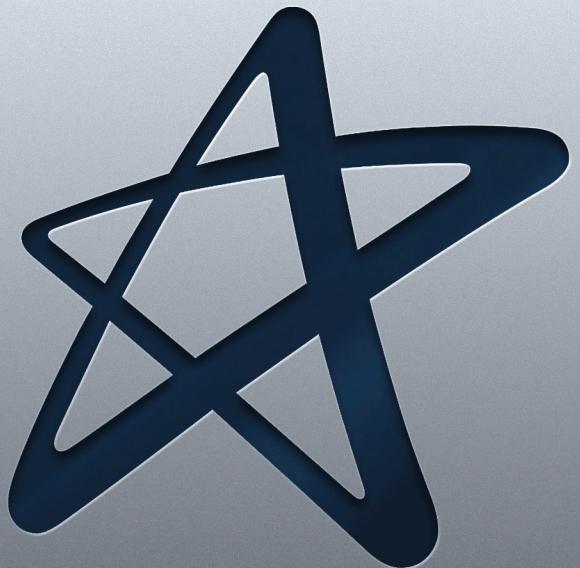


Arquitetura de Sistemas Distribuídos



Cruzeiro do Sul Virtual
Educação a distância

Material Teórico



Modelos de Aplicação

Responsável pelo Conteúdo:

Prof. Me. Max D'Angelo Pereira

Revisão Textual:

Prof. Me. Luciano Vieira Francisco

UNIDADE

Modelos de Aplicação



- **Introdução;**
- **Modelos de Arquitetura;**
- **Cliente-Servidor;**
- **Peer-to-Peer.**



OBJETIVO DE APRENDIZADO

- Diferenciar os principais modelos de aplicação e contextualizar os mesmos em situações práticas.

Orientações de estudo

Para que o conteúdo desta Disciplina seja bem aproveitado e haja maior aplicabilidade na sua formação acadêmica e atuação profissional, siga algumas recomendações básicas:

Determine um horário fixo para estudar.

Mantenha o foco! Evite se distrair com as redes sociais.

Procure manter contato com seus colegas e tutores para trocar ideias! Isso amplia a aprendizagem.

Seja original! Nunca plágie trabalhos.

Aproveite as indicações de Material Complementar.

Conserve seu material e local de estudos sempre organizados.

Não se esqueça de se alimentar e de se manter hidratado.

Assim:

- ✓ Organize seus estudos de maneira que passem a fazer parte da sua rotina. Por exemplo, você poderá determinar um dia e horário fixos como seu “momento do estudo”;
- ✓ Procure se alimentar e se hidratar quando for estudar; lembre-se de que uma alimentação saudável pode proporcionar melhor aproveitamento do estudo;
- ✓ No material de cada Unidade, há leituras indicadas e, entre elas, artigos científicos, livros, vídeos e sites para aprofundar os conhecimentos adquiridos ao longo da Unidade. Além disso, você também encontrará sugestões de conteúdo extra no item **Material Complementar**, que ampliarão sua interpretação e auxiliarão no pleno entendimento dos temas abordados;
- ✓ Após o contato com o conteúdo proposto, participe dos debates mediados em fóruns de discussão, pois irão auxiliar a verificar o quanto você absorveu de conhecimento, além de propiciar o contato com seus colegas e tutores, o que se apresenta como rico espaço de troca de ideias e de aprendizagem.

Introdução

Na Unidade anterior você aprendeu que um sistema distribuído, em sua definição mais simples, é um grupo de computadores trabalhando juntos para funcionar como um único computador ao usuário final. Essas máquinas têm um estado compartilhado, operam simultaneamente e podem falhar independentemente, sem afetar o tempo de atividade de todo o sistema.

Você já entendeu os conceitos básicos de sistemas distribuídos, agora, contudo, examinaremos a perspectiva de forma incremental, de modo a conhecer os modelos de sistemas distribuídos e, por meio de exemplos de aplicações, enriquecer o nosso conhecimento.



Será que todo sistema distribuído pode ser implementado da mesma forma? Ou existem diferentes modelos de sistemas distribuídos?



