INFRAESTRUTURA PARA DESENVOLVIMENTO WEB

UNIDADE II

CLOUD COMPUTING

Elaboração

Miriã Falcão Freitas

Produção

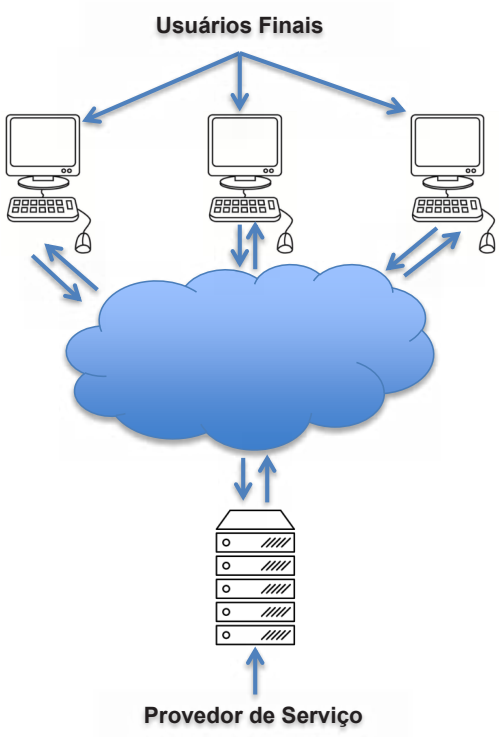
Equipe Técnica de Avaliação, Revisão Linguística e Editoração

SUMÁRIO

Sempre que você viaja de ônibus, pega uma passagem para o seu destino e mantém-se dentro do ônibus até chegar ao seu destino. Da mesma forma, outros passageiros também pegam passagens e viajam no mesmo ônibus que você e dificilmente importa a você aonde eles vão. Quando chega, você desce do ônibus agradecendo ao motorista. A computação em nuvem é exatamente como aquele ônibus, transportando dados e informações para diferentes usuários e permitindo, ainda, usar seu serviço com custo mínimo.

O termo **nuvem** veio de um *design* de rede usado por engenheiros de rede para representar a localização de vários dispositivos de rede e a interconexão. A forma desse *design* de rede era como uma nuvem, ilustrada na Figura 5.

Figura 5. *Design* de rede como nuvem



Fonte: Elaborada pela autora.

Com o aumento do uso do computador e de dispositivos móveis, o armazenamento de dados tornou-se uma prioridade em todos os campos. Hoje, empresas de grande e pequeno porte geram muitos dados e acabam gastando uma quantia de dinheiro significativa para armazenar e proteger esses dados. No entanto, nem todas as empresas podem arcar com

os altos custos da infraestrutura de TI interna e com os serviços de suporte de *backup*. Para eles, a computação em nuvem é uma solução mais barata. Talvez sua eficiência no armazenamento de dados, computação e menor custo de manutenção tenha conseguido atrair empresas ainda maiores.

A computação em nuvem diminui a demanda de *hardware* e *software* do lado do usuário. A única coisa que o usuário deve executar é o *software* de interface dos sistemas de computação em nuvem, que pode ser tão simples quanto um navegador da *web*, e a rede em nuvem cuida do resto. Todos nós experimentamos a computação em nuvem em algum momento, alguns dos populares serviços em nuvem que usamos ou ainda estamos usando são serviços de correio como Gmail, Hotmail, Yahoo, etc.

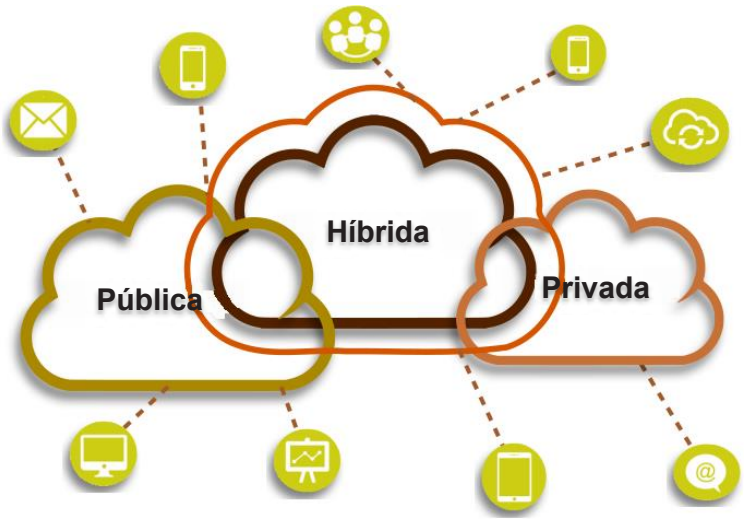
Ao acessar o serviço de *e-mail*, nossos dados são armazenados no servidor em nuvem e não em nosso computador. A tecnologia e a infraestrutura por trás da nuvem são invisíveis. Para o usuário final não importa se os serviços em nuvem são baseados em HTTP, XML, *Ruby*, PHP ou outras tecnologias específicas, desde que sejam amigáveis e funcionais. Um usuário individual pode se conectar ao sistema em nuvem a partir de seus próprios dispositivos, como *desktop*, *laptop* ou celular.

A economia é o principal motivo da adoção de serviços em nuvem por muitas organizações. A computação em nuvem oferece a liberdade de usar serviços de acordo com os requisitos e pagar apenas pelo que você usa. A seguir, estão os benefícios da computação em nuvem:

1. Infraestrutura de TI e custos com computadores mais baixos para os usuários.
2. Performance melhorada.
3. Menos problemas de manutenção.
4. Atualizações instantâneas de *software*.
5. Melhor compatibilidade entre sistemas operacionais.
6. Restauração e recuperação de dados.
7. Desempenho e escalabilidade.
8. Maior capacidade de armazenamento.
9. Aumento da segurança dos dados.

A Figura 6 ilustra os três principais tipos de nuvem que você pode utilizar de acordo com suas necessidades.

Figura 6. Tipos de nuvem



Fonte: Iperius Backup Brasil (2019).

» **Nuvem pública**: uma nuvem pública é provavelmente a opção de computação em nuvem mais comumente entendida. É aqui que todos os serviços e infraestrutura de suporte são gerenciados externamente pela internet e compartilhados por vários usuários (ou inquilinos).

Um bom exemplo de nuvem pública no nível do consumidor individual é um serviço de *streaming* como Netflix, por exemplo. Os usuários assinam o serviço através de uma conta individual, mas acessam os mesmos serviços através da plataforma pela internet.

A vantagem de usar uma nuvem pública é o aumento da eficiência e a subsequente relação custo-benefício dos recursos compartilhados. As nuvens públicas são geralmente mais baratas que as soluções de nuvem privada e híbrida (bem como a computação tradicional no local) porque dependem de economias de escala. Os usuários não precisam pagar pelos serviços que não estão usando e não precisam se preocupar em gerenciar e manter a infraestrutura física.

» **Nuvem privada**: uma nuvem privada fornece serviços de TI através da internet ou de uma rede privada para selecionar usuários, e não para o público em geral. Em vez de ter vários inquilinos, como uma nuvem pública, normalmente uma nuvem privada possui apenas um inquilino. Todos os dados são protegidos por um *firewall*. Essa é uma escolha popular para muitas empresas que desejam a agilidade da nuvem com maior personalização e segurança.

Nuvens privadas podem residir no local ou fora dele. O recurso distintivo é o inquilino único e privado que mantém maior controle sobre os serviços de TI. As

nuvens privadas são escolhas populares para organizações que têm altas prioridades em segurança e conformidade.

» **Nuvem híbrida**: os ambientes de nuvem híbrida combinam elementos de nuvem pública e privada em vários graus. Apesar de operar de forma independente, as nuvens em ambiente híbrido se comunicam-se por uma conexão criptografada e permitem a portabilidade de dados e aplicativos.

  Essa é uma solução em nuvem cada vez mais popular, pois permite às organizações maior flexibilidade para atender às suas necessidades de TI.

Nos Capítulos 1, 2 e 3 vamos conhecer em detalhes a infraestrutura de uma nuvem, abordando a arquitetura em nuvem, *containers* e virtualização.

CAPÍTULO 1

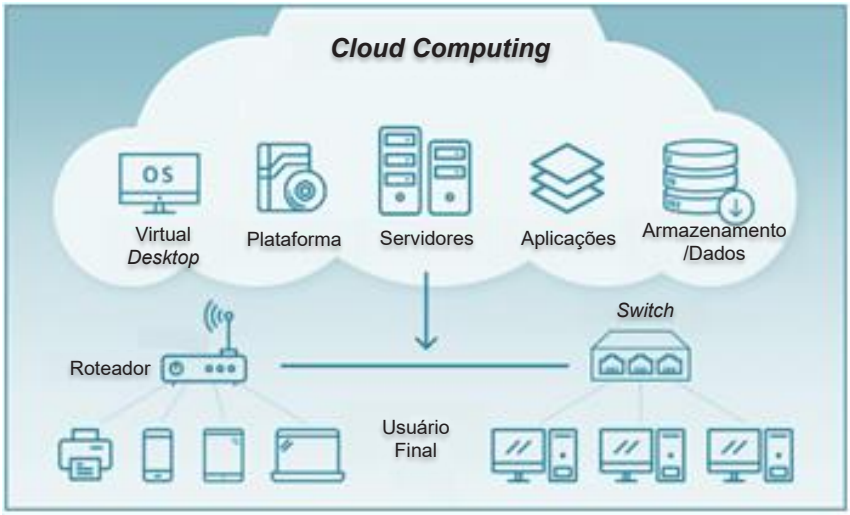
ARQUITETURA EM NUVEM

As nuvens estão por toda parte e influenciam a vida cotidiana. Desde seus e-mails, mídias sociais, aplicativos de compras até transações bancárias, todos usam a nuvem de uma maneira ou de outra. Aproximadamente metade das agências do governo dos EUA está usando a nuvem e cerca de 60% dos tomadores de decisão americanos de TI afirmam que a nuvem é segura (ALLIED, 2015).

É de seu conhecimento que a computação em nuvem refere-se à ideia de usar uma rede de servidores remotos na internet para acumular, gerenciar e rotear os dados. Fornecer qualquer serviço de computação na internet é computação em nuvem. Esses serviços de computação em nuvem podem ser armazenamento, servidor, banco de dados, *software*, rede, inteligência e análise.

As empresas estão adotando a computação em nuvem para múltiplos benefícios, como minimizar despesas de capital, autosserviços sob demanda, escalabilidade global, desempenho ideal, segurança, alta produtividade e confiabilidade. Vamos entender as principais áreas da computação em nuvem na a Figura 7.

