Sistemas Distribuídos

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, aparentam ser um sistema único (Tanenbaum e Steen, 2008).

Camadas que compõe um sistema distribuído:

- Aplicação
- Middleware.
- Sistema Operacional.

Em um sistema distribuído, a camada de middleware é um dos fatores principais para o bom funcionamento de aplicações distribuídas, esta 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 sistema operacional com programas.

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

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

- Disponibilidade alta e fácil de acesso ao sistema e todos os seus recursos, tanto pelas máquinas que fazem parte do sistema distribuído, quanto ao 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, ter facilidade para inclusão de novas máquinas e recursos nesse ambiente que já funciona. Sendo assim esse sistema pode ser expandido facilmente.

Modelos de Arquitetura de Redes de Computadores

Atualmente existem três tipos de arquitetura de computadores (Maia, 2013):

- Arquitetura cliente-servidor.
- Arquitetura ponto a ponto.
- Arquitetura descentralizada.

Arquitetura Cliente-Servidor

Essa talvez seja a arquitetura mais conhecida e utilizada nos sistemas informatizados, tendo sua origem na década de 70.

Neste tipo de arquitetura, teremos alguns serviços e recursos a serem compartilhados para vários usuários, que podem ser disponibilizados em um único computador – sendo chamados de servidores multisserviço – ou sendo segregados, ou seja, um serviço ou recurso por computador e, 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 conectados através de equipamentos de rede, como switches, roteadores e firewalls, tipicamente à internet, 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 conectados através de equipamentos de rede à internet.

Arquitetura Ponto a Ponto

Também é conhecida como arquitetura peer-to-peer, ou simplesmente P2P, a arquitetura ponto a ponto teve sua origem na década de 80. Neste tipo de arquitetura, os computadores, sejam eles laptops, desktops, smartphones ou tablets, possuem o mesmo papel nessa rede, ou seja, funcionam tanto como dispositivos finais quanto como servidores, uma vez que, um mesmo computador pode disponibilizar recursos e serviços para um computador e, similarmente, pode utilizar (ou, muitas vezes dito, consumir) recursos e serviços de um computador nessa rede.

Esses computadores também estarão conectados através de equipamentos de rede, como switches, roteadores e firewalls, tipicamente à internet, para que as pessoas possam compartilhas os recursos e serviços na internet. Importante observar que essa arquitetura, na maioria das vezes, também funciona no nível de rede local (do inglês, local area network).

Arquitetura Descentralizada

Arquitetura mais recente, a partir dos 2000, pode ser vista como uma arquitetura híbrida entre a cliente-servidor e a ponto a ponto.

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 na tal aplicação) é replicado entre todos os computadores na rede, de maneira que exista um chamado consenso entre os computadores nessa rede.

Essa é a arquitetura utilizada pelas plataformas baseadas em blockchain, e tem 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 utilizar esse tipo de arquitetura é a de que não há uma entidade que controle sua aplicação, como tipicamente ocorre nas arquiteturas cliente-servidor.

Sistemas Centralizados

Neste tipo de sistema um conjunto de máquinas utiliza seus recursos e temos uma máquina centralizada de servidor. Esse tipo de sistema só tem seu funcionamento possível através de mainframes, esses são computadores de grande porte com o objetivo de receber uma grande quantidade de informações e processá-las.

Vantagens e Desvantagens de Sistemas Centralizados

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

Sistemas Paralelos

Esse tipo de sistema tem como objetivo executar de forma simultânea 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 tem o comportamento de uma só máquina.

Na maioria das vezes os sistemas paralelos são baseados em clusters, eles são um conjunto de máquinas interligadas que dividem sua carga de processamento. A maioria dos processadores encontrados no mercado hoje em dia tem vários núcleos, o que quer dizer que a maioria dos computadores pessoais seguem o modelo de sistemas paralelos, onde os diversos núcleos do processador dividem o processamento das informações de forma paralela.

Vantagens e Desvantagens de Sistemas Paralelos

- Escalabilidade.
- Produtividade.
- Economia.
- Dificuldade para gerenciamento.
- Segurança.

Sistemas Paralelos Fortemente Acoplados

Os sistemas paralelos fortemente acoplados abrangem um conjunto bem grande de sistemas computacionais, que abrangem, por exemplo, os laptops, smartphones e os desktops, adquiridos através de websites de varejo ou lojas de departamentos.

