Compreendendo e implementando microfrontends em AWS

AWS Orientação prescritiva

AWS Orientação prescritiva: Compreendendo e implementando microfrontends em AWS

Table of Contents

Compreendendo e implementando microfront-ends em AWS

Amazon Web Services (colaboradores)

Julho de 2024 (histórico do documento)

À medida que as organizações buscam agilidade e escalabilidade, a arquitetura monolítica convencional geralmente se torna um gargalo, impedindo o rápido desenvolvimento e a implantação. Os microfront-ends mitigam isso dividindo interfaces de usuário complexas em componentes menores e independentes que podem ser desenvolvidos, testados e implantados de forma autônoma. Essa abordagem aumenta a eficiência das equipes de desenvolvimento e facilita a colaboração entre back-end e front-end, promovendo o alinhamento dos sistemas distribuídos. end-to-end

Essa orientação prescritiva é personalizada para ajudar líderes de TI, proprietários de produtos e arquitetos em diversos domínios profissionais a entender a arquitetura de microfront-end e criar aplicativos de microfront-end na Amazon Web Services ().AWS

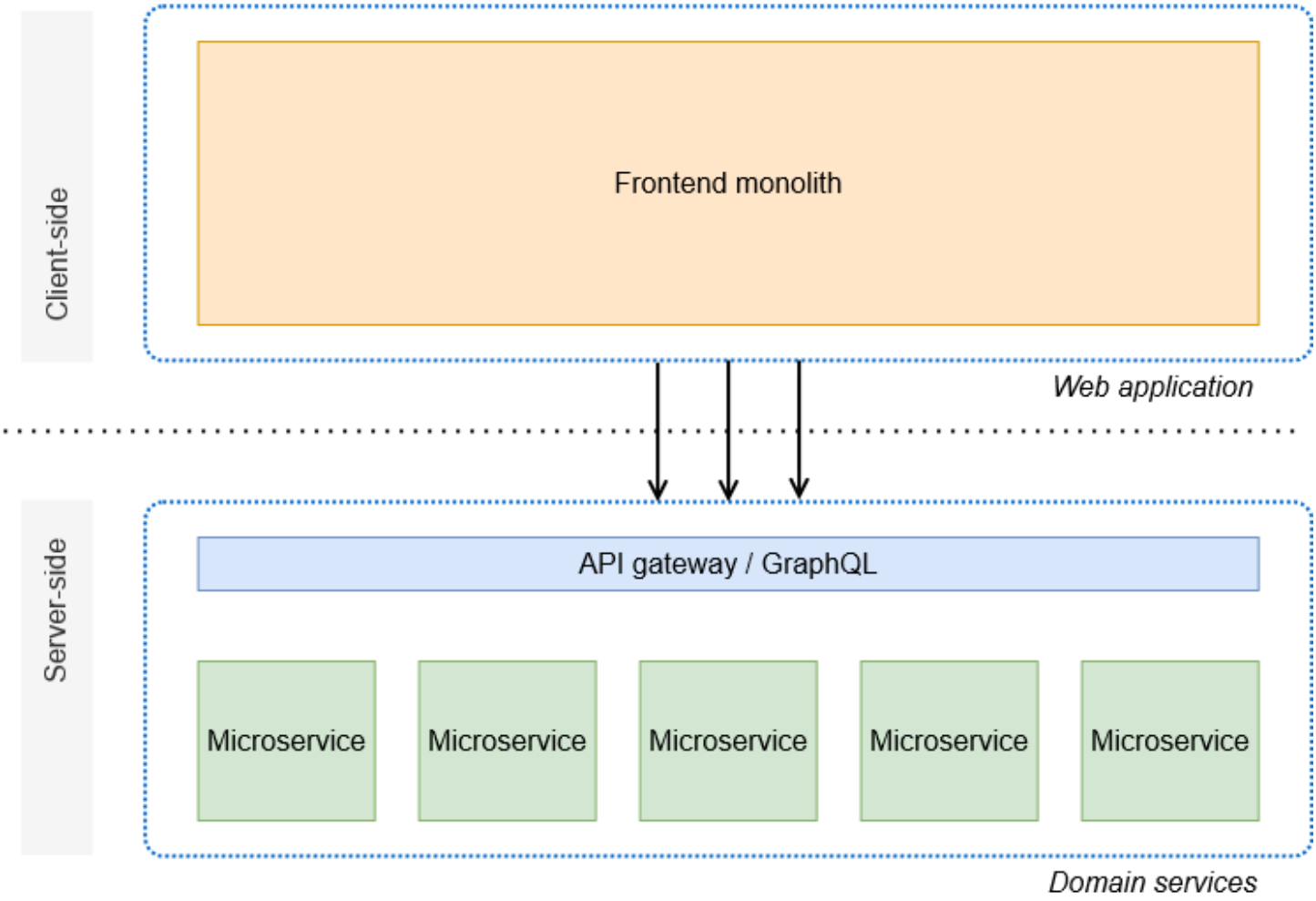
Visão geral do

Os microfront-ends são uma arquitetura construída com base na decomposição de front-ends de aplicativos em artefatos desenvolvidos e implantados de forma independente. Ao dividir front-ends grandes em artefatos de software autônomos, você pode encapsular a lógica de negócios e reduzir as dependências. Isso permite a entrega mais rápida e frequente de incrementos de produtos.