Figura 7. Computação em nuvem



Fonte: Pinterest (2020).

A computação em nuvem apresenta três níveis de conectividade, incluindo a nuvem, dispositivos de rede como roteador e *switches* e usuário final.

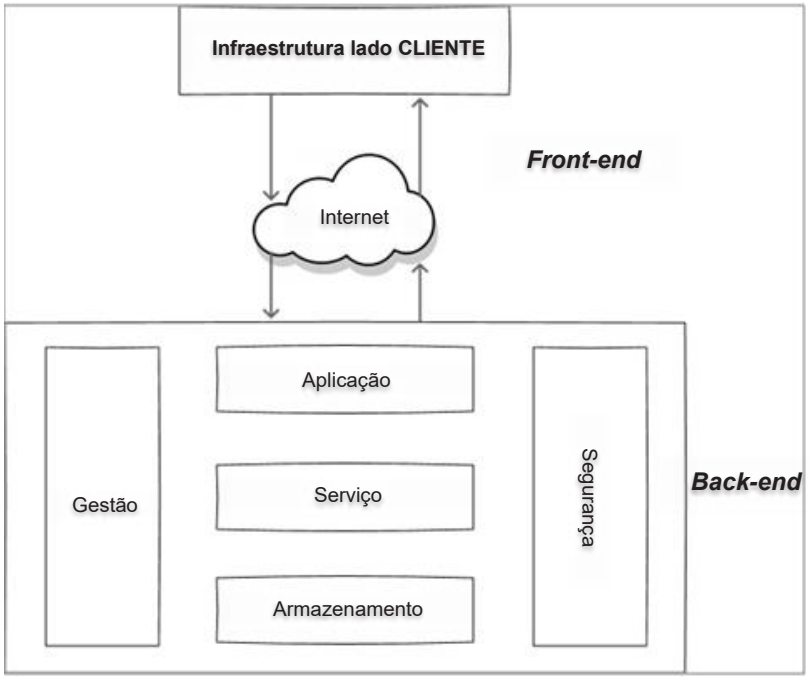
A nuvem compreende recursos como área de trabalho virtual, plataforma de *software*, servidores, aplicativos e armazenamento de dados. Eles processam dados através de roteadores e *switches*.

E o usuário final pode acessar as informações de qualquer dispositivo.

A arquitetura de computação em nuvem compreende dois componentes fundamentais, o *front-end* e *back-end*. O *front-end* trabalha como cliente nessa arquitetura e se comunica com o *back-end* por meio de uma rede ou internet. Na arquitetura de computação em nuvem, o lado do cliente ou o *front-end* é visível para o usuário final. O *front-end* envia consultas ao *back-end* por meio do *middleware*.

O *back-end* protege os dados e responde às consultas feitas pelo *front-end*. O *back-end* é uma parte maior de toda a arquitetura de computação em nuvem, como mostrado na Figura 8.

Figura 8. Arquitetura de computação em nuvem



Fonte: Adaptado de W3 Schools (2020).

Todo esse modelo de serviço em nuvem é chamado de *back-end* como serviço ou *BaaS* (*Back-end as a Service*).

Em um ambiente de negócios, é importante descobrir os componentes de *software* e *hardware* adequados que criam todo o ambiente de nuvem. Embora você possa escolher o *hardware* como peças prontas para uso e escolher o *software* de acordo com os requisitos e o orçamento da empresa. Os principais provedores de serviços em nuvem oferecem todo o pacote de *hardware* e *software* emparelhados.

Se você planeja migrar para a nuvem, a seleção da arquitetura de *software* em nuvem apropriada para os seus negócios é uma das decisões de negócios mais importantes.

O planejamento ineficaz da arquitetura da computação em nuvem pode levar a uma escalabilidade zero com baixo custo-benefício. A arquitetura de computação em nuvem adequada permite que você cuide de todos os componentes de *software* e *hardware*.

Os componentes fundamentais da arquitetura de computação em nuvem são:

» Plataforma *front-end*

» Plataforma de *back-end*.

» Entrega baseada em nuvem.

Além das plataformas *front-end* e *back-end*, a entrega baseada em nuvem permite transmitir informações através de várias infraestruturas de nuvem, como Infraestrutura como Serviço (IaaS), Plataforma como Serviço (PaaS) e *Software* como Serviço (SaaS).

1.1. Arquitetura de nuvem *front-end*

A infraestrutura de *front-end* inclui tudo aquilo que o usuário final interage. É a assimilação mais ampla de vários subcomponentes que, juntos, oferecem a interface do usuário. E constitui uma parte essencial de como o usuário final se conecta à infraestrutura de computação em nuvem. A infraestrutura de nuvem *front-end* inclui componentes como redes locais, navegadores e aplicativos da *web*.

Os principais componentes da nuvem *front-end* estão descritos abaixo:

» **Interface do usuário**: a interface do usuário refere-se a todas as coisas que o usuário final acessa para enviar solicitações ou executar qualquer tarefa na nuvem. Algumas das interfaces de usuário populares baseadas na nuvem são Google Docs, Gmail, etc.

» ***Software***: a arquitetura do *software* no *front-end* é o *software* executado no lado do usuário. A arquitetura do *software front-end* compreende principalmente aplicativos ou navegadores do lado do cliente.

» **Dispositivo ou rede do cliente**: sendo uma parte crucial da arquitetura de *front-end*, o dispositivo ou a rede do cliente refere-se ao *hardware* no lado do usuário final. Pode ser qualquer dispositivo de entrada ou PC. Na computação em nuvem, o dispositivo do lado do cliente não requer capacidade extraordinária para processar a carga pesada. A nuvem pode suportar toda a carga pesada e processar o mesmo.

1.2. Arquitetura de nuvem *back-end*

A arquitetura de *back-end* na nuvem capacita a arquitetura de *front-end*. Ele inclui *hardware* e armazenamento e eles estão localizados em um servidor remoto. O provedor de serviços em nuvem controla e lida com essa arquitetura de nuvem *back-end*.

A arquitetura ideal da nuvem de *back-end* sempre deve ser robusta, pois mantém toda a infraestrutura na nuvem. Os principais componentes da arquitetura de nuvem de *back-end* são:

- **Aplicativo**: o Aplicativo é uma parte substancial da arquitetura de *back-end*. Refere-se à interface que o *back-end* oferece ao usuário final para enviar consultas. Essa camada do *back-end* cuida das solicitações e requisitos do cliente.
- **Serviço**: essa é uma área mágica da arquitetura da nuvem de *back-end*. Ele adiciona utilidade a toda a arquitetura de *back-end*. O serviço lida com todas as tarefas executadas no sistema de computação em nuvem. Alguns dos serviços em nuvem são ambientes de desenvolvimento de aplicativos, armazenamento e serviços da *web*. Além disso, o serviço pode executar uma ampla variedade de tarefas no tempo de execução na nuvem.
- ***Cloud Runtime***: o termo *Cloud Runtime* é o conceito em que os serviços são executados. É como um sistema operacional de nuvem em que tecnologia como virtualização é usada. A virtualização uma tecnologia essencial na nuvem, que permite vários tempos de execução no mesmo servidor. Por exemplo, a virtualização é uma maneira pela qual podemos criar uma base de *software*. Em palavras simples, é a representação virtual de aplicativos, servidores, armazenamento e redes. Quando criamos tempos de execução com o suporte de *software* de virtualização, eles são chamados de Hypervisores. Alguns dos principais hypervisores são o *Oracle Virtual Box*, o *Oracle* VM para x86, o *VMWare Fusion,* etc.
- **Armazenamento**: o armazenamento na nuvem é onde os dados residem em um aplicativo em nuvem. O armazenamento de dados varia de acordo com o serviço de nuvem diferente fornecido. No entanto, todos eles têm um segmento dedicado comum para armazenamento em nuvem. Alguns dos exemplos de armazenamento são unidades de estado sólido, discos rígidos, armazenamento persistente *Intel Optane DC*, etc. Os discos rígidos nos compartimentos do servidor formam armazenamento na arquitetura de *back-end* na nuvem. E, especialmente em um sistema de computação em nuvem, o *software* particiona as unidades conforme as necessidades do sistema operacional na nuvem para executar inúmeros serviços.

» **Infraestrutura**: o mecanismo que administra todos os serviços de *software* em nuvem é chamado de infraestrutura. Inclui CPU, placa-mãe, unidade de processamento gráfico (GPU), placas de rede, placas aceleradoras, etc. Os modelos de infraestrutura sempre dependem das cargas de trabalho dos clientes.

» **Gerenciamento**: o *software* de gerenciamento aloca recursos específicos para tarefas específicas e responsáveis pelo funcionamento impecável de qualquer ambiente em nuvem. Em termos técnicos, o gerenciamento é o *middleware* e coordena a arquitetura de *front-end* e *back-end* em um sistema de computação em nuvem.

» **Segurança**: a segurança é parte integrante e crítica de qualquer infraestrutura de computação em nuvem. Criamos infraestrutura de segurança, mantendo o processo de depuração em mente. Em caso de qualquer problema, a depuração deve ser fácil. O *backup* regular de armazenamento é o primeiro passo para garantir a segurança em um sistema de computação em nuvem. E os *firewalls* virtuais são outros elementos cruciais da infraestrutura de segurança na nuvem.

1.3. Entrega baseada em nuvem

A entrega baseada em nuvem é tudo o que estamos oferecendo aos usuários finais da nuvem por meio de algum software, infraestrutura e plataformas. Podemos fornecer serviços de computação em nuvem através dos modelos abaixo mencionados:

» ***Software* como serviço (SaaS)**: oferecendo serviços de computação em nuvem por meio de software ou assinatura licenciada. Nesse modelo de entrega, os usuários finais não precisam comprar ou instalar nenhum *hardware* em seus locais.

» **Plataforma como serviço (PaaS)**: este modelo oferece uma plataforma que permite aos usuários finais desenvolver, executar e gerenciar aplicativos na nuvem. No PaaS, um provedor de serviços terceirizado facilita as ferramentas de *hardware* e *software*.

» **Infraestrutura como serviço (IaaS)**: esse modelo facilita o *hardware* do computador, como tecnologia de rede, servidores, armazenamento e espaço do *data center* como serviço. Também inclui o fornecimento de tecnologia de virtualização e sistema operacional.

CAPÍTULO 2

CONTAINERS

As mudanças na infraestrutura de TI existentes em todo o mundo são muito proeminentes agora. A transformação digital é o principal tema de foco de quase todos os negócios atualmente. Toda empresa entende que as infraestruturas herdadas não conseguem lidar com as demandas emergentes da transformação digital. Os *containers* na computação em nuvem são um dos recursos promissores para garantir a transformação digital.

