto que os *cgroups* gerem esses mesmos recursos para um grupo de processos.[Hwang et al., 2012]

2.4 Unikernels

Esta tecnologia, representa uma nova abordagem na construção de núcleos de sistemas operativos, especializados, que contêm somente o código da aplicação, o ambiente de execução e as respectivas dependências. Assim, o código da aplicação é fundido com o código do núcleo do sistema e, o *Exokernel* gerado pode ser depois executado num *Hypervisor*. Tal como se pode perceber na figura 2.2 O principal objectivo desta tecnologia, é a redução máxima do tamanho do *Exokernel* criado. Como consequência, o desempenho relativo às comunicações em rede é comprovadamente melhor e, o tempo de inicialização destas instâncias é medido em milissegundos!

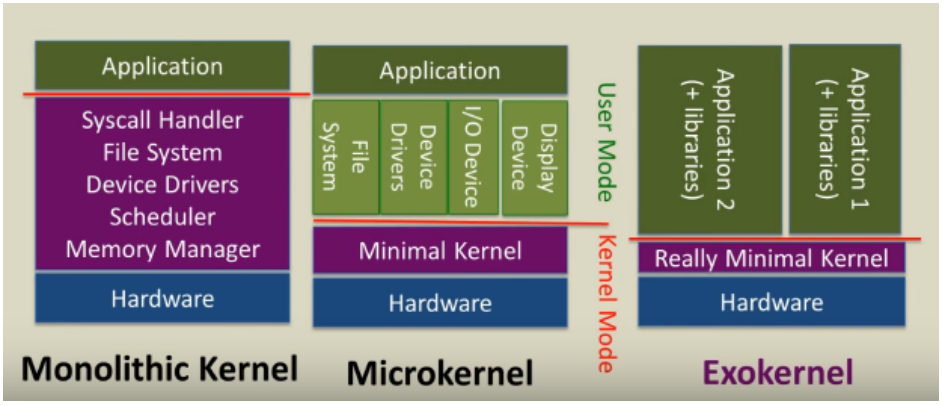

Figura 2.2: Tipos de núcleos presentes nos *hypervisors* e nos *unikernels*

O melhor desempenho relativamente às comunicações em rede, é atribuído em parte, ao facto da aplicação incluída no *Unikernel*, ser executada no mesmo espaço de endereçamento dos módulos constituintes do núcleo do sistema, alijando assim a mudança dos contextos de execução, entre a aplicação e o respectivo núcleo. No conceito do *Unikernel*, a delineação entre o modo utilizador e o modo supervisor,

não existe! Logo, tanto a aplicação como os módulos constituintes do núcleo, são executados com os mesmo privilégios. Ainda, a possibilidade da existência de ataques é amplamente reduzida, porque os serviços e o código relativo a bibliotecas, desnecessários à execução da aplicação, são removidos, deixando somente as dependências necessárias do núcleo do sistema. E o isolamento existente entre várias instâncias de *Unikernels*, é garantido pelo *Hypervisor*.[De Lucia, 2017]

A filosofia dos *Unikernels*, é a execução de uma única aplicação ou serviço, correspondente a um só processo. Por tal, se for necessária a execução de múltiplas instâncias em paralelo, terão que ser iniciadas múltiplas instâncias do *Unikernel*, pelo *Hypervisor*. Logo, as restrições relativas ao uso e agendamento dos recursos de *hardware*, são geridas pelo *Hypervisor*.[De Lucia, 2017]

2.5 Conclusões

É interessante perceber pela figura 2.3, as grandes diferenças conceptuais, entre as várias tecnologias existentes actualmente. E são estes conceitos que vão servir de base de reflexão, para a escolha do modelo a utilizar, na concretização dos objectivos deste projecto.

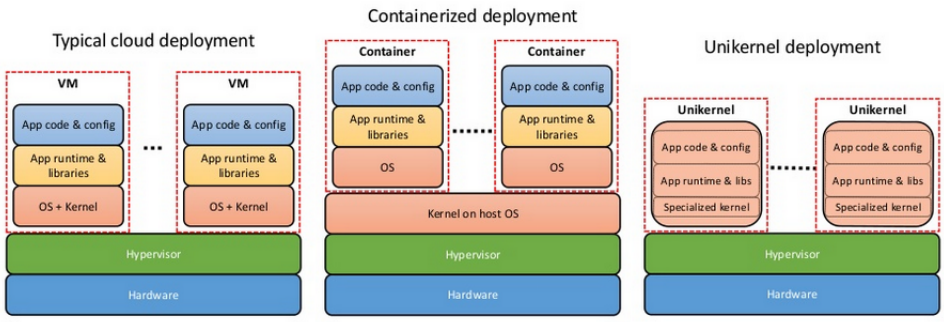

Figura 2.3: Arquitecturas: Diferenças conceptuais

Capítulo 3

Tecnologias e Ferramentas Utilizadas

3.1 Introdução

No que concerne às ferramentas tecnológicas disponíveis no mercado, analisando as múltiplas possibilidades de escolha a usar na virtualização do ambiente EPOS, foram tomados alguns critérios, por forma a facilitar o uso e o apoio dessas mesmas ferramentas, considerando o cenário de atecnia e descoberta, característico desta fase académica. Esses critérios foram: o custo e a aceitação tecnológica. Por um lado, analisando as possibilidades credíveis e conceituadas, relativas às soluções *Open Source*, para assim se evitar um custo pecuniário na aquisição de licenças. Por outro lado, para que haja alguma facilidade em encontrar apoio e respostas, nos canais públicos de informação, por forma a solucionar rapidamente, todos os obstáculos vislumbrados.

