KLS

Sistemas Distribuídos

Sistemas Distribuídos

Caique Silva Pereira

Dados Internacionais de Catalogação na Publicação (CIP)

Pereira, Caique Silva
P436s Sistemas distribuídos / Caique Silva Pereira. – Londrina : Editora e Distribuidora Educacional S.A., 2019.
184 p.

ISBN 978-85-522-1443-4

1. Sistemas distribuídos. 2. Modelos de sistemas. 3. Comunicação entre processos. I. Pereira, Caique Silva. II. Título.

CDD 620

Thamiris Mantovani CRB-8/9491

2019
Editora e Distribuidora Educacional S.A.
Avenida Paris, 675 – Parque Residencial João Piza
CEP: 86041-100 — Londrina — PR
e-mail: editora.educacional@kroton.com.br
Homepage: http://www.kroton.com.br/

Sumário

Palavras do autor

Caro estudante, seja bem-vindo aos estudos dos sistemas distribuídos! Atualmente, com a expansão e disseminação da Internet, ao falarmos em "sistemas distribuídos" para um profissional de TI, ele já imagina redes de computadores, conexões, computação em nuvem, entre vários outros termos da área, que remetem à ideia do tema. Diversas aplicações atuais fazem uso do conceito de sistema distribuído, incluindo jogos multiplayer online, sistemas de transações financeiras, motores de busca utilizados por sites de pesquisa e a própria Internet. Diante desse universo, é imprescindível que você, profissional de tecnologia, compreenda tais sistemas, se apropriando dos seus diversos componentes, que possibilitam criar sistemas robustos e resistentes a falhas e que serão utilizados nas mais variadas aplicações modernas.

Como ocorre em toda implantação de um sistema ou aplicação, temos objetivos bem definidos a serem alcançados e desafios a serem superados. Além disso, devemos identificar o melhor modelo a ser adotado, de acordo com a aplicação desenvolvida. Atualmente, a maioria dos sistemas são implementados em plataformas virtuais com tecnologias de software que fornecem contêineres – plataformas virtuais otimizadas para gerenciar aplicações de maneira otimizada –, que geralmente são disponibilizadas em serviços de *cloud* (nuvem), razão pela qual a segurança desses sistemas é um ponto crítico: saber identificar as principais ameaças e os tipos de ataque é fundamental para a implantação de um sistema seguro.

Esse livro didático está dividido em quatro unidades da seguinte forma:

- Na Unidade 1, veremos os conceitos fundamentais e arquiteturas de sistemas distribuídos, assim como definições e exemplos do dia a dia.
- Na Unidade 2, serão definidos os objetivos principais, tais como compartilhamento de recursos, confiabilidade, desempenho e alta performance, os desafios mais recorrentes e seus aspectos de projeto.
- Na Unidade 3, veremos como esses sistemas distribuídos podem ser implantados por meio de tecnologias de virtualização e conteinerização, mais especificamente o Docker, incluindo o procedimento de instalação e utilização dos principais comandos.

- Na Unidade 4, veremos alguns aspectos de segurança, como as principais vulnerabilidades encontradas em sistemas distribuídos e, após a análise de segurança, podemos finalmente avançar para utilização de tecnologias que permitem uma integração mais sólida entre os nós da rede, como o RPC, e seus diferentes mecanismos de implementação, como o JRMI.

Agora, chegou a hora de arregaçarmos as mangas e seguirmos em frente com o estudo dos sistemas distribuídos, o que lhe proporcionará um leque de oportunidades no mercado de trabalho, assim como conhecimentos importantes em relação a tecnologias atuais. Importante frisar que todas essas possibilidades dependem, em grande parte, de seu empenho e dedicação aos estudos.

Unidade 1

Conceitos e arquitetura de sistemas distribuídos

Convite ao estudo

Você sabia que entender os conceitos de sistemas distribuídos é de extrema importância atualmente? Grande parte das aplicações mais populares utilizam esse modelo, em que várias máquinas são interligadas por meio de redes de computadores. Você já parou para pensar como os sistemas computacionais evoluíram até os dias atuais? Nesta unidade, vamos compreender essa evolução, passando por todas as suas etapas até chegar nos sistemas distribuídos. Você sabe classificar os diferentes sistemas computacionais e distinguir suas características? Após completar esta Unidade, você será capaz de realizar essa tarefa, utilizando seu raciocínio crítico e criatividade para resolução dos problemas propostos.

É fato que a demanda por profissionais com conhecimento em implantação de sistemas distribuídos tem aumentado principalmente em consultorias de TI que prestam serviços especializados para seus clientes nos mais diversos segmentos. Você iniciou seu estágio em uma dessas consultorias e, após o período de integração na empresa, sua primeira atividade externa será avaliar os ativos de TI de um cliente dessa consultoria. Será possível ganhar a confiança do cliente com seus conhecimentos sobre sistemas distribuídos?

Iniciaremos agora os estudos sobre os sistemas distribuídos, o que lhe proporcionará um leque de oportunidades no mercado de trabalho, assim como conhecimentos importantes em relação a tecnologias atuais. Importante frisar que quanto mais você se dedicar, mais poderá aproveitar os ensinamentos transmitidos neste material.

Seção 1.1

Fundamentação de sistemas distribuídos

Diálogo aberto

Caro estudante, iniciamos agora nossa primeira seção no estudo de sistemas distribuídos. Você já ouviu falar de um jogo chamado *Smite*? Esse jogo on-line utiliza um sistema distribuído em sua arquitetura. Mas, afinal, o que é um sistema distribuído? Para compreender essa definição, nesta seção vamos trabalhar conceitos fundamentais dos sistemas computacionais, incluindo os tipos de interligações (arquiteturas existentes). Você já se sentiu incomodado por não conseguir compreender um sistema ou aplicação mais complexa, formada por mais de um computador? Pois bem, na situação-problema à qual você será exposto, terá chance de trabalhar os conceitos que lhe permitirão familiarizar-se de maneira gradual com esses sistemas e aplicações de grande porte e complexidade. Vamos lá!

Você recebeu sua primeira missão em seu novo estágio na consultoria. Sua primeira atividade externa será avaliar os ativos de TI de um cliente dessa consultoria, que possui três lojas de varejo, porém não possui departamento (nem conhecimento) de informática. Essas lojas funcionam em regime familiar, e o dono é um senhor que, apesar de estar prestes a se aposentar, continua sempre querendo fazer o negócio da sua família crescer. O dono das lojas, não conhece nada de TI, então você decidiu explicar como os computadores se comunicam entre si, afinal, o patriarca das lojas provavelmente deve ter ouvido falar de algumas coisas sobre equipamentos de redes, e você sabe que, dessa forma, aumenta as chances de ganhar a simpatia do dono, o que é importante para o sucesso de uma possível parceria com a consultoria na qual você trabalha. Sendo assim, elabore uma apresentação no formato de slides, juntamente com um relatório técnico sobre como a informação é transmitida na comunicação entre dois computadores.

Agora, utilize o conteúdo da seção para solucionar a situação problema apresentada acima. Com certeza você será capaz de elaborar uma boa apresentação que impressione seu cliente!

Não pode faltar

Os sistemas distribuídos são muito utilizados pelos desenvolvedores nos mais variados tipos de aplicações. Mas, afinal, o que são sistemas distribuídos e por que são tão importantes?

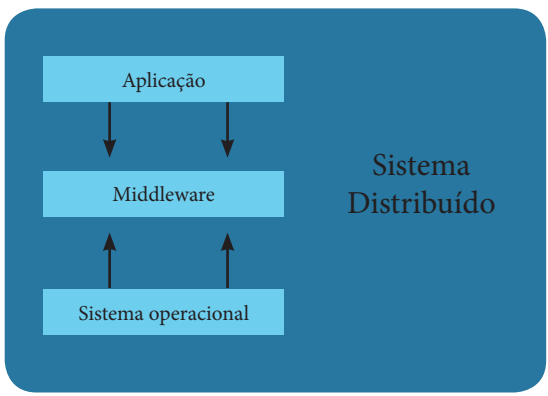
Um sistema distribuído é um conjunto de computadores interligados via rede, mas, para o usuário final das aplicações, que são executadas através deles, aparenta ser um sistema único (TANENBAUM; STEEN, 2008). Um de seus principais aspectos é que os computadores que fazem parte de sistemas distribuídos têm o funcionamento independente, ou seja, cada um age por si próprio, e muitas vezes os sistemas e os hardwares dessas máquinas são totalmente diferentes, mas ainda assim eles aparentam ao usuário serem uma coisa só. Esses computadores estão ligados por meio de rede, o que possibilita seu funcionamento de forma distribuída.

Assimile

As principais aplicações e sistemas da atualidade utilizam o modelo de sistemas distribuídos. Esse conceito faz com que o sistema não dependa apenas de uma máquina, pois toda a carga do ambiente estará distribuída entre várias máquinas. Então, é importante assimilar que os sistemas distribuídos ocultam o conjunto de máquinas que o fazem funcionar, aparentando ser algo único.

A camada de middleware é um dos fatores principais para o bom funcionamento de aplicações distribuídas. Essa camada é um software que está entre os programas criados e o sistema operacional, ou seja, é uma camada central, que tem a função de permitir que haja um gerenciamento de dados e uma comunicação entre camadas para o funcionamento de aplicações distribuídas. O middleware funciona como uma camada de tradução para interligar o sistema operacional com os programas (COULOURIS et al., 2013). A Figura 1.1 representa as camadas que compõem um sistema distribuído.

Figura 1.1 | Camadas que compõem um sistema distribuído



Fonte: elaborada pelo autor.

As mais diversas redes sociais que utilizamos hoje em dia com conteúdo multimídias são exemplos de sistemas distribuídos, assim como sites de pesquisas e plataformas de vídeos online.

Exemplificando

Vamos imaginar que queremos ver a página de notícias esportivas do nosso time do coração em um portal de notícias X, que segue o modelo de sistemas distribuídos. Sendo assim, a página acessada está sendo fornecida por um conjunto de quatro servidores que, para o usuário final, vão aparentar ser algo único. Pensando nisso, uma das funções dos sistemas distribuídos é que mesmo que algum dos quatro servidores que estão mantendo essa página no ar esteja desligado, os outros devem assumir sua função, dividindo a carga, e o conteúdo deve estar no ar e o sistema em funcionamento.

"Os modelos de sistemas distribuídos servem para definir a forma como os diversos objetos distribuídos se comunicam e interagem entre si" (TANENBAUM; STEEN, 2008, p. 2). Os sistemas distribuídos seguem tendências e modelos para seu funcionamento. Por exemplo, se pensarmos em variados sistemas operacionais, a ação de abrir uma pasta é representada por dois cliques do mouse.

Quando trabalhamos com sistemas distribuídos temos objetivos claros a serem alcançados em nosso sistema, com sua criação, a saber:

- Disponibilidade alta e fácil acesso ao sistema e todos os seus recursos, tanto pelas máquinas que fazem parte do sistema distribuído como pelo usuário final.
- Devemos, também, ocultar ao usuário que os recursos de nosso sistema são distribuídos; essa é uma característica muito importante.
- O sistema distribuído deve ser aberto, ou seja, deve facilitar a inclusão de novas máquinas e recursos no ambiente em funcionamento, e, sendo assim, esse sistema pode ser expandido facilmente.

Reflita

Você conseguiu ver alguns conceitos de sistemas distribuídos e agora pode imaginar o quanto esse conceito é utilizado em aplicações modernas. Agora que você já os conhece, consegue identificar entre os serviços que mais utiliza quais seguem esse modelo?

Além disso, será que algum serviço que você utiliza pode ser otimizado aplicando esses conceitos? Vamos refletir.

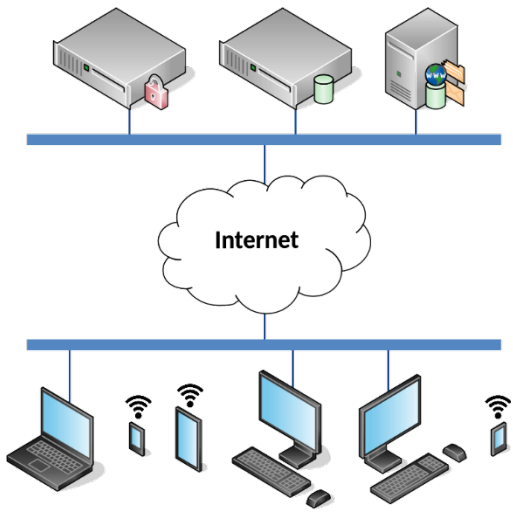
Modelos de arquitetura de redes de computadores

Atualmente, existem três tipos de arquiteturas de computadores (MAIA, 2013): a "clássica" cliente-servidor, a ponto a ponto e a descentralizada. Vejamos agora suas principais características e exemplos cotidianos reais em que essas arquiteturas são utilizadas.

Arquitetura cliente-servidor

Essa talvez seja a arquitetura mais conhecida e utilizada nos sistemas informatizados, tendo sua origem na década de 70. A Figura 1.2 demostra a topologia física dessa arquitetura.

Figura 1.2 | Arquitetura cliente-servidor



Fonte: elaborada pelo autor.

Nesse tipo de arquitetura, teremos alguns serviços e recursos a serem compartilhados entre vários usuários, que podem ser disponibilizados em um único computador, chamado de servidor multisserviço, ou em computadores segregados, ou seja, um serviço ou recurso por computador, neste caso, sendo chamados de acordo com os serviços disponibilizados (por exemplo, servidor de banco de dados, servidor de autenticação de usuários, etc.).

Esses servidores estarão, em geral, conectados à Internet por equipamentos de rede, como switches, roteadores e firewalls, para que as pessoas possam acessá-los remotamente e utilizar os serviços e recursos disponibilizados

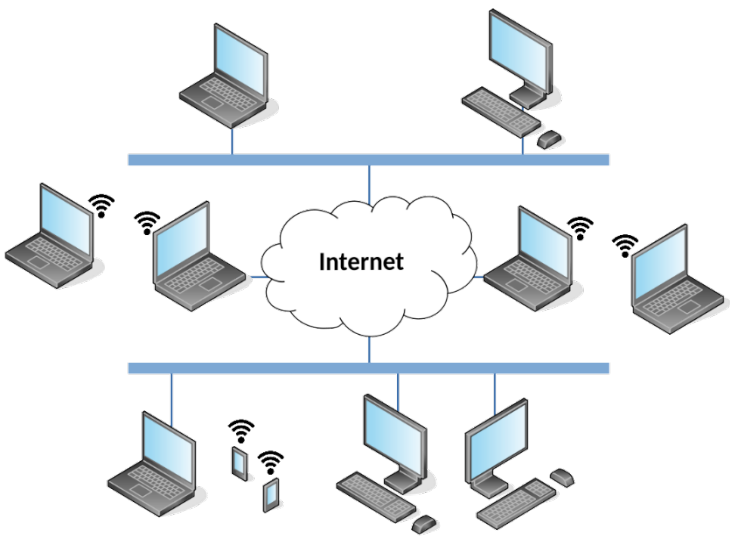
por eles em máquinas denominadas cliente (por exemplo, laptops, desktops, smartphones e tablets), que, por sua vez, também estarão conectadas por equipamentos de rede à Internet. Importante observar que essa arquitetura, na maioria das vezes, também funciona no nível de rede local (do inglês, *Local Area Network* – LAN).

Você faz uso da arquitetura cliente-servidor na maior parte do tempo, quando acessa, por meio do navegador de Internet instalado no seu laptop, um website de comércio eletrônico, quando verifica seus e-mails no aplicativo no smartphone ou quando joga um game online no console de videogame (por exemplo, ao jogar o *Call of Duty* em um console de videogame).

Arquitetura ponto a ponto

A arquitetura ponto a ponto, também conhecida como arquitetura *peer-to-peer*, ou simplesmente P2P, teve sua origem na década de 80. A Figura 1.3 demostra a topologia física dessa arquitetura.

Figura 1.3 | Arquitetura ponto a ponto



Fonte: elaborada pelo autor.

Nesse tipo de arquitetura, os computadores, sejam eles laptops, desktops, smartphones ou tablets, possuem o mesmo papel nessa rede, ou seja, funcionam como dispositivos finais e como servidores, uma vez que, um mesmo computador pode disponibilizar recursos e serviços para um computador e, similarmente, pode utilizar (diz-se também consumir) recursos e serviços de um computador nessa rede. Esses computadores também estarão

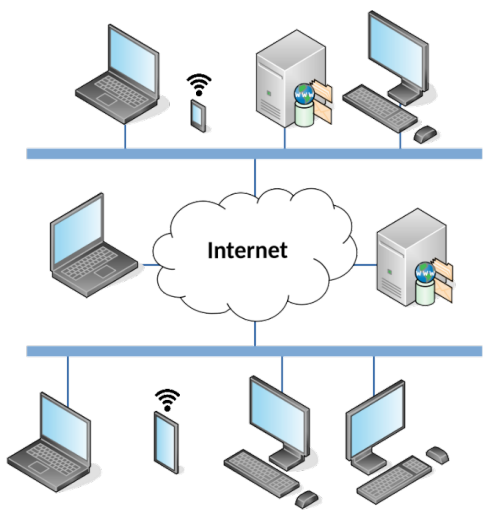
conectados à Internet por meio de equipamentos de rede, como switches, roteadores e firewalls, para que as pessoas possam compartilhar os recursos e serviços. Importante observar que essa arquitetura, na maioria das vezes, também funciona no nível de rede local.

Você faz uso desse tipo de arquitetura quando compartilha uma foto, via Bluetooth, com um colega que tenha pareado o aparelho dele na rede Bluetooth com o seu. Um outro exemplo comum é quando você faz o download de um arquivo muito grande, tal como a imagem de um sistema operacional, por meio de um arquivo torrent dessa imagem (por exemplo, baixar uma imagem do Ubuntu Desktop para testar esse sistema GNU/Linux em uma máquina virtual). A vantagem de utilizar esse tipo de arquitetura em relação à cliente-servidor – para compartilhar arquivos – é que, tipicamente, o download é concluído muito mais rápido, pelo fato de você baixar diferentes partes do arquivo de diversas fontes (outros computadores que possuem esse mesmo arquivo) na Internet, utilizando para tal o protocolo BitTorrent, através de suas diversas implementações, como o μTorrent.

Arquitetura descentralizada

Arquitetura mais recente, a partir dos anos 2000, pode ser vista como uma arquitetura híbrida entre a cliente-servidor e a ponto a ponto. A Figura 1.4 demostra a topologia física dessa arquitetura.

Figura 1.4 | Arquitetura descentralizada



Fonte: elaborada pelo autor.

Na arquitetura descentralizada, os computadores são os próprios servidores da aplicação (serviço ou recurso a ser compartilhado), o que se assemelha à arquitetura ponto a ponto. Entretanto, diferentemente do que ocorre na arquitetura ponto a ponto, o estado da aplicação (por exemplo, os valores atuais das variáveis utilizadas em tal aplicação) é replicado entre todos os computadores na rede, de maneira que exista um consenso entre eles nessa rede. Há ainda a possibilidade de alguns computadores simplesmente utilizarem a aplicação (em vez de serem servidores dela), tendo o papel puramente de "clientes" nesse caso, o que remete à arquitetura cliente-servidor.

Essa é a arquitetura utilizada pelas plataformas baseadas em Blockchain, que têm se tornado mais populares após o advento do Bitcoin. Aplicações que funcionam sobre esse tipo de plataforma são chamadas de *dApps* (do inglês, *Decentralized Application*). Uma das principais vantagens ao se utilizar esse tipo de arquitetura é que não há uma entidade que controle sua aplicação, como tipicamente ocorre nas arquiteturas cliente-servidor. Por exemplo, seus e-mails do Gmail são controlados pela Google, que armazena os dados e poderia, hipoteticamente, acessar dados privados ou decidir encerrar seus serviços, situação na qual você estaria impossibilitado de optar por decisão contrária, caso não seja um acionista da empresa. Na arquitetura descentralizada, há então uma garantia de transparência aos usuários de uma determinada aplicação, uma vez que a aplicação e seus dados são armazenados por computadores dos próprios usuários que participam da rede, e não por uma empresa apenas.

Exemplos desse tipo de arquitetura podem ser vistos em qualquer sistema de transações financeiras das chamadas criptomoedas (como o próprio Bitcoin), mas não se limita apenas a esse tipo de aplicação. Um exemplo é a *Steemit*, aplicação de rede social, similar ao Twitter, mas que faz uso da arquitetura descentralizada.

Integração de sistemas computacionais

A integração de sistemas computacionais refere-se a formas e, por extensão, a tecnologias utilizadas para que haja a comunicação ou, em palavras mais simples, a troca de informação entre dois sistemas computacionais. Você já parou para pensar como é a comunicação entre o aplicativo do Facebook instalado no seu smartphone e os servidores da plataforma? É desse tipo de comunicação que estamos falando ao tratar da integração desses sistemas.

Normalmente, as aplicações, principalmente web, podem ser divididas em duas partes: a chamada *front-end* (ou *client-side*) e a *back-end* (ou

server-side). A parte do *front-end* é aquela na qual você interage ao visitar um website por meio do navegador de internet ou pelo aplicativo instalado em seu smartphone. Já a parte *back-end* é aquela que cuida da parte das regras de negócio da aplicação, em que é definido o que deve acontecer quando você efetua o login na tela inicial do Facebook, por exemplo, e também é a parte que lida com o banco de dados, em que as informações do usuário e senha são armazenadas.

Normalmente, a integração entre essas duas partes se dá por meio do protocolo HTTP (o protocolo padrão utilizado na navegação web), que faz com que, ao clicar no botão de login, seus dados sejam enviados, normalmente por meio da Internet, para que a aplicação possa validar tais informações e liberar (ou negar) seu acesso ao sistema.

Mais adiante nessa disciplina, veremos em detalhes como essa troca de informações é realizada, quais são os formatos dos dados transmitidos e veremos também que a comunicação por meio do protocolo HTTP é uma das formas de integração, mas não a única.

Continue firme em seus estudos, pois ainda vem muita coisa interessante e útil pela frente!

Pesquise mais

Você pode fixar seu conhecimento assistindo ao vídeo indicado, que aborda alguns conceitos básicos de sistemas distribuídos.

SISTEMAS DISTRIBUÍDOS - LICENCIATURA EM COMPUTAÇÃO. **Aula 1 -** Sistemas Distribuídos - Conceitos Básicos. 12 ago. 2014.

Assim finalizamos esta seção. Espero que o conhecimento adquirido seja de importância para seu crescimento profissional. Bons estudos e até a próxima seção!

Sem medo de errar

Atuando na sua primeira consultoria, você se depara com um dono de lojas de varejo que está com muitas dúvidas sobre como funciona a comunicação entre computadores. Vamos ajudá-lo a entender melhor esse assunto? Para isso, vamos elaborar uma apresentação e um relatório que deixarão tudo mais claro para ele. Para elaborar o relatório, no qual apresentará um pouco de como os computadores se comunicam, utilize os conteúdos apresentados nesta seção.

Para ajudá-lo nessa tarefa, temos as seguintes sugestões:

- Você pode tirar ideias para seu relatório lendo este artigo sobre a história das redes de computadores.
- FIGUEIREDO, I. L. História das redes de computadores. **Oficina da net,** [S.l.], 15 mar. 2013.
- Você pode tirar novas ideias, assistindo a esse vídeo que fala de evolução histórica de redes de computadores.
- PREPARACAODIGITAL. **Redes e internet.** 12 jun. 2013.
- Em seu relatório, nessa primeira fase, tente ser bem detalhista e não utilizar muitos termos técnicos para que uma pessoa leiga na área consiga entender.
- Você pode utilizar os modelos de arquiteturas de computadores apresentados na seção (cliente-servidor, arquitetura descentralizada e ponto a ponto) para enriquecer seu relatório técnico.
- Você pode tirar novas ideias desse vídeo que fala de evolução histórica de redes de computadores:
- JOÃO RUBENS MARCHETE FILHO. **Princípios de Desenvolvimento Web** - Arquitetura Cliente Servidor (Parte 1). 19 ago. 2015.
- Você pode consultar na internet outras imagens dos modelos de arquitetura apresentados nesta seção para dar ao cliente uma visão diferente em sua apresentação. Por exemplo, pesquisar na web imagens que mostrem a ligação dos computadores e servidores por meio da arquitetura do tipo cliente-servidor, ou pesquisar na web imagens que ilustrem a ligação dos computadores na arquitetura do tipo ponto a ponto.
- Você também pode pesquisar como funciona a comunicação entre computadores por meio de rede, desde seu início, e trazer em sua apresentação uma evolução. Você pode descrever como funcionava a comunicação entre computadores em tempos primórdios, por exemplo, entre os computadores do modelo Eniac (1946), primeiro computador do mundo, ou entre computadores do modelo IBM 360, um pouco mais evoluídos.
- Pense, por exemplo, em um processo envolvendo o acesso restrito a uma página, por meio de login e senha, e como ocorre a resposta do servidor dizendo se o login ocorreu com sucesso ou não.

Não se esqueça de organizar as informações em uma apresentação e em um relatório técnico, pois todo projeto precisa ser devidamente documentado.

Avançando na prática

Acessando um site

Descrição da situação-problema

Navegar pela Internet se tornou uma tarefa rotineira. Basta ter uma dúvida, uma curiosidade, que já recorremos a um mecanismo de busca. Entre os vários serviços oferecidos na Internet, sites que apresentam dados de previsão meteorológica são bastante requisitados. Pense em um usuário acessando um desses sites, que utiliza o modelo de arquitetura de computadores do tipo cliente-servidor. O usuário quer verificar a previsão do tempo para sua cidade, que é um conteúdo desse site. Sua tarefa será descrever, passo a passo, todas as etapas que ocorrem nesse processo de comunicação entre a máquina cliente (usuário) e a máquina servidor (site).

Resolução da situação-problema

O processo de requisição de um serviço entre cliente e servidor pode ser descrito, em linhas gerais, em seis passos.

1 - Máquina Cliente: o navegador chama a URL referente ao site. Nesse processo, são envolvidos diversos recursos computacionais, como os protocolos de comunicação, os meios de transmissão e o hardware da rede, que envolve diversos equipamentos e softwares.

2 - Máquina Cliente: é enviada uma requisição do tipo HTTP para o Servidor, isso quer dizer que vamos utilizar o protocolo de transferência de páginas de internet para nos ajudar com o acesso.

3 - Máquina Servidor: recebemos uma requisição do tipo HTTP vinda da máquina Cliente.

4 - Máquina Servidor: enviamos uma resposta via protocolo HTTP com a página que a máquina Cliente quer acessar.

5 - Máquina Cliente: recebe a resposta via protocolo HTTP da máquina Servidor.

6 - Máquina Cliente: faz o download da página recebida e apresenta o resultado, que será uma página web com conteúdo multimídia, como imagens, vídeos, links, textos, etc.

Com o avanço nos estudos, a descrição de um procedimento de requisição de serviços poderá ficar mais detalhada, pois, como sabemos, existem muitos componentes e sistemas envolvidos nesse processo.

Faça valer a pena

1. Os sistemas distribuídos têm muitas características semelhantes a quaisquer outros sistemas de rede. Entretanto, uma diferença principal está no fato de que, nos sistemas distribuídos, a aplicação é replicada (ou distribuída) entre as diferentes máquinas, comportando-se como se estivesse rodando em uma única máquina.

Considerando os conceitos inerentes aos sistemas distribuídos, analise as afirmações abaixo e escolha a opção correta.

I - As aplicações se comportam como se estivessem rodando em uma única máquina.
II - Os sistemas distribuídos possuem um elemento central (intermediário), conhecido como middleware, entre o sistema operacional e a aplicação.
III - Sistemas distribuídos são, necessariamente, sistemas que possuem sistemas operacionais especiais.
IV - Os sistemas distribuídos são utilizados em aplicações modernas de grande porte.
V - Uma das características dos sistemas distribuídos é a facilidade para incluir novas máquinas em um sistema em funcionamento.

a) Somente a afirmação I está correta.
b) Somente as afirmações II e III estão corretas.
c) Somente a afirmação III está incorreta.
d) Somente as afirmações I, II e V estão corretas.
e) Todas as afirmações estão corretas

2. As arquiteturas de computadores são fundamentais em todas as áreas da tecnologia da informação, e os variados tipos existentes são escolhidos de acordo com o projeto proposto, pensando em atender à demanda do projeto e ter um bom funcionamento. Uma das arquiteturas mais populares tem como desvantagem o conceito de que as máquinas clientes podem gerar requisições, mas não podem oferecê-las a outras máquinas, sobrecarregando assim o servidor.

Identifique a qual arquitetura de computadores a desvantagem citada se refere:
a) Ponto a ponto.
b) Cliente-servidor.
c) Arquitetura descentralizada.
d) Sistemas distribuídos.
e) Arquitetura híbrida.

3. Quando falamos de sistemas distribuídos, somos obrigados a falar de middleware. Esse software é muito importante para que um sistema distri-

buído consiga atingir suas metas e objetivos e é essencial para o funcionamento das aplicações distribuídas.

O middleware tem várias funções dentro de uma aplicação que utiliza os conceitos de sistemas distribuídos. Identifique a alternativa que corresponde a uma dessas funções.
a) O middleware conecta a camada de cliente com a camada de servidor.
b) O middleware conecta duas máquinas em uma arquitetura ponto a ponto.
c) O middleware trabalha como uma camada oculta para interligar diversas máquinas e software.
d) O middleware tem a função de ligar máquinas clientes a aplicações.
e) O middleware funciona como um tradutor para recebimento de requisições dos tipos HTTP e HTTPS.

Seção 1.2

Classificações dos sistemas computacionais

Diálogo aberto

Caro estudante, nesta seção, vamos trabalhar com classificações de sistemas computacionais e entender suas categorias, avançando, assim, com o estudo de sistemas distribuídos. Você já parou para pensar como evoluímos dos primeiros computadores até os smartphones? Você já analisou os requisitos de um computador, vendo seu processador, e tentou identificar em qual categoria esse computador se enquadra? Pois bem, na situação-problema à qual será exposto, você terá a chance de trabalhar os conceitos que lhe permitirão identificar os mais diversos tipos de computadores e conhecer um pouco de sua evolução histórica. Vamos lá!

Na seção anterior, você fez uma apresentação e elaborou um relatório técnico ao dono das lojas, que gostou, principalmente porque ele já tinha ouvido falar de alguns termos que você mencionou. Agora, será importante mostrar um pouco mais de conhecimento especializado, para convencer o dono de que você sabe do que está falando, e, para isso, decide explicar como os sistemas computacionais são classificados. Selecione cinco sistemas computacionais atuais pertencentes a diferentes categorias e classifique-os conforme o conteúdo que lhe será apresentado nesta seção. Aponte suas vantagens e desvantagens, elaborando um guia para identificação e categorização dos sistemas.

Para cumprir essa missão, você aprenderá como classificar um sistema computacional, o que são e quais as diferenças entre sistemas centralizados e paralelos, bem como o que é um sistema fortemente acoplado e um sistema fracamente acoplado. Leia com atenção e aproveite a oportunidade para impressionar seu cliente!

Não pode faltar

Quando trabalhamos com sistemas computacionais, temos muitos fatores que devemos planejar antes do início do projeto para o desenvolvimento de nossos sistemas. Um dos levantamentos mais importantes a serem atingidos é a classificação do sistema computacional que será utilizado. Por meio dessa classificação, é possível ter uma noção de quanto cobrar pelo serviço, estimar o tempo que será utilizado para o desenvolvimento e escolher as melhores ferramentas de monitoração e configuração a serem utilizadas, de acordo com o tipo de sistema computacional apontado, entre outros fatores (TANENBAUM; STEEN, 2008).

Na hora de classificarmos nossos sistemas computacionais podemos dividi-los em dois grupos: sistemas centralizados e sistemas paralelos. Quando falamos de sistemas paralelos, temos duas subcategorias: os sistemas fortemente acoplados e sistemas fracamente acoplados (COULOURIS et al., 2013).

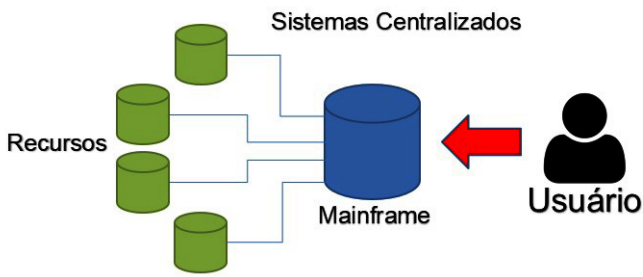
Assimile

Na hora do planejamento de um projeto, a classificação do tipo de sistema que será utilizado será fundamental para que seu projeto tenha o custo adequado e seja entregue dentro do prazo acordado para a sua criação.

Sistemas centralizados e sistemas paralelos

Quando falamos de tipos de sistemas computacionais temos que falar dos tipos **centralizados** e **paralelos**. Começaremos falando dos sistemas centralizados, em que um conjunto de máquinas utiliza seus próprios recursos e há uma máquina centralizadora de servidor. Esse tipo de sistema só tem seu funcionamento possível por meio de *mainframes*, que são computadores de grande porte que recebem uma grande quantidade de informações e as processam. Podemos observar de outra maneira, na Figura 1.5, uma ilustração do funcionamento desse tipo de sistema:

Figura 1.5 | Arquitetura centralizada



Fonte: elaborada pelo autor.

Na Figura 1.5, podemos ver o funcionamento de um sistema centralizado, em que o usuário acessa um *mainframe* e ele redireciona para os recursos necessários, assim como servidores de aplicação ou banco de dados.

Reflita

Dentre as características citadas, você acredita que é viável o uso de sistemas centralizados? Em quais situações você pode indicar esse tipo de sistema como a melhor opção?

Quando falamos de sistemas centralizados, temos entre as principais vantagens (TANENBAUM; STEEN, 2008):

- Estabilidade e robustez.
- Segurança.
- Facilidade de gerenciamento.

Destacamos, também, que todas as informações são processadas em um único servidor, o que proporciona grande facilidade para gerenciamento e uma maior segurança dos dados processados. Já entre as principais desvantagens, temos (TANENBAUM; STEEN, 2008):

- Escalabilidade e produtividade.
- Utilização de linguagens de programação antigas e falta de profissionais qualificados.
- Servidores de grande porte, com necessidade de salas especiais.
- Valor elevado para implementação e falta de interface gráfica.

Aqui, destacamos que, por as informações serem processadas em um único servidor, temos a produtividade e escalabilidade de nossos sistemas muito prejudicadas. Além disso, o investimento é alto, tanto na parte de hardware quanto na contratação de profissionais qualificados e na disponibilidade de local físico para alocação do servidor.

Exemplificando

Os sistemas centralizados são utilizados por muitos tipos de aplicações nos dias atuais. Um ótimo exemplo de utilização de sistemas centralizados são os bancos. A grande maioria utiliza sistemas computacionais baseados em *mainframes*, como centralizadores, que têm como prioridades a segurança e escalabilidade. De acordo com o tipo crítico de dados e informações sigilosas que são obrigados a processar diariamente, o tipo de sistema centralizado é a melhor opção neste caso.

Agora que vimos os sistemas centralizados, vamos entender o seu oposto, os sistemas paralelos. Esse tipo de sistema tem como objetivo executar simultaneamente várias partes de uma mesma aplicação. Essa execução simultânea pode ser feita em um processador, em uma máquina com vários processadores ou até mesmo em algumas máquinas interligadas que têm o comportamento de uma só máquina. Sistemas paralelos são sistemas que possuem mais de um processador para processar suas informações. Todos

os processamentos que devem ser feitos pelo sistema são resolvidos de forma paralela entre os processadores (TANENBAUM; STEEN, 2008).

Na maioria das vezes, os sistemas paralelos são baseados em *clusters,* conjuntos de máquinas interligadas que dividem sua carga de processamento. A maioria dos processadores encontrados atualmente no mercado tem vários núcleos, o que quer dizer que a maioria dos computadores pessoais seguem o modelo de sistemas paralelos, em que os diversos núcleos do processador dividem o processamento das informações de forma paralela. Quando falamos de sistemas paralelos, temos entre as principais vantagens (COULOURIS et al., 2013):

- Escalabilidade.
- Produtividade.
- Economia.

Destacamos aqui que todas as informações são processadas em diversas máquinas ou processadores, o que proporciona grande produtividade no processamento dos dados. Além disso, comparando com os custos de equipamentos para sistemas centralizados, os sistemas paralelos acabam sendo mais econômicos. Aplicando esse conceito em um sistema que necessite de um envio de imagem do usuário, se esse sistema trabalhar de forma paralela, um pedaço da aplicação fará o carregamento da imagem e o outro pedaço mostrará a barra de progresso do carregamento.

Entre as principais desvantagens, temos (COULOURIS et al., 2013):

- Dificuldade para gerenciamento.
- Segurança.

Comparado com o modelo centralizado, o tipo paralelo perde em segurança, pois os dados são processados em vários locais ao mesmo tempo, facilitando assim o acesso aos dados por vários servidores.

Reflita

Entre as características citadas, você acredita que é viável o uso de sistemas paralelos? Em quais situações você pode indicar esse tipo de sistemas como a melhor opção?

Sistemas paralelos fortemente acoplados

Os sistemas paralelos fortemente acoplados abrangem um conjunto bem grande de sistemas computacionais, como notebooks, smartphones

e desktops, que podem ser adquiridos em websites de varejo ou em lojas de departamentos. Duas características fundamentais diferenciam esses sistemas dos demais (COULOURIS et al, 2013):

- A comunicação entre processadores ocorre por meio de barramentos internos, que são conjuntos de sinais digitais por meio dos quais os processadores transmitem e recebem dados.
- Os processadores compartilham a mesma memória principal (conhecida popularmente como RAM – do inglês, *Random Access Memory*).

Exemplificando

Provavelmente, você possui um smartphone que utiliza no seu dia a dia, certo? Supondo que você tenha um Galaxy S8, da Samsung, de acordo com as especificações disponíveis no website do fabricante, esse modelo de aparelho possui 4 GB de RAM e um processador octa *core* (ou seja, de oito núcleos). Esses oito núcleos se comunicam entre si por barramentos internos ao equipamento e compartilham essa memória total de 4 GB. Esse é um exemplo de sistema paralelo fortemente acoplado.

Os sistemas fortemente acoplados, uma vez que também pertencem à categoria de sistemas paralelos, possuem mais do que um processador (ou núcleo), permitindo que vários programas sejam executados simultaneamente, ou seja, de maneira concorrente, ou, ainda, que um mesmo programa "pesado", por exemplo, um algoritmo de treinamento de uma rede neural profunda (*deep learning*), possa ser dividido em partes menores para que seu processamento ocorra em vários processadores ao mesmo tempo, diminuindo, assim, o tempo necessário para treinar essa rede.

Os sistemas paralelos fortemente acoplados ainda são subdivididos em sistemas de multiprocessamento assimétricos e multiprocessamento simétricos (TANENBAUM; STEEN, 2008), embora, atualmente, os sistemas de multiprocessamento simétricos sejam predominantes. Nestes últimos não existe a ideia de hierarquia entre os processadores, a fim de evitar um possível gargalo na interação entre os processadores escravos, uma vez que estes têm, eventualmente, que fazer solicitações ao processador mestre, problema que ocorria nas arquiteturas com multiprocessamento assimétrico.

Sistemas paralelos fracamente acoplados

Os sistemas paralelos fracamente acoplados abrangem os sistemas de maior escala, sempre conectados por rede de computadores (TANENBAUM; STEEN, 2008). Nessa categoria estão os sistemas que utilizam as arquiteturas

discutidas anteriormente, na primeira seção do livro. Sendo assim, todos os sistemas de rede que utilizam arquiteturas do tipo cliente-servidor, ponto a ponto ou descentralizadas pertencem a essa categoria. Tais sistemas são denominados fracamente acoplados exatamente pelo fato de que a interligação entre os elementos do sistema se dá via rede e não internamente ao hardware, o que passa a ideia de acoplamento mais flexível, ou sej, menos rígido que dos sistemas fortemente acoplados.

Reflita

Você acredita que a maioria dos sistemas utilizados atualmente pertencem a qual categoria: sistemas fortemente acoplados ou sistemas fracamente acoplados? Pense na maioria das aplicações que você realiza em seu dia a dia para responder a esta pergunta.

Os sistemas fracamente acoplados apresentam várias vantagens em relação aos sistemas fortemente acoplados, tais como (TANENBAUM; STEEN, 2008):

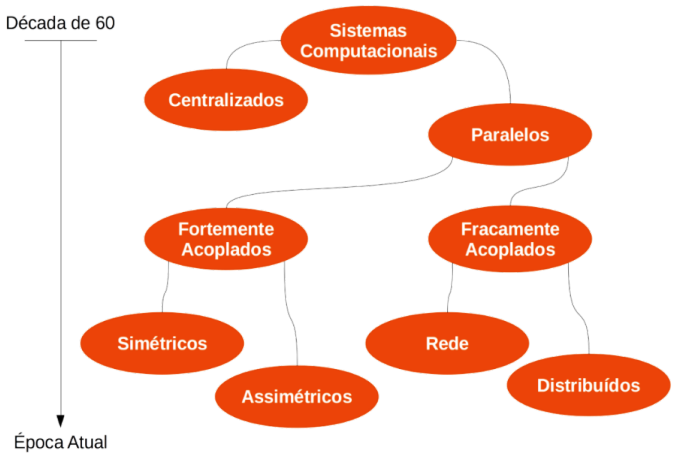
- Desempenho.
- Flexibilidade.
- Escalabilidade.

Em contrapartida, possui algumas desvantagens, entre as quais podemos citar: a velocidade na transferência de dados e a segurança, que é mais vulnerável nesses sistemas. Muitas vezes, essa questão de velocidade na transferência dos dados, que é maior em sistemas fortemente acoplados que em sistemas fracamente acoplados, pode não parecer fazer sentido, mas pense na seguinte situação: a taxa de transferência de dados (teórica) de um HD externo conectado à porta USB 3.0 do seu laptop é de 5 Gbps nominais, ao passo que a taxa de transferência de dados (teórica) em uma rede cabeada, atualmente fornecida para usuários domésticos pelos roteadores e pontos de acesso, na maioria dos fabricantes é de 1 Gbps nominais, ou seja, cinco vezes menor.

Os sistemas distribuídos também pertencem à categoria de sistemas fracamente acoplados, sendo assim, herdam as mesmas características, ou seja, utilizam arquiteturas do tipo cliente-servidor, ponto a ponto ou descentralizadas e comunicam-se via rede de computadores (seja via cabos elétricos, óticos ou comunicação sem fio). Entretanto, tais sistemas se diferenciam por possuir uma integração mais forte entre as máquinas. Nós veremos os sistemas distribuídos mais detalhadamente nas próximas unidades deste livro.

A Figura 1.6 apresenta um diagrama de classificação dos sistemas computacionais, visando facilitar a compreensão desses diferentes sistemas em uma visão mais ampla, incluindo uma referência cronológica.

Figura 1.6 | Classificação dos sistemas computacionais



Fonte: adaptada de Tanenbaum e Steen (2008).

Pesquise mais

- Você pode fixar seu conhecimento assistindo a esse vídeo, que fala de processamento paralelo:
  PROF. EDSON PEDRO FERLIN. **Processamento Paralelo:** Resumo. 2 mar. 2016.
- Uma outra dica que fala de processamento centralizado:
  ZURUBABEL. **Redes de Computadores (Aula 5) -** Modelos Computacionais: Computação Centralizada. 15 dez. 2016
- E para fixar seu conhecimento sobre sistemas distribuídos, assista a esse vídeo que fala de alguns de seus conceitos básicos:
  DICIONÁRIO DE INFORMÁTICA. **CPU ou Processador do Computador.** 5 out. 2016.

Com todo o conteúdo estudado nesta seção, você está apto para identificar e classificar os mais diversos sistemas computacionais. Você viu como é importante ter essa competência? Agora vamos continuar os estudos e seguir em frente com mais conceitos de sistemas distribuídos. Até breve!