Os *containers* na computação em nuvem são a unidade padrão de *software* para o código de empacotamento e todas as dependências relacionadas. Portanto, um aplicativo pode ser executado de maneira rápida e eficaz em diferentes ambientes de computação. Os *containers* fornecem um mecanismo de empacotamento lógico que envolve abstrações de aplicativos dos ambientes reais em que os aplicativos são executados.

Para fazer isso, os *containers* tiram vantagem de uma forma de virtualização do sistema operacional (SO) na qual os recursos do SO (no caso do kernel Linux, nomeadamente os *namespaces* e primitivos do *cgroups*) são aproveitados para isolar processos e controlar a quantidade de CPU, memória e disco aos quais esses processos têm acesso.

Os *containers* são pequenos, rápidos e portáteis porque, diferentemente de uma máquina virtual, eles não precisam incluir um sistema operacional convidado em todas as instâncias e podem, em vez disso, simplesmente aproveitar os recursos e recursos do sistema operacional *host*.

Os *containers* ajudam a diferenciar as preocupações essenciais envolvidas no desenvolvimento de aplicativos. Por exemplo, os desenvolvedores podem se concentrar claramente nas dependências e na lógica do aplicativo. Por outro lado, a equipe de operações pode enfatizar a implantação e o gerenciamento. O ponto positivo aqui é que cada membro da equipe de desenvolvimento de aplicativos pode se concentrar nas principais funcionalidades.

A maneira mais fácil de entender um *container* é entender como ele difere de uma máquina virtual tradicional (VM). Na virtualização tradicional, seja no local ou na nuvem, um hypervisor é aproveitado para virtualizar o *hardware* físico. Cada VM contém um sistema operacional convidado, uma cópia virtual do *hardware* que o sistema operacional precisa para executar, juntamente com um aplicativo e suas bibliotecas e dependências associadas.

Em vez de virtualizar o *hardware* subjacente, os *containers* virtualizam o sistema operacional (normalmente Linux), de modo que cada *container* individual contenha

apenas o aplicativo, suas bibliotecas e dependências. A ausência do sistema operacional convidado é o motivo pelo qual os *containers* são tão leves e, portanto, rápidos e portáteis.

A principal vantagem dos *containers*, especialmente em comparação com uma VM, é fornecer um nível de abstração que os torna leves e portáteis.

- **Leve**: os *containers* compartilham o kernel do sistema operacional da máquina, eliminando a necessidade de uma instância completa do sistema operacional por aplicativo e tornando os arquivos de *container* pequenos e fáceis de usar. Seu tamanho menor, especialmente em comparação com as máquinas virtuais, significa que eles podem acelerar rapidamente e oferecer melhor suporte a aplicativos nativos da nuvem que escalam horizontalmente.
- **Portátil e independente de plataforma**: os *containers* carregam todas as suas dependências, o que significa que o *software* pode ser gravado uma vez e executado sem precisar ser reconfigurado nos *laptops*, na nuvem e nos ambientes de computação local.
- **Oferece suporte ao desenvolvimento e arquitetura modernos**: devido a uma combinação de portabilidade/consistência de implantação entre plataformas e seu tamanho pequeno, os *containers* são ideais para padrões modernos de desenvolvimento e aplicativos – como DevOps, sem servidor e microsserviços.
- **Melhora a utilização**: como as VMs anteriores, os *containers* permitem que desenvolvedores e operadores aprimorem a utilização de CPU e memória de máquinas físicas. Onde os *containers* vão ainda mais longe é que, porque eles também habilitam arquiteturas de microsserviço, os componentes do aplicativo podem ser implantados e dimensionados de forma mais granular, uma alternativa atraente para ter que escalar um aplicativo monolítico inteiro porque um único componente está sofrendo com a carga.
- **Isolamento de falhas**: o isolamento de falhas é uma característica inerente aos *containers*, pois cada aplicativo em *container* opera independentemente. Portanto, a falha em um *container* não influenciaria a atividade dos *containers*, sendo, assim, mais fácil para as equipes de desenvolvimento identificar e resolver problemas técnicos em um *container*. Como resultado, você não precisa colocar outros *containers* no tempo de inatividade. Mais importante ainda, o mecanismo de *container* pode usar técnicas de isolamento de segurança do SO para isolar falhas nos *containers*.

Os *containers* estão se tornando cada vez mais importantes, especialmente em ambientes em nuvem. Muitas organizações estão considerando os *containers* como uma substituição

das VMs como a plataforma de computação de uso geral para seus aplicativos e cargas de trabalho. Mas dentro desse escopo muito amplo, há casos de uso importantes em que os *containers* são especialmente relevantes.

- **Microsserviços**: os *containers* são pequenos e leves, o que os torna uma boa combinação para arquiteturas de microsserviços, onde os aplicativos são construídos com muitos serviços menores, de acoplamento fraco e implementáveis independentemente.
- **DevOps**: a combinação de microsserviços como arquitetura e *containers* como plataforma é uma base comum para muitas equipes que adotam o DevOps como a maneira como constroem, enviam e executam *softwares*.
- **Híbrido**: como os *containers* podem ser executados de maneira consistente em qualquer lugar, em ambientes de *laptop*, local e em nuvem, eles são uma arquitetura subjacente ideal para cenários de nuvem híbrida que as organizações se encontram operando em uma mistura de várias nuvens públicas em combinação com seu próprio *datacenter*.
- **Modernização e migração de aplicativos**: uma das abordagens mais comuns para a modernização de aplicativos começa por contê-los para que eles possam ser migrados para a nuvem.

O *software* precisa ser projetado e empacotado de maneira diferente para tirar proveito dos *containers* – um processo geralmente chamado de conteinerização. Vamos ver esse processo em detalhes na seção a seguir.

"Os *containers* na computação em nuvem evoluíram de um chavão de segurança. A implantação da tecnologia *é* um elemento essencial da proteção da infraestrutura de TI.". Disponível em: https://searchcloudsecurity.techtarget.com/feature/Cloud-containers-what-they-are-and-how-they-work. (SHAPLAND; COLE, 2019).

"Os *containers* estão explodindo no cenário de desenvolvimento de aplicativos, especialmente quando se trata de computação em nuvem". Disponível em: https://techbeacon.com/enterprise-it/essential-guide-software-containers-application-development. (LINTHICUM, 2020)

2.1. Conteinerização

Veja como funciona o processo de Conteinerização, assista ao vídeo ***Containerization Explained***. Disponível em: https://www.youtube.com/watch?time_continue=6&v=0qotVMX-J5s&feature=emb_logo. (IBM CLOUD, 2019).

A conteinerização tornou-se uma tendência importante no desenvolvimento de *software* como alternativa ou companheira da virtualização. Envolve encapsular ou empacotar o código do *software* e todas as suas dependências para que ele possa executar de maneira uniforme e consistente em qualquer infraestrutura. A tecnologia está amadurecendo rapidamente, resultando em benefícios mensuráveis para desenvolvedores e equipes de operações, bem como na infraestrutura geral de *software*.

A conteinerização permite que os desenvolvedores criem e implantem aplicativos de maneira mais rápida e segura. Com os métodos tradicionais, o código é desenvolvido em um ambiente de computação específico que, quando transferido para um novo local, geralmente resulta em *bugs* e erros. Por exemplo, quando um desenvolvedor transfere código de um computador *desktop* para uma máquina virtual (VM) ou de um Linux para um sistema operacional Windows. A conteinerização elimina esse problema, agrupando o código do aplicativo junto com os arquivos de configuração, bibliotecas e dependências relacionadas necessários para sua execução. Esse único pacote de *software* ou *container* é retirado do sistema operacional *host* e, portanto, fica sozinho e se torna portátil – capaz de rodar em qualquer plataforma ou nuvem, sem problemas.

O conceito de conteinerização e isolamento de processos tem décadas, mas o surgimento do *Docker Engine* de código aberto em 2013, um padrão do setor para *containers* com ferramentas simples de desenvolvedor e uma abordagem de empacotamento universal, acelerou a adoção dessa tecnologia. A empresa de pesquisa Gartner projeta que mais de 50% das empresas usarão a tecnologia de *containers* até 2020. E os resultados de uma pesquisa realizada pela IBM no final de 2017 realizada sugerem que a adoção está acontecendo ainda mais rapidamente, revelando que 59% dos adotantes melhoraram a qualidade dos aplicativos e reduziram os defeitos.

Os *containers* são geralmente chamados de "leves", o que significa que eles compartilham o kernel do sistema operacional da máquina e não exigem a sobrecarga de associar um sistema operacional a cada aplicativo. Os *containers* são inerentemente menores em capacidade do que uma VM e requerem menos tempo de inicialização, permitindo que muito mais *containers* sejam executados na mesma capacidade de computação que uma única VM. Isso aumenta a eficiência do servidor e, por sua vez, reduz os custos de servidor e licenciamento.

Simplificando, a conteinerização permite que os aplicativos sejam "gravados uma vez e executados em qualquer lugar". Essa portabilidade é importante em termos de processo de desenvolvimento e compatibilidade de fornecedores. Também oferece outros benefícios notáveis, como isolamento de falhas, facilidade de gerenciamento e segurança, para citar alguns.

2.1.1. Container de aplicativos

Os *containers* encapsulam um aplicativo como um único pacote executável de *software* que agrupa o código do aplicativo junto com todos os arquivos de configuração, bibliotecas e dependências relacionadas necessários para sua execução. Os aplicativos em *containers* são "isolados", pois não são agrupados em uma cópia do sistema operacional. Em vez disso, um mecanismo de tempo de execução de código aberto (como o mecanismo de tempo de execução do Docker) é instalado no sistema operacional do *host* e se torna o canal para os *containers* compartilharem um sistema operacional com outros *containers* no mesmo sistema de computação.

Outras camadas de *container*, como compartimentos e bibliotecas comuns, também podem ser compartilhadas entre vários *containers*. Isso elimina a sobrecarga de executar um sistema operacional em cada aplicativo e reduz a capacidade dos *containers* e agiliza a inicialização, aumentando a eficiência do servidor. O isolamento de aplicativos como *containers* também reduz a chance de o código malicioso presente em um *container* impactar outros *containers* ou invadir o sistema *host*.

A abstração do sistema operacional *host* torna os aplicativos em *container* portáteis e capazes de executar de maneira uniforme e consistente em qualquer plataforma ou nuvem. Os *containers* podem ser facilmente transportados de um computador *desktop* para uma máquina virtual (VM) ou de um sistema operacional Linux para Windows e serão executados de forma consistente em infraestruturas virtualizadas ou em servidores tradicionais *bare metal* ou na nuvem local. Isso garante que os desenvolvedores de *software* possam continuar usando as ferramentas e processos com os quais estão mais confortáveis.