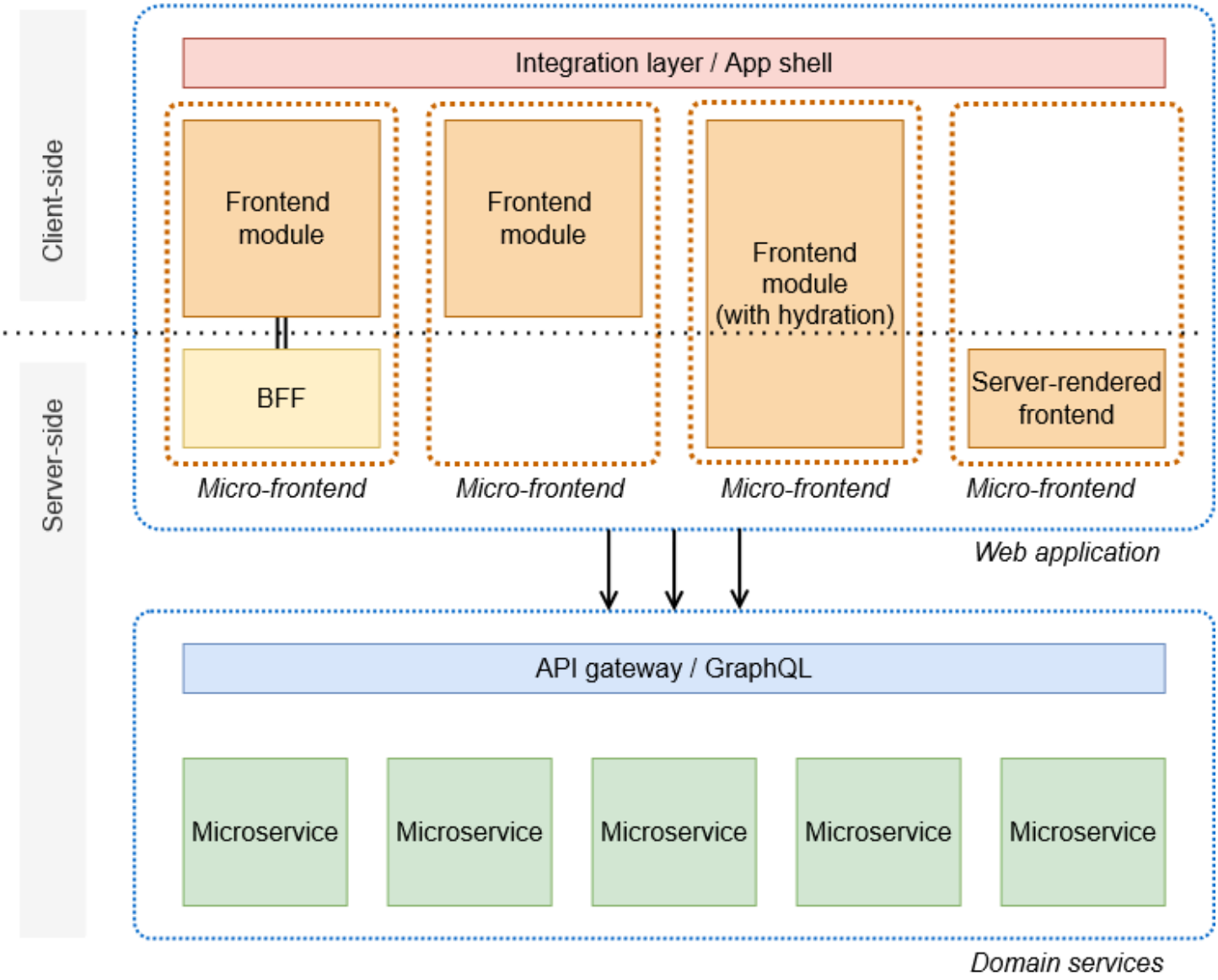
Os microfront-ends são semelhantes aos microsserviços. Na verdade, o termo microfrontend é derivado do termo microsserviço e tem como objetivo transmitir a noção de microsserviço como frontend. Enquanto uma arquitetura de microsserviços normalmente combina um sistema distribuído no back-end com um front-end monolítico, os microfront-ends são serviços de front-end distribuídos independentes. Esses serviços podem ser configurados de duas maneiras:

- Somente front-end, integrando-se a uma camada de API compartilhada por trás da qual é executada uma arquitetura de microsserviços
- Full-stack, o que significa que cada microfrontend tem sua própria implementação de back-end.

O diagrama a seguir mostra uma arquitetura tradicional de microsserviços, com um monólito de front-end que usa um gateway de API para se conectar aos microsserviços de back-end.



O diagrama a seguir mostra uma arquitetura de microfront-end com diferentes implementações de microsserviços.



Conforme mostrado no diagrama anterior, você pode usar microfrontends com arquiteturas de renderização do lado do cliente ou do lado do servidor:

- Os microfront-ends renderizados do lado do cliente podem ser consumidos diretamente por um API Gateway APIs centralizado.
- A equipe pode criar um backend-for-frontend (BFF) dentro do contexto limitado para reduzir a conversação do frontend em relação ao. APIs
- No lado do servidor, os microfrontends podem ser expressos com uma abordagem do lado do servidor aumentada no lado do cliente usando uma técnica chamada hidratação. Quando uma página é renderizada pelo navegador, o associado JavaScript é hidratado para permitir interações com elementos da interface do usuário, como clicar em um botão.

- Os microfront-ends podem renderizar no back-end e usar hiperlinks para direcionar para uma nova parte de um site.

Os microfront-ends são ideais para organizações que desejam fazer o seguinte:

- Expanda com várias equipes trabalhando no mesmo projeto.
- Adote a descentralização da tomada de decisões, capacitando os desenvolvedores a inovar dentro dos limites identificados dos sistemas.

Essa abordagem reduz significativamente a carga cognitiva das equipes, porque elas se tornam responsáveis por partes específicas do sistema. Isso aumenta a agilidade dos negócios porque modificações podem ser feitas em uma parte do sistema sem interromper o resto.

Os microfront-ends são uma abordagem arquitetônica distinta. Embora existam maneiras diferentes de criar microfront-ends, todas elas têm características comuns:

- Uma arquitetura de microfrontend é composta por vários elementos independentes. A estrutura é semelhante à modularização que acontece com microsserviços no back-end.
- Um microfrontend é totalmente responsável pela implementação do front-end dentro de seu contexto limitado, que compreende o seguinte:
  - Interface do usuário
  - Dados
  - Estado ou sessão
  - Lógica de negócios
  - Fluxo

Um contexto limitado é um sistema internamente consistente com limites cuidadosamente projetados que medeiam o que pode entrar e sair. Um microfrontend deve compartilhar o mínimo possível de lógica de negócios e dados com outros microfront-ends. Onde quer que o compartilhamento precise acontecer, ele ocorre por meio de interfaces claramente definidas, como eventos personalizados ou fluxos reativos. No entanto, quando se trata de algumas questões transversais, como um sistema de design ou bibliotecas de registro, o compartilhamento intencional é bem-vindo.

Um padrão recomendado é criar microfrontends usando equipes multifuncionais. Isso significa que cada microfrontend é desenvolvido pela mesma equipe que trabalha do back-end ao frontend.

A propriedade da equipe é crucial, desde a codificação até a operacionalização do sistema na produção.

Esta orientação não pretende recomendar uma abordagem específica. Em vez disso, ele discute diferentes padrões, melhores práticas, compensações e considerações arquitetônicas e organizacionais.

Conceitos fundamentais

A arquitetura de microfront-end é fortemente inspirada em três conceitos arquitetônicos anteriores:

- O design orientado por domínio é o modelo mental para estruturar aplicativos complexos em domínios coerentes.
- Os sistemas distribuídos são uma abordagem para criar aplicativos como subsistemas fracamente acoplados que são desenvolvidos de forma independente e executados em sua própria infraestrutura dedicada.
- A computação em nuvem é uma abordagem para executar a infraestrutura de TI como serviços com um pay-as-you-go modelo.

