Microservices (Hands-ON)

FIAP SHIFT

2023

Abril, 2023

MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

4 MÓDULOS | 16 HORAS

LIVE

MÊS	VALOR	OK
FEVEREIRO	184,00	
MARÇO	184,00	
ABRIL	184,00	
MAIO	184,00	
JUNHO	184,00	
JULHO	184,00	
AGOSTO	184,00	
SETEMBRO	184,00	
OUTUBRO	184,00	
NOVEMBRO	184,00	

MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

4 MÓDULOS | 16 HORAS

LIVE

PROPOSTA

DOMINE ARQUITETURA DE MICROSERVICES NA PRÁTICA

MICROSERVICES É UM ESTILO DE ARQUITETURA PARA DESENVOLVIMENTO DE SISTEMAS BASTANTE UTILIZADO NO AMBIENTE DE CLOUD PARA DEV. WEB. SIM, O MERCADO ESTÁ SUBSTITUINDO O SOA PELO MICROSERVICES. E ALGUNS COMPONENTES SÃO ESSENCIAIS PARA ESSA MUDANÇA: API GATEWAY E USO DE CONTAINERS.

ESTE CURSO É HANDS-ON E TEM COMO FOCO A FORMAÇÃO DE PROFISSIONAIS QUE DOMINAM A ARQUITETURA DE MICROSERVICES E SEUS COMPONENTES. VOCÊ VAI IMPLEMENTAR ESTA ARQUITETURA POR MEIO DA CODIFICAÇÃO DE MICROSERVICES EM DIFERENTES LINGUAGENS, COMO JAVA, PYTHONE JAVASCRIPT (EXECUTADO EM AMBIENTE NODE.JS).

UTILIZAREMOS:

- DOCKER CONTAINERS PARA CONSUMIR E PRODUZIR UMA API RESTFUL
- STACK ELK NO MONITORAMENTO DE PERFORMANCE DE APLICAÇÕES (APM), PRINCIPALMENTE SUAS APIS

AO FINAL DO TREINAMENTO, VOCÊ VAI DESENVOLVER APLICAÇÕES QUE SIGAM OS PADRÕES DE PROJETO CONFORME AS BOAS PRÁTICAS DESTA ARQUITETURA. E SABERÁ COMO MIGRAR UMA APLICAÇÃO MONOLÍTICA PARA A ARQUITETURA DE MICROSERVICES.

MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

4 MÓDULOS | 16 HORAS

LIVE

FOCO
ENTENDA A ARQUITETURA DE MICROSERVICES E SEUS COMPONENTES, E EXECUTE A IMPLEMENTAÇÃO NAS LINGUAGENS JAVA, PYTHON E JAVASCRIPT (EXECUTADO EM AMBIENTE NODE.JS).

PARA QUEM
PROFISSIONAIS DEVOPS E ARQUITETOS DE TI QUE QUEREM TER EXPERIÊNCIA HANDS-ON NO DESENVOLVIMENTO E UTILIZAÇÃO DA ARQUITETURA DE MICROSERVICES.

FORMATO
LIVE
SUAS AULAS SÃO ENCONTROS ONLINE SEMPRE AO VIVO. ISSO SIGNIFICA QUE EM TODAS AS ETAPAS DO CURSO, AS AULAS SERÃO EM TEMPO REAL. VOCÊ TERÁ CONTATO DIRETO COM O PROFESSOR, PODERÁ TIRAR SUAS DÚVIDAS MOMENTO A MOMENTO, FAZER NETWORKING E AINDA EVOLUIR TROCANDO IDEIAS COM A SUA TURMA.

PRÉ-REQUISITOS
FORMAÇÃO EM TI (SERÃO ABORDADOS TEMAS COMO REDES, SOA, CLOUD, BANCO DE DADOS E INTEGRAÇÃO DE SISTEMAS). EXPERIÊNCIA MÍNIMA EM QUALQUER LINGUAGEM DE PROGRAMAÇÃO E USO DO DOCKER.

MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

4 MÓDULOS | 16 HORAS

LIVE

MÓDULO 1: MICROSERVICES

- DEFININDO MICROSERVICES
- SOA VERSUS MICROSERVICES
- A ARQUITETURA DE MICROSERVICES
- CONSUMINDO MICROSERVICES VIA API WEBHOOKS

MÓDULO 2: DESENVOLVENDO MICROSERVICES

- CONSTRUINDO O MICROSERVICES DA CAMADA DE APRESENTAÇÃO AO BACKEND
- ARQUITETURA REST
- CONTAINERS E MICROSERVICES: FLUXO CI/CD
- MICROSERVICES E DATABASES: QUERIES E CONSISTENCY

MÓDULO 3: MICROSERVICES NO DIA A DIA

- DIVIDINDO O MONOLITICO
- MONITORAMENTO DE MICROSERVICES
- ELASTIC APM, ELASTICSEARCH E KIBANA

MÓDULO 4: LABORATÓRIOS HANDS-ON (ALGUNS ENVOLVEM O USO DE DOCKER)

- CONSTRUÇÃO DE UM MICROSSERVIÇO EM PYTHON
- COMUNICAÇÃO ASSÍNCRONA USANDO KAFKA, NODEJS E PYTHON
- JAVA SPRING DATA REST PARA ENTENDER O HATEOAS
- CI/CD E SONNARQUBE COM O USO DE TESTES AUTOMATIZADOS ATRAVÉS DO CUCUMBER (FRAMEWORK BDD)
- USO DE API GATEWAY PARA AUTENTICAÇÃO E AUTORIZAÇÃO
- OBSERVABILIDADE PARA MONITORAR TEMPO DE RESPOSTAS DE APLICAÇÕES USANDO ELASTIC APM

MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

4 MÓDULOS | 16 HORAS

LIVE

CALENDÁRIO DE AULAS

AULA 1	04 DE ABRIL - TERÇA-FEIRA DAS 19H00 ÀS 23H00
AULA 2	05 DE ABRIL - QUARTA-FEIRA DAS 19H00 ÀS 22H00
AULA 3	06 DE ABRIL - QUINTA-FEIRA DAS 19H00 ÀS 22H00
AULA 4	11 DE ABRIL - TERÇA-FEIRA DAS 19H00 ÀS 22H00
AULA 5	12 DE ABRIL - QUARTA-FEIRA DAS 19H00 ÀS 22H00

AULA	OK
1	
2	
3	
4	
5	

MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

MÓDULO 1

DEFININDO MICROSERVICES

Microsserviços são uma abordagem arquitetônica e organizacional do desenvolvimento de software na qual o software consiste em pequenos serviços independentes que se comunicam usando APIs bem definidas. Esses serviços pertencem a pequenas equipes autossuficientes.

As arquiteturas de microsserviços facilitam a escalabilidade e agilizam o desenvolvimento de aplicativos, habilitando a inovação e acelerando o tempo de introdução de novos recursos no mercado.

Diferenças entre as arquiteturas monolítica e de microsserviços
Com as arquiteturas monolíticas, todos os processos são altamente acoplados e executam como um único serviço. Isso significa que se um processo do aplicativo apresentar um pico de demanda, toda a arquitetura deverá ser escalada. A complexidade da adição ou do aprimoramento de recursos de aplicativos monolíticos aumenta com o crescimento da base de código. Essa complexidade limita a experimentação e dificulta a implementação de novas ideias. As arquiteturas monolíticas aumentam o risco de disponibilidade de aplicativos, pois muitos processos dependentes e altamente acoplados aumentam o impacto da falha de um único processo.

Com uma arquitetura de microsserviços, um aplicativo é criado como componentes independentes que executam cada processo do aplicativo como um serviço. Esses serviços se comunicam por meio de uma interface bem definida usando APIs leves. Os serviços são criados para recursos empresariais e cada serviço realiza uma única função. Como são executados de forma independente, cada serviço pode ser atualizado, implantado e escalado para atender a demanda de funções específicas de um aplicativo.

MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

MÓDULO 1

DEFININDO MICROSERVICES

Características dos microsserviços

Autônomos
Cada serviço do componente de uma arquitetura de microsserviços pode ser desenvolvido, implantado, operado e escalado sem afetar o funcionamento de outros serviços. Os serviços não precisam compartilhar nenhum código ou implementação com os outros serviços. Todas as comunicações entre componentes individuais ocorrem por meio de APIs bem definidas.

Especializados
Cada serviço é projetado para ter um conjunto de recursos e é dedicado à solução de um problema específico. Se os desenvolvedores acrescentarem mais código a um serviço ao longo do tempo, aumentando sua complexidade, ele poderá ser dividido em serviços menores.

Benefícios dos microsserviços

Agilidade
Os microsserviços promovem uma organização de equipes pequenas e independentes que são proprietárias de seus serviços. As equipes atuam dentro de um contexto pequeno e claramente compreendido e têm autonomia para trabalhar de forma mais independente e rápida. O resultado é a aceleração dos ciclos de desenvolvimento. Você obtém benefícios significativos do throughput agregado da organização.

Escalabilidade flexível
Os microsserviços permitem que cada serviço seja escalado de forma independente para atender à demanda do recurso de aplicativo oferecido por esse serviço. Isso permite que as equipes dimensionem corretamente as necessidades de infraestrutura, meçam com precisão o custo de um recurso e mantenham a disponibilidade quando um serviço experimenta um pico de demanda.

MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

MÓDULO 1

DEFININDO MICROSERVICES

Fácil implantação
Os microsserviços permitem a integração e a entrega contínuas, o que facilita o teste de novas ideias e sua reversão caso algo não funcione corretamente. O baixo custo de falha permite a experimentação, facilita a atualização do código e acelera o tempo de introdução de novos recursos no mercado.

Liberdade tecnológica
As arquiteturas de microsserviços não seguem uma abordagem generalista. As equipes são livres para escolher a melhor ferramenta para resolver problemas específicos. O resultado é que as equipes que criam microsserviços podem optar pela melhor ferramenta para cada tarefa.

Código reutilizável
A divisão do software em módulos pequenos e bem definidos permite que as equipes usem funções para várias finalidades. Um serviço criado para uma determinada função pode ser usado como componente básico para outro recurso. Isso permite que os aplicativos sejam reutilizados, pois os desenvolvedores podem criar recursos sem precisar escrever código.

Resiliência
A independência do serviço aumenta a resistência a falhas do aplicativo. Em uma arquitetura monolítica, a falha de um único componente poderá causar a falha de todo o aplicativo. Com os microsserviços, os aplicativos lidam com a falha total do serviço degradando a funcionalidade, sem interromper todo o aplicativo.

MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

MÓDULO 1

DEFININDO MICROSERVICES

A plataforma mais completa para microsserviços

A AWS integrou componentes básicos que oferecem suporte a qualquer arquitetura de aplicativos, independentemente de escala, carga ou complexidade.

Computação

Capacidade de processamento para microsserviços.

Contêineres

Amazon Elastic Container Service

Um serviço de gerenciamento de contêineres com altos níveis de escalabilidade e performance que oferece suporte a contêineres do Docker, além de permitir que você execute facilmente aplicativos em um cluster gerenciado de instâncias do Amazon EC2.

Agora, usando o **Amazon ECS**, o Coursera pode implantar alterações de software em minutos, em vez de horas, em um ambiente isolado de recursos.

Arquiteturas Sem servidor

AWS Lambda

O AWS Lambda permite que você execute código sem provisionar ou gerenciar servidores. Basta fazer upload do código e o Lambda gerencia tudo o que é necessário para executar e alterar a escala do código com alta disponibilidade.

A **Localytics** usou o AWS Lambda para construir microsserviços que permitiram às equipes de desenvolvimento criar análises personalizadas sem o suporte central.

MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

MÓDULO 1

DEFININDO MICROSERVICES

Armazenamento e bancos de dados
Armazenamento físico de dados escalável, resiliente e seguro.

Armazenamento em cache
Amazon ElastiCache
O Amazon ElastiCache aprimora a performance de serviços, permitindo que você recupere informações de caches de memória rápidos e gerenciáveis, em vez de depender inteiramente de bancos de dados baseados em disco, que são mais lentos.

Armazenamento de objetos
Amazon S3
O Amazon S3 disponibiliza a desenvolvedores e equipes de TI um armazenamento de objetos altamente confiável, seguro e escalável para todos os dados, qualquer que seja o volume.

Bancos de dados NoSQL
Amazon DynamoDB
Um serviço de banco de dados NoSQL rápido e flexível para todos os aplicativos que precisam de latência consistente abaixo de 10 milissegundos, em qualquer escala.

Bancos de dados relacionais
Amazon RDS
Configure, opere e escale facilmente um banco de dados relacional na nuvem. Escolha entre seis mecanismos populares de banco de dados, incluindo Oracle, Microsoft SQL Server, PostgreSQL, MySQL e MariaDB.

Amazon Aurora
Um mecanismo de banco de dados relacional que combina a velocidade e a confiabilidade de bancos de dados comerciais avançados com a simplicidade e a economia de bancos de dados de código aberto. Entregue até cinco vezes o throughput do MySQL padrão executando no mesmo hardware.

A **Remind** reduziu os tempos de resposta de aplicativos em 200% criando uma PaaS para microsserviços na Amazon ECS.

MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

MÓDULO 1

DEFININDO MICROSERVICES

Redes

Serviços de redes com alto throughput e latência inferior a um milissegundo.

Descoberta de serviços

AWS Cloud Map

O AWS Cloud Map é uma descoberta de serviços para todos os recursos de nuvem. Com o Cloud Map, você pode definir nomes personalizados para os recursos do aplicativo, e ele manterá a localização atualizada desses recursos em constante mudança.

Malha de serviços

AWS App Mesh

O AWS App Mesh facilita o monitoramento e o controle de microsserviços executados na AWS. O App Mesh padroniza a forma de comunicação dos microsserviços, oferecendo visibilidade completa e ajudando a garantir alta disponibilidade para os aplicativos.

Elastic Load Balancing

Application Load Balancer

O Application Load Balancer executa balanceamento de carga de tráfego HTTP e HTTPS na camada de aplicativos (camada 7) e oferece roteamento avançado de solicitações para a entrega de arquiteturas modernas de aplicativos, incluindo microsserviços e contêineres.

Network Load Balancer

O Network Load Balancer oferece balanceamento de carga de alta performance que opera na camada de conexão de rede (camada 4) e permite rotear conexões para microsserviços com base em dados do protocolo IP. O Network Load Balancer pode tratar milhões se solicitações por segundo mantendo latências ultrabaixas.

Proxy de API

Amazon API Gateway

O Amazon API Gateway oferece uma plataforma abrangente para gerenciamento de APIs. O Amazon API Gateway permite processar centenas de milhares de chamadas de API simultâneas e se encarrega das atividades de gerenciamento de tráfego, autorização e controle de acesso, monitoramento e gerenciamento de versões de APIs.

MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

MÓDULO 1

DEFININDO MICROSERVICES

DNS

Amazon Route 53

O Amazon Route 53 é um web service altamente disponível e escalável de Domain Name System (DNS) na nuvem que conecta eficazmente as solicitações à infraestrutura executada na AWS. O serviço pode ser usado para verificações de integridade de IP e descoberta de serviços para microsserviços.

Após reprojetar seu aplicativo como microsserviços executados na AWS, a **Airtime** disponibiliza sua experiência social para os clientes de modo mais rápido, confiável e sem atrasos.

Sistema de mensagens

Publique e coordene comunicações entre processos.

Publicação e assinatura de mensagens

Amazon Simple Notification Service (Amazon SNS)

O Amazon SNS é um serviço gerenciado de sistemas de mensagens do tipo publicação/assinatura (pub/sub) que facilita o desacoplamento e a escalabilidade de microsserviços, sistemas distribuídos e aplicativos sem servidor.

Enfileiramento de mensagens

Amazon Simple Queue Service (Amazon SQS)

O Amazon SQS é um serviço gerenciado de enfileiramento de mensagens que facilita o desacoplamento e a escalabilidade de microsserviços, sistemas distribuídos e aplicativos sem servidor.

A **Lyft** usa a AWS para ter maior agilidade empresarial e gerenciar seu crescimento exponencial. Com os produtos da AWS, a empresa oferece suporte a mais de 100 microsserviços que aprimoram cada elemento da experiência dos clientes. Saiba mais »

DEFININDO MICROSERVICES

Registro em log e monitoramento

Monitore a performance e a utilização de recursos dos serviços. Rastreie em arquiteturas complexas para resolução de problemas e otimização.

Monitoramento de APIs

AWS CloudTrail

Com o CloudTrail, é possível registrar em log, monitorar continuamente e reter a atividade da conta relacionada às ações executadas na infraestrutura. O histórico de eventos do CloudTrail simplifica a análise de segurança, o rastreamento de alterações de recursos e a solução de problemas.

