GCC129 - Sistemas Distribuídos

Relatório Técnico 2

Modelos de Arquitetura para Aplicações Distribuídas

Nome: Mateus Carvalho Gonçalves - 201810245
Turma: 10A

1. Introdução

Um sistema distribuído, como qualquer outro sistema computacional, precisa tratar requisitos de desempenho, segurança e transparência, entre outros. Para isso, é necessário planejar o desenvolvimento do software de maneira que consiga a melhor eficiência para seu fim, e os modelos arquiteturais ajudam nesse processo.

A partir de descrições simplificadas e abstratas, esses modelos buscam definir aspectos de componentes e comunicação, como características e funções de determinados tipos de componentes, delimitar como e com quem podem se comunicar, etc. Basicamente, é a organização lógica dos componentes para formação do sistema distribuído, isso gera um impacto até mesmo na organização física do hardware.

Um sistema distribuído deve ser projetado para funcionar corretamente, com desempenho aceitável, na maior variedade possível de circunstâncias, assim como diante de falhas e ameaças de segurança. Por isso, os modelos de arquiteturas são importantes e devem ser bem planejados para melhorar a comunicação interna e principalmente com serviços de outros fornecedores, e sustentar mudanças de demanda e alterações na topologia de rede, com o menor impacto possível na entrega do produto.

Posto isso, existem modelos centralizados, descentralizados e híbridos, que misturam características dos dois primeiros. Nas seções 2, 3 e 4 serão discutidos os modelos de arquitetura Cliente-Servidor, Microsserviços e Peer-to-Peer, respectivamente, sendo os dois primeiros modelos centralizados, em sua essência[1], e o último descentralizado.

[1] Com isso, quer dizer-se que a arquitetura possui mais características daquele modelo, porém são mais frequentes os casos em que essas arquiteturas são usadas de forma híbrida.

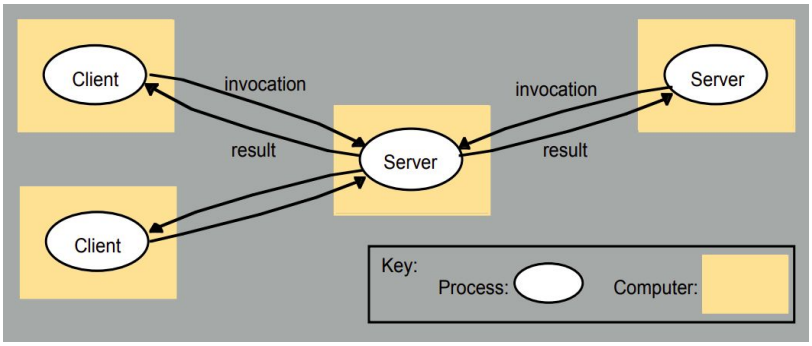
2. Cliente-Servidor

Esse modelo de arquitetura é o mais utilizado em sistemas atualmente, principalmente com o advento da Internet. Num modelo Cliente-Servidor básico, temos:

- Servidor: implementa um serviço específico;
- Cliente: faz requisições de serviços.

Com isso, é possível concluir que esse modelo implementa o protocolo request-response. É interessante apontar que os componentes básicos servidores não atuam de forma estática, ou seja, eles podem atuar como clientes fazendo requisições de serviços para outros servidores, como mostrado na Figura 1. É importante lembrar que apesar desses serem os componentes básicos desse modelo, podem existir vários outros componentes em um sistema desse tipo.

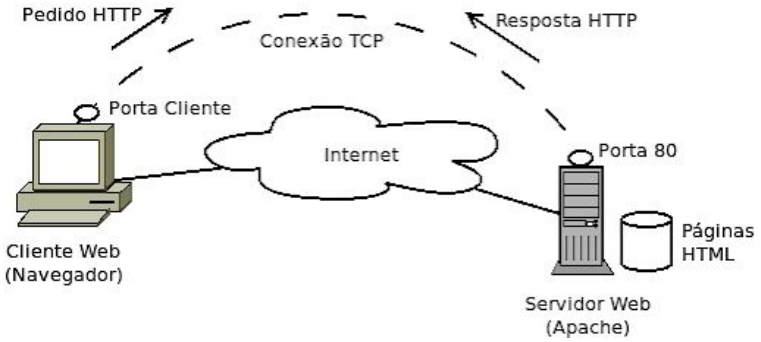
Figura 1. Representação de requisições do modelo Cliente-Servidor



A disponibilidade dos servidores é um dos fatores cruciais na implementação Cliente-Servidor. Para garantir que os serviços estejam sempre disponíveis, é comum que os dados sejam replicados em múltiplos servidores e o fluxo de requisições pode ser controlado por meio de balanceamento de carga.