Figura 1

Fonte: Getty Images

Ao longo desta Unidade você será capaz de compreender os diferentes modelos de arquitetura de um sistema distribuído e relacionar com aplicações existentes em seu dia a dia. Além disso, conhecerá o conceito de *middleware* e as principais soluções existentes para esse importante componente de sistemas distribuídos.

Modelos de Arquitetura

O modelo de arquitetura de um sistema distribuído envolve o posicionamento de suas partes e os relacionamentos entre as quais. Cada tipo de modelo é destinado a fornecer uma descrição abstrata e simplificada, mas consistente, de um aspecto relevante do projeto de um sistema distribuído.

Você deve entender os modelos de arquitetura como as referências para o projeto de *software*, aplicação ou sistema. Então, você poderá concluir que a escolha do modelo de arquitetura mais adequado ao seu sistema ou à sua aplicação será aquele que conseguir cumprir os atributos de qualidade e atender aos requisitos de *software*.



Importante!

Não existe uma “receita” para definir o modelo a ser adotado; o importante é que você conheça as diferenças e características dos modelos arquitetônicos.

Quando um sistema de *software* é projetado, deve garantir que a arquitetura do sistema atenda à sua atual e futura demanda em **termos de atributos de qualidade e requisitos do software**.



A ISO/IEC 25010:2011 é uma norma ISO para a qualidade de produto de *software*, definindo um conjunto de parâmetros com o objetivo de padronizar a avaliação da qualidade de *software*. Essa norma substituiu a antiga ISO/IEC 9126.

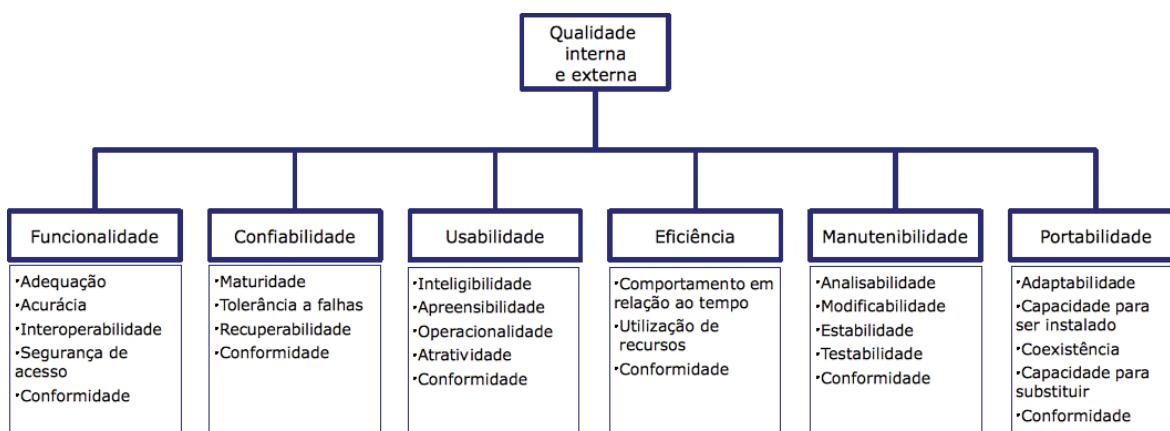


Figura 2 – Atributos de qualidade de *software* (ISO/IEC 9126)

Fonte: Wikimedia Commons



Especificação de requisitos de *software*: uma especificação de requisitos de *software* é uma descrição de um sistema de *software* a ser desenvolvido. A especificação de requisitos de *software* estabelece requisitos funcionais e não funcionais, podendo incluir um conjunto de casos de uso que descrevem as interações do usuário que o *software* deve fornecer. A especificação de requisitos de *software* estabelece a base para um acordo entre clientes e contratados – ou fornecedores – sobre como o produto de *software* deve funcionar.

Esses **modelos de arquitetura** apresentam um sistema em função das tarefas e dos objetivos computacionais e de comunicação realizados por seus componentes. Pense que aqui o foco será a divisão de responsabilidades entre os componentes do sistema e a alocação física desses componentes à infraestrutura de rede.