Duas características fundamentais diferenciam esses sistemas em relação ao demais (Coulouris, 2013):

- Comunicação entre processadores ocorre por meio de barramentos internos, que são conjunto de sinais digitais onde os processadores transmitem e recebem dados.
- Os processadores compartilham a mesma memória principal (conhecida popularmente como RAM – do inglês, random access memory).

Sistemas Paralelos Fracamente Acoplados

Os sistemas paralelos fracamente acoplados abrangem os sistemas de maior escala, sempre conectados via rede de computadores (Tanenbaum e Steen, 2008). Sendo assim, todos os sistemas de rede, que utilizam arquiteturas do tipo cliente-servidor, ponto a ponto ou descentralizadas pertencem a esta 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, menos rígido que dos sistemas fortemente acoplados.

Vantagens e Desvantagens de Sistemas Fracamente Acoplados em Relação ao Sistemas Fortemente Acoplados

- Desempenho.
- Flexibilidade.
- Escalabilidade.
- A velocidade na transferência de dados e segurança é mais vulnerável.

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

Fundamentação de Sistemas Distribuídos

Mesmo que você não saiba, hoje mesmo você já deve ter acessado um sistema distribuído. Agora 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, via 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, aparentam ser um sistema único como, por exemplo, uma única máquina ou um único software (Tanenbaum, 2008).

Um de seus principais aspectos é de 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 hardware dessas máquinas são totalmente diferentes, porém eles aparentam ao usuário serem uma coisa só. Esses computadores estão ligados via rede e só assim é possível seu funcionamento de forma distribuída.

Os sistemas distribuídos são, sem sombra de dúvida, mais utilizados em arquiteturas do tipo cliente-servidor. Como vimos anteriormente, esse tipo de arquitetura possui recursos compartilhados (tanto a nível de hardware quanto a nível de software) que permite com que milhares de clientes tenham acesso ao recurso, e inclusive possam utilizar esses recursos de forma como se houvesse uma comunicação direta entre as máquinas cliente.

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.

Computação em Cluster

Esse tipo de computação é formado por um conjunto de máquinas com hardware semelhantes, ou seja, as máquinas que compõe o cluster possuem características homogênea, conforme Tanenbaum (2013). O conjunto de máquinas que compõe o cluster são ligadas via rede local (LAN). 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 que é subdividido em partes menores e cada parte é executada em uma máquina (ou nó) desse cluster, de forma distribuída, de forma a obter um aumento significativo de desempenho e, consequentemente, executar determinada tarefa em menos tempo.

Geralmente as máquinas desse tipo de sistema são fortemente acopladas 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.

Computação em Grid

Conforme Tanenbaum (2013), esse tipo de computação é formado por um conjunto de máquinas com características diferentes, entre elas o hardware e os sistemas operacionais podem 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).

Diferenças entre Clusters e Grids

Muitas vezes, pode parecer que clusters e grids são a mesma coisa, mas existe uma diferença fundamental que difere esses dois tipos de sistemas distribuídos.

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.

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 destes responsável por um certo tipo de pesquisa.

Sincronização de Relógios

Sistemas formados por múltiplos computadores necessitam sincronizar suas ações entre si, em uma das maneiras mais utilizadas, dada sua simplicidade e popularidade, é sincronização horária, através do protocolo conhecido como Network Time Protocol – NTP. 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 (desktopos, laptops, servidores) e demais dispositivos de rede.

Sistemas Distribuídos

Vamos retomar 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 possuem 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 (máquinas estas, também chamadas de nós). Esses recursos são, na maioria das vezes, bancos de dados, link 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 exclusivo dos sistemas de rede – é uma característica muito importante de um sistema distribuído.

- **Vantagens:** 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 diretamente e indiretamente, no curso.
- **Desvantagens:** 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).

Confiabilidade

A análise morfológica da palavra confiabilidade nos mostra que esta refere-se 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 a segurança do sistema, porém não tem relação alguma com a parte de segurança do sistema, conforme observa 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, como ocorre a comunicação entre processos nos sistemas distribuídos onde são aplicados os conceitos de confiabilidade.

Podemos observar que se ocorrer algum problema no canal de comunicação isso é 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).