Monitoramento de aplicativos e recursos

Amazon CloudWatch

Você pode usar o Amazon CloudWatch para coletar e rastrear métricas, coletar e monitorar arquivos de log, definir alarmes e reagir automaticamente a alterações nos serviços e recursos da AWS executados.

Rastreamento distribuído

AWS X-Ray

Obtenha uma visualização completa sobre as solicitações, conforme elas percorrem o aplicativo, além de um mapa dos componentes subjacentes do aplicativo. À medida que um conjunto de microsserviços trabalha para processar uma solicitação, o AWS X-Ray pode fornecer uma visualização centralizada dos logs, o que permite monitorar e resolver problemas de interações complexas.

Usando microsserviços hospedados no Amazon ECS, a **Shippable** conseguiu se dedicar à entrega de recursos aos clientes e acelerou o a implantação desses recursos de uma vez por semana para vários por dia.

MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

MÓDULO 1

DEFININDO MICROSERVICES

DevOps

Gerencie o ciclo de vida do código, da confirmação à execução.

Repositórios de imagens de contêiner

Amazon Elastic Container Registry (Amazon ECR)

Um registro gerenciado de contêineres do Docker que facilita o armazenamento, o gerenciamento e a implantação de imagens de contêineres do Docker. O Amazon ECR é integrado ao Amazon Elastic Container Service (Amazon ECS), o que simplifica o fluxo de trabalho de contêineres, do desenvolvimento até a produção.

Entrega contínua

Ferramentas do desenvolvedor da AWS

As ferramentas do desenvolvedor da AWS são um conjunto de serviços que permite que desenvolvedores e profissionais de operações de TI que trabalham com DevOps possam entregar software com rapidez e segurança. Juntos, esses serviços ajudam a armazenar e controlar com segurança versões de código-fonte de aplicativos, além de criar, testar e implantar automaticamente aplicativos no ambiente local ou na AWS.

A **Gilt** mudou de um datacenter local para a AWS para aproveitar a velocidade e a eficiência de uma infraestrutura de microsserviços baseados na nuvem.

MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

MÓDULO 1

SOA VERSUS MICROSERVICES

Arquitetura Orientada a Serviços (SOA)

Utilizada, geralmente, em grandes corporações, a arquitetura orientada a serviços contém integrações de dados e códigos necessários para executar uma função de negócios completa, em que os serviços podem ser acessados de maneira remota, sendo possível interagir e atualizá-los de forma independente.

Entre os principais benefícios do SOA estão flexibilidade, agilidade e redução de custos na reutilização de serviços.

Um dos componentes mais importantes da arquitetura orientada a serviços é o enterprise service bus (ESB), um barramento de serviços corporativos que disponibiliza com maior facilidade os serviços do sistema para os usuários e outras aplicações, acelerando os processos de integração.

Embora não seja uma arquitetura nova no mercado, a evolução das políticas, regulamentações e modelos de negócios convergiram para que hoje o SOA fosse pensado como um fornecedor de soluções.

Por isso, as organizações governamentais e comerciais têm tirado proveito dessa tecnologia, nos últimos anos, para se tornarem mais inovadores e adaptáveis às mudanças.

Com uma abordagem que facilita a criação de serviços de negócios interoperáveis e flexivelmente acoplados para fácil compartilhamento dentro das empresas e entre elas, o SOA extrai o valor da reutilização e agilidade possibilitadas por ele.

Além disso, os sistemas implementados hoje sobrevivem a seus implementadores originais, na forma de aplicativos empresariais.

MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

MÓDULO 1

SOA VERSUS MICROSERVICES

Benefícios da SOA
De modo geral, entre os principais benefícios da arquitetura orientada a serviços a serem destacados, estão:

- Diminuição do tempo de desenvolvimento, agilizando as entregas;
- Baixo acoplamento entre as partes do sistema, facilitando a manutenção;
- Isolamento da estrutura de um serviço, trazendo flexibilidade durante mudanças;
- Facilidade de agregar novas tecnologias a plataformas, agilizando as integrações de sistema;
- Possibilidade de reutilização de componentes, aumentando o retorno sobre os investimentos (ROI), com o reuso.

Desvantagens da SOA
Como desvantagens, o SOA pode aumentar a complexidade na governança, dada a grande quantidade de serviços que precisa ser gerenciada.

Também é difícil conseguir “responsáveis” para cada serviço, pois muitos serviços são altamente reutilizáveis por diversos sistemas.

A performance também é outro ponto preocupante, pois depende diretamente do servidor onde o serviço está publicado, além da rede.

Nesse modelo, a escalabilidade também precisa ser observada, pois em momentos de pico não se consegue facilmente criar novas instâncias de ESB.

Outra questão do SOA é a dificuldade de montar um modelo de DevOps, com equipes independentes (squads) e com autonomia no desenvolvimento, pois no DevOps, a implantação de um serviço, mesmo com falha, não afeta nenhuma outra parte do sistema, o que não é realidade do SOA, que propõe o máximo reuso de serviços.

MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

MÓDULO 1

SOA VERSUS MICROSERVICES

SOA vs. Microsserviços:
Definir qual arquitetura de software é a melhor solução para o seu negócio vai depender do tipo de problema que você quer resolver.

No SOA, os serviços são independentes da plataforma ou da tecnologia. Eles podem ser desenvolvidos usando qualquer linguagem de programação em qualquer sistema operacional, desde que se comuniquem entre si.

O resultado disso é a integração com outros sistemas, serviços e aplicações. Contudo, toda a arquitetura orientada a serviços é baseada em padrões de comunicação. Então, apesar da interoperabilidade e diversidade de aplicações e serviços dentro de um mesmo produto, eles utilizam a mesma comunicação.

Já o foco da arquitetura dos microsserviços é a independência. Eles não precisam, necessariamente, ser escritos usando a mesma linguagem de programação. Assim, os microsserviços são recomendáveis, principalmente, para projetos de alta complexidade, ajudando a antecipar entregas.

Entretanto, os microsserviços demandam cuidados e esforços adicionais para manter sua gestão, governança e segurança operacional.

Se bem implantados, tanto os microsserviços quanto o SOA trazem benefícios importantes para uma empresa.

Para algumas empresas, o ideal é que eles coexistam, em um modelo de integração de dados híbrido, com cada um carregando suas vantagens e sendo utilizados para projetos em que melhor se adaptam. Nesse caso, a experiência com as tecnologias pode ser a diferença entre as boas e as más decisões para seus projetos.

MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

MÓDULO 1

CONSUMINDO MICROSERVICES VIA API WEBHOOKS

O que é um webhook?

Webhook é uma função de callback baseada em HTTP que viabiliza a comunicação entre duas interfaces de programação de aplicações (APIs). Os webhooks são usados por várias web apps para receber pequenos volumes de dados de outras aplicações e também podem ser usados para acionar workflows de automação em ambientes GitOps.

Webhooks para o desenvolvimento de aplicações

O que é uma API?

Uma API é um conjunto de definições e protocolos usado no desenvolvimento e na integração de aplicações. Às vezes, a comunicação entre APIs é descrita como um contrato entre um provedor e um usuário de informações, estabelecendo o conteúdo exigido pelo consumidor (a chamada) e o conteúdo exigido pelo produtor (a resposta). Essa relação também é descrita como a aplicação cliente que chama a aplicação servidor. No entanto, essas funções podem se inverter, dependendo da aplicação que esteja solicitando os dados.

Em geral, as APIs web usam HTTP para solicitar dados de outras aplicações e definir a estrutura das mensagens de resposta, que normalmente são enviadas em um arquivo XML ou JSON. Tanto XML quanto JSON são formatos de preferência porque apresentam os dados de forma simplificada, o que facilita a manipulação por outras aplicações.

Quando uma API cliente solicita dados de uma API servidor, ela está chamando para verificar a ocorrência de um evento específico, ou seja, se os dados do servidor mudaram e podem ser úteis ao cliente. Nesse processo (conhecido como pesquisa), o cliente envia uma solicitação HTTP em intervalos regulares até a API do servidor enviar os dados relevantes, às vezes chamados de payload.

MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

MÓDULO 1

CONSUMINDO MICROSERVICES VIA API WEBHOOKS

A aplicação cliente não sabe o estado da aplicação servidor, então faz pesquisas com a API do servidor para conseguir uma atualização, repetindo a chamada até a ocorrência do evento específico. No entanto, o servidor só enviará os dados solicitados quando as informações estiverem disponíveis. A aplicação cliente precisa continuar pedindo pela atualização e espera até a ocorrência do evento relevante.

Qual o diferencial dos webhooks?
Para configurar um webhook, o cliente oferece uma URL exclusiva para a API servidor e especifica sobre qual evento ele quer as informações. Quando o webhook estiver configurado, o cliente não precisará mais das pesquisas. O servidor enviará automaticamente o payload relevante para o URL do webhook do cliente quando o evento especificado ocorrer.

Em geral, os webhooks são descritos como APIs reversas ou APIs de push, porque colocam a responsabilidade da comunicação no servidor, e não no cliente. Em vez de o cliente enviar as solicitações HTTP pedindo os dados até receber uma resposta, o servidor envia para o cliente uma solicitação HTTP POST exclusiva quando os dados estiverem disponíveis. Apesar dos apelidos, os webhooks não são APIs. Eles trabalham juntos. Uma aplicação precisa ter uma API para usar um webhook.

O nome webhook é uma combinação simples de web, referente à comunicação baseada em HTTP, e a função de programação hooking, utilizada pelas aplicações para interceptar chamadas ou outros eventos relevantes. Webhooks pegam o evento que ocorre na aplicação servidor e solicitam o envio do payload para o cliente via web. Jeff Lindsay ajudou a disseminar o conceito em 2007, no post "Webhooks para revolucionar a web".

As equipes de TI usam vários métodos para proteger as aplicações que se comunicam por webhooks. A maioria das aplicações habilitadas para webhook adicionam uma chave secreta ao cabeçalho da solicitação de payload para que o cliente possa confirmar a identidade do servidor. Em geral, os webhooks são protegidos por autenticação Mutual Transport Layer Security (mTLS), que verifica o cliente e o servidor antes do envio do payload. As aplicações cliente também costumam usar criptografia SSL no URL do webhook para assegurar a privacidade da transferência de dados.

MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

MÓDULO 1

CONSUMINDO MICROSERVICES VIA API WEBHOOKS

Webhooks:

- **Eliminam a necessidade de pesquisas:** isso economiza recursos da aplicação cliente.
- **São rápidos de configurar:** se uma aplicação dá suporte a webhooks, fica fácil de configurá-los usando a interface de usuário da aplicação servidor. É nesse momento que o cliente insere o URL do webhook da aplicação e define os parâmetros básicos, como o evento relevante.
- **Automatizam a transferência de dados:** o payload é enviado assim que o evento especificado ocorre na aplicação servidor. Essa troca é iniciada pelo evento, então a transferência de dados do servidor para o cliente acontece em tempo real.
- **São ótimos para payloads específicos e lightweight:** os webhooks dependem do servidor para determinar o volume de dados enviados. Cabe ao cliente interpretar e usar o payload de maneira produtiva. Como o cliente não tem controle sobre o tamanho ou o momento da transferência de dados, os webhooks processam pequenos volumes de informações entre dois endpoints, normalmente na forma de notificações.

Webhooks para desenvolvimento de infraestrutura
Em geral, os webhooks são usados para simplificar a comunicação entre duas aplicações, mas também podem ser utilizados para automatizar fluxos de trabalho de infraestrutura como código (IaC) e viabilizar práticas de GitOps.

O que é infraestrutura como código (IaC)?
Infraestrutura como código (IaC) é o gerenciamento e provisionamento de infraestrutura por código, em vez de processos manuais. O controle de versão é uma parte importante da IaC. Os arquivos de configuração devem pertencer à fonte como qualquer outro código-fonte de software. Ao implantar a infraestrutura como código, também é possível separá-la em módulos, que podem ser combinados de diferentes maneiras por meio da automação.

Ao automatizar o provisionamento da infraestrutura com a IaC, os desenvolvedores não precisam provisionar e gerenciar manualmente servidores, sistemas operacionais, armazenamento e outros componentes de infraestrutura sempre que criam ou implantam uma aplicação. A codificação da infraestrutura oferece um template de provisionamento. Também é possível fazer isso manualmente, mas esses processos podem ser automatizados com uma plataforma de estado desejado de nível empresarial, como o Red Hat® Ansible Automation® Platform.

MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

MÓDULO 1

CONSUMINDO MICROSERVICES VIA API WEBHOOKS

IaC declarativa e IaC imperativa
Há dois tipos de abordagem de IaC: declarativa e imperativa.

A abordagem declarativa define o estado desejado do sistema, incluindo os recursos necessários, as propriedades que eles precisam ter e uma ferramenta de IaC para configurá-lo.

Essa abordagem também mantém uma lista do estado atual dos objetos do seu sistema, simplificando o gerenciamento da desativação da infraestrutura.

Por outro lado, a abordagem imperativa define os comandos específicos necessários para alcançar a configuração desejada. Depois, esses comandos precisam ser executados na ordem correta.

Muitas das ferramentas de IaC que usam uma abordagem declarativa provisionam automaticamente a infraestrutura desejada. Se você alterar o estado desejado, uma ferramenta de IaC declarativa aplicará as alterações para você. Uma ferramenta imperativa exige que você saiba como as alterações deverão ser aplicadas.

As ferramentas de IaC, geralmente, funcionam nas duas abordagens, mas costumam preferir uma delas.

Benefícios
O provisionamento de infraestrutura sempre foi um processo manual, caro e demorado. Agora, o gerenciamento de infraestrutura migrou do hardware físico em datacenters (ainda que eles ainda sejam um componente da sua organização) para virtualização, containers e cloud computing.

Com a cloud computing, o número de componentes de infraestrutura aumentou, mais aplicações são colocadas em produção diariamente e as infraestruturas precisam ser flexíveis para as frequentes alterações, escalas e desativações. Hoje em dia, sem a implementação de uma prática de IaC, fica cada vez mais difícil gerenciar a escala da infraestrutura.

A IaC pode ajudar sua organização a gerenciar as necessidades de infraestrutura de TI, melhorando a consistência e reduzindo erros e a necessidade de configuração manual.

MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

MÓDULO 1

CONSUMINDO MICROSERVICES VIA API WEBHOOKS

O que é GitOps?
Muitas vezes considerado uma evolução da IaC, o GitOps é uma abordagem estratégica para gerenciar configurações de infraestruturas e aplicações usando o Git, um sistema de controle de versões open source. Seguindo as práticas do GitOps, os desenvolvedores utilizam o Git como a única fonte de verdade para infraestrutura e aplicações declarativas e usam as pull requests do Git para gerenciar automaticamente o provisionamento e a implantação da infraestrutura. O repositório Git contém todo o estado do sistema, de modo que o registro de mudanças fique visível e possa ser auditado.

Como os webhooks funcionam?
Os webhooks reduzem o número de etapas necessárias para implementar e gerenciar pipelines de implantação centrados no Git e podem ser usados para automatizar o lançamento de fluxos de trabalho inteiros da IaC. Um webhook funciona da mesma maneira em um ambiente de GitOps e entre duas aplicações. Quando acionado por um evento específico, uma API envia um payload para outra. A diferença está no tipo de evento que aciona o webhook e no que o destinatário faz com o payload.

Nesse contexto, o repositório Git exerce a função de aplicação servidor enquanto a plataforma de estado desejado, que gerencia o estado da infraestrutura, exerce a função de aplicação cliente. Um webhook pode estar configurado para acionar a comunicação sempre que o repositório for alterado. Por exemplo, se uma parte do código for atualizada e enviada para o repositório Git, esse evento acionará o webhook. Então, o repositório envia automaticamente o payload para o endereço do webhook da plataforma de estado desejado, informando sobre a alteração no código.

Esse uso dos webhooks permite que as plataformas de estado desejado fiquem sempre atualizadas sobre mudanças na infraestrutura sem precisar monitorar ativamente os repositórios. Se a plataforma de estado desejado também tiver suporte para automação, os webhooks podem ser usados para acionar fluxos de trabalho de IaC. Por exemplo, com uma plataforma de estado desejado de nível empresarial, como o Red Hat Ansible Automation Platform, um administrador de sistemas pode usar os webhooks para implantar automaticamente as alterações mais recentes nos hosts gerenciados.

MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

MÓDULO 1

MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

MÓDULO 2

Arquitetura REST

REST, ou Representational State Transfer, é um estilo arquitetônico aplicado para fornecer padrões entre sistemas de computador na web, facilitando a comunicação entre eles. Os sistemas em conformidade com REST, muitas vezes conhecidos como sistemas RESTful, são reconhecidos pelo jeito como separam as preocupações do cliente e do servidor.