Design orientado por domínio

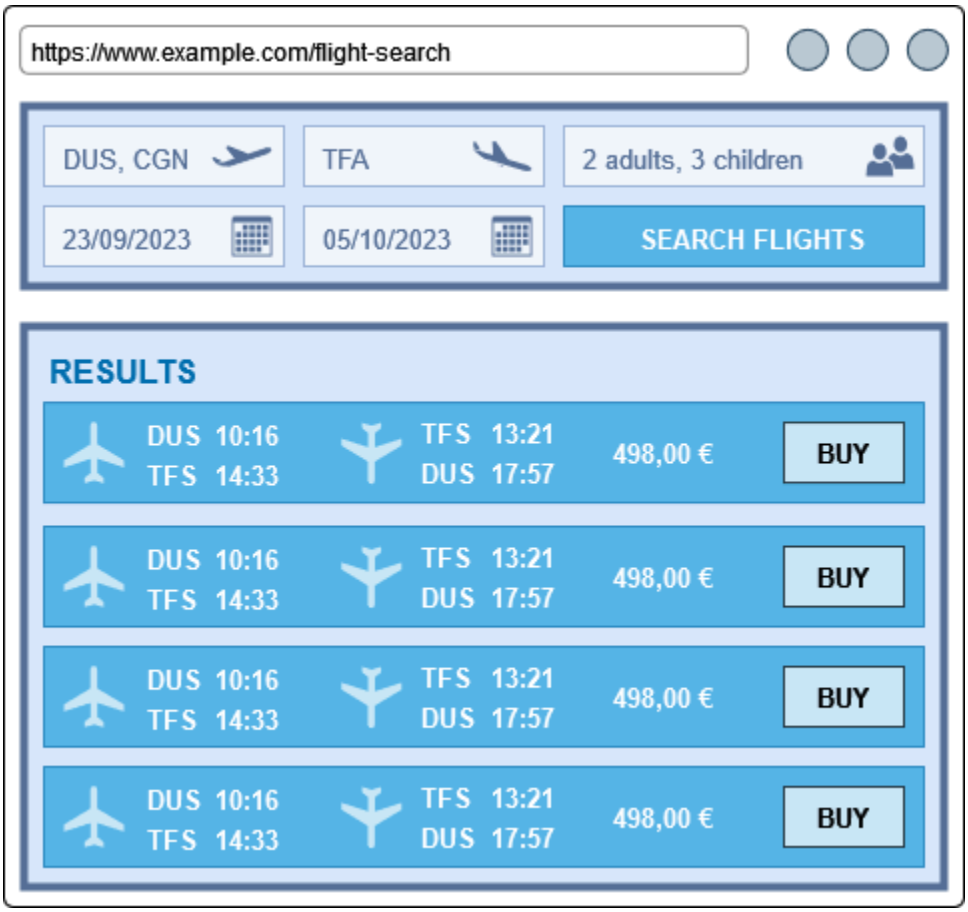
O design orientado por domínio (DDD) é um paradigma desenvolvido por Eric Evans. Em seu livro de 2003, Domain-Driven Design: Tackling Complexity in the Heart of Software, Evans postula que o desenvolvimento de software deve ser orientado por questões comerciais e não técnicas. Evans propõe que os projetos de TI desenvolvam primeiro uma linguagem onipresente que ajude especialistas técnicos e de domínio a encontrar um entendimento compartilhado. Com base nessa linguagem, eles podem formular um modelo mutuamente compreendido da realidade empresarial.

Por mais óbvia que seja essa abordagem, muitos projetos de software sofrem com a desconexão entre os negócios e a TI. Essas desconexões geralmente causam mal-entendidos significativos, o que leva a excessos orçamentários, diminuição da qualidade ou falha do projeto.

Evans introduz vários outros termos importantes, um dos quais é o contexto limitado. Um contexto limitado é um segmento independente de um grande aplicativo de TI que contém a solução ou implementação para exatamente uma preocupação comercial. Um aplicativo grande consistirá em vários contextos limitados que são vagamente acoplados por meio de padrões de integração. Esses contextos limitados podem até ter seus próprios dialetos da linguagem onipresente. Por exemplo, um usuário no contexto de pagamento de um aplicativo pode ter aspectos diferentes de um usuário no contexto de entrega porque a noção de frete seria irrelevante durante o pagamento.

Evans não define o quão pequeno ou grande um contexto limitado deve ser. O tamanho é determinado pelo projeto de software e pode evoluir com o tempo. Bons indicadores dos limites de um contexto são o grau de coesão entre as entidades (objetos de domínio) e a lógica de negócios.

No contexto de microfrontends, o design orientado por domínio pode ser ilustrado pelo exemplo de uma página da web complexa, como uma página de reserva de voos.



Nesta página, os principais blocos de construção são um formulário de pesquisa, um painel de filtros e a lista de resultados. Para identificar os limites, você deve identificar contextos funcionais independentes. Além disso, considere aspectos não funcionais, como reutilização, desempenho e segurança. O indicador mais importante de que “coisas que pertencem umas às outras” são seus padrões de comunicação. Se alguns elementos em uma arquitetura precisam se comunicar com frequência e trocar informações complexas, eles provavelmente compartilham o mesmo contexto limitado.

Elementos individuais da interface do usuário, como botões, não são contextos limitados, porque não são funcionalmente independentes. Além disso, a página inteira não é adequada para um contexto limitado, pois pode ser dividida em contextos independentes menores. Uma abordagem razoável é tratar o formulário de pesquisa como um contexto limitado e tratar a lista de resultados como o segundo contexto limitado. Cada um desses dois contextos limitados agora pode ser implementado como um microfrontend separado.

Sistemas distribuídos

Para facilitar a manutenção e apoiar a capacidade de evolução, a maioria das soluções de TI não triviais são modulares. Nesse caso, modular significa que os sistemas de TI consistem em blocos de construção identificáveis que são desacoplados por meio de interfaces para obter a separação das preocupações.

Além de serem modulares, os sistemas distribuídos devem ser sistemas independentes por si só. Em um sistema meramente modular, cada módulo é idealmente encapsulado e expõe suas funções por meio de interfaces, mas não pode ser implantado de forma independente ou mesmo funcionar sozinho. Além disso, os módulos geralmente seguem o mesmo ciclo de vida de outros módulos que fazem parte do mesmo sistema. Os blocos de construção de um sistema distribuído, por outro lado, têm seus próprios ciclos de vida. Aplicando o paradigma de design orientado por domínio, cada componente básico aborda um domínio ou subdomínio comercial e vive em seu próprio contexto limitado.

Quando sistemas distribuídos interagem durante o tempo de construção, uma abordagem comum é desenvolver mecanismos para identificar problemas rapidamente. Por exemplo, você pode adotar linguagens digitadas e investir pesadamente em testes unitários. Várias equipes podem colaborar no desenvolvimento e na manutenção de módulos, geralmente distribuídos como bibliotecas para sistemas consumirem com ferramentas como npm, Apache Maven e pip. NuGet

Durante o tempo de execução, os sistemas distribuídos que interagem normalmente são de propriedade de equipes individuais. O consumo de dependências causa complexidade operacional devido ao tratamento de erros, ao balanceamento de desempenho e à segurança. Investimentos em testes de integração e observabilidade são fundamentais para reduzir riscos.

Atualmente, os exemplos mais populares de sistemas distribuídos são os microsserviços. Nas arquiteturas de microsserviços, os serviços de back-end são orientados por domínio (e não por questões técnicas, como interface de usuário ou autenticação) e de propriedade de equipes autônomas. Os microfront-ends compartilham os mesmos princípios, estendendo o escopo da solução ao frontend.

Computação em nuvem

A computação em nuvem é uma forma de comprar infraestrutura de TI como serviços com um pay-as-you-go modelo, em vez de construir seus próprios data centers e comprar hardware para operá-los localmente. A computação em nuvem oferece várias vantagens:

- Sua organização ganha agilidade comercial significativa ao ser capaz de experimentar novas tecnologias sem ter que assumir compromissos financeiros grandes e de longo prazo antecipadamente.
- Ao usar um provedor de nuvem como AWS, sua organização pode acessar um amplo portfólio de serviços altamente integráveis e de baixa manutenção (como gateways de API, bancos de dados, orquestração de contêineres e recursos de nuvem). O acesso a esses serviços libera sua equipe para se concentrar no trabalho que diferencia sua organização da concorrência.
- Quando sua organização estiver pronta para implantar uma solução globalmente, você poderá implantar a solução na infraestrutura de nuvem em todo o mundo.

A computação em nuvem oferece suporte a microfront-ends fornecendo uma infraestrutura altamente gerenciada. Isso facilita end-to-end a propriedade de equipes multifuncionais. Embora a equipe deva ter um forte conhecimento operacional, as tarefas manuais de provisionamento da infraestrutura, atualizações do sistema operacional e rede seriam uma distração.