Uma das características importantes do teorema CAP, que pode ser observada na representação da imagem que acabamos de ver, é de que nunca podemos atingir os três pilares em sua totalidade dentro de um sistema. A forma como foi elaborado só 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.

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, esta funcionará em uma plataforma com várias máquinas, chamadas de nós, que replicam a 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 a 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.

- **Confidencialidade:** A confidencialidade dos dados refere-se ao acesso ao dado, por indivíduos ou sistemas não autorizados.
- **Integridade:** A integridade dos dados refere-se a quando, além de ser acessado, o dado foi modificado.

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 curso e ameaças (Coulouris et al., 2013).

Pontos de Atenção

Ainda conforme Coulouris et al. (2013), os pontos de atenção em relação a 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 nas quais esses serviços se comunicam são, intrinsecamente, 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 ser 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 mesa 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, certifique-se de considerar que ataques podem ocorrer de inúmeras fontes, e que podem explorar vulnerabilidades utilizando inclusive ataques do tipo força-bruta (que tentam descobrir senhas por tentativa e erro, através de simples “chutes”).

Escalabilidade

A escalabilidade é outro aspecto importantíssimo de um sistema distribuído. Escalabilidade é um termo comum em termos de 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 o mesmo aumente ou diminua a quantidade de recursos.

Diminuição da Capacidade do Sistema: Imagine a seguinte situação: Você criou uma aplicação web que distribui conteúdo em vídeo para preparar estudantes a fazer 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".

Dois aspectos importantes de serrem 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 data centers 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.

Exemplo de Escalabilidade Geográfica

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, de um furacão passar por Arizona, e comprometer aquele data centers, todos os usuários poderão acessar o data centers de São Paulo, incluindo os do hemisfério Norte (ainda que a usabilidade, do ponto de vista dos usuários do hemisfério Norte, seja ligeiramente comprometida, devido a maior distância desse data centers).

Escalabilidade em termos administrativos 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 imagem de escalabilidade geográfica, que estudamos anteriormente, 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.

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 um vídeo, compartilhar arquivos na nuvem, etc. Exemplos desse tipo de computador são computadores do tipo desktop, laptops e smartphones. Essas máquinas consideradas clientes têm a função de enviar e receber requisições/solicitações de máquinas dos diversos tipos de servidores, que entraremos em detalhes.

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

- Em um sistema distribuído, as máquinas sempre são responsáveis por iniciar as solicitações/requisições ao servidor.
- A máquina cliente aguarda por respostas de outros servidores.
- A máquina cliente recebe respostas de outros servidores.
- A máquina cliente, geralmente, se conecta a um pequeno número de servidores de uma só vez.
- 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. 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, dentre outras informações.

Já na próxima, 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. Lhe parece familiar? Toda resposta que uma máquina cliente recebe é “traduzida” em um arquivo com formato HTML, pois essa é a linguagem oficial usad na web. Mesmo que os desenvolvedores tenham usado frameworks: Java, .NET, Python, etc., para implementar o sistema, a resposta sempre será traduzida em HTML.

Definição de Virtualização

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”.

- O grande objetivo da virtualização é fornecer uma versão virtual de tecnologias essenciais em computação, como: Redes, armazenamento, hardware, entre outros. Além disso, podemos virtualizar aplicações.

Quando utilizamos virtualização, representamos os dispositivos físicos por meio de entidades de software. Assim, nossos servidores e workstations tornam-se o que chamamos de máquinas virtuais ou VMs.

- A parte de armazenamento de dados é conhecida como SDS – Software Defined Storage.
- A parte da rede é chamada de SDN – Software Defined Networking.

Unindo esses elementos com um conjunto de máquinas virtuais temos um SDDC – Software Defined Data Center.

Componentes da Virtualização

Segundo Redhat (2018), a virtualização possui três componentes principais:

- **Hospedeiro:** Como chamamos a máquina física onde existem máquinas virtuais.
- **Convidado:** Como são chamadas as máquinas virtuais, ou computadores virtualizado.
- **Camada de Virtualização:** O software que permite criar sistemas convidados sobre sistemas hospedeiros.

Para o usuário final não faz diferença se a máquina acessada é física ou virtual, as duas funcionam da mesma forma e isso acaba sendo imperceptível.

Principais Fatores que Levam a Utilização de Virtualização

A seguir, conheça quais são esses fatores.