Para a gestão das VMs, a escolha recaiu no *Xen*, descrito na secção 3.2, por cumprir os critérios definidos, por ser um *Hypervisor* leve, rápido e eficiente, e por ser uma experiência que considero proveitosa, na medida em que é amplamente usado no mundo empresarial.

Para a virtualização das aplicações, proposta inicialmente com o uso de *Containers*, a opção favorável será o *Docker*, detalhado na secção 3.3, mais uma vez pela condição dos critérios definidos, por ser seguramente a tecnologia mais evoluída do mercado e, mais uma vez também, por me proporcionar uma experiência enriquecedora, visto ter uma aceitação colossal no mundo empresarial.

Para a a criação dos *Exokernels*, com base nas aplicações que têm uma funcionalidade mais simples e, que parecem enquadrar-se na utilização desta tecnologia, foi eleita a ferramenta *Unik*, esmiuçada na secção 3.4, que para além de estar

relativamente bem documentada, tem a característica particular, de incorporar o funcionamento dos principais projectos em desenvolvimento na área.

3.2 Xen

Foi instalada no Servidor do EPOS, a **versão 4.9.0** deste magnífico *software*, que cria um *Hypervisor* do tipo 1, com arquitectura micro-kernel, com suporte à para-virtualização e à virtualização completa. O *Xen*, tem também a capacidade de usar um modo especial de virtualização, chamado Para-virtualisation on HVM (PVHVM), que combina a HVM com o uso de vários dispositivos para-virtualizados, como o armazenamento, a rede e outros. Este modo misto de virtualização, oferece o melhor dos dois mundos, por reduzir a carga computacional que deriva da emulação dos dispositivos e, ao mesmo tempo, proporciona uma aceleração do *hardware*, relativamente ao processamento e ao acesso à memória. O *Xen* é um projecto *Open Source*, desenvolvido na Universidade de Cambridge.

As VMs criadas e executadas pelo *Xen*, são chamadas de domínios. Existe um domínio especial chamado *dom0*, que é executado em modo privilegiado logo quando o *Xen* é inicializado no sistema e, que tem a incumbência de controlar o *Hypervisor* e inicializar outro tipo de domínios, esquematizados na figura 3.1 à direita, chamados *domU*. Os domínios pertencentes a este tipo, são executados em modo não privilegiado, por não terem as incumbências do *dom0*. O *dom0*, consegue comunicar com os *domU*s através de um canal de memória contido no *Hypervisor*. E, também através do *Hypervisor*, consegue alocar e mapear recursos de *hardware* para os *domU*s. Também, os *Drivers* necessários para aceder aos dispositivos de *hardware*, estão contidos no *dom0*. Por esta razão, o tamanho do *Hypervisor* é relativamente pequeno.[Xen.org, 2018]

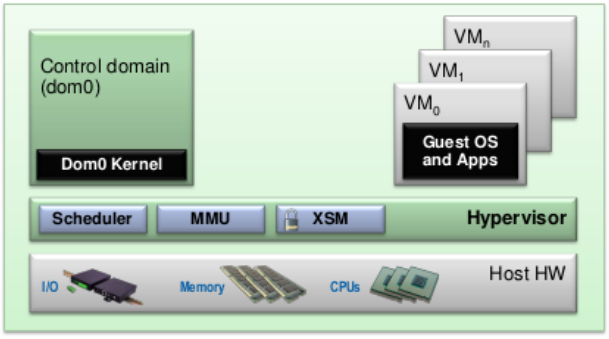

Figura 3.1: Arquitectura Xen

Cada *Datapath* para-virtualizado, está separado em dois blocos: o *backend* que reside no *dom0* e, que gera o dispositivo virtual e; o *frontend* que reside num *domU* e, que permite ao sistema operativo convidado aceder ao dispositivo virtual. Ambos os blocos, usam uma interface de alto débito baseada em memória partilhada, para a transferência de dados entre o *dom0* e o respectivo *domU*. Os *Datapaths* mais importantes, são: o *net-back/net-front* para o sistema de rede e; o *blk-back/blk-front* para o sistema de armazenamento de dados.[Xen.org, 2018]

3.3 Docker

O *Docker*, é uma plataforma *Open Source*, de desenvolvimento, provisionamento e execução de aplicações que, tem como base, a linguagem de programação *Go* da ©*Google*. Cada aplicação virtualizada é tratada como um micro serviço, que pode a posteriori, fazer parte de uma aplicação maior.