No estilo REST, a implementação do cliente e do servidor pode ser feita de forma independente, sem que cada um conheça o outro. Isso significa que o código do lado do cliente pode ser alterado a qualquer momento, sem afetar a operação do servidor, e o contrário também é válido.

Ao utilizar uma interface REST, clientes diferentes atingem os mesmos pontos finais REST, executam ações iguais e recebem as mesmas respostas. Como os sistemas REST interagem por meio de operações padrão sobre recursos, eles não dependem da implementação de interfaces.

Essas restrições ajudam as aplicações RESTful a alcançar maior nível de confiabilidade, além de possibilitar um desempenho rápido e escalável. Todos os componentes podem ser gerenciados, atualizados e reutilizados sem afetar o sistema como um todo, mesmo durante a operação.

Quais os conceitos aplicados ao REST?

Client-server
A separação das preocupações é o princípio por trás das restrições relacionadas à relação cliente-servidor. Desde que cada parte saiba o formato das mensagens a serem enviadas à outra, elas podem ser mantidas modulares e separadas.

Separando as preocupações da interface do usuário daquelas do armazenamento de dados, é possível otimizar a flexibilidade da interface em todas as plataformas e melhorar a escalabilidade, tornando mais simples os elementos do servidor. Além disso, a separação permite a cada elemento a capacidade de evoluir de forma independente.

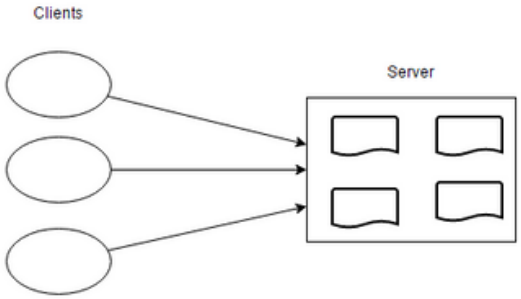
MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

MÓDULO 2

Arquitetura REST

Stateless
Os sistemas que seguem o paradigma REST são stateless, o que pode ser traduzido livremente como “sem estado”. Isso significa que o servidor não precisa saber nada sobre o estado em que o cliente se encontra e vice-versa.

Dessa forma, tanto o servidor quanto o cliente podem compreender qualquer mensagem recebida, mesmo sem ver as anteriores. Isso acontece porque o sistema não utiliza comandos, mas os substantivos da Web, que descrevem qualquer arquivo que você precise armazenar ou utilizar.



Cacheable
As restrições de cache requerem que as informações contidas em uma resposta a uma solicitação sejam, diretamente ou não, definidas como cacheáveis ou não cacheáveis. Caso uma resposta for armazenável em cache, então, é dado ao cliente o direito de utilizar novamente esses dados para atividades similares no futuro.

A vantagem de acrescentar restrições de cache é que elas têm o potencial de eliminar parcial ou completamente algumas interações, melhorando a eficiência, a escalabilidade e o desempenho percebido pelo usuário. Isso torna a experiência muito mais fluida e eficiente.

Uniform interface
O principal elemento que distingue o REST de outras opções com base na rede é o seu foco em uma interface uniforme entre os componentes. O princípio de generalidade, característica da engenharia de software, quando aplicado à interface de componentes, simplifica a arquitetura do sistema e torna as interações mais visíveis.

MÓDULO 2

Arquitetura REST

A fim de obter uniformidade na interface, são necessárias múltiplas restrições para orientar o comportamento dos componentes. REST é definido por quatro fatores:

- identificação de recursos;
- gerenciamento de recursos por meio de representações;
- mensagens auto-descritivas;
- hipermídia.

Layered system
Para melhorar ainda mais o desempenho para as atuais e crescentes exigências da Internet, é possível acrescentar restrições de sistema por camadas. Assim, a arquitetura é constituída por camadas que seguem uma ordem hierárquica. Isso restringe o comportamento dos elementos, de modo que cada um deles só possa "enxergar" a camada com que estão interagindo no momento.

Como utilizar o REST?
Como você viu, os clientes que lidam com a arquitetura REST geram requerimentos para reaver ou alterar recursos. Já os servidores assumem o papel de responder tais demandas. Uma solicitação, geralmente, consiste em:

- verbo HTTP, que define o tipo de operação a realizar;
- header, que permite ao cliente transmitir informações sobre o pedido;
- caminho para um recurso;
- corpo de mensagem opcional contendo dados.

Sobre o primeiro item, o verbo HTTP, é fundamental conhecer quatro entradas básicas:

- GET — recuperar um recurso específico (por id) ou uma coleção de recursos;
- POST — criar um novo recurso;
- PUT — atualizar um recurso específico (por id);
- DELETE — remover um recurso específico por id.

No cabeçalho da solicitação, o cliente envia o tipo de conteúdo que é capaz de receber do servidor. Esse é o campo chamado ACCEPT, e garante que o servidor não envie dados que não possam ser compreendidos ou processados pelo cliente.

MÓDULO 2

Arquitetura REST

Os pedidos devem conter um caminho para o recurso no qual a operação deve ser realizada. Nas APIs RESTful, os caminhos devem ser projetados para ajudar o cliente a saber o que está acontecendo.

Convencionalmente, a primeira parte do caminho deve ser a forma plural do recurso. Isso mantém os caminhos agrupados simples de ler e fáceis de entender. Os caminhos devem conter as informações necessárias para localizar um recurso com o grau de especificidade necessário.

Quais os benefícios de utilizar o REST?
Um benefício principal do uso desse estilo, tanto do ponto de vista do cliente quanto do servidor, é que as interações baseadas em REST acontecem usando estruturas que são familiares a qualquer um que esteja acostumado a usar o HTTP da Internet. Um exemplo desse arranjo são as interações baseadas em REST — todas comunicam seu status usando códigos de status HTTP padrão.

Portanto, um erro 404 significa que um recurso solicitado não foi encontrado. Um código 401 quer dizer que a solicitação não foi autorizada. Em um código 200, tudo está ok e, em um 500, houve um erro de aplicação irrecuperável no servidor.

Da mesma forma, detalhes como a criptografia e a integridade do transporte de dados são resolvidos não adicionando novas estruturas ou tecnologias, mas confiando nos conhecidos sistemas de criptografia Secure Sockets Layer (SSL) e Transport Layer Security (TLS). Assim, toda a arquitetura REST é construída sobre conceitos com os quais a maioria dos desenvolvedores já está familiarizada.

REST é, também, um estilo arquitetônico independente do idioma. Aplicações baseadas nesse estilo podem ser escritas usando qualquer linguagem, seja Java, seja Kotlin, .NET, AngularJS ou JavaScript.

Desde que uma linguagem de programação possa fazer solicitações baseadas na web usando HTTP, é possível que essa linguagem seja usada para ativar uma API ou serviço web RESTful.

Arquitetura REST

Da mesma forma, serviços Web RESTful podem ser escritos usando qualquer linguagem. Assim, os desenvolvedores encarregados de implementar tais serviços podem escolher tecnologias que funcionem melhor para cada situação.

Outro benefício do uso de REST é sua abrangência. No lado do servidor, há uma variedade de frameworks baseados no estilo para ajudar os desenvolvedores a criar serviços web RESTful, incluindo RESTlet e Apache CXF.

Do lado do cliente, todos os novos frameworks JavaScript, tais como JQuery, Node.js, Angular e EmberJS, têm bibliotecas padrão incorporadas em suas APIs, que fazem da utilização de serviços web RESTful e do consumo dos dados baseados em XML ou JSON um esforço relativamente simples.

A utilização do estilo de arquitetura REST é fundamental para o estabelecimento de um ambiente favorável à realização das ações necessárias para manter um website em bom funcionamento. Em relação a outras soluções no mercado, o REST se destaca pela facilidade de uso, o que o torna uma opção muito relevante para todos os níveis de conhecimento.

MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

MÓDULO 2

CONTAINERS E MICROSERVICES: FLUXO CI/CD

O que é CI/CD?

Quando falamos sobre CI/CD, estamos realmente falando de vários processos relacionados: integração contínua, entrega contínua e implantação contínua.

- Integração contínua. As alterações de código são frequentemente mescladas no branch principal. Os processos automatizados de build e teste garantem que o código no branch principal seja sempre de qualidade de produção.

- Entrega contínua. Todas as alterações de código que passam pelo processo de CI são publicadas automaticamente em um ambiente semelhante à produção. A implantação no ambiente de produção dinâmico pode exigir aprovação manual, mas caso contrário, é automatizada. A meta é que o seu código esteja sempre pronto para implantação na produção.

- Implantação contínua. As alterações de código que passam nas duas etapas anteriores são implantadas automaticamente na produção.

Aqui estão algumas metas de um processo de CI/CD robusto para uma arquitetura de microsserviços:

- Cada equipe pode criar e implantar os serviços que ela possui independentemente, sem afetar ou interromper outras equipes.

- Antes de uma nova versão de um serviço ser implantada na produção, ela é implantada em ambientes de desenvolvimento/teste/garantia de qualidade para validação. Há restrições de qualidade impostas em cada estágio.

- Uma nova versão de um serviço pode ser implantada lado a lado com a versão anterior.

- Há políticas de controle de acesso suficientes em vigor.

- Para cargas de trabalho conteinerizadas, você pode confiar nas imagens de contêiner que estão implantadas na produção.

MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

MÓDULO 2

CONTAINERS E MICROSERVICES: FLUXO CI/CD

O que são containers?
Uma forma de conduzir uma cultura DevOps deve ter em mente que a descentralização de uma arquitetura é a chave para um processo mais eficiente. Por isso, o uso de Containers é tão importante.

Trata-se de uma tecnologia de virtualização que isola e empacota códigos (e suas dependências) para facilitar a execução de uma aplicação. Ou seja, dentro de uma lógica de microsserviços, os containers oferecem essa possibilidade de tornar serviços ou funcionalidades independentes, funcionais em qualquer plataforma, sem um setup de máquina virtual dedicada.

É a tecnologia que fundamenta os microsserviços.

E para quem busca se aprofundar no assunto e implementar uma arquitetura de microsserviços em seu modelo de negócios, vale conhecer sobre as tecnologias complementares.

Docker: Uma tecnologia própria para containers

O Docker é uma tecnologia complementar e específica para containers. Além de possibilitar a gestão de toda infraestrutura de uma aplicação, o Docker reproduz o processo de empacotamento dentro de um container.

De forma prática, ele possibilita a portabilidade de ambientes inteiros para outros hosts de forma ágil e simplificada, o que reduz o tempo de deploy por exemplo. Afinal, não há necessidades de ajustes posteriores para seu pleno funcionamento.

A tecnologia cria imagens de arquivos e utiliza o back-end default LXC, que possibilita a limitação de recursos por cada container.

Kubernetes: Orquestração de containers

O Kubernetes é um projeto open-source projetado especialmente para os microsserviços. Ele potencializa as capacidades do Docker, sendo uma ferramenta para publicação gestão eficiente não de um container, mas vários de forma orquestrada.

MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

MÓDULO 2

CONTAINERS E MICROSERVICES: FLUXO CI/CD

Com o Kubernetes, você automatiza as atividades relacionadas aos containers de ponta a ponta: da implantação até a sua gestão. Tamanha gama de recursos e a fácil adaptabilidade torna o Kubernetes uma das soluções mais utilizadas nas empresas, especialmente pelo seu potencial de escalabilidade, automação e resiliência.

Como usar os Containers e Microsserviços

A busca pela modernização dos modelos de negócio impactou profundamente o dia a dia de desenvolvimento de empresas do ramo tecnológico. Em especial, hoje em dia, a exigência é por aplicações eficazes e ágeis — e, principalmente, com continuidade.

Seja na integração como na entrega, a continuidade deve ser sempre o objetivo ao desenvolver uma aplicação. Por isso, métodos antigos como o da arquitetura monolítica já podem ser considerados passado — e containers e microsserviços, o futuro.

A descentralização tira o peso de um único sistema e o compartilha com vários outros, específicos e prontos para cada serviço e funcionalidade diferente.

Com containers e microsserviços como base do seu desenvolvimento, você garante mais velocidade de deploy e uma maior abrangência (e eficácia) na automação de processos, bem como seu monitoramento.

Com o DevOps e suas ferramentas, o cenário se mostra ainda mais promissor. Em um ambiente, você assegura o nivelamento do controle de qualidade (QA), Homologação e Produção.

Assim, com responsabilidades e objetivos compartilhados, é mais fácil se adequar às exigências de um mercado volátil e desafiado, conquistando melhores resultados e um maior diferencial competitivo para o futuro!

MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

MÓDULO 2

MICROSERVICES E DATABASES: QUERIES E CONSISTENCY

Como Gerenciar e Integrar Dados em Microsserviços?
A tarefa de integrar dados em microsserviços é uma das principais preocupações de quem opta por adotar essa arquitetura de desenvolvimento. Em microsserviços, os bancos de dados devem funcionar de forma independente, mas consistente.

Por isso, gerenciar dados em microsserviços de forma eficiente é a única maneira de garantir a consistência dos dados e assegurar que todos os serviços funcionem em sua eficiência máxima. Entenda mais sobre o assunto!

Os desafios de integrar dados em microsserviços
O armazenamento separado de dados para cada serviço é uma premissa dos microsserviços e faz parte de um dos principais objetivos ligados à adoção desse tipo de arquitetura: a garantia de independência dos serviços.

Se diversos serviços usarem o mesmo banco de dados, podem surgir problemas como a interferência de desempenho entre os serviços, falta de escalabilidade e a impossibilidade de usar formato de dados que são mais adequados para cada serviço.

Ou seja, o compartilhamento de banco de dados geralmente é descartado quando os microsserviços são usados.

No entanto, essa abordagem leva a desafios como a redundância de dados, que podem ficar duplicados e levar a problemas de consistência.

Se um dado é atualizado, essa atualização será replicada para todos os serviços?

Nesse contexto, o principal objetivo de gerenciar dados em microsserviços é garantir que os bancos de dados de cada microsserviço continuem funcionando de forma independente um do outro, mas que ainda exista consistência e integração entre eles.

Como gerenciar dados em microsserviços?

Ao migrar da arquitetura monolítica para os microsserviços, é preciso ter uma estratégia para extrair os dados do bloco único e levá-los para cada microsserviço.

MICROSERVICES E DATABASES: QUERIES E CONSISTENCY

Geralmente, isso é feito seguindo quatro passos:

1. Criação do serviço e delimitação dos dados a serem utilizados por ele;
2. Desacoplamento das aplicações da arquitetura monolítica migrando-as para o microsserviço recém-criado;
3. Movimentação dos dados do banco compartilhado para um novo banco de dados privado;
4. Repetição desse processo até que todas as aplicações sejam levadas do monolítico para os microsserviços.

Uma vez separados, é preciso adotar medidas para integrar dados e garantir a persistência e a consistência deles. A missão de integrar dados pode ser realizada por meio de um banco de dados relacional.

Para isso, podem ser utilizados diferentes modelos de consistência de dados.

Consistência Forte versus Consistência Eventual

Consistência forte significa que uma variável que aparece em diferentes partes do sistema só é atualizada quando todos os nós concordam com o novo valor.

Se a variável é alterada em apenas um nó, nenhuma mudança é visível para o cliente. Esse tipo de semântica é usada, por exemplo, em transações.

A consistência eventual permite mais flexibilidade aos microsserviços.

Pode ocorrer, em um sistema distribuído, que um nó seja atualizado e o novo valor exibido, mesmo que os demais nós ainda estejam exibindo um valor desatualizado. Ou seja, a consistência eventual permite que exista um delay entre os nós do sistema. Dessa forma, os dados exibidos não estão incorretos, mas desatualizados.

MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

MÓDULO 2

MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

MÓDULO 2

MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

4 MÓDULOS | 16 HORAS

LIVE

MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

MÓDULO 3

DIVIDINDO O MONOLITICO

DIVIDIR UM APLICATIVO MONOLÍTICO EM MICROSSERVIÇOS
COM O AMAZON ELASTIC CONTAINER SERVICE, DOCKER E AMAZON EC2

- Visão geral

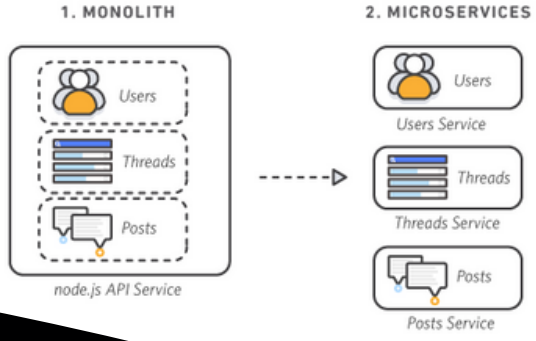
Neste tutorial, você implantará um aplicativo monolítico node.js em um contêiner do Docker, e depois desacoplará o aplicativo em microsserviços sem qualquer tempo de inatividade. O aplicativo node.js hospeda um fórum simples com discussões e mensagens entre os usuários.

