Ciencia da Computação

Blog com dicas C#, Java, Orientação a Objetos e Sistemas Distrbuídos

domingo, 30 de março de 2014

Arquitetura de Sistemas Distribuídos

Arquitetura

Sistemas distribuídos são sistemas complexos, que contem diversos componentes espalhados por várias máquinas em diferentes redes de computador. Saber administrar e organizar estes sistemas é um ponto crucial para obter um melhor desempenho.

Existem dois tipos organização que devem ser feitos, ao se planejar o desenvolvomento de um sistema distribuído, o tipo lógico que trata de onde e como organizar os softwares deste sistema, e o físico, que trata da organização física das máquinas em cada rede.

A arquitetura de um sistema distribuído deve levar em conta quais são os seus componentes de software e como eles interagem entre si, para que com esta análise, seja possível decidir qual é a melhor maneira de alocar as máquinas disponíveis para constituí-lo.

Construir um sistema distribuído depende da instanciação de vários componentes de software, como um servidor web, um servidor de cache, um servidor de banco de dados, servidor da camada de negócios, um firewall, um proxy, etc, e existem diversas maneiras diferentes de se organizar estes sistemas.

Esta organização também é conhecida como arquitetura de software, que pode ser classificada em dois tipos a centralizada onde uma máquina contém muitos dos componentes daquele sistema e a descentralizada, onde cada software roda em um servidor diferente.

Monitorar as máquinas destes sistemas também é um passo importante para obter-se alta disponibilidade. Fatores como o desempenho ou o tempo de resposta de uma certa máquina podem ser utilizados para se decidir quando criar novas instâncias daquele software ou camada, ou quando derrubá-las para economizar recursos. O monitoramento também pode indicar áreas de gargalo onde melhorias de desempenho devem ser implementadas a fim de obter-se um melhor aproveitamento do parque máquinas disponíveis.

Existem ferramentas de mercado que ajudam a descobrir o comportamento dos seus usuários, monitorando os seus cliques, o movimento do mouse. Estas ferramentas também disponibilizam dados como, sexo, idade, profissão, onde mora, quais lugares frequenta, quando acorda, que horas chega ao trabalho e algumas vezes inclusive a renda provável.

Estes dados são capturados pelas empresas afim de prover anúncios, ou mostrar produtos de forma a conseguir uma maior taxa de vendas. Como cada usuário gera uma quantidade considerável de dados, e os sites possuem milhares de usuários, a forma de armazenar e processar estes dados será diferente, eles geralmente não se encaixam em um sistema de armazenamento comum, como um banco de dados SQL, mas isto será discutido em um outro momento neste blog.

Estilos de arquitetura

A discussão sobre os estilos de arquitetura será realizada no nível dos componentes, em como eles se conectam entre si, como eles trocam dados e finalmente em como eles serão configurados em um sistema.
Os estilos de arquitetura de componentes são mostrados a seguir:

3. Arquitetura de Dados Centralizados
4. Arquitetura baseada em Eventos

Na arquitetura nivelada os componentes são organizados em várias camadas e uma requisição percorre os componentes apenas em uma direção.

Em uma arquitetura baseada em Objetos cada objeto é definido como um componente, e estes componentes são conectados através de chamadas do tipo RPC, que será utilizada por sistemas do tipo Cliente-Servidor.

Arquitetura baseada em dados centralizado diz que todos os processos devem se comunicar através de um sistema de repositórios (ativo ou passivo) comum. Exemplo temos softwares que utilizam apenas um tipo de banco de dados.

Arquitetura baseada em eventos os processos se comunicam através da propagação de eventos os quais opcionalmente poderão carregar dados. Como exemplo temos sistemas de publish/subscribe, ou mesmo aplicativos que usam o Akka Toolkit. Um processo publica um evento enquanto o middleware garante que alguém estará escutando aquele evento, a grande vantagem desta arquitetura é que os processos são desacoplados e ocorrem em paralelo, sem depender um do outro.

Arquiteturas de Sistema

Sistemas distribuídos são divididos em várias camadas, neste tópico serão abortados os sistemas de duas, três e multicamadas.

Duas Camadas

Um sistema de duas camadas é composto por um servidor, que contém algum tipo de armazenamento de dados como um banco de dados ou um sistema de arquivos distribuídos e por um cliente, onde ficam as regras de negócio e a interface do usuário. Este sistema era muito comum nos anos 90, onde os sistemas não eram tão complexos e poucas máquinas se conectavam ao servidor de banco de dados.