Sem medo de errar

Para continuar impressionando o dono das lojas de varejo para quem sua consultoria está prestando um serviço, você deve criar um relatório com vantagens e desvantagens da utilização de sistemas centralizados e paralelos, ajudando-o a decidir qual é mais viável de se utilizar. Nesse mesmo relatório, você deverá colocar exemplos de computadores que pertençam a cinco categorias diferentes e mostrá-los ao proprietário. Para ajudá-lo nessa tarefa, temos as seguintes sugestões:

Pesquise na internet sobre a evolução histórica dos computadores, isso vai ajudá-lo a conhecer várias categorias de computadores e enriquecer seu relatório. Na Figura 1.7, podemos ver um exemplo de sistema centralizado, são os computadores conhecidos como *mainframes*. Esses computadores, populares no final da década de 1960 e no começo da década de 1970, são caracterizados por já utilizarem microprocessadores e por possibilitarem a multiprogramação, ou seja, execução de mais de uma tarefa, na forma de escalonamento por fatia de tempo (técnica conhecida como *spooling*). Esses computadores ainda são utilizados, geralmente em sistemas bancários. A linguagem de programação emergente na época era o Cobol, do qual você já deve ter ouvido falar.

Figura 1.7 | Sistemas centralizados

Fonte: iStock.

E nos computadores mais antigos, como no Eniac, um dos pioneiros e mais famosos sistemas centralizados, como eram programados? Pesquise como era feito um programa no Eniac, por exemplo. Uma dica: não existia uma linguagem de programação naquela época, e esses computadores eram construídos a partir

de válvulas, elementos fisicamente parecidos com uma lâmpada incandescente, e que se comportavam como uma chave liga-desliga. Agora é sua vez de relatar um exemplo real para cada um dos sistemas computacionais da árvore de classificação da Figura 1.6, apresentada no texto principal.

Você já ouviu falar em supercomputadores? Talvez seu cliente nunca tenha escutado falar sobre este assunto. Por que você não acrescenta a esse guia computadores dessa categoria e mostra alguns exemplos de utilização? Sabia que temos um supercomputador aqui no Brasil? Mais especificamente no Rio de Janeiro, e, não por acaso, foi batizado de Santos Dumont. Segundo a Prefeitura de Petrópolis (2018), esse computador é utilizado por mais de 50 projetos. Para saber mais, leia a matéria *Supercomputador Santos Dumont poderá ser usado por empresas inovadoras*, que está disponível no site da própria Prefeitura. Contudo, a lista de supercomputadores é grande, e o Santos Dumont não figura entre as primeiras posições. Por que você não pesquisa qual o supercomputador mais poderoso, atualmente? Descobrir onde ele fica, qual empresa o criou e quais tipos de pesquisas são realizadas nele pode ser recompensador!

Ainda referente aos diferentes sistemas computacionais, você pode falar um pouco sobre smartphones e como os primeiros computadores de tamanho físico excessivo evoluíram até os "computadores de mão". Para ajudá-lo, a Tabela 1.1 mostra uma comparação entre as características técnicas de um smartphone e de um notebook populares atualmente. Faça uma pesquisa a fim de completar essa tabela com as informações que estão faltando.

Tabela 1.1 | Comparativo entre um smartphone e um notebook

Característica	iPhone XS	Macbook Pro
Tamanho de tela (pol)		13,3
Peso (g)		
RAM (GB)		8
Clock do processador (GHz)	2,49	
Largura (cm)		
Profundidade (cm)		21,24
Espessura (cm)	0,77	
Sistema Operacional		
Capacidade de carga (mAh)		14.370

Fonte: elaborada pelo autor.

Avançando na prática

Qual sistema você utilizaria?

Descrição da situação-problema

Uma corretora de valores, já atuante no mercado brasileiro, decide atualizar seu sistema, cujas rotinas existentes não preveem o uso de transações financeiras baseadas em criptomoedas. Para adicionar tal funcionalidade, você, como arquiteto de sistemas, deve definir se essa instituição financeira deve continuar utilizando o sistema centralizado (atual) ou sugerir a implementação de outro tipo de sistema. Diante desse cenário, qual a melhor alternativa que você pode propor ao seu cliente? Justifique sua resposta com as vantagens desta escolha.

Resolução da situação-problema

Como a corretora já possui um sistema implementado e funcional, baseado em um sistema centralizado (*mainframe*), e deseja apenas adicionar a funcionalidade de lidar com transações de criptomoedas, a alternativa mais viável será a de manter o sistema centralizado já existente. Lembre-se de que, apesar de antigo, o sistema centralizado possui, entre suas vantagens, a questão de **segurança**: o acesso de pessoas não autorizadas a esse tipo de sistema é muito mais difícil e está menos sujeito a ataques cibernéticos que sistemas mais atuais (fracamente acoplados, no caso). Perceba, então, que nem sempre a melhor solução é migrar para tecnologias e arquiteturas mais novas. É importante que você consiga ampliar esse poder de discernimento para se tornar um profissional mais competente.

Faça valer a pena

1. Os sistemas centralizados são muito utilizados nos mais diversos tipos de aplicações. Um ponto importante é que essa categoria de sistemas tem muitas vantagens e desvantagens. Quando nossos sistemas precisam de um alto grau de segurança, esse tipo de sistema é o mais aconselhado.
Sabendo que os sistemas centralizados possuem muitas características importantes, analise as afirmações abaixo e escolha a opção correta de acordo com as características de sistemas centralizados.

I - Computadores de grande porte e processamentos centralizados em uma única máquina.
II – Computadores de grande porte e utilização de *mainframes*.
III – Computadores de pequeno porte e sistemas com alta segurança.

IV – Computadores de pequeno porte e processamentos centralizados em uma única máquina.
V – Computadores de grande porte e processamentos paralelos em várias máquinas.

a) Somente a afirmação I está correta.
b) Somente as afirmações I e II estão corretas.
c) Somente a afirmação III está incorreta.
d) Somente as afirmações III e IV estão incorretas.
e) Todas as afirmações estão corretas.

2. Os sistemas paralelos dividem o processamento das informações em vários processadores, que podem estar em máquinas diferentes ou até mesmo em um processador dividido em alguns núcleos. Aplicações baseadas nesse tipo de sistemas são muito utilizadas.

Os sistemas paralelos muitas vezes têm um conjunto de máquinas interligadas, que dividem sua carga de processamento. Identifique a tecnologia abaixo que corresponde a esse conceito.
a) *Mainframes*.
b) *Clusters*.
c) Nuvem.
d) Servidores de aplicação.
e) *Middleware*.

3. Os sistemas computacionais são divididos em várias categorias. Saber classificar sistemas é fundamental para todo o processo de desenvolvimento de uma aplicação. As categorias nos ajudam a entender o funcionamento de toda aplicação.

De acordo com o texto-base, identifique nas alternativas a categoria em que podemos enquadrar os sistemas distribuídos.
a) Sistemas fracamente acoplados.
b) Sistemas fortemente acoplados.
c) Sistemas centralizados.
d) Sistemas monoprogramáveis.
e) Sistemas acoplados simétricos.

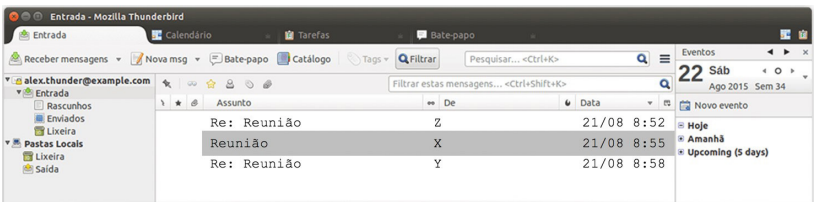
Seção 1.3

Conceitos de sistemas distribuídos

Diálogo aberto

Caro aluno, já aconteceu de você receber algum e-mail de resposta a um e-mail original, mas com a indicação de horário equivocada, ou seja, o horário de recebimento do e-mail original aparecer como posterior ao horário do e-mail de resposta? Veja um exemplo disso na Figura 1.8.

Figura 1.8 | Exemplo de problema em uma caixa de e-mails



Fonte: captura de tela do Mozilla, elaborada pelo autor.

Esse exemplo da Figura 1.8 ilustra um problema de sincronização de relógios nos sistemas computacionais que se comunicam em rede. Nesta seção, vamos trabalhar conceitos e definições de sistemas distribuídos, também vamos ver as diferenças entre *clusters* e *grids*, além de exemplos de sistemas distribuídos e de sincronização de relógios no sistema operacional Windows.

Você sabia que, para o funcionamento correto de uma aplicação distribuída, os relógios de todas máquinas devem estar sincronizados? Além disso, já ouviu falar em um servidor NTP, que serve como base para sincronizar os relógios das máquinas?

Após suas apresentações e relatórios entregues ao dono das lojas, ele percebe que você sabe do que está falando, e confessa que ele não entende como os computadores podem conversar entre si. Entre outras coisas, você fala da importância do sincronismo entre as máquinas e decide mostrar que sistemas em rede não são tão complicados assim.

Você já parou para pensar quais comandos devem ser usados para que essa sincronização funcione adequadamente? Qualquer computador pode ser usado para realizar a configuração ou é necessário um servidor específico para essa tarefa? Se for necessário um servidor, qual é o mais utilizado? Para esclarecer essas dúvidas, você decide exemplificar o funcionamento de um sistema em rede de maneira simples e prática, acessando remotamente e sincronizando o relógio local dos computadores das três lojas com a internet,

por meio do seu notebook, fazendo, assim, um acesso remoto e utilizando comandos específicos, relacionados aos servidores NTP. Para que o proprietário consiga fazer o procedimento quando a sua loja adquirir uma nova máquina, crie um relatório técnico com os comandos necessários, passo a passo, para que ele mesmo sincronize a nova máquina com as demais.

Para completar esse desafio, nesta seção, você verá em detalhes o funcionamento desse processo de sincronização de relógios, incluindo comandos específicos a serem utilizados, tanto para configuração como para constatação de que o serviço de sincronização está funcionando de maneira adequada. Ficou curioso? Espero que você se sinta motivado a dedicar seu tempo e seus esforços a um estudo que lhe proporcionará chances reais de assimilar os conceitos e práticas que você utilizará na sua vida profissional. Vamos lá!

Não pode faltar

Definição e exemplos de sistemas distribuídos

Mesmo que você não saiba, hoje mesmo você já deve ter acessado um sistema distribuído. Você deve estar pensando: como assim? Ao abrir o navegador de sua preferência e acessar uma página de internet, você está usando um sistema distribuído. Essa simples ação rotineira em nosso dia a dia, por meio de um smartphone ou computador, utiliza um sistema distribuído, mas, afinal, o que é um sistema distribuído?

Um sistema distribuído é um conjunto de computadores que são interligados via rede, mas, para o usuário final das aplicações que são executadas através deles, aparenta ser um sistema único, como uma única máquina ou um único software (TANENBAUM; STEEN, 2008). Um de seus principais aspectos é que os computadores que fazem parte de sistemas distribuídos têm o funcionamento independente, ou seja, cada um age por si próprio e, muitas vezes, os sistemas e hardwares dessas máquinas são totalmente diferentes, porém aparentam ser uma coisa só para o usuário. Esses computadores estão ligados por rede e só assim é possível seu funcionamento de forma distribuída.

A principal motivação para construir e utilizar sistemas distribuídos é proveniente do desejo de compartilhar recursos. O termo "recurso" é bastante abstrato, mas caracteriza bem o conjunto de elementos que podem ser compartilhados de maneira útil em um sistema de computadores interligados em rede. Ele abrange desde componentes de hardware, como discos e impressoras, até entidades definidas pelo software, como arquivos, bancos de dados e objetos de dados de todos os tipos, conforme indica Coulouris et al. (2013).

Assimile

Um sistema distribuído é formado por vários nós/máquinas que executam uma função dentro do sistema distribuído, de modo que, para o usuário final, aparentam ser uma única máquina.

Os sistemas distribuídos são, sem sombra de dúvida, mais utilizados em arquiteturas do tipo cliente-servidor e, como vimos anteriormente, esse tipo de arquitetura apresenta recursos compartilhados (tanto a nível de hardware quanto a nível de software) para permitir que milhares de clientes tenham acesso a esses recursos e possam utilizá-los como se houvesse uma comunicação direta entre as máquinas e o cliente.

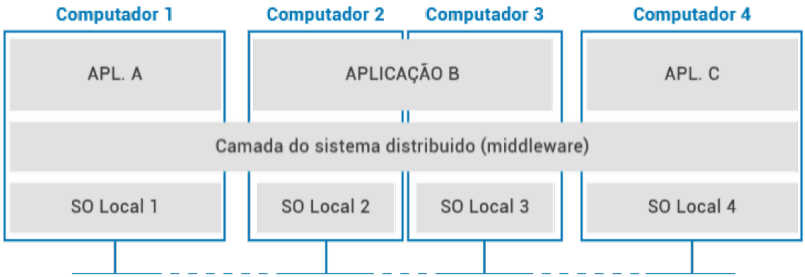
Exemplificando

A maioria dos jogos multijogadores online atuais utilizam a arquitetura cliente-servidor para seu funcionamento. Esse tipo de jogo também é um exemplo de sistema distribuído.

Imagine um jogo online de guerra em que o jogador X enfrenta o jogador Y. O jogador X na verdade tem três avatares dentro de uma sessão de jogo, pois o primeiro é aquele controlado por ele, o segundo é chamado de imagem autoritária, que é a cópia de seu avatar feita pelo servidor do jogo, a qual, ao ser enviada aos outros jogadores, funciona como o terceiro avatar. Ou seja, você movimenta o seu avatar e sua posição é enviada ao servidor que gera uma cópia com seu posicionamento e transfere-a para o jogador Y. Com isso, temos a arquitetura cliente-servidor em funcionamento.

As redes estão por toda parte e servem de base para muitos serviços cotidianos que agora consideramos naturais. Por exemplo, a Internet e a *World Wide Web* associada a ela, a pesquisa na web, os jogos on-line, os e-mails, as redes sociais, o e-Commerce, etc. (COULOURIS et al., 2013). Os sistemas distribuídos podem ser considerados como uma solução mais robusta em resposta aos sistemas puramente de rede. Nas próximas seções, estudaremos a fundo vários elementos de arquitetura e projeto de tais sistemas, incluindo seus objetivos e desafios, mas, apenas a título de conceituação, boa parte dessa robustez se dá graças a um componente conhecido como middleware que, em linhas gerais, é uma camada de software situada entre as aplicações e o sistema operacional, conforme ilustrado na Figura 1.9.

Figura 1.9 | Middleware e sistemas distribuídos



Fonte: elaborada pelo autor.

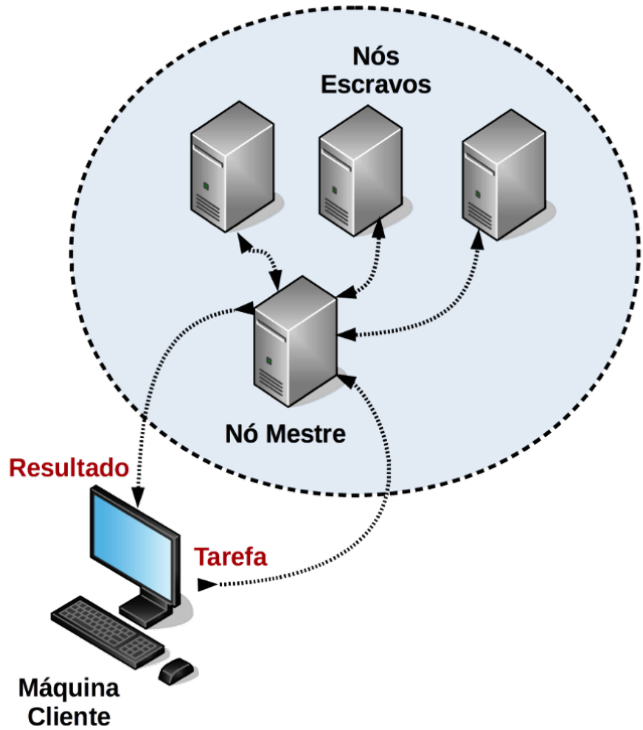
Computação em *cluster* e computação em *grid*

Os sistemas distribuídos podem ser classificados em diferentes categorias, de acordo com sua arquitetura e finalidade, sendo os mais comuns computação em *cluster* e computação em grid. Agora vamos falar um pouco sobre algumas características da computação em *cluster*. Esse tipo de computação é formado por um conjunto de máquinas com hardware semelhantes, ou seja, as máquinas que compõem o *cluster* possuem características homogêneas, de acordo com Tanenbaum e Steen (2008). O conjunto de máquinas que compõem o *cluster* são ligadas por rede local (LAN).

Quando falamos da parte de software da computação em *cluster*, temos algumas características importantes. Na maioria das vezes, o sistema operacional entre as máquinas que formam o *cluster* é equivalente. Além disso, é frequente que um único programa funcione de forma paralela, ou seja, um programa é subdividido em partes menores e cada parte é executada em uma máquina (ou nó) desse *cluster*, de forma distribuída, a fim de obter um aumento significativo de desempenho e, consequentemente, executar determinada tarefa em menos tempo. Geralmente as máquinas desse tipo de sistema são fortemente acopladas (conforme definição apresentada na Seção 1.2) em suas ligações, muitas vezes podem até compartilhar a mesma memória RAM entre várias máquinas.

Há sempre uma das máquinas que chamamos de **nó mestre**, ou seja, a máquina principal que gerencia o funcionamento da aplicação entre todos os nós. O nó mestre faz a interface com o usuário, aloca tarefas e administra a fila de tarefas. A Figura 1.10 ilustra um *cluster*, que se encontra dentro do círculo pontilhado.

Figura 1.10 | Exemplo de cluster



Fonte: elaborada pelo autor.

Computação em *grid*

Agora que já conhecemos a computação em *cluster*, vamos falar da computação em *grid*, ou grades, e apontar algumas características. Conforme Tanenbaum e Steen (2008), esse tipo de computação é formado por um conjunto de máquinas com características diferentes, podendo o hardware e os sistemas operacionais ser de fabricantes diferentes. Com isso, temos uma característica heterogênea na computação em *grid*. Essencialmente um sistema de computação em *grid* interliga vários *clusters*. Um exemplo de *grid* é o CineGrid, que trabalha no desenvolvimento de ferramentas colaborativas multimídia (CINEGRID, 2018). Na Figura 1.11, vemos uma parte desse *grid*, com as ligações entre *clusters* no Brasil; mas saiba que o CineGrid interliga outros centros de pesquisa, em várias partes do mundo.

Figura 1.11 | Exemplo de grid- CineGrid

Fonte: CineGrid (2018, [s.p.]).

Diferenças entre *clusters* e *grids*

Muitas vezes, pode parecer que *clusters* e *grids* são a mesma coisa, mas existe uma característica fundamental que difere esses dois tipos de sistemas distribuídos. Para facilitar o entendimento das diferenças entre eles, podemos pensar que *clusters* são sistemas homogêneos, ou seja, são criados para executar alguma tarefa específica que, em geral, necessita de um alto poder de processamento e, portanto, levaria muito tempo para ser executado em um computador convencional.

Exemplificando

Um exemplo de tarefa que leva muito tempo para ser executada é o treinamento de redes neurais artificiais de aprendizagem profunda (*deep learning*) para uso em ferramentas de chat on-line. Nesse tipo de aplicação, os chamados *bots* – que, nesse caso, são o resultado da rede neural treinada – conversam com e respondem questionamentos dos clientes (usuários) do serviço de chat online de uma determinada empresa. Recentemente, muitos sistemas bancários têm utilizado esse tipo de recurso. Supondo que para executar o treinamento desse tipo de rede em computador convencional seriam necessários dois dias, ao executarmos essa tarefa em um *cluster*, esse treinamento poderia ser realizado em questão de minutos.

Para diminuirmos o tempo de processamento, poderíamos executar esse tipo de tarefa (treinamento de redes neurais artificiais de aprendizagem profunda) em um computador com alto poder de processamento, porém o custo desse tipo de computador pode tornar essa opção inviável. Segundo o *Lawrence Livermore National Laboratory* (2018), o Sequoia é utilizado para realizar simulações numéricas referentes à física de armas nucleares. Assim sendo, este computador é um *cluster* homogêneo, visto que executa uma pesquisa de finalidade específica.

Pesquise mais

Segundo Barroso, Dean e Hölzle (2003), que fez uma análise da arquitetura do *cluster* utilizado pelo serviço de busca de um dos maiores fornecedores desse tipo de serviço na atualidade, a relação custo-benefício do *cluster* formado por máquinas convencionais de mercado é melhor do que um *cluster* com poucas máquinas de maior poder de processamento. Leia mais sobre essa notícia no artigo:

BARROSO, L. A.; DEAN, J.; HÖLZLE, U. Web search for a planet: The Google cluster architecture. **IEEE Micro,** v. 23, n. 2, p. 22-28, 2003.

Por sua vez, podemos pensar que os *grids* têm uma abordagem heterogênea, ou seja, são criados para executarem diferentes tarefas, de certa maneira relacionadas entre si, formando um centro de pesquisas de caráter multidisciplinar. Uma maneira ainda mais simples de entender essa característica é enxergar o grid como um conjunto de dois ou mais clusters, cada um deles responsável por um certo tipo de pesquisa.

Reflita

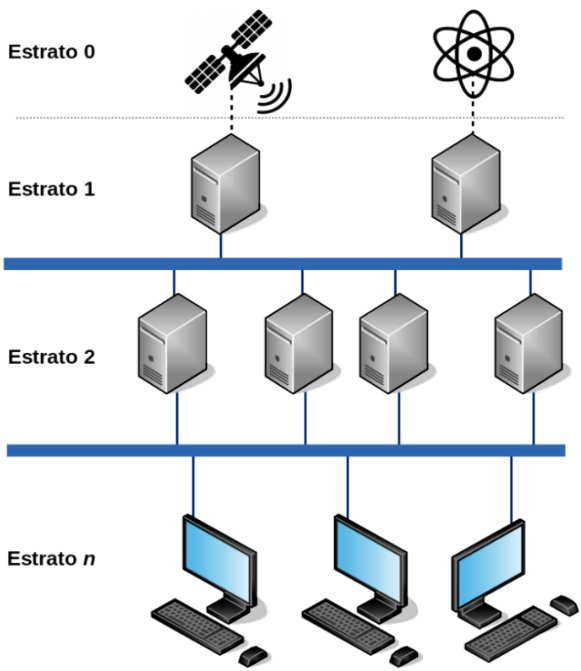
Agora que você já conhece os diferentes tipos de classificação dos sistemas computacionais, você acredita que essa classificação atual tende a mudar ao longo do tempo? Se sim, em quanto tempo? Você acha que no futuro teremos apenas sistemas fracamente acoplados?

Sincronização de Relógios

Sistemas formados por múltiplos computadores necessitam sincronizar suas ações entre si, e uma das maneiras mais utilizadas, dada sua simplicidade e popularidade, é sincronização horária, por meio do protocolo conhecido como *Network Time Protocol* (NTP) (NTP, 2018). Esse protocolo, por sua vez, utiliza o protocolo de transporte de dados *User Datagram Protocol* (UDP), operando na porta 123. Essencialmente esse protocolo é utilizado para sincronização do relógio das máquinas locais (desktops, notebooks, servidores) e demais dispositivos de rede.

A referência horária é dada por sistemas de altíssima precisão, como os relógios atômicos (NTP, 2018). Dada a precisão desses sistemas, computadores conectados a eles pertencem a uma camada de servidores chamada de estrato 1 (os sistemas de alta precisão em si pertencem a uma camada topológica chamada de estrato 0). Segundo NTP (2018), como não existem muitos servidores no mundo conectados diretamente a relógios atômicos, outros servidores são conectados aos de estrato 1, que por sua vez formam uma segunda camada de servidores de horário, chamada de estrato 2, e essa hierarquia se estende até os servidores de estrato 15, conforme podemos ver na Figura 1.12. Os computadores dos usuários são configurados para atualizar a informação horária por meio da rede, consultando servidores de estratos com valores mais altos.

Figura 1.12 | Sistema de servidores NTP



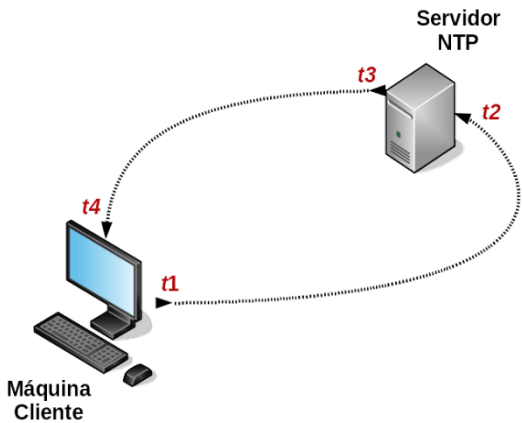
Fonte: elaborada pelo autor.

Aqui no Brasil, por exemplo, temos o Observatório Nacional (ON) que é o Órgão oficial responsável pela geração, conservação e disseminação da Hora Legal Brasileira.

Um aspecto muito interessante do protocolo NTP é que ele é projetado para verificar a latência (atraso, delay) entre a máquina cliente e a máquina servidora,

e a implementação disso é na verdade muito simples: essencialmente, de tempos em tempos, a máquina cliente faz uma consulta a um servidor NTP para verificar em quanto seu relógio está atrasado (ou adiantado) em relação ao servidor horário de referência, processando a seguinte operação: $(t_4 - t_1) - (t_3 - t_2)$, em que t_1 é a hora, minuto, segundo e milésimos de segundo (também chamada de timestamp) da máquina cliente ao enviar uma requisição (através de um pacote, no contexto de redes de computadores) para o servidor NTP; t_2 é o timestamp do servidor ao receber essa requisição; t_3 refere-se ao timestamp em que um pacote de resposta a essa requisição é enviada ao cliente; e t_4 é o timestamp em que o cliente recebe a resposta do servidor NTP. Esse cálculo, portanto, resulta em quanto o sistema operacional deverá atrasar (ou adiantar) o relógio da máquina local para que ela esteja sincronizada com a referência horária em questão, por exemplo, a Hora Legal Brasileira, em nosso caso. Um exemplo desse processo é ilustrado na Figura 1.13.

Figura 1.13 | Exemplo de sincronização através do protocolo NTP



Fonte: elaborada pelo autor.

Pesquise mais

Você pode fixar seu conhecimento assistindo a este vídeo, que fala de características, tipos e história dos *clusters*:

CANAL TI. **CLUSTER (O que é, história, tipos de Cluster).** 7 set. 2017.

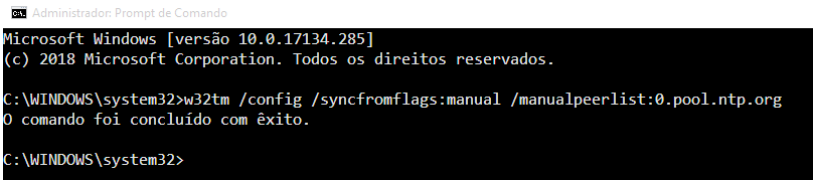
Alguns serviços, como os de acesso remoto e de autenticação de usuários, podem não funcionar adequadamente caso haja uma diferença muito grande no horário da máquina cliente (solicitante) em relação à máquina servidora (que roda e disponibiliza o serviço). Por esse motivo, é muito importante

saber como habilitar a sincronização horária das máquinas utilizando o NTP. A configuração, bem como suas etapas, varia entre os sistemas operacionais e entre suas versões. Para realizar essa configuração em um sistema operacional Windows 10, siga o procedimento abaixo:

1. Abra o Prompt de Comando (CMD), que pode ser acessado pelo menu iniciar, ou pelo campo de busca, digitando *prompt ou cmd.*

2. Na janela do CMD, vamos inserir o código a seguir e pressionar a tecla "Enter" (Figura 1.14):
   `w32tm /config /syncfromflags:manual /manualpeerlist:0.pool.ntp.org`

Figura 1.14 | Apontando para o servidor NTP

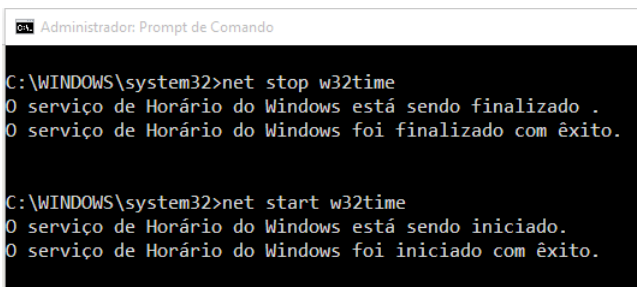


Fonte: captura de tela do Prompt de Comando do Windows 10, elaborada pelo autor.

Na Figura 1.14, podemos observar que ao final do comando aparece a mensagem "O comando foi concluído com êxito". Lembre-se: para executar os comandos, você deve estar como um usuário Administrador ou executar o CMD como Administrador. Agora que fizemos o apontamento para o servidor NTP que vamos utilizar, devemos reiniciar o serviço de data e hora para aplicar as alterações:

3. Vamos utilizar o comando `net stop w32time e net start w32time` para parar o serviço e iniciar, reiniciando-o, conforme podemos observar na Figura 1.15:

Figura 1.15 | Reiniciando o serviço de data e hora.

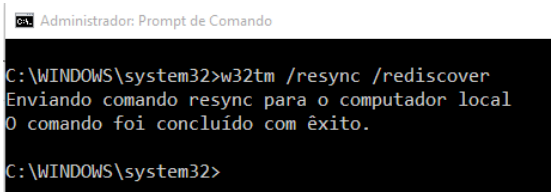


Fonte: captura de tela do Prompt de Comando do Windows 10, elaborada pelo autor.

Após isso, vamos forçar uma sincronização de data e hora para verificar se está tudo funcionando corretamente.

4. Podemos forçar uma sincronização por meio do comando `w32tm /resync /rediscover`, conforme observamos na Figura 1.16.

Figura 1.16 | Sincronizando data e hora



Fonte: captura de tela do Prompt de Comando do Windows 10, elaborada pelo autor.

Por meio da saída da Figura 1.16, podemos observar a mensagem que será dada, informando que nossa configuração foi feita com sucesso.

Sem medo de errar

Seu cliente comprou uma nova máquina para a empresa e, como você havia dito para ele que é importante que os relógios de todas as máquinas da companhia estejam sincronizados, ele lhe pediu um relatório técnico com o tutorial passo a passo de como fazer isso. Para ajudá-lo, ele informou que o sistema operacional da nova máquina é da família Windows.

A primeira informação relevante para o relatório é o sistema operacional, pois, a depender da escolha, o processo de configuração se difere. Nesse caso, você deixará registrado como configurar para o sistema Windows. A segunda informação importante no seu relatório é o conceito de servidor NTP, portanto, usando suas palavras e ilustrações, inclua esse item no documento.

Agora vem a parte mais desafiadora para quem não tem familiaridade com o mundo da tecnologia: realizar a configuração da data e hora por meio dos comandos necessários. Para que tudo funcione corretamente, é preciso que o servidor NTP esteja instalado em todos os equipamentos, inclusive no que fará a configuração. Faça a configuração da seguinte forma:

Abra o Prompt de Comando (CMD):

- Na janela do CMD vamos inserir o código a seguir e pressionar a tecla "Enter": `w32tm /config /syncfromflags:manual /manualpeerlist:0.pool.ntp.org`.

- Vamos utilizar o comando `net stop w32time` e `net start w32time` para parar o serviço e iniciar, reiniciando-o.
- Vamos forçar uma sincronização através do comando `w32tm /resync /rediscover`.

Com isso, fizemos o apontamento, reiniciamos o serviço e verificamos se a data e a hora estão sincronizadas. Que tal impressionar seu cliente e complementar seu relatório realizando uma pesquisa sobre a configuração para o sistema operacional Linux?

Avançando na prática

Sincronizando relógio em máquinas Linux

Descrição da situação-problema

Você está prestando uma consultoria para uma rede de supermercados e já configurou os computadores da rede para sincronizar os relógios com o servidor NTP (NTP, 2018). Todas as máquinas com o sistema operacional Windows estavam com seus relógios sincronizados. Contudo, o gerente lembrou que havia uma máquina Linux em que estava hospedado o site da empresa em PHP, e gostaria também de sincronizar o relógio dessa máquina como as outras. Você o informou que isso era possível e se propôs a criar uma lista de comandos que ele mesmo poderia realizar para resolver o problema.

Resolução da situação-problema

Utilizando o a versão GNU/Linux Ubuntu (18.04), devemos, por meio do Terminal, editar o arquivo de configuração NTP, utilizando o comando:

```
- sudo nano /etc/ntp.conf
```

Em seguida, vamos inserir as informações do servidor NTP.br (pool.ntp.br) no local das informações de pool que estão no arquivo, portanto, devemos acrescentar a seguinte linha:

```
- pool pool.ntp.br
```

Agora devemos reiniciar o serviço NTP:

```
- sudo service ntp restart
```

Assim, conseguimos sincronizar as informações de data e hora em uma máquina Linux.

Faça valer a pena

1. Quando trabalhamos com sistemas distribuídos, há um nó que chamamos de nó mestre. Esse nó tem uma grande importância no funcionamento de nosso sistema e pode ser considerado o nó principal de um sistema que funciona com a arquitetura distribuída.

Considerando as características do nó mestre da computação em cluster, analise as opções abaixo e escolha a que não corresponde a uma tarefa no nó mestre.
I – Interface entre servidores e usuários.
II – Controlar a fila de tarefas entre os servidores.
III – Fazer o balanceamento das aplicações entre os outros nós.
IV – Interface entre os usuários.
V – O nó mestre recebe tarefas de outros nós para execução do sistema.

a) A opção I não é uma função do nó mestre.
b) A opção II não é uma função do nó mestre.
c) A opção III não é uma função do nó mestre.
d) A opção IV não é uma função do nó mestre.
e) A opção V não é uma função do nó mestre.

2. Sistemas formados por múltiplos computadores, como os sistemas distribuídos, necessitam sincronizar suas ações entre si, e uma das maneiras mais utilizadas, dada sua simplicidade e popularidade, é a sincronização horária, a qual é necessária para o funcionamento da maioria das aplicações.

Para fazer a sincronização horária entre os computadores que foram os nossos sistemas distribuídos, utilizamos um protocolo muito popular. Identifique a alternativa que corresponde ao protocolo utilizado para sincronização horária.
a) *Transmission Control Protocol* – TCP.
b) *Network Time Protocol* – NTP.
c) *User Datagram Protocol* – UDP.
d) *HyperText Transfer Protocol* – HTTP.
e) *SSH Remote Protocol* – SSH.

3. Os sistemas distribuídos estão presentes em quase todas as aplicações que utilizamos atualmente. Além disso, podemos apontar vários tipos de sistemas distribuídos, como computação em cluster e em grid, que são muito populares quando falamos em sistemas distribuídos.

Considerando as características da computação em cluster e grid, analise as afirmações abaixo e escolha a opção correta.
I – As máquinas que compõem um sistema com computação em grid têm características homogêneas.

II - As máquinas que compõem um sistema com computação em cluster têm características heterogêneas.
III – Entre a computação em cluster e grid, não conseguimos apontar diferenças.
IV – Geralmente o sistema operacional e o hardware das máquinas que compõem um sistema em computação em cluster são equivalentes.
V – Geralmente o sistema operacional e o hardware das máquinas que compõem um sistema em computação em cluster são de fabricantes e configurações diferentes.

a) Somente a afirmação I está correta.
b) Somente as afirmações II e IV estão corretas.
c) Somente a afirmação V está correta.
d) Somente a afirmação IV está correta.
e) Somente as afirmações I e II estão corretas.

Referências

CINEGRID. **CineGrid Brasil**. [S.l., s.d.]. Disponível em: https://cinegridbr.org/. Acesso em: 20 set. 2018.

COULOURIS, G. et al. **Sistemas Distribuídos** - Conceitos e Projeto. 5. ed. Porto Alegre: Editora Bookman, 2013.

LAWRENCE LIVERMORE NATIONAL LABORATORY. **Sequoia.** [S.l.; s.d.]. Disponível em: https://computation.llnl.gov/computers/sequoia/. Acesso em: 12 out. 2018.

MAIA, L. P. **Arquitetura de redes de computadores.** Rio de Janeiro: LTC, 2013.

NTP. **Network Time Protocol project.** [S.l.], 2018. Disponível em: http://www.ntp.org/. Acesso em: 18 out. 2018.

PREFEITURA DE PETRÓPOLIS. **Supercomputador Santos Dumont poderá ser usado por empresas inovadoras**. Petrópolis, 30 jan. 2018. Disponível em: http://www.petropolis.rj.gov.br/pmp/index.php/imprensa/noticias/item/8423-supercomputador-santos-dumont-poder%C3%A1-ser-usado-por-empresas-inovadoras.html. Acesso em: 24 out. 2018.

TANENBAUM, A. S; STEEN, M. V. **Sistemas Distribuídos** - Princípios e Paradigmas. 2 ed. São Paulo: Pearson, 2008.

Unidade 2

Objetivos, desafios e modelos de sistemas distribuídos

Convite ao estudo

Olá, caro aluno. Seja bem-vindo a mais uma unidade!

Agora que já conhecemos alguns conceitos de sistemas distribuídos, vamos aprofundar ainda mais nossos conhecimentos. Você já parou para pensar quais objetivos queremos atingir quando implantamos um sistema distribuído? Nesta unidade, vamos entender os principais objetivos a serem alcançados em uma implementação. Você sabia que, além de conhecer os principais objetivos de um sistema distribuído, também temos uma série de aspectos que são de extrema importância para projetarmos um sistema distribuído de maneira adequada? Por exemplo: segurança, escalabilidade, granularidade, heterogeneidade, tolerância às falhas, entre outros. Nesta unidade, vamos aprender a lidar com esses e outros aspectos de projetos.

Além disso, como você já está familiarizado com o modelo de arquitetura de computadores cliente-servidor, chegou a hora de detalharmos qual a função das máquinas cliente, servidor e *workstation*, no contexto de sistemas distribuídos.

Na primeira seção você entenderá quais são os objetivos de um sistema distribuído. Já na segunda, você estudará alguns aspectos pertinentes a esse tipo de sistema e, por fim, você aprenderá como o modelo cliente-servidor é aplicado em um sistema distribuído.

Você consegue apontar para um determinado cenário e verificar se é viável a implementação de um sistema distribuído? Após completar esta unidade, você será capaz de realizar essa tarefa, utilizando seu raciocínio crítico e criatividade para resolução dos problemas propostos.

Sua empresa está desenvolvendo um sistema de grande porte para um dos seus clientes mais importantes e você é uma peça fundamental nesse projeto por ser o único desenvolvedor da equipe que possui conhecimento e experiência na implantação de sistemas distribuídos. Será que você é capaz de guiar a sua equipe em relação a todos os aspectos de projeto necessários para a implementação desse sistema, de maneira robusta e funcional?

Agora, chegou a hora de prosseguir com os estudos dos Sistemas Distribuídos, o que irá lhe proporcionar conhecimentos e oportunidades no mercado de trabalho. Importante frisar que quanto mais você se dedicar, mais poderá aproveitar os ensinamentos transmitidos neste material.

Seção 2.1

Objetivos dos Sistemas Distribuídos

Diálogo aberto

Caro estudante,

Iniciamos agora mais uma seção no estudo de sistemas distribuídos. Você já ouviu falar de um comunicador de mensagens instantâneas chamado *Discord*? Esse aplicativo está disponível para plataformas *desktop* e *mobile*, e utiliza um sistema distribuído em sua arquitetura, mas, afinal, quais objetivos foram relevantes para a implantação dessa arquitetura distribuída? Nesta seção, vamos trabalhar os principais objetivos que tornam viável a implementação de um sistema distribuído. Ao comprar um novo computador, você já teve que decidir qual sistema operacional seria mais viável para o tipo de tarefas que você irá executar? Pois bem, na situação problema à qual será exposto, você terá que escolher e apontar os objetivos que o levaram a escolha de um sistema distribuído para implementação do projeto. Vamos lá!

Em sua nova empresa, você é um arquiteto de sistemas que está prestes a fornecer um sistema de controle de manutenção preventiva para várias frotas de veículos de grandes transportadoras no estado de Minas Gerais. O produto que sua empresa está desenvolvendo parece de fato promissor, entretanto, à exceção de você, o time de desenvolvimento é formado por desenvolvedores que nunca haviam trabalhado com sistemas distribuídos, e você deve orientá-los para que todos do time entendam a importância e os pontos de atenção no desenvolvimento desse tipo de sistema, certificando-se, assim, de que o produto final seja robusto e funcional.

Um estudo de caso parece ser o ponto de partida ideal para explicar ao seu time o que um sistema distribuído real tenta solucionar. Para o sistema de controle de manutenção preventiva que sua empresa deve implementar, identifique quais objetivos provavelmente devem alcançados neste sistema e apresente a modelagem do mesmo utilizando os seguintes diagramas UML: caso de uso, sequência e de implantação conforme com o contexto.

Agora utilize o seu conhecimento atrelado ao conteúdo da Seção para solucionar a situação problema apresentada acima. Tenho certeza de que você será capaz de guiar seu time e desenvolver um projeto de sucesso atingindo todos os objetivos esperados tanto pela sua empresa, quanto pelo seu cliente.

Não pode faltar

Vamos começar retomando o conceito de sistemas distribuídos tendo como base a definição de Tanenbaum e Steen (2008, p. 1): "um sistema distribuído é um conjunto de computadores independentes que se apresenta a seus usuários como um sistema único e coerente". Ainda segundo Tanenbaum e Steen (2008), os sistemas distribuídos têm três objetivos principais: compartilhamento de recursos, confiabilidade e desempenho.

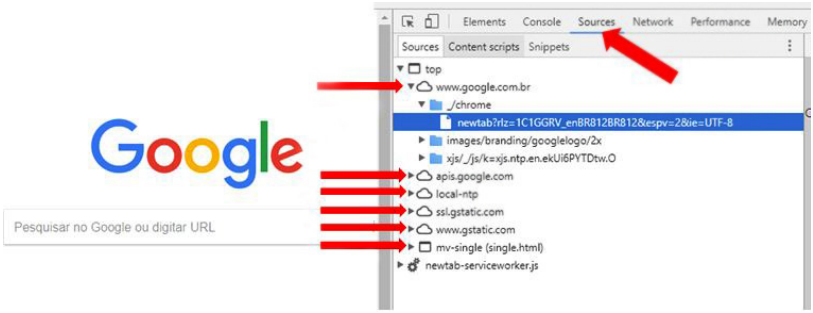
Compartilhamento de recursos

O compartilhamento de recursos refere-se à capacidade do sistema em compartilhar o acesso a quaisquer recursos utilizados pelos sistemas entre as máquinas que fazem parte da arquitetura (também chamadas de nós). Esses recursos são, na maioria das vezes, bancos de dados, links de rede que se conectam à Internet, serviços de autenticação, entre outros. Apesar de não ser um objetivo exclusivo dos sistemas distribuídos – uma vez que também é um objetivo dos sistemas de rede –, é uma característica muito importante de um sistema distribuído. A vantagem em compartilhar recursos está na economia financeira, uma vez que, caso não haja tal possibilidade de compartilhamento, mais réplicas de um determinado recurso devem estar presentes em cada nó do sistema, o que impacta direta e indiretamente no custo. Entretanto, como aspecto negativo associado a esse compartilhamento de recursos, temos a questão da segurança; uma vez que o fato de mais máquinas terem acesso ao recurso implica que o sistema possui mais pontos de acesso, e esses pontos de acesso podem ser explorados por hackers, tanto no sentido de rastreamento da comunicação quanto na própria questão de invasão de privacidade e integridade dos dados (COULOURIS, 2013).