Para todos os experimentos envolvidos neste projecto, foi instalada e usada, a **versão 18.03.1** respeitante à edição Community Edition (CE).

No que concerne à sua arquitectura, foi feito o desenvolvimento e a implementação de uma biblioteca chamada *libcontainer*, que providencia um ambiente de execução transversal a todas as distribuições de *Linux*. No entanto, ao longo do seu desenvolvimento, foram adicionadas várias camadas de abstracção, para haver um maior entrosamento com o grande ecossistema *Open Source* e, para ir de encontro aos padrões do sector.[Inc., 2018]

Como se pode visualizar na figura 3.2, no ambiente de execução do *Docker*, o componente que inicia os processos de: criação; execução e eliminação de imagens, é o *Docker-Engine*, que recebe os pedidos gerados no Command-line Interface (CLI) e os comunica ao *containerd* através de uma API, para que sejam depois processados.

Por sua vez, o *containerd*, é o *Daemon* que trata de: todas as tarefas de baixo nível relativas à gestão das imagens; do armazenamento; da distribuição das imagens; das conexões à rede e outras.

Por último, o *runC* é o módulo que efectivamente executa as instâncias das imagens e, trata da interface de comunicação de baixo nível, associada ao próprio núcleo do sistema, no que concerne entre outros, aos *cgroups* e aos *namespaces*.

Quando as instâncias em execução, são terminadas por alguma razão, o *containerd-shim*, é o módulo que se certifica de terminar todos os processos e subprocessos, ainda presentes em memória.[Inc., 2018]

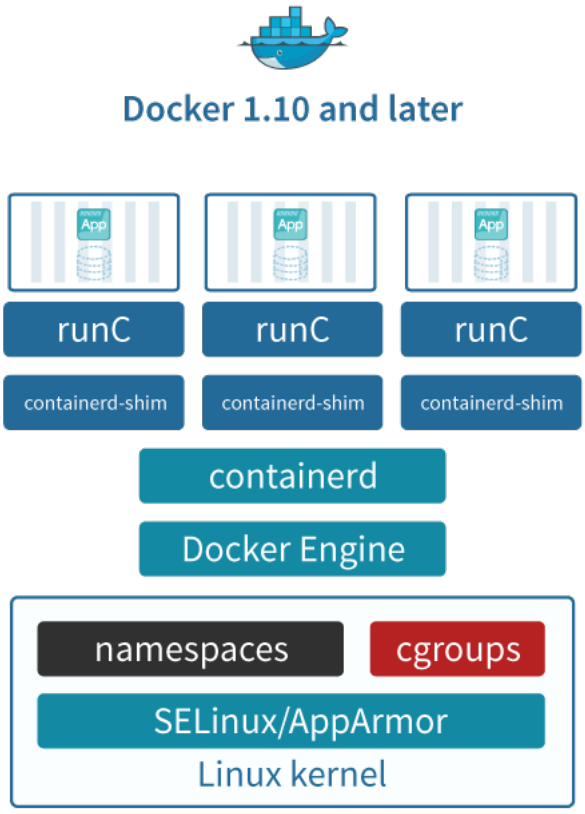

Figura 3.2: Arquitectura Docker

Outro aspecto muito importante a referir, é o uso do Another Union File System (AUFS) como sistema de ficheiros, para a criação das imagens por camadas, só de leitura e de forma incremental. As diferentes camadas, que podem ter diferentes sistemas de ficheiros, são mantidas e geridas pelo *Driver* de armazenamento de dados, que possibilita a sua partilha por diferentes imagens. Partilha essa que permite operações como: o *build*, o *copy*, o *pull* e o *push*, de uma forma rápida e compacta.

Na criação de cada imagem, é gerada uma pequena camada de escrita e, todas as alterações efectuadas nessa imagem são aí guardadas. O que significa que várias imagens podem partilhar o acesso à mesma camada de escrita e ao mesmo tempo, ter o seu próprio estado.

O *Docker*, por defeito, usa a tecnologia Copy on Write (CoW). Esta tecnologia, permite uma excelente optimização, tanto no tamanho das imagens, como no tempo de inicialização dos próprios *Containers*.[Inc., 2018]

3.4 Unik

Esta ferramenta, ainda na sua **versão 0.0.0**, está numa fase experimental e, é na sua essência, uma plataforma de compilação e instalação de *Unikernels*. Aliás, a plataforma mais completa do mercado, por ser a única a suportar a criação de vários tipos de *Unikernels*, de uma forma sólida e completamente funcional. É de referir, que também consegue interagir com vários *Hypervisors* de diferentes arquitecturas.[solo.io, 2018]

O *Daemon Unik*, é constituído por uma API, que trata pedidos provenientes da CLI ou de algum cliente HyperText Transfer Protocol (HTTP) e, que determina, qual o *Compilador* e o *Hypervisor* a usar para processar esses mesmos pedidos.