Neste modelo o servidor será responsável pelo armazenamento dos dados compartilhado, os clientes serão responsáveis por processar os dados retornados pelo servidor, fazendo com que o tráfego de dados seja muito alto entre o servidor e os seus clientes, pois muitas vezes os dados são selecionados crus no servidor e enviados ao cliente, que é quem realiza um tratamento nos dados. Por exemplo para fazer uma query ordenada, todos os dados serão selecionados no servidor e a ordenação será realizada no cliente.

Esta configuração funciona muito bem em locais onde a velocidade de rede alta, como em uma rede local, já que o tráfego de dados pela rede será alto.

A grande vantagem desta configuração é que o processamento dos dados será realizado por muitas máquinas diferentes, aliviando o processamento no servidor. Entretanto como o código de execução está instalado em várias máquinas diferentes, a instalação e manutenção destas máquinas torna-se uma tarefa muito complicada, fazendo com que a execução do software dependa das várias plataformas onde cada uma delas estiver instalada, o que costuma gerar muitos erros e incompatibilidades.

Um outro ponto a ser observado é que como a interface com o usuário está junto das regras de negócio, muitas vezes o código que desenha a interface e o código das regras de negócio estão misturados, o que dificulta a manutenção deste tipo de sistema, pois os designers (responsáveis pelas telas) e os desenvolvedores (responsáveis pelas regras de negócio) acabam trabalhando no mesmo código.

Três Camadas

Com a popularização de aplicativos web, ou mesmo a evolução dos escritórios para multi pontos em várias cidades, tornou necessário o desenvolvimento de uma nova arquitetura para atender estas novas limitações.

Em um sistema web ou multi pontos, velocidade da rede é bem inferior ao de uma rede local, além existirem muitas plataformas que acessam a web, como celular, pda, tablet, netbooks, pcs, servidores, etc. Com este novo modo de negócio a utilização de um sistema distribuído de duas camadas caiu em desuso.

A criação da terceira camada, serviu para parametrizar a separação entre as camadas de software, o que não existia no sistema de duas camadas.

Pela definição as três camadas deste sistemas são, o repositório de dados, a camada intermediária, que contem as regras de negócio e por último a camada de interface, que contem as interfaces do sistema, como telas, páginas web, etc.

A separação das regras de negócio em uma camada, facilitou a organização do código, isolando o código que será utilizado pelos desenvolvedores, dos códigos que serão alterados pelos designers, além de possibilitar o escalonamento desta camada.

- Linux
- SOA

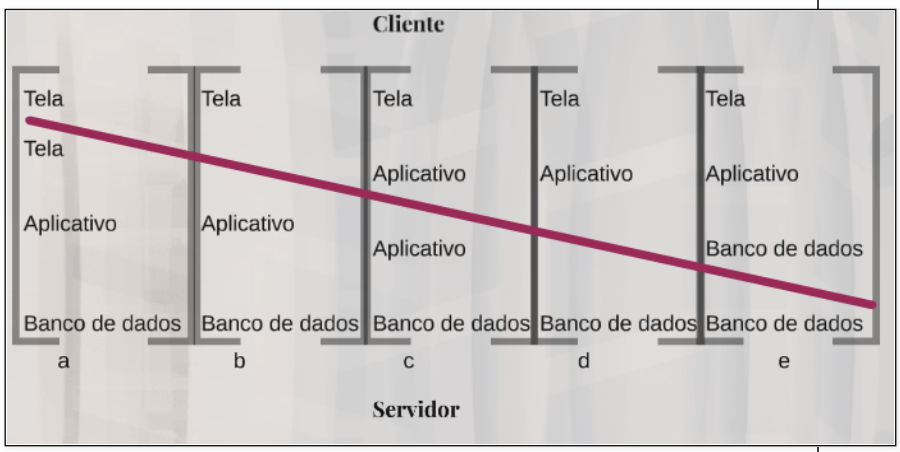
Arquivo do blog

► 2020 (8)
► 2018 (5)
► 2017 (4)
► 2016 (5)
► 2015 (16)
▼ 2014 (34)
► Dezembro (1)
► Novembro (8)
► Outubro (10)
► Setembro (5)
► Maio (3)
► Abril (3)
▼ Março (4)
Arquitetura de Sistemas Distribuídos
Sistemas Distribuídos Comunicação Multicast
Sistemas Distribuídos Comunicação (Socket, stream,...
Sistemas Distribuídos Comunicação (Protocolo de re...