Exemplificando

O conjunto de ferramentas para desenvolvedores disponível nos navegadores de internet constitui um leque de opções muito úteis para diversos profissionais de TI que lidam com ambiente de rede e Internet. Para acessá-lo, você pode usar o menu de opções do seu navegador ou, no caso do Chrome usar o atalho *Ctrl + Shift + i*. Veja na Figura 2.1 uma captura de tela da ferramenta Sources, disponível nesse conjunto. Ela permite visualizar as fontes que estão alimentando o site. A Figura 2.1 ilustra os servidores que estão sendo acessados ao simplesmente abrir uma página do Google. Abra alguns portais de notícias e vejam a grande quantidade e diversidade de fontes que são acessadas, um rico exemplo de sistema distribuído.

Figura 2.1 | Ferramenta *Sources* do navegador



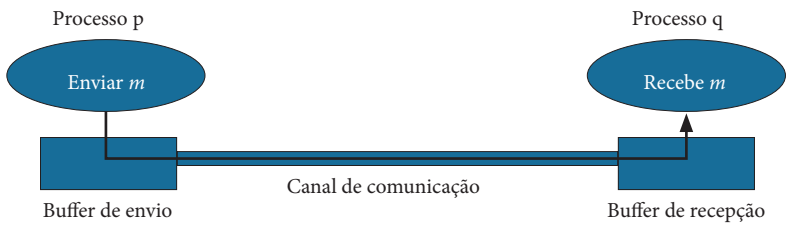
Fonte: captura de tela do Google Chrome, elaborada pelo autor.

Confiabilidade

A análise morfológica da palavra confiabilidade nos mostra que esta se refere à probabilidade de um produto executar a sua função prevista de forma que atenda ou exceda as expectativas, ou seja, está relacionada ao funcionamento adequado, conforme foi planejado. Podemos confundir a confiabilidade acreditando ser algo relacionado à segurança do sistema, porém, não tem relação alguma com a parte de segurança do sistema, conforme observam Colouris et al. (2013).

A confiabilidade nos sistemas distribuídos é maior que nos sistemas centralizados. Porém, qualquer problema relacionado a processos ou canal de comunicação/transmissão pode surtir efeitos diretos sobre a execução do sistema. Podemos observar, na Figura 2.2, como ocorre a comunicação entre processos nos sistemas distribuídos em que são aplicados os conceitos de confiabilidade:

Figura 2.2 | Comunicação entre processos em um sistema distribuído.

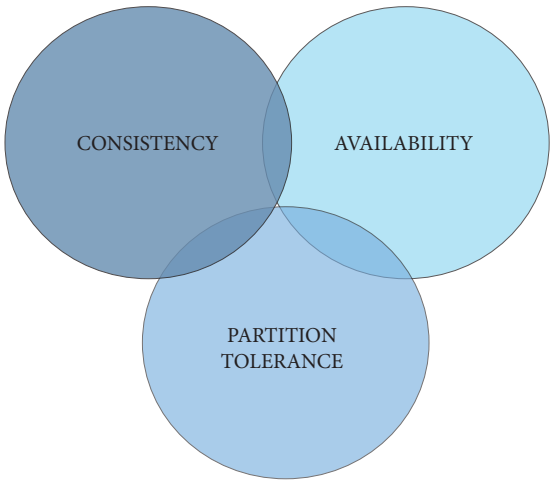


Fonte: elaborada pelo Autor.

Podemos observar na Figura 2.2 que, se ocorrer algum problema no canal de comunicação, isso será refletido em todo processo de comunicação e funcionamento do sistema. A confiabilidade de um sistema é baseada em três pilares:

consistência, disponibilidade e resiliência (tolerância a falhas), conforme o teorema CAP. O teorema CAP é baseado nesses pilares e sua sigla vem das palavras "*Consistency*", "*Availability*" e "*Partition Tolerance*" (GREINER, 2014). Podemos observar, na Figura 2.3, a representação desse teorema.

Figura 2.3 | Representação do Teorema CAP.



Fonte: adaptada de Reiner (2014, [s.p.]).

Uma das características importantes do teorema CAP, que pode ser observada na representação da Figura 2.3, é que nunca podemos atingir os três pilares em sua totalidade dentro de um sistema. A forma como foi elaborado permite que você tenha apenas dois dos pilares em evidência em seu sistema. Ou seja, caso selecione dois pilares em seu sistema, o terceiro ficará enfraquecido.

Assimile

O teorema CAP consiste em três pilares: consistência, disponibilidade e resiliência (tolerância a falhas). Seu sistema só poderá ter foco em dois deles, e o terceiro será enfraquecido. Por exemplo: Se optar por consistência e disponibilidade, seu sistema terá menor resiliência. E assim ocorre em todas as suas combinações e possibilidades.

Desempenho

Aumentar o desempenho de um sistema também é um objetivo de sistemas distribuídos. Se fizermos uma comparação, os sistemas distribuídos, na maioria dos casos, apresentam melhor desempenho do que os sistemas

centralizados. Isto ocorre porque, em um sistema distribuído, temos múltiplas instancias, tanto de hardware quanto de software, para realizar o processamento necessário. Como podemos medir esse desempenho? Conforme Colouris (2013, p. 83), podemos utilizar como parâmetros:

- Tempo de resposta do servidor;
- *Throughput* (taxa de transferência);
- Quantidade de recursos consumidos pela rede;
- Resultados de *benchmarks* (execução do sistema);
- Tempo de transmissão dos dados;

Verificando os resultados das ações acima, podemos medir o desempenho de nosso sistema distribuído e descobrir se é satisfatório.

Reflita

Agora que você já conhece alguns parâmetros relacionados ao desempenho dos sistemas distribuídos, ou seja, já tem uma forma de mensurar essa grandeza, o que você pode fazer com essa informação? Quais seriam possíveis valores para os parâmetros exemplificados de um sistema real? Será que você é capaz de mensurar alguns deles? Como você faria para mensurá-los?

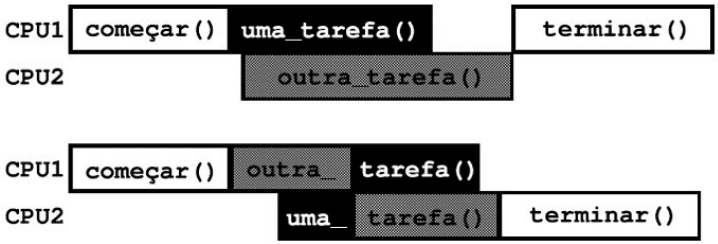
Granularidade

Quando falamos de granularidade, estamos falando da capacidade de um sistema ser executado por vários processadores (CPUs) diferentes. Para funcionar de forma correta, o sistema também deve ser desenvolvido por meio de métodos ou funções que estejam disponíveis para execução de forma paralela.

Estamos falando de granularidade fina quando o tempo de execução de uma tarefa compensa os custos de criação, comunicação e sincronização (TANENBAUM; STEEN, 2008). Falamos de granularidade grossa, quando o tempo de ocupação dos processadores disponíveis for o máximo possível na execução do sistema.

Na Figura 2.4, podemos observar um exemplo de como a granularidade afeta a execução de um sistema qualquer.

Figura 2.4 | Granularidade



Fonte: elaborada pelo autor.

Podemos observar, na Figura 2.4, que as CPUs podem executar métodos em paralelo para finalizar a execução do sistema. Portanto, para aplicar o conceito de granularidade o nosso programa, deve ser desenvolvido de forma organizada, com métodos para que sua execução possa ser dividida entre vários recursos (processadores, neste caso) diferentes, de forma a ter um melhor aproveitamento dos mesmos.

Desafios e obstáculos dos sistemas distribuídos

Os sistemas distribuídos normalmente possuem as seguintes metas, segundo Tanenbaum e Steen (2008):

- Abertura;
- Concorrência;
- Escalabilidade;
- Heterogeneidade;
- Segurança;
- Tolerância a falhas;
- Transparência.

Entretanto, devem ser analisados mais como desafios a serem atingidos, uma vez que nem sempre o sistema distribuído conseguirá atingir todas essas metas de maneira integral. A seguir, descreveremos a que se refere cada uma dessas metas, na visão do renomado autor Andrew Tanenbaum (TANEMBAUM; STEEN, 2008), embora os itens Segurança, Escalabilidade, Tolerância a Falhas e Heterogeneidade mereçam uma atenção especial, razão pela qual detalharemos eles ainda mais nas próximas seções deste material.

Abertura, no contexto de sistemas distribuídos, refere-se a quanto o sistema é modularizado ou, em outras palavras, quanto é fácil integrar e alterar

tecnologias e *frameworks* sem que o sistema seja comprometido. Você já deve ter ouvido falar em micros serviços, não é verdade? Pois bem, esse nome é a maneira atual ("hype") de dizer que o sistema possui uma grande abertura. Importante observar que, independentemente do que o termo "abertura" possa passar, é uma coisa positiva em sistemas distribuídos. É importante ter em mente esse conceito, pois, à primeira vista, um sistema mais "aberto" parece ser uma coisa negativa, no sentido de estar mais vulnerável a falhas e, na verdade, em sistemas distribuídos, isso não é verdade. Segurança não tem relação nenhuma com essa abertura, mas isso é um assunto a ser detalhado em uma próxima oportunidade. Por exemplo, se o sistema utiliza tecnologias não-proprietárias, dizemos que a abertura desse tal sistema é maior.

Concorrência refere-se à capacidade de o sistema poder ser acessado e utilizado de maneira simultânea, concorrente (ao mesmo tempo), por vários usuários. Aqui cabe novamente a ressalva de que, no contexto de sistemas distribuídos, o termo concorrência não tem uma conotação negativa (como ocorre, por exemplo, no comércio); apenas refere-se a um sistema que dá suporte a acessos simultâneos. Por exemplo, várias pessoas podem acessar um website de comércio eletrônico, certo? Pois bem, isso é um acesso concorrente.

Escalabilidade refere-se à capacidade de o sistema poder ser ampliado para suportar, por exemplo, maior quantidade de acessos simultâneos ou realizar uma tarefa mais rapidamente (ao ampliarmos a capacidade de processamento desse sistema). Perceba como algumas metas podem estar inter-relacionadas nos sistemas distribuídos, como é o caso da escalabilidade e da concorrência. Por exemplo, quando uma empresa de *games* necessita aumentar a quantidade de servidores de um determinado jogo online para possibilitar uma maior quantidade de jogadores em uma partida multiplayer, eles estão escalando o sistema (nesse exemplo, aumentando a escala do mesmo).

Heterogeneidade significa que o sistema distribuído é capaz de funcionar em uma arquitetura heterogênea, tanto em termos de hardware quanto em termos de software. Em termos de hardware, significa que o sistema distribuído consegue operar em nós com características de hardware diferentes, por exemplo, entre máquinas com diferentes valores de memória principal (RAM), memória de armazenamento (HD, SSD, etc.) e capacidade de processamento (*clock* dos processadores). Em termos de software, significa que o sistema distribuído suporta, por exemplo, diferentes sistemas operacionais, com alguns nós utilizando MS-Windows, enquanto outros utilizam alguma distribuição GNU/Linux. Isso pode parecer difícil de ser alcançado, mas existe um elemento que facilita nosso trabalho, o *middleware*, que, como o nome sugere, é uma camada de software que fica situada entre

a sua aplicação e o sistema operacional. Não se preocupe se esse termo e seu papel não estiverem claros ainda, pois detalharemos esse assunto em uma aula posterior. Por exemplo, quando um *website* roda em máquinas com diferentes sistemas operacionais ou características de hardware (por exemplo, uma máquina do tipo servidor que possui mais memória RAM que outra máquina dentro do mesmo sistema distribuído), dizemos que se trata de um sistema distribuído heterogêneo.

Segurança refere-se à violação da informação (dado), bem como a sua integridade, ou seja, é o mesmo conceito geral de segurança em sistemas informatizados de TI. Como é um assunto muito importante, também separamos uma seção especial para tratar desse assunto em uma próxima aula.

Tolerância a falhas refere-se à capacidade do sistema distribuído se auto recuperar na ocorrência de uma (ou mais) falhas. Pode parecer desanimador, mas acredite: falhas em um sistema computacional sempre ocorrem, em vários momentos, ao longo do uso de um sistema, por uma série de razões que serão detalhadas mais adiante. Aproveitando, você sabia que, no nosso idioma português existe uma palavra para expressar essa ideia de "capacidade do sistema distribuído se auto recuperar na ocorrência de uma (ou mais) falhas"? Isso mesmo, apenas uma palavra: resiliência. Então da próxima vez que quiser passar essa ideia de maneira mais concisa, utilize a palavra resiliência.

Por fim, **Transparência** refere-se, novamente, do ponto de vista do usuário, a quão transparente o sistema é, ou seja, quanto o cliente "desconhece" do funcionamento interno do sistema, e isso é uma característica positiva: quanto menos o usuário necessitar saber da implementação e funcionamento do sistema ou seja, quanto mais transparente o sistema for, melhor para o usuário.

No Quadro 2.1, podemos observar os Tipos de Transparência, segundo Tanenbaum e Steen (2008):

Quadro 2.1 | Tipos de transparência

Tipo	Descrição
Acesso	Oculta diferenças na representação de dados e no modo de acesso.
Localização	O local do recurso é desconhecido.
Migração e Relocação	O recurso pode ser movido, inclusive enquanto está em uso.
Replicação	Múltiplas instâncias de um recurso podem ser utilizadas para aumentar a confiabilidade e desempenho.
Concorrência	Vários usuários podem compartilhar o mesmo recurso.
Falha	A falha e a recuperação de um recurso não afetam o sistema.

Fonte: elaborado pelo autor.

Continue firme em seus estudos, que ainda vem muita coisa interessante e útil pela frente!

Pesquise mais

Você pode fixar seu conhecimento assistindo a este vídeo, que fala de alguns conceitos básicos de sistemas distribuídos, assim como os objetivos estudados nesta seção.

INSTITUTO DE GESTÃO E TECNOLOGIA DA INFORMAÇÃO. **Princípios de Sistemas Distribuídos**. 15 nov. 2017.

Assim finalizamos mais uma seção no estudo dos sistemas distribuídos. Espero que o conhecimento adquirido seja de importância para seu crescimento profissional. Bons estudos e até a próxima seção!

Sem medo de errar

Você é o analista de confiança de sua empresa para liderar o projeto de implantação de um sistema de controle de manutenção preventiva para várias frotas de veículos de grandes transportadoras no Estado de Minas Gerais. O sucesso desse projeto influencia diretamente no crescimento de sua empresa e da confiança dos diretores em seu trabalho. Como você é o único analista da equipe com competência para trabalhar com sistemas distribuídos, terá uma função essencial nesse projeto. Vamos colocar a mão na massa e iniciá-lo?

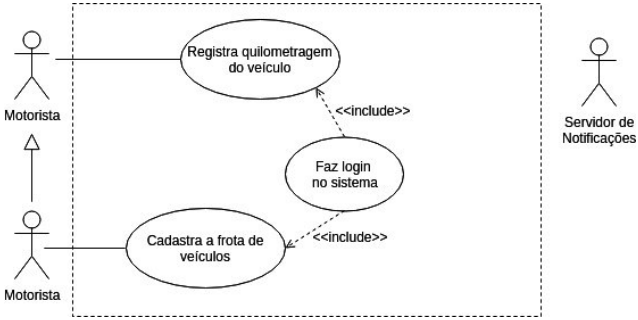
Para elaborar os diagramas UML (caso de uso, de sequência e de implantação) solicitados e apresentá-los a sua equipe, utilize os conteúdos apresentados nesta seção.

Para te ajudar nessa tarefa, temos as seguintes sugestões:

- Você pode elaborar os diagramas UML utilizando a ferramenta online Draw IO.
- Você não se recorda como elaborar diagramas UML de Casos de Uso? Pode fazer uma revisão assistindo ao vídeo *Entenda o Diagrama de Casos de Uso | #7*.

A Figura 2.5 apresenta um possível, entretanto incompleto, Diagrama de Casos de Uso para sistema de controle de manutenção preventiva da frota de veículos.

Figura 2.5 | Exemplo de Diagrama de Casos de Uso

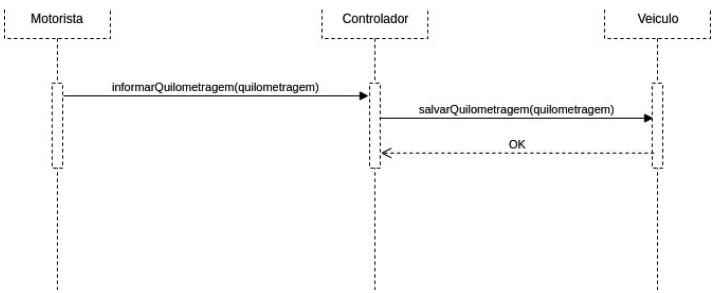


Fonte: elaborada pelo autor.

O Motorista faz o registro da quilometragem atual do veículo que foi disponibilizado para ele. O veículo deve ser cadastrado no sistema somente pelo Gerente da transportadora, que também pode registar a quilometragem do veículo, por isso está fazendo o uso de herança do ator "Motorista". Ambos atores necessitam, obrigatoriamente, realizar o login antes de executar qualquer iteração no sistema, razão da relação entre os casos de uso por meio da palavra *include*. E o ator "Servidor de Notificações"? Bem, ele deve enviar uma notificação para o motorista quando estiver próximo da data limite de manutenção do veículo. Tente completar o diagrama a partir desse exemplo e fique à vontade para incluir outros casos de uso que julgar relevantes ao sistema proposto na situação problema desta seção. Você não se recorda como elaborar diagramas UML de Sequência? Pode fazer uma revisão assistindo ao vídeo *Entenda o Diagrama de Sequência | #10*.

A Figura 2.6 apresenta um possível, entretanto incompleto, Diagrama de Sequência para sistema de controle de manutenção preventiva da frota de veículos.

Figura 2.6 | Exemplo de Diagrama de Sequência

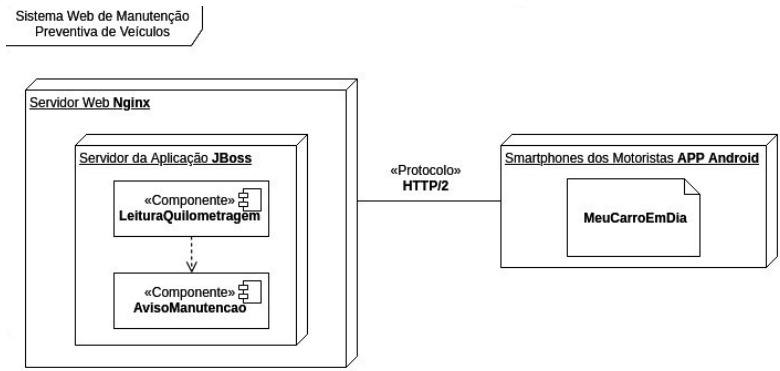


Fonte: elaborada pelo Autor.

O objeto "Motorista" utiliza o método informarQuilometragem, passando como parâmetro um atributo quilometragem para um objeto "Controlador", que deve salvar essa informação em um objeto "Veículo", que será persistido em um bando de dados. O método salvarQuilometragem possui um retorno do tipo String para informar o sucesso ou falha no salvamento do atributo quilometragem. Com certeza, o sistema necessitará de outros objetos e métodos para funcionar na sua plenitude. Você consegue criar outros diagramas de sequência para refletir isso? Tente completar o diagrama a partir desse exemplo. Você não se recorda como elaborar diagramas UML de Implantação? Pode fazer uma revisão assistindo ao vídeo *Entendendo o Diagrama de Implantação | #12.*

A Figura 2.7 apresenta um possível, entretanto incompleto, Diagrama de Implantação para sistema de controle de manutenção preventiva da frota de veículos.

Figura 2.7 | Exemplo de Diagrama de Implantação



Fonte: elaborada pelo Autor.

O artefato Servidor Web utilizará o servidor de *open-source* Nginx, por ter um desempenho reconhecidamente melhor que o servidor Apache. A regra de negócio (lógica) será implementada em Java, e teremos, essencialmente, dois componentes: um referente à leitura de quilometragem e outro referente ao aviso de manutenção (notificação). O cliente será um smartphone com sistema operacional Android, que terá instalado o aplicativo "Meu Carro em Dia", que fará a comunicação com o serviço *back-end* (*server-side*) do seu sistema de controle de manutenção preventiva da frota de veículos através da Internet, utilizando o protocolo HTTP, versão 2, para realizar essa comunicação.

Quais outros artefatos e componentes você acredita que devem ser adicionados a esse diagrama de Implantação? Tente completar o diagrama a partir desse exemplo. Você pode pesquisar na internet os principais fatores que levam a escolha de uma arquitetura distribuída em um projeto e utilizar os argumentos para defender sua adoção neste projeto em particular.

Avançando na prática

Aplicando a granularidade

Descrição da situação-problema

A empresa em que você trabalha criou um algoritmo, ainda em pseudocódigo, para calcular o pagamento referente às horas acumuladas em banco de horas de seus funcionários, sendo que essa funcionalidade não havia sido implantada no ERP atual da empresa. Portanto, da maneira que foi implementado esse novo módulo, não foi aplicado o conceito de granularidade, e, assim, o atual software ERP que segue o modelo de arquitetura distribuída não estaria apto a recebê-lo. Você poderia adaptar o pseudocódigo abaixo adequando o programa para o aumento de granularidade, de forma a ter um melhor aproveitamento dos recursos de processamento (CPU) das máquinas?

Pseudocódigo:

```
Programa BancoDeHoras
var horas,codigo_funcionario: inteiro
var valor,total: real
Inicio
leia codigo_funcionario
leia horas
leia valor
total = valor * horas
escreva "O funcionário de código " + codigo_funcionario + " , possui o total de " + total + " , a receber, referente a seu banco de horas."
fim
```

Resolução da situação-problema

Para adaptar o pseudocódigo do programa "BancoDeHoras" de forma a ter maior granularidade, devemos separar as ações em procedimentos/funções, conforme abaixo:

```
Programa BancoDeHoras
var horas,codigo_funcionario: inteiro
var valor,total: real
Inicio

procedimento ler_dados( )
leia codigo_funcionario
leia horas
leia valor

função calcular_valor(valor,horas): real
total = valor * horas
retorna total

procedimentoexibi_resultado(codigo_funcionario,total)
escreva "O funcionário de código " + codigo_funcio-
nario + " , possui o total de " + total + " , a
receber, referente a seu banco de horas."

fim
```

Faça valer a pena

1. A implementação de sistemas distribuídos ocorre em muitos projetos atuais e populares. A escolha da arquitetura distribuída acaba sendo muito frequente, pois a maioria das aplicações tem os mesmos requisitos de funcionamento que são representados nos principais objetivos de uma arquitetura distribuída.

Sobre os principais objetivos de um sistema distribuído, marque V para verdadeiro ou F para falso:
( ) Um dos principais objetivos de um sistema distribuído é de que os recursos estejam compartilhados entre os nós do sistema.
( ) Espera-se que um sistema distribuído tenha maior robustez e melhor desempenho que os sistemas puramente de rede.
( ) Uma das características de um sistema distribuído é que sua segurança é maior do que os sistemas puramente de redes.
( ) Os três componentes principais que definem a confiabilidade de um sistema distribuído são: consistência, disponibilidade e resiliência.

Assinale a alternativa que representa a sequência CORRETA:
a) V - V - V - F.
b) V - F - V - F.
c) V - V - F - V.
d) V - V - V - V.
e) F - V - V - F.

2. Um aspecto importante ao estudar sobre sistemas distribuídos é a compreensão do teorema CAP, que define uma relação entre os aspectos de consistência, disponibilidade e resiliência, derivados dos termos em inglês *consistency, availability* e *partition-tolerance*.

Sabendo que o teorema CAP define uma associação entre elementos, analise as afirmações abaixo e escolha a opção correta de acordo com o relacionamento adequado.
I – Quanto maior a disponibilidade e a consistência, menor será a resiliência.
II – Quanto maior a resiliência e a disponibilidade, maior será consistência.
III – Quanto maior a resiliência e a disponibilidade, menor será a consistência.
IV – Quanto maior a resiliência e a consistência, menor será a tolerância a falhas.
V – Quanto maior a resiliência e a consistência, menor será a disponibilidade.

a) Somente a afirmação V está correta.
b) Somente as afirmações I e III estão corretas.
c) Somente a afirmação III está correta.
d) Somente as afirmações I, III e V estão corretas.
e) Todas as afirmações estão corretas.

3. A implantação de um sistema distribuído enfrenta muitos desafios, são eles: Abertura, Segurança, Transparência, Escalabilidade, Tolerância a falhas, Concorrência e Heterogeneidade. Cada um desses desafios apresenta obstáculos com várias características pelas quais podemos identificá-los.

Custos, Perda de desempenho, Prevenção à falta de recursos e Gargalos no sistema são características de um dos obstáculos enfrentados pelos sistemas distribuídos. Identifique a alternativa que representa o obstáculo que mais se enquadra nessas características.

a) Segurança.
b) Transparência.
c) Escalabilidade.
d) Concorrência.
e) Heterogeneidade.

Seção 2.2

Aspectos de projeto dos Sistemas Distribuídos

Diálogo aberto

Caro estudante,

Nesta seção, vamos explorar com maior detalhamento alguns dos principais aspectos de projeto em sistemas distribuídos, nomeadamente: a segurança, escalabilidade, resiliência e heterogeneidade, que são diferenciais em relação aos sistemas puramente de rede – mas que não sejam sistemas distribuídos.

Você já pensou em como os serviços de *streaming* de vídeo mais populares da atualidade conseguem suportar uma quantidade enorme de acessos simultâneos? Ainda, quando não há tanta demanda assim, será que empresas de grande porte mantém ligados centenas de servidores mesmo que não estejam em uso? Já pensou no consumo de energia dessas máquinas ociosas? É exatamente nesta seção que você notará que entender sobre, neste caso, escalabilidade, ajuda as empresas por trás desses serviços a maximizarem seu faturamento, evitando uso desnecessário de recursos, sem comprometer a usabilidade e consequente satisfação de seus clientes. Curioso para descobrir como? Então vamos dar seguimento aos nossos estudos!

Na seção anterior, você foi apresentado a diversos aspectos importantes no desenvolvimento de sistemas distribuídos e, entre eles, merecem destaque especial a questão de segurança, escalabilidade, resiliência e heterogeneidade, razão pela qual dedicamos uma seção inteira apenas para tratar desses assuntos.

Você esteve no papel de um arquiteto de sistemas de uma empresa que esteva desenvolvendo um sistema distribuído para controle de manutenção preventiva de veículos, e identificou os principais objetivos que esse sistema deve atingir. Agora é hora de avançar no projeto, pesquisando e apresentando *frameworks* atuais, utilizados por grandes empresas no segmento de serviços em TI, e que o sistema esteja apto a ser comercializado. Legal, não é mesmo? Prepare um relatório e uma apresentação com exemplos e descrição desses *frameworks*. É sempre muito bom unir a teoria estudada com as práticas do mercado de trabalho, para que sua formação seja o mais completa possível e, desta forma, você esteja preparado para pleitear as melhores oportunidades no mercado de trabalho.

Segurança

Sem dúvida, um dos aspectos mais importantes no projeto de sistemas distribuídos é a segurança. Tipicamente, seja qual for a aplicação desenvolvida, sendo um sistema distribuído, funcionará em uma plataforma com várias máquinas, chamadas de nós, que replicam tal aplicação e, conforme já sabemos, a comunicação entre essas máquinas sempre ocorre por meio de redes de comunicação, tipicamente cabeadas.

A partir dessa análise, questões referentes à segurança desse sistema devem ser levadas em consideração. Segundo Coulouris et al. (2013), em termos de sistemas distribuídos, podemos pensar em dois níveis: o da confidencialidade e o da integridade dos dados.

- A confidencialidade dos dados refere-se ao acesso ao dado por indivíduos ou sistemas não autorizados.
- A integridade dos dados refere-se, além do acesso, à modificação do dado.

Exemplificando

Uma violação na confidencialidade dos dados ocorre quando, por exemplo, um hacker – indivíduo mal-intencionado, consegue acessar o valor de algum atributo em um banco de dados, mesmo que ele não saiba o significado daquele dado.

Um exemplo de integridade dos dados é quando um usuário atualiza sua senha em uma rede social, essa senha é atualizada em uma base de dados, ou seja, ocorre uma alteração nos dados. Quando esse tipo de alteração é feita por usuários não autorizados, como hackers, temos uma falha na integridade de dados.

Obviamente a segurança é um tema altamente complexo, e existem várias disciplinas que abordam diferentes aspectos dessa interessante área de estudo, mas, em linhas gerais, o projeto de sistemas distribuídos em termos de segurança remete a um exercício de equilíbrio entre custo e ameaças (COULOURIS et al., 2013). Ainda conforme Coulouris et al (2013), os pontos de atenção em relação à segurança, no projeto de sistemas distribuídos, são:

- Portas são expostas: sistemas distribuídos são construídos com base em um conjunto de processos que oferecem serviços e compartilham

informação. As portas de comunicação pelas quais esses serviços se comunicam são, intrinsicamente, abertas (para que clientes possam acessar tais serviços) e, dessa forma, um hacker pode enviar mensagem a qualquer uma dessas portas.

- Redes de computadores não são seguras: remetentes de mensagens podem ser falsificados, ou seja, um e-mail enviado por caique@caique.com pode não ter sido enviado pelo Caique; endereços IP podem estar duplicados, de forma que alguém malicioso possa receber as mesmas mensagens de um destinatário válido, etc.
- A validade das chaves criptográficas deve ser limitada: quanto mais tempo uma mesma chave estiver válida e ativa, maiores são as chances de esta estar comprometida, por ter maiores chances de ser conhecida (e explorada) por uma quantidade maior de pessoas e sistemas.
- Algoritmos de criptografia podem ter falhas: na atualidade, a melhor prática é de divulgar publicamente os algoritmos de criptografia para que a comunidade e entidades especialistas possam validar o algoritmo e sugerir melhorias, de forma que a privacidade esteja garantida pela chave criptográfica, e não pela inacessibilidade ao algoritmo utilizado.
- Hackers podem ter acesso a recursos poderosos: o custo dos recursos computacionais tem diminuído cada vez mais, de forma que máquinas poderosas estão acessíveis para a maioria da população. Assim, sempre considere que ataques podem ocorrer de inúmeras fontes e podem explorar vulnerabilidades utilizando inclusive ataques do tipo força-bruta (que tentam descobrir senhas por tentativa e erro, com simples "chutes").

Reflita

Você certamente deve ter ouvido falar, em meados de 2017, do WannaCry, um software malicioso do tipo *ransomware* que bloqueia o acesso a recursos através de criptografia e a vítima só consegue descriptografá-los através do pagamento de um resgate. Considerando esse tipo de ataque, você diria que afetou a confidencialidade, ou a integridade dos dados? Quais pontos de atenção em relação à segurança devem ter sido explorados por esse tipo de ataque? O que você faria para minimizar a possibilidade desse tipo de ataque?

Escalabilidade

A escalabilidade é outro aspecto importantíssimo de um sistema distribuído. Escalabilidade é um termo comum em redes de computadores, e

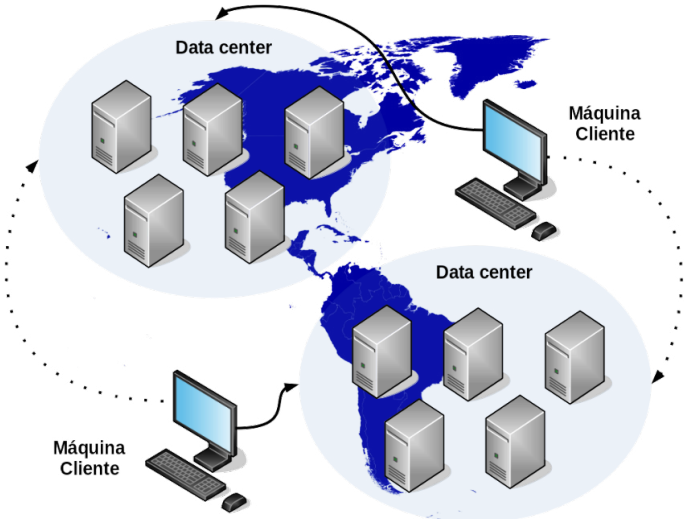
está intimamente ligada ao tamanho da rede. Segundo Tanenbaum e Steen (2008), um sistema cujo desempenho aumenta com o acréscimo de hardware e software, proporcionalmente à capacidade acrescida, é chamado escalável. É importante notar, entretanto, que um sistema dito escalável permite que se aumente ou diminua a quantidade de recursos. Você deve estar se perguntando: por que eu diminuiria a capacidade do meu sistema? Imagine a seguinte situação: você criou uma aplicação web que distribui conteúdo em vídeo para preparar estudantes para fazerem a prova do ENEM. Você roda essa aplicação, de maneira replicada, em um conjunto de servidores em nuvem de algum provedor de *cloud computing* conhecido do mercado, digamos, com 10 nós. Apesar da ideia ser excelente, você nota que a quantidade de usuários que utiliza sua plataforma cai drasticamente entre os meses de novembro e junho, uma vez que os estudantes começam normalmente a se preparar para esse exame – que ocorre anualmente, entre outubro e novembro – a partir de julho, quando estão de férias. Supondo que você paga para esse provedor de *cloud computing* R$ 150,00 mensais para que este disponibilize os 10 nós de maneira contínua, não seria interessante que, nos meses de menor demanda, você diminuísse a quantidade de servidores para, por exemplo, metade e, assim, pague nesse período a quantia de, digamos R$ 75,00 mensais? Nesse cenário, sua economia seria de R$ 600,00, que você poderia investir em outros projetos. Esse é um exemplo típico de escalabilidade "para baixo".

Assimile

A escalabilidade pode ser alterada, ou seja, aumentada ou diminuída, de maneira automatizada, sem a intervenção do desenvolvedor, por meio de ferramentas, como Chef e Ansible. Podemos exemplificar a função de ambas as ferramentas em um cenário de um portal de notícias, que tem, durante a madrugada, baixa demanda de acessos. As ferramentas exercem o papel de diminuir automaticamente os recursos utilizados. Em outro exemplo, quando sai uma notícia muito esperada e os acessos ao portal sobem drasticamente, as ferramentas exercem o papel de aumentar os recursos disponibilizados para garantir o bom funcionamento.

Dois aspectos importantes a serem levados em consideração em relação à escalabilidade são em termos geográficos e administrativos. Escalabilidade em termos geográficos refere-se ao sistema que, apesar de apresentar-se como único para o usuário, está rodando em várias réplicas, em dois ou mais datacenters geograficamente distintos. Podemos, por exemplo, utilizar um determinado provedor de *cloud computing* que possua *data centers* no Estado de São Paulo, aqui no Brasil, e no Estado do Arizona, nos EUA. A Figura 2.8 ilustra tal cenário.

Figura 2.8 | Exemplo de escalabilidade geográfica



Fonte: elaborada pelo autor.

O benefício desse tipo de configuração é fornecer uma melhor experiência – em termos de conectividade e latência (atrasos na rede) para os usuários, uma vez que os usuários mais próximos do hemisfério Norte podem acessar a aplicação através dos data centers nos EUA, e os usuários do hemisfério Sul podem acessar a aplicação através dos *data centers* no Brasil. Outra vantagem é que, na ocorrência de um desastre, por exemplo, um furação passar por Arizona, que comprometa o *data center*, todos os usuários poderão acessar o *data center* de outra localidade, incluindo os usuários do mais próximos ao *data* center atingido (ainda que a usabilidade, do ponto de vista desses usuários, seja ligeiramente comprometida, devido à maior distância desse *data center*).

Em termos administrativos, escalabilidade refere-se ao escopo administrativo, que é afetado pela escalabilidade geográfica e é um conceito bastante simples de ser compreendido, embora muitas vezes ignorado. Imagine que, no cenário da Figura 2.8, os *links* de comunicação do lado dos EUA sejam fornecidos por provedores de Internet daquela região, ao passo que os *links* de comunicação no lado do Brasil sejam fornecidos por provedores de Internet daqui. Caso o link no lado dos EUA fique indisponível, não vai adiantar entrar em contato com o provedor de Internet daqui do Brasil, pois é uma empresa diferente da que fornece o serviço nos EUA, administrativamente falando. Ou seja, o escopo administrativo foi, inerentemente, ampliado, o que significa que o responsável pelo sistema distribuído terá mais trabalho para

administrá-lo, incluindo, por exemplo, a necessidade de abrir um chamado de suporte técnico em outro idioma.

Tolerância a falhas / Resiliência

Quando falamos de aspectos no projeto de sistemas distribuídos, temos que falar de resiliência de processos, esse aspecto é também um dos principais objetivos a serem atingidos em uma aplicação distribuída. A resiliência de processos está relacionada com o sistema ter uma comunicação confiável entre as camadas de Cliente e Servidor. Sua ideia básica é de que os seus processos da nossa aplicação sejam replicados em grupos, isso faz com que o sistema tenha uma proteção contra falhas relacionadas a processos.

Para conseguir criar projetos tolerantes a falhas temos que ter em nossos sistemas uma detecção de falhas, assim como conseguir mascarar todas as falhas apresentadas e a replicação de nosso sistema, para que ela seja imperceptível. Só podemos atingir estas características de acordo com questões de projetos, e a mais importante neste caso é verificar qual o grupo de processos que a nossa aplicação deverá conter. Para isso, vamos entender o que são os grupos Simples e grupos Hierárquicos, e como esses processos são associados a esses grupos.

Quando falamos de resiliência de processos, organizamos em grupos os processos que são considerados idênticos. Portanto, quando uma mensagem é enviada a um grupo de nosso sistema, a ideia é que todos os processos membros desse grupo recebam a mensagem. Se ocorrer uma falha em um dos processos no tratamento da mensagem, outro processo desse grupo deve tratar a mensagem no lugar do processo com falha.

Em um sistema distribuído temos grupos dinâmicos, ou seja, os processos podem se mover entre os grupos que são criados. Um processo de um cadastro de usuário do site XYZ pode enviar uma mensagem para a realização do cadastro a um grupo de servidores sem precisar quantos existem e quem eles são.

Grupo Simples X Grupo Hierárquico

Quando falamos de grupos Simples, todos os processos são iguais e todas decisões são tomadas entre todos os processos, ou seja, de forma coletiva. A grande vantagem deste tipo de grupo é que não há um ponto único para falha. Mesmo que ocorra falha ou caia algum processo, o grupo continua mantendo o serviço em funcionamento. Portanto, a execução do sistema não é centralizada a um só ponto.

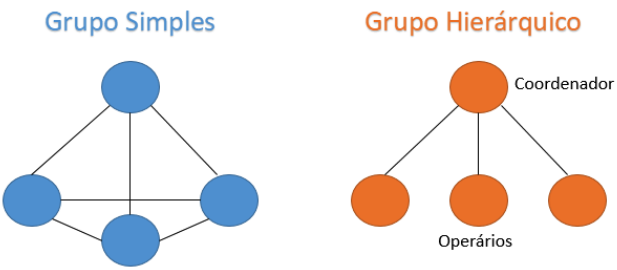
A desvantagem que encontramos nesse tipo de grupo é que a tomada de decisão tende a demorar mais pois cada decisão deve ser priorizada pelos processos, tendo assim uma votação antes da tomada de decisão.

Já os grupos Hierárquicos, como o nome mesmo já diz, são baseados em uma hierarquia, portanto existem processos considerados mais importantes e que controlam toda execução. Nesses grupos temos um processo chamado "coordenador" e os demais chamamos de "operários". Sempre que chega uma nova requisição no sistema ela é enviada ao processo coordenador, que decide o melhor operário para executá-la.

A grande vantagem desse tipo de grupo é que as decisões são centralizadas, portanto temos mais agilidade na tomada de decisão, o que gera um retorno mais rápido. Já a grande desvantagem apontada é que, caso ocorra uma falha no processo coordenador, o serviço todo para.

Podemos observar na Figura 2.9 a estrutura dos grupos apresentados e como seus processos se comunicam.

Figura 2.9 | Estrutura dos grupos e comunicação dos processos



Fonte: elaborada pelo autor.

Podemos observar que, na comunicação simples, os processos se comunicam entre si e decidem em conjunto qual o processo mais adequado para executar determinada ação durante o funcionamento de um sistema. Já na comunicação hierárquica os processos estão divididos entre "coordenador" e "operários", em que o processo coordenador irá definir o processo operário mais adequado para executar determinada ação durante o funcionamento de um sistema.

Heterogeneidade

O termo heterogeneidade vem de heterogêneo, ou seja, algo desigual, que apresenta estrutura, função ou distribuição diferentes. Portanto, aplicando isso ao conceito de computação, quando temos heterogeneidade estamos falando de um sistema que contenha em sua composição máquinas (nós) de sistemas

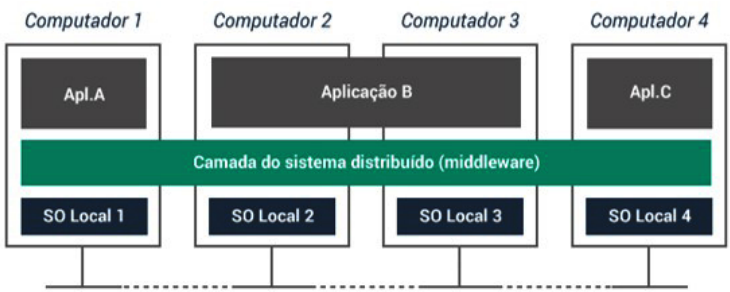
operacionais, recursos (hardware) e até mesmo fabricantes diferentes. Esse é um dos aspectos mais frequentes de um sistema que utiliza arquitetura distribuída. Geralmente, o sistema é composto por máquinas de diversas características diferentes que se comunicam para manter o funcionamento.

Os protocolos de redes são fundamentais para que essa comunicação entre máquinas diferentes ocorra, porém, na maioria das aplicações distribuídas, precisamos também de um middleware para que essa comunicação ocorra.

O middleware pode ser considerado um conjunto de padrões e funcionalidades que atua como uma camada central entre a nossa plataforma, o sistema operacional e as nossas aplicações. Essa camada central permite que em um sistema distribuído rodem diferentes aplicações em diferentes plataformas e que todas elas consigam se comunicar adequadamente.

Na Figura 2.10 temos um exemplo da atuação da camada Middleware perante diferentes computadores (máquinas), aplicações e sistemas operacionais. Podemos observar que a camada middleware liga aplicações A, B e C com os sistemas operacionais 1, 2, 3 e 4. Essas aplicações e sistemas operacionais estão em diferentes computadores, 1, 2, 3 e 4. Estes irão interagir entre si para a execução de um sistema distribuído, e isso só é possível devido à camada do sistema distribuído, chamada middleware.

Figura 2.10 | Middleware



Fonte: elaborada pelo autor.

Assimile

Em uma aplicação escrita em diferentes máquinas, com sistemas operacionais diferentes, é necessária a comunicação entre essas máquinas. Cada uma das máquinas fornecem a própria representação de dados e formas de comunicação, e, nesse caso, o Middleware atua como uma camada de tradução para que seja possível a comunicação correta entre as máquinas para o funcionamento do sistema.

Existem vários frameworks utilizados para implementação de plataformas de middleware, e os mais conhecidos são:

- CORBA - Framework de implementação de middleware baseado na linguagem de programação C++.
- .NET Remoting - Framework de implementação de middleware, baseado no ambiente de programação .NET, que pode utilizar várias linguagens diferentes, por exemplo: Visual Basic, C#, entre outras.
- Akka.NET - Framework de implementação de middleware, baseado no ambiente de programação .NET, que pode utilizar várias linguagens diferentes, como por exemplo: Visual Basic, C#, entre outras.
- Java RMI - Framework de implementação de middleware, baseado na linguagem de programação JAVA.
- JAX-WS - Framework de implementação de middleware, baseado na linguagem de programação JAVA.