Como os microfront-ends vivem em contextos limitados, as equipes podem escolher o serviço mais adequado para executá-los. Por exemplo, as equipes podem escolher entre funções de nuvem e contêineres para computação, e podem escolher entre diferentes tipos de bancos de dados SQL e NoSQL ou caches na memória. As equipes podem até mesmo criar seus microfront-ends em um kit de ferramentas altamente integrado AWS Amplify, como o, que vem com blocos de construção pré-configurados para infraestrutura sem servidor.

Comparando microfront-ends com arquiteturas alternativas

Como acontece com todas as estratégias de arquitetura, a decisão de adotar microfront-ends deve ser baseada em critérios de avaliação orientados pelos princípios da sua organização. Os microfront-ends têm vantagens e desvantagens. Se sua organização decidir usar microfrontends, você deve ter estratégias implementadas para enfrentar os desafios dos sistemas distribuídos

Ao escolher uma arquitetura de aplicativo, as alternativas mais populares aos microfront-ends são monólitos, aplicativos de n camadas e microsserviços em combinação com um front-end de aplicativo de página única (SPA). Todas essas abordagens são válidas e cada uma delas tem vantagens e desvantagens.

Monólitos

Um aplicativo pequeno que não precisa de mudanças frequentes pode ser entregue rapidamente como um monólito. Mesmo em situações em que se espera um crescimento significativo, um monólito é um primeiro passo natural. Posteriormente, o monólito pode ser retirado ou refatorado em uma estrutura mais flexível. Ao começar com um monólito, sua organização pode entrar no mercado, obter feedback dos clientes e melhorar o produto mais rapidamente.

No entanto, os aplicativos monolíticos tendem a se degradar se não forem mantidos com cuidado ou à medida que a base de código aumenta de tamanho com o tempo. Quando várias equipes contribuem significativamente para a mesma base de código, raramente todas contribuem para sua manutenção e operações. Isso resulta em um desequilíbrio de responsabilidades, o que afeta a velocidade e causa ineficiências. Ao mesmo tempo, o acoplamento inadvertido entre os módulos de um monólito leva a efeitos colaterais indesejados à medida que a base de código evolui. Esses efeitos colaterais podem resultar em avarias e interrupções.

Aplicativos de nível N

Um aplicativo mais complexo que tenha um ritmo de evolução relativamente estático pode ser construído como uma arquitetura de três camadas (apresentação, aplicativo, dados), com uma camada REST ou GraphQL entre o front-end e o back-end. Isso é muito mais flexível, e equipes de diferentes níveis podem se desenvolver de forma independente até certo ponto. A desvantagem de um aplicativo de n camadas é que é muito mais difícil implantar a funcionalidade. O front-end e o back-end são separados por meio de um contrato de API, portanto, as alterações importantes devem ser implantadas em conjunto ou a API deve ser versionada.

Considere o seguinte cenário comum: se o lançamento de um novo recurso exigir uma alteração no esquema de dados, pode levar dias para que os proprietários do produto cheguem a um acordo sobre um conjunto de funcionalidades com uma equipe de front-end. Em seguida, a equipe de front-end solicitará que a equipe de back-end desenvolva e libere a funcionalidade de sua parte. A equipe de back-end trabalhará com os proprietários dos dados para lançar uma atualização do esquema do banco de dados. Em seguida, a equipe de back-end lançará uma nova versão da API, para que a equipe de front-end possa desenvolver e lançar suas alterações. Nesse cenário, a propagação de todas as mudanças na produção pode levar semanas ou até meses, porque cada equipe tem sua própria lista de pendências, prioridades e mecanismos para desenvolver, testar e lançar mudanças.

Microsserviços

Em uma arquitetura de microsserviços, o back-end é decomposto em pequenos serviços, cada um abordando uma preocupação comercial específica dentro de um contexto limitado. Cada microsserviço também é fortemente desacoplado de outros serviços ao expor um contrato de interface claramente definido.

Vale ressaltar que contextos limitados e contratos de interface também devem existir em monólitos bem elaborados e arquiteturas de n camadas. Em uma arquitetura de microsserviços, no entanto, a comunicação acontece pela rede, geralmente pelo protocolo HTTP, e os serviços têm uma infraestrutura de tempo de execução dedicada. Isso dá suporte ao desenvolvimento, entrega e operação independentes de cada serviço de back-end.

Escolhendo a abordagem para suas necessidades

Monólitos e arquiteturas de n camadas agrupam várias preocupações de domínio em um único artefato técnico. Isso facilita o gerenciamento de aspectos como dependências e fluxo interno de dados, mas dificulta a entrega de novas funcionalidades. Para manter uma base de código coerente, uma equipe geralmente investe tempo em refatoração e desacoplamento devido à grande base de código que precisa lidar.

Os aplicativos desenvolvidos por algumas equipes podem não precisar da complexidade adicional que vem com a mudança para microfront-ends. Isso é especialmente verdadeiro se as equipes não estiverem pagando as penalidades de alto acoplamento e longos prazos de entrega para liberar as mudanças.

Em resumo, arquiteturas mais complexas e distribuídas geralmente são a escolha certa para aplicativos complexos e de rápida evolução. Para aplicativos de pequeno a médio porte, uma

arquitetura distribuída não é necessariamente superior a uma monolítica, especialmente se o aplicativo não evoluir drasticamente em um curto período de tempo.

Decisões arquitetônicas em microfrontends

As equipes que aplicam um padrão de arquitetura de microfront-end para seus aplicativos devem tomar várias decisões sobre arquitetura desde o início:

- Identificação de microfront-ends e definição de limites
- Composição de páginas e visualizações com microfront-ends
- Roteamento, gerenciamento de estado e comunicação entre microfront-ends
- Gerenciando dependências para questões transversais

As seções a seguir abordam esses tópicos com mais profundidade.

Ao tomar decisões de arquitetura, é essencial ter as métricas corretas e entender os padrões de uso, as características do aplicativo e as vantagens e desvantagens. Por exemplo, um site de comércio eletrônico tem características e padrões de uso diferentes em comparação com uma ferramenta de edição de vídeo ou painéis de observabilidade.

Aplicativos voltados ao público com alto tráfego e curta profundidade de sessão podem ser otimizados para métricas de carregamento inicial da página, como Time to Interactive (TTI) e First Contentful Paint (FCP). Por outro lado, um aplicativo no qual os usuários fazem login no início do dia e com o qual continuam interagindo durante todo o dia pode ser otimizado para a experiência no aplicativo. A equipe de aplicativos pode otimizar a métrica First Input Delay (FID) após cada navegação, em vez do carregamento inicial da página.

Os sites públicos devem atender a vários ambientes de navegador. Aplicativos corporativos com restrições conhecidas no ambiente do cliente podem otimizar sua composição de microfront-end de acordo com suas restrições.

Não há uma única escolha certa para as decisões de arquitetura. Entenda as vantagens e desvantagens, o contexto em que a empresa opera, os padrões de uso e as métricas para orientar as decisões adequadas para cada aplicativo individual.

Identificação dos limites do microfrontend

Para melhorar a autonomia da equipe, os recursos de negócios fornecidos por um aplicativo podem ser decompostos em vários microfront-ends com dependências mínimas entre si.

Seguindo a metodologia DDD discutida anteriormente, as equipes podem dividir um domínio de aplicativo em subdomínios de negócios e contextos limitados. As equipes autônomas podem então possuir a funcionalidade de seus contextos limitados e fornecer esses contextos como microfront-ends.