► 2013 (17)

Multicamadas

Na arquitetura multicamada existem diversos tipos de separação que podemos realizar entre o cliente e o servidor, que serão mostradas na figura abaixo:



A linha roxa indica a separação de o que é realizado no servidor e o que é realizado no cliente.

Começando a analise da esquerda para a direita, no primeiro caso *"a"* o servidor tem total controle do que será mostrado ao usuário, neste caso temos uma tela burra que somente mostra informações para o usuário não realiza nenhuma validação.

Uma alternativa a este caso é colocar toda a camada de interface na máquina do cliente, como mostra o caso *"b"*, nestes casos a aplicação será dividida em duas, o front end o qual se comunica com o servidor em um protocolo específico do aplicativo e neste caso o front end ainda não faz nenhum tipo de processamento para apresentar a interface ao usuário. E um aplicativo que será executado no servidor com todas as regras de negócio, algo similar a web 1.0.

No caso *c* é mostrado que também podemos mover parte do aplicativo para o cliente e um exemplo típico deste caso é um aplicativo que tenha um formulário que deve ser preenchido completamente antes de ser processado, neste caso o front end pode checar a consistência dos dados e interagir com o usuário, um exemplo de aplicativo assim são os forms que preenchemos ao se cadastrar em sites que usam a Web 2.0. Se o nome do usuário já está cadastrado, eles já avisam ao usuário pedindo a troca, assim como CPF inválido, etc.

Muitos dos sistemas cliente servidor possuem as organizações mostradas nos casos *d* e *e*. Estes casos são utilizados quando a máquina cliente é um PC que se conectam a um sistema distribuído de arquivos, ou um banco de dados, através de uma rede interna. Essencialmente todas as operações como regra de negócios rodam no cliente, e somente o armazenamento dos arquivos/dados é feito no servidor.

Além da separação física das camadas, cliente-servidor, um sistema multicamadas consiste em separar as camadas de execução do código, como serviços, repositórios, lógica, apis, etc.

Em um sistema multicamadas, pode conter mais de um sistema de armazenamento de dados, como um sistema de cache, ou um sistema de armazenamento de arquivos (imagens, etc).

Arquiteturas Descentralizadas

Cada camada de um aplicativo multicamadas corresponde diretamente a organização das aplicações e a distribuição em camadas é conhecida como distribuição vertical. Esta distribuição vertical consiste em dividir o aplicativo em camadas lógicas que ficam alocadas em diferentes servidores.

Nos sistemas de hoje, o que conta geralmente é como os clientes e os servidores estão distribuídos, o que é chamado de distribuição horizontal, ou também de sistemas ponto a ponto

sistemas podem rodar cada um em um servidor diferente, ou podemos ter serviços classificados por tipos, por exemplo serviços referentes ao usuário, como o cadastro, alteração e remoção da conta do usuário, rodar em um servidor isolado.

Estes serviços podem ser colocados em servidores bem pequenos que ficarão isolados.

Nesta arquitetura tem-se a possibilidade de escalar muito bem cada tipo de serviço, se o sistema está passando por uma grande demanda no cadastro dos usuários, podemos replicar as máquinas que serão responsáveis por este tipo de serviço e esta demanda será atendida.

Este comportamento é bem diferente de um sistema onde todos os serviços ficam em uma mesma máquina, neste caso para atender uma demanda destas, deveríamos replicar todos os serviços do sistema.

Como desvantagens nesta arquitetura estão a maior dificuldade de administrar uma maior quantidade de sistemas rodando em paralelo, assim como repositório de código, entre outros. Um outro fator que pode ser citado é, dependendo de como foi implementado o software, o compartilhamento de código comum pode ficar complicado, e decisões erradas de arquitetura podem levar a reescrever boa parte do código.

Arquiteturas ponto a ponto Estruturadas

Neste tipo de arquitetura os componentes são organizados dentro de uma rede e eles são construídos através da configuração de rede. A configuração mais utilizada é a de Tabela Hash Distribuída. Neste sistema os itens de dados recebem uma chave randômica de 128 ou 160 bits. Além dos dados cada ponto, ou cada nó da rede, também recebe uma chave randômica. O ponto chave deste tipo de sistema é conseguir implementar um sistema eficiente e determinístico em que cada chave aponta para apenas um objeto.

Em um sistema de Chord (Acorde) os nós ficam disponíveis em círculo já que cada nó aponta, pelo menos, para o seu sucessor.