Conceitos Básicos



Rede de Comunicação: um sistema que conecta componentes finais, intermediários e outros dispositivos que permitem a troca de informações. Uma rede pode ser de qualquer tamanho, desde apenas dois componentes a muitos dispositivos;

Componente ou Sistema Intermediário: um dispositivo que opera como um elemento de ligação entre dois ou mais componentes finais – por exemplo, roteador, *switch*, repetidor;

Componente ou Sistema Final: um dispositivo que utiliza ou fornece aplicação ou serviços de rede para usuários finais – por exemplo, uma estação de trabalho, um *smartphone*, um servidor *web*, um servidor DNS.

São rotulados como **sistemas finais** porque ficam nas extremidades de uma rede. Os sistemas finais conectados à *internet* também são chamados de *hosts* – porque podem ser hospedeiros (*hosts*) de aplicações para a *internet*.

Pense que os sistemas finais podem ser posicionados em uma rede de maneiras diferentes em relação aos outros. Ou seja, podem se comunicar e compartilhar recursos de acordo com específicos modelos de interação.

Segundo Coulouris e colaboradores (2013), os modelos de arquitetura podem ser classificados em dois estilos básicos de interação:

- **Cliente-servidor** – *client-server*;
- **Ponto a ponto** – *Peer-to-Peer* (P2P).

Você verá que atualmente é comum encontrar modelos que são variações ou combinações dos dois modelos básicos, vejamos:

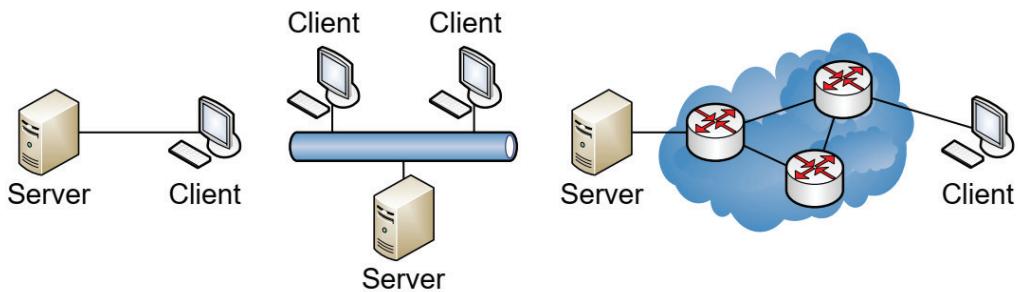


Figura 3 – Exemplos de modelo de arquitetura *client-server*

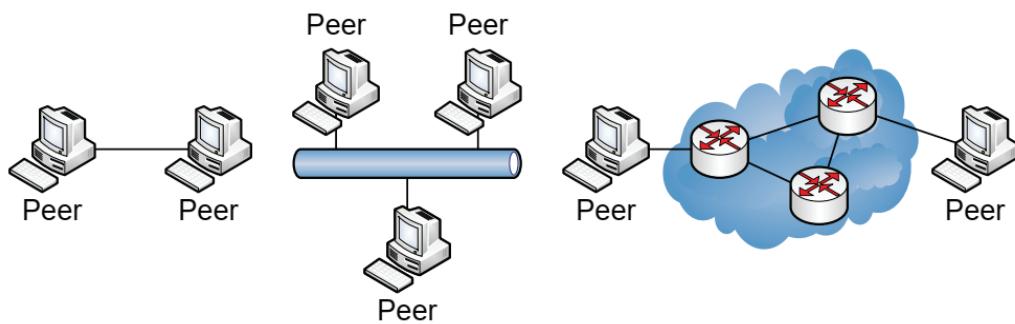


Figura 4 – Exemplos de modelo de arquitetura *peer-to-peer*



Trocando ideias...

Neste ponto você deve fazer uma relação com outras disciplinas de redes de computadores. Os modelos cliente-servidor e P2P operam na camada de aplicação do modelo TCP/IP.