Em particular, quando a API recebe um pedido *build*, o compilador apropriado, é chamado para construir a imagem generalizada, que por sua vez é passada ao *Hypervisor* escolhido, para em seguida a processar com o método *stage*, transformando-a numa imagem auto-inicializável e optimizada para o referido *Hypervisor*.[solo.io, 2018]

3.5 Conclusões

Para além de todas as funcionalidades proporcionados por estas ferramentas, que permitem a optimização dos recursos físicos, importa salientar alguns aspectos como a segurança, fundamental para o bom funcionamento e a integridade das aplicações e respectivos ambientes virtualizados. A tabela 3.1 enfatiza as diferenças mais relevantes entre estas tecnologias, na sua execução.

Tipo	**Aplicação**	**Nível de Privilégio**	**Proveniência do Isolamento**	**Tamanho da Imagem**
Máquina Virtual	Xen	Nível 0 ou -1	Hypervisor	Grande
Container	Docker	Nível 3 forçado ao Nível 0	Kernel	Médio
Unikernel	Unik	Nível 0	Hypervisor	Pequeno

Tabela 3.1: Diferenças entre os vários Ambientes de Execução

O maior nível de isolamento criado entre instâncias virtualizadas, é indubitavelmente proporcionado pelo *Hypervisor*, que consegue utilizar de forma mais optimizada, a segmentação da memória física pelos diferentes níveis de privilégios de processamento e, por extensões de virtualização do *hardware*.

Capítulo 4

Implementação e Testes

4.1 Introdução

Após toda a reflexão e ponderação dos múltiplos factores a considerar, chega o momento da preparação do ambiente, com todas as: instalações; configurações e optimizações, necessárias ao bom funcionamento do sistema EPOS.

Relativamente ao Servidor, são mencionadas detalhadamente na secção 4.2, todas as suas características físicas, referindo-se também o sistema operativo e o *Hypervisor*, fundamentais ao desempenho das suas funções. De seguida, na secção 4.3, é feita uma descrição pormenorizada, de todas as configurações e optimizações feitas, tanto no próprio sistema, como nas ferramentas de virtualização usadas. Passando para a secção 4.4, são demonstradas todas as implementações, relativas ao processo de virtualização das aplicações do ambiente EPOS. Decorrido todo esse processo, na secção 4.5, são efectuados todos os testes e todas as comparações, para análise dos resultados obtidos, com os processos de virtualização aplicados. Por último, é feita na secção 4.5, uma breve referência às limitações tecnológicas encontradas, nas ferramentas usadas em todo o percurso.

4.2 Características do Servidor

Para a feitura deste projecto, foi disponibilizado um *miniPC* da marca ©*A1 Touch*, com as características perfeitas para este tipo de função. É um sistema com uma capacidade de processamento considerável, compacto e amigo do ambiente. Outro aspecto importante a referir, baseia-se no facto de ser perfeito para o caso de haver a necessidade de efectuar um escalamento dos recursos virtualizados. Uma *Cluster* local composta por unidades iguais deste sistema, tornar-se-ia perfeito para a aplicação da funcionalidade *Swarm* existente no *Docker*. É feita uma

descrição detalhada dos componentes do sistema disponibilizado, na tabela 4.1.

Servidor	
CPU	Intel i3-4030U 1.9Ghz 2 Cores 4 Threads 3MB L2 Cache
Memoria	4096 MB DDR3 1333MHz
Armazenamento	120GB SSD
Rede	2x 1Gbit
Sistema Operativo	Debian 9.4
Hypervisor	Xen 4.9

Tabela 4.1: Características do Servidor

4.3 Configurações do Sistema

O *Xen*, é um *Hypervisor* que requer algumas configurações para que funcione de forma optimizada.

Relativamente à memória, sendo um recurso limitado neste sistema, é recomendada a atribuição da sua totalidade ao *dom0*, para que este, possa de forma dinâmica, ceder este recurso aos *domU*s, à medida que as VMs sejam instanciadas. No entanto, tendo a perfeita noção da quantidade de *VM*s que irão ser executadas no sistema e, porque com o fenómeno da reatribuição da memória, existe um decréscimo de desempenho na rede e na inicialização destas *VM*s, optou-se por atribuir **1GB** ao *dom0*, que será suficiente para que este desempenhe as suas funções.

No que concerne à capacidade de processamento, exactamente pelas mesmas razões e com os mesmos argumentos relativamente ao desempenho, optou-se por atribuir **1 vCPU**, dos quatro criados pelo *Hypervisor*, para gerir os 2 CPUs físicos do sistema.

Quanto à configuração e gestão da rede do Servidor, seguindo também as recomendações indicadas pela comunidade utilizadora do *Xen*, foi criado um dispositivo virtual em modo *bridge*, para possibilitar a todos os cliente virtuais e físicos do sistema, a conexão à rede. Na listagem 4.1, é apresentado o ficheiro de sistema do *Debian*, com toda a configuração necessária para o funcionamento na rede do EPOS.