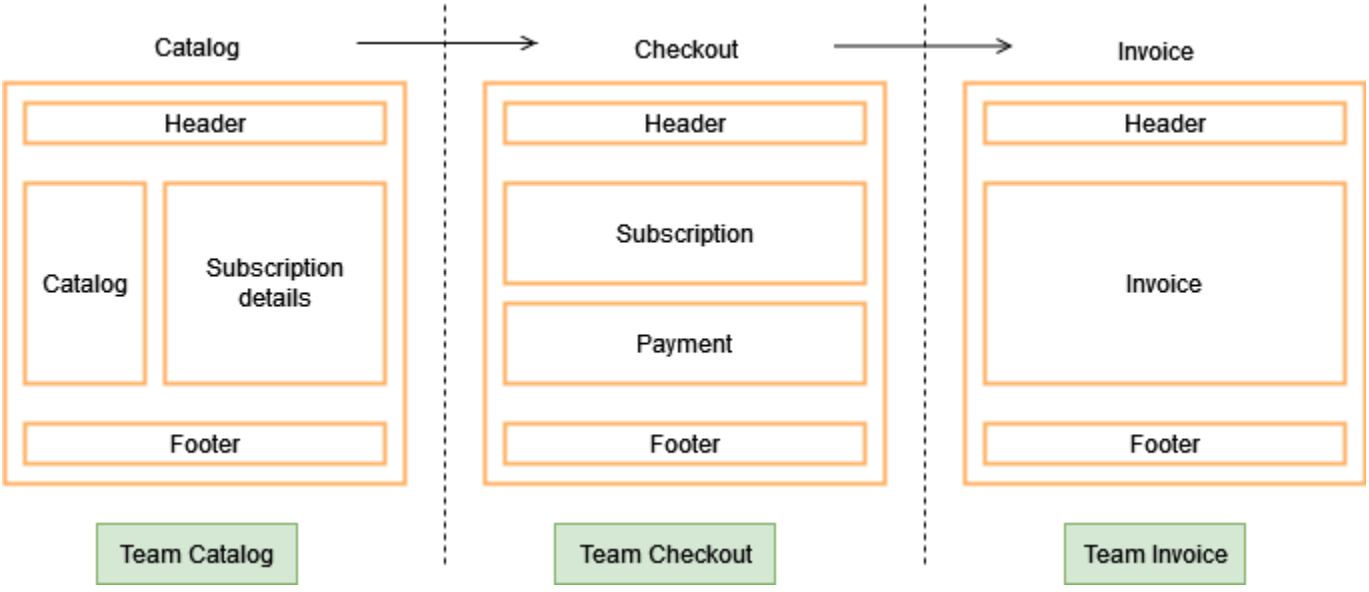
Um contexto limitado bem definido deve minimizar a sobreposição funcional e a necessidade de comunicação em tempo de execução entre contextos. A comunicação necessária pode ser implementada com métodos orientados a eventos. Isso não é diferente da arquitetura orientada a eventos para desenvolvimento de microsserviços.

Um aplicativo bem arquitetado também deve oferecer suporte à entrega de futuras extensões por novas equipes para fornecer uma experiência consistente aos clientes.

Como dividir um aplicativo monolítico em microfront-ends

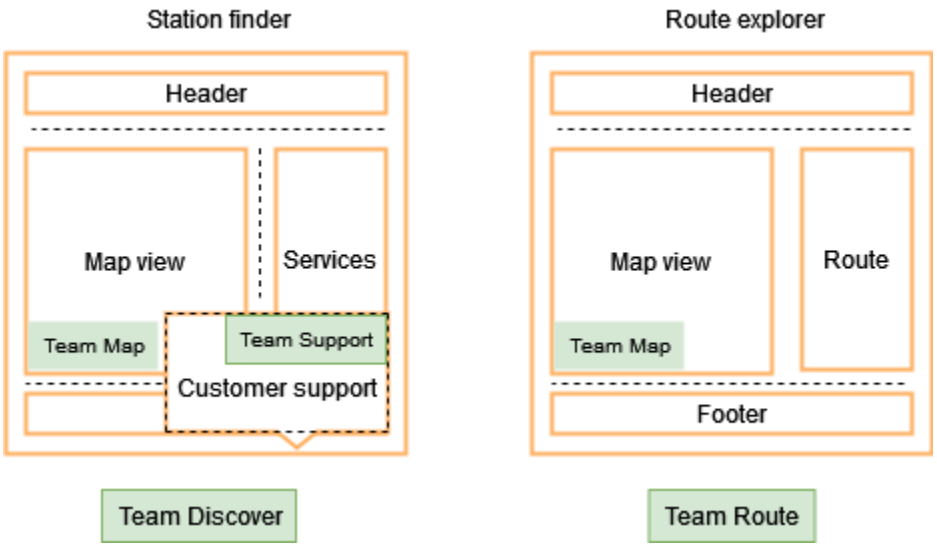
A seção Visão geral incluiu um exemplo de identificação de contextos funcionais independentes em uma página da web. Vários padrões para dividir a funcionalidade na interface do usuário surgem.

Por exemplo, quando os domínios de negócios formam estágios de uma jornada do usuário, uma divisão vertical no frontend pode ser aplicada, na qual uma coleção de visualizações da jornada do usuário é fornecida como microfront-ends. O diagrama a seguir mostra uma divisão vertical, em que as etapas de catálogo, finalização de compra e fatura são fornecidas por equipes separadas como microfront-ends separados.



Para algumas aplicações, a divisão vertical sozinha pode não ser suficiente. Por exemplo, algumas funcionalidades podem precisar ser fornecidas em várias exibições. Para esses aplicativos, você

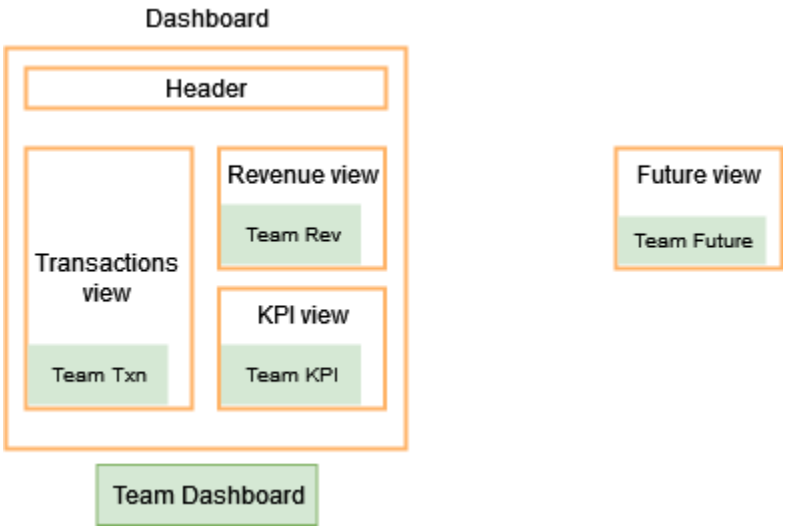
pode aplicar uma divisão mista. O diagrama a seguir mostra uma solução dividida mista na qual os micro-front-ends do Station Finder e do Route Explorer usam a funcionalidade de visualização de mapa.