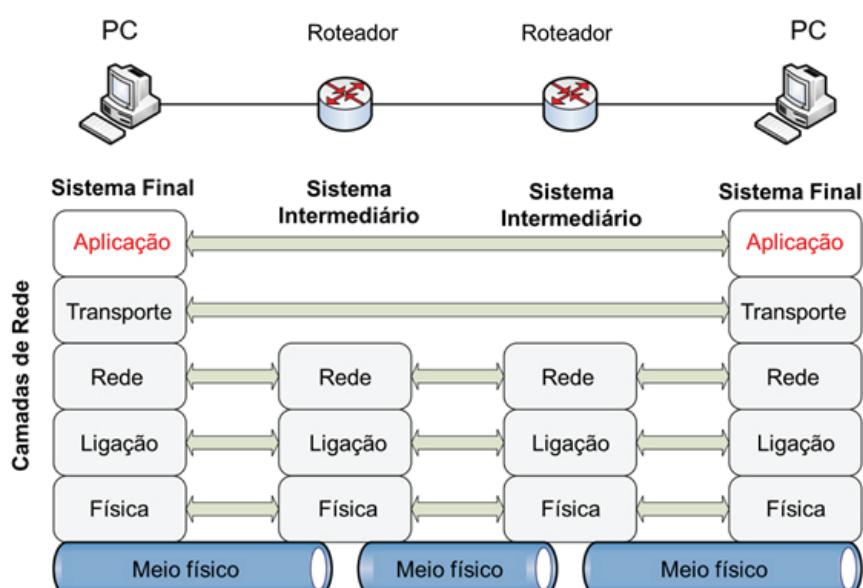


Figura 5 – Camadas de redes



Você pode verificar um resumo do modelo de camadas de redes, igualmente conhecido como modelo OSI. Disponível em: <http://bit.ly/326li4c>

Os sistemas cliente-servidor e P2P são implementados como redes virtuais, com nós e *links* lógicos, construídos sobre uma rede existente – comumente a *internet* ou redes locais (LAN).

Essas redes virtuais são chamadas de **redes de sobreposição** – *overlay network*. A sobreposição é uma exibição lógica que pode não refletir diretamente a topologia de rede física.

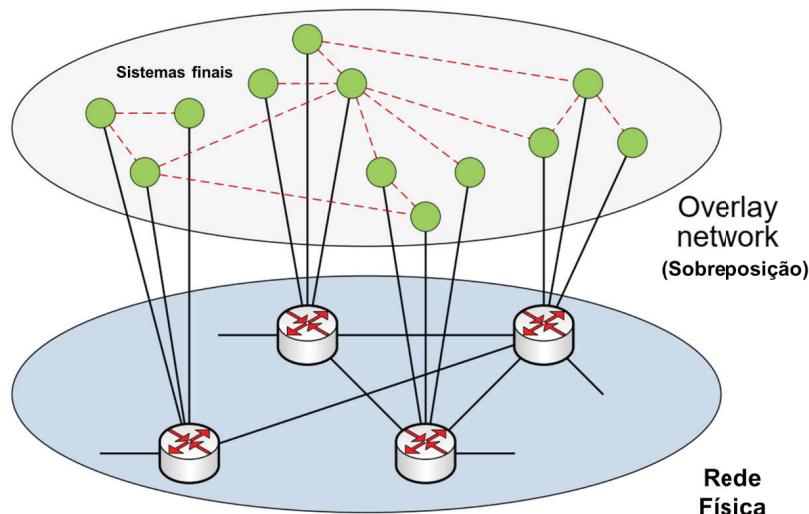


Figura 6 – Rede de sobreposição *versus* rede física



Os modelos de arquitetura que você aprenderá na sequência são relativos às redes de sobreposição – *overlay network*.

Middleware

Para sustentar computadores e redes variadas e, ao mesmo tempo, oferecer uma visão de sistema único, os sistemas distribuídos costumam ser organizados por meio de uma camada de *software*, comumente chamada de *middleware* (TANENBAUM; VAN STEEN, 2007).