- **Diminuição de Espaço Físico:** Muitas vezes, o ambiente corporativo não tem espaço físico para suportar servidores, com todos requisitos necessários.
- **Rapidez na Implantação:** Máquinas Virtuais são mais rápidas de serem implantadas do que máquinas físicas.
- **Redução de Custos Administrativos:** Os custos administrativos para se manter uma máquina física são bem maiores do que 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:** Podemos aproveitar melhor os recursos de um servidor físico, dividindo-os em várias máquinas virtuais.

O Papel da Virtualização em Sistemas Distribuídos

Segundo Coulouris et al. (2013), são dois os tipos de virtualização muito úteis no contexto de sistemas distribuídos:

- **Virtualização de Redes:** Coulouris et al. (2013) observam, muito adequadamente, que a vantagem da criação e 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 rede virtual possa ser otimizada para aquela aplicação em particular, sem a necessidade de alterar as características da rede física.
- **Virtualização de Sistemas:** Para Coulouris et al. (2013), a virtualização de sistemas é uma alternativa interessante por permitir a emular o hardware de uma máquina física, permitindo assim que várias máquinas virtuais, cada uma com um sistema operacional, possam coexistir e se comunicar.

Por fim, para terminar, vale salientar que se você já utilizou ou leu a respeito de computação em nuvem, deve saber que, independentemente do tipo de serviço que você contrata, e independentemente 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).

- SaaS (do inglês, Software as a Service).

Virtualização

Definição de Conteinerização

- O conteiner funciona como uma tecnologia que dá o suporte para o funcionamento de uma aplicação, pode ser considerado a emulação de nossa aplicação.

Como um grande diferencial, podemos dizer que os conteiners são mais “leves” do que as máquinas virtuais, isso se deve ao fato de eles possuírem uma arquitetura mais otimizada. Além disso, quando se trata de conteiners, não há, necessariamente, a necessidade de instalação de um sistema operacional completo, visto que as plataformas de conteinerização aproveitam bibliotecas compartilhadas com o sistema operacional hospedeiro.

Por essa razão, os conteiners ocupam menos espaço em disco e consomem menos RAM e processamento que as máquinas virtuais e, assim, possibilita a utilização de mais conteiners em uma mesma máquina física, o que, por sua vez, favorece o uso de uma arquitetura mais modular para as aplicações.

Características da Conteinerização

Duas das principais características a favor da conteinerização são:

- O baixo acoplamento entre os conteiners.
- A facilidade de migração entre os provedores de cloud computing.

Ambas se devem ao fato de que a ideia do conteiner é “empacotar” a sua aplicação em um módulo que é facilmente instalado em qualquer sistema operacional que suporte o uso de conteiners (e os principais sistemas operacionais, utilizados em servidores, possuem esse suporte).

Exemplo de Conteiner

Existem implementações que podem ser consideradas conteiners de sistema, como, por exemplo, os Linux conteiners. Essas tecnologias têm um comportamento muito parecido ao de uma máquina virtual, mas, na verdade, são equivalente a novas instâncias do sistema, utilizando todas as técnicas disponíveis para isso.

- Os conteiners Linux funcionam compartilhando o kernel da máquina real no qual estão em funcionamento.

Uma limitação encontrada nos conteiners é que quando estão compartilhando o kernel da máquina física ou até mesmo virtual onde estão hospedados e em execução, tratam-se de conteiners Linux rodando em máquinas Linux com o mesmo kernel do host.

Segundo linuxcontainer (2018), o projeto Linux container é o guarda-chuva por trás do Linux Container, que possuem as gerações:

- **LXC:** É uma interface de espaço de usuário para os recursos de contenção de kernel do Linux.
- **LXD:** É um gerenciador de conteiners de próxima geração.

- **LXCFS:** É um sistema de arquivos simples do userspace projetado para contornar algumas limitações atuais do kernel do Linux.

O Papel da Conteinerização em Sistemas Distribuídos

Os sistemas distribuídos fazem uso extensivo dos conteiners, no contexto de microsserviços. A ideia dos microsserviços está associada a empresas que possuem sistemas altamente dinâmicos, e ao termo modularidade. Dentre as vantagens de um sistema distribuído baseado em microsserviços, podemos apontar que:

- Quanto menores são as partes, mais fácil entendê-las.
- Cada microsserviço pode ser executado e escalado de maneira concorrente e independente entre si.
- 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.