Podemos ver por que as empresas estão adotando rapidamente a conteinerização como uma abordagem superior ao desenvolvimento e gerenciamento de aplicativos. A conteinerização permite que os desenvolvedores criem e implantem aplicativos com mais rapidez e segurança, seja um monólito tradicional (um aplicativo de *software* de camada única) ou um microsserviço modular (uma coleção de serviços de baixo acoplamento). Novos aplicativos baseados em nuvem podem ser criados desde o início como microsserviços em *container*, dividindo um aplicativo complexo em uma série de serviços especializados e gerenciáveis menores. Os aplicativos existentes podem ser reembalados em *containers* (ou microsserviços em *container)* que usam recursos de computação com mais eficiência.

2.1.2. Tipos de conteinerização

O rápido crescimento do interesse e do uso de soluções baseadas em *containers* levou à necessidade de padrões em torno da tecnologia de *containers* e à abordagem do código de *software* de embalagem. A *Open Container Initiative* (OCI), criada em junho de 2015 pela Docker e outros líderes do setor, promove padrões e especificações comuns, mínimos e abertos em torno da tecnologia de *containers*. Por esse motivo, a OCI está ajudando a ampliar as opções de mecanismos de código aberto. Os usuários não ficarão presos à tecnologia de um fornecedor específico, mas poderão aproveitar as tecnologias certificadas pela OCI que lhes permitem criar aplicativos em *containers* usando um conjunto diversificado de ferramentas de DevOps e executá-las de forma consistente na(s) infraestrutura(s) a sua escolha.

Hoje, o Docker é uma das tecnologias de mecanismo de *container* mais conhecidas e altamente usadas (veja o Docker em mais detalhes na seção 2.1), mas não é a única opção disponível. O ecossistema está padronizando o container e alternativas como *CoreOS RTK*, *Mesos Containerizer*, *LXC Linux Containers*, *OpenVZ* e *Crio-d*. Os recursos e os padrões podem ser diferentes, mas a adoção e o aproveitamento das especificações OCI à medida que elas evoluem garantirão que as soluções sejam neutras em relação aos fornecedores, certificadas para serem executadas em vários sistemas operacionais e utilizáveis em vários ambientes.

2.1.3. Microsserviços e conteinerização

Grandes e pequenas empresas de *software* estão adotando os microsserviços como uma abordagem superior ao desenvolvimento e gerenciamento de aplicativos, em comparação com o modelo monolítico anterior que combina um aplicativo de *software* com a interface do usuário associada e o banco de dados subjacente em uma única unidade, em uma única plataforma de servidor. Com os microsserviços, um aplicativo complexo é dividido em uma série de serviços menores e mais especializados, cada um com seu próprio banco de dados e sua própria lógica de negócios. Os microsserviços se comunicam entre si através de interfaces comuns (como APIs) e interfaces REST (como HTTP). Usando microsserviços, as equipes de desenvolvimento podem se concentrar em atualizar áreas específicas de um aplicativo sem impactá-lo como um todo, resultando em desenvolvimento, teste e implantação mais rápidos.

Os conceitos por trás dos microsserviços e da conteinerização são semelhantes, pois são práticas de desenvolvimento de *software* que transformam aplicativos em coleções de serviços ou componentes menores, portáteis, escaláveis, eficientes e fáceis de gerenciar.

Além disso, os microsserviços e a conteinerização funcionam bem quando usados juntos. Os *containers* fornecem um encapsulamento leve de qualquer aplicativo, seja um monólito tradicional ou um microsserviço modular. Um microsserviço, desenvolvido dentro de um *container*, obtém todos os benefícios inerentes à conteinerização – portabilidade em termos do processo de desenvolvimento e compatibilidade do fornecedor (sem bloqueio do fornecedor), bem como agilidade do desenvolvedor, isolamento de falhas, eficiência do servidor, automação de instalação, dimensionamento e gerenciamento e camadas de segurança, entre outros.

As comunicações de hoje estão se movendo rapidamente para a nuvem, onde os usuários podem desenvolver aplicativos com rapidez e eficiência. Aplicativos e dados baseados na nuvem são acessíveis a partir de qualquer dispositivo conectado à internet, permitindo que os membros da equipe trabalhem remotamente e em movimento. Os Provedores de Serviços em Nuvem (CSPs) gerenciam a infraestrutura subjacente, o que economiza para as organizações o custo de servidores e outros equipamentos e também fornece backups de rede automatizados para confiabilidade adicional. As infraestruturas de nuvem são dimensionadas sob demanda e podem ajustar dinamicamente recursos, capacidade e infraestrutura de computação à medida que os requisitos de carga mudam. Além disso, os CSPs atualizam regularmente as ofertas, oferecendo aos usuários acesso contínuo à mais recente tecnologia inovadora.

Containers, microsserviços e computação em nuvem estão trabalhando juntos para levar o desenvolvimento e a entrega de aplicativos a novos níveis não possíveis com as metodologias e ambientes tradicionais. Essas abordagens de próxima geração adicionam agilidade, eficiência, confiabilidade e segurança ao ciclo de vida de desenvolvimento de *software* – o que leva a uma entrega mais rápida de aplicativos e aprimoramentos para os usuários finais e o mercado.

Aprofunde mais seu conhecimento em Microsserviços, consulta um guia completo. Disponível em: https://www.ibm.com/cloud/learn/microservices. (IBM, 2019).

2.1.4. Segurança

Os aplicativos em *containers* possuem inerentemente um nível de segurança, pois podem ser executados como processos isolados e operar independentemente de outros *containers*. Verdadeiramente isolados, isso pode impedir que qualquer código malicioso afete outros *containers* ou invada o sistema *host*. No entanto, as camadas de aplicativos em um *container* geralmente são compartilhadas entre *containers*. Em termos de eficiência de recursos, isso é uma vantagem, mas, também, abre as portas para violações de interferência e segurança em *containers*. O mesmo poderia ser dito do sistema

operacional compartilhado, pois vários *containers* podem ser associados ao mesmo sistema operacional *host*. As ameaças à segurança do sistema operacional comum podem afetar todos os *containers* associados e, inversamente, uma violação do *container* pode invadir o sistema operacional *host*.

Mas, e a própria imagem do *container*? Como os aplicativos e os componentes de código aberto empacotados em um *container* podem melhorar a segurança? Os fornecedores de tecnologia de *containers*, como o Docker, continuam a enfrentar ativamente os desafios de segurança de *containers*. A conteinerização adotou uma abordagem de "segurança por padrão", acreditando que a segurança deve ser inerente à plataforma e não uma solução implantada e configurada separadamente. Para esse fim, o mecanismo de *container* suporta todas as propriedades de isolamento padrão inerentes ao sistema operacional subjacente. As permissões de segurança podem ser definidas para impedir que componentes indesejados entrem automaticamente em *containers* ou limitar a comunicação com recursos desnecessários.

Por exemplo, os *Namespaces* do Linux ajudam a fornecer uma visão isolada do sistema para cada *container*; isso inclui rede, pontos de montagem, IDs de processo, IDs de usuário, comunicação entre processos e configurações de nome de *host*. Os espaços para nome podem ser usados para limitar o acesso a qualquer um desses recursos por meio de processos em cada *container*. Normalmente, os subsistemas que não possuem suporte ao Namespace não são acessíveis de dentro de um *container*. Os administradores podem criar e gerenciar facilmente essas "restrições de isolamento" em cada aplicativo em *container* por meio de uma interface de usuário simples.

Os pesquisadores estão trabalhando para fortalecer ainda mais a segurança de *containers* Linux, e uma ampla gama de soluções de segurança estão disponíveis para automatizar a detecção e resposta a ameaças em uma empresa, monitorar e reforçar a conformidade para atender aos padrões e políticas de segurança do mercado, para garantir o fluxo seguro de dados através de aplicativos e terminais e muito mais.

Aprenda estratégias para dimensionar a segurança em *containers* em organizações de qualquer tamanho. Disponível em: https://www.ibm.com/cloud/blog/scale-security-innovating-microservices-fast. (PARIS-WHITE, 2018).

2.2. Dockers

Docker é uma plataforma de conteinerização de código aberto. O *Docker* permite que os desenvolvedores empacotem aplicativos em *containers* – componentes executáveis padronizados que combinam o código-fonte do aplicativo com todas as bibliotecas

e dependências do sistema operacional (SO) necessárias para executar o código em qualquer ambiente.

Embora os desenvolvedores possam criar *containers* sem o *Docker*, o *Docker* torna mais fácil, mais simples e seguro criar, implantar e gerenciar *containers*. É essencialmente um *kit* de ferramentas que permite aos desenvolvedores criar, implantar, executar, atualizar e interromper *containers* usando comandos simples e automação para economia de trabalho.

Docker também se refere à Docker Inc., a empresa que vende a versão comercial do *Docker*, e ao projeto de código aberto do *Docker*, para o qual a Docker Inc. e muitas outras organizações e indivíduos contribuem.

O *Docker* é tão popular hoje em dia que *Docker*'e *containers* são usados de forma intercambiável, mas as primeiras tecnologias relacionadas a *containers* estavam disponíveis por anos – até décadas – antes de o *Docker* ser lançado ao público em 2013. Mais notavelmente, em 2008, o LXC (*Linux Containers*) foi implementado no kernel Linux, permitindo totalmente virtualização para uma única instância de Linux.

As primeiras versões do *Docker* utilizaram o LXC exclusivamente, mas o *Docker* logo desenvolveu sua própria tecnologia de *container* personalizada, que permitiu o seguinte:

- **Portabilidade aprimorada e contínua**: enquanto os *containers* LXC geralmente fazem referência a configurações específicas da máquina, os *containers* do Docker são executados sem modificações em qualquer ambiente de desktop, *data center* e nuvem.
- **Peso ainda mais leve e atualizações mais granulares**: com o LXC, vários processos podem ser combinados em um único *container*. Nos *containers* do *Docker*, apenas um processo pode ser executado em cada *container*. Isso possibilita a criação de um aplicativo que pode continuar em execução enquanto uma de suas partes é retirada para atualização ou reparo.
- **Criação automatizada de *container***: o *Docker* pode criar automaticamente um *container* com base no código-fonte do aplicativo.
- **Controle de versão do *container***: o *Docker* pode rastrear versões de uma imagem de *container*, reverter para versões anteriores e rastrear quem criou uma versão e como. Ele pode até carregar apenas os deltas entre uma versão existente e uma nova.
- **Reutilização de *containers***: os *containers* existentes podem ser usados como **imagens de base** – basicamente como modelos para criar novos *containers*.