A camada de *middleware* é situada logicamente entre uma camada de nível mais alto – composta de aplicações – e uma camada abaixo, que consiste em sistemas operacionais e componentes básicos de comunicação – veja uma ilustração deste conceito:



Figura 7 – Exemplo de camadas entre o *middleware*

Fonte: Coulouris e colaboradores, 2013

Middleware pode ser entendido como uma camada de *software* que não é uma aplicação propriamente dita e que não faz parte do sistema operacional. Essa camada esconde detalhes de dispositivos de *hardware* e *software* para fornecer uma interface mais simples de programar as aplicações.

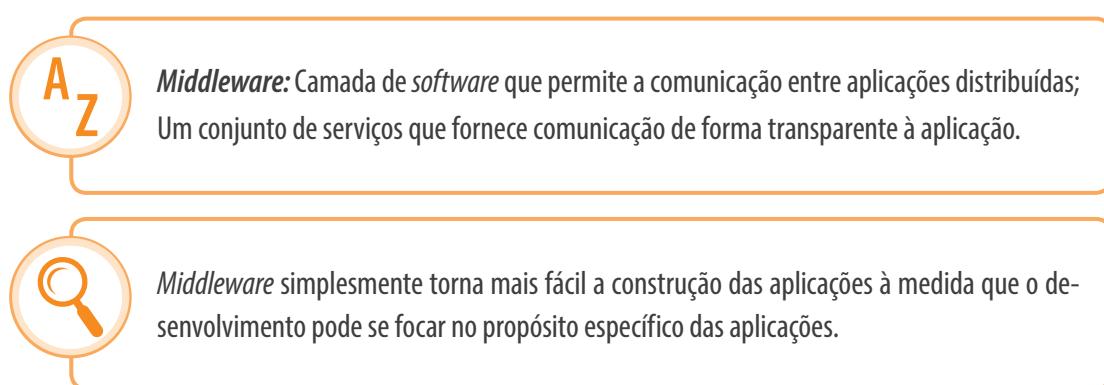


Figura 8 – Esquema conceitual de interação do *middleware* com o sistema operacional

Fonte: Silva, 2017

Middleware pode ser definido como os serviços encontrados acima da camada de transporte – ou seja, sobre TCP/IP –, mas abaixo da aplicação – logo, abaixo das **API**, no nível do aplicativo.

A
Z

API: é um conjunto de rotinas e padrões de programação ao acesso a um aplicativo de *software*. A sigla API refere-se ao termo, em inglês, *Application Programming Interface* – em português, interface de programação de aplicativos.

Cliente-Servidor

O modelo cliente-servidor deve ser entendido por meio da divisão das responsabilidades entre os componentes do sistema e de acordo com dois papéis bem definidos:

- **Servidores** – *Servers*: responsáveis por gerenciar e controlar o acesso aos recursos do sistema;
 - » Têm papéis passivos;
 - » Respondem aos seus clientes agindo em cada solicitação e retornando resultados;
 - » Geralmente suportam vários clientes.
- **Clientes** – *Clients*: são as interfaces por onde os usuários interagem com os servidores de modo a terem acesso aos recursos que estes gerenciam.
 - » Têm funções ativas e iniciam cada sessão de comunicação enviando solicitações para servidores;
 - » Devem ter conhecimento dos servidores disponíveis e serviços que estes fornecem;
 - » Comunicam-se apenas com os servidores;
 - » Não podem comunicar-se entre si.

Funções de Hardware

Os termos *cliente* e *servidor* geralmente se referem às funções primárias desempenhadas pelo *hardware* em rede.

Um *cliente* geralmente é algo como um computador pessoal (PC), *notebook* ou *smartphone* utilizado por um indivíduo e de maneira geral inicia conversas envian- do solicitações.



Figura 9 – Exemplos de dispositivos comuns no papel de cliente

Fonte: Getty Images

Um *servidor* é, geralmente, uma máquina poderosa dedicada a responder a solicitações de clientes; localizado, na maioria das vezes, geograficamente distante do *cliente*.