Aplicativos do tipo portal ou painel geralmente reúnem recursos de front-end em uma única visualização. Nesses tipos de aplicativos, cada widget pode ser fornecido como um microfrontend, e o aplicativo de hospedagem define as restrições e interfaces que os microfront-ends devem implementar.

Essa abordagem fornece um mecanismo para que os microfront-ends lidem com questões como tamanho da janela de visualização, provedores de autenticação, definições de configuração e metadados. Esses tipos de aplicativos otimizam a extensibilidade. Novos recursos podem ser desenvolvidos por novas equipes para escalar os recursos do painel.

O diagrama a seguir mostra um aplicativo de painel desenvolvido por três equipes individuais que fazem parte do Team Dashboard.



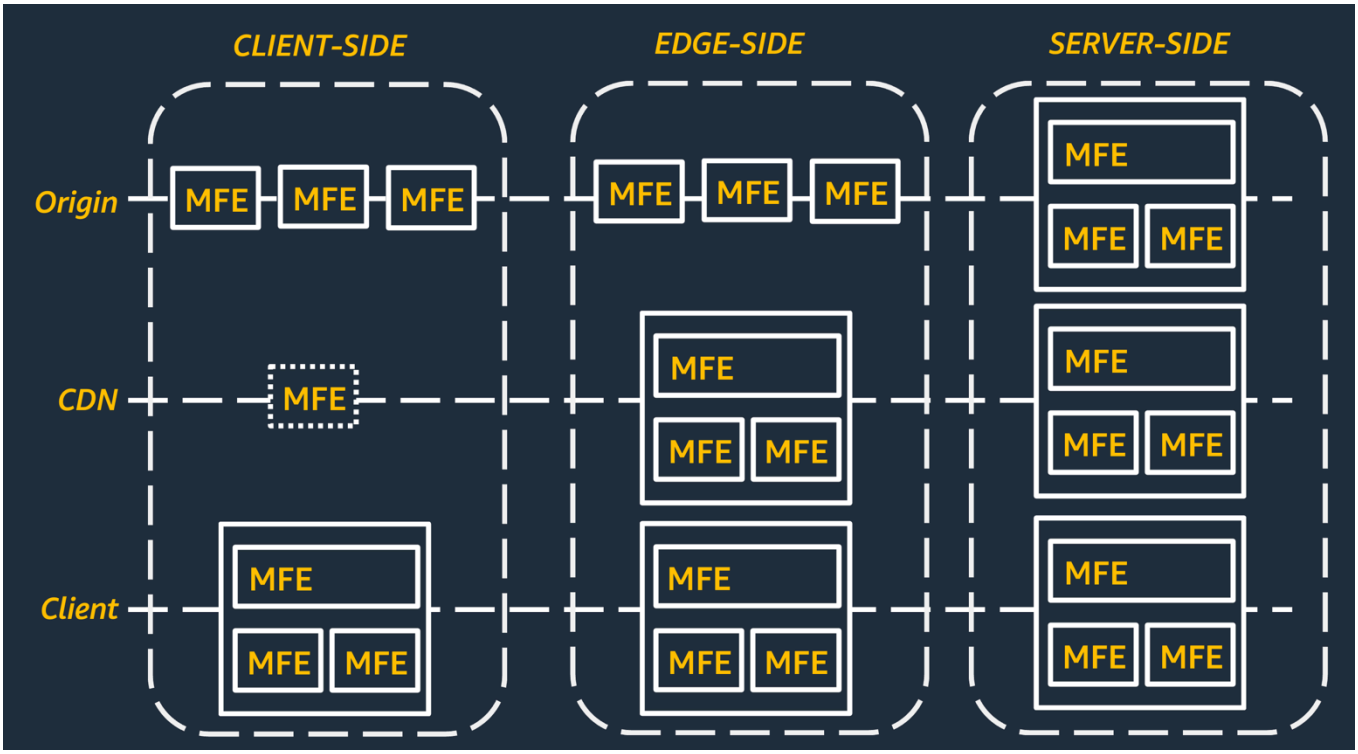
No diagrama, a visão futura representa novos recursos desenvolvidos por novas equipes para escalar o Team Dashboard e os recursos do painel.

Os aplicativos de portal e painel geralmente compõem a funcionalidade usando uma divisão mista na interface do usuário. Os microfront-ends são configuráveis com configurações bem definidas, incluindo restrições de posição e tamanho.

Composição de páginas e visualizações com microfront-ends

Você pode compor visualizações de um aplicativo com composição do lado do cliente, composição do lado da borda e composição do lado do servidor. Os padrões de composição têm características diferentes em termos de habilidades de equipe necessárias, tolerância a falhas, desempenho e comportamento do cache.

O diagrama a seguir mostra como a composição acontece nas camadas do lado do cliente, do lado da borda e do lado do servidor de uma arquitetura de microfrontend.



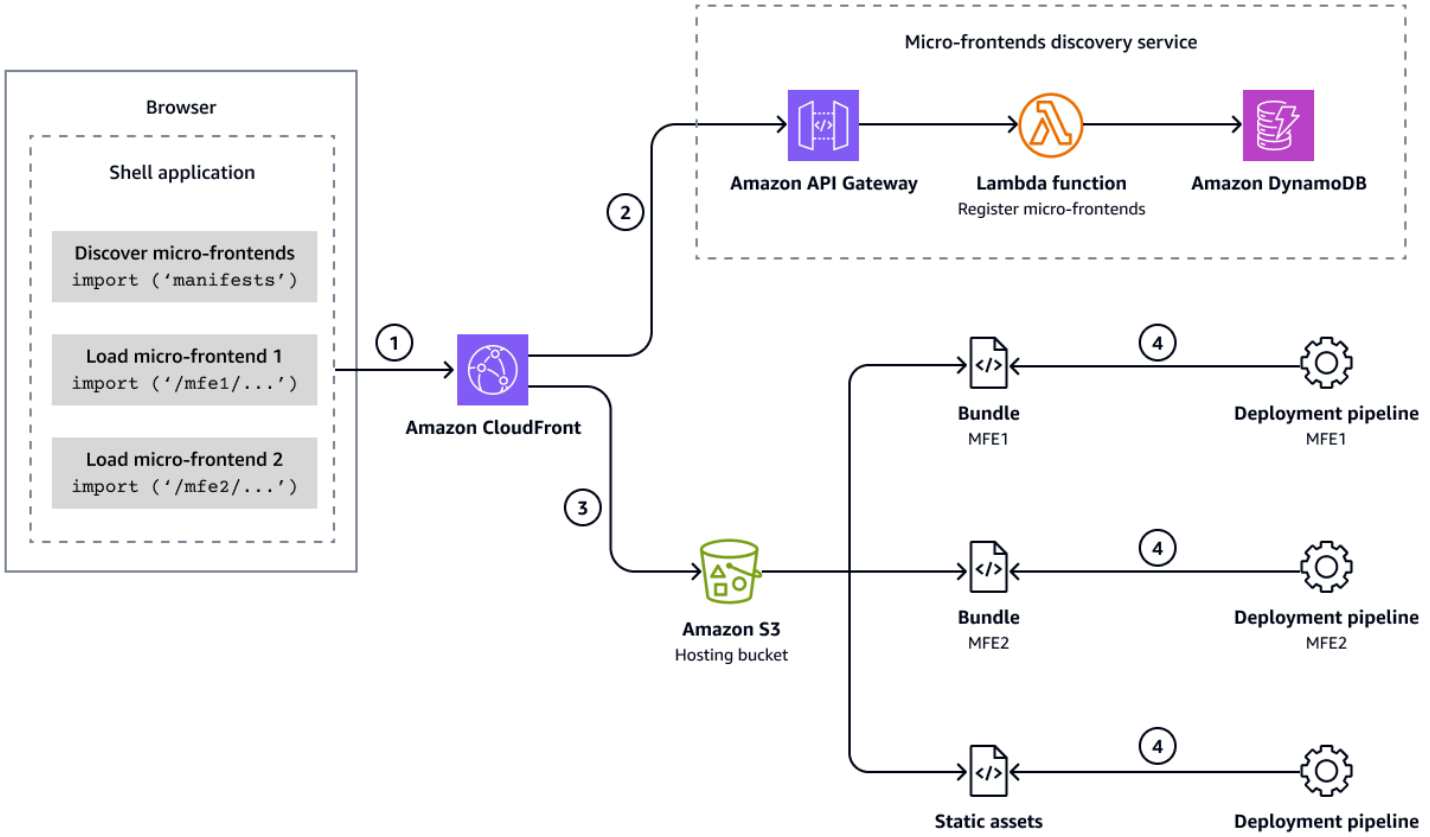
As camadas do lado do cliente, do lado da borda e do lado do servidor são discutidas nas seções a seguir.