Porém caso cada nó contivesse somente o seu antecessor, uma busca por um nó poderia ter que percorrer a rede inteira até encontrá-lo, por isso no sistema Chord cada nó também contém uma tabela que aponta para os outros nós do círculo, sendo que esta tabela pode conter até todos os elementos do circulo, formando uma espécie de cache dos endereços dos componentes do sistema. Com esta tabela o tempo de busca de um nó dentro de um Chord cai consideravelmente.

Para um novo nó se juntar a rede, ele precisa saber somente o dado do seu antecessor, já quando um nó deixa a rede, uma reorganização deverá ser feita, já que o seu sucessor deverá alterar o seu antecessor para o antecessor do nó que está deixando a rede.

Content Addressable Network (CAN) é um outro sistema baseado em DHT, que foi desenvolvida para ser escalável, tolerante a falhas e auto organizável. O design desta rede é um plano cartesiano de coordenadas virtuais, onde são colocados os endereços virtuais dos pontos de rede o que não tem nenhum relacionamento com a localização física desta rede. Cada ponto é uma área do plano cartesiano e cada ponto da rede contém os endereços de todos os seus vizinhos e ele usa o posicionamento virtual para estabelecer a melhor rota entre dois pontos de uma rede. Quando um novo ponto é adicionado ele pega um valor arbitrário de um ponto, o qual terá a sua área divida em dois, sendo que o ponto novo e o ponto escolhido, ficarão cada um com uma metade da sua área anterior.

Porém quando um ponto sai da rede um dos seus vizinhos assumirá a sua área, o que nem sempre gerará um retângulo. Neste caso todos os vizinhos do nó que está deixando a rede é informado do seu novo vizinho, gerando uma distorção no caminho da rede. Por isso existe um processamento que roda em background para reorganizar a rede, que neste caso irá cuidar da reorganização daquela área da rede.

Arquiteturas de ponto a ponto não estruturadas

Sistemas de ponto a ponto não estruturados utilizam algoritmos randômicos para conectar os pontos, fazendo com que cada ponto tenha uma lista de pontos vizinhos que é construída randomicamente. Esta lista possui um parâmetro de tempo de vida, o qual indica quão velho é a conexão com aquele nó. Os nós, quando se comunicam, trocam informações sobre seus vizinhos.

Em um algoritmo que cuida de uma rede não estruturada, existem threads que varrem estas listas tentando otimizar as conexões destas redes.

O principal fator da troca de informações entre dois nós, é a construção de uma lista otimizada, para isso eles devem decidir quais informações descartar na criação da nova lista de vizinhos. Existem muitas formas de se fazer isso, incluindo o descarte das informações que foram enviadas ao outro nó, descartar os nós mais antigos.

Como não existe uma maneira especifica de determinar uma rota entre dois pontos de uma rede não estruturada, este tipo não é escalável.

Superpontos

Os superpontos foram uma solução proposta para resolver o problema de escalabilidade de redes não estruturadas. Nesta proposta os superpontos são nós que contém as ligações para os seus vizinhos, e qualquer nós que queira contactar algum nó em outra rede, deverá antes passar por um superponto. Neste tipo de rede, todo nó comum deve estar conectado a um superponto.

Qualquer ponto que se junte a rede, deverá se conectar a um superponto, e ao deixá-la ele somente deverá atualizar a conexão com o superponto.

O grande desafio desta arquitetura é como eleger os pontos que serão usados como superpontos, uma vez que um superponto deverá ser confiável e duradoura.

Arquiteturas hibridas
Até aqui abordamos separadamente arquiteturas ponto a ponto e de sistemas cliente servidor, mas muitos dos sistemas distribuídos utilizam um mix destas arquiteturas, unindo estes dois tipos de arquiteturas torna-se possível aproveitar os pontos positivos de cada uma delas.
Sistemas Edge Server

Os sistemas de Servidores Edge são um dos exemplos híbridos a serem mostrados neste post. Estes sistemas fazem parte da arquitetura de provedores de internet.

Os provedores de internet são utilizados por muitos usuários, incluindo empresas inteiras e também usuários caseiros. Estes servidores são *deployados* no ponto de conexão com a internet, e a principal função deles é servir conteúdo, aplicando-se alguns filtros e funções de transcrição. Os servidores de edge também são utilizados para replicar parte dos dados que ele pegam da internet, para que com isso, diminua a necessidade de conexão e de transporte de dados entre os provedores e a internet em si. Uma vez tendo a informação mais perto, no caso armazenada no provedor, o usuário final terá um acesso mais rápido as informações, pois ao invés de sua requisição ter que passar pela internet, ela já estará armazenada no seu provedor.