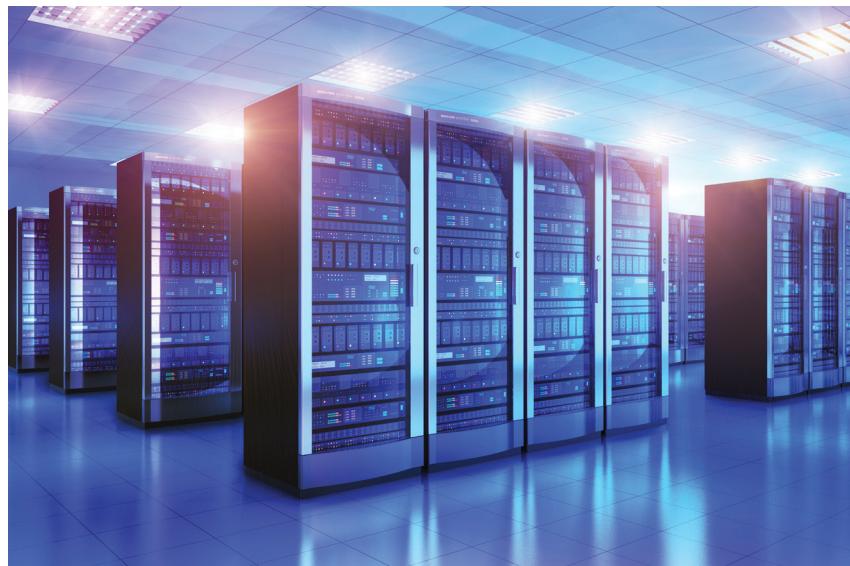


Figura 10 – Exemplo de um *data center* comum no papel de servidor

Fonte: Getty Images

Funções de *Software*

O TCP/IP utiliza diferentes partes de *software* para muitos protocolos a fim de implementar funções de *cliente* e *servidor*. O *software* do cliente é comumente encontrado no *hardware* do cliente; o *software* do servidor será encontrado no *hardware* do servidor.



Importante!

A regra mencionada não vale para 100% dos casos! Afinal, alguns dispositivos podem executar *software* cliente e servidor ao mesmo tempo – por exemplo, quando você desenvolve aplicações e realiza os testes localmente.

Clientes Web: Mozilla Firefox, Internet Explorer, Google Chrome etc.

Servidores Web: Apache, Microsoft IIS, Sun, Google Web Server etc.



Veja as estatísticas dos navegadores mais utilizados para acesso ao site da *W3Schools*.

Disponível em: <http://bit.ly/2Nshveh>

Veja as estatísticas dos servidores web mais utilizados segundo a pesquisa da *Netcraft*.

Disponível em: <http://bit.ly/2NuG1M1>

Camadas Lógicas

Agora você será apresentado(a) ao conceito de camadas – *tiers* – de aplicação. Uma aplicação cliente-servidor possui, pelo menos, duas camadas lógicas: a camada de *software* que fica no cliente e a camada de *software* que está no servidor.

Em geral, os sistemas de *software* são implementados com uma terceira camada, representação que damos o nome de **modelo em três camadas** – *3 logical tiers*.

As camadas são partes de *software*, cada qual responsável por uma tarefa específica na aplicação. Essa estratificação é um modelo de referência; na prática, os limites podem não ser tão nítidos.



Figura 11 – Representação conceitual do modelo de três camadas

Fonte: Acervo do Conteudista



Importante!

A divisão em camadas lógicas de *software* não necessita seguir a mesma divisão em camadas físicas do *hardware*.

A **camada de apresentação** é responsável por exibir as informações e interagir com o usuário. Deve disponibilizar as informações de forma adequada e responder apropriadamente à entrada do usuário.

Já a **camada de aplicação** processa comandos, toma decisões lógicas, executa cálculos e coordena o sistema; ademais, move e processa dados entre as camadas de apresentação e de dados – é nesta camada que as regras de negócio serão implementadas.

A **camada de dados** é responsável por gerenciar o banco de dados, ou os arquivos utilizados pela aplicação. Nessa camada é comum o uso de um Sistema Gerenciador de Banco de Dados (SGBD) – por exemplo, *MySQL*, *Oracle*, *DB2* etc.

Arquitetura em três camadas é um termo utilizado para descrever as **camadas físicas** do modelo.