Apesar da criação e do controle de um conteiner serem realizados com simples comandos, uma aplicação é composta por vários microsserviços, representando cada um deles um (ou mais) conteiner(s).

Em síntese, gerenciar dezenas ou centenas de conteiner, de maneira isolada, pode ser uma tarefa muito trabalhosa, razão pela qual as empresas utilizam a ferramenta “orquestração” para criar, gerenciar e remover conteiners.

- Atualmente, a ferramenta de orquestração mais popular e, portanto, mais solicitada no mercado de trabalho, é o Kubernetes, do Google, que serve para “orquestrar” conteiners criados com o Docker.

Simulando Sistemas Distribuídos com Docker

Docker

É uma famosa plataforma genérica de conteinerização, conceito parecido com virtualização, porém considerado “mais leve” que este. Atualmente, conteiners são populares devido à facilidade e à flexibilidade que advêm de seu uso. Assim, chegou o momento de aprendermos a utilizar essa famosa ferramenta.

Instalação do Docker

Todo o procedimento de instalação deve ser feito com um usuário com permissões de “administrador”. No caso que será apresentado a seguir, utilizaremos o sistema operacional Ubuntu e o “root”, por meio do comando “sudo su”. Em resumo, o processo de instalação é bem simples, sendo somente necessário avançar as seguintes etapas:

1. Antes de instalar a ferramenta, devemos remover versões anteriores do Docker que possam estar instaladas, utilizando o seguinte 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.
2. 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. Para isso, são

necessários o seguinte comando: *sudo apt-get update*. Esse comando tem o objetivo de realizar uma atualização de pacotes do Ubuntu.

3. Através do seguinte comando atualizamos os pacotes necessários para a instalação do Docker: *sudo apt install apt-transport-http ca-certificates curl software-properties-common*.
4. Após atualizarmos os repositórios, adicionaremos o repositório de instalação do Docker. Para isso, é necessário o uso do seguinte comando: *curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -*. Com esse comando, apontamos o caminho de instalação oficial do Docker, no qual o Ubuntu deve acessar quando fizermos a instalação.
5. Após o apontamento da URL onde está disponível 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".
6. Depois de utilizar 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.
7. Após adicionar o repositório para download do Docker, devemos mais uma vez atualizar o "apt-get", conforme o comando seguinte para aplicar as alterações: *sudo apt-get update*.
8. Neste momento, utilizaremos o comando "sudo" de instalação do Docker, para utilizar a permissão "admin" do sistema e o "apt-get install" para fazer a instalação. Por fim, deve-se inserir o nome do programa que será instalado, 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.

Iniciando e Testando o Docker

Para ver se o Docker foi instalado corretamente, devemos iniciar o serviço do Docker e, após isso, verificar se ele está em execução. Para tanto, é necessário utilizar os seguintes comandos: *sudo service docker start service docker status*.

Agora que instalamos e verificamos seu funcionamento, o sistema está apto a receber as especificações que queremos criar. Para isso, é possível usar o Docker Swarm. Nesse cenário é possível agrupar vários hosts em um mesmo pool de recursos, o que facilita o deploy de containers (Diedrich, 2018).

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

Por meio dos comandos que serão apresentados a seguir, podemos simular algumas características de sistemas distribuídos com Docker.

Comando Executado "Fora" do(s) Nó(s)

Um dos comandos importantes é o de criação de um novo manager para o nosso cluster. Através dele, criamos um manager chamado de "mestre":

docker-machine create –driver virtualbox mestre

Ao executarmos esse comando, o começo dele significa que estamos criando uma nova máquina através do Docker ("docker-machine create").

Comando Executado “Dentro” dos Nó(s) Manager

O comando a seguir pode ser utilizado para iniciar o cluster através do framework swarm, de gerenciamento de conteiners:

docker swarm init –advertise-addr < IP DO MESTRE >

O comando a seguir pode ser utilizado para iniciar o cluster através do framework swarm, de gerenciamento de conteiners.

