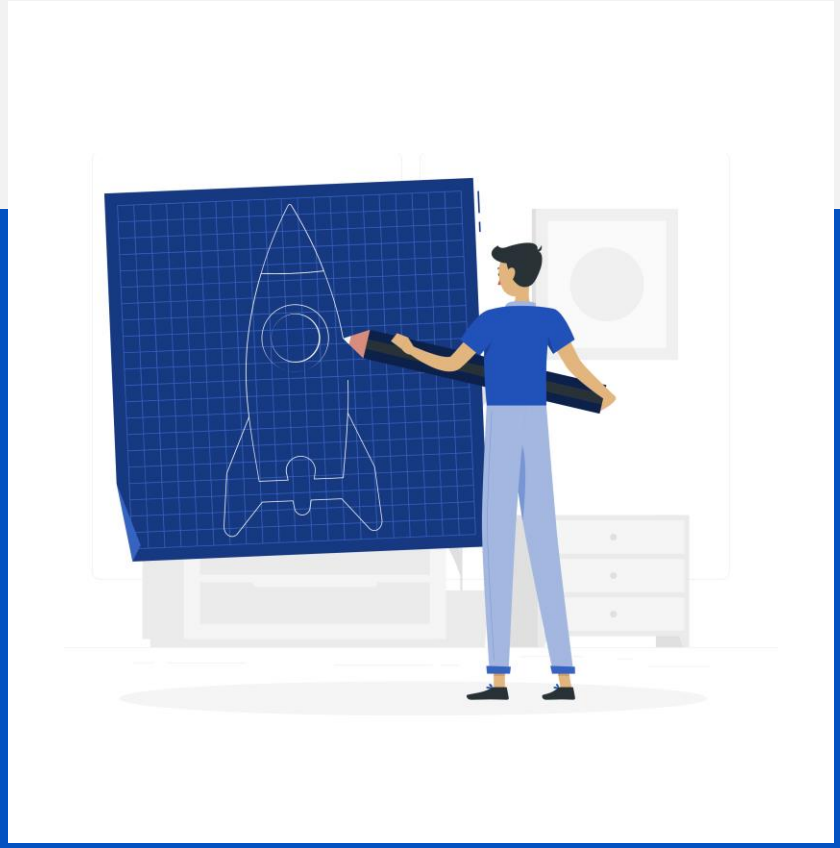
escola britânica de artes criativas & tecnologia



Módulo 11

Conceitos avançados para microsserviços – Parte 2

Módulo 11

Aulas 1 e 2 – Observabilidade (logs e métricas)

- Tão importante quanto garantir o bom desenvolvimento de um sistema é ser capaz de observar, a qualquer momento, o estado do mesmo.
- Por isso, observabilidade é a habilidade de se medir o estado atual de um sistema baseado nos dados que este gera, como logs, métricas e traces.
- Cada hardware, software e componente de infraestrutura de nuvem, assim como cada contêiner e microsserviço, gera registros de todas as atividades que executam.
- O objetivo da observabilidade é entender o que está acontecendo através de todos os ambientes e tecnologias de uma empresa para que os times de desenvolvimento possam detectar e resolver problemas antes que os usuários finais sejam impactados.

Módulo 11

Aulas 1 e 2 – Observabilidade (logs e métricas)

- Observabilidade ajuda os times a manterem seus sistemas eficientes e confiáveis ajudando-os a entender o que está acontecendo em ambientes altamente distribuídos.
- Além disso, permite que se veja o que está lento ou quebrado e o que precisa ser feito para melhorar sua performance.
- Independentemente de sua empresa usar microsserviços ou não, é absolutamente crucial ter mecanismos de observabilidade implementados.
- Lembre-se de que, se você não se antecipar e descobrir problemas em produção antes deles acontecerem, quem fará isso por você será o seu cliente!

Módulo 11

Aulas 1 e 2 – Observabilidade (logs e métricas)

- São três os pilares de observabilidade: logs, métricas e traces. Falaremos deles separadamente a seguir.
- Logs são absolutamente essenciais a qualquer aplicação e muito úteis para depurar algum incidente ou problema em produção. Servem para mostrar parâmetros de request/response, como controles de fluxo (se passou em determinado método), etc.
- Sem logs, não há como descobrir o que houve em ambiente produtivo. Você terá que tentar adivinhar quais foram os parâmetros de entrada e como foi que o sistema se comportou com eles, o que pode tomar muito tempo até que o problema seja encontrado e resolvido (isso se você encontrar).
- Logs podem ter vários níveis (INFO, DEBUG, ERROR, etc.) e suas ferramentas de observabilidade podem se aproveitar disso para armazená-los de forma inteligente dependendo do ambiente. Por exemplo, produção poderia conter apenas INFO e ERROR.