- Por que isso é importante

As arquiteturas monolíticas tradicionais são difíceis de escalar. À medida que a base de códigos de um aplicativo cresce, fica cada vez mais complexo mantê-la e atualizá-la. Apresentar novos recursos, línguas, estruturas e tecnologias fica mais difícil, limitando inovações e novas ideias.
Com uma arquitetura de microsserviços, cada componente do aplicativo roda sozinho e se comunica com outros serviços por meio de uma API bem definida. Os microsserviços são projetados de acordo com as capacidades dos negócios, e cada serviço realiza uma única função. Os microsserviços podem ser escritos com o uso de estruturas e linguagens de programação diferentes, e você pode implantá-los separadamente, como um único serviço ou como um grupo de serviços.

- Arquitetura do aplicativo

Durante este tutorial, nós mostraremos como você pode executar um aplicativo monolítico simples em um contêiner do Docker, implantar o mesmo aplicativo como microsserviços, e também migrar o tráfego para os microsserviços sem tempo de inatividade. Assim que terminar, você poderá usar este tutorial e o código que há nele como modelo para construir e implantar seus próprios microsserviços em contêineres na AWS.



MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

MÓDULO 3

DIVIDINDO O MONOLITICO

Arquitetura monolítica

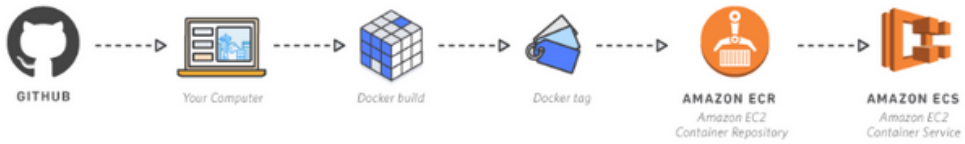
O aplicativo node.js todo roda em um contêiner como um único serviço, e cada contêiner apresenta os mesmos recursos que todos os outros contêineres. Caso algum recurso do aplicativo apresente um pico de demanda, a arquitetura toda deve ser escalada.

Arquitetura de microsserviços

Cada recurso de um aplicativo node.js roda como um serviço separado dentro do seu próprio contêiner. Os serviços podem escalar e ser atualizados separadamente dos outros.

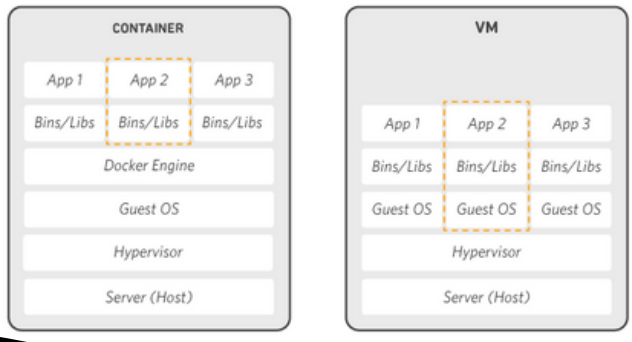
Módulo um – Conteinerizar o monólito

Neste módulo, você compilará a imagem do contêiner para o aplicativo monolítico node.js e o enviará para o Amazon Elastic Container Registry. Comece a criar



O que é um contêiner?

Os contêineres permitem empacotar facilmente o código, as configurações e as dependências de um aplicativo em componentes básicos fáceis de usar que oferecem consistência ambiental, eficiência operacional, produtividade de desenvolvedores e controle de versões. Os contêineres podem ajudar a garantir rapidez, confiabilidade e consistência de implantação de aplicativos, independentemente do ambiente de implantação.



MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

MÓDULO 3

DIVIDINDO O MONOLITICO

Por que usar contêineres?

Velocidade
A execução de um contêiner com uma nova versão do código pode ser feita sem sobrecarga de implantação significativa. A velocidade operacional é melhorada, pois o código criado em um contêiner na máquina local do desenvolvedor pode ser facilmente movido para um servidor de testes simplesmente movendo o contêiner. No momento da compilação, este contêiner pode ser vinculado a outros contêineres necessários para executar a pilha de aplicativos.

Controle de dependência e pipeline melhorado
Uma imagem de contêiner do Docker é a captura de um momento específico do código e das dependências de um aplicativo. Isso possibilita que uma organização de engenharia crie um pipeline padrão para o ciclo de vida do aplicativo. Por exemplo:

1. Os desenvolvedores compilam e executam o contêiner localmente.
2. O servidor de integração contínua executa o mesmo contêiner e realiza testes de integração com ele para garantir que as expectativas sejam atendidas.
3. O mesmo contêiner é enviado para um ambiente de preparação, onde seu comportamento de tempo de execução pode ser verificado com o uso de testes de carga ou controle de qualidade manual.
4. O mesmo contêiner é enviado para produção.

Ser capaz de compilar, testar, enviar e executar exatamente o mesmo contêiner em todos os estágios da integração e da implantação do pipeline torna consideravelmente mais fácil o fornecimento de um aplicativo confiável e de alta qualidade.

Densidade e eficiência de recurso
Os contêineres facilitam a eficiência de recursos aprimorados ao permitir que múltiplos processos heterogêneos sejam executados em um único sistema. A eficiência de recurso é um resultado natural de técnicas de isolamento e alocação que os contêineres usam. Os contêineres podem ser limitados para consumir quantidades determinadas da CPU e da memória de um host. Ao perceber quais recursos um contêiner são necessários e quais estão disponíveis no servidor host subjacente, você pode dimensionar os recursos de computação usados com hosts menores ou aumentar a densidade de processos em execução em um único host grande, aumentando assim a disponibilidade e otimizando o consumo de recursos.

MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

MÓDULO 3

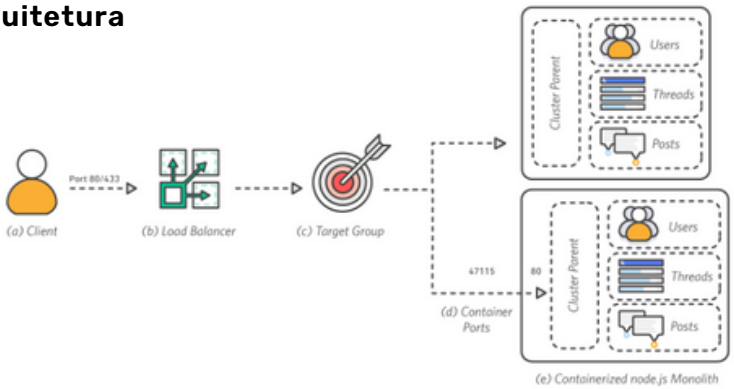
DIVIDINDO O MONOLITICO

Flexibilidade
A flexibilidade de contêineres do Docker tem como base sua portabilidade, facilidade de implantação e seu tamanho pequeno. Em contraste com a instalação e a configuração exigidas em uma VM, os serviços de pacotes dentro dos contêineres permitem que eles sejam facilmente movidos entre hosts, isolados da falha de outros serviços próximos e protegidos contra correções errôneas ou atualizações de software no sistema do host.

Módulo dois – Implantar o monólito

Neste módulo, você usará o Amazon Elastic Container Service (Amazon ECS) para instanciar um cluster gerenciado de instâncias de computação EC2 e implantar sua imagem como um contêiner em execução no cluster. Comece a criar

Visão geral da arquitetura



a. Cliente: O cliente faz uma solicitação pela porta 80 para o load balancer.

b. Load Balancer: O load balancer distribui solicitações em todas as portas disponíveis.

c. Grupos de destino: As instâncias são registradas no grupo de destino do aplicativo.

d. Portas de contêiner: Cada contêiner executa um único processo de aplicativo que liga o cluster node.js pai à porta 80 dentro de seu namespace.

e. Monólito node.js em contêineres: O cluster node.js pai é responsável por distribuir o tráfego para os trabalhadores no aplicativo monolítico. Essa arquitetura é em contêiner, mas ainda monolítica, porque cada contêiner possui todos os mesmos recursos do restante dos contêineres.

MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

MÓDULO 3

DIVIDINDO O MONOLITICO

O que é o Amazon Elastic Container Service?
O Amazon Elastic Container Service (Amazon ECS) é um serviço de gerenciamento de contêineres de alta escalabilidade e performance que aceita contêineres do Docker e permite que você execute facilmente aplicativos em um cluster gerenciado de instâncias do Amazon EC2. Com chamadas de API simples, você pode executar e interromper aplicações habilitadas para o Docker, consultar o estado completo do seu cluster e acessar muitos recursos conhecidos, como security groups, Elastic Load Balancing, volumes do EBS e funções do IAM.

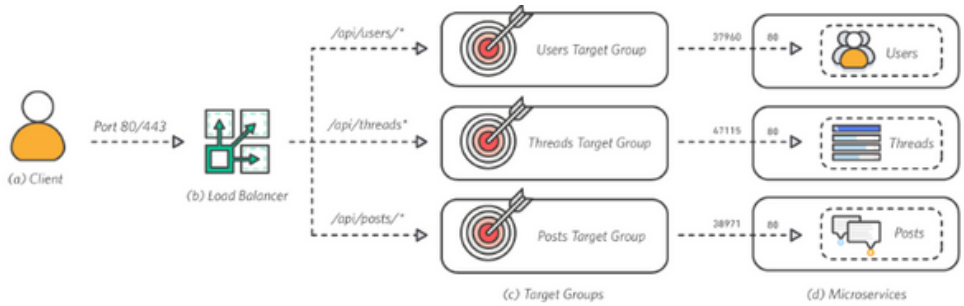
Você pode usar o Amazon ECS para programar a colocação de contêineres no cluster com base em suas necessidades de recursos e requisitos de disponibilidade. Você também pode integrar agendadores próprios ou de terceiros para atender a requisitos específicos dos negócios ou da aplicação.

Não há cobrança adicional para o Amazon ECS. Você paga pelos recursos da AWS (por exemplo, instâncias do EC2 ou volumes do EBS) criados para armazenar e executar o aplicativo.

Módulo três – Dividir o monólito

Neste módulo, você dividirá o aplicativo node.js em vários serviços interconectados e enviará cada imagem do serviço a um repositório do Amazon Elastic Container Registry (Amazon ECR). Comece a criar

Visão geral da arquitetura
A arquitetura final do aplicativo usa o Amazon Elastic Container Service (Amazon ECS) e o Application Load Balancer (ALB).



MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

MÓDULO 3

DIVIDINDO O MONOLITICO

a. Cliente: O cliente faz solicitações de tráfego pela porta 80.

b. Load Balancer: O ALB direciona o tráfego externo para o serviço correto. O ALB inspeciona a solicitação do cliente e usa as regras de roteamento para direcionar a solicitação a uma instância e uma porta para o grupo de destino que corresponde à regra.

c. Grupos de destino: Cada serviço tem um grupo de destino que monitora as instâncias e portas de cada contêiner em execução para esse serviço.

d. Microsserviços: O Amazon ECS implanta cada serviço em um contêiner em um cluster inteiro do EC2. Cada contêiner lida apenas com um único recurso.

Por que microsserviços?

Isolamento de falhas
Mesmo as melhores organizações de engenharia podem ter e têm falhas graves na produção. Além de seguir todas as melhores práticas padrão para lidar com falhas harmoniosamente, uma abordagem que pode limitar o impacto de tais falhas é a criação de microsserviços. Uma boa arquitetura de microsserviço significa que se uma pequena parte do seu serviço falhar, apenas essa parte do seu serviço deixará de funcionar. O resto do serviço poderá continuar a funcionar adequadamente.

Isolamento para segurança
Em um aplicativo monolítico, se um recurso tiver uma violação de segurança, por exemplo, uma vulnerabilidade que permita a execução remota de código, você deverá supor que um invasor pode ter obtido acesso a todos os outros recursos do sistema também. Isso pode ser perigoso se, por exemplo, seu recurso de upload de avatar tiver um problema de segurança que comprometa seu banco de dados com senhas de usuários. Separar recursos em microsserviços usando o Amazon ECS permite que você proteja o acesso a recursos da AWS ao dar a cada serviço sua própria função de AWS Identity and Access Management (IAM). Quando as melhores práticas de microsserviço são seguidas, o resultado é que se um invasor comprometer um serviço, ele só terá acesso aos recursos desse serviço e não poderá acessar horizontalmente outros recursos de outros serviços sem invadir esses serviços também.

MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

MÓDULO 3

DIVIDINDO O MONOLITICO

Escalabilidade independente
Quando os recursos são divididos em microsserviços, o volume de infraestrutura e o número de instâncias usados por cada classe de microsserviço pode ser escalado para mais ou para menos de modo independente. Isso facilita a medição dos custos de um recurso específico e identifica recursos que podem precisar prioritariamente de otimização. Se um recurso específico estiver enfrentando problemas com sua necessidade de reservas, outros recursos não serão impactados e poderão manter sua performance confiável.

Velocidade de desenvolvimento
Microsserviços reduzem os riscos no desenvolvimento, o que pode favorecer uma equipe a criar mais rapidamente. Em um monólito, adicionar um novo recurso pode afetar todos os outros recursos contidos nesse monólito. Os desenvolvedores precisam considerar com cuidado o impacto de qualquer código que eles adicionam e garantir que não corrompam nada. Por outro lado, uma arquitetura adequada de microsserviço tem um novo código para um novo recurso que entra em um novo serviço. Os desenvolvedores podem ter certeza de que qualquer código que eles criem não poderá de fato afetar o código existente, a menos que eles explicitamente criem uma conexão entre dois microsserviços.

Módulo quatro – Implantar microsserviços

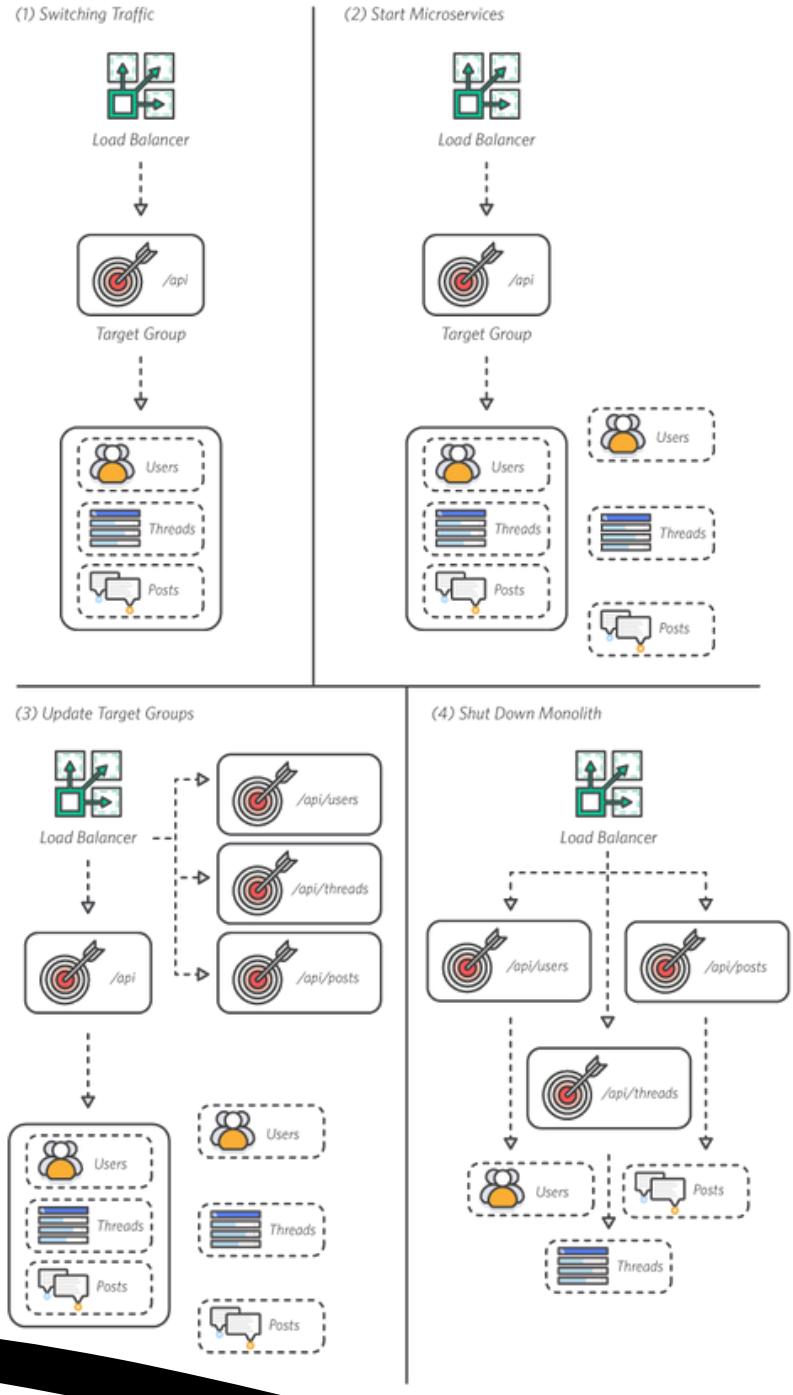
Neste módulo, você implantará o aplicativo node.js como um conjunto de serviços interconectados por trás de um Application Load Balancer (ALB). Depois, você usará o ALB para mudar facilmente o tráfego do monólito para os microsserviços. Comece a criar

Visão geral da arquitetura
Este é o processo que você seguirá para implantar os microsserviços e mudar com segurança o tráfego do aplicativo, deixando para trás o monólito.

MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

MÓDULO 3

DIVIDINDO O MONOLITICO



MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

MÓDULO 3

DIVIDINDO O MONOLITICO

1. **Mude o tráfego:** Esta é a configuração inicial. O aplicativo monolítico node.js em execução em um contêiner no Amazon ECS.
2. **Inicie microsserviços:** Usando as três imagens de contêineres que você criou e enviou para o Amazon ECR no módulo anterior, você iniciará três microsserviços no cluster existente do Amazon ECS.
3. **Configure os grupos de destino:** Como no Módulo 2, você adicionará um grupo de destino para cada serviço e atualizará as regras do ALB para se conectar aos novos microsserviços.
4. **Desative o monólito:** Ao alterar uma regra no ALB, você iniciará o roteamento do tráfego para os microsserviços em execução. Após a verificação do redirecionamento do tráfego, desative o monólito.

Módulo cinco – Limpar

Neste módulo, você encerrará os recursos que criou durante este tutorial. Você interromperá os serviços em execução no Amazon ECS, excluirá o ALB e excluirá a pilha do AWS CloudFormation para encerrar o cluster ECS, incluindo todas as instâncias do EC2 subjacentes.

A limpeza não é obrigatória, mas ajudará a evitar cobranças contínuas para manter esses serviços em execução. Comece a limpeza

MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

MÓDULO 3

MONITORAMENTO DE MICROSERVICES

O monitoramento de microsserviços é um requisito fundamental quando o assunto é arquitetura de microsserviços. Conforme já abordamos em outros artigos aqui no blog, eles proporcionam uma nova maneira de construir, entregar e gerenciar aplicativos corporativos. No entanto, o monitoramento do seu sistema.

Isso porque à medida que as empresas adotam microsserviços, é preciso repensar suas práticas de observabilidade. Portanto, neste artigo traremos alguns dos princípios do monitoramento de microsserviços para te ajudar a garantir um bom desempenho. Acompanhe.

Por que fazer o monitoramento de microsserviços

A princípio, os sistemas de monitoramento de microsserviços são essenciais em um sistema de alta disponibilidade. Afinal, eles identificam e reportam a ocorrência de erros e falhas. Portanto, listamos os princípios que vão tornar o monitoramento de microsserviços mais inteligente e eficiente.

1. **Monitore os containers e tudo o que é executado dentro deles**

Como blocos de construção de microsserviços, os containers são caixas pretas que abrangem o laptop do desenvolvedor até a nuvem. Mas é difícil executar funções básicas como, por exemplo, monitorar ou solucionar problemas de um serviço. Portanto, é necessário saber o que está sendo executado no contêiner. É preciso também saber como o aplicativo e o código estão sendo executados e se estão gerando métricas personalizadas importantes.

Outro problema é o crescimento da organização. Ao executar milhares de hosts com dezenas de milhares de containers, as implantações podem ficar muito caras. Para um monitoramento de microsserviços correto, há algumas opções.
Você pode instrumentar o código diretamente, executar containers secundários (sidecars) aproveitar a instrumentação universal no nível do kernel para ver todos os aplicativos e atividades do contêiner. É importante observar que cada abordagem tem vantagens e desvantagens. Logo, você precisará revisar qual delas atende aos seus objetivos mais importantes.

MÓDULO 3

MONITORAMENTO DE MICROSERVICES

2. Utilize sistemas de orquestração

No monitoramento de microsserviços, a orquestração automatiza a implantação, o gerenciamento, a escala e a rede dos containers. Ela é ideal para as empresas que precisam implantar e gerenciar centenas de milhares de hosts.

Para se ter ideia, um dos processos mais críticos que você pode executar é rastrear informações agregadas de todos os containers associados a uma função ou serviço.

Se você faz parte de uma equipe de desenvolvimento, pode usar um sistema de orquestração como Kubernetes, Mesosphere DC / OS ou Docker Swarm para orquestrar seus microsserviços. Entretanto, se você trabalha numa equipe devops, recomendamos a redefinição dos alertas do sistema para chegar o mais perto possível do monitoramento da experiência do serviço.

3. Prepare-se para a expansão dos serviços

Todo ambiente nativo de contêiner muda de maneira dinâmica. Isso expõe as vulnerabilidades de qualquer sistema de monitoramento. Ajustar as métricas pode funcionar bem para uma média de 30 containers. Contudo, o monitoramento de microsserviços deve ser capaz de expandir e contrair junto com os serviços, sem ação humana numa espécie de descoberta de serviço (service discovery).

Logo, caso você defina de forma manual, em qual serviço um contêiner é incluído para monitoramento, pode ser que você perca novos containers criados durante o dia pelo Kubernetes ou Mesos.

4. Monitore APIs

As APIs são os únicos elementos de um serviço aparente para outras equipes e o seu monitoramento deve ir além das verificações binárias up-down padrão.

Isso significa que é importante entender os endpoints usados com mais frequência. Descobrir os terminais de serviços mais lentos também mostra áreas que precisam de otimização.
Ser capaz de rastrear chamadas de serviço através do sistema ainda ajudará a empresa a entender a experiência do usuário. Esse aspecto do monitoramento de API também dividirá as informações em visualizações baseadas em infraestrutura e aplicativos do ambiente de microsserviços.

MÓDULO 3

MONITORAMENTO DE MICROSERVICES

5. Mapeie o monitoramento para a estrutura da sua empresa

Sobre o monitoramento de microsserviços, é fundamental considerar os aspectos pessoais do monitoramento de software. Isso significa uma mudança abrangente na cultura da empresa.

Ou seja, equipes menores e com muita autonomia, mas focadas em objetivos estratégicos. Primeiramente, cada equipe mantém mais controle do que nunca sobre os idiomas usados, resolução de bugs ou responsabilidades operacionais.

Logo depois, você pode criar uma plataforma de monitoramento que permite que cada equipe de microsserviço isole seus alertas, métricas e painéis, ao mesmo tempo em que fornece às operações uma visão global do sistema.

Principais ferramentas

No monitoramento de microsserviços, o logging ajuda a fornecer dados que permitem a identificação de falhas nos serviços e na comunicação entre eles, mesmo que seja um problema de negócio. Veja agora as principais ferramentas em log e monitoramento:

• FluentD: é um coletor de dados de código aberto que possibilita unificar a coleta de dados e seu consumo para melhor entendimento dos dados.

• Prometheus: uma aplicação para monitoramento de eventos e alertas. Ele captura métricas em tempo real em um banco de dados temporal. No entanto o Prometheus não conta com uma ferramenta de exibição dos seus dados.

• Grafana: é uma ferramenta que gera gráficos. Geralmente é utilizado juntamente com o Prometheus, fornecendo um dashboard para as métricas colhidas por ele.

• Dynatrace: funciona como ferramenta de monitoramento de aplicações bemcomo ferramenta de gerenciamento de aplicações. Ela oferece uma excelente visão de todo o ambiente, além de suas dependências, a partir da instalação de um agente (Dynatrace OneAgent).

MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

MÓDULO 3

ELASTIC APM, ELASTICSEARCH E KIBANA

O que é Elasticsearch?

Elasticsearch é um mecanismo de pesquisa e análise de código aberto distribuído, construído no Apache Lucene e desenvolvido em Java. Começou como uma versão escalável da estrutura de pesquisa de código aberto Lucene e, em seguida, adicionou a capacidade de escalar horizontalmente os índices Lucene. O Elasticsearch permite que você armazene, pesquise e analise grandes volumes de dados com rapidez e quase em tempo real e forneça respostas em milissegundos. É capaz de obter respostas de pesquisa rápidas porque, em vez de pesquisar o texto diretamente, ele pesquisa um índice. Ele usa uma estrutura baseada em documentos em vez de tabelas e esquemas e vem com APIs REST abrangentes para armazenar e pesquisar os dados. Basicamente, você pode pensar no Elasticsearch como um servidor que pode processar solicitações JSON e devolver dados JSON.

Como funciona o Elasticsearch?

Alguns conceitos básicos de como ele organiza os dados e seus componentes

Documentos

Documentos são a unidade básica de informação que pode ser indexada no Elasticsearch expressa em JSON, que é o formato global de intercâmbio de dados da Internet. Você pode pensar em um documento como uma linha em um banco de dados relacional, representando uma determinada entidade – aquilo que você está procurando. No Elasticsearch, um documento pode ser mais do que apenas texto, pode ser qualquer dado estruturado codificado em JSON. Esses dados podem ser números, strings e datas. Cada documento possui um ID único e um determinado tipo de dados, que descreve o tipo de entidade do documento. Por exemplo, um documento pode representar um artigo de enciclopédia ou entradas de log de um servidor da web.

Índices

Um índice é uma coleção de documentos com características semelhantes. Um índice é a entidade de nível mais alto que você pode consultar no Elasticsearch. Você pode pensar no índice como sendo semelhante a um banco de dados em um esquema de banco de dados relacional. Quaisquer documentos em um índice são geralmente relacionados logicamente. No contexto de um site de comércio eletrônico, por exemplo, você pode ter um índice para Clientes, um para Produtos, um para Pedidos e assim por diante. Um índice é identificado por um nome que é usado para se referir ao índice ao executar operações de indexação, pesquisa, atualização e exclusão nos documentos nele.

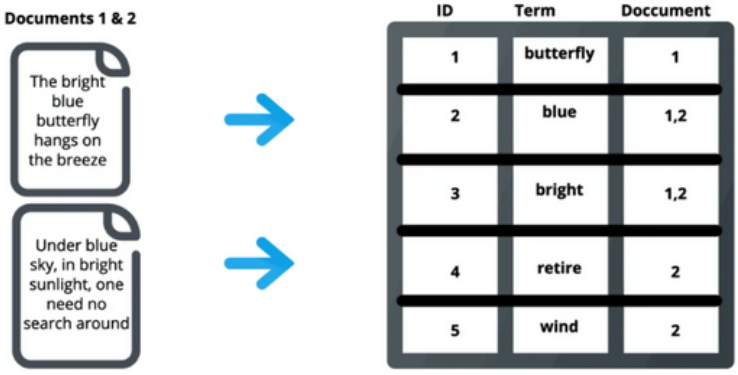
MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

MÓDULO 3

ELASTIC APM, ELASTICSEARCH E KIBANA

Índice Invertido

Um índice no Elasticsearch é, na verdade, o que chamamos de índice invertido, que é o mecanismo pelo qual todos os mecanismos de pesquisa funcionam. É uma estrutura de dados que armazena um mapeamento de conteúdo, como palavras ou números, até suas localizações em um documento ou conjunto de documentos. Basicamente, é uma estrutura de dados semelhante a um hashmap que o direciona de uma palavra para um documento. Um índice invertido não armazena strings diretamente e, em vez disso, divide cada documento em termos de pesquisa individuais (ou seja, cada palavra) e mapeia cada termo de pesquisa para os documentos nos quais esses termos de pesquisa ocorrem. Por exemplo, na imagem abaixo, o termo "melhor" ocorre no documento 2, portanto, está mapeado para esse documento. Isso serve como uma pesquisa rápida de onde encontrar os termos de pesquisa em um determinado documento. Usando índices invertidos distribuídos.



Componentes de back-end

Conjunto: Um cluster Elasticsearch é um grupo de uma ou mais instâncias de nó que estão conectadas. O poder de um cluster Elasticsearch está na distribuição de tarefas, pesquisa e indexação em todos os nós do cluster.

Node (Nó): Um nó é um único servidor que faz parte de um cluster. Um nó armazena dados e participa dos recursos de indexação e pesquisa do cluster. Um nó Elasticsearch pode ser configurado de diferentes maneiras:

- **Node Master (nó mestre):** controla o cluster Elasticsearch e é responsável por todas as operações de todo o cluster, como criar / excluir um índice e adicionar / remover nós.

- **Data Node (nó de dados):** armazena dados e executa operações relacionadas a dados, como pesquisa e agregação.

MÓDULO 3

ELASTIC APM, ELASTICSEARCH E KIBANA

- **Client Node (nó cliente):** encaminha solicitações de cluster para o nó mestre e solicitações relacionadas a dados para nós de dados.

Shards (Fragmentos): Elasticsearch fornece a capacidade de subdividir o índice em várias partes chamadas shards. Cada fragmento é em si um "índice" totalmente funcional e independente que pode ser hospedado em qualquer nó de um cluster. Distribuindo os documentos em um índice em vários fragmentos e distribuindo esses fragmentos em vários nós, o Elasticsearch pode garantir redundância, o que protege contra falhas de hardware e aumenta a capacidade de consulta conforme os nós são adicionados a um cluster.

Réplicas: Elasticsearch permite que você faça uma ou mais cópias dos fragmentos de seu índice, que são chamados de "fragmentos de réplica" ou apenas "réplicas". Basicamente, um shard de réplica é uma cópia de um shard primário. Cada documento em um índice pertence a um fragmento primário. As réplicas fornecem cópias redundantes de seus dados para proteger contra falhas de hardware e aumentar a capacidade de atender a solicitações de leitura, como pesquisa ou recuperação de um documento.

O que é Kibana?

Kibana é uma ferramenta de visualização e gerenciamento de dados para Elasticsearch que fornece histogramas, gráficos de linha, gráficos de pizza e mapas em tempo real. Ele permite que você visualize seus dados do Elasticsearch e navegue no Elastic Stack. Você pode selecionar a forma como dá forma aos seus dados começando com uma pergunta para descobrir aonde a visualização interativa o levará. Por exemplo, como Kibana é frequentemente usado para análise de log, ele permite que você responda a perguntas sobre a origem de seus acessos na web, seus URLs de distribuição e assim por diante. Se você não está construindo seu próprio aplicativo em cima do Elasticsearch, Kibana é uma ótima maneira de pesquisar e visualizar seu índice com uma interface do usuário poderosa e flexível. No entanto, uma grande desvantagem é que cada visualização só pode funcionar em um único índice / padrão de índice. Então, se você tiver índices com dados estritamente diferentes, você terá que criar visualizações separadas para cada um.

O que é o Elastic APM?

O Elastic APM é uma ferramenta da Elastic que possibilita a coleta de dados de performance de aplicações.

MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

MÓDULO 3

ELASTIC APM, ELASTICSEARCH E KIBANA

Se você já utiliza o Elastic Stack para ver logs e métricas da sua aplicação, certamente vai gostar de utilizar o Elastic APM para agregar ainda mais valor em suas análises e troubleshooting da sua aplicação.

O Elastic APM é gratuito e aberto para a comunidade e atualmente está disponível para as seguintes linguagens:

- Java
- Go
- Node.Js
- PHP
- Ruby
- Python
- .NET

Além dos agentes dessas linguagens, também existe o Real User Monitoring (JavaScript), através dele é possível coletar dados da experiência do usuário em aplicações Web.

MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

MÓDULO 3

MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

MÓDULO 3

MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

MÓDULO 4

CONSTRUÇÃO DE UM MICROSSERVIÇO EM PYTHON

Arquitetura Baseada em Eventos

Basicamente uma plataforma orientada a eventos é constituída por um consumer, um API Gateway, um message broker e um producer.

O consumer trata-se o cliente que vai requisitar algum recurso de nossa API. O Message Broker vai se conectar com nosso API Gateway para orquestrar as chamadas à API.

Os producers são os microservices que receberam as chamadas via evento e retornam o resultado de sua execução.

Se nossos recursos estiverem corretamente conectados ao Message Broker praticamente todos os componentes da API podem se comunicar entre si apenas por eventos, sem a necessidade de estarem programaticamente conectados.