Sistemas Colaborativos Distribuídos

Estes sistemas proveem um sistema descentralizado de distribuição de conteúdo. Um bom exemplo deste tipo de sistema é o BitTorrent, que é um sistema de download ponto a ponto. Neste sistema os arquivos são quebrados em pacotes e quando um usuário deseja pegar algum arquivo, ele poderá baixar vários pacotes de lugares diferentes. Um dos principais pontos do design do BitTorrent é a colaboração, pois uma vez que um usuário não compartilha nada, o servidor irá baixar a velocidade com a que ele realiza os downloads.

Para fazer um download de um arquivo, o usuário deverá acessar um diretório global, que contém os arquivos .torrent. Estes arquivos carregam as informações sobre o arquivo que o usuário deseja baixar, incluindo as informações conhecidas com tracker.

Um tracker é um servidor que o usuário deverá se conectar para saber onde se encontram os nós ativos que estão provendo partes deste arquivo. Os trackers ordenam os nós de acordo com a velocidade de conexão dele, a quantidade de arquivos compartilhados, etc.

Claramente o BitTorrent faz a utilização de sistemas centralizados e sistemas descentralizados.

Um outro exemplo clássico de sistemas colaborativos é o Globule, que se trata de um sistema de colaboração de conteúdo distribuído. Este sistema se baseia nos sistemas de arquitetura Edge discutidos anteriormente. Porém neste caso, ao invés de provedores de internet disponibilizarem os servidores Edge, usuários comuns e empresas disponibilizam servidores web onde o conteúdo da rede Globule será armazenado. Estes servidores devem conter as seguintes funcionalidades:

O middleware fica em uma camada entre os aplicativos e as plataformas dos sistemas distribuídos, e o que geralmente acontece é que eles seguem um estilo de arquitetura específico. Com isso o desenvolvimento de sistemas fica mais simples, entretanto nem sempre o estilo de arquitetura do middleware condiz com o aplicativo que o desenvolvedor irá fazer.

Interceptadores

Os interceptadores são um tipo de software que desvia o fluxo de um aplicativo, permitindo a execução de um código específico, entre duas camadas. Alguns exemplos de interceptadores incluem particionamento dos pacotes e a sua montagem posterior, transformações na mensagem a fim de adaptá-la, incluindo modificações de encoding, etc.

Monitoramento

Um dos principais pontos de um sistema distribuídos é o monitoramento, uma vez que o hardware, o software, a rede a energia podem falhar e esperamos que os sistemas continuem funcionando quando estas falhas acontecem.

Estes sistemas necessitam de um monitoramento contínuo para que em qualquer uma destas falhas, alguma atitude seja tomada a fim de corrigir o problema.

Além de monitorar a energia, a rede e o hardware estão funcionando, o sistema também deve monitorar se o hardware não está sendo super utilizado ou subutilizado. Isso acontece pois este dois casos afetam o negócio, o primeiro com um baixo desempenho, e o segundo com um custo maior do que o necessário.

O monitoramento deve prover respostas de se o hardware está sendo suficiente para a execução do sistema, ou se mais máquinas, ou máquinas mais potentes serão necessárias.

Sistemas distribuídos auto administrados

Sistemas Auto administrados são aqueles em que existe uma camada de software responsável por tomar atitudes que resolvem as falhas detectadas, por exemplo em um sistema distribuído de múltiplas zonas (quando o sistema está em mais de um datacenter em locais diferentes), quando um dos seus datacenters tenha problemas e fique offline, o sistema automaticamente redistribuí as chamadas para o datacenter que está funcionando e se necessário aumenta a quantidade de máquinas naquele datacenter automaticamente, sem interferência humana, para que este problema não afete o desempenho do software.

Quando acontecer um pico de demanda, o dispositivo que administra o sistema deverá aumentar a quantidade de máquinas disponíveis, e quando houver uma queda na demanda, o sistema deverá diminuir as máquinas disponíveis e assim o gasto com máquinas será otimizado evitando desperdícios ou mesmo lentidão.

Mesmo tendo um software administrando o sistema, é sempre bom ter alguém responsável para ficar de sobreaviso caso o software de monitoramento falhe ou mesmo alguma falha que o sistema não esteja preparado para resolver aconteça.

Sistemas distribuídos com gerenciamento manual

Um sistema com gerenciamento manual possui as mesmas características de um sistema auto administrado a grande diferença entre os dois é que aqui a administração do sistema depende de um humano, com isso qualquer decisão de modificação será tomada por uma pessoa, ao invés de por um software.