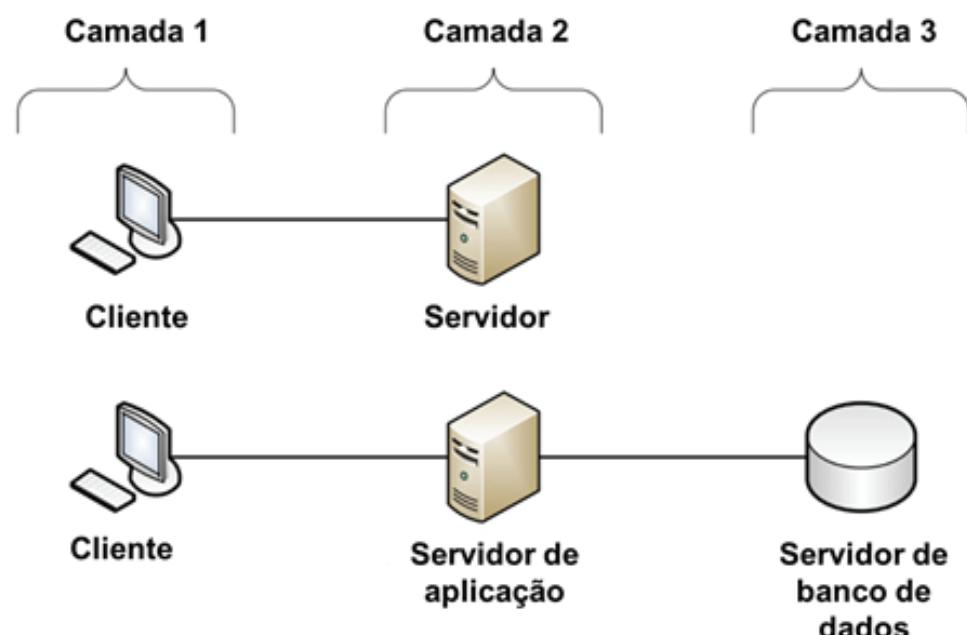


Figura 12 – As três camadas físicas



Importante!

Caso você encontre uma aplicação que possui, em sua implementação, mais do que três camadas físicas, saiba que são conhecidas como arquiteturas N camadas – *multi-tier*.

A separação em camadas lógicas e físicas torna cada sistema mais flexível, possibilitando que as alterações possam ser realizadas de modo independente. O desempenho também pode ser melhorado com a adição de novas camadas físicas para平衡ar o tráfego e dividir o processamento.

Peer-to-Peer

O modelo *Peer-to-Peer* (P2P) caracteriza-se pelo fato de os componentes finais terem capacidades e responsabilidades equivalentes em qualquer uma das partes, podendo, por exemplo, iniciar uma sessão de comunicação sem a necessidade de um servidor. Os participantes de uma rede P2P compartilham uma parte de seus próprios recursos de *hardware*, tais como as capacidades de armazenamento, processamento etc. Esses recursos compartilhados são necessários para fornecer o serviço ou conteúdo oferecido pela rede P2P.



Importante!

Os participantes de uma rede P2P são provedores de recursos e, ao mesmo tempo, solicitantes de recursos.