» **Bibliotecas de *container* compartilhadas**: os desenvolvedores podem acessar um registro de código aberto contendo milhares de *containers* contribuídos pelo usuário.

2.2.1. Ferramentas *Docker*

Algumas das ferramentas e terminologia que você encontrará ao usar o Docker incluem o seguinte:

» ***DockerFile***: todo *container* do *Docker* começa com um arquivo de texto simples, contendo instruções sobre como criar a imagem do *container* do *Docker*. *O DockerFile* automatiza o processo de criação de imagens do *Docker*. É essencialmente uma lista de comandos que o *Docker Engine* executará para montar a imagem.

» **Imagens do *Docker***: as imagens do Docker contêm código-fonte executável do aplicativo, bem como todas as ferramentas, bibliotecas e dependências que o código do aplicativo precisa executar como um *container*. Quando você executa a imagem do *Docker*, ela se torna uma instância (ou várias instâncias) do *container*.

É possível criar uma imagem do *Docker* do zero, mas a maioria dos desenvolvedores as extrai de repositórios comuns. Várias imagens do *Docker* podem ser criadas a partir de uma única imagem base e compartilharão os pontos comuns da sua pilha.

As imagens do *Docker* são compostas de camadas e cada camada corresponde a uma versão da imagem. Sempre que um desenvolvedor faz alterações na imagem, uma nova camada superior é criada e essa camada superior substitui a camada superior anterior como a versão atual da imagem. As camadas anteriores são salvas para reversões ou para serem reutilizadas em outros projetos.

Sempre que um *container* é criado a partir de uma imagem do *Docker*, outra nova camada chamada **camada de *container*** é criada. As alterações feitas no *container* – como a adição ou exclusão de arquivos – são salvas apenas na camada do *container* e existem apenas enquanto o *container* estiver em execução. Esse processo iterativo de criação de imagem permite maior eficiência geral, pois várias instâncias de *container* ativo podem ser executadas a partir de apenas uma imagem-base e, quando o fazem, utilizam uma pilha comum.

» ***Containers Docker***: os *containers* do *Docker* são instâncias ativas e em execução das imagens do *Docker*. Embora as imagens do *Docker* sejam arquivos somente de leitura, os *containers* são conteúdo ativo, efêmero e executável. Os usuários podem interagir com eles e os administradores podem ajustar suas configurações e condições.

» **Docker Hub**: o *Docker Hub* é o repositório público de imagens do *Docker* que se autodenomina a "maior biblioteca e comunidade do mundo para imagens de *containers*". Possui mais de 100.000 imagens de *containers* provenientes de fornecedores de *software* comercial, projetos de código aberto e desenvolvedores individuais. Inclui imagens que foram produzidas pela *Docker* Inc., imagens certificadas pertencentes ao *Docker Trusted Registry* e muitos milhares de outras imagens.

  Todos os usuários do *Docker Hub* podem compartilhar suas imagens à vontade. Eles também podem baixar imagens de base predefinidas para usar como ponto de partida para qualquer projeto de *container*.

» **Implantação e orquestração do *Docker***: se você estiver executando apenas alguns *containers*, é bastante simples gerenciar seu aplicativo no próprio *Docker Engine*. Mas se a sua implantação compreender milhares de *containers* e centenas de serviços, é quase impossível gerenciar sem a ajuda dessas ferramentas criadas especificamente para esse fim.

» ***Docker Compose***: se você estiver criando um aplicativo fora dos processos em vários *containers* que residem no mesmo *host*, poderá usar o *Docker Compose* para gerenciar a arquitetura do aplicativo. O *Docker Compose* cria um arquivo *YAML* que especifica quais serviços estão incluídos no aplicativo e pode implantar e executar *containers* com um único comando. Usando o *Docker Compose*, você também pode definir volumes persistentes para armazenamento, especificar nós de base e documentar e configurar dependências de serviço.

» ***Kubernets:*** veremos em detalhes na seção 2.3.

2.3. Kubernets

Assista ao vídeo ***Kubernetes Explained***. Disponível em: https://www.youtube.com/watch?v=aSrqRSk43lY&feature=emb_logo. (IBM CLOUD, 2019).

O *Kubernetes* – também conhecido como *k8s* ou *kube* – é uma plataforma de orquestração de *containers* para agendar e automatizar a implantação, o gerenciamento e o dimensionamento de aplicativos em *container*.

O *Kubernetes* foi desenvolvido por engenheiros do Google antes de ser de código aberto em 2014. É um descendente do *Borg*, uma plataforma de orquestração de *containers* usada internamente no Google.

Hoje, o *Kubernetes* e o ecossistema mais amplo de *containers* estão amadurecendo em uma plataforma de computação de uso geral e ecossistema que rivaliza – se não ultrapassa – as máquinas virtuais (VMs) como os blocos de construção básicos da infraestrutura e aplicativos modernos em nuvem. Esse ecossistema permite que as organizações forneçam uma Plataforma como Serviço (PaaS) de alta produtividade que lide com várias tarefas e problemas relacionados à infraestrutura e às operações que envolvem o desenvolvimento nativo da nuvem, para que as equipes de desenvolvimento possam se concentrar apenas em codificação e inovação.

À medida que os *containers* proliferavam – hoje, uma organização pode ter centenas ou milhares deles –, as equipes de operações precisam agendar e automatizar a implantação, a rede, a escalabilidade e a disponibilidade dos *containers*. E assim nasceu o mercado de orquestração de *containers*.

Enquanto outras opções de orquestração de *containers* – principalmente *Docker Swarm* e *Apache Mesos* – ganharam força no início, o *Kubernetes* rapidamente se tornou o mais amplamente adotado (de fato, a certa altura, foi o projeto de mais rápido crescimento na história do *software* de código aberto).

Os desenvolvedores escolheram (e continuam a escolher) o *Kubernetes* por sua ampla funcionalidade, seu vasto e crescente ecossistema de ferramentas de suporte de código aberto e seu suporte e portabilidade entre os principais provedores de nuvem (alguns dos quais agora oferecem serviços totalmente gerenciados do *Kubernetes*).

O *Kubernetes* agenda e automatiza estas e outras tarefas relacionadas ao container:

- **Implantação**: implante um número especificado de *containers* em um *host* especificado e mantenha-os em execução no estado desejado.
- **Lançamentos**: um lançamento é uma alteração em uma implantação. O *Kubernetes* permite iniciar, pausar, retomar ou reverter lançamentos.
- **Descoberta de serviço**: o *Kubernetes* pode expor automaticamente um *container* à Internet ou a outros *containers* usando um nome DNS ou endereço IP.
- **Provisionamento de armazenamento**: defina o *Kubernetes* para montar armazenamento local ou na nuvem persistente para seus *containers*, conforme necessário.
- **Balanceamento de carga e dimensionamento**: quando o tráfego para um *container* aumenta, o *Kubernetes* pode empregar balanceamento de carga e dimensionamento para distribuí-lo pela rede para manter a estabilidade.

» **Autocorreção para alta disponibilidade**: quando um *container* falha, o *Kubernetes* pode reiniciar ou substituí-lo automaticamente; também pode retirar recipientes que não atendem aos seus requisitos de verificação de saúde.

2.3.1. Arquitetura *Kubernetes*

Os principais componentes da arquitetura *Kubernetes* incluem o seguinte:

2.3.2. Clusters e nós (computação)

Clusters são os blocos de construção da arquitetura *Kubernetes*. Os *clusters* são compostos de nós, cada um dos quais representa um único *host* de computação (máquina virtual ou física).

Cada *cluster* consiste em vários nós do trabalhador que implantam, executam e gerenciam aplicativos em *container* e um nó principal que controla e monitora os nós do trabalhador.

O nó-mestre executa um serviço do planejador que automatiza quando e onde os *containers* são implantados com base nos requisitos de implantação definidos pelo desenvolvedor e na capacidade de computação disponível. Cada nó do trabalhador inclui a ferramenta que está sendo usada para gerenciar os *containers* – como o *Docker* – e um agente de *software* chamado *Kubelet*, que recebe e executa pedidos do nó principal.

Aprofunde seus conhecimentos sobre os *clusters* do *Kubernetes*, confira o *blog*: ***Clusters* do *Kubernetes*: arquitetura para entrega rápida e controlada de aplicativos na nuvem.** Disponível em: https://www.ibm.com/cloud/blog/new-builders/kubernetes-clusters-architecture-for-rapid-controlled-cloud-app-delivery. (VENNAM, 2019).

2.3.3. Pods e implantações *(software)*

Os *pods* são grupos de *containers* que compartilham os mesmos recursos de computação e a mesma rede. Eles também são a unidade de escalabilidade no *Kubernetes*: se um *container* em um pod estiver obtendo mais tráfego do que ele pode suportar, o *Kubernetes* replicará o *pod* para outros nós no *cluster*. Por esse motivo, é uma boa prática manter os *pods* compactos para que eles contenham apenas *containers* que precisam compartilhar recursos.

A implantação controla a criação e o estado do aplicativo em *container* e o mantém em execução. Ele especifica quantas réplicas de um *pod* devem ser executadas no cluster. Se um *pod* falhar, a implantação criará um novo.

Para saber mais sobre as implantações de *Kubernetes*, assista ao vídeo: ***Kubernetes Deployments***". Disponível em: https://www.youtube.com/watch?v=Sulw5ndbE88&feature=emb_logo. (IBM CLOUD, 2019).

2.3.4. Malha de serviço *Istio*

O *Kubernetes* pode implantar e dimensionar *pods*, mas não pode gerenciar ou automatizar o roteamento entre eles e não fornece nenhuma ferramenta para monitorar, proteger ou depurar essas conexões. À medida que o número de *containers* em um *cluster* aumenta, o número de possíveis caminhos de conexão entre eles aumenta exponencialmente (por exemplo, dois *containers* têm duas conexões em potencial, mas 10 *pods* têm 90), criando um possível pesadelo de configuração e gerenciamento.

O *Istio* possui uma camada de malha de serviço de código aberto para *clusters Kubernetes*. Para cada *cluster* do *Kubernetes*, o *Istio* adiciona um *container* lateral – essencialmente invisível para o programador e o administrador – que configura, monitora e gerencia interações entre os outros *containers*.

Com o *Istio*, você define uma política única que configura as conexões entre os *containers* para que você não precise configurar cada conexão individualmente. Isso facilita a depuração de conexões entre *containers*.