Listing 4.1: Configuração da rede no *Servidor*

```
# This file describes the network interfaces available
    on your system
```

```
# and how to activate them. For more information, see
   interfaces(5).

source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
allow-hotplug enp3s0
iface enp3s0 inet manual

# The bridge interface
auto xenbr0
iface xenbr0 inet static
        bridge_ports enp3s0
        address 10.0.7.100
        netmask 255.255.255.0
        broadcast 10.0.7.255
        gateway 10.0.7.1

bridge_stp off
bridge_waitport 0
bridge_fd 0
```

No que concerne à organização do espaço disponível em disco do Servidor, foi usado o LVM para fazer essa gestão. Também seguindo as boas práticas da administração de sistemas, foram criados vários volumes lógicos dentro do mesmo grupo, para reforço da segurança do próprio Servidor e, para necessidades futuras de expansibilidade. É mostrado na tabela 4.2, um pequeno resumo dos volumes criados.

Grupo dos volumes: epos		
Volume lógico	**Caminho**	**Tamanho**
root	/dev/epos/root	18.62GiB
home	/dev/epos/home	18.62GiB
swap	/dev/epos/swap	3.72GiB
vmCentOS1	/dev/epos/vmCentOS1	10.00GiB

Tabela 4.2: Detalhes do LVM no Servidor

Com o objectivo de criar um ambiente de comunicação entre as entidades físicas e lógicas, o mais homogéneo possível e, respeitando as características do próprio ambiente EPOS, foi feita a configuração de rede da VM que suporta os *Containers*, tal como é mostrado na listagem 4.2.

Listing 4.2: Configuração da rede na VM *CentOS 7*

```
# Generated by dracut initrd
TYPE=Ethernet
bOOTPROTO=none
IPADDR=10.0.7.200
PREFIX=24
GATEWAY=10.0.7.1
DNS1=8.8.8.8
DNS2=8.8.4.4
DEFROUTE=yes
IPV4_FAILURE_FATAL=no
NAME=eth0
UUID=ea8abb5f-ea72-4422-95a7-89abd5ceffa9
DEVICE=eth0
ONBOOT=yes
```

Para a configuração da ferramenta de criação de imagens *Unikernel*, foi necessária a criação de um ficheiro com o nome ***daemon-config.yaml***, que permite ao respectivo *Daemon*, saber qual o *Hypervisor* que está associado ao sistema hospedeiro, para que o *Unik* possa criar os *Exokernels*. A respectiva configuração está descrita na listagem 4.3.

Listing 4.3: Configuração do *Daemon* do *Unik*

```
providers:
  xen:
    - name: my-xen
      xen_bridge: xenbr0
      pv_kernel: /boot/xen/grub-x86_64-xen.bin
version: 0.0.0
```

4.4 Detalhe das Implementações

Com o propósito de alojar as imagens das aplicações do sistema EPOS, no formato *Docker*, foi criada uma *VM* com o sistema operativo *CentOS* versão 7, caracterizada pela listagem 4.4. Como se pode constactar, esta VM irá ter **512MB** de

memória e **2 vCPUs** alocados pelo *Xen*, que se entende de momento, ser o suficiente para executar uma instância de cada aplicação do referido sistema. Também se pode verificar, que está a ser atribuído um volume lógico de armazenamento, enunciado na tabela 4.2. Por último, é de referir que, a última instrução presente na listagem supracitada, dá indicação de que esta VM irá ser para-virtualizada.

Listing 4.4: *Script* de criação da VM *CentOS 7* no *Xen*

```
name="centos7-VM"
vcpus=2
memory=512
disk=['phy:/dev/epos/vmCentOS1,xvda,rw']
vif=['bridge=xenbr0']
on_poweroff="destroy"
on_reboot="restart"
on_crash="restart"
bootloader="pygrub"
```

De seguida, é feita uma apresentação dos *Scripts* de criação das imagens, usados pelo *Docker*. Estes *Scripts*, são de certa forma auto-explicativos, no entanto, é feita uma síntese do significado das palavras chave, para futura referência:

- **FROM** indica a imagem base, que será descarregada e usada para a construção da imagem descrita no *Script*.
- **ADD** copia o ficheiro descrito para a localização referenciada, se não for um ficheiro compactado. Se for, descompacta o seu conteúdo e, move esse mesmo conteúdo para o destino referenciado.
- **LABEL** é usado para documentação e conveniência do criador do *Script*.
- **RUN** executa um comando durante a criação da imagem.
- **COPY** copia o ficheiro descrito para a localização referenciada.
- **VOLUME** indica uma directoria, cujo conteúdo será salvo e mantido, mesmo depois do *Container* ser terminado.
- **EXPOSE** expõe o porto de comunicação descrito.
- **CMD** executa um comando, quando a imagem é instanciada.
- **WORKDIR** define uma directoria como referência de ponto de entrada, quando a imagem é instanciada.