Módulo 11

Aulas 1 e 2 – Observabilidade (logs e métricas)

- Logs devem ser úteis e não devem ser muito verbosos. Lembre-se de que armazenar logs gera custos de armazenagem e que longos textos sendo logados aumentam a quantidade de dados sendo armazenados por eles!
- Logs úteis devem conter elementos únicos e que possam identificar facilmente os fluxos pelos quais uma requisição passou e quem foi o chamador. Como exemplos, temos IDs de transação ou de usuários, nomes de métodos, etc.
- Ter uma ferramenta de pesquisa e indexação de logs é fundamental para facilitar o trabalho de depuração dos desenvolvedores. Ninguém quer ter que se conectar a contêineres individuais para ver logs!
- Deve-se também ter algum mecanismo de filtragem antes da indexação de logs. Você não vai querer ter dados de usuários (nome, endereço, etc.) ou de cartões de crédito sendo logados e indexados abertamente para todos da empresa verem, não é mesmo?
- Logs devem estar presentes em todas as chamadas ao seu sistema!

Módulo 11

Aulas 1 e 2 – Observabilidade (logs e métricas)

- Já métricas definem os dados sobre o estado de seu serviço/sistema. Tão importante quanto os logs, servem para que os desenvolvedores possam detectar problemas ANTES deles acontecerem!
- Exemplos de métricas incluem health checks, status de retorno de endpoints de uma API (200, 400, 500, etc.), latências (P50, P99), uso de disco ou de CPU, perda de pacotes de rede, etc.
- Métricas normalmente são mostradas em ferramentas que provêm dashboards para consultas dos dados gerados por elas (Grafana é uma delas).
- Métricas podem ser tanto de aplicação (como as mencionadas anteriormente), como de negócio (geradas manualmente através da agregação de logs e outras métricas).

Módulo 11

Aulas 1 e 2 – Observabilidade (logs e métricas)

- Exemplo de métricas de negócio: transações de crédito e débito, número de pagamentos negados por operadora, quantidade de clientes que se cadastraram na última hora, etc.
- Métricas de aplicação são úteis para que se identifiquem problemas assim que eles começarem a acontecer. Latências P99 podem ser úteis para pegar a degradação de um endpoint, status de retorno de uma API podem pegar um deployment ruim, entre outros.
- Ter somente logs pode não ser o suficiente para se antecipar a problemas ou entender o que está acontecendo com seus sistemas.
- Idealmente, a geração de métricas deve ser incorporada já na fase de design de sua aplicação. Se todos os serviços novos já puderem nascer com ferramentas para geração de métricas embutidas no código, melhor ainda!
- Assim como logs, é essencial ter uma ferramenta onde todos na empresa possam ver as métricas de seus sistemas.

Módulo 11

Aulas 1 e 2 – Observabilidade (logs e métricas)

- Métricas também podem ser usadas para a criação de alertas para seus sistemas. Alertas servem para que os desenvolvedores possam saber de problemas sem que eles precisem fazer consultas ativamente às ferramentas que mostram essas métricas.
- Alertas podem ser consultados posteriormente e usados em reuniões de post-mortem para que providências sejam tomadas para que os problemas detectados por eles não voltem a acontecer.

Módulo 11

Aulas 1 e 2 – Observabilidade (logs e métricas)

- Por fim, traces são importantes para saber por onde uma requisição passou. Eles identificam, fim-a-fim, todos os sistemas por onde uma chamada passou.
- Em ambientes distribuídos, é muito comum que uma mesma requisição passe por múltiplos sistemas. Imagine um cenário em que, para cadastrar um usuário, o sistema A chame o sistema B que, por sua vez, chama o sistema C para completar a operação.
- Como cada sistema é de um time diferente, os logs de cada um deles podem ser diferentes entre si e não necessariamente conterem todas as informações de que você precisa para investigar um problema.
- É aí que entra o trace. Neste caso, também chamado de trace distribuído, dado que será logado por múltiplos serviços.

Módulo 11

Aulas 1 e 2 – Observabilidade (logs e métricas)

- Ferramentas de tracing normalmente adicionam um header HTTP às chamadas com um identificador único que deveria ser logado por todos os sistemas.
- Desta forma, ao pesquisar-se por este identificador único, conseguiríamos saber todos os logs associados a ele independentemente do sistema pelo qual a requisição passou.
- É comum também ter um sistema centralizador de traces distribuídos. Este sistema seria o responsável por agregar os logs de cada sistema com um mesmo trace e mostrá-lo numa tela para o usuário mediante pesquisa pelo identificador do trace.
- Infelizmente, normalmente é o pilar de observabilidade mais esquecido pelas empresas dada a dificuldade inicial de se estabelecer o ferramental adequado para isso.