Composição do cliente

Carregue e anexe dinamicamente microfront-ends como fragmentos do Document Object Model (DOM) no cliente (navegador ou visualização da web móvel). Os artefatos do microfront-end, como JavaScript arquivos CSS, podem ser carregados das redes de entrega de conteúdo (CDNs) para reduzir a latência. A composição do lado do cliente requer o seguinte:

- Uma equipe para possuir e manter um aplicativo shell ou uma estrutura de microfront-end para permitir a descoberta, o carregamento e a renderização de componentes de microfront-end em tempo de execução no navegador
- Altos níveis de habilidade em tecnologias de front-end, como HTML, CSS e JavaScript compreensão aprofundada dos ambientes de navegador
- Otimização da quantidade de JavaScript carga em uma página e disciplina para evitar conflitos de namespaces globais

O diagrama a seguir mostra um exemplo de AWS arquitetura para composição do lado do cliente sem servidor.



A composição do lado do cliente acontece no ambiente do navegador por meio de um aplicativo shell. O diagrama mostra os seguintes detalhes:

1. Depois que o aplicativo shell é carregado, ele faz uma solicitação inicial à Amazon CloudFront para descobrir os microfront-ends a serem carregados por meio de um endpoint de manifesto.
2. Os manifestos contêm informações sobre cada microfrontend (por exemplo, nome, URL, versão e comportamento alternativo). Os manifestos são atendidos pelo serviço de descoberta de microfront-ends. No diagrama, esse serviço de descoberta é representado pelo Amazon API Gateway, uma AWS Lambda função e pelo Amazon DynamoDB. O aplicativo shell usa as informações do manifesto para solicitar microfront-ends individuais para compor a página dentro de um determinado layout.
3. Cada pacote de microfront-end é composto por arquivos estáticos (como JavaScript CSS e HTML). Os arquivos são hospedados em um bucket do Amazon Simple Storage Service (Amazon S3) e servidos por meio dele. CloudFront
4. As equipes podem implantar novas versões de seus microfront-ends e atualizar as informações do manifesto usando pipelines de implantação de sua propriedade.

Composição lateral

Use técnicas de transclusão, como Edge Side Includes (ESI) ou Server Side Includes (SSI) suportadas por alguns CDNs e proxies na frente dos servidores de origem para compor uma página antes de enviá-la pela rede aos clientes. O ESI exige o seguinte:

- Uma CDN com capacidade ESI ou uma implantação de proxy na frente dos microfront-ends do lado do servidor. Implementações de proxy HAProxy, como Varnish e NGINX, oferecem suporte a SSI.
- Uma compreensão do uso e das limitações das implementações de ESI e SSI.

As equipes que iniciam novos aplicativos normalmente não escolhem a composição lateral para seu padrão de composição. No entanto, esse padrão pode fornecer um caminho para aplicativos legados que dependem da transclusão.

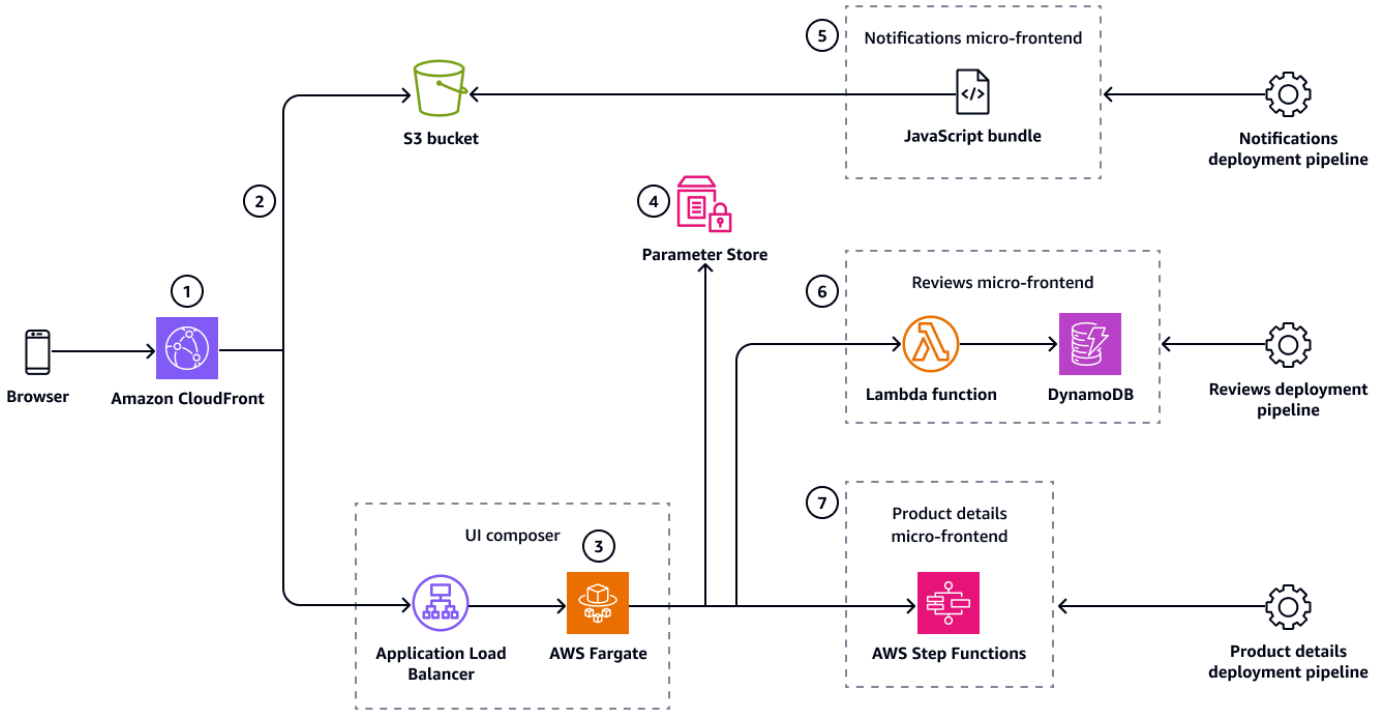
Composição do servidor

Use servidores de origem para compor páginas antes que elas sejam armazenadas em cache na borda. Isso pode ser feito com tecnologias tradicionais, como PHP, Jakarta Server Pages (JSP) ou bibliotecas de modelos, para compor as páginas incluindo fragmentos de microfrontends. Você também pode usar JavaScript estruturas, como Next.js, em execução no servidor para compor páginas no servidor com renderização do lado do servidor (SSR).