Listing 4.5: *Dockerfile* da aplicação: *database*

```
# CentOS 7.4.1708 base image
FROM scratch
ADD centos-docker.tar.xz /
LABEL name="CentOS Base Image" \
      vendor="CentOS" \
      license="GPLv2" \
      build-date="20170911"
RUN yum update -y && \
# PostgreSQL 10.2 installation
    rpm -Uvh https://yum.postgresql.org/10/redhat/rhel
       -7-x86_64/pgdg-centos10-10-2.noarch.rpm && \
    yum install -y postgresql10-server \
                   postgresql10 && \
    yum clean all && rm -rf /var/cache/yum
# Needed configuration scripts
COPY systemctl.py /usr/bin/systemctl
COPY gnss-europe.pgdump.sql \
     user.sql \
     NodeManager_sql_fix.sql /tmp/
# Server initialization
RUN /usr/pgsql-10/bin/postgresql-10-setup initdb
# Users configuration file
COPY postgresql.conf pg_hba.conf /var/lib/pgsql/10/
   data/
RUN chown postgres:postgres /var/lib/pgsql/10/data/
   postgresql.conf /var/lib/pgsql/10/data/pg_hba.conf
   && \
    systemctl start postgresql-10.service && \
    systemctl enable postgresql-10.service && \
    su - postgres -c 'psql -f /tmp/gnss-europe.pgdump.
       sql' && \
    su - postgres -c 'psql -c "CREATE DATABASE
       products_queue;"' && \
    su - postgres -c 'psql -d "gnss-europe-v0-2-14" -f
        /tmp/user.sql' && \
    su - postgres -c 'psql -d "products_queue" -f /tmp
       /user.sql' && \
    su - postgres -c 'psql -d "gnss-europe-v0-2-14" -f
        /tmp/NodeManager_sql_fix.sql'
```

```
# Environment setup
VOLUME ["/var/lib/pgsql/10/data"]
EXPOSE 5432
CMD systemctl start postgresql-10.service
```

Listing 4.6: *Dockerfile* da aplicação: *EPOS_GLASS_Framework*

```
# CentOS 7.4.1708 base image
FROM scratch
ADD centos-docker.tar.xz /
LABEL name="CentOS Base Image" \
      vendor="CentOS" \
      license="GPLv2" \
      build-date="20170911"
RUN yum update -y && \
# Java OpenJDK 8 installation
    yum install -y java-1.8.0-openjdk \
                   java-1.8.0-openjdk-devel && \
    yum clean all && rm -rf /var/cache/yum
# GlassFish 5.0 installation
ADD glassfish.tar.xz /opt/glassfish5/
# Needed files
COPY database.properties /opt/glassfish5/glassfish/
   domains/domain1/config/
# Glass Framework & Glass Web Client & Node Manager
   installation
COPY GlassFramework.war \
     epos_validation.war \
     NodeManager.war /opt/glassfish5/glassfish/domains
        /domain1/autodeploy/
# Environment setup
EXPOSE 80 8080
CMD /opt/glassfish5/glassfish/bin/asadmin start-domain
    --verbose=true
```

Listing 4.7: *Dockerfile* da aplicação: *Products_Portal*

```
# CentOS 7.4.1708 base image
FROM scratch
ADD centos-docker.tar.xz /
```

```
LABEL name="CentOS Base Image" \
      vendor="CentOS" \
      license="GPLv2" \
      build-date="20170911"
RUN yum update -y && \
# PHP 7.2 installation
## Apache 2.4.6 also needed and installed as a PHP
   depedency
    yum install -y https://dl.fedoraproject.org/pub/
       epel/epel-release-latest-7.noarch.rpm \
                   http://rpms.remirepo.net/enterprise
                      /remi-release-7.rpm && \
    yum install -y yum-utils && \
    yum-config-manager --enable remi-php72 && \
    yum install -y cronie \
                   php \
                   php-pecl-mcrypt \
                   php-cli \
                   php-gd \
                   php-curl \
                   php-ldap \
                   php-zip \
                   php-fileinfo \
                   php-opcache \
                   php-process \
                   php-xml \
                   php-mbstring \
                   php-pgsql && \
    yum clean all && rm -rf /var/cache/yum
# Server files
RUN rm -rf /var/www
ADD Products_Portal.tar.xz /var/www/
# cron job
COPY systemctl.py /usr/bin/systemctl
COPY crontab /etc/cron.d/requests-cron
# Environment setup
RUN chown -R apache:apache /var/www && \
    chmod 0644 /etc/cron.d/requests-cron
EXPOSE 443
CMD php /var/www/artisan key:generate && \
    php /var/www/artisan config:cache && \
```

```
    php /var/www/artisan migrate && \
    systemctl start crond.service && \
    systemctl enable crond.service && \
    apachectl -DFOREGROUND
```

Listing 4.8: *Dockerfile* da aplicação: *FWSS - Flask Web Service Server*

```
# CentOS 7.4.1708 base image
FROM scratch
ADD centos-docker.tar.xz /
LABEL name="CentOS Base Image" \
      vendor="CentOS" \
      license="GPLv2" \
      build-date="20170911"
RUN yum update -y && \
# Pip installation
    yum install -y epel-release
RUN yum install -y python-pip && \
    yum clean all && rm -rf /var/cache/yum && \
    pip install --upgrade pip && \
# Flask installation
    pip install flask flask-cors sqlalchemy psycopg2-
       binary requests xmltodict
# Server files
ADD FWSS.tar.xz /opt/fwss/
# Environment setup
EXPOSE 5555
WORKDIR /opt/fwss
CMD python web_server.py
```

Por último, aproveitando uma ferramenta de automação incluída no *Docker*, chamada *Docker-Compose*, foi criado um *Script* global de instanciação de imagens, que permite a iniciação de um conjunto de *Containers* em simultâneo, com o comando:

docker-compose up -d

onde a opção ***-d***, permite que as instâncias permaneçam em execução, em pano de fundo.