Na Prática

Vamos ver a seguir um exemplo de projeto de microservices orientado a eventos construído usando os frameworks Python Flask e Namek, bem como Docker e RabbitMQ como message broker.

Basicamente a responsabilidade do nosso microservices será receber um nome de cidade ou coordenadas geográficas e retornar uma playlist relacionada do Spotify baseada no clima atual da localidade.

Nossa principal implementação trata-se do código do microservices em si. Para isso vamos utilizar o framework Python Nameko. O Nameko foi desenvolvido especificamente para construir microservices, é muito leve e performático.

MÓDULO 4

CONSTRUÇÃO DE UM MICROSSERVIÇO EM PYTHON

```
import os
import json
import sys

from urllib import quote
from nameko.rpc import rpc, RpcProxy
import requests

openweather_base_url = 'http://api.openweathermap.org/data/2.5/weather'
spotify_base_url = 'https://api.spotify.com/v1/'


def error(msg):
    """Return a Exception object

    This function is a reuse function to handle exceptions
    """
    raise Exception(msg)


def request(qstring, jwt):
    """
    Return the result from a Request
    """
    bearer = 'Bearer {}'.format(jwt)

    # JWT necessary to the Spotify Authorization
    if jwt is None:
        return requests.request('GET', qstring)
    else:
        # If is not a Spotify call
        return requests.request(
            'GET', qstring, headers={
                'Authorization': bearer})


def get_playlists(weather, jwt):
    """ Return a list of Spotify playlists
```

CONSTRUÇÃO DE UM MICROSSERVIÇO EM PYTHON

```
From a Weather predict, we consult the
  Spotify API to return results based on the
  weather description.
  """

  filtered = []

  if weather is None:
     error('Missing parameter')

  if weather['weather'] is None:
     error('Invalid location')

  # Main attributte contains the temperature meassure
  main = weather['main']

  # Convert string to int
  temperature = int(main['temp'])

  # Default Genre
  genre = 'classical'

  # If temperature (celcius) is above 30 degrees, suggest tracks for party
  # In case temperature is between 15 and 30 degrees, suggest pop music tracks
  # If it's a bit chilly (between 10 and 14 degrees), suggest rock music tracks
  # Otherwise, if it's freezing outside, suggests classical music tracks
  if temperature > 30:
     genre = 'party'
  elif temperature >= 15 and temperature <= 30:
     genre = 'pop'
  elif temperature >= 10 and temperature <= 14:
     genre = 'rock'

  # Get current weather to check temperatury
  current = weather['weather'][0]

  qstring = '{}search?q=name:{}&type=playlist'.format(
     spotify_base_url, quote(current['description']))

  resp = request(qstring, jwt).json()
```

CONSTRUÇÃO DE UM MICROSSERVIÇO EM PYTHON

```
 # Is receives nothing
   if 'error' in resp:
      e = resp['error']
      error(e['message'])

   # getting plalists itens
   items = resp['playlists']['items']

   # Build the list of the names
   for playlist in items:
      filtered.append(playlist['name'])

   return filtered


def get_weather(args, appid):
   """Return a Weather object
    This call consumes the open Weather Map API to retreive the weather information about some geographi
coordinate or city name.
   """

   # Check if seach by City
   if 'city' not in args:
      # Check if search by coordinates
      if 'lat' not in args or 'lon' not in args:
         error('Missing parameter')
      else:
         # Query string to coordinates
         r_str = '{}?units=metric&lat={}&lon={}&appid={}'.format(
            openweather_base_url, args['lat'], args['lon'], appid)
   else:
      # Query string to City
      r_str = '{}?units=metric&q={}&appid={}'.format(
         openweather_base_url, quote(
            args['city']), appid)

   # Requesting the API
   resp = request(r_str, None)

   # Returns a JSON object
   return resp.json()


class PlaylistsService:
   """Playlists Srevice
```

MÓDULO 4

CONSTRUÇÃO DE UM MICROSSERVIÇO EM PYTHON

```
Returns:
        list: A list of playlists
    """

    name = "playlists"

    zipcode_rpc = RpcProxy('playlistsservice')

    @rpc
    def get_playlists(self, appid, jwt, args):
        """[summary]
        Arguments:
            appid (string): Open Weather Map AppID
            jwt   (string): Spotify Authorization Token
            args    (dict): A Dict (city, lat, long)
        Returns:
            list: A list of playlists
        """
        try:
            # Check if is passed args
            if args is None:
                error("Missing parameter")

            # Consuming the OWM API
            weather = get_weather(args, appid)

            # Check response
            if int(weather['cod']) != 200:
                return weather

            # Consuming Spotify API
            playlists = get_playlists(weather, jwt)

            return playlists
        except Exception as e:
            return str({'Error': str(e)})
```

MÓDULO 4

CONSTRUÇÃO DE UM MICROSSERVIÇO EM PYTHON

A outra parte importante é servir o microservices através de eventos. Para isso vamos utilizar o bom e velho Flask. O Flask vai fazer as vezes do nosso API Gateway e servirá nosso microservices na porta http 5000:

```
from flask import Flask, request, jsonify
from nameko.standalone.rpc import ClusterRpcProxy

CONFIG = {'AMQP_URI': "amqp://guest:guest@localhost:5672"}
app = Flask(__name__)


@app.route('/playlists', methods=['GET'])
def playlists():
    """
    Micro Service Based Weather and Spotify API
    This API is made with Flask and Nameko
    ---
    parameters:
      - name: zipcode
        in: path
        required: true
        schema:
          type: integer
        description: location ZipCode
    responses:
      200:
        description: Location data
    """
    with ClusterRpcProxy(CONFIG) as rpc:
        # Get the Spotify Authrization Token from header
        # and OWM AppId from the header
        openwm_appid = request.headers.get('X-OPENWM-APPID')
        spotify_token = request.headers.get('X-SPOTIFY-TOKEN')
# Consuming Nameko service
# Here we pass the OpenWeatherMap AppId
# and the Spotify JWT
result = rpc.playlists.get_playlists(
openwm_appid, spotify_token, request.args)
return jsonify(result), 200


if __name__ == "__main__":
"""Start Flask app to serve mircoservices"""
app.run(host='0.0.0.0')
```

MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

MÓDULO 4

COMUNICAÇÃO ASSÍNCRONA USANDO KAFKA, NODEJS E PYTHON

Código assíncrono

A computação assíncrona está se tornando o carro chefe do manual dos desenvolvedores da computação distribuída.

Cada vez mais dependente de serviços de terceiros, chamadas SDK remotos e computação em nuvem, os softwares precisam trabalhar com o delay entre as respostas e permitir que o usuário continue utilizando seu produto.

Código assíncrono é um código não bloqueante que recebe um input de dados e retorna um resultado, completo ou parcial. O retorno pode ser feito por meio de um endpoint de callback ou uma nova consulta.

Mas e quando precisamos receber a resposta completa em uma única chamada? Para isso utilizamos uma arquitetura baseada em evento.

Com essa arquitetura podemos efetuar uma chamada à uma API. O message borker converte a requisição em um evento e o microservice responsável pelo recurso requisitado responderá.

Esse tipo de abordagem pode resolver muitos problemas de assincronicidade pois pode escalar facilmente as instâncias do serviço.

Integração assíncrona com o Apache Kafka

Os microsserviços mudaram o panorama do desenvolvimento. Eles tornaram os desenvolvedores mais ágeis ao reduzir as dependências, como as camadas de banco de dados compartilhado. No entanto, as aplicações distribuídas que estão sendo criadas pelos desenvolvedores ainda precisam de algum tipo de integração para compartilhar dados. Uma opção de integração muito usada, conhecida como método síncrono, utiliza interfaces de programação de aplicações (APIs) para compartilhar dados entre usuários diferentes.

MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

MÓDULO 4

COMUNICAÇÃO ASSÍNCRONA USANDO KAFKA, NODEJS E PYTHON

Uma segunda opção de integração, o método assíncrono, envolve a replicação de dados em um armazenamento intermediário. É aí que o Apache Kafka entra em cena, na transmissão de dados de outras equipes de desenvolvimento para preencher o armazenamento de dados. Assim, é possível compartilhar os dados entre várias equipes e respectivas aplicações.

As equipes de microsserviços têm requisitos de integração diferentes de outras equipes de desenvolvimento em cascata tradicionais. Elas requerem três recursos fundamentais:

1. Integrações distribuídas: integrações leves e baseadas em padrões que podem ser implantadas continuamente quando necessário e não estão limitadas por implantações do tipo ESB centralizado.
2. APIs: serviços baseados em API para incentivar um ecossistema de parceiros, clientes e desenvolvedores que podem oferecer serviços confiáveis e lucrativos.
3. Containers: plataforma para desenvolver, gerenciar e escalar aplicações nativas em nuvem e conectadas. Os containers possibilitam o desenvolvimento de artefatos lean que podem ser implantados individualmente, fazem parte de processos de DevOps e são compatíveis com soluções prontas de clusterização, assegurando uma alta disponibilidade.

A Red Hat chama essa abordagem de "integração ágil", o que permite que as integrações façam parte de processos de desenvolvimento de aplicações, fornecendo soluções mais ágeis e adaptáveis. Um elemento da integração ágil é a liberdade de usar a integração síncrona ou assíncrona, dependendo das necessidades específicas da aplicação. O Apache Kafka é uma excelente opção na integração assíncrona orientada por eventos para aumentar o uso de integração síncrona e APIs, o que oferece ainda mais compatibilidade com microsserviços e possibilita a integração ágil. Dessa forma, o Apache Kafka pode ser uma parte importante da sua iniciativa para simplificar os processos de desenvolvimento, incentivar a inovação, poupar tempo e, em última análise, acelerar o time to market de novas funcionalidades, aplicações e serviços.

MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

MÓDULO 4

JAVA SPRING DATA REST PARA ENTENDER O HATEOAS

O que é HATEOAS?

HATEOAS é uma restrição que faz parte da arquitetura de aplicações REST, cujo objetivo é ajudar os clientes a consumirem o serviço sem a necessidade de conhecimento prévio profundo da API.

O acrônimo HATEOAS vem de Hypermedia As the Engine Of Application State e o termo “hypermedia” no seu nome já dá uma ideia de como este componente funciona em uma aplicação RESTful. Ao ser implementado, a API passa a fornecer links que indicarão aos clientes como navegar através dos seus recursos.

Com isso, o cliente não precisa ter um conhecimento profundo da API, basta conhecer a URL de inicial e partir dos links fornecidos poderá acessar todos os recursos de forma circular, se guiando através das requisições realizadas.

Um exemplo clássico para explicar o HATEOAS, é o Hypertext, do HTML. Quando se necessita obter uma informação de um site, como um curso aqui da Treinaweb, o usuário pode acessar a página inicial (treinaweb.com.br), nela clicar em Cursos e depois no curso desejado. No fim, o usuário terá acessado uma URL como treinaweb.com.br/curso/java-fundamentos-de-jax-ws-e-jax-rs, onde conseguirá visualizar o curso. Não foi necessário que o usuário soubesse de antemão a URL desejada, pois é importante que o site, através de links, guie-o e auxilie-o na busca do recurso desejado. Com o HATEOAS uma API REST segue o mesmo padrão

Um exemplo simples do HATEOAS

Por exemplo, ao acessar uma API de cursos, na URL api.treinaweb.com.br/cursos, o cliente terá o seguinte resultado:

MÓDULO 4

JAVA SPRING DATA REST PARA ENTENDER O HATEOAS


```
{
   "cursos": [
      {
         "id": 1,
         "nome": "C# (C Sharp)",
         "aulas": [
            {
               "id": 1,
               "titulo": "Título da aula 3"
            },
            {
               "id": 2,
               "titulo": "Título da aula 3"
            },
            {
               "id": 3,
               "titulo": "Título da aula 3"
            },
         ]
      },
      {
         "id": 2,
         "nome": "PHP",
         "aulas": [
            {
               "id": 1,
               "titulo": "Título da aula 3"
            },
            {
               "id": 2,
               "titulo": "Título da aula 3"
            },
            {
               "id": 3,
               "titulo": "Título da aula 3"
            },
         ]
      },
      {
         "id": 3,
```

MÓDULO 4

JAVA SPRING DATA REST PARA ENTENDER O HATEOAS


```
 "nome": "Java",
        "aulas": [
           {
              "id": 1,
              "titulo": "Título da aula 3"
           },
           {
              "id": 2,
              "titulo": "Título da aula 3"
           },
           {
              "id": 3,
              "titulo": "Título da aula 3"
           },
        ]
     },
  ]
}
```


Se esta API implementasse o padrão HATEOAS, a resposta obtida poderia ser outra. No lugar de listar as aulas, cada curso teria um recurso próprio que retornaria suas aulas. Evitando assim que elas fiquem expostas do corpo da requisição. Algo como o exemplo abaixo:


```
{
  "cursos": [
     {
        "id": 1,
        "nome": "C# (C Sharp)",
        "aulas": "api.treinaweb.com.br/cursos/1/aulas"
     },
     {
        "id": 2,
        "nome": "PHP",
        "aulas": "api.treinaweb.com.br/cursos/2/aulas"
     },
     {
        "id": 3,
        "nome": "Java",
        "aulas": "api.treinaweb.com.br/cursos/3/aulas"
     },
  ]
}
```

JAVA SPRING DATA REST PARA ENTENDER O HATEOAS

Desta forma, se o cliente desejasse obter as aulas do curso de PHP, em seguida acessaria a URL api.treinaweb.com.br/cursos/2/aulas. Note que nem é necessário montar a URL, a API já fornece o caminho correto completo.

Especificação do HATEOAS
No exemplo anterior, o HATEOAS foi exemplificado como uma URL na resposta da API, entretanto, na prática, recomenda-se que a implementação da HATEOAS siga um dos padrões abaixo:

RFC 5988
A especificação RFC 5988 da IETF define como links devem ser implementados. De acordo com ela, cada link deve ter as informações:

- URI: Cada link deve conter uma URI, representada no atributo href;
- Tipo de relação: Descreve como a URI se relaciona com o recurso atual, representado pelo atributo rel, devidado de relationship;
- Atributos para URI: Para descrever melhor a URI podem ser adicionados atributos como: hreflang, media, title e type.

JSON Hypermedia API Language (HAL)
JSON HAL é uma especificação, proposta por Mike Kelly, ainda não aprovada, mas já comumente utilizada, que define dois MIME Types:

application/hal+xml
application/hal+json

Que ao serem enviados na solicitação, a API REST deve retornar uma propriedade links, contendo as informações:

URI: A URI do recurso, representada pelo atributo href;
Tipo de relação: Descreve como a URI se relaciona com o recurso atual, representado pelo atributo rel;
Tipo: Descreve o tipo de conteúdo obtido ou do tipo de verbo que deve ser utilizado para acessar a URI. Representado pelo atributo type.
Desta forma, na prática, uma API REST que implemente HATEOAS retornar uma resposta como a abaixo:

MÓDULO 4

JAVA SPRING DATA REST PARA ENTENDER O HATEOAS


```
{
   "cursos": [
      {
         "id": 1,
         "nome": "C# (C Sharp)",
         "links": [
            {
               "type": "GET",
               "rel": "self",
               "uri": "api.treinaweb.com.br/cursos/1"
            },
            {
               "type": "GET",
               "rel": "curso_aulas",
               "uri": "api.treinaweb.com.br/cursos/1/aulas"
            },
            {
               "type": "PUT",
               "rel": "curso_atualizacao",
               "uri": "api.treinaweb.com.br/cursos/1"
            },
            {
               "type": "DELETE",
               "rel": "curso_exclusao",
               "uri": "api.treinaweb.com.br/cursos/1"
            }
         ]
      },
      {
         "id": 2,
         "nome": "PHP",
         "links": [
            {
               "type": "GET",
               "rel": "self",
               "uri": "api.treinaweb.com.br/cursos/2"
            },
            {
               "type": "GET",
               "rel": "curso_aulas",
               "uri": "api.treinaweb.com.br/cursos/2/aulas"
            },
```

MÓDULO 4

JAVA SPRING DATA REST PARA ENTENDER O HATEOAS


```
{
            "type": "PUT",
            "rel": "curso_atualizacao",
            "uri": "api.treinaweb.com.br/cursos/2"
          },
          {
            "type": "DELETE",
            "rel": "curso_exclusao",
            "uri": "api.treinaweb.com.br/cursos/2"
          }
        ]
    },
    {
      "id": 3,
      "nome": "Java",
      "links": [
        {
          "type": "GET",
          "rel": "self",
          "uri": "api.treinaweb.com.br/cursos/3"
        },
        {
          "type": "GET",
          "rel": "curso_aulas",
          "uri": "api.treinaweb.com.br/cursos/3/aulas"
        },
        {
          "type": "PUT",
          "rel": "curso_atualizacao",
          "uri": "api.treinaweb.com.br/cursos/3"
        },
        {
          "type": "DELETE",
          "rel": "curso_exclusao",
          "uri": "api.treinaweb.com.br/cursos/3"
        }
      ]
    }
  ]
}
```

MÓDULO 4

JAVA SPRING DATA REST PARA ENTENDER O HATEOAS

Vantagens do HATEOAS

Uma das principais vantagens do HATEOAS é permitir representar o estado de um recurso ou as limitações de um usuário sem a necessidade da implementação de outras lógicas no cliente.