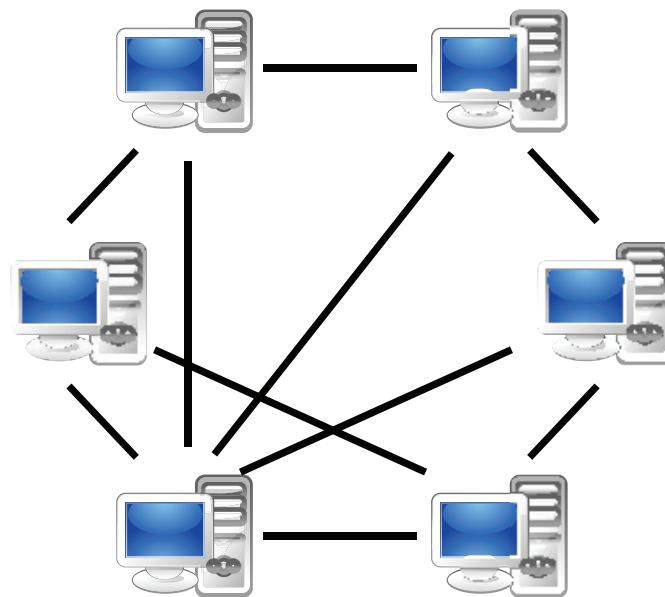


Figura 13 – Um sistema P2P sem uma infraestrutura central

Fonte: Wikimedia Commons

Em redes P2P, os fluxos de dados recebidos e enviados tendem a ser simétricos; isso ocorre porque cada *host* conectado opera simultaneamente como cliente e servidor, recebendo e transmitindo, em média, a mesma quantidade de dados.



Em Síntese

O modelo P2P não possui a noção de clientes e servidores, pois todos os participantes da rede são pares – *peers*.

Benefícios e Deficiências

Benefícios do P2P:

- Não há necessidade de aplicações dedicadas e servidores de banco de dados;
- Aumenta a possibilidade de escalabilidade e confiabilidade, pois não há um único ponto de falha.

Deficiências do P2P:

- Mais difícil de implementar políticas de segurança;
- Falta de controle centralizado;
- Computadores com recursos compartilhados tendem a ter desempenho pior que servidores dedicados.

Um exemplo de transmissão de dados via *peer-to-peer* corresponde aos *torrents*, em que você pode encontrar arquivos compartilhados, inclusive, que possuem direitos autorais. No Brasil, a Lei dos direitos autorais (Art. 105) proíbe qualquer tipo de reprodução de conteúdo protegido que não seja autorizado.



Na sua opinião, o fato de as redes P2P permitirem acesso a conteúdo protegido por direitos autorais – filmes, músicas, livros, softwares etc. – é uma vantagem ou deficiência?



Leia sobre o caso *Napster* em – <http://bit.ly/320QstX>

Exemplos de Aplicações do Modelo P2P

- **Comunicação:** *Skype*;
- **Bases de Dados:** DNS, NTP;
- **Jogos on-line:** *Counter Strike*, *Unreal*;
- **Compartilhamento:** *eMule*, *BitTorrent*, *Gnutella*;
- **Computação em grid:** *WCGrid*;
- **Content Delivery Networks (CDN):** *Amazon WS*, *Azure*;
- **Redes anônimas:** *Tor*;
- **Criptomoedas:** *Bitcoin*, *Ether*;
- **Outros:** *blockchain*, *JXTA*.

Material Complementar

Indicações para saber mais sobre os assuntos abordados nesta Unidade:



Livros

Engenharia de Requisitos

SOMMERVILLE, I. Engenharia de Requisitos. In: **Engenharia de Software**. 9. ed. São Paulo: Pearson, 2011. p. 57-77.



Vídeos

Tor: Entenda como esta rede garante o seu anonimato na Internet

<https://youtu.be/ju5U4kXgdoY>



Leitura

O que é *Blockchain*? Conheça a História, o Funcionamento e as Vantagens dessa Tecnologia

<http://bit.ly/323JpAD>

Introdução ao padrão MVC

<http://bit.ly/324upT3>

Referências

COULOURIS, G. *et al.* **Sistemas distribuídos:** conceitos e projeto. 5. ed. Porto Alegre, RS: Bookman, 2013.

ISO/IEC 9126. 2019. Disponível em: <https://pt.wikipedia.org/wiki/ISO/IEC_9126>. Acesso em: 5 fev. 2019.

MODELO OSI. 2019. Disponível em: <https://pt.wikipedia.org/wiki/Modelo_OSI>. Acesso em: 6 fev. 2019.

SILVA, M. L. C. Arquitetura de sistemas distribuídos. Maranhão: Instituto Federal de Educação Ciência e Tecnologia do Maranhão, 2017. Disponível em: <<https://docplayer.com.br/129528-Arquitetura-de-sistemas-distribuidos.html>>. Acesso em: 6 fev. 2019.

TANENBAUM, A. S.; VAN STEEN, M. **Sistemas distribuídos:** princípios e paradigmas. 2. ed. São Paulo: Pearson Prentice Hall, 2007.



Cruzeiro do Sul
Educacional