O *Istio* também fornece um painel que as equipes e administradores do *DevOps* podem usar para monitorar a latência, erros de tempo de serviço e outras características das conexões entre *containers*. Além disso, ele cria segurança – especificamente, gerenciamento de identidade que impede a falsificação de uma chamada de serviço entre *containers* – e recursos de autenticação/autorização/auditoria (AAA) que os profissionais de segurança podem usar para monitorar o cluster.

Saiba mais sobre o *Istio*. Disponível em: https://www.ibm.com/cloud/learn/istio. (IBM, 2019).

2.3.5. Computação *Knative* e sem servidor

Knative (pronuncia-se *kay-nativ*') é uma plataforma de código aberto que fica no topo do *Kubernetes* e oferece duas classes importantes de benefícios para o desenvolvimento nativo da nuvem:

2.3.6. Knative *fornece um* onramp *fácil para computação sem servidor*

A computação sem servidor é uma maneira relativamente nova de implantar código que torna os aplicativos nativos da nuvem mais eficientes e econômicos. Em vez de implantar uma instância contínua de código que fica ociosa enquanto aguarda solicitações, a computação sem servidor exibe o código "conforme necessário" – aumentando ou diminuindo conforme a demanda flutua – e reduz o código quando não está em uso. Dessa forma, a computação sem servidor evita o desperdício de capacidade e energia da computação e reduz os custos porque você paga apenas para executar o código quando ele estiver em execução.

O *Knative* permite que os desenvolvedores construam um *container* uma vez e o executem como um serviço de *software* ou como uma função sem servidor. É tudo transparente para o desenvolvedor: o *Knative* lida com os detalhes em segundo plano, e o desenvolvedor pode se concentrar no código.

2.3.7. Knative *simplifica o desenvolvimento e a orquestração de containers*

Para os desenvolvedores, o código de *container* requer muitas etapas repetitivas e a orquestração de *containers* requer muita configuração e *script* – por exemplo, gerar arquivos de configuração, instalar dependências, gerenciar *log* e rastreio e escrever *scripts* de integração/implantação contínua (CI/CD).

O *Knative* facilita essas tarefas automatizando-as através de três componentes:

- ***Build*:** o componente *Build* do *Knative* transforma automaticamente o código-fonte em um *container* ou função nativa da nuvem. Especificamente, ele extrai o código do repositório, instala as dependências necessárias, cria a imagem do *container* e o coloca em um registro de *container* para uso de outros desenvolvedores. Os desenvolvedores precisam especificar a localização desses componentes para que o *Knative* possa encontrá-los, mas, uma vez feito, o *Knative* automatiza a compilação.
- ***Servir*:** o componente *Servir* executa *containers* como serviços escaláveis; ele pode escalar até milhares de instâncias de *container* ou reduzir para nenhum (chamado escalonamento para zero). Além disso, o *Servir* possui dois recursos muito úteis:
  - Configuração: que salva versões de um *container* (chamadas de instantâneos) toda vez que você coloca o *container* em produção e permite executar essas versões simultaneamente.

- Roteamento de serviço: que permite direcionar diferentes quantidades de tráfego para essas versões. Você pode usar esses recursos juntos para, gradualmente, criar um *rollout* de *container* ou para realizar um teste canário de um aplicativo em *container* antes de colocá-lo na produção global.

» **Evento**: permite que eventos especificados acionem serviços ou funções baseadas em *container*. Isso é especialmente essencial para os recursos sem servidor da *Knative*; algo precisa dizer ao sistema para ativar uma função quando necessário. O evento permite que as equipes expressem "interesse" nos tipos de eventos e, em seguida, ele se conecta automaticamente ao produtor do evento e os encaminha para o *container*, eliminando a necessidade de programar essas conexões.

Se você estiver pronto para começar a trabalhar com o *Kubernetes* ou procurar desenvolver suas habilidades com as ferramentas do ecossistema do *Kubernetes*, tente um destes tutoriais:

Rede *Kubernetes*: um laboratório sobre conceitos básicos de rede. Disponível em: https://developer.ibm.com/tutorials/kubernetes-networking-101-lab/. (KIM, 2019).

Depure e registre seus aplicativos *Kubernetes*. Disponível em: https://developer.ibm.com/tutorials/debug-and-log-your-kubernetes-app/. (NEEMAN, 2019).

Knative 101: exercícios projetados para ajuda-lo a entender o Knative. Disponível em: https://developer.ibm.com/tutorials/knative-101-labs/. (DAVIS; VENNAM, 2019).

CAPÍTULO 3
VIRTUALIZAÇÃO

Você sabe o que é uma Máquina Virtual (VM)?

Máquinas virtuais (VMs) são ambientes virtuais que simulam uma computação física em forma de *software*. Eles normalmente incluem vários arquivos que contêm a configuração da VM, o armazenamento do disco rígido virtual e alguns instantâneos da VM que preservam seu estado em determinado momento.

Para uma visão geral completa sobre VMs, consulte o texto: **Máquinas Virtuais: um guia completo**. Disponível em: https://www.ibm.com/cloud/learn/virtual-machines. (IBM, 2019).

Você sabe o que são *Hypervisores?*

Um *hypervisor* é a camada de *software* que coordena as VMs. Ele serve como uma interface entre a VM e o *hardware* físico subjacente, garantindo que cada um tenha acesso aos recursos físicos que precisa executar. Ele também garante que as VMs não interfiram entre si, afetando o espaço de memória ou os ciclos de computação.

Existem dois tipos de hipervisores:

- **Os hypervisores tipo 1 ou *bare-metal***: interagem com os recursos físicos subjacentes, substituindo por completo o sistema operacional tradicional. Eles costumam aparecer em cenários de servidor virtual.
- **Os hypervisores do tipo 2**: são executados como um aplicativo em um sistema operacional existente. Mais comumente usados em dispositivos de terminal para executar sistemas operacionais alternativos, eles carregam uma sobrecarga de desempenho porque precisam usar o sistema operacional *host* para acessar e coordenar os recursos de *hardware* subjacentes.

O texto: ***Hypervisors:* um guia completo** fornece uma visão abrangente sobre hipervisores. Disponível em: https://www.ibm.com/cloud/learn/hypervisors. (IBM, 2019).

A virtualização usa o *software* para criar uma camada de abstração sobre o *hardware* do computador que permite que os elementos de *hardware* de um único computador – processadores, memória, armazenamento e mais – sejam divididos em vários computadores virtuais, comumente chamados de **máquinas virtuais (VMs)**. Cada VM executa seu próprio sistema operacional (SO) e se comporta como um computador independente, mesmo executando em apenas uma parte do *hardware* subjacente do computador.

Daqui resulta que a virtualização permite uma utilização mais eficiente do *hardware* físico do computador e permite maior retorno do investimento em *hardware* da organização.

Hoje, a virtualização é uma prática-padrão na arquitetura corporativa de TI. É também a tecnologia que impulsiona a economia da computação em nuvem. A virtualização permite que os provedores de nuvem atendam aos usuários com seu *hardware* físico existente; ele permite que os usuários da nuvem comprem apenas os recursos de computação de que precisam quando precisam e escalem esses recursos de maneira econômica com o aumento da carga de trabalho.

Para saber uma visão geral de como a virtualização funciona, assista ao vídeo: ***Virtualization Deployments***. Disponível em: https://www.youtube.com/watch?v=FZR0rG3HKIk&feature=emb_logo. (IBM CLOUD, 2019).

A virtualização traz vários benefícios para os operadores de *data center* e provedores de serviços:

» **Eficiência de recursos**: antes da virtualização, cada servidor de aplicativos exigia sua própria CPU física dedicada – a equipe de TI comprava e configurava um servidor separado para cada aplicativo que desejava executar. (A TI preferia um aplicativo e um sistema operacional (SO) por computador por motivos de confiabilidade.) Invariavelmente, cada servidor físico seria subutilizado. Por outro lado, a virtualização de servidores permite executar vários aplicativos – cada um em sua própria VM com seu próprio SO – em um único computador físico (geralmente um servidor *x86*) sem sacrificar a confiabilidade. Isso permite a utilização máxima da capacidade de computação do hardware físico.

» **Gerenciamento mais fácil**: a substituição de computadores físicos por VMs definidas por *softwares* facilita o uso e o gerenciamento de políticas escritas em *software*. Isso permite criar fluxos de trabalho automatizados de gerenciamento de serviços de TI. Por exemplo, ferramentas automatizadas de implantação e configuração permitem que os administradores definam coleções de máquinas e aplicativos virtuais como serviços, em modelos de *software*. Isso significa que eles podem instalar esses serviços repetidamente e de forma consistente, sem consumir muito tempo. Os administradores podem usar políticas de segurança de virtualização para estabelecer determinadas configurações de segurança com base na função da máquina virtual. As políticas podem até aumentar a eficiência dos recursos desativando máquinas virtuais não utilizadas para economizar espaço e energia computacional.

» **Tempo de inatividade mínimo**: falhas no sistema operacional e nos aplicativos podem causar tempo de inatividade e prejudicar a produtividade do usuário. Os

administradores podem executar várias máquinas virtuais redundantes entre si e realizar *failover* entre elas quando surgirem problemas. A execução de vários servidores físicos redundantes é mais cara.

» **Aprovisionamento mais rápido:** a compra, instalação e configuração de *hardware* para cada aplicativo é demorada. Desde que o *hardware* já esteja instalado, o provisionamento de máquinas virtuais para executar todos os seus aplicativos é significativamente mais rápido. Você pode até automatizá-lo usando o *software* de gerenciamento e incorporá-lo aos fluxos de trabalho existentes.

Várias empresas oferecem soluções de virtualização que cobrem tarefas específicas do *data center* ou cenários de virtualização de *desktops* focados no usuário final. Exemplos mais conhecidos incluem o *VMware*, especializado em virtualização de servidores, *desktops*, redes e armazenamento; *Citrix*, que tem um nicho na virtualização de aplicativos, mas, também, oferece soluções de virtualização de servidores e *desktops* virtuais; e Microsoft, cuja solução de virtualização *Hyper-V* é fornecida com o Windows e se concentra nas versões virtuais de servidores e computadores de mesa.

3.1. Tipos de virtualização

Até este ponto, discutimos a virtualização de servidores, mas muitos outros elementos da infraestrutura de TI podem ser virtualizados para oferecer vantagens significativas aos gerentes de TI (em particular) e à empresa como um todo. Nesta seção, abordaremos os seguintes tipos de virtualização:

» Virtualização de *desktop*.

» Virtualização de rede.

» Virtualização de armazenamento.

» Virtualização de dados.

» Virtualização de aplicativos.

» Virtualização de *data center*.