- **“node Is”:** Podemos consultar os nós que fazem parte do cluster utilizando o comando “node Is”.
- **“inspect”:** Em algumas ocasiões, precisamos verificar informações sobre um nó específico. Para isso, utilizamos o comando Docker “inspect”. No exemplo a seguir trazemos informações sobre o nó chamado “escravo1”: *docker inspect escravo1*.
- **“ps”:** Depois de criar o serviço de internet chamado “WEB”, podemos utilizar o comando “ps”, seguido do nome do serviço, para verificar suas informações. Por exemplo: *docker service ps WEB.*
- **“update”:** Podemos utilizar o comando “update” para alterar a versão do “Nginx”. Por exemplo, é possível alterar a versão 1.12.1 para a 1.13.5: *docker service update –image nginx:1.13.5 WEB.*

Comando Executado “Dentro” do(s) Nó(s) Worker

O comando a seguir pode ser utilizado para adicionar um nó escravo (worker) ao nosso cluster. 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, conforma a sintaxe representada no comando:

docker swarm join –token < IP do mestre:Porta >

Comando Executado “Dentro” de Cada Um dos Nós (Managers e Workers)

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. Conforme observamos a seguir sua utilização:

docket system prune -all

Para terminar os estudos desta aula, vamos aprender a orquestrar o servidor web Apache em um cluster simples. Primeiramente, você deve estar logado na plataforma de “playground” do Docker. Em seguida, será necessário:

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 através do botão “Add New Instance”.
2. No nó que você deseja que seja o mestre, digite o seguinte comando: *docket swarm init –advertise-addr < endereço-ip-desse-nó >*. Ao executar esse comando, é apresentada uma saída, com a mensagem: “Para adicionar um worker ao swarm, execute o seguinte comando”, a qual você deve copiar. Esse comando dever[a ser executado em cada um dos demais nós do cluster, adicionando cada um deles como workers desse cluster.

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*.
4. Para saber em quais nós as 5 réplicas desse serviço estão sendo executadas, digite o seguinte comando: *docker service ps WEB*.
5. Por fim, precisamos acessar a página de boas-vindas do servidor Apache através do(s) endereço(s) IPv4 de cada nó onde esse serviço web estiver rodando. Para acessar esta página, basta clicar na porta que foi mapeada por você na parte superior (que aparece como um hyperlink), 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.

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. Vale ressaltar que, caso ocorra algum problema em um dos nós que mantém a aplicação, o outro nó assume a execução.

Segurança em Sistemas Distribuídos

Características da Segurança de Sistemas Distribuídos

Todo sistema distribuído implementado têm, como um dos principais e mais importantes, aspectos de projeto a segurança, conforme 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.

Podemos dividir a parte de segurança de sistemas distribuídos em duas: Permissão de acessos a serviços e recurso disponíveis no sistema e comunicação entre máquinas que contém mais deu m processo e usuários diferentes (Goodrich e Tamassia, 2012).

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

- **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.
- **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.
- **Autenticidade:** Podem 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.
- **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.
- **Disponibilidade:** Significa garantir que a informação que esteja sempre disponível para quem precisar dela. Podemos obter a disponibilidade do sistema utilizando as políticas de segurança corretamente em nosso sistema.

Aplicando esses fatores, que acabamos de conhecer, 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.

Ameaças ao Sistema Distribuído

Para produzir um sistema que seja seguro contra diversas ameaças, precisamos aprender 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 a 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 dessas 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:** A máquina do invasor faz envio ou recebimento de mensagens utilizando a identidade de outra máquina autorizada pela aplicação.
- **Message Tampering:** 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.

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:

- Cavalo-de-Tróia (Trojan horse).
- Vírus.
- Worm.
- Spyware.
- Keyloggers.

- Backdoor.
- Adware.
- Exploits.
- Hoax.

Protocolo TCP e UDP

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.

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 é 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.

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).

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).

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 tabela.

Funcionalidade	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.

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.

Middlewares

Na comunicação entre máquinas em um sistema distribuído, é comum o uso de middlewares que, 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.

O middlewares pode interligar 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 já estudado, a comunicação via sockets envolve as camadas um a quatro do modelo de referência ISO/OSI. Já a comunicação via RCP 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.

Modelos de Comunicação

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 a tabela:

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 Middleware)	Linguagens de programação para web.

Dependendo da linguagem de programação a qual o desenvolvedor tem mais familiaridade, existem vários frameworks que podem ser adotados para implementação do RPC.