Daqui a algumas seções vamos utilizar o JAVA-RMI para criar alguns exemplos.

Pesquise mais

- Você pode fixar seu conhecimento assistindo a este vídeo sobre os aspectos de projeto de um sistema distribuído:
  SISTEMAS DISTRIBUÍDOS LICENCIATURA EM COMPUTAÇÃO. **Aula 3 -** Sistemas Distribuídos – Características. 12 ago. 2014.
- Ainda, você pode assistir a esse vídeo sobre a comunicação entre entidades em um sistema distribuído, SISTEMAS DISTRIBUÍDOS LICENCIATURA EM COMPUTAÇÃO. **Aula 2 -** Sistemas Distribuídos - Modelos Arquiteturais**.** 12 ago. 2014.

Com o conteúdo estudado nesta seção, você está apto a entender os aspectos mais relevantes de um projeto de implantação de um sistema distribuído. Percebeu como é importante definir esses objetivos em um projeto de sistemas distribuídos? Agora, vamos continuar os estudos e seguir em frente com mais conceitos de sistemas distribuídos. Até breve!

Sem medo de errar

Você continua trabalhando no sistema de controle de manutenção preventiva para várias frotas de veículos de grandes transportadoras no Estado de Minas Gerais. Seu papel nesse projeto é orientar toda sua equipe para utilização de sistemas distribuídos, para que, ao final, o produto seja robusto e funcional.

Agora é hora de avançar no projeto, pesquisando e apresentando frameworks atuais, utilizados por grandes empresas no segmento de serviços em TI, que o sistema esteja apto a ser comercializado. Legal, não é mesmo? Prepare um relatório e uma apresentação com exemplos e descrição desses frameworks.

É muito importante que a equipe enxergue também as dificuldades em se implementar um sistema distribuído. Assim, para o sistema distribuído a ser implementado, identifique, de acordo com o material apresentado nesta seção, três desafios que podem surgir ao implementar o mesmo. Pense também em como você pode justificar esta resposta, ou seja: por que esses seriam desafios para que o sistema distribuído atingisse seus objetivos? Além disso, você deve apresentar soluções para Escalabilidade do seu sistema distribuído, para isso, faça um relatório comparando três frameworks atuais que facilitam atingir a escalabilidade desejada em sistemas distribuídos e aponte a melhor opção para esse projeto. Ao final, crie uma apresentação para sua equipe, com os três desafios escolhidos e a comparação dos 3 frameworks pesquisados.

Para solucionar essa tarefa, utilize os conteúdos apresentados nesta seção. Sugerimos basear-se nos aspectos de projetos apresentados nesta seção: Segurança, Escalabilidade, Resiliência e Heterogeneidade. Para enriquecer sua apresentação, fale sobre os desafios encontrados para atingir êxito nos aspectos de projetos escolhidos.

Para a pesquisa e comparação dos Frameworks você pode consultar as seguintes documentações:

- Java RMI <https://docs.oracle.com/javase/7/docs/technotes/guides/rmi/hello/hello-world.html>
- CORBA <https://docs.oracle.com/javase/7/docs/technotes/guides/idl/corba.html>
- AKKA.NET <https://petabridge.com/training/akka-remoting/>
- JAX-WS <https://docs.oracle.com/javaee/6/tutorial/doc/bnayl.html>

Após a comparação dos Frameworks, você deve escolher o que melhor auxilia a Escalabilidade do projeto, não se esquecendo de justificar sua

escolha para impressionar a equipe. Você pode comparar as linguagens de programação utilizadas por cada um dos frameworks, verificando sua popularidade, o conteúdo disponível para aprendizado na internet, os requisitos mínimos para execução e desenvolvimento na linguagem, entre outros aspectos. Você também pode comparar os recursos necessários para execução de cada um dos frameworks, descobrindo, assim, qual é mais viável para este tipo de sistema.

Avançando na prática

Criação de sistema de vendas

Descrição da situação-problema

Uma distribuidora de produtos da região, está querendo criar um sistema de vendas para auxiliar os seus 10 vendedores, que emitem pedidos por alguns bairros mais próximos. Para isso, a distribuidora resolveu contatar você, pois a atual equipe da empresa não tem experiência em implementação de sistemas. O responsável pelo setor de TI da distribuidora explicou que, em seu cenário, eles possuem três máquinas muito distintas, com diferentes sistemas operacionais disponíveis para a criação e execução do sistema. Portanto, você deve encaminhar um e-mail ao responsável pela TI apresentando o aspecto de projeto que se enquadra nesta situação e mostrar como solucionar esse problema para a pequena equipe de TI da distribuidora. O responsável pela TI também deseja receber algumas informações sobre CORBA; informe-as no e-mail.

Resolução da situação-problema

Caro responsável pelo departamento de TI da distribuidora,

Analisando o ambiente indicado com 3 máquinas distintas em questões de hardware e software, para a criação do sistema de vendas, vocês devem utilizar uma arquitetura distribuída. Além disso, o principal aspecto de projeto que sua equipe deve levar em consideração é o de heterogeneidade, pois ele retrata a capacidade de diferentes máquinas funcionarem em conjunto para rodar uma aplicação. Portanto, para atingir esse objetivo, você precisará de um middleware que funcione como uma camada de comunicação entre a aplicação e os sistemas operacionais das diferentes máquinas. Para a implementação do middleware, será necessário um framework de implementação. Aconselhamos utilizar o JAVA RMI ou CORBA, que são muito populares.

O CORBA é uma arquitetura aberta para o desenvolvimento de aplicações distribuídas em um ambiente com várias linguagens e plataformas diferentes. Suas principais características são: utilização da arquitetura SOA, tecnologia disponível desde a década de 90, orientação a objetos e eficiência no transporte de dados, inclusive binários.

Um sistema desenvolvido em CORBA adota um modelo de comunicação cliente-servidor. Nesse modelo, um servidor é um provedor de serviços, um cliente é um consumidor de serviços e um componente pode atuar tanto como cliente como servidor.

Qualquer dúvida que você e sua equipe tiverem em relação à implementação, por favor, entrem em contato!

Atenciosamente,

Analista de TI.

Faça valer a pena

1. Quando planejamos implantar um sistema distribuído, há vários fatores importantes que devem ser levados em consideração. Podemos chamá-los de aspectos de projeto, e, no projeto de sistemas distribuídos, os principais são: Segurança, Escalabilidade, Resiliência e Heterogeneidade.

A capacidade de máquinas com diferentes sistemas operacionais se comunicarem na execução de um sistema se relaciona a qual aspecto de projeto?
a) Segurança.
b) Escalabilidade.
c) Resiliência.
d) Heterogeneidade.
e) Transparência.

2. Um dos aspectos mais importantes no projeto de sistemas distribuídos é a segurança. Tipicamente, seja qual for a aplicação desenvolvida, sendo um sistema distribuído, esta funcionará em uma plataforma com várias máquinas, chamadas de nós, que replicam tal aplicação. Essa plataforma tem a comunicação entre as máquinas feita através de redes de comunicação.

Sabendo que a segurança é um aspecto muito importante, analise as afirmações abaixo e escolha a opção com pontos de atenção em relação à segurança do sistema.
I – Máquinas de diferentes sistemas operacionais se comunicam.
II – Portas abertas.

III – Algoritmos de criptografia podem ter falhas.
IV – Camada de Middleware.
V – A validade das chaves criptográficas.

a) Somente as afirmações I e IV são pontos de atenção em relação à segurança.
b) Somente as afirmações I e II são pontos de atenção em relação à segurança.
c) Somente as afirmações II, III e V são pontos de atenção em relação à segurança.
d) Somente as afirmações II e V são pontos de atenção em relação à segurança.
e) Somente as afirmações I e III são pontos de atenção em relação à segurança.

3. O termo Escalabilidade trata de outro aspecto de projeto de sistemas distribuídos. Um sistema cujo desempenho aumenta com o acréscimo de hardware e software, proporcionalmente à capacidade acrescida, é chamado escalável. Esse termo está relacionado diretamente ao desempenho da aplicação e ao consumo de seus recursos.

Sobre o aspecto de projeto de Escalabilidade em um sistema distribuído, marque V para verdadeiro ou F para falso:
I - ( ) Um sistema Escalável é aquele que está disponível para acesso 100% do tempo.
II - ( ) Espera-se que um sistema distribuído escalável possa aumentar ou diminuir a quantidade de seus recursos.
III - ( ) Escalabilidade em termos geográficos refere-se ao sistema que está rodando em várias réplicas, em dois ou mais datacenters geograficamente distintos.
IV - ( ) O termo Escalabilidade está relacionado à comunicação de máquinas e processos na execução do sistema.

Assinale a alternativa que representa a sequência CORRETA:
a) V - V - V - F.
b) F - V - V - V.
c) V - F - V - F.
d) F - F - V - F.
e) F - V - V - F.

Seção 2.3

Clientes e Servidores

Diálogo aberto

Caro aluno, seja bem-vindo a mais uma seção do nosso estudo sobre sistemas distribuídos. Você já parou para pensar nos diferentes papeis que uma mesma máquina pode exercer para o funcionamento de um sistema? Cada papel exercido por uma máquina é de suma importância para o funcionamento correto de uma aplicação. Além disso, as máquinas são divididas em categorias, cada qual com sua função e importância. Você já parou para pensar que, na máquina utilizada por você para realizar seus estudos, há uma função bem definida em relação às capacidades e limitações de acesso que uma aplicação pode ter?

Caso você precise acessar um artigo na nuvem, indicado por seu professor de sistemas distribuídos, sua máquina exerce a função de máquina Cliente e o servidor de nuvem em que o arquivo está hospedado exerce a função de máquina Servidor.

Nesta seção, vamos trabalhar conceitos e definições de máquinas do tipo cliente, servidor e workstation. Você sabia que existem tantas categorias de máquinas? Além disso, já ouviu falar em algum desses tipos?

Após sua liderança na equipe do projeto de um sistema de controle de manutenção preventiva para várias frotas de veículos de grandes transportadoras, você conseguiu, por meio dos diagramas, mostrar claramente para sua equipe quais objetivos devem ser alcançados com esse sistema e quais as principais dificuldades que vocês deverão encontrar na implementação. Agora o seu time precisa estar muito bem familiarizado com os diferentes papéis que as máquinas (PCs, smartphones, etc.) assumem em um sistema distribuído. Através do smartphone e dos PCs, os funcionários irão lançar a quilometragem dos veículos para que as máquinas servidor recebam essas informações e as processem, informando assim o controle de manutenção. Para o sistema de controle de manutenção preventiva que sua empresa deve implementar, você deve exemplificar uma máquina para cada umas das seguintes categorias: cliente, servidor e workstation. Como você pode fazer uma análise de custo benefício para o hardware mais indicado em cada categoria para este projeto em específico? Para ajudar sua equipe, você deve utilizar todos os seus conhecimentos, apontando quais máquinas do sistema de controle de manutenção preventiva para várias frotas de veículos de grandes transportadoras se enquadram em cada categoria. Entregue todas as informações em forma de relatório.

Para completar esse desafio, nesta seção você verá como funcionam as máquinas das categorias cliente, servidor e workstation, incluindo onde são aplicadas e suas vantagens e desvantagens. Ficou curioso? Espero que você se sinta motivado para adquirir esses novos conhecimentos, utilizando seus esforços neste estudo. Os conceitos apresentados devem enriquecer sua vida profissional. Vamos lá!

Não pode faltar

Introdução aos modelos Clientes e Servidores

Mesmo que você não saiba, o modelo de arquitetura de rede do tipo cliente-servidor é um dos modelos mais utilizados de arquitetura, em diversos tipos de sistemas, tais como entretenimento, aplicações B2B (*business-to-business*), B2C (*business-to-consumer*), etc. Gostaríamos que você utilizasse esse modelo para entender os diferentes papéis entre as máquinas do tipo cliente e as do tipo servidor. Nesse tipo de arquitetura, teremos alguns serviços e recursos a serem compartilhados por vários usuários, e que podem ser disponibilizados em um único computador – sendo chamados de Servidores multisserviço – ou ser segregados, ou seja, um serviço ou recurso por computador, caso em que são chamados de acordo com os serviços disponibilizados (por exemplo, Servidor de banco de dados, Servidor de autenticação de usuários, etc.).

Esses Servidores estarão conectados à Internet através de equipamentos de rede, tais como switches, roteadores e firewalls, para que as pessoas possam acessá-los remotamente e utilizar os serviços e recursos disponibilizados por estes através de máquinas denominadas Cliente (por exemplo, laptops, desktops, smartphones e tablets), que por sua vez também estarão conectadas através de equipamentos de rede à Internet. Importante observar que essa arquitetura, na maioria das vezes, também funciona no nível de rede local (do inglês, *Local Area Network* – LAN).

Máquinas Clientes

As máquinas Cliente são as mais simples de serem entendidas, pois são as mesmas máquinas que utilizamos no nosso dia a dia para execução das tarefas de propósito geral, rotineiras, como acessar um website, jogar um game, redigir um documento digital, assistir a um vídeo, compartilhar arquivos na nuvem, etc. Exemplos desse tipo de computador são computadores desktop, laptops e smartphones. Essas máquinas Cliente têm a função

de enviar e receber requisições/solicitações de máquinas dos diversos tipos de servidores, que entraremos em detalhes.

Assimile

Quando você acessa um website, qualquer que seja seu conteúdo, a máquina cliente tem a função de gerar uma requisição a um servidor utilizando o protocolo HTTP. Após isso, a máquina cliente recebe esse conteúdo através da resposta do protocolo HTTP gerado por outra máquina.

Algumas das características desse tipo de máquina, quando fazem solicitações para uma outra máquina, são:

(i) Em um sistema distribuído, as máquinas cliente sempre são responsáveis por iniciar as solicitações/requisições ao servidor;

(ii) A máquina cliente aguarda por respostas de outros servidores;

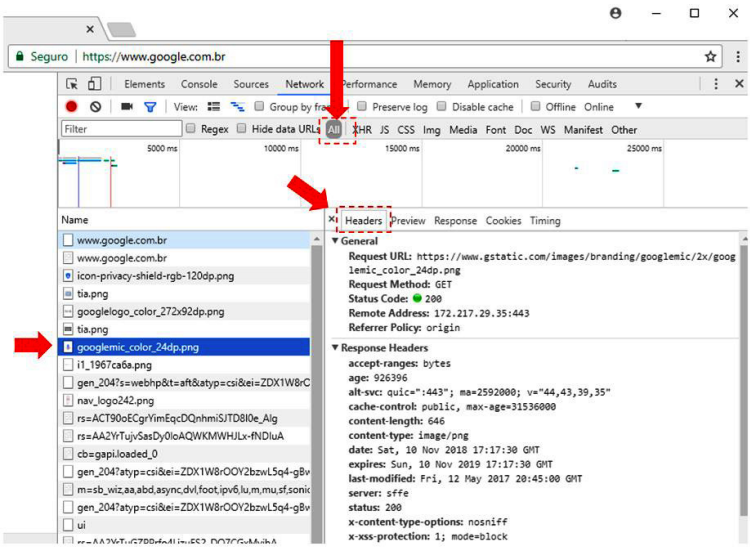
(iii) A máquina cliente recebe respostas de outros servidores;

(iv) A máquina cliente, geralmente, se conecta a um pequeno número de servidores de uma só vez;

(v) A máquina cliente, geralmente, interage diretamente com os usuários finais através de uma interface gráfica.

Uma das ferramentas disponíveis aos usuários mais curiosos, que pode ser usada para explorar requisições da máquina cliente, é o kit "Ferramentas do desenvolvedor", disponível nos navegadores de internet. Observe a Figura 2.11, ao acessar o endereço oficial do Google, com a ferramenta aberta, e pedir para mostrar todos os arquivos que "estão vindo" do servidor (opção *All*), você pode selecionar um arquivo específico e, através dos cabeçalhos (*Headers*), pode verificar informações técnicas, como o método de requisição, que nesse caso foi o GET, o endereço remoto, a status da solicitação, entre outras informações.

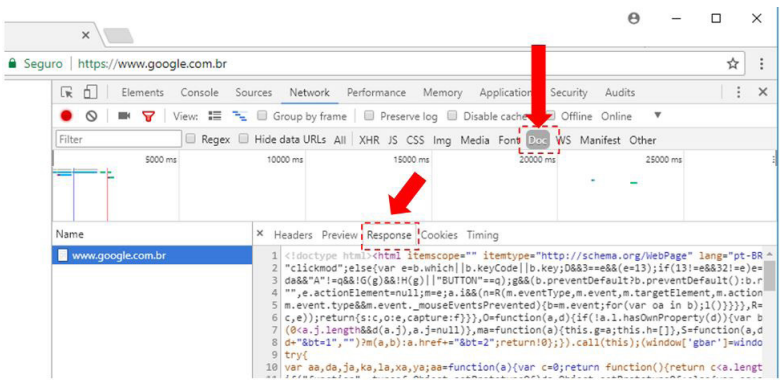
Figura 2.11 | Ferramentas do desenvolvedor – Opção *All*



Fonte: captura de tela do Google Chrome, elaborada pelo autor.

Já na Figura 2.12, ao invés de selecionarmos a opção para mostrar todos os arquivos, se optarmos por ver somente os arquivos (Doc), podemos ver a resposta que foi enviada. Embora a resposta seja confusa, observe o comando inicial `<!doctype html>`. Parece familiar? Toda resposta que uma máquina cliente recebe é "traduzida" em um arquivo com formato HTML, pois essa é a linguagem oficial usada na web. Mesmo que os desenvolvedores tenham usado frameworks Java, .NET, Python, etc. para implementar o sistema, a resposta sempre será traduzida em HTML.

Figura 2.8 | Ferramentas do desenvolvedor – Opção *Doc*



Fonte: captura de tela do Google Chrome, elaborada pelo autor.

Máquinas Servidores

Um servidor, de uma forma geral, consiste em um equipamento responsável por receber solicitações de usuários externos e, em seguida, essas solicitações são processadas por sistemas instalados nesse hardware e, consequentemente, uma resposta é enviada para o usuário solicitante.

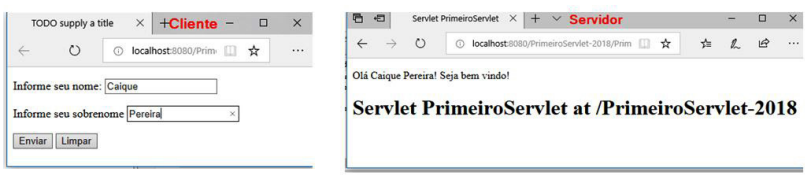
Dependendo do contexto, podemos definir servidores como as máquinas conectadas à rede que recebem as solicitações ou até mesmo os programas instalados nessas máquinas. Diferentemente do que se acredita, recursos não físicos, como outros sistemas também podem ser denominados servidores. Um bom exemplo dessa abordagem é a utilização de servidores Web, que, por sua vez, são sistemas responsáveis por receber requisições pela internet ou por uma rede local e retornar o conteúdo das páginas que são visualizadas nos diversos navegadores, "servindo" simultaneamente diversos usuários (YADAV, 2009).

Tipos de Servidores: No que diz respeito ao software, servidores podem ser de diversos tipos. Entre eles, podemos citar os servidores web, os servidores de e-mail, servidores FTP, servidores de autenticação e servidores de banco de dados. Agora, vamos entender melhor cada um deles.

Servidores Web: responsáveis pela maioria dos conteúdos solicitados por meio de um endereço, seja ele um IP ou uma URL. Suas respostas normalmente são páginas que podem ser exibidas por meio de um browser. Na Figura 2.13 (a), temos o preenchimento do formulário com nome e sobrenome do lado do cliente e o processamento do servidor e resposta enviada até a máquina cliente novamente. Mas, como é possível realizar a comunicação cliente-servidor? Existem diversas maneiras de implementar tal tarefa, usando as várias linguagens de programação disponíveis. Nesse caso, usamos a tecnologia *Servlet* do Java, que recebeu os dados do formulário através do comando "`request.getParamenter()`"; com isso, guardamos os valores em variáveis e exibimos com o comando "`out.println()`", como podemos observar na Figura 2.13 (b).

Figura 2.13 | Exemplo de aplicação em máquina servidor

a) Exemplo de resposta do servidor



b) Tecnologia usada para implementar

```
response.setContentType("text/html;charset=UTF-8");
try (PrintWriter out = response.getWriter()) {
    /* TODO output your page here. You may use following sample code. */
    out.println("<!DOCTYPE html>");
    out.println("<html>");
    out.println("<head>");
    out.println("<title>Servlet PrimeiroServlet</title>");
    out.println("</head>");
    out.println("<body>");
    String nome = request.getParameter("nome");
    String sobrenome = request.getParameter("snome");
    // int valor = Integer.ParseInt(request.getParameter("v1"));
    // double valor = Double.ParseDouble(request.getParameter("v1"));
    out.println("<p>Olá " + nome + " " + sobrenome + "! Seja bem vindo!");
    out.println("<h1>Servlet PrimeiroServlet at " + request.getContextPath() + "</h1>");
    out.println("</body>");
    out.println("</html>");
```

Fonte: captura de tela do Eclipse, elaborada pelo autor.

Exemplificando

Três exemplos bastante populares de servidores web são o **Apache** e o **Nginx,** em ambientes livres, e o IIS em ambientes proprietários.

O servidor web IIS (*Internet Information Service*) consiste em um servidor web executado em ambiente Windows. Também conhecido como Windows web server, esse servidor é disponibilizado juntamente com o Windows, dependendo apenas de uma ativação de recursos para funcionar. Esse servidor é compatível com desde páginas HTML simples até páginas dinâmicas construídas em ASP.NET. Além disso, o servidor já vem nativo em todas as versões do Windows e só precisamos ativá-lo.

Para ativar o IIS, siga os passos abaixo:

- Acesse o “Painel de Controle”;
- Clique em “Ativar e Desativar Funcionalidades do Windows”;
- Escolha a opção “Serviços World Wide Web”.
- Selecione as ferramentas padrões para IIS e Console de Gerenciamento do IIS.

Caso o sistema operacional seja Windows 10, o mesmo serviço pode ser encontrado no utilitário “Adicionar e Remover Programas”, no “Painel de Controle”. Em seguida escolha a opção “Ativar e Desativar Recursos do Windows”. Habilite os “Serviços de Informação da Internet” e aguarde a instalação das ferramentas necessárias.

Terminada a Instalação, é possível acessar o IIS digitando o nome do mesmo na barra de busca do menu iniciar.

Após a sua instalação, o servidor IIS traz um site adicionado por padrão, contudo é comum adicionarmos diversos sites em um mesmo servidor. Assim, os passos a seguir podem ser utilizados para cadastrar um novo site:

1. Acesse o servidor Web com privilégios de administrador;
2. Acesse o Painel de Controle;
3. Abra o Gerenciador de Serviços da Internet dentro de Ferramentas Administrativas;
4. Escolha a criação de um Novo Site;
5. Após o início do Assistente para Criação de Site, clique em Avançar;
6. Informe uma descrição interna para o site;
7. Informe um endereço IP para o site;
8. Digite o número da porta TCP destinada à publicação do site;
9. Informe o nome de cabeçalho de Host;
10. Clique em Avançar;
11. Aponte para o local no qual se encontram os arquivos do site;
12. Defina as permissões de acesso;
13. Clique em Concluir.

Para verificarmos se nosso servidor está funcionando e se nosso site está acessível, a melhor forma é publicar um arquivo de teste e tentar acessá-lo pela rede. O código a seguir corresponde a uma página HTML simples, que nos ajudará a verificar nossas configurações de servidor. Abra um bloco de notas, digite o código a seguir e salve como index.html.

```
<html>
  <head>
   <title> Aula de Sistemas Distribuídos </title>
    <meta charset="UTF-8">
  </head>
  <body>
    <h1>Aula de Sistemas Distribuídos</h1>
   <p> Conhecendo o servidor IIS – <b> Professor Caique </b> </p>
  </body>
</html>
```

Após a criação do site no servidor IIS e a associação do mesmo com o a pasta que contém o arquivo de teste acima, basta acessarmos o mesmo pelo navegador para verificar sua disponibilidade, como podemos observar na Figura 2.14:

Figura 2.14 | Execução de página através do servidor IIS

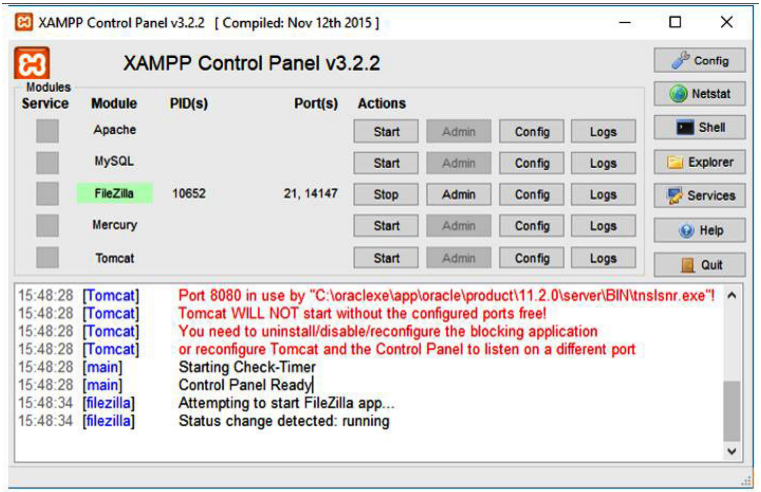
Fonte: captura de tela do Eclipse, elaborada pelo autor.

Servidores de E-mail: responsáveis por receber e enviar e-mails, assim como armazenar as caixas de entrada e de saída dos usuários. Um dos grandes servidores de e-mail utilizados nos ambientes corporativos é o Microsoft Exchange; através dele, muitas empresas mantém os serviços de e-mail tanto interna quanto externamente. Para a configuração do serviço, devemos informar o caminho/endereço do servidor de e-mail e o nome de usuário que geralmente está cadastrado no servidor de autenticação Active Directory, que será detalhado em breve.

Servidores FTP: o FTP (*File Transfer Protocol*) é um protocolo específico para transferência de arquivos cliente-servidor. Geralmente, é dessa forma que os sites são transferidos para um servidor de hospedagem. Uma hospedagem é um servidor de conteúdo em que disponibilizamos arquivos de internet (HTML, CSS e imagens) para serem acessados através de navegadores. Dois exemplos bastante populares de servidores FTP são o VSFTP, para ambientes livres, e o *FileZilla*, tanto para ambientes proprietários como para ambientes livres. Um exemplo clássico da utilização desse protocolo é referente ao próprio desenvolvimento de sistemas web. Quando os desenvolvedores finalizam o sistema, eles precisam disponibilizá-lo e, para isso, transferem os arquivos para um sistema de hospedagem. Para acessar um servidor FPT, o cliente precisará de uma interface, o que é feito por meio de aplicativos como o *FireFTP*, *SmartFTP*, *FileZilla Client*, entre vários outros. Agora, vamos aprender a usar o serviço com o servidor FTP *Filezilla*, através do XAMPP, e o cliente *Filezilla*.

Nosso primeiro passo é configurar o lado Servidor do *Filezilla* FTP. Para isso vamos abrir o XAMPP e iniciar o serviço *Filezilla*, como mostra a Figura 2.15:

Figura 2.15 | Iniciando o servidor FTP *Filezilla*

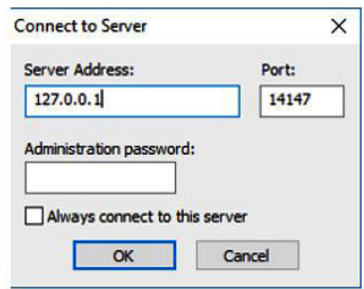


Fonte: captura de tela do XAMPP, elaborada pelo autor.

Feito isso vamos clicar em Admin e começar a configurar. Deixaremos as credenciais de acesso como estão e clicaremos em "OK" (Figura 2.16):

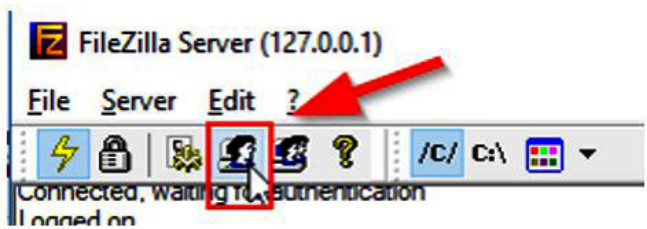
Agora, vamos criar o nosso usuário FTP para ser acessado através do Cliente. Vamos até o ícone destacado na Figura 2.17 ou no menu "*Edit* >> *Users*":

Figura 2.16 | Iniciando o servidor FTP *Filezilla*



Fonte: captura de tela do XAMPP, elaborada pelo autor.

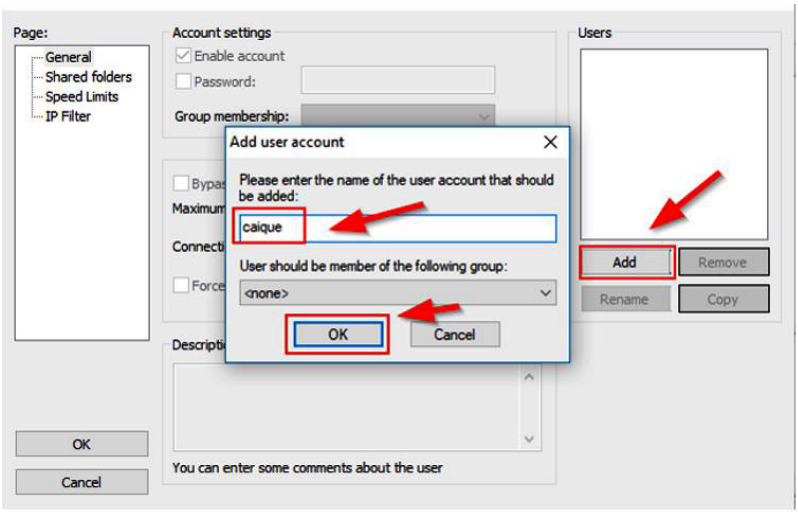
Figura 2.17 | Iniciando o servidor FTP *Filezilla*



Fonte: captura de tela do *FileZilla*, elaborada pelo autor.

Feito isso clicamos em "*Add*", definimos um nome e avançamos, conforme a Figura 2.18:

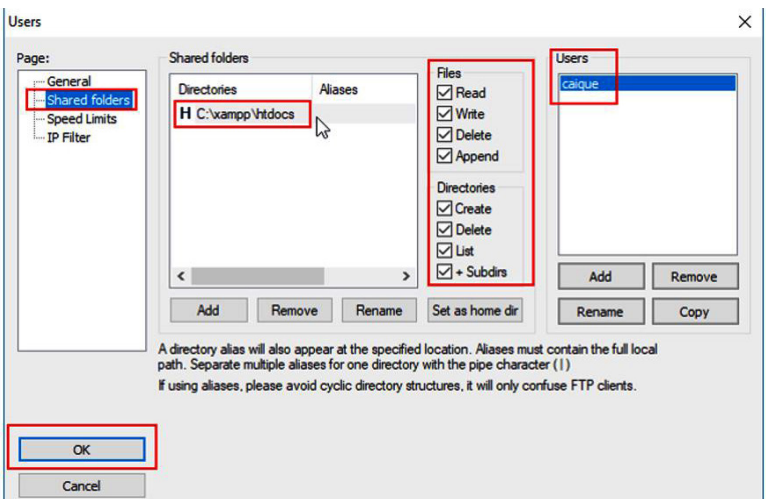
Figura 2.18 | Iniciando o servidor FTP *Filezilla*



Fonte: captura de tela do *FileZilla*, elaborada pelo autor.

Agora vamos definir o diretório padrão do usuário FTP e suas permissões, conforme ilustra a Figura 2.19.

Figura 2.19 | Iniciando o servidor FTP *Filezilla*



Fonte: captura de tela do *FileZilla*, elaborada pelo autor.

Agora nosso servidor FTP está apto a receber conexões de Clientes. Vamos fazer isso de duas formas: pelo Cliente *FileZilla* e pelo Servidor Linux.

Para acessar via Cliente *FileZilla*, vamos precisar da ferramenta *FileZilla Client*. Você pode fazer o download no site oficial. Agora podemos acessar inserindo as credenciais criadas anteriormente, assim, podemos fazer upload e download (transferências) de arquivos através do protocolo FTP.

Pelo Linux, devemos inserir no terminar o comando FTP e o "caminho" do servidor: `ftp 127.0.0.1`. Agora, o servidor vai pedir nossas credenciais de acesso, criadas anteriormente. Utilizamos o comando "ls" para listar os arquivos que estão no diretório, desta forma, podemos listar todos os arquivos que estão no armazenamento de nosso servidor FTP.

Servidores de Autenticação: possibilitam que sistemas internos utilizem sua base de usuários para validar acessos. Dois exemplos bastante populares de servidores de autenticação são o SAMBA4, para ambientes livres, e o *Active Directory*, para ambientes proprietários, que é muito utilizado em ambientes corporativos.

Servidores de Banco de Dados: possibilitam o armazenamento de informações importantes para o funcionamento do sistema de maneira persistente (não-volátil). Esse tipo de servidor manipula informações armazenadas no banco de dados, por exemplo, dados de um usuário cadastrado em alguma plataforma de compras. Dois exemplos bastante populares de servidores de banco de dados são o Oracle, para ambientes proprietários, e o PostgreSQL, para ambientes livres.

Além dos tipos de servidores apresentados anteriormente, existem dezenas de outros tipos. É importante observar que um servidor é uma máquina que está em funcionamento permanente, vinte e quatro horas por dia, sete dias por semana, trezentos e sessenta e cinco dias por ano, aguardando requisições por parte dos clientes. Além disso, nem sempre o servidor é uma máquina física: embora não seja muito comum, podemos – para fins de teste – instalar um programa de computador que tem como principal objetivo executar a função de um servidor dedicado. Outro aspecto interessante é que, dependendo da capacidade de hardware (processador, RAM e disco de armazenamento), vários servidores podem rodar em uma mesma máquina.

Reflita

Agora que você conhece diversos tipos de servidores, já parou para pensar quantos deles são utilizados em suas tarefas diárias, como ao acessar uma página de notícias ou um serviço de e-mail; ou em onde estão salvas suas informações cadastradas na sua rede social preferida?

Máquinas Workstation

As máquinas do tipo *workstations*, conhecidas também como estações de trabalho, são máquinas que possuem especificações de hardware superiores aos computadores comuns. O termo máquina *workstation* é utilizado para definir um computador montado de maneira exclusiva para executar uma atividade que necessite de recursos como memória RAM, disco rígido e processador mais poderosos do que em computadores comuns.

Máquinas desse tipo precisam ter um alto desempenho na execução de tarefas de cunho profissional, por exemplo, arquitetura, desenho industrial, aplicação de softwares gráficos de modelagem 3D, edição de fotos, edição de vídeos, inteligência artificial e aplicações de *Big Data*. Geralmente, esse tipo de máquina utiliza softwares e ambientes proprietários, isso quer dizer que dependem de licenças. Essas licenças muitas vezes são adquiridas para o conjunto todo de máquinas da empresa. Esse tipo de licenciamento se difere do que estamos acostumados com máquinas domésticas.

Pesquise mais

- Você pode fixar seu conhecimento assistindo a este vídeo sobre modelo cliente e servidor:
  LÉO MATOS. **MODELO CLIENTE SERVIDOR -** Professor Léo Matos. 5 jul. 2012.
- Veja também este vídeo sobre os tipos de servidores:
  TIAGO SCALCO. **Tipo de Servidores -** Redes de computadores. 21 jun. 2016.
- Por fim, assista a este vídeo sobre como funciona um data center:
  CANALTECH. **Você sabe o que é e como funciona um data center?** 11 jul. 2013.

Assim, finalizamos esta seção. Bons estudos e até a próxima!

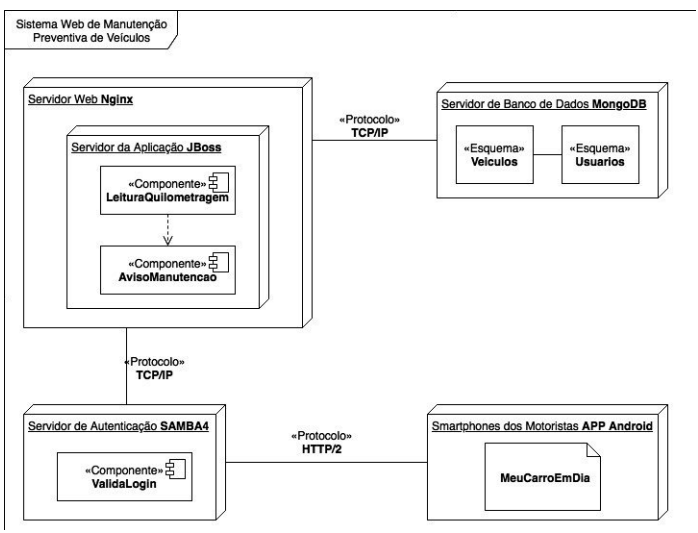
Sem medo de errar

Você continua participando do desenvolvimento do sistema de controle de manutenção preventiva para várias frotas de veículos de grandes transportadoras no Estado de Minas Gerais. Como você sabe, o time de desenvolvimento é formado por desenvolvedores que nunca haviam trabalhado com sistemas distribuídos, e você os está orientando para que o produto final seja robusto e funcional.

Após sua liderança na equipe do projeto, você conseguiu, através dos diagramas, mostrar claramente para sua equipe quais objetivos devem ser alcançados com esse sistema e quais as principais dificuldades que vocês deverão encontrar na implementação. Agora o seu time precisa estar muito bem familiarizado com os diferentes papéis que as máquinas (PCs, smartphones, etc.) assumem em um sistema distribuído. Para o sistema de controle de manutenção preventiva que sua empresa deve implementar, exemplifique uma máquina para cada umas das seguintes categorias: cliente, servidor e *workstation*. Descreva suas funcionalidades nesse sistema para comprovar sua resposta e faça uma análise de custo benefício para o hardware mais indicado em cada categoria para esse projeto em específico. Coloque todas essas informações em um relatório. Utilize o diagrama de implantação da Situação Problema 1 desta unidade para auxiliar nesta tarefa.

Vamos utilizar o diagrama da Figura 2.20 como o de implantação do nosso sistema.

Figura 2.20 | Diagrama de implantação do sistema de controle de manutenção preventiva para frotas de veículos de grandes transportadoras no Estado de Minas Gerais.



Fonte: elaborada pelo autor.

Vamos começar a nossa resolução definindo um exemplo de cada máquina nesse sistema e apontando suas funcionalidades.

Máquinas clientes – são os smartphones dos motoristas que devem fazer o lançamento da quilometragem dos veículos, e esse lançamento deve retornar o prazo apontado pelo servidor para a próxima manutenção.

Máquinas servidores – nesse tipo de sistema, precisaremos de uma máquina do tipo servidor web para receber as informações enviadas pelas máquinas clientes dos motoristas. Precisamos de um servidor de aplicação para "rodar" nosso sistema. Também precisaremos de um servidor de autenticação para validar o acesso somente para usuários credenciados. Por fim, precisaremos de um servidor de banco de dados para armazenar as informações da frota de veículos, assim como informações sobre as manutenções.

Máquinas *workstation* – nesse tipo de sistema, essas máquinas são utilizadas pela equipe de desenvolvimento, todas com as licenças necessárias em ambientes proprietários e nos seus softwares específicos para o projeto.

Máquinas clientes – smartphones com sistema operacional Android, a partir da versão 5.0 (*Lollipop*).

Máquinas servidores – Nginx para servidor web, JBoss para servidor de aplicação, SAMBA4 para servidor de autenticação e MongoDB para servidor de banco de dados.

Máquina workstation – máquinas capazes de executar/desenvolver aplicações em na linguagem Java.

Agora que já temos as configurações e funcionalidades desses servidores, você deve fazer uma análise apontando o custo benefícios das máquinas que são capazes de executar os servidores listados acima. Utilize a internet como seu aliado para cotar os preços e definir as máquinas mais viáveis para o seu projeto.

Avançando na prática

Licenciamento de máquinas do tipo workstation

Descrição da situação-problema

No ambiente coorporativo onde você trabalha, iniciou-se o desenvolvimento de um novo projeto. Para a implantação e desenvolvimento desse projeto, foram criadas equipes de infraestrutura e desenvolvimento. Nelas, teremos várias máquinas do tipo workstation que utilizam os sistemas operacionais Windows Server 2016 e Windows 10. Além disso, temos outras

ferramentas que necessitam de licenças, como o Visual Studio 2017, Pacote Office e SQL Server. Seu gerente pediu para que você faça um levantamento das licenças necessárias em cada um dos softwares mencionados na situação deste novo projeto. Crie uma lista com as licenças mais comuns e monte um relatório para que ele tenha os tipos de licenças que serão necessárias e encaminha para a equipe do financeiro da empresa.

Resolução da situação-problema

Windows 10 – O tipo de licença utilizada pelo sistema operacional Windows é a ESD (*Electronic Software Delivery*). Essa licença é equivalente a antiga licença FPP (*Full Packaged Product*) e permite reinstalar o sistema quantas vezes forem necessárias, como ao realizar uma formatação do equipamento ou mesmo migrar essa licença instalando em outro equipamento. O pacote mais viável oferecido pela Microsoft é o Microsoft 365 *Enterprise,* que contém o Windows 10 *Enterprise* e o Office 365, que podem cobrir as licenças de dois softwares que vamos precisar para o projeto.

Windows Server 2016 – O tipo de licença utilizada pelo sistema operacional Windows Server 2016 também é a ESD (*Electronic Software Delivery*). O plano mais viável para este tipo de projeto é da versão *Essentials*, que é utilizada por pequenas empresas com até 25 usuários e 50 dispositivos.

Pacote Office – O pacote Office utilizará a mesma licença do Windows 10, o Microsoft 365 *Enterprise,* que possui o Office 365 incluso em sua composição.

Visual Studio 2017 – Para utilizar o *Visual Studio* na empresa, precisamos da versão *Professional*. O *Visual Studio Professional* é licenciado por usuário. Cada usuário licenciado pode instalar e utilizar o software em vários dispositivos para conceber, desenvolver, testar e demonstrar os seus programas. O software incluído na licença é a versão atual, o Visual Studio Professional 2017.

SQL *Server* – Para o SQL *Server*, precisaremos da versão 2017 *Enterprise*. Essa versão do software tem um Licenciamento por volume, hospedagem.

Todos os softwares, licenças e pacotes listados nesse relatório são aplicados a máquinas do tipo workstation dentro de ambientes corporativos.

Faça valer a pena

1. As máquinas dos tipos cliente e servidor estão presentes em diversas aplicações e sistemas atuais, utilizados diariamente por boa parte dos usuários de tecnologia. Esses tipos de máquinas estão fortemente ligados à arquitetura de redes de computadores cliente-servidor, que é uma das mais populares arquiteturas utilizadas.

Considerando as características das máquinas dos tipos cliente e servidor, analise as afirmações abaixo e escolha a opção correta.
I – As máquinas do tipo servidor respondem às solicitações recebidas de máquinas clientes.
II – As máquinas clientes aguardam as respostas das máquinas servidores.
III – As máquinas dos tipos clientes e servidores exercem o mesmo papel em um sistema.
IV – Inicialmente, as máquinas do tipo servidor acessam aplicações nas máquinas do tipo clientes.
V – As máquinas servidor podem executar aplicações dos tipos web, banco de dados, autenticação, entre outros.

a) Somente a afirmação I está correta.
b) Somente as afirmações II e IV estão corretas.
c) Somente a afirmação I e V está correta.
d) Somente a afirmação I, II e IV está correta.
e) Somente as afirmações I, II e V estão corretas.