» Virtualização de CPU.

» Virtualização de GPU.

» Virtualização Linux.

» Virtualização em nuvem.

3.1.1. Virtualização de *desktop*

A virtualização de área de trabalho permite executar vários sistemas operacionais de área de trabalho, cada um em sua própria VM no mesmo computador.

Existem dois tipos de virtualização de desktop:

» **A infraestrutura de área de trabalho virtual (VDI)** executa várias áreas de trabalho em VMs em um servidor central e as transmite aos usuários que efetuam *login* em dispositivos *thin client*. Dessa maneira, a VDI permite que uma organização forneça aos usuários acesso a uma variedade de sistemas operacionais de qualquer dispositivo, sem instalar sistemas operacionais em nenhum dispositivo.

» **A virtualização da área de trabalho local** executa um hypervisor em um computador local, permitindo ao usuário executar um ou mais sistemas operacionais adicionais nesse computador e alternar de um sistema operacional para outro, conforme necessário, sem alterar nada sobre o sistema operacional principal.

Saiba mais sobre áreas de trabalho virtuais, consulte: ***Desktop* como serviço (DaaS): um guia completo**. Disponível em: https://www.ibm.com/cloud/learn/desktop-as-a-service. (IBM, 2018).

3.1.2. Virtualização de rede

A virtualização de rede usa *software* para criar uma "visão" da rede que um administrador pode usar para gerenciá-la a partir de um único console. Abstrai elementos e funções de *hardware* (por exemplo, conexões, comutadores, roteadores, etc.) e abstrai-os em *software* executado em um hypervisor. O administrador da rede pode modificar e controlar esses elementos sem tocar nos componentes físicos subjacentes, o que simplifica drasticamente o gerenciamento de rede.

Os tipos de virtualização de rede incluem **redes definidas por** *software* (SDN), que virtualizam o *hardware* que controla o roteamento de tráfego de rede (chamado de "plano de controle") e *a* **virtualização de funções de rede (NFV)**, que virtualiza um ou mais dispositivos de *hardware* que fornecem uma rede específica (por exemplo, *firewall*, balanceador de carga ou analisador de tráfego), facilitando a configuração, o provisionamento e o gerenciamento desses dispositivos.

3.1.3. Virtualização de armazenamento

A virtualização de armazenamento permite que todos os dispositivos de armazenamento na rede que estejam instalados em servidores individuais ou unidades de armazenamento

independentes sejam acessados e gerenciados como um único dispositivo de armazenamento. Especificamente, a virtualização de armazenamento reúne todos os blocos de armazenamento em um único *pool* compartilhado, a partir do qual eles podem ser atribuídos a qualquer VM na rede, conforme necessário. A virtualização de armazenamento facilita o provisionamento de armazenamento para VMs e faz uso máximo de todo o armazenamento disponível na rede.

Para uma visão mais detalhada da virtualização de armazenamento, consulte: **Armazenamento na nuvem**. Disponível em: https://www.ibm.com/cloud/learn/cloud-storage. (IBM, 2019).

3.1.4. Virtualização de dados

As empresas modernas armazenam dados de vários aplicativos, usando vários formatos de arquivo, em vários locais, desde a nuvem até os sistemas de *hardware* e *software* no local. A virtualização de dados permite que qualquer aplicativo acesse todos esses dados, independentemente da origem, formato ou local.

As ferramentas de virtualização de dados criam uma camada de *software* entre os aplicativos que acessam os dados e os sistemas que os armazenam. A camada converte a solicitação ou consulta de dados de um aplicativo, conforme necessário, e retorna resultados que podem abranger vários sistemas. A virtualização de dados pode ajudar a quebrar os sigilos de dados quando outros tipos de integração não são viáveis, desejáveis ou acessíveis.

3.1.5. Virtualização de aplicativos

A virtualização de aplicativos executa o *software* do aplicativo sem instalá-lo diretamente no sistema operacional do usuário. Isso difere da virtualização completa da área de trabalho, porque apenas o aplicativo é executado em um ambiente virtual – o sistema operacional no dispositivo do usuário final é executado normalmente. Existem três tipos de virtualização de aplicativos:

- **Virtualização de aplicativo local**: o aplicativo inteiro é executado no dispositivo de nó de extremidade, mas é executado em um ambiente de tempo de execução, e não no *hardware* nativo.
- ***Streaming* de aplicativos**: o aplicativo fica em um servidor que envia pequenos componentes do *software* para executar no dispositivo do usuário final quando necessário.

» **Virtualização de aplicativos baseados**: o aplicativo é executado inteiramente em um servidor que envia apenas sua interface do usuário ao dispositivo cliente.

3.1.6. Virtualização de *data center*

A virtualização de *data center* abstrai a maior parte do *hardware* de um *data center* em *software*, permitindo efetivamente que um administrador divida um único *data center* físico em vários data centers virtuais para diferentes clientes.

Cada cliente pode acessar sua própria infraestrutura como serviço (IaaS), que seria executada no mesmo *hardware* físico subjacente. Os *data centers* virtuais oferecem fácil acesso à computação baseada em nuvem, permitindo que a empresa configure rapidamente um ambiente completo de *data center* sem comprar *hardware* de infraestrutura.

3.1.7. Virtualização de CPU

A virtualização de CPU (unidade central de processamento) é a tecnologia fundamental que possibilita hypervisores, máquinas virtuais e sistemas operacionais. Ele permite que uma única CPU seja dividida em várias CPUs virtuais para uso por várias VMs.

Inicialmente, a virtualização da CPU era totalmente definida por *software*, mas muitos dos processadores atuais incluem conjuntos de instruções estendidos que suportam a virtualização da CPU, o que melhora o desempenho da VM.

3.1.8. Virtualização de GPU

Uma GPU (unidade de processamento gráfico) é um processador multinúcleo especial que melhora o desempenho geral da computação, assumindo o processamento gráfico ou matemático de serviço pesado. A virtualização de GPU permite que várias VMs usem toda ou parte da capacidade de processamento de uma única GPU para vídeo mais rápido, inteligência artificial (AI) e outros aplicativos gráficos ou matemáticos.

» **As GPUs de passagem** disponibilizam a GPU inteira para um único SO convidado.

» **VGPUs compartilhadas** dividem núcleos GPU físicas entre várias GPUs virtuais (vGPUs) para uso por VMs com base em servidor.

Para mais informações sobre GPUs e seus recursos, confira o vídeo: ***GPUs Explained***. Disponível em: https://www.youtube.com/watch?v=LfdK-v0SbGI&feature=emb_logo. (IBM CLOUD, 2019).

3.1.9. Virtualização Linux

O Linux inclui seu próprio hypervisor, chamado de máquina virtual baseada em kernel (KVM), que suporta as extensões de processador de virtualização da Intel e da AMD, para que você possa criar VMs baseadas em x86 a partir de um sistema operacional *host* Linux.

Como sistema operacional de código aberto, o Linux é altamente personalizável. Você pode criar VMs executando versões do Linux personalizadas para cargas de trabalho específicas ou versões protegidas por segurança para aplicativos mais sensíveis.

3.1.10. Virtualização em nuvem

Como vimos anteriormente, o modelo de computação em nuvem depende da virtualização. Ao virtualizar servidores, armazenamento e outros recursos físicos do *data center*, os provedores de computação em nuvem podem oferecer uma variedade de serviços aos clientes, incluindo o seguinte:

- **Infraestrutura como serviço (IaaS)**: servidor virtualizado, armazenamento e recursos de rede que você pode configurar com base em seus requisitos.
- **Plataforma como serviço (PaaS)**: ferramentas de desenvolvimento virtualizadas, bancos de dados e outros serviços baseados em nuvem que você pode usar para criar seus próprios aplicativos e soluções baseados em nuvem.
- ***Software* como serviço (SaaS)**: aplicativos de *software* que você usa na nuvem. O SaaS é o serviço baseado em nuvem mais abstraído do *hardware*.

REFERÊNCIAS

ALLIED. **7 Surprising Facts About Cloud Computing**. 2015. Disponível em: https://www.alliedtelecom.net/facts-about-cloud-computing/. Acesso em 23 mar. 2020.

BELSHE, M.; PEON, R. **Hypertext Transfer Protocol Version 2**. 2015. Disponível em: https://httpwg.org/specs/rfc7540.html. Acesso em 19 mar. 2020.

CONFLUENT. **Featuring Apache Kafka in the Netflix Studio and Finance World.** 2020. Disponível em: https://www.confluent.io/blog/how-kafka-is-used-by-netflix/. Acesso em 28 mar. 2020.

CONVENTIONAL COMMITS. **Conventional Commits 1.0.0-beta.**2. 2020. Disponível em: https://www.conventionalcommits.org/en/v1.0.0-beta.2/. Acesso em 29 out. 2020.

CROCKFORD, D. **Introducing JSON**. 2020. Disponível em: https://www.json.org/json-en.html. Acesso em 19 mar. 2020.

CRUZ, G. **Como é trabalhar como Desenvolvedor(a) Back-End, por Giovanni Cruz**. 2017. Disponível em: https://medium.com/trainingcenter/como-%C3%A9-trabalhar-como-desenvolvedor-backend-b40255d75626. Acesso em 23 mar. 2020.

DAVIS, D; VENNAM, B**. Knative 101: Exercises designed to help you achieve an understanding of Knative.** 2019. Disponível em: https://developer.ibm.com/tutorials/knative-101-labs/. Acesso em 24 mar. 2020.

DEVMEDIA. **O que é HTML5.** 2012. Disponível em: https://www.devmedia.com.br/o-que-e-o-html5/25820. Acesso em 20 mar. 2020.

ECMA. **ECMAScript Language Specification**. 2018. Disponível em: http://ecma-international.org/ecma-262/9.0/index.html#Title. Acesso em 19 mar. 2020.

FREECODCAMP. **Introduction to NPM Scripts**. 2018. Disponível em: https://www.freecodecamp.org/news/introduction-to-npm-scripts-1dbb2ae01633/. Acesso em 29 out. 2020.

FORSUM, G. **Backend infrastructure at Spotify**. 2013. Disponível em: https://labs.spotify.com/2013/03/15/backend-infrastructure-at-spotify/. Acesso em 21 mar. 2020.

FOWLER, C. O programador apaixonado. Casa do Código, 1º ed, 2014.

GITHUB. 2020. Disponível em: https://github.com/nodejs/node. Acesso em 29 out. 2020.

GRUNT. 2020. Disponível em: https://gruntjs.com/. Acesso em 29 out. 2020.

GULP. 2020. Disponível em: https://gulpjs.com/. Acesso em 29 out. 2020.