A web funciona basicamente seguindo esse modelo, onde o navegador é um processo cliente e faz requisições HTTP a um ou mais servidores, que respondem com os dados solicitados, como mostra a Figura 2.

Figura 2. Modelo Cliente-Servidor na internet utilizando requisições HTTP



3. Microsserviços

A ideia principal dos Microsserviços é descentralizar o trabalho dos servidores. Ao invés da aplicação inteira estar disponível em um servidor (ou em servidores replicados), é feita a modularização dos serviços para que apenas ele seja executado quando necessário. Esse modelo permite que as funções sejam replicadas de maneira mais eficiente, já que pode-se escolher apenas o serviço que possui maior demanda.

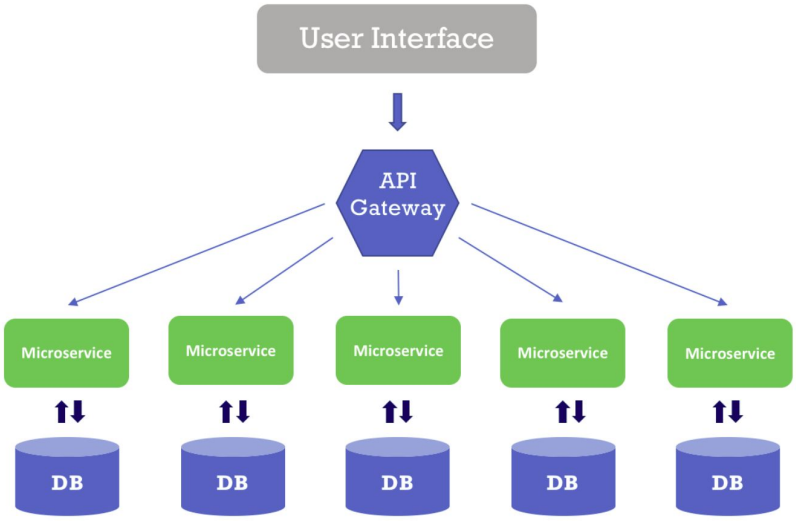
Além disso, essa arquitetura permite que os bancos de dados sejam modularizados para atender apenas um microsserviço, aumentando a eficiência de busca.

Com essa descentralização, é possível que a aplicação comece a ter muitos endpoints e dificulte o acesso dos usuários, entretanto é possível contornar esse problema com um API gateway, que irá receber todas as requisições e direcionar para o microsserviço correto (Figura 3).

Entre outros benefícios dos Microsserviços que ainda não foram apontados, podemos citar: projetos focados em uma aplicação mais atômica, combinação de tecnologias (escolha de uma tecnologia mais viável para implementar um serviço), isolamento de falha e agilidade de correção de bugs, base de código menor, etc.

Uma das desvantagens dessa arquitetura é o aumento no tempo de resposta, uma vez que o caminho pode aumentar e é necessário a instanciação de um container a cada requisição. Outros problemas podem ser a gestão de vários produtos ao mesmo tempo, complexidade de implementação, consistência de dados, etc.

Figura 3. Arquitetura de Microsserviços com API Gateway



A empresa onde faço estágio[2], por exemplo, utiliza microsserviços para modularizar o produto vende. Ela possui um grande produto integrado, mas também

[2] O nome da empresa não foi citado por questões de sigilo. Entrei há pouco na empresa e ainda não compreendo totalmente quais os limites de informação que devem ser respeitados.

fornece os módulos separadamente na necessidade dos clientes, e inclusive possui clusters modularizados para cada alguns tipos de serviços.

4. Peer-to-Peer (P2P)

Fazendo um paralelo com a arquitetura Cliente-Servidor, no modelo Peer-to-Peer (P2P) todas as máquinas podem se comportam tanto como cliente quanto como servidor, ou seja, elas tanto fazem requisições quanto respondem a elas (Figura 4). Isso permite uma boa boa escalabilidade da aplicação, uma vez que cada novo usuário também contribui para servir aos demais. Entretanto, é um modelo de difícil administração e alta complexidade de implementação.

Por causa dessa descentralização da prestação dos serviços, a disponibilidade de um serviço é imprevisível, pois é necessário que um “cliente” que possui aquele serviço esteja conectado à aplicação no momento.

Aplicações de compartilhamento de arquivos torrent são os exemplos de implementação mais conhecidos desse modelo de arquitetura, assim como o Skype. Todavia, é importante frisar que essas aplicações seguem um modelo híbrido, ou seja, geralmente possuem um servidor para “fazer a ponte” entre os usuários e o restante do serviço é por P2P.

Figura 4. Representação da comunicação dos computadores no modelo P2P