2. As máquinas do tipo *workstations*, conhecidas também como estações de trabalho, são computadores montados de maneira exclusiva para executar uma atividade que necessite de recursos em abundância. Máquinas desse tipo precisam ter um alto desempenho na execução de tarefas de cunho profissional.
Sobre as características de máquinas do tipo *workstation*, marque V para verdadeiro ou F para falso:
( ) Uma máquina *workstation* é utilizada para executar tarefas de cunho profissional que necessitem de recursos de hardware.
( ) As máquinas *workstation* tem como principal função responder a requisições de servidores.
( ) Tarefas como produção de desenho industrial, aplicação de softwares gráficos de modelagem 3D, edição de fotos, edição de vídeos, inteligência artificial e aplicações de Big Data são consideradas atividades profissionais exercidas por máquinas *workstation*.
( ) As máquinas *workstation* são equivalentes às máquinas cliente. O que as diferencia é que as máquinas *workstation* são projetadas para alto desempenho e execução de tarefas profissionais.

Assinale a alternativa que representa a sequência correta:
a) V - F - V - V
b) F - F - V - V
c) V - V - F - F
d) F - V - F - F
e) V - V - V - V

3. No que diz respeito ao software, servidores podem ser de diversos tipos. Dentre eles, podemos citar: Servidor web, Servidor de Autenticação, Servidor de E-mail, Servidor FTP e Servidor de Banco de dados. Cada um dos tipos apontados exerce uma função diferente em um sistema e, na maioria das vezes, mais de um tipo trabalha em conjunto.

Esse tipo de servidor é responsável pela maioria dos conteúdos solicitados por meio de um endereço, seja ele um IP ou uma URL. Suas respostas normalmente são páginas que podem ser exibidas via browser. Assinale a alternativa que corresponda ao tipo de servidor que mais combine com essas características.
a) Servidor de Autenticação.
b) Servidor de Banco de Dados.
c) Servidor FTP.
d) Servidor Web.
e) Servidor de E-mail.

Referências

COULOURIS, G. et al. **Sistemas Distribuídos**. Porto Alegre: Bookman, 2013.

ESTUDO NA WEB. **Entenda o Diagrama de Casos de Uso |** #7. 2 dez. 2015. Disponível em: https://youtu.be/xrcgbMQdM8Y?list=PLvSnBLKgRyHWINbSrLEEE6cen3ghoM_LG. Acesso em: 2 out. 2018.

ESTUDO NA WEB. **Entenda o Diagrama de Sequência |** #10. 2 nov. 2015. Disponível em: https://youtu.be/ypP6HQdDxYM?list=PLvSnBLKgRyHWINbSrLEEE6cen3ghoM_LG. Acesso em: 2 out. 2018.

ESTUDO NA WEB. **Entendendo o Diagrama de Implantação |** #12. 20 jan. 2018. Disponível em: https://youtu.be/P0wXFFsdMzI?list=PLvSnBLKgRyHWINbSrLEEE6cen3ghoM_LG. Acesso em: 02 out. 2018.

FILEZILLA. **Quick download links**. Disponível em: https://filezilla-project.org. Acesso em: 10 jan. 2019.

GREINER, R. **CAP Theorem:** Revisited. 14 ago. 2014. Disponível em: http://robertgreiner.com/2014/08/cap-theorem-revisited/. Acesso em: 25 mar. 2018.

MAIA, L. P. **Arquitetura de redes de computadores**. Rio de Janeiro: LTC, 2013.

YADAV, S. C. **An Introduction to Client/Server Computing**. New Delhi: New Age International, 2009.

MICROSOFT. **Como licenciar o SQL Server.** Disponível em: https://www.microsoft.com/pt-br/sql-server/sql-server-2017-pricing#ft2. Acesso em: 23 out. 2018.

MICROSOFT. **Microsoft 365 Enterprise**. Disponível em: https://www.microsoft.com/pt-br/microsoft-365/enterprise/home?rtc=1. Acesso em: 23 out. 2018.

MICROSOFT. **Preços e licenciamento para o Windows Server 2019**. Disponível em: https://www.microsoft.com/pt-br/cloud-platform/windows-server-pricing. Acesso em: 23 out. 2018.

MICROSOFT. **Visual Studio Professional 2017**. Disponível em: https://www.microsoft.com/pt-pt/p/visual-studio-professional-2017/dg7gmgf0dst5. Acesso em: 23 out. 2018.

ORACLE. **CORBA Technology and the Java™ Platform Standard Edition**. Disponível em: https://docs.oracle.com/javase/7/docs/technotes/guides/idl/corba.html. Acesso em: 8 jan. 2019.

ORACLE. **Getting Started Using Java™ RMI**. Disponível em: https://docs.oracle.com/javase/7/docs/technotes/guides/rmi/hello/hello-world.html. Acesso em: 8 jan. 2019.

ORACLE. **The Java EE 6 Tutorial**. Disponível em: https://docs.oracle.com/javaee/6/tutorial/doc/bnayl.html. Acesso em: 8 jan. 2019.

PETABRIDGE. **Building Networked .NET Applications with Akka.Remote**. Disponível em: https://petabridge.com/training/akka-remoting/. Acesso em: 8 jan. 2019.

TANENBAUM, A. S; STEEN, M. V. **Sistemas Distribuídos** - Princípios e Paradigmas. 2 ed. São Paulo: Pearson, 2008.

Unidade 3

Virtualização e Conteinerização

Convite ao estudo

Caro aluno, seja bem-vindo a mais uma unidade. Agora que já conhecemos os papéis que as diferentes máquinas podem assumir, vamos verificar, na prática, como instalar e configurar máquinas com diferentes propósitos. Você já parou para pensar em quantos sistemas operacionais diferentes existem? Será que há um único sistema que pode ser instalado em uma máquina do tipo cliente e em uma do tipo servidor? Além disso, qual é a melhor forma de testar e se familiarizar com esses diferentes sistemas operacionais? Nesta unidade, vamos entender conceitos muito importantes e como aplicá-los de maneira prática, para que você consiga responder às questões anteriores e muitas outras. Talvez você já tenha ouvido falar de máquinas virtuais, mas será que já ouviu sobre conteinerização? Ainda que sim, de maneira prática, como você pode conteinerizar uma aplicação web, por exemplo? Se respondeu não a alguma dessas perguntas, não se preocupe! A ideia aqui é mostrar que, caso você estude com afinco, pesquise e faça todas as atividades sugeridas, aprenderá a implementar e configurar contêineres para sistemas distribuídos.

Uma importante empresa do ramo de tecnologia que presta serviços a praticamente todos os bancos corporativos está com uma vaga aberta de trainee DevOp e, por isso, deu início a um processo seletivo. Interessado em ingressar nessa empresa, você está disputando a vaga e, na última etapa do processo seletivo, será preciso pôr seus conhecimentos à prova. Você será capaz de atingir todos os objetivos conforme a necessidade da empresa?

Na primeira seção desta unidade, você entenderá quais são os princípios e as melhores práticas relacionadas à ideia de virtualização. Já na segunda seção, será apresentado ao – relativamente novo – conceito de contêineres e entenderá o motivo de muitas empresas de grande porte, que possuem sistemas distribuídos altamente escaláveis, estarem migrando cada vez mais para esse tipo de solução. Por fim, você terá a oportunidade de colocar em prática conceitos importantes sobre conteinerização, por meio do uso de uma ferramenta largamente utilizada no mercado de trabalho.

Chegou a hora de prosseguir com os estudos dos sistemas distribuídos, o que vai lhe proporcionar conhecimentos e oportunidades no mercado de trabalho. Lembre-se de que, quanto mais você se dedicar, mais poderá aproveitar os ensinamentos transmitidos neste material.

Seção 3.1

Virtualização

Diálogo aberto

Caro estudante, você já deve ter ouvido falar de (talvez até utilizado) softwares como o VirtualBox e o VMWare. Nessa seção vamos entender, conceitualmente e de maneira prática, a importância desse tipo de tecnologia e quais problemas ela veio solucionar. Saiba que sua importância vai muito além do simples fato de testar ou "dar uma olhada" na nova versão de um sistema operacional. Na situação-problema à qual você será exposto, terá que criar uma máquina virtual com uma configuração previamente definida, de forma que será necessário entender os principais parâmetros que devem ser ajustados para tal. Caso não saiba como fazer isso, a máquina nem passará da fase inicial de *boot* do sistema.

Você está participando de um processo seletivo para uma oportunidade muito interessante e bem remunerada na área de DevOps da maior empresa nacional de portal de notícias, cujos clientes que consomem o conteúdo disponibilizado por essa empresa são bancos, em sua maioria. Na entrevista, além da Gerente de RH, também está participando o Coordenador de Infraestrutura, que será o seu futuro gestor.

Inicialmente, o Coordenador pede para você criar uma máquina virtual e testá-la: neste contexto, crie uma máquina virtual com o software VirtualBox. Esse tipo de procedimento é rotineiro nesta empresa que necessita de máquinas para execução de diversos serviços de tecnologia.

Utilize o seu conhecimento atrelado ao conteúdo da seção para solucionar a situação-problema apresentada acima.

Não pode faltar

Definição de virtualização

A virtualização se torna cada vez mais popular com o passar dos anos e é amplamente utilizada nos ambientes corporativos e, em alguns casos, até domesticamente. O grande objetivo da virtualização é fornecer uma versão virtual de tecnologias essenciais em computação, por exemplo, redes, armazenamento, hardware, entre outros. Além disso podemos também, virtualizar aplicações. Segundo Dawson e Wolf (2011, [s.p.]), a "virtualização desacopla as tarefas e a parte funcional das aplicações da infraestrutura física

necessária para seu funcionamento, permitindo uma flexibilidade e agilidade sem precedentes em termos de armazenamento, servidores e desktops".

Assimile

Atualmente, é possível fornecer uma versão virtual de tecnologias essenciais em computação. Por exemplo, para um site permanecer no ar por 24 horas, não precisamos ter um servidor web/aplicação físico mantendo seu funcionamento, podemos ter um servidor virtual ou máquina virtual, como são chamadas, tipo de servidor que substitui ou emula o funcionamento de um servidor físico.

Quando virtualizamos recurso de hardware, como memória RAM, processador, placas de vídeo, placa de rede, entre outros, temos uma máquina virtual que funciona com os recursos de hardware em formato virtual. Sabendo disso, podemos instalar um sistema operacional sobre outro sistema, ou seja, sobre nossa máquina física podemos ter várias máquinas virtuais. Esses recursos de hardware são representados por softwares na virtualização. A mesma coisa também pode ser feita com a rede: é possível criar uma infraestrutura lógica de rede sobre uma rede física. Nela podemos personalizar e configurar de formas diferentes da rede física, conforme nossas necessidades. Também é possível ter em nossa residência uma rede física "A" composta pelas redes lógicas "B" e "C".

Quando utilizamos virtualização, representamos os dispositivos físicos por meio de entidades de software, assim, nossos servidores e *workstations* se tornam o que chamamos de máquinas virtuais, ou VMs. A parte de armazenamento de dados é conhecida como *Software Defined Storage* (SDS), ou armazenamento definido por software. Já a parte de rede é chamada de *Software Defined Networking* (SDN), ou rede definida por software. Unindo todos esses elementos com um conjunto de máquinas virtuais, temos um *Software Defined Data Center* (SDDC), ou data center definido por software.

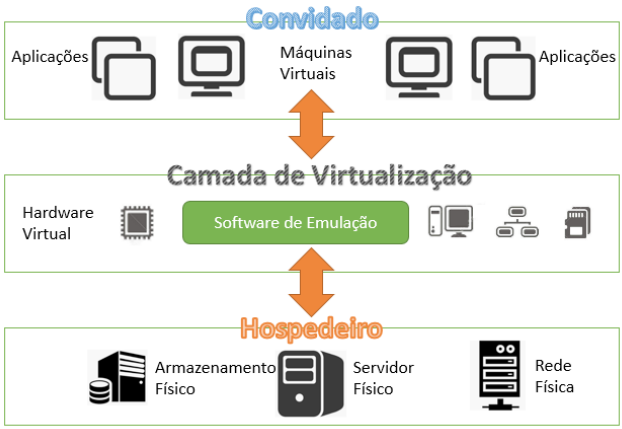
Portanto, as máquinas virtuais são abstrações de hardware de computadores que permitem que uma única máquina física aja como se fosse várias máquinas diferentes. Assim, podemos ter vários sistemas operacionais distintos rodando em uma só máquina. A virtualização contém três componentes principais, conforme o Portal Redhat (2018):

- Hospedeiro: como chamamos a máquina física em que existem máquinas virtuais.
- Convidado: como são chamadas as máquinas virtuais ou computadores virtualizados.

- Camada de virtualização: o software que permite criar sistemas convidados sobre sistemas hospedeiros.

Podemos observar na Figura 3.1 a interação dos três componentes principais da virtualização e exemplos de como são compostos:

Figura 3.1 | Elementos de máquinas virtuais



Fonte: elaborada pelo autor.

Para o usuário final, não faz diferença se a máquina acessada é física ou virtual, pois as duas funcionam da mesma forma, o que acaba sendo imperceptível. Nesse cenário de virtualização podemos ter em um mesmo servidor uma máquina virtual com o sistema operacional Windows Server, uma máquina virtual com Linux e uma máquina com FreeBSD, por exemplo. Os principais fatores que levam à utilização de virtualização são:

- Diminuição de espaço físico: muitas vezes, o ambiente corporativo não tem espaço físico para suportar servidores, com todos requisitos necessários, como a refrigeração adequada para eles.
- Rapidez na implantação: máquinas virtuais são implantadas mais rapidamente do que máquinas físicas.
- Redução de custos administrativos: os custos administrativos para se manter uma máquina física são bem maiores do que os custos referentes às máquinas virtuais.
- Economia de energia elétrica: como podemos ter várias máquinas virtuais funcionando sobre apenas uma máquina física, consequentemente, economizaremos energia, tendo menos máquinas alimentadas.

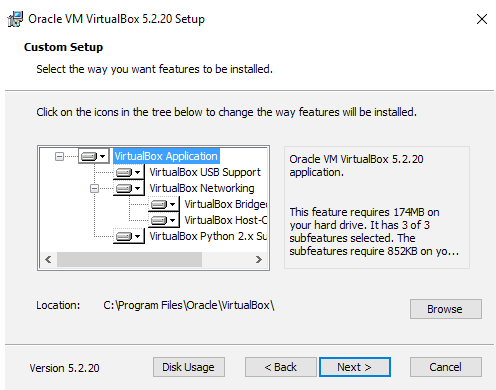
- Aproveitamento da capacidade de computação e performance: é possível aproveitar melhor os recursos de um servidor físico dividindo-os em várias máquinas virtuais.

VirtualBox

Existem vários softwares nos quais é possível criar e administrar máquinas virtuais, e os mais populares são VirtualBox, VM Ware e Hyper-V Microsoft. Nesta seção vamos utilizar o VirtualBox©, fabricado pela Oracle®. Esse software tem uma licença do tipo *GNU General Public License* (Licença Pública Geral GNU), GNU GPL ou simplesmente GPL. A licença GPL permite a utilização e o estudo do software de maneira livre para quaisquer fins.

O download do VirtualBox pode ser realizado no site do VirtualBox, escolhendo a versão adequada para o seu sistema operacional. No mesmo site há uma extensa documentação que nos auxilia com todo processo de manipulação da ferramenta, assim como informações sobre a tecnologia de virtualização, disponível em: https://www.virtualbox.org/manual/UserManual.html. Acesso em: 6 fev. 2019. Após o download do instalador, devemos selecionar a opção *Next*. Agora devemos escolher quais elementos da instalação queremos e qual será o local de instalação. No nosso caso deixaremos tudo conforme o padrão. Lembrando que você deve ter 175MB livres em seu HD para a instalação do VirtualBox. Após essas configurações devemos selecionar "Next", conforme a Figura 3.2:

Figura 3.2 | Instalando o VirtualBox



Fonte: captura de tela do VirtualBox.

Agora devemos escolher opções referentes a atalhos do VirtualBox e selecionar *Next* novamente. Feito isso, é necessário selecionar *Install* para iniciar nossa instalação. Agora devemos aguardar alguns minutos, o tempo de instalação varia de máquina para máquina. Ao término, é preciso deixar marcada a opção que abre o Virtualbox após a instalação e selecionar *Finish*. Podemos observar na Figura 3.3 a interface do Oracle VM VirtualBox, em que temos várias opções. A opção *Novo* é a que permite criar nossas máquinas virtuais. Com elas são habilitadas as opções de *Configurações* sobre a máquina selecionada: *Descartar* e *Iniciar*.

Figura 3.3 | Tela inicial do Virtualbox

Fonte: captura de tela do Virtualbox.

Dentro da opção *Novo* definimos todas as configurações de nossa máquina virtual, como seu nome, sistema operacional, quantidade de memória RAM, capacidade de disco rígido, processamento, entre outras.

O papel da virtualização em sistemas distribuídos

Dois tipos de virtualização são muito úteis no contexto de sistemas distribuídos, conforme Coulouris et al. (2013):

- Virtualização de redes;
- Virtualização de sistemas.

Esses autores observam, muito adequadamente, que a vantagem da criação e da utilização de redes virtuais advém do fato de que uma rede virtual específica para um determinado tipo de aplicação pode ser criada sobre uma rede física real, de forma que a virtual possa ser otimizada para aquela aplicação em particular, sem a necessidade de alterar as características da rede física.

Exemplificando

Imagine que você está desenvolvendo um sistema distribuído de um serviço de *streaming* de vídeo que, obviamente, é composto por vários elementos, como banco de dados, servidor web, servidor de e-mail,

servidor de autenticação etc. Suponha que o arquiteto de sistemas da empresa tenha optado por utilizar dois bancos de dados diferentes, um para armazenar informações gerais, como dados dos assinantes, datas de vencimento de assinaturas etc., do tipo SQL, e outro para armazenar os vídeos em si, com características mais adequadas para otimizar a troca de informações, do tipo NoSQL. Esses dois bancos de dados não precisam (e, por questões de segurança, nem devem) saber da existência um do outro. Para atingir esse objetivo, tipicamente as empresas criam duas redes virtuais dedicadas, de forma que, além de estarem isoladas entre si, podem ser otimizadas de acordo com a natureza do banco de dados, prevalecendo a comunicação de segmentos UDP para o banco de dados NoSQL, por exemplo, em detrimento dos segmentos TCP.

Para Coulouris et al. (2013), a virtualização de sistemas é uma alternativa interessante por permitir emular o hardware de uma máquina física, permitindo, assim, que várias máquinas virtuais, cada uma com um sistema operacional, coexistam e se comuniquem. Os autores ainda salientam que a principal vantagem da virtualização de sistemas está no fato de que as aplicações já escritas e validadas, que dependem de um sistema operacional em específico e que necessitam se comunicar e interagir com outra aplicação em um sistema operacional diferente, podem assim fazê-lo, através da virtualização dos sistemas operacionais, sem a necessidade de que a aplicação seja reescrita ou recompilada.

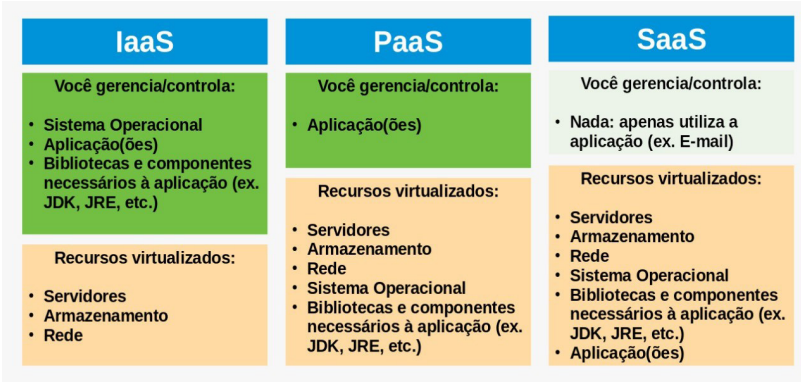
Reflita

Pense na seguinte situação: na empresa em que você trabalha, o sistema de planejamento de recursos, conhecido simplesmente por ERP (do inglês *Enterprise Resource Planning*), utiliza um módulo de controle de estoque escrito pelo próprio time de desenvolvedores da companhia, na linguagem Asp.NET que, por sua vez, utiliza funcionalidades específicas da plataforma Windows, não sendo 100% compatível com alternativas como o .NET *Core* (recentemente aberto para a comunidade). Para economizar nos gastos e, principalmente, aumentar a disponibilidade do sistema ERP utilizado pelos colaboradores da empresa, suas filiais e seus representantes comerciais, o Diretor Executivo, ou CEO (do inglês C*hief Executive Officer*), decide migrar para uma solução em nuvem, de algum provedor de *cloud computing* de mercado. Por questões financeiras, o Diretor de TI, ou CIO (do inglês *Chief Information Officer*), opta por utilizar um Sistema Operacional GNU/Linux. Como fazer para adaptar o módulo de controle de estoque? Reescrever o código seria uma opção, mas os desenvolvedores que fizeram esse

módulo já não trabalham mais na empresa, e o código é muito extenso e complexo, o que significa que sua reescrita impactaria em um aumento significativo de tempo até que o "novo" ERP esteja disponível. Qual seria sua solução para esse problema?

Se você já utilizou ou leu a respeito de computação em nuvem, deve saber que, independentemente do tipo de serviço que você contrata e do provedor desse serviço, você já estará utilizando a virtualização em algum nível, e esses serviços são tipicamente categorizados como IaaS (do inglês *Infrastructure as a Service*), PaaS (do inglês, *Platform as a Service*) e SaaS (do inglês, *Software as a Service*). Para entender melhor essa ideia, veja a Figura 3.4.

Figura 3.4 | Serviços em nuvem e níveis de virtualização



Fonte: elaborada pelo autor.

Arquitetura de virtualização

A maioria das pessoas, quando ouve falar de virtualização, pensa em um sistema operacional "dentro" de um software como, por exemplo, o VirtualBox da Oracle, instalado em uma máquina física obviamente com um sistema operacional instalado. Partindo desse contexto, podemos destacar algumas características: esse sistema operacional instalado "dentro" da máquina física refere-se a uma máquina virtual, pois é similar à sua máquina física, porém, puramente emulada via software. O software que permite emular uma máquina física é, de maneira genérica, chamado de *hypervisor* e é responsável por desacoplar a máquina física da virtual, bem como alocar os recursos da máquina física, de acordo com a necessidade da máquina virtual, conforme artigo da VMWare, uma das empresas mais conhecidas na área de virtualização.

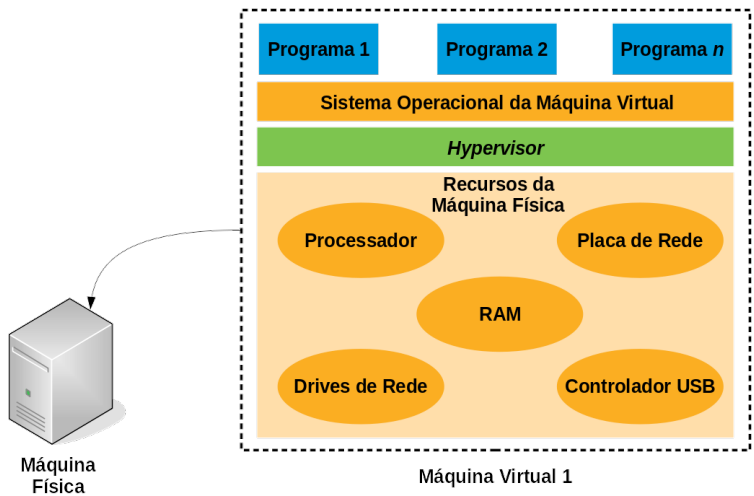
Exemplificando

Exemplos populares de *hypervisors* são o VMWare Player, da VMWare e o VirtualBox, da Oracle. Existem várias outras opções de *hypervisors* disponíveis no mercado, não tão populares, mas ainda assim bastante utilizadas, como o caso do QEMU (abreviação para *Quick Emulator*), que é o *hypervisor* utilizado pelo Android Studio (IDE oficial da Google para desenvolvimento de aplicativos Android), para emular o sistema operacional Android e poder testar os aplicativos em máquinas físicas com sistemas operacionais MS-Windows, GNU/Linux e macOS, conforme artigo do portal de desenvolvedores da Google:

ANDROID Studio. **Start the emulator from the command line**. Developers. Última atualização em: 2019.

Podemos instalar várias máquinas virtuais em uma única máquina física, limitando-se, é claro, às capacidades de processamento, RAM e armazenamento da máquina física. As máquinas físicas são tipicamente chamadas de *hosts* (hospedeiros), e as máquinas virtuais são tipicamente chamadas de *guests* (convidados), embora não seja tão comum a utilização dos nomes traduzidos para o nosso idioma. A Figura 3.5 ilustra esses elementos.

Figura 3.5 | Elementos de máquinas virtuais



Fonte: elaborada pelo autor.

Pesquise mais

- Você pode fixar seu conhecimento acessando a documentação do VirtualBox, que auxilia em todo processo de manipulação da ferramenta, assim como fornece informações sobre a tecnologia de virtualização.
- ORACLE® VM VirtualBox®. **User Manual**. VirtualBox, [s.d.].
- Você pode fixar seu conhecimento assistindo ao vídeo que fala sobre conceitos de máquina virtual, conforme estudado nesta seção.
- DOMÍNIO DA TECNOLOGIA. **Máquinas Virtuais #1 - O que é uma máquina virtual?** 23 jul. 2015.
- Você pode fixar seu conhecimento assistindo ao vídeo que mostra a criação de uma máquina virtual utilizando o VirtualBox, conforme estudado nesta seção.
- ARPHANET TUTORIAIS. **Como criar uma Máquina Virtual no VirtualBOX - Instalar e configurar com Windows 7**. 15 mai. 2013.

Assim, finalizamos mais uma seção no estudo dos sistemas distribuídos. Esperamos que o conhecimento adquirido seja de grande importância para seu crescimento profissional. Bons estudos e até a próxima seção!

Sem medo de errar

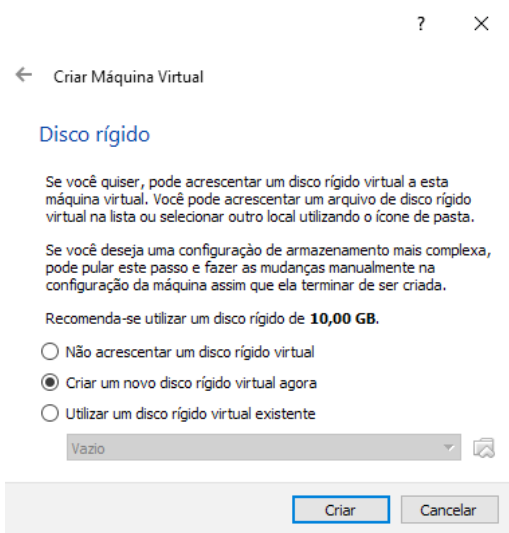
Você está participando de um processo seletivo para uma oportunidade como trainee na área de DevOps da maior empresa nacional de portal de notícias, cujos clientes que consomem o conteúdo disponibilizado por essa empresa são bancos, em sua maioria. Na entrevista, você será submetido a algumas atividades práticas sobre virtualização, definidas pelo Coordenador de Infraestrutura. Sua primeira atividade é criar uma máquina virtual. Crie uma máquina virtual com o software VirtualBox, com sistema operacional GNU/Linux, distribuição Ubuntu Desktop, nas configurações padrão, e acesse o website http://www.lovemondays.com.br, a partir da máquina virtual recém-criada, utilizando o navegador Firefox, da Mozilla. A primeira coisa que devemos fazer é realizar o download do sistema operacional Ubuntu na versão Desktop que foi solicitada, pode ser feito no site do Ubuntu, disponível em: https://www.ubuntu.com/download/desktop. Acesso em: 7 fev. 2019. Agora, no VirtualBox, devemos ir até a opção *Novo*, conforme a Figura 3.6:

Figura 3.6 | Criando uma máquina virtual

Fonte: captura de tela do VirtualBox, elaborada pelo autor.

Agora, devemos definir um nome para nossa máquina virtual, o tipo de sistema operacional e a versão. É necessário também estabelecer a quantidade de memória RAM que será usada pela máquina virtual. O próximo passo é selecionar a criação de um novo disco virtual. Caso existam discos virtuais prontos, também podemos utilizá-los. Na Figura 3.7 criamos um novo:

Figura 3.7 | Criando um disco virtual



Fonte: elaborada pelo autor.

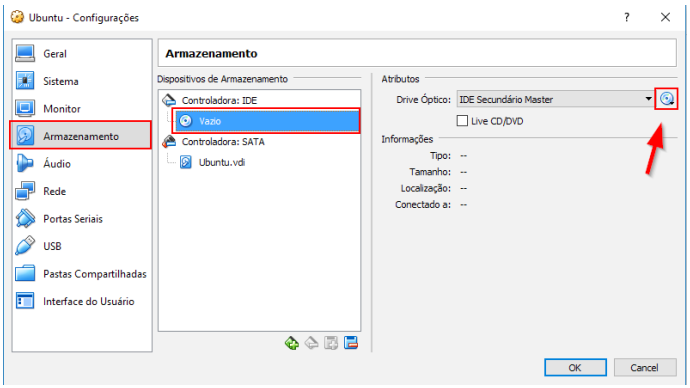
Agora que as configurações estão definidas e a máquina virtual já aparece na lista, vamos até a opção *Configurações*, representada pelo ícone de uma engrenagem, para apontar o caminho de nossa imagem (iso) que contém o sistema operacional. Podemos observar na Figura 3.8 que a opção *Configurações* está habilitada:

Figura 3.8 | Definindo as configurações da máquina virtual

Fonte: captura de tela do VirtualBox, elaborada pelo autor.

A Figura 3.9 mostra como apontamos o caminho da imagem de instalação do sistema operacional. Para isso ocorrer, devemos criar um leitor de CD virtual e indicar a imagem como se fosse um CD-ROM. Acessamos o menu *Armazenamento*, depois *Vazio* e, então, o ícone de um novo CD para localizar nossa imagem.

Figura 3.9 | Caminho da imagem de instalação do sistema operacional



Fonte: captura de tela do Ubuntu, elaborada pelo autor.

Após a seleção, o local em que estava escrito *Vazio* passa ter o nome da imagem do Ubuntu. Depois do apontamento do caminho da imagem, devemos pressionar o botão OK e, então, o botão *Iniciar*, representado pelo ícone de uma seta na cor verde. Feito isso, será aberta uma nova janela de execução da máquina virtual. Agora, devemos pressionar a tecla *Enter* do teclado para iniciar nossa máquina via CD, assim, teremos a opção de instalar nossa máquina, seguindo as etapas de instalação até a conclusão. Depois de finalizar a instalação, podemos ver a execução da máquina virtual. Agora, só resta ir até o site do Mozilla Firefox, realizar o download e a instalação e acessar o site *lovemondays*, conforme solicitado.

Avançando na prática

Instalação de VMs Ubuntu Server para ambiente web

Descrição da situação-problema

Você faz parte da equipe de infraestrutura de uma empresa, e foi solicitada a criação de um ambiente com dez máquinas virtuais com sistema operacional GNU/Linux e distribuição Ubuntu Server. Essas máquinas serão responsáveis por manter uma aplicação web no ar. Sua parte nesse projeto é criar a primeira máquina para que o restante da equipe "clone" as demais.

Resolução da situação-problema

A primeira coisa a se fazer é realizar o download do sistema operacional Ubuntu na versão Server, pelo site do Ubuntu, disponível em: https://www.ubuntu.com/download/server. Acesso em: 7 fev. 2019.

- No VirtualBox, vá ir até a opção *Novo*.
- Defina um nome para a máquina virtual, o tipo de sistema operacional e a versão.
- É preciso estabelecer a quantidade de memória RAM que será utilizada pela máquina virtual.
- Agora, deve-se selecionar a criação de um novo disco virtual ; caso existam discos virtuais prontos, é possível utilizá-los.
- Em seguida, defina o tamanho do HD da máquina virtual.

- Agora que as configurações foram efetuadas, a máquina virtual já aparece na lista de máquinas. Vá até a opção *Configurações*, representada pelo ícone de uma engrenagem, para apontar o caminho de nossa imagem (iso) que contém o sistema operacional.
- Aponte o caminho da imagem de instalação do sistema operacional. Para isso, é necessário criar um leitor de CD virtual e indicar a imagem como se fosse um CD-ROM.
- Acesse o menu *Armazenamento*, depois *Vazio* e, então, o ícone de um novo CD para localizar a imagem.
- Após a seleção, o local em que estava escrito *Vazio* passa ter o nome da imagem do Ubuntu.
- Depois do apontamento do caminho da imagem, pressione o botão OK e, em seguida, o botão *Iniciar*, representado pelo ícone de uma seta na cor verde. Feito isso, será aberta uma nova janela de execução da máquina virtual.
- Pressione a tecla *Enter* do teclado para iniciar a máquina via CD. Assim, tem-se a opção de instalar a máquina, seguindo as etapas de instalação até a conclusão.
- Depois de finalizar a instalação, é possível ver a execução da máquina virtual.

Faça valer a pena

1. A virtualização está dividida em três principais componentes: hospedeiros, camada de virtualização e convidados. Cada um tem o papel fundamental em um processo de virtualização, e entender a diferença entre eles é essencial.

Como são chamadas as máquinas virtuais, ou computadores virtualizados?

a) Hospedeiros.
b) Hosts.
c) Camada de Virtualização.
d) Convidados.
e) Software de Máquina Virtual.

2. Existem muitos fatores importantes que podem ser levados em consideração como vantagens para se utilizar a tecnologia de virtualização em um ambiente

composto por vários servidores. Na atualidade, muitas empresas optam por utilizar ambientes virtualizados devido à vasta lista de vantagens sobre ambientes físicos.

Sobre as características de virtualização, marque V para verdadeiro ou F para falso:

( ) Diminuição de espaço físico.
( ) Demanda mais tempo para implantação.
( ) Economia de energia elétrica.
( ) Melhor aproveitamento da capacidade de computação e da performance.

Assinale a alternativa que representa a sequência CORRETA:

a) V - V - V - V.
b) V - F - V - V.
c) V - F - V - F.
d) V - F - F - F.
e) V - V - F - V.

3. Com o software Oracle VM VirtualBox, podemos criar máquinas virtuais. Esse software funciona como uma camada de virtualização entre as máquinas físicas e virtuais, e nele é possível simular vários recursos de hardware de forma virtual para utilização de máquinas virtuais.

Ao criar uma nova máquina virtual, deseja-se instalar o sistema operacional X, que está em uma imagem (iso) na pasta de documentos do computador físico. Para isso, é necessário criar um CD-ROM ou disco virtual com o caminho dessa imagem, para que ele seja reconhecido dentro da máquina virtual. Em qual menu de Configurações devemos fazer essa tarefa?

a) Discos e CDs.
b) USB.
c) Armazenamento.
d) Pastas Compartilhadas.
e) Redes.

Seção 3.2

Conteinerização

Diálogo aberto

Caro aluno, o desenvolvimento de um software, seja ele desktop, web ou um app para smartphone, envolve a utilização de diversos recursos, por exemplo, um sistema gerenciador de banco de dados, um (ou mais) ambiente integrado de desenvolvimento, uma (ou mais) linguagem de programação etc. Ao fazer suas escolhas e implementar a solução, o próximo passo é disponibilizá-la, mas, nesse momento, muitos problemas de incompatibilidade podem ocorrer. Uma frase típica é "mas no meu computador funcionou!". Será que existem soluções para evitar esse tipo de problema? Existem e começaremos a explorá-las!

Nesta seção vamos ter a oportunidade de conhecer uma tecnologia muito requisitada no mercado de trabalho atualmente: os contêineres. Você já ouviu falar de uma das implementações mais utilizadas de contêineres no mundo, o Docker? Se não, aproveite a oportunidade. Sabia que há várias vagas de emprego requisitando conhecimentos dessa implementação aqui no Brasil e mundo afora? Não perca o foco!

Você se lembra da sua entrevista para a tão cobiçada vaga na área de DevOps? Avançando mais uma etapa, o coordenador pergunta se, na sua opinião, existem tecnologias similares, porém, mais eficientes que a virtualização, para rodar aplicações sem interferir no sistema operacional da máquina física, e você logo se lembra dos contêineres. Sendo assim, deverá criar um relatório técnico indicando o uso de uma tecnologia atual que traga vantagens e diminua o consumo de recursos para a criação de um novo ambiente que execute um sistema ERP que será implementado daqui a alguns meses na empresa. Os ERPs são sistemas integrados de gestão empresarial, que interligam todos os dados e processos de uma empresa dentro do sistema. Portanto é um dos mais importantes para um bom funcionamento de uma companhia, o que quer dizer que, quanto melhor for executado, mais pontos o profissional que planejou sua execução vai ganhar com os seus superiores. Então, chegou a hora de impressionar o coordenador em busca da vaga que você almeja, está pronto?

Definição de conteinerização

Uma das tecnologias mais populares que temos atualmente é o uso de contêineres para a execução de sistemas dos mais variados tipos. Isso ocorre devido à facilidade e à flexibilidade que advêm do uso dos mesmos. O contêiner funciona como uma tecnologia que dá o suporte para o funcionamento de uma aplicação e pode ser considerado a emulação de nossa aplicação. Quando a aplicação é executada através de um contêiner, ela tem todas as bibliotecas e os elementos necessários para o funcionamento disponíveis dentro do contêiner. Uma maneira simples para entender o que são os chamados contêineres é imaginar que eles permitem a criação de ambientes virtuais isolados e independentes para serem utilizados por aplicações, similar ao resultado do uso de máquinas virtuais. Entretanto, um grande diferencial está no fato de que os contêineres são mais leves que as máquinas virtuais, por possuírem uma arquitetura mais otimizada.

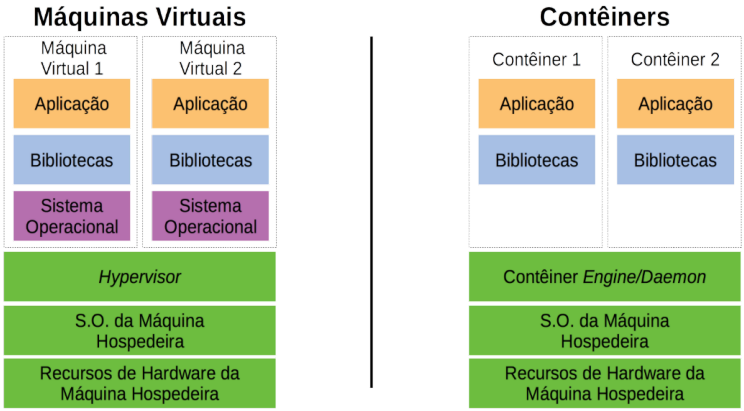
Assimile

O container traz muitas facilidades e é considerado uma das principais tendências de TI. Sua utilização simplifica a aplicação da metodologia DevOps e facilita o desenvolvimento. Grandes empresas, como a Google, usam essa tecnologia.

Se fosse necessário desenvolver um sistema em linguagem C para o cadastro de produtos do estoque de uma loja de varejo, poderíamos criar um contêiner que tivesse todas as bibliotecas essenciais para o funcionamento do sistema e, assim, nossa aplicação seria virtualizada através de um contêiner, sem a necessidade de um sistema operacional, pois, neste caso, já precisaríamos de uma máquina virtual.

Conforme pode ser visto na Figura 3.10, uma grande vantagem de contêineres em relação às máquinas virtuais é que não há, obrigatoriamente, a necessidade de instalar um sistema operacional completo, visto que as plataformas de conteinerização aproveitam bibliotecas compartilhadas com o sistema operacional hospedeiro. Por essa razão, os contêineres ocupam menos espaço em disco e consomem menos RAM e processamento que as máquinas virtuais e, assim, possibilita a utilização de mais contêineres em uma mesma máquina física, favorecendo o uso de uma arquitetura mais modular para as aplicações.

Figura 3.10 | Comparação das arquiteturas de máquinas virtuais ***versus*** contêineres



Fonte: elaborada pelo autor.

Reflita

Agora que conhecemos algumas vantagens dos contêineres em relação às máquinas virtuais, quais benefícios uma empresa que executa aplicações em máquinas virtuais poderia adquirir migrando a execução de suas aplicações para contêineres? Você acredita que seria benéfica essa migração?

Duas das principais características a favor da conteinerização são o baixo acoplamento entre os contêineres e a facilidade de migração entre provedores de *cloud computing*. Ambas se devem ao fato de que a ideia do contêiner é "empacotar" a sua aplicação em um módulo que é facilmente instalado em qualquer sistema operacional que suporte o uso de contêineres (e os principais sistemas operacionais utilizados em servidores possuem esse suporte).

No lugar do *hypervisor*, quando tratamos de máquinas virtuais, temos os chamados *contêiner engines* (por vezes chamados de contêiner *daemons*). Existem várias implementações para esses *engines*, como o Docker, o LXD, o Rkt, o Mesos e o Windows Server Containers, mas o mais popular entre eles é o Docker, sobre o qual aprenderemos na próxima seção. Porém, para começar a ter ideia do que se trata, quando falamos do Docker, estamos na realidade falando de uma empresa – chamada Docker – que foi responsável pela popularização dos contêineres, por meio de eventos e divulgação de material técnico. Essa companhia criou sua própria implementação que, coincidentemente, é chamada de Docker, cuja instalação, configuração e gerenciamento (e, consequentemente, curva de aprendizado) é relativamente mais simples comparando com outras implementações (Docker, 2018).

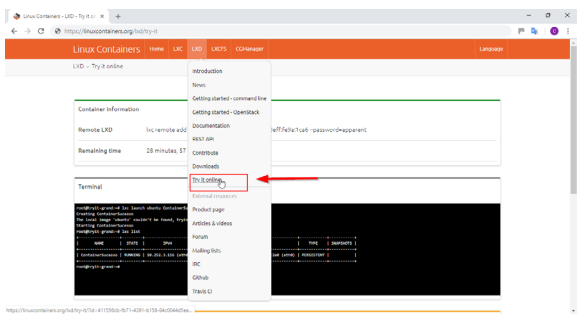
Existem implementações que podem ser consideradas contêineres de sistema, por exemplo, os Linux. Essas tecnologias têm um comportamento muito parecido ao de uma máquina virtual, mas, na verdade, são equivalentes a novas instâncias do sistema utilizando todas as técnicas disponíveis para isso. Os contêineres Linux funcionam compartilhando o kernel da máquina real onde estão em funcionamento. Uma limitação encontrada nos contêineres é que, quando estão compartilhando o kernel da máquina física ou até mesmo virtual em que estão hospedados e em execução, tratam-se de contêineres Linux rodando em máquinas Linux com o mesmo kernel do host, ou seja, mesmo que você substitua todas as bibliotecas e frameworks utilizados pelo seu contêiner, o kernel será sempre o mesmo do host que o hospeda.

Segundo Linux Container (2018), o projeto Linux container é o guarda-chuva por trás do Linux Containers, que possuem as gerações LXC, LXD e LXCFS. O LXC é uma interface de espaço de usuário para os recursos de contenção de kernel do Linux. Por meio de uma API poderosa e ferramentas simples, permite que usuários do Linux criem e gerenciem facilmente contêineres de sistema ou aplicativo. O LXD é um gerenciador de contêineres de próxima geração, que oferece uma experiência de usuário semelhante às máquinas virtuais, mas usando contêineres do Linux. Já o LXCFS é um sistema de arquivos simples do *userspace* projetado para contornar algumas limitações atuais do kernel do Linux.

Exemplificando

Podemos até mesmo testar um ambiente com contêineres Linux pelo Portal *Linux Containers*, disponível em: https://linuxcontainers.org/, acesso em: 7 fev. 2019. Esse portal disponibiliza toda a documentação com comandos utilizados no LXC e no LXD. Conforme mencionado, podemos praticar alguns comandos e testar ambos os ambientes através do menu *try it online*, como podemos observar na Figura 3.11:

Figura 3.11 | Área de treinamento do portal ***Linux Containers***



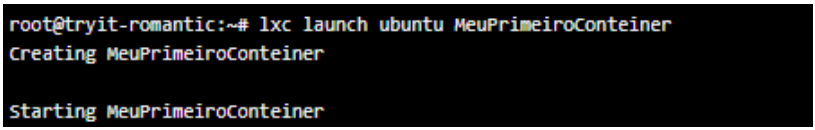
Fonte: Captura de tela do portal https://linuxcontainers.org/. Acesso em: 7 fev. 2019.