Aulas 1 e 2 – Observabilidade (logs e métricas)

- Por fim, assim como o pilar de métricas, idealmente deveria ser implantado na fase de design das aplicações e sistemas para que todos já pudessem nascer com a capacidade de gerar traces distribuídos.
- Nos screencasts do módulo, veremos alguns exemplos de logs, métricas e traces.

Módulo 11

Aulas 3 e 4 – Descoberta de serviços

- Em um mundo cheio de microsserviços, começa a surgir a necessidade de termos um meio de fazer com que um serviço encontre o outro sem exatamente saber onde cada um está rodando.
- Um microsserviço normalmente roda em ambientes virtualizados ou dentro de contêineres. O número de instâncias de um serviço e sua localização (endereços de IP, nomes de máquina) mudam dinamicamente, o que faz com que o processo de encontrá-lo fique mais difícil.
- O mecanismo de descoberta de serviços nos ajuda a descobrir onde cada instância de um serviço está localizada, agindo como uma espécie de registro no qual todos os endereços de todas as instâncias são rastreados.
- Lembre-se dos 12 fatores que estudamos anteriormente: um microsserviço deveria ser derrubado e reiniciado sem qualquer prejuízo para a aplicação. Isso implica também em endereços de IP dinâmicos para as instâncias.

Módulo 11

Aulas 3 e 4 – Descoberta de serviços

- Sem saber o endereço e porta pelos quais uma instância responde, não há como chamar um serviço. E é aqui onde o mecanismo de descoberta de serviços se torna essencial.
- E como este mecanismo funciona? Normalmente, provendo duas operações distintas: o registro das instâncias de um serviço e uma forma de encontrar este serviço após ele ser registrado.

- É mais ou menos assim:
  1. Serviço A precisa chamar o serviço B.
  2. Serviço B se registra no mecanismo de descoberta de serviços.
  3. Serviço A pergunta ao registro de serviços onde o Serviço B está.
  4. O registro de serviços busca a localidade do Serviço B e a devolve ao serviço A.
  5. Serviço A, então, chama o Serviço B através do endereço retornado pelo registro de serviços.

Módulo 11

Aulas 3 e 4 – Descoberta de serviços

- É importante conhecer dois conceitos que envolvem um serviço chamando o outro: o de upstream e o de downstream.
- Serviços upstreams são aqueles que chamam o seu serviço.
- Serviços downstream são aqueles que o seu serviço chama (ou seja, ele vira o serviço upstream deles).

- Existem basicamente duas formas de se implementar um mecanismo de descoberta de serviços e elas dependem de onde partirá a ordem de descobrir os serviços: do lado do cliente ou do lado do servidor.
- No caso de implementações do lado do cliente, o serviço upstream é o responsável por determinar as localidades de rede de seus serviços downstream e por fazer o balancemento de carga entre eles.

Aulas 3 e 4 – Descoberta de serviços

- Este balanceamento de carga normalmente é feito com o algoritmo de round robin, ou seja, as chamadas são feitas às instâncias de forma alternada e uma a uma.
- Contudo, isso pode trazer uma séria desvantagem: todos os serviços de sua empresa deverão implementar as mesmas regras de descoberta de serviços e de balanceamento de carga.
- No caso de implementações do lado do servidor, é necessária uma entidade intermediária para fazer o papel do Balanceador de Carga (load balancer). Serviços upstream, então, fazem suas requisições ao balanceador de carga e não ao registro de serviços. E cabe ao balanceador de carga consultar o registro de serviços para descobrir os endereços e, então, balancear as requisições entre as instâncias encontradas.

Aulas 3 e 4 – Descoberta de serviços

- Isso permite que os serviços upstream não precisem ter uma lógica de descoberta e balanceamento de carga, o que também diminui o acoplamento entre eles e seus downstreams.
- Este é o tipo de implementação mais comum nas empresas e normalmente o mais aconselhado. O preço de se ter um balanceador de carga se paga facilmente através dos benefícios desta implementação.
- Por fim, precisamos falar sobre o registro de serviços. Existem basicamente duas formas de se implementar isto: através do auto-registro ou do registro através de terceiros (third-parties).
- No primeiro caso, cada instância de serviço é responsável por registrar-se (ou remover-se) no/do registro de serviços. Para manter seu registro ativo, pode enviar um heartbeat (uma requisição que indica que a instância está viva).
- Apesar de simples, acopla as instâncias dos serviços ao registro de serviços, fazendo com que esta lógica precise ser implementada em cada uma das linguagens e frameworks utilizados pelos diferentes serviços da empresa.