Como se pode verificar, este *Script*, dá a possibilidade de definir a criação de uma rede de comunicações e de um volume, que serão usados pelo conjunto das instâncias. É de referir também que, para qualquer sobreposição de regras, entre este *Script* global e os *Scripts* individuais das imagens, prevalecerá a regra presente neste *Script*, apresentado na listagem 4.9.

Listing 4.9: *Script* do *Docker-Compose*

```
version: '3'
services:
  epos-glass:
    build: ./EPOS_GLASS_Framework
    container_name: glass
    restart: on-failure
    ports:
      - 8080:8080
      - 80:80
    networks:
      epos-network:
        ipv4_address: 10.0.7.204
    depends_on:
      - epos-postgres
  epos-flask:
    build: ./Flask_Web_Service_Server
    container_name: flask
    restart: on-failure
    ports:
      - 5555:5555
    networks:
      epos-network:
        ipv4_address: 10.0.7.203
    depends_on:
      - epos-postgres
      - epos-glass
  epos-portal:
    build: ./Products_Portal
    container_name: products
    restart: on-failure
    ports:
      - 443:443
    networks:
      epos-network:
```

```
        ipv4_address: 10.0.7.202
    depends_on:
      - epos-postgres
      - epos-glass
  epos-postgres:
    build: ./Data_Base
    container_name: database
    restart: on-failure
    ports:
      - 5432:5432
    networks:
      epos-network:
        ipv4_address: 10.0.7.201
    volumes:
      - epos-volume:/var/lib/pgsql/10/data

networks:
  epos-network:
    driver: macvlan
    driver_opts:
      parent: eth0
    ipam:
      config:
        - subnet: 10.0.7.0/24

volumes:
  epos-volume:
```

Para a criação dos *Exokernels*, o método disponibilizado pela ferramenta *Unik*, é baseado em comandos executados em CLI. Comandos esses que serão descritos de seguida.

A aplicação ***EPOS_Sync_System***, por ser uma aplicação criada em *Java*, representa um exemplo ideal para ser executado com um *Unikernel*. Para efectuarmos a criação da sua VM, começamos por inicializar o *Daemon* do *Unik* com o comando:

unik daemon

De seguida, referência-se o sistema hospedeiro, que servirá de palco de operações, com o comando:

unik target –host localhost

Para o *Java*, teremos que integrar o código da aplicação num projecto *Maven* e, incluir no ficheiro ***pom.xml*** do projecto, a referência a alguns *Plugins*. Também teremos que criar um manifesto nesse mesmo projecto, com indicação de como construir o ficheiro executável final.

Depois, situados na directoria principal do projecto, iniciamos a construção do *Exokernel*, com o comando:

unik build –name EPOS_Sync_System –path ./ –base osv –language java –provider xen

Quando a imagem estiver criada, ela pode ser instanciada com o comando:

unik run –instanceName Sync –imageName EPOS_Sync_System

Por último, será atribuído um IP a essa instância e, podemos aceder-lhe por HTTP, no referido IP em conjunto com o porto **4000**.

No caso da aplicação ***indexGD***, por ser uma aplicação criada em *Python*, o método é ligeiramente diferente, mas que será descrito na íntegra, para comodidade do leitor.

Começamos por inicializar o *Daemon* do *Unik* com o comando:

unik daemon

De seguida, referência-se o sistema hospedeiro, que servirá de palco de operações, com o comando:

unik target –host localhost

Para o *Python*, teremos que criar um manifesto junto do código fonte, com indicação do ficheiro executável final.

Depois, situados na directoria principal do projecto, iniciamos a construção do *Exokernel*, com o comando:

unik build –name indexGD –path ./ –base rump –language python –provider xen

Quando a imagem estiver criada, ela pode ser instanciada com o comando:

unik run –instanceName index –imageName indexGD

Por último, será atribuído um IP a essa instância e, podemos aceder-lhe por HTTP, no referido IP em conjunto com o porto **4000**.

4.5 Testes e Comparações

Por forma a compreender as diferenças de desempenho e de consumo de recursos, entre as diferentes tecnologias, foram efectuadas algumas medições, que mostraram valores algo inesperados e, que são apresentados na tabela 4.3.

Imagem	Tamanho	Memória	Tempo
Máquina virtual			
CentOS 7	2356MB	119,00MB	16950ms
Unikernel			
EPOS_Sync_System	88,01MB	–	3391ms
indexGD	120,10MB	–	3829ms
Container			
database	492,00MB	11,02MB	476ms
EPOS_GLASS_Framework	727,00MB	460,30MB	607ms
Products_Portal	894,00MB	9,82MB	301ms
FWSS - Flask Web Service Server	525,00MB	78,80MB	320ms

Tabela 4.3: Utilização de recursos

4.6 Limitações Tecnológicas

Há que referir dois pontos importantes, que foram razão de grande desgaste cognitivo e, de forma associada, significaram muitos dias de estagnação, relativamente ao desenvolvimento deste trabalho!