Dentro da opção de treinamentos, é possível usar o *Terminal*, disponível para executar alguns comandos e observar seu comportamento. Utilizamos o comando abaixo para criar o contêiner através da implementação Linux Containers:

```
lxc launch ubuntu MeuPrimeiroConteiner
```

Pode-se observar o contêiner com o nome *MeuPrimeiroConteiner* sendo criado no terminal e dando o retorno abaixo:

Figura 3.12 | Criando um contêiner por meio do portal ***Linux Containers***



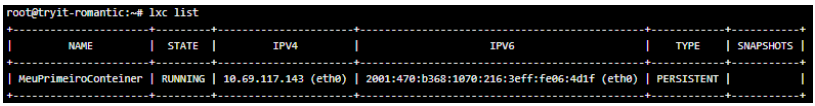
Fonte: Captura de tela do portal https://linuxcontainers.org/. Acesso em: 7 fev. 2019.

Feito isso, é possível consultar a lista de contêineres criados no terminal utilizando o comando abaixo:

```
lxc list
```

Pode-se observar na Figura 3.13 que o contêiner criado anteriormente já aparece como resultado em nossa listagem. Além disso, informações como o status da máquina e os números de seus respectivos IPV4 e IPV6 também aparecem, como podemos ver a seguir:

Figura 3.13 | Listando os contêineres criados através do portal ***Linux Containers***



Fonte: Captura de tela do portal https://linuxcontainers.org/. Acesso em: 7 fev. 2019.

É possível executar comandos dentro de um contêiner utilizando o comando abaixo:

```
lxc exec nomedocontainer – comando
```

Há opções de iniciar, parar e excluir contêineres usando os comandos:

```
lxc start
```

```
lxc stop
```

```
lxc delete
```

Além das informações do contêiner exibidas através da listagem que foi vista anteriormente, podemos observar uma série de informações sobre nosso contêiner como consumo de memória RAM, consumo de redes, entre outros dados. Para isso, utilizamos o comando de sintaxe a seguir:

```
lxc info nomedocontainer
```

Pesquise mais

Existe uma tendência no mercado em que as empresas migram suas aplicações que rodam em máquinas virtuais para aplicações que rodam em plataformas conteinerizadas, devido, principalmente, ao favorecimento da modularização. Para mais informações, recomenda-se a leitura do artigo do especialista em hospedagem na nuvem Ruslan Synytsky, co-fundador da Jelastic e consultor técnico da revista Forbes, que compara as diferenças entre máquinas virtuais e contêineres, demostrando as vantagens de migrar para plataformas 100% conteinerizadas.

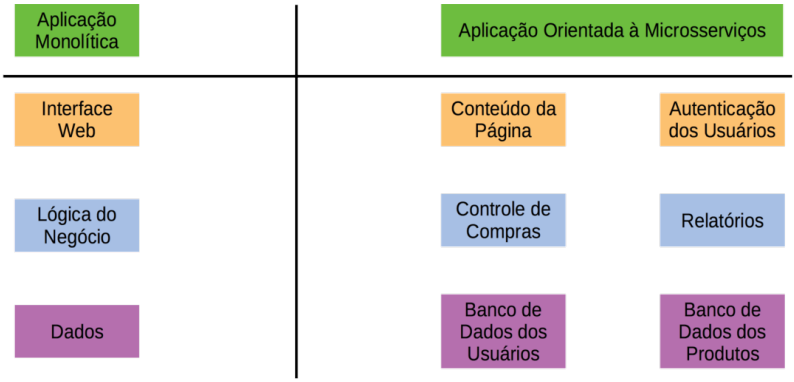
O papel da conteinerização em sistemas distribuídos

Os sistemas distribuídos fazem uso extensivo dos contêineres no contexto de microsserviços. A ideia dos microsserviços está associada a empresas que possuem sistemas altamente dinâmicos e ao termo modularidade. Um portal de notícias por exemplo, é composto por vários elementos, como um (ou mais) banco de dados, um (ou mais) *framework front-end* utilizado para desenvolver interfaces, *framework back-end* para desenvolver a parte dinâmica do sistema, *frameworks* para gerenciar mensagens entre servidores e clientes (por exemplo, o RabbitMQ) etc. Caso o sistema possua uma arquitetura monolítica, ou seja, uma forte dependência entre esses elementos, será muito difícil substituir alguns dos elementos citados anteriormente sem causar uma interrupção completa no sistema. Por outro lado, em uma arquitetura baseada em microsserviços, esses componentes têm um baixo acoplamento entre si, ou seja, o grau de dependência entre os componentes é baixo, de forma que, caso você tenha de fazer uma substituição, terá um impacto bem menor na indisponibilidade do seu sistema, e os contêineres serão artefatos fundamentais para atingir esse baixo acoplamento, pois cada um dos elementos pode ser criado e implantado em um contêiner separado.

Dentre as vantagens de um sistema distribuído baseado em microsserviços, podemos apontar que quanto menores são as partes, mais fácil entendê-las. Além disso, cada microsserviço pode ser executado e escalado de maneira concorrente e independente entre si. Outra vantagem decorrente

disso é que, como esses elementos possuem um baixo acoplamento entre si, um projeto de grande porte pode ser trabalhado de maneira razoavelmente independente entre as equipes de trabalho. A Figura 3.14 ilustra uma aplicação monolítica em relação a uma aplicação baseada em microsserviços.

Figura 3.14 | Exemplo de aplicações baseadas em microsserviços



Fonte: elaborada pelo autor.

Assimile

Atualmente, existe uma grande tendência de empresas migrarem de aplicações monolíticas para aplicações orientadas a microsserviços, por vantagens como escalabilidade e independência entre seus elementos (também conhecido como baixo acoplamento).

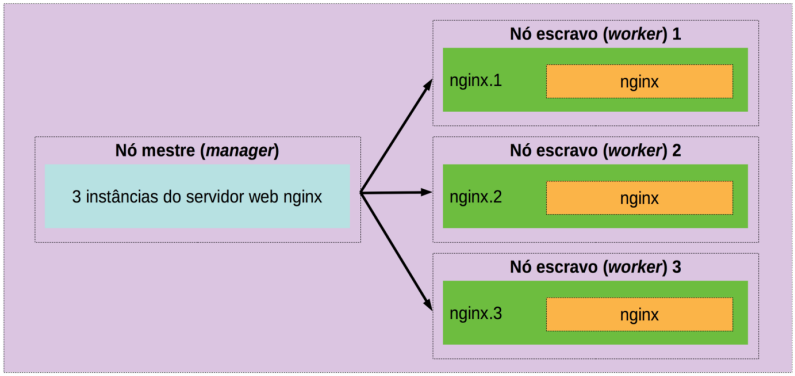
Como veremos na próxima seção (parte prática), apesar de a criação e o controle de um contêiner ser feita com simples comandos, podemos observar pela Figura 3.14 que uma aplicação é composta por não apenas um, mas vários microsserviços, cada um representando um ou mais contêineres. Gerenciar dezenas ou centenas de contêiner, de maneira isolada, pode ser uma tarefa muito trabalhosa, razão pela qual as empresas utilizam alguma ferramenta para criar, gerenciar e remover contêineres. Esse tipo de ferramenta é conhecido no mercado de trabalho como ferramenta de orquestração de contêineres. Acredita-se que a ideia esteja relacionada ao fato de que, em uma orquestra, um maestro coordena os músicos de maneira que o objetivo conjunto (obra tocada) seja o melhor possível. Nessa analogia, as ferramentas de orquestração realizam o mesmo papel do maestro, em que os músicos seriam os contêineres.

Atualmente, a ferramenta de orquestração mais popular e, portanto, mais solicitada no mercado de trabalho é o Kubernetes, da Google, que serve

para orquestrar contêineres criados com o Docker. Importante notar que o próprio *framework* do Docker possui uma ferramenta de orquestração nativa, instalada automaticamente com o Docker. Essa ferramenta é chamada de Swarm. Em alguns aspectos, inclusive pelo fato de ser uma ferramenta totalmente integrada com o Docker, o Swarm é a mais indicada para um primeiro contato com o conceito de orquestração de contêineres, razão pela qual essa será a ferramenta adotada nesse livro.

O Swarm foi integrado ao Docker a partir da versão 1.12.0, com o chamado *swarm mode*, em junho de 2016, conforme noticiado no blog oficial do Docker. A Figura 3.15 ajudará a entender os componentes do Swarm e como eles estão relacionados aos contêineres.

Figura 3.15 | Arquitetura do Swarm do Docker



Fonte: elaborada pelo autor.

O retângulo destacado em lilás representa o que o Docker chama de Swarm, que nada mais é que um *cluster* (conjunto de computadores interligados que funcionam como um grande sistema), formado por vários nós, que podem rodar uma aplicação – no exemplo, o servidor web Nginx – de maneira integrada e distribuída. Para que os nós possam estar integrados, de maneira que, caso uma instância do Nginx falhe por conta de um dos nós escravos estar indisponível, esta ser automaticamente instanciada em outro nó, é necessário que exista um nó mestre que faça essa gestão. Podem existir vários nós com a função de mestre do cluster (que o Docker chama de *manager*), para que, caso o nó mestre falhe, outro nó mestre assuma seu lugar. Os nós escravos (que o Docker chama de *worker*) rodam a quantidade de instâncias (frequentemente chamadas de réplicas) solicitadas pelo nó mestre e, para tal, é preciso que exista um contêiner "dentro" de cada nó escravo, o que está representado pelo retângulo laranja na Figura 3.15.

Pesquise mais

- Você pode aumentar seu conhecimento assistindo ao vídeo em que Jobert "RedRat", especialista em arquiteturas distribuídas, explica o porquê utilizar.
  LOCAWEB. TecnoPapo: porquê utilizar microsserviços, com Jobert "RedRat". 24 maio 2017.
- Recomenda-se também assistir a outro vídeo, em que Ramon Durães, consultor certificado MVP, fala sobre como o Docker e o Kubernetes se encaixam no contexto de aplicações baseadas em microsserviços.
  RAMON DURÃES. Tecnologia: Microservices, docker e Kubernetes. 1 dez. 2017.
- Outra sugestão é que você assine a newsletter oficial do Docker: é gratuito e semanalmente receberá novidades, poderá ler sobre entrevistas e acompanhar tutoriais sobre o Docker e tecnologias que se integram ao mesmo.
  DOCKER Blog (Portal). **Get the Latest Docker News by Email**.

Caro aluno, agora você já sabe o que é contêiner e logo mais estará apto para utilizar essa importante tecnologia! Bons estudos!

Sem medo de errar

Continuando o processo seletivo para aquela tão cobiçada vaga na área de DevOps, o coordenador o submete a um teste, no qual você deve criar um relatório para indicar uma tecnologia que otimize o desempenho de um sistema ERP, que logo será implementado na empresa.

Vamos auxiliá-lo com a resolução dessa situação-problema.

Depois de analisar o cenário de tecnologias atuais, foi decidido que criar um ambiente com contêineres é a melhor opção para impressionar o coordenador, devido à popularidade e às vantagens dessa tecnologia. Portanto, para justificar essa escolha, abaixo estão citadas as vantagens que o uso de contêiner deve trazer na execução de um sistema de ERP:

1. Como um dos principais fatores é a economia de recursos, é possível justificar a escolha apontando a melhor eficiência na execução do sistema e grande economia de recursos, ainda mais se a equipe de desenvolvimento utilizar a arquitetura de microsserviços.

2. Com o uso de contêineres, pode-se automatizar implantações e atualizações do novo sistema. Como o sistema de ERP é novo na empresa, é normal que tenha muitas atualizações até que chegue a

uma versão estável que satisfaça todas equipes, e o uso de contêineres vai facilitar muito as atualizações.

3. Garantia da escalabilidade do sistema ERP nos contêineres de maneira ágil e otimizada.
4. Capacidade de orquestrar contêineres em múltiplos hosts.
5. Conforme apontado na primeira vantagem, o uso de contêiner permite o uso do hardware de forma otimizada, ou seja, faz com que a redução do consumo de recursos seja evidente.

Através do apontamento destas vantagens, o coordenador terá a certeza de que você entende muito bem do que está falando e com certeza vai avançá-lo para a próxima etapa da entrevista!

Avançando na prática

Utilizando Linux contêiner

Descrição da situação-problema

A empresa na qual você trabalha está na vanguarda da tecnologia e, portanto, já utiliza tecnologia de contêiner para rodar as aplicações fornecidas a seus clientes. Entretanto, há uma reclamação de um dos grandes clientes da empresa: seu principal sistema de CRM (auxiliam na gestão do relacionamento com o cliente) está funcionando com alguns travamentos. Para começar a diagnosticar esse problema, foi solicitado que o estagiário fosse até o cliente e criasse dois contêineres utilizando a tecnologia Linux Contêiner, recolhesse as informações de consumo de memória de ambos e, após isso, parasse a execução e excluísse ambos no ambiente de produção. Como o estagiário não conhece a tecnologia Linux Contêiner, foi solicitado que você criasse um tutorial passo a passo em uma plataforma de testes e encaminhasse para o estagiário.

Resolução da situação-problema

Como deverá mostrar os comandos que devem ser utilizados pelo estagiário no importante cliente, utilize a plataforma de Linux Contêiner online para exemplificar a utilização dos comandos, através do portal *Linux Containers* disponível em https://linuxcontainers.org/lxd/try-it/ (acesso em 8 fev. 2019). Como foi solicitada a criação de dois contêineres, deixe os comandos abaixo como exemplo para o estagiário:

```
lxc launch ubuntu ConteinerTeste1
```

```
lxc launch ubuntu ConteinerTeste2
```

Agora que já tem a criação dos contêineres, você pode consultar o consumo de memória através dos comandos:

```
lxc info ConteinerTeste1
```

```
lxc info ConteinerTeste2
```

É possível ver o parâmetro `Memory usage` como uma das saídas dos comandos na Figura 3.16 a seguir:

Figura 3.16 | Listando as informações dos contêineres através do portal *Linux Containers*

Fonte: captura de tela da área do usuário do Portal "Linux Containers".

Agora, você deve parar a execução dos contêineres e excluí-los do ambiente usando os comandos a seguir:

```
lxc stop ConteinerTeste1
```

```
lxc delete ConteinerTeste1
```

```
lxc stop ConteinerTeste2
```

```
lxc delete ConteinerTeste2
```

Feito isso, sua tarefa está finalizada!

Faça valer a pena

1. Existem algumas ferramentas, chamadas de ferramentas de "orquestração de contêineres", que facilitam e automatizam o gerenciamento de um conjunto de contêineres. Assim sendo, é de extrema importância a familiaridade com esse tipo de ferramenta.

Entre as alternativas abaixo, assinale aquela que apresenta somente ferramentas de orquestração de contêineres.

a) Docker e Kubernetes.
b) Mesos e Docker.
c) Mesos, Docker e Kubernetes.
d) Kubernetes e Swarm.
e) Docker e Swarm.

2. Os contêineres têm se popularizado cada vez mais devido a uma série de vantagens desse tipo de tecnologia, se comparados à tecnologia antecessor, de máquinas virtuais. Uma alternativa para implementação de contêiner é o Linux Contêiner, embora existam outras implementações.

De acordo com os comandos LXC do Linux Contêiner, assinale a alternativa que mostra corretamente a sintaxe para a criação de um novo contêiner ubuntu, com o nome `NovoConteiner`.

a) `lxc start NovoConteiner`
b) `lxc start ubuntu NovoConteiner`
c) `lxc launch ubuntu NovoConteiner`
d) `lxc create ubuntu NovoConteiner`
e) `lxc new NovoConteiner`

3. O Linux Contêiner LXC e LXD tem seu uso se tornando cada vez mais popular, tanto que as ferramentas de orquestração de contêineres dão suporte a esta implementação. Sendo assim, é importante que conheçamos alguns comandos básicos dessa implementação.

Em relação ao seguinte comando, assinale a alternativa que apresenta a sintaxe correta do comando que traz informações do contêiner como seu status de ligado ou desligado, endereço de IPV4, endereço de IPV6, entre outras informações. Vamos utilizar um contêiner chamado `Conteiner_Kro`.

a) `lxc info Conteiner_Kro`
b) `lxc list`
c) `lxc launch ubuntu Conteiner_Kro`
d) `lxc view infos Conteiner_Kro`
e) `lxc Conteiner_Kro status,ipv4,ipv6 show`

Seção 3.3

Simulando sistemas distribuídos com Docker

Diálogo aberto

Caro aluno, você já pensou de que maneira são orquestrados os serviços quando acessamos um website? Uma das principais plataformas de conteinerização utilizadas atualmente é o Docker, principal assunto desta seção. Aqui vamos trabalhar com a instalação do Docker no sistema operacional GNU/Linux Ubuntu e aprender alguns dos principais comandos que podem ser utilizados na plataforma Docker. Você já parou para pensar em quais comandos devem ser usados para que essa orquestração de serviços funcione adequadamente?

Quando falamos de sistemas distribuídos, é inevitável falar sobre virtualização e conteinerização. Na prática, essas tecnologias são frequentemente utilizadas por grandes empresas. Você concluiu seu curso e está participando de um processo seletivo para uma oportunidade como trainee na área de DevOps da maior empresa nacional de portal de notícias, cujos clientes que consomem o conteúdo disponibilizado por essa empresa são bancos, em sua maioria. Você está se saindo muito bem no processo seletivo e agora será submetido ao teste final, uma atividade prática com Docker.

O coordenador está gostando de seu desempenho nos testes e diz que, caso você consiga orquestrar o servidor web Apache em um *cluster* simples, a vaga será sua. Desta forma:

I. Crie um *cluster* com cinco réplicas do servidor web Apache utilizando o Docker Swarm;

II. Verifique em quais nós do *cluster* esse serviço está rodando;

III. Acesse a página de boas-vindas desse servidor Apache através do(s) endereço(s) IPv4 de cada nó onde esse serviço web estiver rodando.

Para completar o desafio, nessa seção você verá, em detalhes, como se utiliza o Docker, incluindo comandos específicos a serem utilizados, tanto para configuração, quanto para constatação de que o serviço de Apache está funcionando de maneira adequada. Ficou curioso? Vamos lá!

Não pode faltar

O Docker é uma famosa plataforma genérica de conteinerização. Conforme já estudamos, o conceito de conteinerização é parecido com virtualização, porém é considerado "mais leve". Contêineres são muito populares atualmente devido à facilidade e à flexibilidade que advêm de seu uso. Portanto, agora chegou a hora de colocar a mão na massa e aprender a utilizar essa famosa ferramenta.

Instalação do Docker

Vamos fazer a instalação do Docker em uma das distribuições populares GNU/Linux, o sistema operacional Ubuntu, cuja versão utilizada foi a 14.04.5 LTS. Para isso, devemos seguir os passos a seguir. Todo o procedimento deve ser feito com um usuário com permissões de administrador. Nesse caso, utilizaremos o *root* através do comando `sudo su`.

Você também pode instalar o Docker no sistema operacional Windows, fazendo o download da versão mais atual. O processo de instalação é bem simples, é preciso somente avançar as etapas do instalador.

Assimile

Todos os procedimentos utilizados para instalação foram retirados da documentação oficial do Docker. As ferramentas de tecnologia atualizam-se a tal velocidade que somente acompanhando a documentação oficial é possível manter-se atualizado. Portanto, como um profissional engajado, procure sempre pelas fontes oficiais das ferramentas.

Antes de instalar a ferramenta, devemos remover versões anteriores do Docker que possam estar instaladas, usando o comando:

```
sudo apt-get remove docker docker-engine docker.io
```

Caso não tenha nenhuma versão instalada, será exibida a mensagem que foi impossível encontrar o pacote *docker-engine*.

Antes de instalar o Docker CE pela primeira vez em uma nova máquina host, você precisa configurar o repositório do Docker, atualizando os pacotes de sua máquina. Depois, você pode instalar e atualizar o Docker do repositório. Portanto, execute os comandos que vão atualizar a máquina:

```
sudo apt-get update
```

O comando acima tem o objetivo de realizar uma atualização de pacotes do Ubuntu. Na execução desse comando, o tempo pode variar de acordo com os pacotes que precisam ser atualizados, da velocidade da máquina e da conexão com a internet.

Através do comando abaixo atualizamos os pacotes necessários para a instalação do Docker. Para obter uma instalação bem-sucedida, é bom ter os programas utilizados nela em suas últimas versões:

```
sudo apt-get install \ apt-transport-https \ ca-
certificates \ curl \    software-properties-common
```

Após atualizar os repositórios, vamos adicionar o repositório de instalação do Docker, usando o seguinte comando:

```
curl -fsSL https://download.docker.com/linux/
ubuntu/gpg | sudo apt-key add -
```

Nesse comando, estamos fazendo o apontamento do caminho de instalação oficial do Docker que o Ubuntu deve acessar quando queremos efetuar a instalação. Após o apontamento da URL indicando o local para download o Docker para Ubuntu, será possível adicionar o repositório no próximo comando:

```
sudo add-apt-repository \ "deb [arch=amd64]
https://download.docker.com/linux/ubuntu \  $(lsb_
release -cs) \   stable"
```

O comando utiliza a permissão considerada `admin` nos sistemas operacionais da família Linux, através da palavra `sudo`. Depois de usar a permissão de usuário do `sudo`, utilizamos o `add-apt-repository` para adicionar o repositório, que pode ser comparado a uma loja de aplicativos, responsável pelo download do Docker na versão Ubuntu. O restante do comando é o caminho do repositório.

Após adicionar o repositório para download do Docker, como mostra a Figura 3.17, devemos mais uma vez atualizar o `apt-get`, conforme o comando seguinte para aplicar as alterações:

```
sudo apt-get update
```

Figura 3.17 | Saída do comando que adiciona o repositório de instalação do Docker

```
root@Caigol-note:/# sudo add-apt-repository \
>    "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
>    $(lsb_release -cs) \
>    stable"
root@Caigol-note:/# sudo apt-get update
```

Fonte: captura de tela elaborada pelo autor.

Agora vamos utilizar o comando de instalação do Docker. Lembrando que, para que esse comando funcione, devemos seguir as etapas de apontar o repositório em que o Docker está disponível e adicionar esse repositório em nossa lista. No comando utilizamos o `sudo` para usar a permissão `admin` do sistema e o `apt-get install` para a fazer a instalação, por último o nome do programa que vamos instalar, no caso, o Docker (`docker-ce`).

```
sudo apt-get install docker-ce
```

Com todas as configurações feitas para atualizar os pacotes necessários e adicionar o repositório que contém o Docker, agora é possível fazer a instalação. Veja na Figura 3.18 o comando sendo aplicado no terminal e também o início da sua execução.

Figura 3.18 | Saída do comando que faz a instalação do Docker

```
root@Caigol-note: /
root@Caigol-note:/# sudo apt-get install docker-ce
Lendo listas de pacotes... Pronto
Construindo árvore de dependências
Lendo informação de estado... Pronto
Os seguintes pacotes foram instalados automaticamente e já não são necessários:
  libfreetype6 os-prober
Utilize 'apt-get autoremove' para os remover.
Os pacotes extra a seguir serão instalados:
  aufs-tools cgroup-lite git git-man liberror-perl libltdl7
  libsystemd-journal0 pigz
```

Fonte: captura de tela elaborada pelo autor.

Iniciando e testando o Docker

Para ver se o Docker foi instalado corretamente, devemos iniciar o serviço do Docker e verificar se ele está em execução. Podemos fazer isso com os seguintes comandos:

```
sudo service docker start
```

```
service docker status
```

O comando `sudo` utiliza a permissão de usuário administrador para execução do restante da linha, e as instruções `service docker start` iniciam o serviço Docker que foi instalado anteriormente.

Após a inicialização do Docker, utilizamos o comando `service docker status` que mostra o status do serviço, ou seja, se ele está em execução ou parado.

Caso ocorra algum erro ao iniciar e testar o Docker, você deve executar os dois comandos seguintes e tentar novamente:

```
sudo apt-get update
sudo apt-get upgrade
```

Esses comandos utilizam a permissão de usuário administrador através do `sudo` e, além disso, executam uma atualização de pacotes essenciais para o bom funcionamento do sistema operacional, assim como o bom funcionamento dos serviços que rodam através dele, como o Docker.

Agora que instalamos e verificamos seu funcionamento, o sistema está apto a receber as especificidades que queremos criar. Para isso, é possível usar o Docker Swarm. Essa ferramenta é nativa e permite a criação de clusters de Docker. Nesse cenário, é possível agrupar vários hosts em um mesmo *pool* de recursos, o que facilita o *deploy* de contêineres (DIEDRICH, 2018). Esse framework é integrado ao Docker Engine a partir da versão 1.12.0, com o chamado "swarm mode".

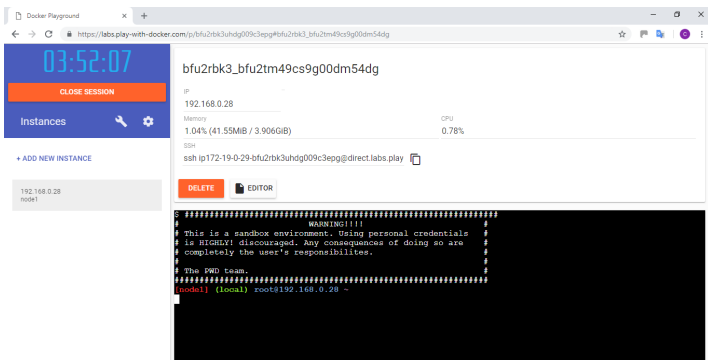
Uma alternativa à instalação do Docker é o portal *Play With Docker*, plataforma pela qual é possível testar o Docker utilizando seus comandos diretamente via navegador, mediante um cadastro.

Após a criação do login na plataforma, teremos acesso a uma interface para criação de instâncias como *clusters* e nós.

Exemplificando

Através do botão *Add new instance*, podemos criar nós para o nosso *cluster* e, então, é possível entrar com comandos na plataforma. Vamos conhecer um dos principais comandos para depois interagirmos com a plataforma. A Figura 3.19 exibe a interface após a criação de um nó:

Figura 3.19 | Plataforma *Play With Docker*



Fonte: captura de tela elaborada pelo autor.

Através dos comandos listados abaixo podemos simular algumas características de sistemas distribuídos com Docker, como criação de nós dos tipos mestre (*manager*) e escravo (*worker*) dentro do nosso *cluster*, gerenciamento de contêiner, réplica de serviços etc.

Comando executado fora do(s) nó(s):

Agora vamos ver alguns comandos que são executados fora dos nós do ambiente Docker. Um dos mais importantes é o de criação de um novo *manager* para o *cluster*. Através do comando, criamos um *manager* chamado de mestre:

```
docker-machine create --driver virtualbox mestre
```

O começo desse comando significa que estamos criando uma nova máquina através do Docker. Utilizaremos `docker-machine create` como driver no VirtualBox e daremos o nome de mestre para nossa máquina.

Agora, vamos fazer a criação de uma nova máquina: o comando segue o mesmo padrão do anterior, mas desta vez vamos chamar a máquina de "escravo1". Ela será um *worker* para o *cluster*, conforme comando abaixo:

```
docker-machine create --driver virtualbox escravo1
```

Podemos verificar o IP de qualquer uma das máquinas de nosso cluster utilizando o comando `docker-machine ip` e, a seguir, consultamos o IP de nossa máquina chamada "mestre":

```
docker-machine ip mestre
```

Um dos comandos Linux mais populares utilizados para acesso de máquinas/servidores é o ssh, que significa "*Secure Shell*", em que fazemos um acesso ou uma conexão através desse protocolo de rede criptográfica, que aplica mais segurança aos serviços de redes. Na utilização do Docker o mesmo comando é usado com o objetivo de acessar os nós criados que, em nosso ambiente, possuem os nomes "mestre" e "escravo1".

```
docker-machine ssh mestre
```

```
docker-machine ssh escravo1
```

É possível verificar os nós (nodes) criados através do comando "ls". Podemos criar um escravo para o cluster com hardware diferente do padrão. Também pode-se ver no comando a seguir que a máquina chamada "escravo5" que definimos a memória com o valor de 512 MB através da instrução `virtualbox-memory "512"` e um disco rígido de 5 GB através da instrução `virtualbox-disk-size "5000"`:

```
docker-machine create --virtualbox-memory "512"
--virtualbox-disk-size "5000" --driver virtualbox
escravo5
```

Comando executado dentro do(s) nó(s) *manager*:

Agora vamos ver os alguns comandos executados dentro do manager de nosso ambiente Docker. O comando a seguir pode ser utilizado para iniciar o cluster através do framework *Swarm*, de gerenciamento de contêineres.

```
docker swarm init --advertise-addr <IP do mestre>
```

Podemos consultar os nós que fazem parte do cluster utilizando o comando `node ls`:

```
docker node ls
```

Às vezes precisamos verificar informações sobre um nó específico. Para isso, utilizamos o comando Docker `inspect`. O exemplo a seguir contém informações sobre o nó chamado "`escravo1`":

```
docker inspect escravo1
```

A seguir, é possível observar um exemplo de criação de um novo serviço (tarefa) para o cluster. Nesse caso, criamos um serviço Web através do popular servidor web nginx, que contém 3 réplicas:

```
docker service create --name WEB --publish 85:80
--replicas=3 nginx:1.12.1
```

No parâmetro `name` definimos um nome para a execução do serviço, em nosso caso, `WEB`. Através do parâmetro `publish` definimos uma porta de execução para o Docker e uma para o servidor web `Nginx`, no caso, "85" e "80" e, por último, definimos que o serviço terá 3 réplicas e utilizará o `Nginx` na versão 1.12.1.

Agora que criamos nosso serviço de internet chamado `WEB`, podemos utilizar o comando `ps` seguido do nome do serviço para verificar suas informações. Portanto, o comando a seguir mostra informações sobre um serviço específico aqui denominado como `WEB`:

```
docker service ps WEB
```

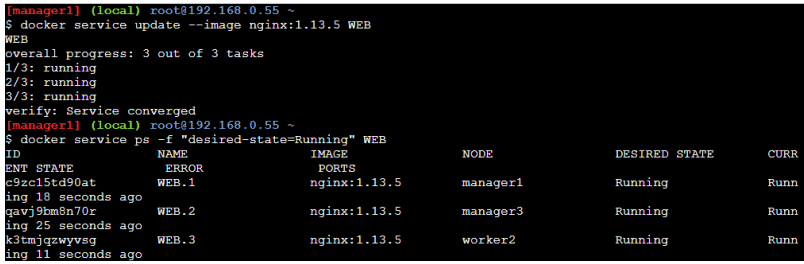
É possível usar os comandos a seguir para alterar a versão das instâncias do `Nginx`. No primeiro comando atualizamos, através do comando `update`, a versão do `Nginx` para 1.13.5 (anteriormente estávamos com a versão 1.12.1). Após isso, utilizamos o comando `ps` com o parâmetro `-f` que significa *filter* ou *find*, que pode ser considerado um filtro para a busca,

em que usamos a palavra-chave *Running* para descobrir qual serviço está em execução e se sua versão foi mesmo atualizada. O resultado dos comandos está ilustrado na Figura 3.20.

```
docker service update --image nginx:1.13.5 WEB
```

```
docker service ps -f "desired-state=Running" WEB
```

Figura 3.20 | Plataforma *Play With Docker*



Fonte: elaborada pelo autor.

Caso quiséssemos desfazer nossa atualização de versão das instâncias do Nginx, podemos utilizar o comando `update` como parâmetro `rollback` aplicado sobre o serviço `WEB`. Com isso, teremos a volta da versão das instâncias do `Nginx` para a versão anterior. Que pode ser consultado através do mesmo comando `ps` utilizado no exemplo anterior, conforme a seguir:

```
docker service update --rollback WEB
```

```
docker service ps -f "desired-state=Running" WEB
```

Caso seja necessário parar algum nó e seus serviços, podemos utilizar o comando `update` seguido do parâmetro `availability drain` e o nome do nó que desejamos parar, como em nosso exemplo, o nó denominado como `escravo2`:

```
docker node update --availability drain escravo2
```

Quando executamos esse comando, tanto o nó quantos os serviços que estão em execução serão parados.

Comando executado dentro do(s) nó(s) *worker*:

Agora vamos ver um dos comandos que são executados dentro dos nós escravos (*worker*) do ambiente Docker. O comando a seguir pode ser utilizado para adicionar um *worker* ao nosso cluster (Figura 3.21). Para isso, devemos acessar o nó escravo que queremos adicionar a um cluster e executar o comando a seguir, passando o *token* de segurança e o IP do nó mestre

(*manager*) seguido por porta, conforme a sintaxe representada no comando:

```
docker swarm join --token <TOKEN> <IP do mestre:Porta>
```

Figura 3.21 | Plataforma ***Play With Docker***

Fonte: captura de tela elaborada pelo autor.

Comando executado dentro de cada um dos nós (*managers* e *workers*):

Agora vamos ver um dos comandos executados dentro dos nós tanto *worker*, quanto *managers* de nosso ambiente Docker. Quando executamos o comando Docker `system prune` com o parâmetro `all` dentro de um nó, ele será responsável por apagar/deletar tudo o que foi feito dentro do mesmo. Observamos a seguir sua utilização:

```
docker system prune --all
```

Agora que já conhecemos alguns comandos, vamos através da plataforma *Play With Docker* orquestrar um servidor web Apache em um cluster simples. Primeiramente, você deve estar logado na plataforma de *playground* do Docker, conforme orientações já apresentadas anteriormente. Uma vez que obteve acesso à plataforma, você deverá:

1. Criar um cluster com 3 nós, que serão suficientes para analisarmos nosso cluster sem comprometer a usabilidade da plataforma de testes do Docker. Sendo assim, você deve adicionar 3 nós, que farão parte do cluster, através do botão *Add new instance*.

2. No nó que você deseja que seja o mestre, digite o seguinte comando:

   docker swarm init --advertise-addr <endereço IP desse nó>

   Este comando define o nó como *manager* do cluster. Repare que, ao executá-lo, é apresentada uma saída com a mensagem: "Para adicionar um *worker* ao *swarm*, execute o seguinte comando", conforme pode ser visto (em inglês) na Figura 3.22. Sendo assim, é necessário copiar a saída apresentada a você (que vai ser diferente da destacada em amarelo na Figura 3.22), pois esse comando deverá ser executado em cada um dos demais nós do cluster, adicionando-os como *workers* desse cluster.

Figura 3.22 | Saída do comando `docker swarm init`

Fonte: captura de tela elaborada pelo autor.

3. Agora que os nós estão criados e seus papéis definidos, para criar o serviço que estará rodando (de maneira distribuída, replicada) do servidor web Apache, digite o seguinte comando no nó mestre:

   docker service create --name WEB --publish 80:80 --replicas=5 httpd

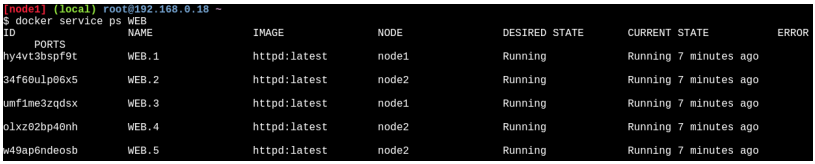
   Esse comando cria 5 instâncias de um servidor web Apache, que responderá na porta mapeada (80, nesse caso) e, para facilitar sua monitoração, demos um nome "amigável" a este serviço: WEB.

4. Para saber em quais nós as 5 réplicas desse serviço estão sendo executadas, digite o seguinte comando:

   docker service ps WEB

   No caso da Figura 3.23 abaixo, podemos ver que 2 instâncias estão rodando no nó 1, e as outras 3 estão rodando no nó 2.

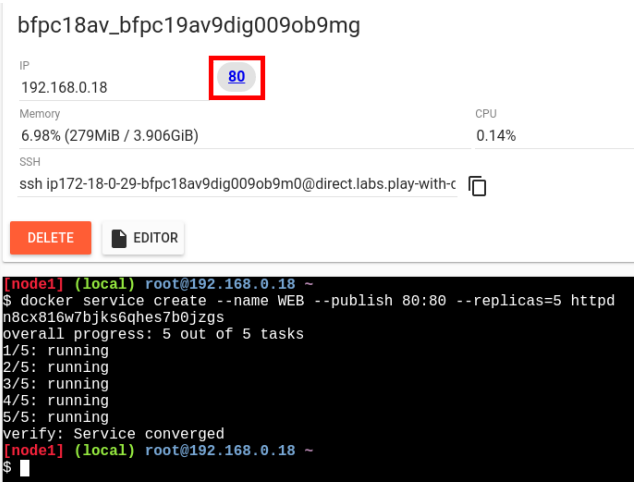
Figura 3.23 | Porta mapeada aparece como um hyperlink



Fonte: captura de tela elaborada pelo autor.

5. Por fim, precisamos acessar a página de boas-vindas desse servidor Apache através do(s) endereço(s) IPv4 de cada nó em que esse serviço web estiver rodando. Reparou que, ao criar o serviço, a porta que você mapeou aparece na parte superior, como um hyperlink? Caso não tenha percebido, veja a porta 80, destacada em vermelho na Figura 3.24. Para acessar a página de boas-vindas, basta clicar nessa porta, em cada um dos nós onde esse serviço está rodando (no caso, nós 1 e 2 do cluster), para vermos a famosa mensagem "*It works!*" do Apache.

Figura 3.24 | Porta mapeada aparece como um hyperlink



Fonte: captura de tela elaborada pelo autor.

Esta sequência de comandos que utilizamos para orquestrar um servidor web Apache em 3 nós é a configuração utilizada em sistemas distribuídos, para que os acessos a um website sejam balanceados e, caso ocorra algum problema em um dos nós que mantêm a aplicação, o outro nó assume a execução.

Reflita

Parou para pensar em quantas aplicações esses conceitos podem ser aplicados? E quantas melhorias as implantações desses conceitos podem trazer a um sistema web?

Nesta seção você conheceu alguns comandos Docker e aprendeu a fazer uma configuração de cluster e de nós para servidor web.

Bons estudos!

Sem medo de errar

Mais um passo e a tão cobiçada vaga naquela grande empresa será sua! O coordenador está gostando de seu desempenho nos testes e diz que, caso você consiga orquestrar o servidor web Apache em um cluster simples, será contratado! Desta forma:

I. Crie um cluster com cinco réplicas do servidor web Apache utilizando o Docker Swarm;

II. Verifique em quais nós do cluster esse serviço está rodando;

III. Acesse a página de boas-vindas desse servidor Apache através do(s) endereço(s) IPv4 de cada nó onde esse serviço web estiver rodando.

Para resolver esse desafio, primeiramente você deve estar logado na plataforma de playground do Docker, conforme orientações já apresentadas anteriormente. Uma vez que obteve acesso à plataforma, você deverá seguir os passos abaixo:

1. Como não foi especificada a quantidade de nós do cluster, crie o mesmo com 3 nós, que serão suficientes para analisar o cluster sem comprometer a usabilidade da plataforma de testes do Docker. Sendo assim, adicione 3 nós, que farão parte do cluster, através do botão *Add new instance*.

2. No nó que você deseja que seja o mestre, digite o seguinte comando:

   `docker swarm init --advertise-addr <endereço IP desse nó>`

   Este comando define o nó como *manager* do cluster. Repare que, ao executá-lo, é apresentada uma saída com a mensagem: "Para adicionar um *worker* ao swarm, execute o seguinte comando", conforme pode ser visto (em inglês) na Figura 3.22. Sendo assim, copie a saída que foi apresentada a você (que vai ser diferente da destacada em amarelo na Figura 3.22), pois esse comando deverá ser executado em cada um dos demais nós do cluster, adicionando-os como *workers* desse cluster.

Figura 3.22 | Saída do comando `docker swarm init`

```
[node1] (local) root@192.168.0.18 ~
$ docker swarm init --advertise-addr 192.168.0.18
Swarm initialized: current node (1v3ezy8z8ydd1jd1j8kcdjk8v) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-1eu6jq2a7j5w76ulzfevphb207kucl3nymzkgm317n4a3ohvqc-eexepp3i9erwvr
vjsdvk6tq52 192.168.0.18:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.
```

Fonte: captura de tela elaborada pelo autor.

3. Agora que os nós estão criados e seus papéis definidos, para criar o serviço que estará rodando (de maneira distribuída, replicada) do servidor web Apache, digite o seguinte comando no nó mestre:

   `docker service create --name WEB --publish 80:80 --replicas=5 httpd`

   Esse comando cria 5 instâncias de um servidor web Apache, que responderá na porta mapeada (80, nesse caso) e, para facilitar sua monitoração, demos um nome "amigável" a este serviço: WEB.

4. Para saber em quais nós as 5 réplicas desse serviço estão sendo executadas, digite o seguinte comando:

   docker service ps WEB

   No caso da Figura 3.23 abaixo, podemos ver que 2 instâncias estão rodando no nó 1, e as outras 3 estão rodando no nó 2.

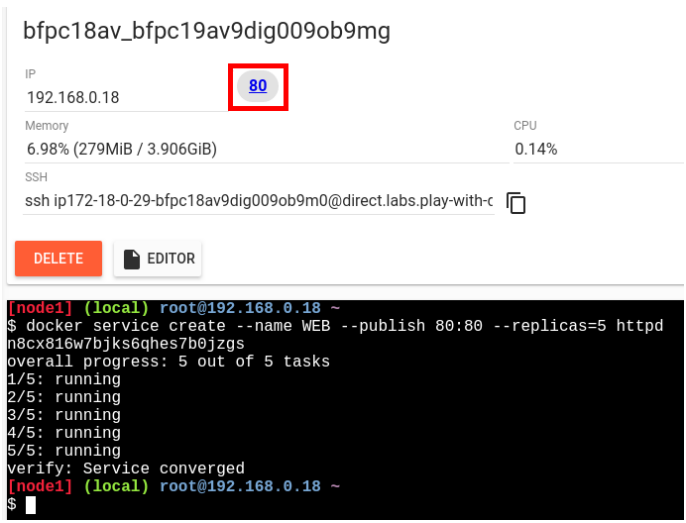
Figura 3.23 | Porta mapeada aparece como um hyperlink

```
[node1] (local) root@192.168.0.18 ~
$ docker service ps WEB
ID             NAME      IMAGE          NODE      DESIRED STATE   CURRENT STATE           ERROR
    PORTS
hy4vt3bspf9t   WEB.1     httpd:latest   node1     Running         Running 7 minutes ago

34f60ulp06x5   WEB.2     httpd:latest   node2     Running         Running 7 minutes ago

umf1me3zqdsx   WEB.3     httpd:latest   node1     Running         Running 7 minutes ago

olxz02bp40nh   WEB.4     httpd:latest   node2     Running         Running 7 minutes ago

w49ap6ndeosb   WEB.5     httpd:latest   node2     Running         Running 7 minutes ago
```

Fonte: captura de tela elaborada pelo autor.

5. Por fim, precisamos acessar a página de boas-vindas desse servidor Apache através do(s) endereço(s) IPv4 de cada nó onde esse serviço web estiver rodando. Reparou que, ao criar o serviço, a porta que você mapeou aparece na parte superior, como um hyperlink? Caso não tenha percebido, veja a porta 80, destacada em vermelho na Figura 3.24. Para acessar a página de boas-vindas, basta clicar nessa porta, em cada um dos nós onde esse serviço está rodando (no caso, nós 1 e 2 do cluster), para vermos a famosa mensagem "*It works!*" do Apache.

Figura 3.24 | Porta mapeada aparece como um hyperlink



Fonte: captura de tela elaborada pelo autor.