Módulo 11

Aulas 3 e 4 – Descoberta de serviços

- Já no registro através de terceiros, é o registro de serviços o responsável por observar todas as mudanças nas instâncias atuais. Isso normalmente é feito através de operações de polling (chamando métodos de health check dos serviços) ou escutando eventos gerados pelo ambiente de deployment.
- Caso o health check de uma instância falhe, ela é automaticamente removida do registro de serviços, dado que ela provavelmente não será capaz de atender a nenhuma requisição.
- A grande desvantagem deste método é ter que manter o registro de serviços como um serviço de alta disponibilidade. Se ele cair, caem todos os demais serviços, dado que ninguém conseguirá chamar ninguém.
- Entretanto, costuma ser a forma mais comum de se implementar o registro de serviços.
- Nos screencasts deste módulo, veremos um pouquinho destes mecanismos em ação.

Módulo 11

Aulas 5 e 6 – Gateways de API

- Um gateway de API é um software que fica em frente a uma API ou grupo de microsserviços. Pense nele como um ponto de entrada para todas as requisições aos serviços de sua empresa.
- Sua função principal é, como dito acima, atuar como ponto único de entrada e, também, padronizar as interações entre os serviços, aplicações e dados de uma empresa e seus clientes internos e, principalmente, externos.
- Em um ambiente de microsserviços, um gateway de API é absolutamente essencial, pois provê uma série de serviços em um único pacote, tais como:
  1. Tradução de protocolos
  2. Descoberta de serviços
  3. Autenticação e autorização do uso de APIs
  4. Balanceamento de carga
  5. Gerenciamento de caches
  6. Monitoramento (logs e métricas)

Módulo 11

Aulas 5 e 6 – Gateways de API

- Imagine o seguinte cenário: para buscar os dados completos de um cliente, é necessário chamar pelo menos três serviços diferentes. Imagine fazer com que um cliente externo chame estes três serviços diferentes apenas para obter esta informação... Isso provavelmente afetaria a adoção de sua API.
- Um gateway de API pode expor um único endpoint para seus clientes externos e orquestrar a chamada a estes três serviços mencionados acima para obter e traduzir a resposta antes de devolvê-la aos chamadores.
- Ele também pode, ainda usando este cenário hipotético de pesquisa de usuários, cachear este resultado de acordo com os filtros usados na entrada, o que eliminaria completamente a necessidade de chamar os serviços internos em caso de consultas repetidas.
- Outra grande vantagem de um gateway de API é permitir que os times de sua empresa cresçam seus sistemas sem necessariamente se preocupar em manter um mesmo padrão de tecnologias. Mas como?

Módulo 11

Aulas 5 e 6 – Gateways de API

- Imagine que o time A de sua empresa seja dono do serviço A, implementado em Java e com comunicação via http. Este serviço provê o endpoint /service/a.
- Há ainda um time B que é dono do serviço B, implementado em Go e com comunicação via gRPC. Este serviço provê o endpoint /service/b.
- Agora, imagine que estes dois serviços sejam responsáveis por retornar os dados de um pagamento, ou seja, ambos precisam ser chamados para termos os dados completos.
- O gateway de API pode orquestrar essas chamadas e, ainda, traduzir os protocolos utilizados para um protocolo em comum. Entretanto, a principal vantagem vem agora!
- Imagine que o time B decidiu mudar o endpoint responsável pelos dados dos pagamentos para /service/novo-b. Ao invés de fazer com que seus clientes externos mudem as integrações deles, é possível mudar uma única configuração no gateway de API para que este chame o novo endpoint do serviço B ao invés do antigo, desde que os dados retornados pelo novo sejam iguais aos do anterior.

Aulas 5 e 6 – Gateways de API

- Mesmo que não fossem, implementar um mecanismo de tradução dos novos dados aos antigos seria um esforço pequeno perto do de fazer seus clientes mudarem as integrações deles.
- Lembre-se de uma coisa: se seus clientes precisam dispender esforço para mudar as APIs que eles chamam de sua empresa, eles podem fazer o mesmo com APIs de seus concorrentes!

- Assim sendo, as principais vantagens de um gateway de API são:
  1. Simplificam a entrega de serviços (como nos exemplos que acabamos de ver).
  2. Fornecem flexibilidade por conta da facilidade de configuração.
  3. Permitem que aplicações legadas possam viver por mais tempo ao invés de investir-se em longas e custosas migrações.
  4. Contribuem para a observabilidade geral da empresa, dadas as métricas que geram por definição.

Módulo 11

Aulas 5 e 6 – Gateways de API

- Contudo, devemos prestar atenção aos seguintes desafios quando usamos um gateway de API:
  1. Qualquer indisponibilidade do gateway de API pode causar uma falha catastrófica nos serviços por trás dele.
  2. Caso o gateway tenha sua segurança comprometida, um atacante poderá ter acesso a todas as APIs de sua empresa.
  3. É necessário atualizar o gateway de API toda vez que um serviço é adicionado, modificado ou removido.
- Como sempre, não existe almoço grátis, não é mesmo? :(

escola
britânica de
artes criativas
& tecnologia

Obrigado!