HOSTGATOR BRASIL**. Tudo o que você precisa para ser um programador front-end**. 2018. Disponível em: https://www.hostgator.com.br/blog/tudo-o-que-voce-precisa-para-ser-um-programador-front-end/. Acesso em 23 mar. 2020.

IBM. **Cloud Storage**. 2019. Disponível em: https://www.ibm.com/cloud/learn/cloud-storage. Acesso em 25 mar. 2020.

IBM. **Hypervisors**. 2019. Disponível em: https://www.ibm.com/cloud/learn/hypervisors. Acesso em 25 mar. 2020.

IBM. **Istio**. 2019. Disponível em: https://www.ibm.com/cloud/learn/istio. Acesso em 24 mar. 2020.

IBM. **Microservices.** Disponível em: https://www.ibm.com/cloud/learn/microservices. Acesso em 24 mar. 2020.

IBM. **Virtual Machines (VMs)**. 2019. Disponível em: https://www.ibm.com/cloud/learn/virtual-machines. Acesso em 25 mar. 2020.

IBM. **Desktop-as-a-Service (DaaS).** 2018. Disponível em: https://www.ibm.com/cloud/learn/desktop-as-a-service. Acesso em 25 mar. 2020.

IBM. **The state of container-based application development**. 2017. Disponível em: https://www.ibm.com/cloud-computing/info/container-development/. Acesso em 24 mar. 2020.

IBM Cloud. Containerization Explained. 2019. Disponível em: https://www.youtube.com/watch?v=0qotVMX-J5s&feature=emb_logo. Acesso em 29 out. 2020.

IBM Cloud. Kubernetes Explained. 2019. Disponível em: https://www.youtube.com/watch?v=aSrqRSk43lY&feature=emb_logo. Acesso em 29 out. 2020.

IBM Cloud. Implantações Kubernetes: comece rapidamente. 2019. Disponível em: https://www.youtube.com/watch?v=Sulw5ndbE88&feature=emb_logo. Acesso em 29 out. 2020.

IBM Cloud. Virtualization Explained. 2019. Disponível em: https://www.youtube.com/watch?v=FZR0rG3HKIk&feature=emb_logo. Acesso em 29 out. 2020.

IBM Cloud. GPUs: Explained. 2019. Disponível em: https://www.youtube.com/watch?v=LfdK-v0SbGI&feature=emb_logo. Acesso em 29 out. 2020.

IGTI BLOG. **O que faz um Desenvolvedor Full Stack?** 2018. Disponível em: https://www.igti.com.br/blog/o-que-faz-um-desenvolvedor-full-stack/. Acesso em 23 mar. 2020.

IPERIUS BACKUP BRASIL. **Computação em nuvem: Inovações em TI para impulsionar os negócios**. 2019. Disponível em: https://www.iperiusbackup.net/pt-br/computacao-em-nuvem-inovacoes-em-ti-para-impulsionar-os-negocios/. Acesso em: 24 mar. 2020.

KATZ, B. **How Much Does It Cost to Fight in the Streaming Wars?** 2019. Disponível em: https://observer.com/2019/10/netflix-disney-apple-amazon-hbo-max-peacock-content-budgets/. Acesso em 28 mar. 2020.

KIM, J. **Kubernetes Networking: a lab on basic networking concepts**. 2019. Disponível em: https://developer.ibm.com/tutorials/kubernetes-networking-101-lab/. Acesso em 24 mar. 2020.

LINDLEY, C. **Front-End Developer Handbook 2018**. 2018. Disponível em: https://frontendmasters.com/books/front-end-handbook/2018/. Acesso em 18 mar. 2020.

LINTHICUM, D. **The essential guide to software containers for application development**. 2020. Disponível em: https://techbeacon.com/enterprise-it/essential-guide-software-containers-application-development. Acesso em 24 mar. 2020.

MAMP. 2020. Disponível em: https://www.mamp.info/en/windows/. Acesso em 29 out. 2020.

MATTOKA, M. **Are you sure you're using microservices?** 2019. Disponível em: https://blog.softwaremill.com/are-you-sure-youre-using-microservices-f8d4e912d014. Acesso em 28 mar. 2020.

MDN. **CSS Reference**. 2020. Disponível em: https://developer.mozilla.org/en-US/docs/Web/CSS/Reference. Acesso em 19 mar. 2020.

MONTEIRO, C. **Desenvolvimento full stack: o que significa ser um profissional completo**. 2019. Disponível em: https://movile.blog/desenvolvimento-full-stack-o-que-significa-ser-um-profissional-completo/. Acesso em 23 mar. 2020.

NARKNEDE, N; SHAPIRA, G; PALINO, T. **Kafka: The definitive guide: real-time data and streams Processing at scale.** 2017. O'Reilly Media, Inc.

NEEMAN, T. **Debug and log your Kubernetes applications.** 2019. Disponível em: https://developer.ibm.com/tutorials/debug-and-log-your-kubernetes-app/. Acesso em 24 mar. 2020.

NETFLIX. **Evolution of the Netflix Data Pipeline**. 2016. Disponível em: https://netflixtechblog.com/evolution-of-the-netflix-data-pipeline-da246ca36905. Acesso em 28 mar. 2020.

NETFLIX. **Spark and Spark Streaming at Netfix-(Kedar Sedekar and Monal Daxini, Netflix)**. 2015. Disponível em: https://www.slideshare.net/SparkSummit/spark-and-spark-streaming-at-netfix-sedakar-daxini. Acesso em 28 mar. 2020.

NODEJS. 2020. Disponível em: https://nodejs.org. Acesso em 29 out. 2020.

OLIVEIRA, W. **Como é trabalhar como Desenvolvedor Front-End, por Felipe Fialho**. 2017. Disponível em: https://medium.com/trainingcenter/como-%C3%A9-trabalhar-como-desenvolvedor-front-end-por-felipe-fialho-1e3efbadef90. Acesso em 23 mar. 2020.

PAGANI, T. **Documentos acessíveis com WAI-ARIA em HTML5.** 2011. Disponível em: https://tableless.com.br/documentos-acessiveis-com-aria-em-html5/. Acesso em 20 mar. 2020.

PARIS-WHITE, D. **Scale security while innovating microservices fast.** 2018. Disponível em: https://www.ibm.com/cloud/blog/scale-security-innovating-microservices-fast. Acesso em 24 mar. 2020.

PINTEREST. 2020. Disponível em: https://i.pinimg.com/564x/57/b8/05/57b805a0f624c0837b10e0c4cc5b1c55.jpg. Acesso em 29 out 2020.

REDMONK. **The Continued Rise of Apache Kafka.** 2017. Disponível em: https://redmonk.com/fryan/2017/05/07/the-continued-rise-of-apache-kafka/. Acesso em 28 mar. 2020.

SHAPLAND, R; COLE, B. **What are cloud containers and how do they work?** 2019. Disponível em: https://searchcloudsecurity.techtarget.com/feature/Cloud-containers-what-they-are-and-how-they-work. Acesso em 24 mar. 2020.

SOMAN, C. *et al.* **uReplicator: Uber Engineering's Robust Apache Kafka Replicator**. 2016. Disponível em: https://eng.uber.com/ureplicator-apache-kafka-replicator/. Acesso em 29 mar. 2020.

STORYBOOK. Build bulletproof UI components faster. 2020. Disponível em: https://storybook.js.org/. Acesso em 29 out. 2020.

TAKE BLOG. **Como se tornar um desenvolvedor back-end? 4 dicas para começar a sua carreira**. 2019. Disponível em: https://take.net/blog/devs/desenvolvedor-back-end/. Acesso em 23 mar. 2020.

TUTORIALSTEACHER.COM. **JavaScript Tutorial**. 2020. Disponível em: https://www.tutorialsteacher.com/javascript/javascript-tutorials. Acesso em 29 out. 2020.

TUTORIALSTEACHER.COM. **Node.js Process Model**. 2020. Disponível em: https://www.tutorialsteacher.com/nodejs/nodejs-process-model. Acesso em 26 mar. 2020.

VENNAM, S. **Kubernetes Clusters: Architecture for Rapid, Controlled Cloud App Delivery.** 2019. Disponível em: https://www.ibm.com/cloud/blog/new-builders/kubernetes-clusters-architecture-for-rapid-controlled-cloud-app-delivery. Acesso em 24 mar. 2020.

WACHAL, M. **Digital transformation with streaming application development.** 2019. Disponível em: https://blog.softwaremill.com/digital-transformation-with-streaming-application-development-100fb4189149. Acesso em 28 mar. 2020.

WAMP SERVER. 2020. Disponível em: https://www.wampserver.com/en/. Acesso em 29 out. 2020.

WARSKI, A. **Evaluating persistent, replicated message queues.** 2017. Disponível em: https://softwaremill.com/mqperf/. Acesso em 28 mar. 2020.

WARSKI, A. **Event sourcing using Kafka.** 2018. Disponível em: https://blog.softwaremill.com/event-sourcing-using-kafka-53dfd72ad45d. Acesso em 28 mar. 2020.

WEBPACK. 2020. Disponível em: <https://webpack.js.org/. Acesso em 29 out. 2020.

WHATWG. **DOM Living Standard**. 2020. Disponível em: https://dom.spec.whatwg.org/. Acesso em 19 mar. 2020.

WHATWG. **HTTP Living Standard**. 2020. Disponível em: https://html.spec.whatwg.org/multipage/introduction.html#introduction. Acesso em 19 mar. 2020.

WHATWG. **URL Living Standard**. 2020. Disponível em: https://url.spec.whatwg.org/. Acesso em 19 mar. 2020.

WODEHOUSE, C. **The role of a front-end web developer: creating user experience & interactivity**. 2015. Disponível em: https://www.upwork.com/hiring/development/front-end-developer/. Acesso em 18 mar. 2020.

W3C BRASIL. **Cartilha de Acessibilidade na Web.** 2020. Disponível em: https://ceweb.br/cartilhas/cartilha-w3cbr-acessibilidade-web-fasciculo-I.html. Acesso em 20 mar. 2020.

W3 SCHOOLS. **Cloud Computing Architecture**. 2020. Disponível em: https://www.w3schools.in/cloud-computing/cloud-computing-architecture/. Acesso em 23 mar. 2020.

W3TECHS. **Usage statistics of PHP for websites**. 2020. Disponível em: https://w3techs.com/technologies/details/pl-php. Acesso em 21 mar. 2020.

WOODIE, A. **The Real-Time Rise of Apacha Kafka.** 2016. Disponível em: https://www.datanami.com/2016/04/06/real-time-rise-apache-kafka/. Acesso em 28 mar. 2020.