O coordenador ficou impressionado com seus conhecimentos e sua desenvoltura ao executar o desafio proposto por ele: missão cumprida, o emprego é seu!

Avançando na prática

Criando um contêiner para um servidor web Apache através do Docker

Descrição da situação-problema

Você deve criar um contêiner para um servidor web Apache através do Docker e verificar se consegue acessar a página de boas-vindas desse servidor Apache, através do endereço IP de *localhost* da máquina física e da porta que foi mapeada na criação do contêiner.

Resolução da situação-problema

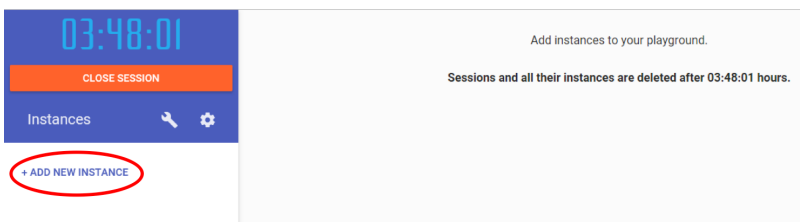
Para ajudá-lo nesta tarefa, siga as seguintes etapas:

1. Acesse o website http://play-with-docker.com/ (acesso em: 8 fev. 2019).

   Esse website, desenvolvido pela própria equipe do Docker, dará acesso – limitado de 4 horas – a um ambiente remoto no qual o Docker já está instalado, bastando digitar os comandos para criação dos contêineres. Perceba que você necessitará realizar um cadastro (gratuito) na plataforma, para posteriormente realizar o *login* nesse ambiente.

2. Uma vez logado, você poderá digitar os comandos no terminal (área preta do portal), conforme ilustrado abaixo, após clicar no botão *Add New Instance*, conforme Figura 3.25:

Figura 3.25 | Acesso ao ambiente remoto do Docker



Fonte: captura de tela elaborada pelo autor.

3. Digite o seguinte comando:

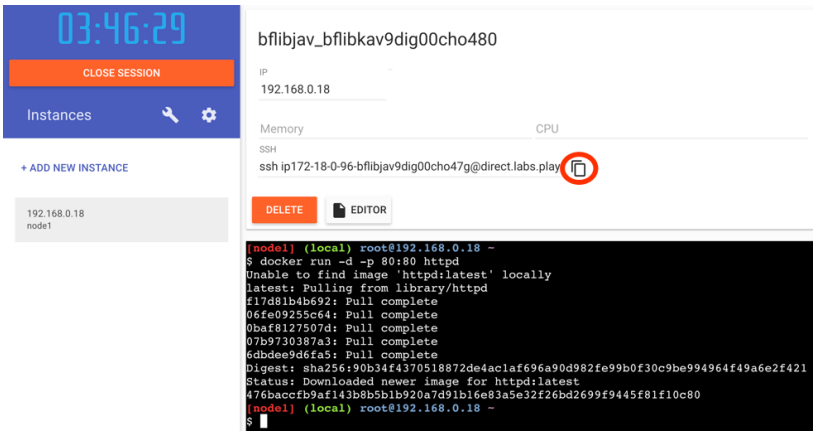
   ```
   docker run -d -p 80:80 httpd
   ```

4. Agora, como o servidor web apache, instalado no contêiner, já deve estar rodando na porta 80, basta acessar a página de boas-vindas pelo navegador web de sua preferência, com o link composto por:

   ip<endereço IP público>-<indentificador do nó>-<porta mapeada>.direct.labs.play-with-docker.com

   As informações necessárias para formatar o link de acesso, conforme padrão acima, podem ser obtidas através da cópia dos dados de acesso via SSH, como destacado em vermelho na Figura 3.26.

Figura 3.26 | Informações de acesso



Fonte: captura de tela elaborada pelo autor.

5. A título de exemplo, o link da figura acima, após adaptado com as informações de acesso, ficou da seguinte forma: ip172-18-0-96-bflibjav9dig00cho47g-80.direct.labs.play-with-docker.com/.

6. Caso você veja a mensagem *It works!*, parabéns: você passou para a próxima – e última – etapa do processo seletivo! Caso não apareça essa mensagem, certifique-se de que você montou o link corretamente, comparando o padrão apresentado na etapa 4, substituindo com as informações referentes à sua conexão.

Faça valer a pena

1. Com o Docker existe uma série de comandos que devemos utilizar para realizar determinadas tarefas dentro do cluster, por exemplo, criação de nós e definição de papéis de nós, como o nó mestre (*manager*) e o nó escravo (*worker*). Com isso, devemos estar atentos à sintaxe na utilização correta dos comandos.

De acordo com o texto, sabemos que há um comando para criação de nós em um cluster. Assinale a alternativa que corresponde à sintaxe correta deste comando para a criação de um nó chamado "mestre":

a) `docker-cluster create --driver virtualbox mestre`
b) `docker-machine create --driver virtualbox mestre`
c) `docker-machine create node --driver virtualbox mestre`
d) `docker-machine new --driver virtualbox mestre`
e) `docker cluster create --node mestre`

2. Com o Docker existe uma série de comandos que devemos utilizar para realizar determinadas tarefas dentro do cluster, por exemplo: acessar nós via nome. Com isso, devemos estar atentos à sintaxe na utilização correta dos comandos.

De acordo com o texto sabemos, que há um comando para acesso de nó via nome. Assinale a alternativa que corresponde à sintaxe correta deste comando para acessar um nó chamado "mestre":

a) `docker ssh mestre`
b) `docker-machine acess mestre`
c) `docker-machine into mestre`
d) `docker-machine ssh mestre`
e) `docker-machine create mestre`

3. A instalação do Docker em máquinas com o sistema operacional Ubuntu da distribuição GNU/Linux segue uma lista de procedimentos de acordo com a documentação oficial do Docker. Cada etapa é importante para que a instalação seja encerrada com sucesso.

Assinale a alternativa que traz a ordem correta de procedimentos que devem ser realizados para a instalação do Docker no sistema operacional Ubuntu.

a) Remover as versões anteriores do Docker, atualizar os pacotes e os repositórios, adicionar o repositório de instalação do Docker, fazer a instalação do Docker e verificar se foi instalado corretamente.

b) Adicionar o repositório de instalação do Docker, fazer a instalação do Docker e verificar se foi instalado corretamente.
c) Atualizar os pacotes e os repositórios e fazer a instalação do Docker.
d) Remover as versões anteriores do Docker, atualizar os pacotes e os repositórios, fazer a instalação do Docker e verificar se foi instalado corretamente.
e) Atualizar os pacotes e o repositório, pois o Docker já é nativo no sistema operacional Ubuntu.

Referências

COULOURIS, G. et al. **Sistemas Distribuídos.** Porto Alegre: Bookman, 2013.

DAWSON, P e WOLF, C. (Portal). **Virtualization Key Initiative Overview**. 22 jul. 2011. Disponível em: https://www.gartner.com/doc/1745020/virtualization-key-initiative-overview/. Acesso em: 03 nov. 2018.

DIEDRICH, C. **Docker Swarm.** Disponível em: https://www.mundodocker.com.br/docker--swarm/. Acesso em: 5 dez. 2018.

DOCKER, C. **Docker Containerization Unlocks the Potential for Dev and Ops.** Disponível em: https://www.docker.com/why-docker. Acesso em: 10 dez. 2018.

DOCKER (Portal) Documentação. Disponível em https://docs.docker.com/. Acesso: 23 nov. 2018.

LINUXCONTAINER. **Infrastructure for container projects.** Disponível em: https://linuxcontainers.org. Acesso em: 10 dez. 2018.

ORACLE. **Oracle VM VirtualBox®: User Manual.** 2018. Disponível em: https://www.virtualbox.org/manual/UserManual.html. Acesso em: 02 nov. 2018.

REDHAT. **O que é virtualização?** 2018. Disponível em: https://www.redhat.com/pt-br/topics/virtualization/what-is-virtualization. Acesso em: 27 nov. 2018.

TANENBAUM, A. S e STEEN, M. V. **Sistemas Distribuídos.** Princípios e Paradigmas. 2 ed. São Paulo: Pearson, 2008.

Unidade 4

Aplicações de sistemas distribuídos e segurança

Convite ao estudo

Olá, caro(a) aluno(a), seja bem-vindo(a) a mais uma unidade! Agora que já conhecemos importantes conceitos de virtualização e conteinerização, que são tecnologias essenciais quando falamos de sistemas distribuídos, vamos entender a parte de segurança de nossos sistemas. Você já parou para pensar quais as principais ameaças que os sistemas distribuídos podem sofrer? Será que existem quantos tipos de ataques que podem afetar nossos sistemas distribuídos? Além disso, qual a melhor forma de se proteger e prevenir esses tipos de ataques? Nesta unidade, vamos entender conceitos muito importantes, para que você consiga responder as questões anteriores e muitas outras. Talvez você já tenha ouvido falar de ataques a sistemas, mas será que já ouviu falar sobre estratégias de violação de segurança? Ainda que sim, como podemos nos prevenir dessas ameaças encontradas em sistemas computacionais, por exemplo? Se você respondeu não a alguma dessas perguntas, não se preocupe! A ideia aqui é mostrar que, caso você estude com afinco, pesquise e faça todas as atividades sugeridas, você aprenderá a criar sistemas distribuídos seguros e com poucas vulnerabilidades.

Após analisarmos as questões de segurança, você estará pronto para fazer as máquinas se comunicarem entre si! Como será que as máquinas em rede conseguem se comunicar e utilizar uma aplicação que esteja em execução em outra máquina? Será que existe mais de uma forma para realizar essa comunicação? Além disso, ainda que exista, qual será a principal diferença entre elas? Ao término desta seção, você conseguirá responder essas questões e, além disso, terá a oportunidade de ver na prática como essas comunicações ocorrem em termos de código.

Você conclui a sua graduação na área de TI e agora pretende abrir sua própria empresa, desenvolvendo e comercializando um sistema CRM (*Customer Relationship Management*), um termo em inglês que pode ser traduzido para a língua portuguesa como Gestão de Relacionamento com o Cliente, que possuirá várias funcionalidades inovadoras, mas também deve possuir as mesmas funcionalidades comumente presentes em sistemas de empresas concorrentes. Como você pretende que sua aplicação tenha compatibilidade com sistemas operacionais comumente encontrados nos laptops dos vendedores das empresas de seus futuros clientes, você decide utilizar a

linguagem Java para desenvolver essa aplicação. Você será capaz de atingir todos os objetivos conforme a necessidade de sua empresa?

Na primeira seção desta Unidade você entenderá conceitos de segurança para sistemas distribuídos. Já na segunda seção, você irá aprender a utilizar Sockets com Java, e entenderá qual a importância dessa aplicação em projetos de sistemas distribuídos. Por fim, você terá a oportunidade de colocar em prática conceitos importantes de Java, através do uso de RPC com Java, que é muito utilizado no mercado de trabalho.

Chegou a hora de prosseguir com os estudos dos sistemas distribuídos, o que irá lhe proporcionar conhecimentos e oportunidades no mercado de trabalho. Lembre-se que, quanto mais você se dedicar, mais poderá aproveitar os ensinamentos transmitidos neste material.

Seção 4.1

Segurança em sistemas distribuídos

Diálogo aberto

Caro(a) estudante, chegou o momento de realizarmos a comunicação entre as máquinas no sistema distribuído, o que significa que utilizaremos algum tipo de comunicação entre essas máquinas, que estão interligadas através de uma rede de computadores. Isso implica, automaticamente, em uma preocupação inerente: a segurança desses sistemas e dos dados que trafegam por ela.

Nesta seção vamos trabalhar, conceitualmente, com os principais tipos de ameaças enfrentadas pelos sistemas distribuídos, assim como discutir possíveis soluções adotadas para cada tipo de ameaça conhecida.

A segurança de sistemas é sempre uma área que desperta muito interesse, dada a sua importância e criticidade, principalmente em sistemas distribuídos que, por sua própria natureza, necessitam de vários protocolos e portas de comunicação, e intercomunicação entre processos, para funcionar adequadamente. Imagine aplicações utilizadas por vários colaboradores, de diferentes cargos, em uma grande empresa: a segurança desse tipo de aplicação necessita ser analisada com atenção redobrada.

Como um jovem empreendedor que você é, deve saber que um sistema CRM é uma aplicação utilizada sempre em ambientes coorporativos, portanto é crucial que você conheça e saiba proteger o seu sistema contra vulnerabilidades de segurança em sistemas distribuídos. Você listará quais são os tipos de ataques que devem ser prevenidos no desenvolvimento de um sistema CRM.

Utilize o seu conhecimento atrelado ao conteúdo da seção para solucionar a situação-problema apresentada. Tenho certeza que você será capaz de passar por essa primeira etapa, rapidamente e sem percalços, deixando o seu sistema de CRM *bullet proof* – à prova de balas –, como os norte-americanos costumam falar.

Não pode faltar

Todo sistema distribuído implementado tem, como um dos principais e mais importantes, aspectos de projeto à segurança, conforme foi estudado anteriormente. Em um sistema distribuído temos uma série de componentes de hardware e software, físicos ou virtualizados, que se comunicam para a execução das aplicações distribuídas. Por conta disso vários tipos de ameaças podem afetar a segurança de nossos sistemas.

Uma das formas mais funcionais de prevenir nossos sistemas atualmente é utilizar uma estratégia de segurança multicamadas. Este tipo de estratégia protege com diferentes tecnologias de segurança os principais pontos de entrada de ameaças. A segurança multicamada aumenta consideravelmente o grau de dificuldade para uma invasão de um intruso, reduzindo drasticamente o risco de um hacker ter acesso indevido à rede e dados de empresas. Com a estratégia de segurança multicamadas, as ameaças encontram muito mais dificuldade em causar algum dano, pois caso ultrapassem alguma camada, deverão ser barradas pela camada seguinte. Quando implementada corretamente, uma estratégia em várias camadas oferecerá proteção contra vírus, *spyware*, *malware*, *phishing*, invasão de redes, spam e vazamento de dados.

Sempre devemos levar em consideração que tudo que é relacionado à segurança em um sistema computacional depende do fator tecnológico, que é composto pelos componentes de hardware e software, e também pelo fator humano que acaba sendo o ponto mais vulnerável do sistema. Por exemplo, quantas páginas falsas *(fakes)* se passando por lojas virtuais famosas ou até mesmo bancos não são encontradas na internet e afetam centenas de usuários diariamente com o roubo de informações sigilosas? Esse tipo de ameaça é chamado de *Trojan Banking,* e é muito frequente sua distribuição através de e-mails e sites infectados.

Podemos dividir a parte de segurança de sistemas distribuídos em duas: permissão de acessos a serviços e recursos disponíveis no sistema e comunicação entre máquinas que contém mais de um processo e usuários diferentes (GOODRICH e TAMASSIA, 2012).

Podemos relacionar a segurança de um sistema distribuído aos seguintes fatores:

1. Confidencialidade

A confidencialidade significa que a informação só estará disponível para os usuários ou máquinas autorizadas. Uma das formas de se garantir confidencialidade das mensagens é através do uso de autenticação baseada em chave privada.

2. Integridade

A integridade significa que a informação armazenada ou transferida é apresentada corretamente para quem precisa fazer a sua consulta. A integridade das mensagens pode ser alcançada através de assinaturas digitais e chaves de sessão.

3. Autenticidade

A autenticidade pode ser alcançada quando criamos permissões de autenticação para os principais usuários e máquinas do serviço, com isso nosso sistema só irá funcionar corretamente com os usuários e máquinas autenticados.

4. Disponibilidade

A disponibilidade significa garantir que a informação esteja sempre disponível para quem precisar dela. Podemos obter a disponibilidade do sistema utilizando as políticas de segurança corretamente em nosso sistema.

5. Não repúdio

O não repúdio ou princípio do não repúdio, como é conhecido, garante a autenticidade de uma informação utilizada por sistemas distribuídos. Ele é uma grande medida de segurança, já que pode ser aplicada a e-mails, imagens, formulários web, arquivos eletrônicos transferidos entre empresas (EDI), entre outros itens. Uma das principais maneiras de se aplicar essa exigência de segurança é através de assinatura digital e certificados digitais.

Pesquise mais

Os certificados digitais são documentos eletrônicos que servem como uma carteira de identidade virtual para identificar e representar uma pessoa ou empresa na internet. Eles são úteis nos casos em que seja extremamente necessário validar a identidade de um usuário, como em transações bancárias. Você pode consultar mais detalhes sobre eles e também como adquiri-los através da matéria do Portal Techtudo.

GUILHERME NETO. **O que é e como conseguir um certificado digital.** 2012. Disponível em: https://www.techtudo.com.br/artigos/noticia/2012/07/o-que-e-e-como-conseguir-um-certificado-digital.html. Acesso em: 18 dez. 2018.

Aplicando esses fatores temos o objetivo de proteger o canal de comunicação de nossas aplicações, tornando assim canais seguros em nossos sistemas distribuídos. O grande objetivo de nossa aplicação distribuída é que ela possa

se comunicar, ou seja, trocar dados, com confiabilidade, integridade e autenticidade. Portanto, uma das principais formas de proteção é deixar a comunicação permitida apenas em máquinas e por usuários autenticados em nosso sistema e com as permissões necessárias.

Podemos observar na Figura 4.1 a comunicação entre processos e máquinas de um sistema distribuído através de um canal seguro. Para que isso ocorra temos que ter as máquinas 1 e 2 autenticadas dentro do nosso sistema. Com isso, caso o sistema sofra um ataque, o atacante não será capaz de ler, copiar, alterar, reenviar, reordenar ou enviar novas mensagens. Dessa forma nosso sistema estará protegido.

Figura 4.1 | Canal seguro de comunicação entre processos/máquinas

Fonte: elaborada pelo autor.

Assimile

Uma das maneiras para ter um canal seguro de comunicação entre os processos/máquinas de nossa aplicação distribuída é autenticar os principais servidores e utilizadores da aplicação.

Para produzir um sistema que seja seguro contra diversas ameaças, precisamos aprender a classificar essas ameaças e entender seus métodos de ataque. As ameaças aos sistemas distribuídos podem ser divididas nas seguintes classes (COULOURIS *et al.*, 2013):

- *Leakage* (vazamento): acesso à informação por agentes não autorizados.
- *Tampering* (falsificação): modificação não autorizada de uma informação.
- *Vandalism* (vandalismo): interferência no funcionamento de um sistema sem ganhos para o criminoso.

Para que o invasor consiga violar um sistema através de algumas das estratégias apresentadas anteriormente, é necessário acessá-lo. Geralmente, todas as máquinas que compõem um sistema distribuído têm canais de comunicação para acesso autorizado às suas facilidades, e através desses canais de comunicação que o acesso não autorizado pode ocorrer.

As estratégias de violações de segurança em sistemas distribuídos dependem da obtenção de acesso aos canais de comunicação de nosso sistema, ou do estabelecimento de canais que escondem conexões, com a autoridade desejada. Fazem parte destas estratégias:

- *Eavesdropping*: acesso a cópias de mensagem sem autorização. Geralmente essa estratégia funciona através da captura de mensagens da rede. Por exemplo, usando a internet um computador pode se passar por outro, quando configurado com o endereço de rede de outro, desta forma ele pode receber as mensagens endereçadas a outro destinatário.
- *Masquerading* (disfarce): a máquina do invasor faz envio ou recebimento de mensagens utilizando a identidade de outra máquina autorizada pela aplicação.
- *Message tampering* (falsificação de mensagem): a máquina do invasor faz a captura e alteração do conteúdo das mensagens e após isso faz a transferência ao destinatário. Uma das maneiras mais fáceis de se defender a esse tipo de ataque é utilizando broadcast para o envio das mensagens, como ocorre nas redes Ethernet.
- *Replaying*: quando o invasor consegue fazer a captura e armazenamento das mensagens por um período de tempo, utilizando para isso o envio atrasado das mensagens aos seus destinatários.

Reflita

Agora que você conhece algumas das estratégias de violação de sistemas distribuídos, já parou para pensar como podemos defender nossos sistemas dessas ameaças?

Para os invasores conseguirem acesso ao sistema é utilizado um método simples de infiltração que pode ser o uso de programas de quebra de senhas para obter as chaves de acesso de algum usuário do sistema. Além desta forma simples e não muito eficaz de invasão, existem outras maneiras mais sutis, que estão se tornando bem conhecidas:

- Vírus: um programa anexado a um hospedeiro legítimo, que se instala sozinho no ambiente alvo, sempre que o programa hospedeiro é executado. Uma vez instalado, ele realiza suas ações criminosas. A notícia do Portal BBC traz um ataque cibernético que paralisou um hospital e foi causado por um vírus. Disponível em: https://www.bbc.com/portuguese/brasil-40870377. Acesso em: 18 dez. 2018.
- *Worm*: um programa que varre um sistema, ou uma rede, replicando-se e buscando bloquear todos os recursos disponíveis, até torná-lo inoperante. Ao contrário do vírus, um *worm* normalmente não destrói dados.
- Cavalo de Troia (*Trojan Horse*): um programa oferecido aos usuários através de um sistema que mostra ser capaz de utilizar uma função útil, mas que tem uma segunda intenção que vem oculta através de uma função. O exemplo mais comum é o *spoof login*, um programa que apresenta aos usuários um diálogo idêntico ao diálogo normal de obtenção de login (nome do usuário) e *password* (senha), mas que na realidade armazena as informações fornecidas pelos usuários em um arquivo, com o objetivo de uso posterior ilícito.
- *Spyware*: são programas espiões utilizados para roubar informações sobre os costumes dos usuários na internet, com o propósito de fornecer uma propaganda preparada de acordo com as pesquisas daquele usuário.
- *Keyloggers*: são programas de computador capazes de monitorar, armazenar e enviar todas as teclas digitadas pela vítima para um invasor. Atualmente, os *keyloggers* são incorporados em outros códigos como trojans, utilizados principalmente para roubo de senhas ou dados bancários.
- *Backdoor* (Porta dos Fundos): é um recurso utilizado por diversos *malwares* para conseguir acesso remoto a uma rede ou sistema infectado.
- Spam: são e-mails indesejáveis, que muitas vezes têm o intuito de conseguir dados do usuário.
- *Adware*: são programas que exibem uma grande quantidade de anúncios sem a autorização do usuário.

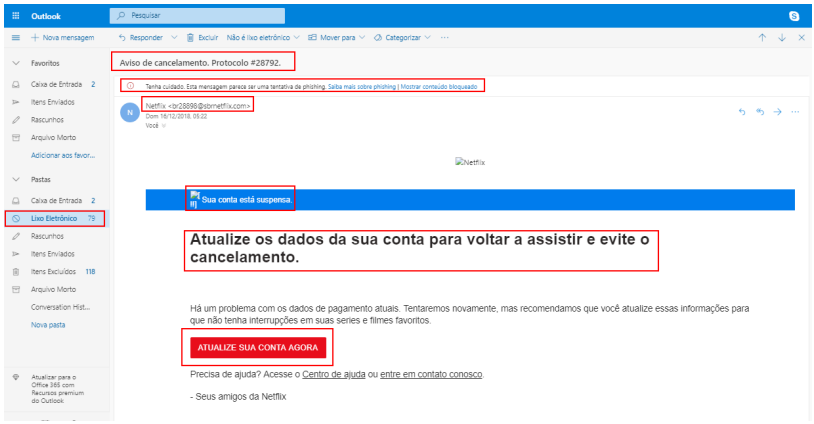
- *Exploits*: programas com conteúdo malicioso, que tendem a utilizar vulnerabilidades do sistema e outros programas.
- *Hoax*: são falsas mensagens que alertam o usuário sobre um determinado tipo de vírus disponível em sua máquina.
- *Phishing*: são programas com o objetivo de adquirir informações sigilosas de um usuário, podem se disfarçar de mensagens, softwares e outras formas de aplicações amigáveis.

Exemplificando

Alguma vez você já fez acesso a algum website desconhecido através de seu smartphone e foi redirecionado a alguma página que afirmava ocorrer algum erro em seu aparelho? Você pode ter sido vítima de uma tentativa de *Phishing*, que ocorreu através da página que tentava através de uma mensagem falsa acessar alguma informação sigilosa.

Podemos observar na Figura 4.2 uma tentativa de *Phising* que inclusive foi movimentada para o Lixo Eletrônico do serviço de e-mail, pois foi identificada como *Phising*. Podemos ver nos espaços destacados em vermelho, que o endereço de e-mail não corresponde ao da empresa que se diz ser na mensagem do e-mail. Além disso, ele pede para que atualize informações e desta forma acaba ocorrendo o vazamento de dados/informações pessoais do usuário.

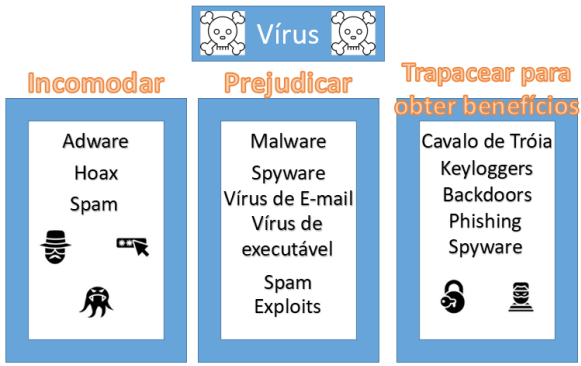
Figura 4.2 | Exemplo de *Phishing*



Fonte: elaborada pelo autor.

Podemos observar na Figura 4.3 uma divisão entre os tipos de vírus listados, onde separamos por vírus com o objetivo de incomodar usuários, prejudicar e trapacear para obter benefícios aos seus invasores.

Figura 4.3 | Divisão dos Vírus por categorias



Fonte: elaborada pelo autor.

Exemplificando

Um serviço muito interessante para verificar se há alguma vulnerabilidade em uma conta de e-mail é o portal "Have I Been Pwned", Portal https://haveibeenpwned.com/ Acesso em: 27 nov. 2018.

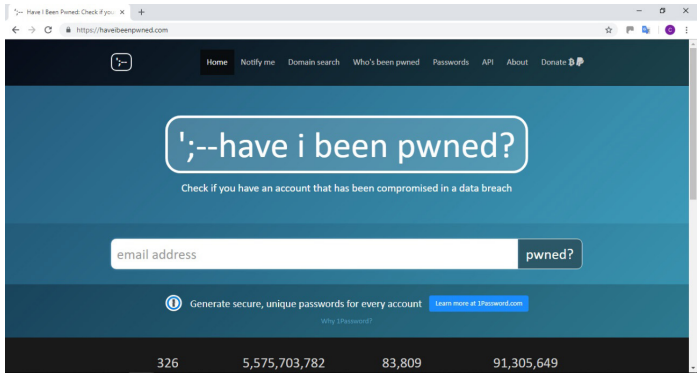
Através desse portal podemos ser alertados sobre quais sites expõem nossas informações através de alguma vulnerabilidade.

Para execução dessa atividade, siga as seguintes etapas:

1. Acesse o portal através do endereço web: https://haveibeenpwned.com/
2. Digite o endereço de e-mail da empresa que você trabalha ou pessoal na caixa de texto, acessível diretamente pela página principal do portal.
3. Clique no botão "Pwned".
4. Caso apareça a mensagem "Oh no – pwned!", as credenciais informadas estão comprometidas. Caso apareça a mensagem "Good news – no pwnage found", as credenciais informadas não foram comprometidas, ao menos até então.

Podemos observar a interface do portal através da Figura 4.4:

Figura 4.4 | Portal “Have I Been Pwned”



Fonte: elaborada pelo autor.

Uma ameaça que é frequente em sistemas distribuídos voltados para internet (web) é a invasão dos servidores webs, que armazenam os arquivos e objetos que são parte do sistema. Este tipo de invasão ocorre através de portas acesso. O que acontece em muitas empresas, é que às vezes há algum servidor mais vulnerável, com suas portas de acesso liberadas para alguns casos que acabam sendo alvo para invasões, através dessa porta de entrada todo o sistema web é contaminado. Já ocorreu em um ambiente coorporativo que trabalhei de uma máquina antiga com um serviço de Wordpress (blogs) quase esquecido, ser utilizada como porta de entrada para hackers acessarem boa parte dos arquivos web importantes da empresa. Uma das formas de se proteger contra essa grande ameaça é utilizar um serviço baseado no protocolo FTP (*File Transfer Protocol*) de transferência de arquivos, que utiliza autenticação para envios de novos arquivos. Esse protocolo é muito utilizado quando falamos em ambientes web. Outra forma de se proteger é implantando o protocolo SSH (*Security Shell*) de acesso seguro, como principal forma de proteção ao acesso dos servidores. Temos outras formas de aumentar a segurança de nossos sistemas distribuídos voltados para web, assim como também temos outros tipos de ameaças. Portanto, devemos nos atualizar frequentemente sobre novas proteções e novas ameaças para que possamos ter um sistema seguro da melhor maneira possível.

Pesquise mais

- Um serviço muito interessante para verificar se há alguma vulnerabilidade em uma conta de e-mail é o portal “Have I Been Pwned”, Portal https://haveibeenpwned.com/. Acesso em: 27 nov. 2018. Através desse portal podemos ser alertados sobre quais sites expõem nossas informações através de alguma vulnerabilidade.

- Você pode fixar seu conhecimento assistindo esse vídeo que mostra os principais tipos de vírus conforme foi estudado nesta seção. Com duração de 6 minutos. Disponível em: Youtube: InterSeptaBR. **Tipos de Vírus para Computadores**. 08 mai. 2012. Disponível em: https://www.youtube.com/watch?v=GK69-u1stJY. Acesso em: 27 nov. 2018.
- Você pode fixar seu conhecimento assistindo esse vídeo que mostra como tratar incidentes de segurança que ocorrem na internet. Com duração de 7 minutos. Disponível em: Youtube: NICbrvideos. **Tratamento de Incidentes de Seguranca na Internet, explicado pelo NIC.br**. 10 nov. 2014. Disponível em: https://www.youtube.com/watch?v=flu6JPRHW04. Acesso em: 27 nov. 2018.

Assim finalizamos mais uma seção no estudo dos sistemas distribuídos. Espero que o conhecimento adquirido seja de importância para seu crescimento profissional, bons estudos e até a próxima seção!

Sem medo de errar

O principal produto da sua empresa é um sistema CRM inovador, tanto em termos de funcionalidades, quanto em facilidade de uso, como valores perceptíveis pelos clientes. Um aspecto importantíssimo para o sucesso desse tipo de produto, é que ele esteja protegido quanto a ataques cibernéticos, é claro. Segundo Colouris *et al.* (2013), ataques em sistemas distribuídos estão atrelados à obtenção de acesso a algum dos diferentes mecanismos de comunicação entre processos, como o próprio RMI, que veremos na última Seção desta Unidade. Sua tarefa, nesse momento, é listar quais são os tipos de ataques que devem ser prevenidos no desenvolvimento de um sistema CRM, lembrando que essa aplicação estará sendo executada em uma plataforma como serviço (PaaS), e será acessada através da internet. Sendo assim, apresento aqui a Tabela 4.1, que lista os principais tipos de ataques que podem ocorrer nesse cenário, bem como as melhores práticas para prevenir-los, conforme Coulouris *et al.* (2013).

Tabela 4.1 | Principais categorias de ataque e medidas preventivas

Categoria de ataque	Definição	Prevenção
Vazamento	Aquisição de informação por pessoal não autorizado.	Conscientização dos colaboradores. Investimento em *appliances* (hardware + software dedicados) de segurança. Uso de autenticação baseada em chave privada, não apenas para usuários, mas também para máquinas e serviços. Assinaturas digitais e chaves de sessão.
Falsificação	Alteração não autorizada da informação.	Investimento em *appliances* (hardware + software dedicados) de segurança. Uso de autenticação baseada em chave privada, não apenas para usuários, mas também para máquinas e serviços. Assinaturas digitais e chaves de sessão.
Vandalismo	Interferência na operação de um sistema sem obtenção de lucro pelo invasor.	Investimento em *appliances* (hardware + software dedicados) de segurança. Uso de autenticação baseada em chave privada, não apenas para usuários, mas também para máquinas e serviços. Assinaturas digitais e chaves de sessão.

Fonte: elaborada pelo autor.

Avançando na prática

Pesquisando se suas credenciais foram comprometidas

Descrição da situação-problema

É importante, à qualquer sistema acessível pela internet e, em especial em sistemas distribuídos, que ao receber o cadastro de novos usuários, os e-mails destes sejam validados, inclusive não apenas no momento do cadastro, mas também de maneira periódica, pois clientes que fazem uso (e consequentemente) de autenticação no sistema, podem ter suas credenciais comprometidas, por mau uso do e-mail corporativo, ou terem sido alvos de engenharia social etc. Desta forma, é importante que você saiba utilizar um ou mais recursos para executar essa verificação em seu sistema.

Resolução da situação-problema

Um serviço muito interessante para esse tipo de tarefa é o portal "Have I Been Pwned", Portal <https://haveibeenpwned.com/>. Acesso em: 27 nov. 2018.

Para execução dessa atividade, siga as seguintes etapas:

1. Acesse o portal através do endereço web: https://haveibeenpwned.com/
2. Digite o endereço de e-mail da empresa que você trabalha (caso possua um e-mail coorporativo) na caixa de texto, acessível diretamente pela página principal do portal.
3. Clique no botão "Pwned".
4. Caso apareça a mensagem "Oh no – pwned!", as credenciais informadas estão comprometidas. Caso apareça a mensagem "Good news – no pwnage found", as credenciais informadas não foram comprometidas, ao menos até então.
5. Repita o processo, a partir da Etapa 1, utilizando agora, na Etapa 2, seu e-mail pessoal.

Faça valer a pena

1. No contexto de segurança de sistemas em rede e, portanto, também em sistemas distribuídos, existem vários tipos de ameaças, como por exemplo, os *Trojans*, *Malware*, Vírus, Spams, *Spywares* e Cavalos de Troia; ameaças estas que têm características e objetivos diferentes, conforme discutido no texto-base desta Seção.

Independentemente do tipo de ameaça, elas podem ser categorizadas em 3 tipos. Quais são esses tipos?
a) Vazamento, vandalismo e falsificação.
b) Danoso, doloso e sem ônus.
c) Falsificação, roubo e hackeamento.
d) Vírus, *Malwares* e *Trojans*.
e) Vazamento, falsificação e sequestro.

2. Para que possamos nos proteger de maneira mais eficiente frente às ameaças as quais os sistemas distribuídos estão sujeitos, é importante entender as diferenças e, mais ainda, os objetivos das principais ameaças, de maneira a utilizarmos técnicas específicas e, portanto, mais assertivas, para proteção contra cada tipo de ameaça.

Sobre os tipos de ameaça, marque V para verdadeiro ou F para falso:
( ) *Keyloggers* capturam teclas digitadas pelas vítimas.
( ) *Backdoors* são vírus que roubam as credenciais das vítimas.
( ) *Spywares* bloqueiam o acesso a um recurso infectado por um vírus.
( ) *Adwares* são programas que exibem anúncios não autorizados.

Assinale a alternativa que representa a sequência CORRETA.
a) V - V - V - V
b) F - V - V - V
c) V - V - F - V
d) V - F - F - V
e) F - V - F - V

3. Sistemas distribuídos podem ser entendidos como máquinas em redes que possuem uma maior integração que sistemas puramente de redes. A razão dessa maior integração reside no fato de que as máquinas que fazem parte desse tipo de sistema fazem uso intensivo de comunicação de informações entre si, através dos canais de comunicação.

Sabendo que a falha na segurança de um sistema distribuído significa que o canal de comunicação foi comprometido e, considerando as afirmações abaixo, assinale a alternativa que apresenta boas práticas para proteção do canal de comunicação em termos de segurança.
I – Permitir acesso somente a usuários autenticados.
II – Permitir acesso somente a serviços autenticados.
III – Permitir acesso exclusivo a usuários anônimos.
IV – Permitir acesso somente a máquinas autenticadas.
V – Permitir uso do canal de comunicação exclusivamente por processos que sejam executados localmente.

a) Somente a afirmação I está correta.
b) Somente as afirmações I e II estão corretas.
c) Somente as afirmações I, II e IV estão corretas.
d) Somente as afirmações II, III e IV estão corretas.
e) Somente as afirmações I, II, IV e V estão corretas.

Seção 4.2

Utilizando *sockets* com Java

Diálogo aberto

Caro(a) estudante, agora que você já entende os principais aspectos de segurança em sistemas distribuídos, já pode iniciar a comunicação entre máquinas conectadas via rede. Você com certeza sabe que as aplicações podem comunicar-se entre si através da rede, como ocorre, por exemplo, quando você troca mensagens de texto através de seu aplicativo de bate-papo preferido com seu melhor amigo(a) ou namorado(a), certo? Mas você já teve alguma vez a curiosidade de pensar como essa comunicação funciona internamente (ou seja, em termos de programação)? Caso tenha respondido afirmativamente a essa pergunta, tenho uma ótima notícia: você está na seção certa! Nesta seção vamos entender, conceitualmente e de maneira prática, como essa comunicação ocorre, quais são os elementos essenciais e como a própria ordem do código pode afetar uma aplicação, podendo abstrair esse conhecimento para diversas linguagens de programação orientadas a objeto. Na situação-problema ao qual você será exposto, terá que criar um serviço de bate-papo (popularmente conhecido pelo termo chat) para o sistema CRM que sua empresa está desenvolvendo. Portanto, crie uma aplicação CLI em Java que utiliza *sockets*, relacionando com os conceitos de aplicação Cliente e Servidor, de forma que o que for digitado no console (terminal) de uma instância servidora da sua aplicação seja replicada – ou seja, apareça na tela – de uma instância cliente da sua aplicação; e vice-versa.

Utilize o seu conhecimento atrelado ao conteúdo da seção para solucionar a situação-problema apresentada acima. Tenho certeza que você será capaz de passar por essa etapa, e com certeza os conhecimentos adquiridos abrirão sua mente para compreender, com um nível maior de detalhamento, várias aplicações que se comunicam via rede de computadores.

Não pode faltar

A comunicação entre máquinas em rede é essencial para diversas aplicações que utilizamos, tanto dentro de empresas, quanto no nosso dia a dia pessoal. Para facilitar o entendimento dos diferentes elementos envolvidos em uma comunicação em rede de computadores, frequentemente utilizamos o modelo de comunicação de 7 camadas ISO/OSI como referência (MAIA, 2013). Nesse modelo, a comunicação dos dados de uma máquina para outra ocorre a partir da quarta camada (a chamada camada de transporte).

Nessa camada, destacam-se dois protocolos muito utilizados para realizar tal comunicação: o protocolo TCP e o protocolo UDP. Apesar da maioria das comunicações utilizarem o protocolo TCP – razão pela qual utilizaremos esse protocolo de comunicação na nossa aplicação de bate-papo – não significa que o protocolo TCP seja melhor que o protocolo UDP. O que ocorre é que esses dois protocolos possuem finalidades diferentes, o que implica que não existe um protocolo "melhor".

Vamos analisar, primeiramente, o protocolo TCP, citar suas vantagens e desvantagens.

O Protocolo de Controle de Transmissão – TCP (do inglês, *Transmission Control Protocol*), é um protocolo utilizado como base para comunicação entre máquinas, onde, caso haja alguma perda de informação, durante a transmissão, aquela informação seja retransmitida (MAIA, 2013). Por essa razão, é dito que o TCP é um protocolo orientado à conexão. Assim, esse é o protocolo de escolha para comunicações que não necessitem de uma transmissão em tempo real, ou seja, que não são muito sensíveis a atrasos, já que essa retransmissão leva mais tempo para ser realizada.

Exemplificando

Um ótimo exemplo de aplicação que utiliza o TCP como protocolo de troca de informações é quando utilizamos o internet banking para realizar um transferência de uma conta para outra: se você analisar sobre a ótica da "orientação à conexão", não importa que demore um pouco mais para que as informações sejam enviadas e recebidas entre uma máquina cliente e uma máquina servidora, desde que a transmissão da informação seja garantida e, se necessário para tal, retransmitida.

Por outro lado, o protocolo UDP (do inglês, *User Datagram Protocol*), é um protocolo utilizado como base para comunicação entre máquinas nas quais é importante que o atraso entre o envio e o recebimento da mensagem seja minimizado. Por essa razão, é dito que o UDP é um protocolo que não é orientado à conexão (referido como *connectionless*). Assim, esse é o protocolo de escolha para comunicações que necessitem de uma transmissão em tempo real, ou seja, não podem ocorrer atrasos, já que esse protocolo não retransmite a informação, como é o caso para a maioria das aplicações de tempo real (*real time*).

Exemplificando

Um ótimo exemplo de aplicação que utiliza o UDP como protocolo de troca de informações é no contexto de um circuito de vídeo vigilância, também conhecido popularmente como circuito fechado de TV (CFTV). Se alguém estiver roubando uma loja, não queremos que um frame seja retransmitido caso haja alguma perda de pacotes pois, caso isso ocorra com frequência, teríamos imagens defasadas em relação ao assalto.

Acredito que agora que você sabe as principais características e diferenças entre os protocolos de comunicação TCP e UDP, consegue entender o porquê de utilizar o protocolo TCP em uma aplicação de bate-papo (chat), certo?

Agora podemos passar para o próximo assunto, a comunicação através de *sockets*.

Os *sockets* utilizam o TCP (ou UDP) para realizar a comunicação entre aplicações que estejam sendo executadas em um sistema operacional, razão pela qual essa comunicação é chamada de interprocessos. Além disso, os *sockets* abstraem, do programador, a necessidade de um aprofundamento nas camadas mais inferiores de comunicação (camadas 1 a 3 do modelo de referência ISO/OSI).

Assimile

Aplicações que estejam sendo executadas em um sistema operacional são chamadas de **processos.** Por exemplo, se você abre a calculadora de seu sistema operacional, ela será um processo em execução.

Para que essa abstração possa ocorrer, existem funcionalidades (por vezes chamadas de primitivas) que normalmente são fornecidas por qualquer implementação de *socket*; em qualquer linguagem de programação orientada a objetos, essas funcionalidades representam métodos, já implementados, em determinadas classes relativas à comunicação via rede. Um resumo dessas funcionalidades é apresentado na Quadro 4.1.

Quadro 4.1 | Primitivas de *sockets* TCP

Significado	Significado
Socket	Cria um novo terminal de comunicação.
Bind	Atrela um endereço IP local a um *socket*.
Listen	Aviso de que o socket está aceitando conexões.
Accept	Aguarda o recebimento de uma solicitação de conexão.
Connect	Ativamente tenta estabelecer conexão com um *socket*.
Send	Envia dados através de uma conexão previamente estabelecida.
Receive	Recebe dados através de uma conexão previamente estabelecida.
Close	Libera a conexão.

Fonte: adaptado de Tanenbaum e Steen (2008).

Em termos práticos, um *socket* é uma combinação de endereço IP e porta de comunicação. Conforme observa Coulouris *et al.* (2013), para um processo de enviar e receber mensagens, seu *socket* precisa estar atrelado a uma porta e a um endereço IP roteável na máquina onde esse processo está sendo executado.

Observe que existe uma ordem "natural" de chamada dessas primitivas, tanto por uma máquina atuando como cliente, quanto por uma máquina atuando como servidor, para que a comunicação entre essas máquinas possa ocorrer. Como veremos a seguir em nosso exemplo com a linguagem Java.

Do ponto de vista de um servidor, este deve executar a primitiva "socket", momento no qual o servidor cria um novo terminal de comunicação, utilizando o protocolo de transporte apropriado à aplicação (TCP ou UDP). Internamente o sistema operacional aloca os recursos necessários para que essa máquina possa enviar e receber mensagens obedecendo as regras daquele – previamente definido – protocolo de transporte. Então deve ser executada a primitiva "bind", responsável por associar um *socket* específico (conjunto de endereço IP e porta de comunicação, conforme mencionado anteriormente) ao servidor, de forma que o sistema operacional saiba dessa associação.