Neste sistema pessoas ficam de olho nos dados providos pelos monitores do sistema e quando algo não está funcionando da maneira esperada o funcionário responsável pelo sistema deverá tomar alguma atitude para corrigir a falha.

Arquitetura de Micro Serviços (Microservices)

Com a evolução da internet e o aparecimento dos serviços as arquiteturas evoluíram afim de se adaptar as novas demandas. No início todos os serviços estavam alocados em apenas um servidor, o que é chamado de arquitetura monolítica, então quando um aplicativo tinha uma alta demanda, por exemplo, para um serviço específico, uma nova máquina deveria ser criada e todos os serviços sobem junto com esta máquina.

Os servidores web também evoluíram e alguns deles ocupam bem pouca memória e

quando um usuário entra em na página principal de um ecommerce, o que pode acontecer quando um email de promoção é disparado para os clientes. Para fazer com que a experiência do usuário, seja a melhor possível, e o tempo de espera para carregar a página não seja longo, réplicas do servidor que contem este serviço devem ser inicializadas, ou em máquinas novas, ou mesmo em máquinas cuja capacidade de processamento estava ociosa.

Nesta arquitetura, a resposta a demandas específicas é bem mais curta e o desperdício de recursos também, já que havendo uma demanda específica apenas aquele serviço vai subir novamente.

Porém, como há muitos servidores rodando ao mesmo tempo, a administração do ambiente se torna mais complexa, já que temos diversos servidores rodando para manter em pé, apenas um aplicativo.

A grande diferença desta arquitetura para uma arquitetura monolítica, onde todos os serviços ficam dentro de um mesmo servidor, é que nela podemos configurar melhor e fazer com que o aproveitamento do hardware seja o melhor possível.

Arquitetura sem servidor (Serverless)

Atualmente existe uma diversificação dos serviços disponibilizados pelos provedores de Cloud. A cada dia um novo serviço surge, e estes serviços podem ser utilizados para executar parte, ou totalmente a lógica de negócio de um aplicativo/site. No início estes serviços foram descritos como Backend as a Service (BaaS).

Inicialmente, o termo serverless foi utilizado para descrever aplicações que dependem muito, ou totalmente, destes serviços dos provedores de cloud para rodar.

Uma outra forma de definirmos a arquitetura Serverless são aplicativos os quais a lógica, ao invés de rodar em servidores que ficam sempre ligados e conectados, elas rodam em serviços que são oferecidos pelos provedores cloud e são disparados por evento, ou são do tipo ephemeral (rodam apenas uma vez) e são totalmente administrados pelo provedor. Estes serviços são conhecidos como Function as a Service (FaaS).

Todos estes serviços disponibilizados pelos provedores de cloud são auto escaláveis, ou seja, eles conseguem responder a grandes variações nas demandas de utilização do seu serviço.

Dentre as principais vantagens desta arquitetura é a facilidade de desenvolvimento, o rápido tempo para colocar algo em produção, não precisar tomar conta dos servidores, planejar alocação ou desalocação de máquinas. Poder desenvolver um software com várias linguagens, já que cada serviço é independente um do outro.

Também podemos citar como vantagem o fato de que os serviços só cobram por tempo de execução, portanto, se você tiver um site/app onde parte do dia ele não recebe visitas nada será cobrado neste período.

Mas este tipo de arquitetura não se aplica para todos os casos e também deve-se analisar as suas desvantagens antes de começar a desenvolver algo sobre ela. Dentre as principais desvantagens temos a dificuldade em debugar uma aplicação de forma local, como não teremos a mesma arquitetura disponível para os desenvolvedores isto pode ser um grande problema. Como não temos controle de onde estes serviços estão rodando, teremos uma latência maior, ou seja o tempo de execução do serviço será maior. Os provedores possuem um limite de memória e tempo de execução para cada função e isto pode ser um problema em alguns casos. Você ficará preso ao ser provedor de cloud, uma vez que cada um possui uma forma diferente destes serviços e migrar de um para o outro pode resultar em ter que reescrever todo o seu software.

Apresentação

Postado por Dirceu Semighini Filho às 19:15

Marcadores: P1SD, Sistemas Distribuídos

Nenhum comentário:

Postagem mais recente Página inicial Postagem mais antiga

Assinar: Postar comentários (Atom)

Dirceu Semighini Filho

Dirceu Semighini Filho

Ver meu perfil completo