Por exemplo, imagine uma API de um banco. Ao consultar o saldo do usuário, caso este tenha saldo em conta, pode ser retornado os seguintes dados:

```
{
   "conta": {
      "numero": 1234,
      "saldo": 5120.09,
      "links":
      [
         {
            "type": "GET",
            "rel": "self",
            "uri": "api.banco.com.br/usuario/1/conta/1234"
         },
         {
            "type": "PUT",
            "rel": "conta_saque",
            "uri": "api.banco.com.br/usuario/1/conta/1234/saque"
         },
         {
            "type": "PUT",
            "rel": "conta_deposito",
            "uri": "api.banco.com.br/usuario/1/conta/1234/deposito"
         }
      ]
   }
}
```

Caso não tenha saldo, o resultado pode mudar para:

```
{
   "conta": {
      "numero": 1234,
      "saldo": -20.14,
      "links":
      [
         {
```

MÓDULO 4

JAVA SPRING DATA REST PARA ENTENDER O HATEOAS

```
 "type": "GET",
            "rel": "self",
            "uri": "api.banco.com.br/usuario/1/conta/1234"
         },
         {
            "type": "PUT",
            "rel": "conta_deposito",
            "uri": "api.banco.com.br/usuario/1/conta/1234/deposito"
         }
      ]
   }
}
```

Desta forma, o cliente não precisa se preocupar com o estado do usuário. Ele saberá que os ações disponíveis serão as retornadas no atributo links.

Outra vantagem é a prevenção do chamado “hardcoding”, quando informações são inseridas diretamente no código da aplicação. Em todos os nossos exemplos vimos que o cliente precisa conhecer apenas a URL base da API, pois esta já irá retornar as demais URLs disponíveis.

Por este comportamento que o HATEOAS geralmente é aplicado em aplicações de proposito mais geral, que são criadas para sobreviverem por um longo período. Onde alterações na API não irão impactar muito nos clientes.

Spring Data REST:
O módulo data rest do spring nos ajuda a disponibilizar operações CRUD com o mínimo de esforço possível. Imagine que você precise implementar um cadastro de produtos com regras complexas, mas que, antes disso, também tenha que ter um cadastro de “categoria de produtos”, que simplesmente insere uma nova categoria sem nenhuma regra complexa para se preocupar. Nesse contexto, o spring data rest pode facilitar bastante o trabalho, disponibilizando todas as operações no padrão REST para isso.

```
<dependencies>
      <dependency>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-starter</artifactId>
      </dependency>
      <dependency>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-starter-data-rest</artifactId></dependency>
```

JAVA SPRING DATA REST PARA ENTENDER O HATEOAS

```
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
      <groupId>com.h2database</groupId>
      <artifactId>h2</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>
  </dependencies>
```

Feito isso, vamos criar a entidade PERSON, que será utilizada para o exemplo:

```
@Entity
public class Person {

   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private Long id;
   private String name;
   private String document;

   [...] //Getters and Setters
}
```

E para finalizar, criamos a classe Repository da entidade, onde vamos ter a anotação responsável por disponibilizar as operações rest.

```
@RepositoryRestResource(collectionResourceRel = "persons", path = "persons")
public interface PersonRepository extends JpaRepository<Person, Long> {
}
```

Vamos fazer um POST para a url http://localhost:8080/persons, com o seguinte payload:

```
{
 "name": "Steve Rogers",
 "document": "0000000001"
}
```

JAVA SPRING DATA REST PARA ENTENDER O HATEOAS

Nesse caso temos como response um 201 Created e o body:

```
{
 "name": "Steve Rogers",
 "document": "0000000001",
 "_links": {
 "self": {
 "href": "http://localhost:8080/persons/1 "
      },
 "person": {
 "href": "http://localhost:8080/persons/1 "
      }
   }
}
```

Note que o retorno já faz referência a url GET do próprio registro.

Para fazer update dos dados, podemos fazer um POST para a url http://localhost:8080/persons/1 com o seguinte payload:

```
{
 "name": "Tony Stark",
 "document": "0000000001"
}
```

E para deletar fazemos DELETE para a mesma url, e nesse caso receberemos como retorno o código 204 No Content.

Após isso, se fizermos um GET para a url acima, passaremos a receber um 404 Not Found, já que o registro foi deletado.

MÓDULO 4

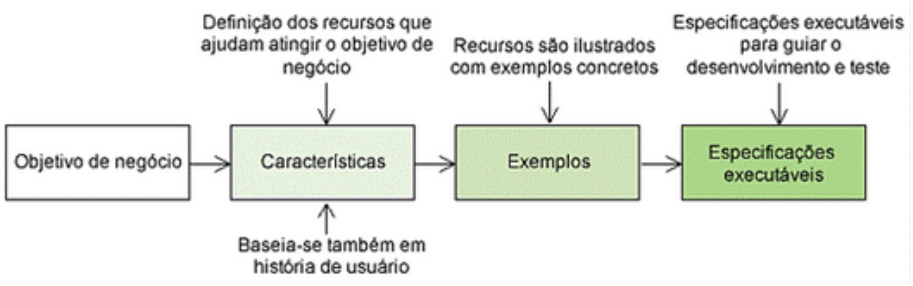
CI/CD E SONNARQUBE COM O USO DE TESTES AUTOMATIZADOS ATRAVÉS DO CUCUMBER (FRAMEWORK BDD)

O que é Behavior Driven Development (BDD)?

O Behavior Driven Development (BDD) ou desenvolvimento orientado por comportamento foi inventado por Dan North no ano de 2000, quando Dan percebeu que muitas equipes tinham dificuldades de adotar e usar eficazmente o TDD, criado como uma versão melhorada do desenvolvimento orientado por testes (TDD, criado por Kent Beck). O BDD não é uma metodologia de desenvolvimento de software, tão pouco um substituto para o XP, Scrum, Kanban, OpenUP, RUP ou qualquer metodologia que o mercado atualmente oferece, mas sim, o BDD incorpora e melhora as ideias de muitas dessas metodologias, ajudando assim e tornando a vida da equipe de software mais fácil. Portanto, o BDD é um conjunto de práticas de engenharia de software projetado para ajudar as equipes a construir e entregar mais rápido software de alta qualidade.

Para explicar o funcionamento do BDD vamos usar o seguinte exemplo: uma equipe praticante de BDD decide implementar uma nova funcionalidade e para isso, eles trabalham em conjunto com os usuários e outras partes interessadas para definir as histórias e cenários do que os usuários esperam dessa funcionalidade. Em particular, os usuários ajudam a definir um conjunto de exemplos concretos que ilustram resultados que a nova funcionalidade deve fornecer. Esses exemplos são criados utilizando um vocabulário comum e podem ser facilmente compreendidos pelos usuários finais e membros da equipe de desenvolvimento de software, e geralmente são expressos usando Cenário(Scenario), Dado(Given), Quando(When) e Então (Then).

Vejamos a Figura 1 que mostra os passos do BDD utilizado pela equipe neste exemplo para especificação da nova funcionalidade.



MÓDULO 4

CI/CD E SONNARQUBE COM O USO DE TESTES AUTOMATIZADOS ATRAVÉS DO CUCUMBER (FRAMEWORK BDD)

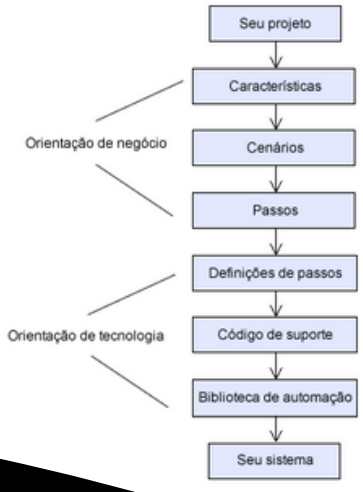
Com base no BDD, a equipe identificou e especificou o seguinte objetivo de negócio, definido com um exemplo concreto. Observe:

- Cenário: Transferir dinheiro para uma conta poupança
- Dado que eu tenho uma conta corrente com 1000.00
- E que eu tenho uma conta de poupança com 2.000,00
- Quando eu transferir 500,00 a partir de minha conta corrente para a minha conta poupança
- Então eu deveria ter 500,00 em minha conta corrente
- E eu deveria ter 2.500,00 em minha conta poupança

Depois de especificada a nova funcionalidade, sempre que possível estes exemplos concretos são automatizados sob a forma de especificações executáveis, que tanto valida o software quanto fornece uma documentação atualizada, técnica e funcional. Logo, existem várias ferramentas e frameworks que apoiam esta fase do BDD, transformando esses requisitos em testes automatizados que ajudam a orientar o desenvolvedor para que a nova funcionalidade seja desenvolvida corretamente e dentro do prazo.

Conhecendo o Cucumber

O Cucumber foi originalmente criado por membros da comunidade Ruby para apoiar o desenvolvimento de testes de aceitação automatizado utilizando a técnica BDD. Desde então o Cucumber cresceu e foi traduzido em várias linguagens, inclusive o Java, permitindo assim que vários de desenvolvedores desfrutem de suas vantagens. Diante disso, vejamos a Figura 2, que ilustra uma visão geral do Cucumber.



MÓDULO 4

USO DE API GATEWAY PARA AUTENTICAÇÃO E AUTORIZAÇÃO

O que é um API Gateway?

O API Gateway é um gerenciador de tráfego que faz interface com o serviço de back-end real ou de dados e aplica políticas, autenticação e controle de acesso geral para chamadas de APIs de forma a proteger dados valiosos. O API Gateway é a maneira de controlar o acesso aos seus sistemas e serviços de back-end e foi projetado para otimizar a comunicação entre clientes externos e seus serviços de back-end, oferecendo aos seus clientes uma experiência perfeita. O API Gateway garante escalabilidade e alta disponibilidade de seus serviços. Ele é responsável por encaminhar a solicitação ao serviço apropriado e enviar uma resposta ao solicitante. O API Gateway mantém uma conexão segura entre seus dados e APIs e gerencia o tráfego e as solicitações das APIs, incluindo balanceamento de carga, tanto dentro quanto fora de sua empresa. O Gateway aplica políticas, autenticação e controle de acesso geral para chamadas de API de forma a proteger dados valiosos. O API Gateway recebe todas as chamadas de API de clientes e as encaminha para o microsserviço correto usando roteamento de solicitação, composição e tradução de protocolo.

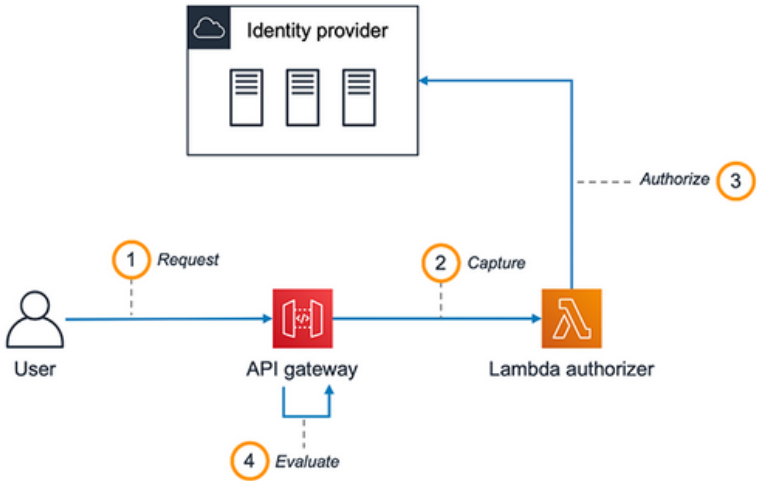
Um dos principais motivos pelos quais o API Gateway é usado é que ele é capaz de chamar vários serviços de backend e agregar os resultados. Em vez dos clientes precisarem enviar uma solicitação para cada serviço individual, eles podem enviá-las para o API Gateway, que então passa a solicitação para o serviço relevante. Além disso, o API Gateway fornece uma alternativa para um estilo de API de tamanho único. O API Gateway também pode expor uma API diferente para cada cliente. Hoje uma necessidade nos ambientes em constante evolução.

MÓDULO 4

USO DE API GATEWAY PARA AUTENTICAÇÃO E AUTORIZAÇÃO

Autenticação

Um API Gateway pode ser usado para autenticar chamadas de API. Dessa forma, mesmo que o cliente precise acessar dados de vários serviços, ele só precisará fazer a autenticação uma vez no gateway. Isso reduz a latência e garante que os processos de autenticação sejam consistentes em todo o aplicativo. Semelhante a como quando um passaporte é usado para verificar sua identidade ou um visto para provar que você tem permissão para trabalhar em um determinado país, o API Gateway fornece várias maneiras para os consumidores se autenticarem e obterem acesso aos seus recursos de API. Os API Gateways podem usar um dos muitos padrões abertos para determinar a identidade ou validade do consumidor (ou seja, OAuth, tokens JWT, chave API, HTTP Basic / Digest, SAML, etc.) ou pode usar meios não padrão para localizar credenciais nos cabeçalhos ou na carga útil da mensagem. Os API Gateways também podem chamar outros sistemas para verificar a identidade, da mesma forma que a polícia pode acessar um banco de dados de criminosos. Além disso, como na alfândega em um aeroporto, um API Gateway também pode verificar se há ameaças em um consumidor de API de entrada. Eles podem usar API Firewalling, validação de conteúdo e verificações de integridade de mensagem, que incluem determinar se uma API foi adulterada. Um API Gateway também pode delegar a avaliação de risco de uma API de entrada a um aplicativo de terceiros para que eles façam a determinação.

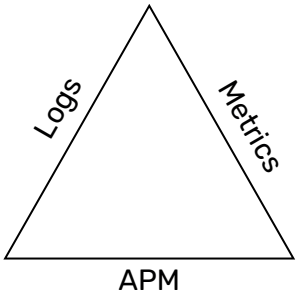


MÓDULO 4

OBSERVABILIDADE PARA MONITORAR TEMPO DE RESPOSTAS DE APLICAÇÕES USANDO ELASTIC APM

O que é o Application Performance Monitoring (APM)?

Quando falamos do APM, acho interessante falar nele como uma das facetas da "observabilidade", que também engloba os registros e métricas de infraestrutura. Os três juntos formam o tripé da observabilidade:



E existem sobreposições nessas áreas, o suficiente para podermos correlacioná-las. Os registros podem indicar que ocorreu um erro, mas não explicar por quê. As métricas podem mostrar que o uso da CPU teve um pico em um servidor, mas não indicar o que o causou. Por isso, quando usamos esses recursos juntos, conseguimos resolver muito mais problemas.

Registros

Primeiro, vamos ver algumas definições. Existe uma diferença sutil entre registros e métricas. No geral, registros são eventos emitidos quando alguma coisa acontece: por exemplo, uma solicitação é recebida e respondida, um arquivo é aberto, um printf é encontrado no código.