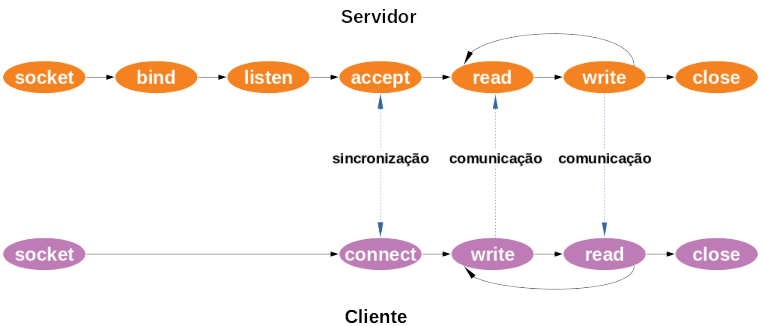
Reflita

Neste ponto imagino que você esteja começando a perceber as vantagens da abstração que um *socket* fornece ao programador/desenvolvedor de uma aplicação, em quais tipos de aplicações podemos utilizar um *socket*?

A seguir, deve ser executada a primitiva "listen", que faz com que o sistema operacional reserve recursos suficientes para ser capaz de lidar com a quantidade de conexões que o servidor suporta. A chamada à primitiva "accept" fica aguardando o estabelecimento de uma conexão por uma máquina cliente; quando esta ocorre, o sistema operacional retorna um *socket* com as mesmas características daquele criado pelo servidor, para a máquina cliente. Como veremos a seguir em nosso exemplo com a linguagem Java.

Agora vamos analisar as primitivas a serem executadas pela máquina cliente. Essa máquina também precisa criar um *socket*, mas não há necessidade de especificar uma porta de comunicação, pois esta é alocada automaticamente pelo sistema operacional, uma vez que seu número não precisa ser explicitamente especificado na máquina remota (diferentemente do que acontece com relação à porta de comunicação do lado do servidor). A primitiva "connect" recebe a informação de qual protocolo de transporte foi adotado pelo servidor e, ao ser executada, fica aguardando o estabelecimento de uma conexão com a máquina servidora. Uma vez que esta foi estabelecida, é possível utilizar as primitivas de "send" e "receive" para enviar e receber dados (ou seja, trocar informações ou mensagens entre as máquinas). Por fim, podemos executar a primitiva "close" para explicitamente encerrar a conexão e liberar os recursos eventualmente utilizados por esta, como por exemplo, a própria porta de comunicação. A Figura 4.5 foi criada a fim de ilustrar esse processo de comunicação de maneira mais integrada.

Figura 4.5 | Processo de comunicação via *sockets* TCP



Fonte: adaptada de Tanenbaum e Steen (2008).

Agora que você entendeu as etapas envolvidas no processo de comunicação de máquinas utilizando *sockets*, vamos analisar um código Java para uma aplicação – didática – de um serviço de bate-papo. Na Figura 4.6 é apresentado o código que seria compilado e executado no servidor, enquanto que na Figura 4.7 é apresentado o código que seria compilado e executado no cliente.

Figura 4.6 | Exemplo de código “Servidor.java”

```
1 import java.io.DataInputStream;
2 import java.io.DataOutputStream;
3 import java.io.IOException;
4 import java.net.ServerSocket;
5 import java.net.Socket;
6
7 public class Servidor {
8
9         public static void main(String[] args) {
10                Socket soc = null;
11                ServerSocket socServidor = null;
12                try {
13                        socServidor = new ServerSocket(5001);
14                        soc = socServidor.accept();
15                        DataInputStream recebido = new DataInputStream(soc.getInputStream());
16                        DataOutputStream enviado = new DataOutputStream(soc.getOutputStream());
17                        System.out.println("(cliente): " + recebido.readUTF());
18                        enviado.writeUTF("O servidor recebeu sua mensagem.");
19                } catch (IOException ex) {
20                        System.err.println("Falha na conexão");
21                } finally {
22                        try {
23                                soc.close();
24                                socServidor.close();
25                        } catch (IOException e) {
26                                System.err.println("Falha ao encerrar a conexão");
27                        }
28                }
29        }
30 }
```

Fonte: elaborada pelo autor.

Figura 4.7 | Exemplo de código “Cliente.java”

```
1 import java.io.DataInputStream;
2 import java.io.DataOutputStream;
3 import java.io.IOException;
4 import java.net.Socket;
5
6 public class Cliente {
7
8         public static void main(String[] args) {
9                 Socket soc = null;
10                try {
11                        soc = new Socket("127.0.0.1", 5001);
12                        DataInputStream recebido = new DataInputStream(soc.getInputStream());
13                        DataOutputStream enviado = new DataOutputStream(soc.getOutputStream());
14                        enviado.writeUTF("Aqui é um cliente falando...");
15                        System.out.println("(servidor): " + recebido.readUTF());
16                } catch (IOException ex) {
17                        System.err.println("Falha na inicializar o servidor");
18                } finally {
19                        try {
20                                soc.close();
21                        } catch (IOException e) {
22                                System.err.println("Falha ao encerrar a conexão");
23                        }
24                }
25        }
26 }
```

Fonte: elaborada pelo autor.

Em relação às Classes utilizadas e, analogamente à Figura 4.6, temos a primitiva “socket” implementada pelo objeto “soc” no código, tanto do servidor na linha 10, quanto do cliente na linha 9.

No código do servidor, a primitiva “bind” e “listen” é implementada pelo objeto “socServidor”, na linha 13. A primitiva “accept” é implementada pelo

método de mesmo nome, "accept()", disponível na Classe "ServerSocket" do Java, na linha 14. No código do cliente, a primitiva "connect" é implementada através da instanciação de um objeto Socket, onde informamos o endereço IP e porta do servidor, na linha 11.

Observe que o endereço IP utilizado no exemplo de código "Cliente.java" (Figura 4.7), na linha 11, "127.0.0.1", representa a comunicação com a própria máquina (comunicação esta chamada de *localhost*), o que significa que tanto a aplicação Servidor quanto a aplicação Cliente, neste caso em particular, estão sendo executadas em uma mesma máquina. A porta utilizada no exemplo, "5001" foi escolhida arbitrariamente, como uma das portas disponíveis (não utilizadas por outra aplicação quando da execução desse exemplo).

Tanto no cliente, nas linhas 12 e 13, quanto no servidor, nas linhas 15 e 16, as primitivas "read" e "write" são implementadas pelos objetos "recebido" e "enviado", respectivamente. Por fim, após a troca de mensagens entre as máquinas, é utilizada a primitiva "close", implementadas pelo método de mesmo nome, "close()", disponível nas Classes "Socket" e "ServerSocket" do Java, uma vez que ambas implementam a interface *Closeable*, do Java, isso ocorre nas linhas 23 e 24 da classe Servidor e na linha 20 da classe Cliente.

Esperamos que, a partir desse simples exemplo, você possa compreender e desmistificar os elementos envolvidos em uma comunicação via *socket*. A partir desse exemplo, tenho certeza de que você será capaz de criar aplicações mais elaboradas, pois os fundamentos necessários já estarão consolidados!

Pesquise mais

O Java possui uma vasta documentação, aconselho que você acesse ao menos as Classes utilizadas no exemplo apresentado para internalizar melhor os conceitos aqui apresentados, bem como entender o porquê da escolha de tais Classes e métodos que o autor fez:

- Documentação do Java. Disponível em: https://docs.oracle.com/javase/7/docs/api/overview-summary.html. Acesso em: 06 dez. 2018.

Você também pode fixar seu conhecimento assistindo esse vídeo que fala das portas TCP e UDP, com duração de 12 minutos, disponível no YouTube: Simplificando TI. O que são portas TCP e UDP? 05 nov. 2018. Disponível em: https://www.youtube.com/watch?v=SMU92puJxdU. Acesso em: 06 dez. 2018.

Você também pode fixar seu conhecimento através desse artigo, publicado pela Oracle, referente aos conceitos de *sockets*, trazendo um outro exemplo de implementação em Java:

Oracle. Lesson: all about sockets. Disponível em Documentação do Java. Disponível em: https://docs.oracle.com/javase/7/docs/api/overviewc -summary.html. Acesso em: 06 dez. 2018. Com esse conteúdo você pode enriquecer ainda mais seus conhecimentos e avançar para a próxima seção!

Finalizamos aqui mais uma seção, continue firme com os seus estudos e até a próxima!

Sem medo de errar

Na situação-problema proposta para essa seção, você deverá criar o serviço do chat para o seu sistema CRM. A proposta é desenvolver uma aplicação CLI (modo texto) em Java que utilize *sockets*, relacionando os conceitos de aplicação Cliente e Servidor, de forma que o que for digitado no console (terminal) de uma instância servidora da sua aplicação seja replicada – ou seja, apareça na tela – de uma instância cliente da sua aplicação; e vice-versa, nesta ordem.

A ideia aqui é que você possa aplicar os conceitos aprendidos de forma que tenha autonomia para modificar o código de exemplo da Figura 4.6 (servidor) e o código de exemplo da Figura 4.7 (cliente), para atender aos requisitos apresentados.

Como, primeiramente, o servidor envia uma mensagem para o cliente, para que este último responda, o que deve ser feito é simplesmente inverter a ordem de escrita/leitura de ambas aplicações, ou seja, inverter as linhas 17 e 18 do código "Servidor"; e inverter as linhas 14 e 15 do código "Cliente". Desta forma, a solução é apresentada na Figura 4.8 (servidor) e Figura 4.9 (cliente).

Figura 4.8 | Exemplo de código “Servidor.java”

```
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;

public class Servidor {

        public static void main(String[] args) {
                Socket soc = null;
                ServerSocket socServidor = null;
                try {
                        socServidor = new ServerSocket(5001);
                        soc = socServidor.accept();
                        DataInputStream recebido = new DataInputStream(soc.getInputStream());
                        DataOutputStream enviado = new DataOutputStream(soc.getOutputStream());
                        enviado.writeUTF("O servidor está escutando.");
                        System.out.println("(cliente): " + recebido.readUTF());
                } catch (IOException ex) {
                        System.err.println("Falha na conexão");
                } finally {
                        try {
                                soc.close();
                                socServidor.close();
                        } catch (IOException e) {
                                System.err.println("Falha ao encerrar a conexão");
                        }
                }
        }
}
```

Fonte: elaborada pelo autor.

Figura 4.9 | Exemplo de código “Cliente.java”.

```
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.Socket;

public class Cliente {

        public static void main(String[] args) {
                Socket soc = null;
                try {
                        soc = new Socket("127.0.0.1", 5001);
                        DataInputStream recebido = new DataInputStream(soc.getInputStream());
                        DataOutputStream enviado = new DataOutputStream(soc.getOutputStream());
                        System.out.println("(servidor): " + recebido.readUTF());
                        enviado.writeUTF("Aqui é o cliente falando...");
                } catch (IOException ex) {
                        System.err.println("Falha na inicializar o servidor");
                } finally {
                        try {
                                soc.close();
                        } catch (IOException e) {
                                System.err.println("Falha ao encerrar a conexão");
                        }
                }
        }
}
```

Fonte: elaborada pelo autor.

Após o término, você deve enviar as classes Java que foram criadas!

Programa de bate-papo no mundo real

Descrição da situação-problema

Caso você não tenha observado, o programa de exemplo apresentado até o momento funciona bem com apenas um cliente conectado, mas no mundo real, teremos vários processos ocorrendo simultaneamente na execução de uma aplicação. Isso significa que, tipicamente, teremos vários clientes conectados a um único servidor. Assim, a sua tarefa é utilizar um conceito de programação que permite que processos, dentro de uma mesma aplicação, possam ser executados de maneira paralela, fazendo com que nossa aplicação de bate-papo seja mais robusta e adequada para uso em um ambiente profissional.

Resolução da situação-problema

A solução de programação a ser utilizada aqui é o uso de ***threads***.

Aplicando esse conceito, uma possível solução é apresentada na Figura 4.10 (servidor) e Figura 4.11 (cliente).

Figura 4.10 | Exemplo de código “Servidor.java” com *threads*

```java
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Scanner;

public class Servidor extends Thread {
    public static void main(String[] args) {
        ServerSocket socServidor = null;
        Socket soc = null;
        try {
            socServidor = new ServerSocket(5001);
            while (true) {
                soc = socServidor.accept();
                Scanner teclado = new Scanner(System.in);
                DataInputStream recebido = new DataInputStream(soc.getInputStream());
                DataOutputStream enviado = new DataOutputStream(soc.getOutputStream());
                new Thread(() -> {
                    while (true) {
                        try {
                            System.out.println("(cliente): " + recebido.readUTF());
                            enviado.writeUTF(teclado.nextLine());
                        } catch (IOException e) {
                            e.printStackTrace();
                        }
                    }
                }).start();
            }
        } catch (IOException ex) {
            System.err.println("Falha na conexão");
        } finally {
            try {
                soc.close();
                socServidor.close();
            } catch (IOException e) {
                System.err.println("Falha ao encerrar a conexão");
            }
        }
    }
}
```

Fonte: elaborada pelo autor.

Figura 4.11 | Exemplo de código "Cliente.java" com *threads*

```java
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.Socket;
import java.util.Scanner;

public class Cliente {
    public static void main(String[] args) {
        Socket soc = null;
        try {
            soc = new Socket("127.0.0.1", 5001);
            DataInputStream recebido = new DataInputStream(soc.getInputStream());
            DataOutputStream enviado = new DataOutputStream(soc.getOutputStream());
            Scanner teclado = new Scanner(System.in);
            while (true) {
                enviado.writeUTF(teclado.nextLine());
                System.out.println("(servidor): " + recebido.readUTF());
            }
        } catch (IOException ex) {
            System.err.println("Falha na conexão");
        } finally {
            try {
                soc.close();
            } catch (IOException e) {
                System.err.println("Falha ao encerrar a conexão");
            }
        }
    }
}
```

Fonte: elaborada pelo autor.

Faça valer a pena

1. A comunicação via *sockets* é muito comum no uso de sistemas distribuídos. Assim sendo, é importante entender seus conceitos, de forma que sua implementação possa ser realizada sem percalços, independentemente da linguagem de programação utilizada.

Quais são os dois componentes principais, que precisam ser conhecidos, em uma comunicação via *sockets*?

a) Endereço IP e protocolo.
b) Protocolo e porta de comunicação.
c) Porta de comunicação e endereço MAC.
d) Protocolo TCP ou protocolo UDP.
e) Endereço IP e porta de comunicação.

2. A utilização de *sockets* em um sistema distribuído serve para que processos, em máquinas distintas, possam se comunicar entre si. Tipicamente, um servidor suporta um grande número de clientes conectados ao mesmo tempo, mas, para tal, o desenvolvedor deve utilizar um conceito muito útil em termos de programação, chamado de *threads*.

Nesse contexto, o uso de threads no código, tanto de um cliente, quanto de um servidor, em uma comunicação utilizando *sockets*, permite que:
( ) As conexões sejam independentes entre si.
( ) A troca de informações possa ser feita de maneira paralela.
( ) A troca de informações possa ser feita de maneira concorrente.
( ) As conexões estejam relacionadas entre si.

Assinale a alternativa que representa a sequência CORRETA.
a) V - V - V - F.
b) V - V - F - V.
c) V - F - F - V.
d) V - F - V - V.
e) F - V - F - V.

3. Muitas aplicações utilizam a comunicação via *sockets* para realizar troca de informações entre processos de diferentes máquinas, sejam essas máquinas sistemas embarcados (como as diversas placas com microcontroladores) ou entre cliente/servidor.

As Classes e métodos que devem ser utilizados na implementação de uma comunicação via *sockets* dependem:
I - Da linguagem de programação adotada.
II - Da versão da linguagem de programação.
III - Do tipo de sistema computacional.
IV - Do suporte a *sockets* estar disponível naquela linguagem de programação.

Assinale a alternativa CORRETA.
a) Apenas a afirmação IV está correta.
b) Apenas as afirmações I, II e IV estão corretas.
c) Apenas a afirmação III está correta.
d) Todas as afirmações estão corretas.
e) Apenas a afirmação I está correta.

Seção 4.3

Utilizando RPC com Java

Diálogo aberto

Caro aluno(a), seja bem-vindo(a) à última seção do nosso livro! Já pensou em quantos propósitos a linguagem de programação Java pode ser utilizada? Podemos utilizar essa popular linguagem com a biblioteca JRMI para implantar um RPC, este será o principal assunto dessa seção. Nesta seção vamos trabalhar com comunicação de máquinas cliente e servidor através de nossa aplicação Java, além disso vamos aprender a implementar um RPC utilizando a biblioteca Java RMI. Isso não é maravilhoso? Você já parou para pensar na importância de sistemas que possuem uma comunicação cliente e servidor? E como são utilizados em quase tudo hoje em dia?

Quando falamos de sistemas distribuídos é inevitável falar sobre essas tecnologias, pois na prática essas tecnologias são frequentemente utilizadas por empresas de diversos portes e ramos. Você concluiu a sua graduação na área de TI e agora pretende abrir sua própria empresa, desenvolvendo e comercializando um sistema CRM que possuirá várias funcionalidades inovadoras, mas também deve possuir as mesmas funcionalidades comumente presentes em sistemas de empresas concorrentes. Como você pretende que sua aplicação tenha compatibilidade com sistemas operacionais comumente encontrados nos laptops dos vendedores das empresas de seus futuros clientes, você decide utilizar a linguagem Java para desenvolver essa aplicação.

Até agora você já preparou seu novo sistema CRM fazendo uma prevenção contra vulnerabilidades de segurança, que geralmente são encontradas em sistemas distribuídos e estudando os possíveis ataques que o sistema pode sofrer e como se defender. Feito isso foi elaborado um chat para o sistema, utilizando Java e Sockets. Agora chegou a parte final, onde você deve criar uma funcionalidade inovadora em seu sistema de CRM na qual, caso os laptops dos vendedores dependam de recursos que estejam disponíveis apenas em algum servidor da empresa; como por exemplo para realizar uma pesquisa de relatórios combinados, a execução dessa pesquisa possa ocorrer de maneira eficiente. Para tal, você sabe que o uso de tecnologias de chamada de processos remotos pode ser uma alternativa interessante. Portanto, utilize o RPC via Java RMI para fazer uma consulta em uma lista de nomes, disponíveis em uma máquina que terá o papel de máquina servidora, com esse arquivo de texto, no formato .txt, e utilize ao menos três outras máquinas, que terão o papel de máquinas cliente (que estejam conectadas em rede

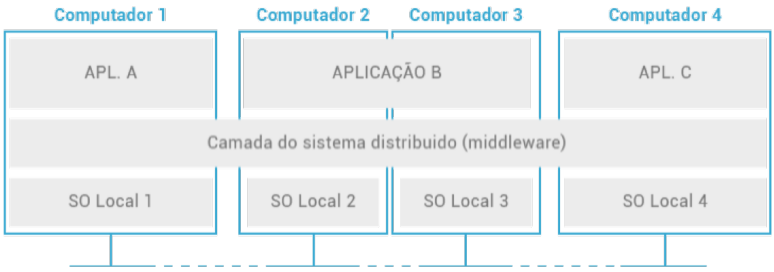
com o servidor), que farão uma chamada a um método remota, disponível apenas no servidor, de maneira que o objeto Java, que possibilita o uso desse método no código da máquina servidora, possa ser utilizado localmente nas máquinas clientes. Esse método remoto deve possuir a seguinte assinatura: `public String verificarNomes(String nome)`, e deve apresentar, no console (terminal) de cada máquina cliente, a quantidade de vezes que o nome informado aparece na lista.

Para completar esse desafio, nessa seção você verá, em detalhes, como se implantar RPC através de Java RMI, incluindo comandos específicos a serem utilizados. Ficou curioso? Espero que você se sinta motivado a dedicar seu tempo e seus esforços em um estudo que lhe proporcionará chances reais de assimilar os conceitos e práticas que você utilizará na sua vida profissional. Vamos lá!

Não pode faltar

Na comunicação entre máquinas em um sistema distribuído, é comum o uso de *middlewares* que, conforme foi estudado, servem como uma camada de abstração entre a chamada de métodos de alto nível – pelas aplicações – e a execução de métodos de baixo nível, dependentes do sistema operacional especificamente instalado naquela máquina. A Figura 4.12 representa essa camada de *middleware* no contexto de sistemas distribuídos.

Figura 4.12 | Camada de *middleware*



Fonte: elaborada pelo autor.

Podemos observar na Figura 4.12 a camada de *middleware* que está interligando aplicações e sistemas operacionais que estão sendo executados em diferentes computadores.

Esse *middleware* nada mais é o do que a implementação – através de algum *framework* – de um modelo de comunicação entre máquinas conhecido genericamente por RPC, do inglês *Remote Procedure Call*. Em outras palavras, o RPC é uma forma de comunicação entre máquinas mais granular

que a comunicação via *sockets* (COULOURIS, 2013). Como visto na Seção anterior, a comunicação via *sockets* envolve as camadas um a quatro do modelo de referência ISO/OSI. Já a comunicação via RPC envolve as sete camadas do modelo de referência ISO/OSI, e é uma forma mais granular de comunicação entre máquinas, uma vez que, diferentemente da comunicação via *sockets*, na qual executamos toda a aplicação, através do uso de RPC, podemos executar apenas um (ou mais) métodos de interesse, implementados em uma máquina, através de outra máquina.

Assimile

A ideia geral de qualquer RPC é que uma máquina, denotada como cliente, possa executar métodos que estejam implementados em outras máquinas, denotada como servidor, via rede, como se esse método tivesse sido implementado localmente – ou seja, na máquina cliente.

Há de se salientar que, apesar de, na prática, utilizarmos esse termo quando queremos nos referir a esse tipo de comunicação entre máquinas, o termo RPC em si é, tecnicamente, apenas uma das formas de comunicação entre máquinas com alta granularidade, conforme observa Coulouris *et al.* (2013), sendo, inclusive, a mais antiga delas (porém ainda utilizada). Ainda conforme Coulouris, existem três modelos de comunicação entre máquinas, conforme o Quadro 4.2.

Quadro 4.2 | Modelos de comunicação entre máquinas.

Modelo de Comunicação	Implementado através de
RPC (*Remote Procedure Call*)	Linguagens de programação estruturadas.
RMI (*Remote Method Invocation*)	Linguagens de programação orientadas a objeto.
MOM (*Message Oriented Midleware*)	Linguagens de programação para web.

Fonte: elaborado pelo autor.

Dependendo da linguagem de programação a qual o desenvolvedor tem mais familiaridade, existem vários *frameworks* que podem ser adotados para implementação do RPC. Abaixo são listados alguns dos mais utilizados, para algumas das linguagens de programação mais populares atualmente:

- **Java**:

JRMI <https://docs.oracle.com/javase/tutorial/rmi/overview.html>
JMS <https://www.oracle.com/technetwork/articles/java/introjms-1577110.html>

- **C#**:

.NET Remoting <https://msdn.microsoft.com/en-us/library/ms973857.aspx>
Akka.net https://getakka.net/articles/intro/tutorial-1.html

- **Python**:

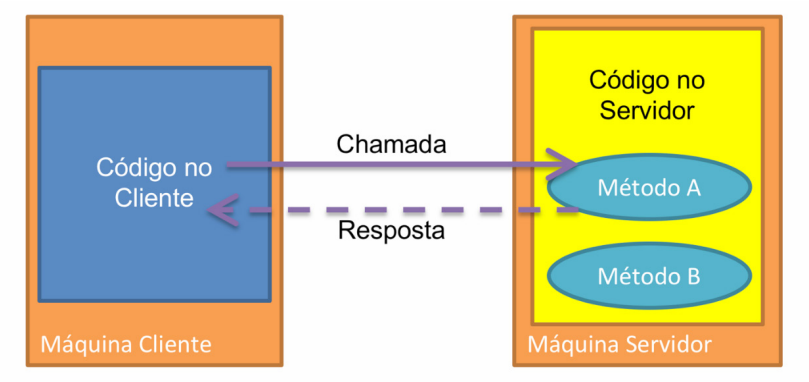
Pyro <https://pythonhosted.org/Pyro4/>
RPyC <https://rpyc.readthedocs.io/en/latest/>

- **JavaScript**:

Jason <https://www.npmjs.com/package/jayson>
Dnode <https://www.npmjs.com/package/dnode>

Como pode-se imaginar, cada um desses *frameworks* possui suas particularidades para implementá-lo, mas a maioria deles possui os componentes ilustrados na Figura 4.13.

Figura 4.13 | Componentes de uma comunicação via RPC



Fonte: elaborada pelo autor.

Para consolidar o entendimento do RPC, nós iremos realizar uma aplicação simples, utilizando o JRMI e, consequentemente, a linguagem Java. Nossa aplicação de exemplo será constituída de apenas um método remoto que será executado pelo cliente (embora, é claro, poderiam ser implementados mais métodos, conforme a necessidade). Esse método irá receber um nome, em letras minúsculas, e então retornará o mesmo nome, em letras maiúsculas.

Primeiramente, devemos criar uma interface onde iremos declarar os métodos que serão implementados no servidor e, desta forma, que poderão ser executados pelo cliente. Trata-se de uma interface comum, daquelas

que você provavelmente deve ter criado em alguma disciplina referente à Orientação a Objetos com Java. A Figura 4.14 apresenta um exemplo de código para essa interface em Java.

Figura 4.14 | Exemplo de código "Interface.java"

```
1 import java.rmi.Remote;
2
3 public interface Interface extends Remote {
4
5     public String converterParaMaiuscula(String palavra) throws Exception;
6
7 }
```

Fonte: elaborada pelo autor.

Como essa interface será implementada para chamadas remotas, através da biblioteca JRMI, devemos explicitamente informar que a mesma irá herdar as propriedades da Classe "Remote", do Java.

Além disso, como a execução desse método, quando implementado no servidor, será uma operação via rede, devemos alertar à *Java Virtual Machine* (JVM) que sua execução poderá gerar falhas, em tempo de execução, dependendo das condições da rede. Para simplificar o código, ao invés de tratarmos esse erro, simplesmente alertamos da ocorrência dele à JVM, através da instrução "throws Exception".

Agora devemos escrever o código que será executado no servidor. A Figura 4.15 apresenta um exemplo de código que implementa o método "converterParaMaiuscula(String palavra)" em Java.

Figura 4.15 | Exemplo de código "Servidor.java"

```
1  import java.rmi.Naming;
2  import java.rmi.registry.LocateRegistry;
3  import java.rmi.server.UnicastRemoteObject;
4
5  public class Servidor extends UnicastRemoteObject implements Interface {
6
7      public static void main(String[] args) {
8          try {
9              System.setProperty("java.rmi.server.hostname", "10.116.201.48");
10             LocateRegistry.createRegistry(1099);
11             Naming.rebind("//10.116.201.48/obj", new Servidor());
12         } catch (Exception e) {
13             System.err.println("Falha ao iniciar o servidor!");
14         }
15     }
16
17     public Servidor() throws Exception {
18     }
19
20     @Override
21     public String converterParaMaiuscula(String palavra) throws Exception {
22         return palavra.toUpperCase();
23     }
24 }
```

Fonte: elaborada pelo autor.

A implementação do método em si é bastante simples: ele recebe um parâmetro do tipo String, e retorna, para o cliente, essa mesma palavra, em letras maiúsculas, como pode ser visto na linha 22 da Figura 4.14. Mas, para que a chamada de métodos remotos funcione, precisamos de mais algumas informações em nosso código.

A primeira delas é informar qual tipo de comunicação será realizada entre o Servidor e a(s) máquina(s) Cliente: isso é feito através da herança da Classe "UnicastRemoteObject", apresentada na linha 5 da Figura 4.14. Essa Classe define que a comunicação será do tipo *unicast*.

Exemplificando

Você utiliza comunicação *unicast* quando envia/recebe mensagens diretamente de uma máquina para outra, ou seja, a chamada comunicação 1 para 1. Um exemplo típico é quando você faz uma chamada de voz (via aplicativo de mensagens instantâneas) para alguém, e então começam a conversar: esse tipo de comunicação é uma comunicação 1 para 1.

Depois, na linha 9 da Figura 4.15, definimos o valor da *hostname* para o endereço IP da máquina. É importante observar que esse endereço deve ser o endereço IP da máquina onde o código "Servidor.java" será executado. No meu caso, o endereço IP da minha máquina era "10.116.201.48".

Na linha 10 da Figura 4.15, definimos qual a porta de comunicação será utilizada. Por padrão de mercado, a porta TCP 1099 é utilizada para comunicação através de JRMI.

Por fim, na linha 11 da Figura 4.15, utilizamos o método "rebind", da Classe "Naming", para definirmos os parâmetros para criação do objeto que será compartilhado remotamente (e que representa a Classe em si – no meu caso, a Classe "Servidor") com um endereço IP. Os parâmetros que definimos são:

- Endereço IP: no meu caso, o endereço da minha máquina era 10.116.201.48.
- Nome do objeto que será compartilhado: no meu caso escolhi, arbitrariamente, o nome "obj".
- Objeto da Classe em questão: no meu caso, utilizei um construtor "vazio" para instanciar um objeto da minha Classe "Servidor", como pode ser visto nas linhas 17 e 18 da Figura 4.15.

É importante se atentar a esses parâmetros pois, caso queira rodar esse código, provavelmente você precisará alterar essas informações. Como por exemplo o parâmetro do endereço de IP, a execução do programa foi feita em minha máquina, portanto utilizei o endereço de IP 10.116.201.48, o mesmo deve ser alterado para o endereço da máquina que você estiver utilizando para a execução.

Reflita

Você já deve saber que o Java adiciona um construtor vazio nas suas Classes por padrão, caso você não adicione um explicitamente. Considerando esse comportamento, você consegue, analisando o meu código da Figura 4.15, entender o porquê de eu ter adicionado um construtor vazio para a minha Classe "Servidor"? Dica: aquele construtor vazio não foi adicionado desnecessariamente.

Por fim, basta escrevermos o código que será executado na(s) máquina(s) Cliente. A Figura 4.16 apresenta um exemplo de código para esse código.

Figura 4.16 | Exemplo de código "Cliente.java"

```
1 import java.rmi.Naming;
2 import java.util.Scanner;
3
4 public class Cliente {
5
6     public static void main(String[] args) throws Exception {
7
8         Interface objetoRemoto = (Interface) Naming.lookup("//10.116.201.48/obj");
9
10        System.out.println(objetoRemoto.converterParaMaiuscula(
11                                        new Scanner(System.in).nextLine())
12                        );
13    }
14 }
```

Fonte: elaborada pelo autor.

No Cliente, a parte principal é "puxarmos" a referência do objeto da Classe "Servidor" para Classe Cliente, ou seja, para a máquina local. Isso é feito através do método "lookup". Note que esse método recebe um único parâmetro, do tipo *String*, e ele deve ser idêntico às informações de endereço IP e nome do objeto que foram inseridas o código do Servidor. Feito isso, já podemos utilizar o objeto como se existisse localmente, dentro do código Cliente, através do nome atribuído neste código (eu escolhi o nome "objetoRemoto") para chamar e utilizar o método "converterParaMaiúscula". Se você seguiu as instruções apresentadas até aqui corretamente, ao inserir uma palavra (ou frase) na máquina Cliente, essa palavra (ou frase) será impressa no console, em letras maiúsculas.

Pesquise mais

O Java possui uma vasta documentação, aconselho que você acesse ao menos as Classes utilizadas no exemplo apresentado para internalizar melhor os conceitos aqui apresentados, bem como entender o porquê da escolha de tais Classes e métodos que o autor fez:

- Documentação do Java. Disponível em: https://docs.oracle.com/javase/7/docs/api/overview-summary.html. Acesso em: 11 dez. 2018.

- Exemplo de criação de RPC através do JRMI. Disponível em: https://docs.oracle.com/javase/7/docs/technotes/guides/rmi/hello/hello-world.html. Acesso em: 11 dez. 2018.

Caro aluno, assim finalizamos nossa última seção de estudos sobre sistemas distribuídos, espero que durante essa jornada do conhecimento você tenha aprendido muitas coisas novas e valorosas. Que as tecnologias estudadas estejam presentes no seu dia a dia profissional e você se torne um profissional de muito sucesso, parabéns por chegar até aqui! Boa sorte.

Sem medo de errar

Agora chegou a parte final do desenvolvimento do seu sistema de CRM. Para a criação do mesmo, utilize o RPC via Java RMI para fazer uma consulta em uma lista de nomes disponíveis em uma máquina que terá o papel de máquina servidora, com esse arquivo de texto, no formato .txt, e utilize ao menos três outras máquinas, que terão o papel de máquinas cliente (que estejam conectadas em rede com o servidor), que farão uma chamada a um método remoto, disponível apenas no servidor.

Esse método remoto deve possuir a seguinte assinatura: public String verificarNomes(String nome), e deve apresentar, no console (terminal) de cada máquina cliente, a quantidade de vezes que o nome informado aparece na lista.

Você pode resolver essa situação-problema da seguinte maneira:

Devemos criar três classes Java: “Interface.java”, “Servidor.java” e “Cliente.java”.

O código da classe Java interface pode ser consultado a seguir:

Figura 4.17 | Código da classe "Interface.java"

```
1 import java.rmi.Remote;
2
3 public interface Interface extends Remote {
4
5     public String verificarNomes(String nome) throws Exception;
6
7 }
```

Fonte: elaborada pelo autor.

O código da classe Java Servidor pode ser consultado nas duas próximas imagens:

Figura 4.18 | Código da classe "Servidor.java", parte 1

```
1 import java.io.BufferedReader;
2 import java.io.FileReader;
3 import java.rmi.Naming;
4 import java.rmi.registry.LocateRegistry;
5 import java.rmi.server.UnicastRemoteObject;
6 import java.util.HashMap;
7 import java.util.Map;
8 import java.util.regex.Matcher;
9 import java.util.regex.Pattern;
10
11 public class Servidor extends UnicastRemoteObject implements Interface {
12
13     public static void main(String[] args) {
14         try {
15             System.setProperty("java.rmi.server.hostname", "10.116.201.48");
16             LocateRegistry.createRegistry(1099);
17             Naming.rebind("//10.116.201.48/obj", new Servidor());
18         } catch (Exception e) {
19             System.err.println("Falha ao iniciar o servidor!");
20         }
21     }
22
23     public Servidor() throws Exception {
24     }
25
26     @Override
27     public String verificarNomes(String nome) throws Exception {
28         return contador("lista.txt", nome);
29     }
30
31     private String contador(String lista, String nome) throws Exception {
32
33         Map<String, Integer> mapa = new HashMap<>();
34         String resultado;
35         try (BufferedReader bf = new BufferedReader(new FileReader(lista))) {
36             resultado = null;
37             String linha = bf.readLine();
```

Fonte: elaborada pelo autor.

Figura 4.19 | Código da classe “Servidor.java”, parte 2

```
38              while (linha != null) {
39                  Matcher padrao = Pattern.compile("(\\d+)|([a-záéíóúçãõôê]+)")
40                                                  .matcher(linha.toLowerCase());
41                  while (padrao.find()) {
42                      Integer freq = mapa.get(padrao.group());
43                      if (freq != null) {
44                          mapa.put(padrao.group(), freq + 1);
45                      } else {
46                          mapa.put(padrao.group(), 1);
47                      }
48                  }
49                  linha = bf.readLine();
50              }
51          }
52          for (Map.Entry<String, Integer> entrada : mapa.entrySet()) {
53              if (entrada.getKey().equals(nome.toLowerCase())) {
54                  resultado =  entrada.getKey() + ": " + entrada.getValue();
55                  break;
56              } else
57                  resultado = nome + ": 0";
58          }
59          return resultado;
60      }
61  }
```

Fonte: elaborada pelo autor.

O código da classe Java Cliente pode ser consultado na Figura 4.20, ela deve ser adicionada nas três máquinas cliente:

Figura 4.20 | Código da classe “Cliente.java”

```
1  import java.rmi.Naming;
2
3  public class Cliente {
4
5      public static void main(String[] args) throws Exception {
6
7          Interface objetoRemoto = (Interface) Naming.lookup("//10.116.201.48/obj");
8
9          System.out.println(objetoRemoto.verificarNomes("Souza"));
10     }
11 }
```

Fonte: elaborada pelo autor.

Você só deve ficar atento pois os endereços de IPs que eu utilizei na minha resolução de problema representam as máquinas do ambiente em que eu estava trabalhando, no seu caso você deve alterá-los para os respectivos endereços de IPs do seu ambiente.

Assim finalizamos a situação-problema, e tenho certeza que o sistema CRM está cada vez melhor.

Avançando na prática

Processo seletivo para DevOps

Descrição da situação-problema

Você estava procurando por uma nova oportunidade de trabalho e resolveu se candidatar a uma vaga de DevOps Junior que encontrou na internet para trabalhar em uma empresa de consultoria. Você percebeu que estava apto para participar deste processo pois a maior parte dos conhecimentos exigidos são de RPC implementados com Java RMI. Você foi chamado para a entrevista e ao chegar na empresa foi submetido a um teste prático onde você deverá criar um método do tipo *String*, para converter palavras recebidas de uma máquina Cliente para caixa baixa (letras minúsculas) através de uma máquina servidor, RPC com Java RMI.

Resolução da situação-problema

Podemos resolver o exercício proposto pelo entrevistador da seguinte forma:

Devemos criar três classes Java: "Interface.java", "Servidor.java" e "Cliente.java".

O código da classe Java interface pode ser consultado a seguir:

Figura 4.21 | Código da classe "Interface.java"

```
1 import java.rmi.Remote;
2
3 public interface Interface extends Remote {
4
5     public String converterParaMinuscula(String palavra) throws Exception;
6
7 }
```

Fonte: elaborada pelo autor.

Figura 4.22 | Código da classe “Servidor.java”

```
import java.rmi.Naming;
import java.rmi.registry.LocateRegistry;
import java.rmi.server.UnicastRemoteObject;

public class Servidor extends UnicastRemoteObject implements Interface {

    public static void main(String[] args) {
        try {
            System.setProperty("java.rmi.server.hostname", "10.116.201.48");
            LocateRegistry.createRegistry(1099);
            Naming.rebind("//10.116.201.48/obj", new Servidor());
        } catch (Exception e) {
            System.err.println("Falha ao iniciar o servidor!");
        }
    }

    public Servidor() throws Exception {
    }

    @Override
    public String converterParaMinuscula(String palavra) throws Exception {
        return palavra.toLowerCase();
    }
}
```

Fonte: elaborada pelo autor.

O código da classe Java Cliente pode ser consultado na Figura a seguir:

Figura 4.23 | Código da classe “Cliente.java”

```
import java.rmi.Naming;
import java.util.Scanner;

public class Cliente {

    public static void main(String[] args) throws Exception {

        Interface objetoRemoto = (Interface) Naming.lookup("//10.116.201.48/obj");

        System.out.println(objetoRemoto.converterParaMinuscula(
                                    new Scanner(System.in).nextLine())
                           );
    }
}
```

Fonte: elaborada pelo autor.

Assim solucionamos nossa situação-problema.

Faça valer a pena

1. Utilizamos o termo RPC quando queremos nos referir a um tipo de comunicação entre máquinas. Esse termo, tecnicamente, é apenas uma das formas de comunicação entre máquinas com alta granularidade, conforme observa Coulouris *et al.* (2013), sendo, inclusive, a mais antiga delas (porém ainda muito utilizada). Ainda conforme Coulouris, existem três modelos de comunicação entre máquinas: RPC, RMI e MOM.

Assinale a alternativa que contém o significado correto das siglas RPC, RMI e MOM.
a) *Remote Procedure Call, Remote Midleware Invocation* e *Message Oriented Method*.
b) *Remote Process Call, Remote Method Invocation* e *Message Oriented Midleware*.
c) *Remote Procedure Call, Remote Midleware Invocation* e *Message Oriented Midleware*.

d) *Remote Process Client, Remote Minute Invite* e *Message Over Match*.
e) *Remote Procedure Call, Remote Method Invocation* e *Message Oriented Midleware*.

2. Dependendo da linguagem de programação a qual o desenvolvedor tem mais familiaridade, existem vários *frameworks* que podem ser adotados para implementação do RPC. Podemos apontar pelo menos um tipo *framework* para cada uma das linguagens mais populares atualmente.

Nesse contexto, analise as afirmativas abaixo e identifique quais delas são verdadeiras ou falsas:
( ) O JRMI e o JMS são *frameworks* de implementação do RPC através da linguagem Java.
( ) O .NET Remoting e o Dnode são *frameworks* de implementação do RPC através da linguagem C#.
( ) O Pyro e o RPyC são *frameworks* de implementação do RPC através da linguagem Python.
( ) A linguagem JavaScript não possui nenhum *framework* de implementação do RPC.

Assinale a alternativa que representa a sequência CORRETA.
a) V - V - V - F.
b) V - V - V - V.
c) V - F - V - F.
d) V - F - F - V.
e) F - V - F - V.

3. A linguagem Java é uma das mais populares do mundo. Ela pode ser utilizada para o desenvolvimento das mais diversas aplicações, como por exemplo um RPC que pode ser implementado em Java através da biblioteca JRMI. Você já deve saber que o Java adiciona um construtor vazio nas suas Classes por padrão, caso você não adicione um explicitamente.

Analisando o código da Figura 4.24, assinale a alternativa que corresponde ao motivo em que foi adicionado um construtor vazio para a Classe "Servidor". O construtor pode ser localizado nas linhas 17 e 18 do código.

Figura 4.24 | Exemplo de código “Servidor.java”

```java
import java.rmi.Naming;
import java.rmi.registry.LocateRegistry;
import java.rmi.server.UnicastRemoteObject;

public class Servidor extends UnicastRemoteObject implements Interface {

    public static void main(String[] args) {
        try {
            System.setProperty("java.rmi.server.hostname", "10.116.201.48");
            LocateRegistry.createRegistry(1099);
            Naming.rebind("//10.116.201.48/obj", new Servidor());
        } catch (Exception e) {
            System.err.println("Falha ao iniciar o servidor!");
        }
    }

    public Servidor() throws Exception {
    }

    @Override
    public String converterParaMaiuscula(String palavra) throws Exception {
        return palavra.toUpperCase();
    }
}
```

Fonte: elaborada pelo autor.

a) O construtor foi adicionado desnecessariamente à classe.
b) O construtor foi adicionado para fazer o tratamento de exceções que não é feito no construtor vazio padrão do Java.
c) O construtor foi adicionado para a execução de métodos específicos.
d) O construtor foi adicionado para receber os dados de um objeto.
e) O construtor foi adicionado para que os erros do tipo “exceptions” sejam ocultados.

Referências

COULOURIS, G. *et al*. **Sistemas Distribuídos**. Porto Alegre: Bookman, 2013.

GUILHERME NETO. **O que é e como conseguir um certificado digital.** 2012. Disponível em: https://www.techtudo.com.br/artigos/noticia/2012/07/o-que-e-e-como-conseguir-um-certificado-digital.html. Acesso em: 18 dez. 2018.

GOODRICH, M. T.; TAMASSIA, R. **Introdução à Segurança de Computadores**. Porto Alegre: Bookman, 2012.

HAVE I BEEN PWNED. Disponível em: https://haveibeenpwned.com/. Acesso em: 27 nov. 2018.

JAVA (Portal) Documentação. Disponível em: https://docs.oracle.com/javase/7/docs/api/overview-summary.html. Acesso em: 11 dez. 2018.

MAIA, L. P. **Arquitetura de redes de computadores**. Rio de Janeiro: LTC, 2013.

ORACLE. **Java™ Platform, Standard Edition 7 API Specification.** Disponível em: https://docs.oracle.com/javase/7/docs/api/overview-summary.html. Acesso em: 30 jan. 2019.

TANENBAUM, A. S; STEEN, M. V. **Sistemas Distribuídos** - Princípios e Paradigmas. 2. ed. São Paulo: Pearson, 2008.

ISBN 978-85-522-1443-4