Em primeiro lugar, evidencia-se o facto de constar no plano de acção inicial, a instalação do *CentOS 7*, como sistema operativo de apoio ao *Xen*, no servidor do ambiente EPOS. Tal não se mostrou possível, por razões que se prendem com uma enorme incompatibilidade entre estes dois sistemas. E pelo que foi possível apurar, tal incompatibilidade é intencional, na medida em que a ©*RedHat* tem o seu próprio *Hypervisor*, chamado *RHEV-H* e, por conseguinte, não é de seu interesse, promover um bom entrosamento entre as referidas tecnologias!

Em segundo lugar, existia o plano inicial de virtualizar a aplicação ***RunQC***, com a tecnologia *Unikernel*. Tal também não se mostrou fazível, pelo facto da referida tecnologia, ainda não suportar a linguagem de programação *Perl*, que concebe a aplicação mencionada.

4.7 Conclusões

Por observação directa da tabela 4.3, é possível perceber que, por um lado, o *Docker*, talvez pelo facto de ter adquirido em 2016, a *Unikernel Systems*, uma empresa de desenvolvimento da tecnologia de *Unikernels*, tem desempenhos, neste caso concreto, superiores ao dos *Unikernels*!

Por outro lado, talvez pelo facto dos casos práticos ideais para o uso dos *Unikernels*, serem muito específicos e, por consequência, não estar a haver a adesão em massa que teve o *Docker*, o seu desenvolvimento está um pouco estagnado no tempo e, as ferramentas que suportam a tecnologia, não são recomendadas em ambientes de produção.

Capítulo 5

Conclusões e Trabalho Futuro

5.1 Conclusões Principais

Foi muito interessante efectuar este estudo, relativamente aprofundado, sobre todas estas técnicas e ferramentas, que envolvem o mundo da virtualização. Houve a oportunidade de perceber quais as tendências do mercado, relativamente à oferta tecnológica e, na prática, quais as aplicações que efectivamente singraram no mundo empresarial e institucional. Também se pôde desmistificar alguns preconceitos que, a priori, favoreciam uma tecnologia em detrimento de outra, referindo-me claramente, à suposta vantagem do uso de *Unikernels* sobre o uso de *Containers*.

Para concluir, é de referir que a virtualização, é um fenómeno em pleno crescendo, que está a provocar o investimento de cifras avultadas em empresas puramente tecnológicas, como no caso dos cerca de doze mil milhões de dólares investidos na *Docker, inc.*! E, tal fenómeno, está a criar cada vez mais interdependências entre o mundo do *software* e o mundo do *hardware*. É sensato portanto, da parte dos profissionais da área de administração de sistemas, acompanhar estas tecnologias de muito perto, porque o domínio da virtualização, é indubitavelmente, uma competência a destacar.

5.2 Trabalho Futuro

Com o término do prazo de entrega do projecto, fica por encontrar uma solução definitva para a aplicação ***RunQC***! No entanto, depois de uma pequena análise, existem algumas possibilidades de resolução que ganham destaque e que são a seguir enumeradas:

1. A solução mais óbvia e, que não deve ser desconsiderada, será a utilização de um *Container* para a virtualizar.

2. Existe a possibilidade mais custosa, obviamente, com a devida permissão dos autores, de migrar o seu código fonte para Python. Isto porque é um método amplamente abordado e utilizado.

3. A minha preferência, vai no entanto, para a construção de um Wrapper em *Python*, para que seja possível a aplicação da técnica de virtualização considerada inicialmente.

Como sugestão futura, fica também a possibilidade de evoluir o ambiente virtulizado do EPOS, caso haja essa necessidade, para uma solução distribuída em Cluster, com o módulo *Swarm* do *Docker*.

Bibliografia

Michael J. De Lucia. A survey on security isolation of virtualization, containers, and unikernels. Technical report, US Army Research Laboratory, 2017.

Doug Hillen. Top 5 benefits of virtualization for your business, Setembro 2016. URL https://www.centergrid.com/top-5-benefits-of-virtualization/.

Kai Hwang, Geoffrey C. Fox, and Jack J. Dongarra. *Distributed and Cloud Computing: From Parallel Processing to the Internet of Things*. Morgan Kaufmann, 2012.

Docker Inc. Docker documentation, 2018. URL https://docs.docker.com/.

European Research Infrastructure on Solid Earth. What is epos, Julho 2018. URL https://www.epos-ip.org/about/what-epos.

Jordan Shropshire. Analysis of monolithic and microkernel architectures: Towards secure hypervisor design. *IEEE Computer Society*, 2014.

solo.io. A platform for automating unikernel compilation and deployment, Julho 2018. URL https://github.com/solo-io/unik.

Xen.org. Manuals and documentation, Fevereiro 2018. URL https://wiki.xenproject.org/wiki/Category:Manual.