Um formato de registro comum, por exemplo, vem do projeto do servidor Apache HTTP (falso e reduzido em tamanho):

```
264.242.88.10 - - [22/Jan/2018:07:08:53 -0800] "GET /ESProductDetailView HTTP/1.1" 200 6291
264.242.88.10 - - [22/Jan/2018:07:08:53 -0800] "POST /intro.m4v HTTP/1.1" 404 7352
264.242.88.10 - - [22/Jan/2018:16:38:53 -0800] "POST /checkout/addresses/ HTTP/1.1" 500 5253
```

MÓDULO 4

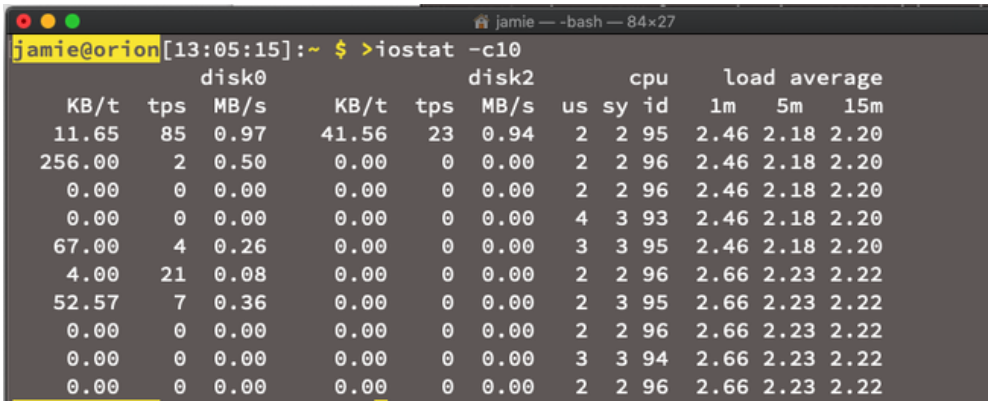
OBSERVABILIDADE PARA MONITORAR TEMPO DE RESPOSTAS DE APLICAÇÕES USANDO ELASTIC APM

Os registros ainda tendem a estar em nível do componente e não do aplicativo como um todo. E eles são bons porque são facilmente entendidos pelas pessoas. Nos exemplos acima, vemos um endereço de IP, um campo que aparentemente não está definido, uma data, a página que o usuário estava acessando (mais o método) e alguns números. Pela minha experiência, sei que os números são o código de resposta (200 é bom, 404 não é bom, mas é melhor do que 500) e a quantidade de dados que retornaram.

Os registros também são úteis porque costumam estar disponíveis no servidor/máquina/contêiner em que o aplicativo ou serviço está sendo executado e, como vemos acima, são compreensíveis. O lado ruim dos registros é a própria natureza deles. Se você não codificá-los, eles poderão ser impressos. É necessário fazer algo equivalente ao puts no Ruby ou system.out.println no Java para que ele possa ser divulgado. Mesmo que você faça isso, é importante formatar. Os registros do Apache acima têm o que parece ser um formato de data estranho. Veja, por exemplo, a data 01/02/2019. Para mim, nos EUA, significa 2 de janeiro de 2019, mas, para um grande número de pessoas, seria 1 de fevereiro. Pense sobre esse tipo de coisa quando estiver formatando enunciados de registro.

Métricas

As métricas, por outro lado, tendem a ser resumos ou contagens periódicas. Nos últimos 10 segundos, a CPU médio foi de 12%, a quantidade de memória usada por um aplicativo foi de 27 MB ou o disco principal estava em 71% da capacidade (o meu, acabei de checar).



MÓDULO 4

OBSERVABILIDADE PARA MONITORAR TEMPO DE RESPOSTAS DE APLICAÇÕES USANDO ELASTIC APM

Veja acima uma captura de tela do iostat em um Mac. São muitas métricas. Elas são úteis para mostrar tendências e histórico e excelentes para tentar criar regras simples, previsíveis e confiáveis para pegar incidentes e anomalias. Um dos problemas com as métricas é que elas tendem a monitorar a camada de infraestrutura, obtendo dados em nível do componente, como host, contêiner e rede, em vez de em nível do aplicativo. Como as métricas tendem a ser de um período, é importante que elas sejam breves para manter seu valor.

APM

O Application Performance Monitoring resolve o abismo entre as métricas e os registros. Os registros e métricas têm mais a ver com infraestrutura e componentes, enquanto o APM se concentra nos aplicativos e permite que profissionais de TI e desenvolvedores monitorem a camada da aplicação do stack, incluindo a experiência do usuário final.

Ao adicionar o APM ao seu monitoramento, você vai:

- Entender em que o seu serviço está gastando tempo e por que ele para de funcionar.
- Ver como os serviços interagem uns com os outros e visualizar gargalos.
- Descobrir com antecedência gargalos e erros de performance para corrigi-los.
- Antes que grande parte dos seus clientes sofram as consequências.
- Aumentar a produtividade da equipe de desenvolvimento.
- Monitorar a experiência do usuário final no navegador.

Algo importante a se observar é que a linguagem do APM é o código (vamos falar mais disso mais à frente).

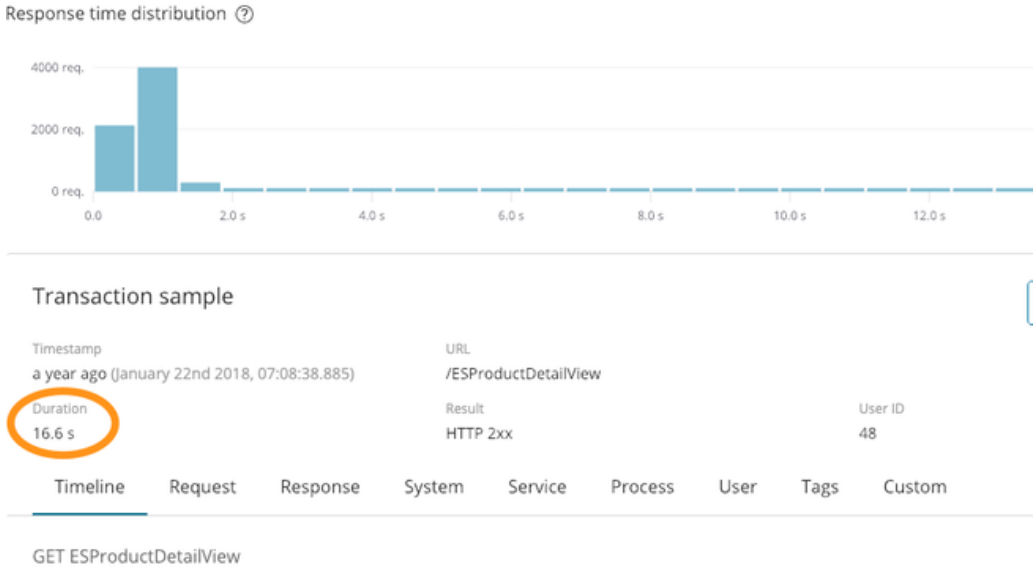
Vamos ver como o APM se compara aos registros. Observemos esta entrada de registro:

264.242.88.10 - - [22/Jan/2018:07:08:53 -0800] "GET /ESProductDetailView HTTP/1.1" 200 6291

Inicialmente, tudo parece bom. Obtivemos sucesso com a resposta (200) e enviamos de volta 6291 bytes. O que ele não mostra é que levou 16,6 segundos, como vemos nessa captura de tela do APM:

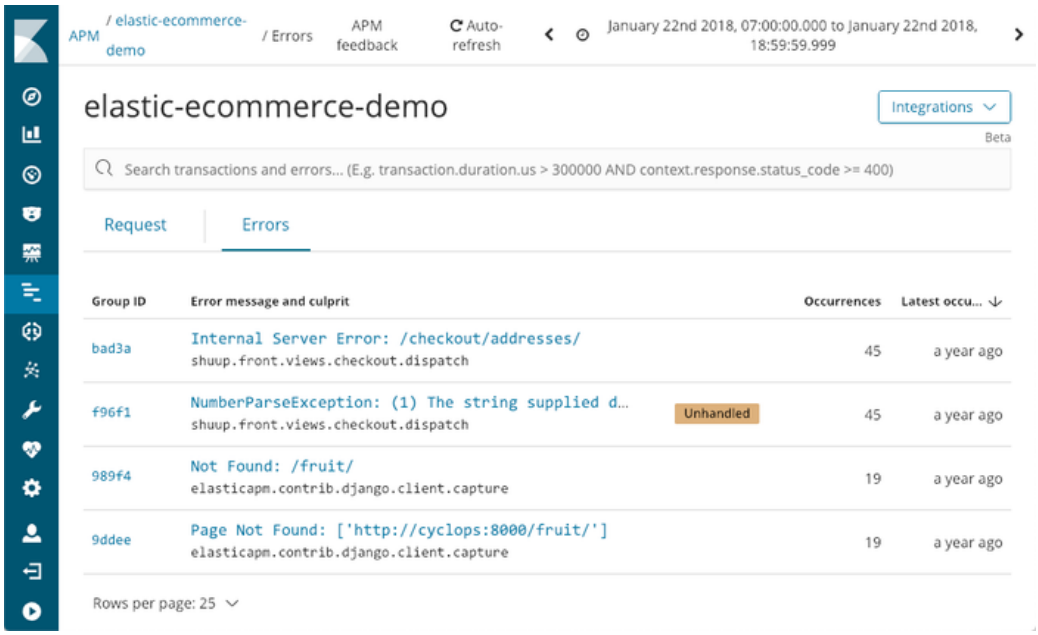
MÓDULO 4

OBSERVABILIDADE PARA MONITORAR TEMPO DE RESPOSTAS DE APLICAÇÕES USANDO ELASTIC APM



Essa última informação é muito importante. Também tivemos um erro nos registros acima:

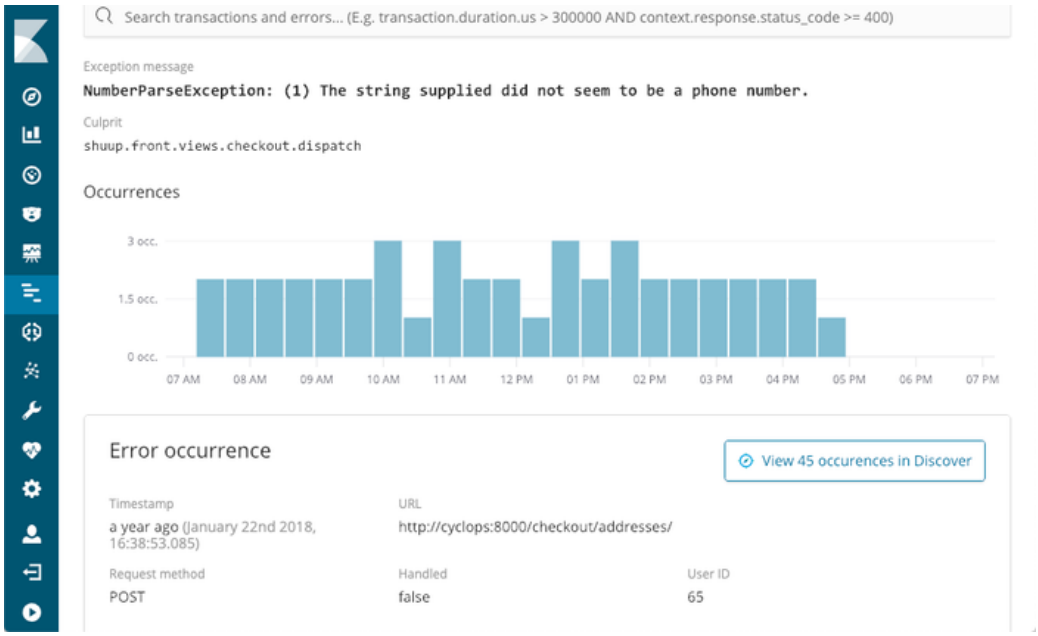
O APM também captura erros para nós:



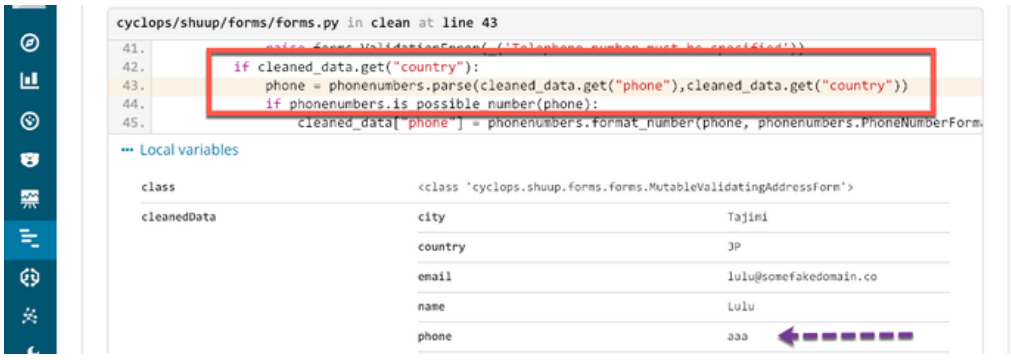
MÓDULO 4

OBSERVABILIDADE PARA MONITORAR TEMPO DE RESPOSTAS DE APLICAÇÕES USANDO ELASTIC APM

Ele mostra qual foi a última vez em que ele aconteceu, com qual frequência aconteceu e se ele foi ou não resolvido pelo aplicativo. Conforme exploramos em busca de uma exceção, usando o NumberParseException como exemplo, recebemos uma visualização da distribuição do número de vezes em que aquele erro ocorreu na janela:



E conseguimos ver imediatamente que aconteceu diversas vezes por período, mas o dia todo. Seria possível encontrar o stack trace correspondente em um dos arquivos de registro, mas provavelmente em contexto e os metadados disponíveis no APM:

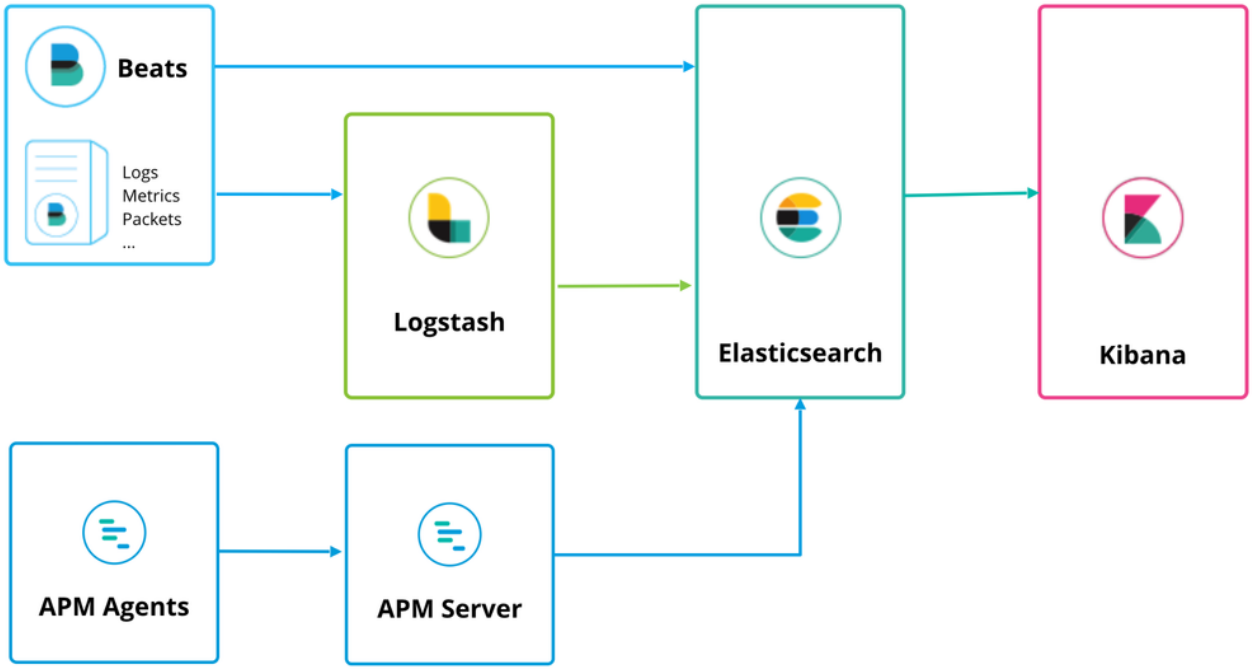


MÓDULO 4

OBSERVABILIDADE PARA MONITORAR TEMPO DE RESPOSTAS DE APLICAÇÕES USANDO ELASTIC APM

Como começar a usar o Elastic APM

O Elastic APM pode funcionar junto com o Logstash e o Beats, com uma topologia de implementação similar:



O servidor do APM funciona como um processador de dados, encaminhando os dados do APM dos agentes do APM para a Elasticsearch. A instalação é bem simples e pode ser encontrada na página “install and run” da documentação ou clicando no logo K do Kibana para acessar a tela inicial do serviço onde você verá a opção “Add APM”.

Resumindo:

O APM nos permite ver o que está acontecendo em nossos aplicativos, em todas as camadas. Com as integrações com o aprendizado de máquina e os alertas e o poder da busca, o Elastic APM adiciona uma outra camada de visibilidade à sua infraestrutura de aplicativos. Podemos usá-lo para visualizar transações, traces, erros e exceções, tudo no contexto de uma interface de usuário preparada do APM. Mesmo quando não temos problemas, podemos aproveitar os dados do Elastic APM para ajudar a priorizar as correções, melhorar o desempenho dos aplicativos e resolver gargalos.

MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

MÓDULO 4

MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

MÓDULO 4

MICROSERVICES (HANDS-ON)
API GATEWAY, WEBHOOKS & ELASTIC APM

MÓDULO 4