Depois que as páginas são renderizadas no servidor, elas podem ser armazenadas em cache CDNs para reduzir a latência. Quando novas versões de microfront-ends são implantadas, as páginas devem ser renderizadas novamente e o cache deve ser atualizado para fornecer as versões mais recentes aos clientes.

A composição do lado do servidor requer uma compreensão profunda do ambiente do servidor para estabelecer padrões de implantação, descoberta de microfront-ends do lado do servidor e gerenciamento de cache.

O diagrama a seguir mostra a composição do lado do servidor.



O diagrama inclui os seguintes componentes e processos:

1. A Amazon CloudFront fornece um ponto de entrada exclusivo para o aplicativo. A distribuição tem duas origens: a primeira para arquivos estáticos e a segunda para o compositor de interface do usuário.
2. Os arquivos estáticos são hospedados em um bucket do Amazon S3. Eles são consumidos pelo navegador e pelo compositor de interface do usuário para modelos HTML.
3. O compositor de interface do usuário é executado em um cluster de contêineres em AWS Fargate. Com uma solução em contêineres, você pode usar recursos de streaming e renderização multiencadeada, se necessário.
4. O Parameter Store, um recurso do AWS Systems Manager, é usado como um sistema básico de descoberta de microfront-ends. Esse recurso fornece um armazenamento de valores-chave usado pelo compositor de interface do usuário para recuperar os endpoints de microfront-end a serem consumidos.
5. O microfrontend de notificações armazena o JavaScript pacote otimizado no bucket do S3. Isso é renderizado no cliente porque ele deve reagir às interações do usuário.
6. O microfrontend de avaliações é composto por uma função Lambda, e as avaliações dos usuários são armazenadas no DynamoDB. O microfrontend de avaliações é totalmente renderizado no lado do servidor e gera um fragmento HTML.

7. O microfrontend de detalhes do produto é um micro-frontend de baixo código que usa. AWS Step Functions O Express Workflow pode ser chamado de forma síncrona e contém a lógica para renderizar o fragmento HTML e uma camada de cache.

Para obter mais informações sobre a composição do lado do servidor, consulte a postagem do blog Microfrontends de renderização do lado do servidor — a arquitetura.

Roteamento e comunicação entre microfront-ends

As opções de roteamento dependem da abordagem de composição. A comunicação pode ser otimizada reduzindo o acoplamento entre os componentes front-end.

Roteamento

Os aplicativos que usam composição do lado do cliente com divisão vertical podem usar roteamento do lado do servidor (aplicativo de várias páginas) ou roteamento do lado do cliente (aplicativo de página única). Se eles usarem uma divisão mista para a composição da interface do usuário, o roteamento do lado do cliente é necessário para oferecer suporte a hierarquias de roteamento mais profundas de microfront-ends em uma página.

Os aplicativos que usam composição do lado da borda e composição do lado do servidor se alinham melhor com o roteamento do lado do servidor ou com a computação de borda, como o Lambda @Edge com a Amazon. CloudFront

Comunicação entre microfront-ends

Com arquiteturas de microfront-end, recomendamos reduzir o acoplamento entre os componentes de front-end. Uma abordagem para reduzir o acoplamento é deixar de fazer chamadas de função síncronas para mensagens assíncronas.

Os tempos de execução do navegador e as interações do usuário são assíncronos por natureza. Os eventos podem ser trocados entre produtores e consumidores por meio de mensagens. Os eventos fornecem uma interface bem definida para comunicação entre microfront-ends.

Se você seguir as práticas do DDD para identificar seus contextos limitados para microfront-ends, a próxima etapa é identificar eventos que devem ser comunicados além dos limites.

O mecanismo de mensagens para eventos pode ser eventos DOM nativos (`CustomEvents`), emissores de JavaScript eventos ou bibliotecas de fluxo reativo fornecidas pelas equipes da

plataforma. Os microfront-ends publicam eventos e se inscrevem em eventos relevantes para seu contexto limitado. Com esse método, os editores e os assinantes não precisam se conhecer. O contrato é a definição do evento. Para uma representação visual disso, consulte a seção Comunique-se com eventos do diagrama Contexto limitado com arquiteturas de eventos.

Gerenciando dependências para questões transversais

O gerenciamento consciente de dependências é crucial para o sucesso de uma arquitetura distribuída, como microfront-ends. O gerenciamento de dependências é uma das partes mais desafiadoras do desenvolvimento de microfront-ends.

Em uma arquitetura de microfront-end, dois aspectos importantes do gerenciamento de dependências são a penalidade de desempenho decorrente da transferência de grandes artefatos de código para o cliente e a sobrecarga nos recursos computacionais. Idealmente, sua organização precisa determinar como as dependências em uma arquitetura de front-end distribuída são mantidas.

Três estratégias viáveis para exigir a manutenção de dependências são não compartilhar nada, usando padrões da web, como mapas de importação e federação de módulos. Outras abordagens são antipadrões porque violam os princípios básicos das arquiteturas distribuídas.

Não compartilhe nada, sempre que possível

A abordagem de não compartilhar nada postula que nenhuma dependência entre artefatos de software independentes deve ser compartilhada, ou pelo menos não na integração ou no tempo de execução. Isso significa que, se dois microfront-ends dependerem da mesma biblioteca, cada um deverá ser incorporado na biblioteca no momento da construção e enviá-la separadamente. Além disso, cada microfrontend deve validar que a biblioteca não polui namespaces globais e recursos compartilhados.

Isso leva a redundâncias, mas é uma troca consciente com a máxima agilidade. Sem dependências de tempo de execução compartilhadas, as equipes têm a máxima flexibilidade para desenvolver o software da maneira que acharem útil, desde que façam isso no escopo de sua solução e não quebrem nenhum contrato de interface.

Em uma plataforma em que os microfrontends seguem o princípio de não compartilhar nada, é importante manter os microfront-ends o mais leves possível. Isso requer desenvolvedores qualificados e diligentes na otimização de seus microfront-ends para desempenho e que não sacrifiquem a experiência do usuário pela experiência do desenvolvedor.

Quando você compartilha o código

Ao tomar a decisão de compartilhar algum código, você pode compartilhá-lo como bibliotecas ou módulos de tempo de execução. Por exemplo, a equipe principal de front-end fornece bibliotecas para consumo de microfront-end por meio de. CDNs As equipes de valor comercial podem carregar as bibliotecas em tempo de execução ou podem usar repositórios de pacotes para publicar suas bibliotecas. As equipes de microfrontend podem desenvolver com base em uma versão específica da biblioteca empacotada no momento da construção, de forma semelhante aos aplicativos móveis que usam estruturas híbridas.

Uma terceira opção é usar um registro de pacotes privado para oferecer suporte à integração em tempo de construção de bibliotecas comuns. Isso reduz o risco de que uma alteração significativa no contrato da biblioteca inicie erros em tempo de execução. No entanto, essa abordagem mais conservadora exige mais governança para sincronizar todos os microfront-ends com as versões mais recentes da biblioteca.

Para melhorar os tempos de carregamento da página, os microfrontends podem externalizar as dependências da biblioteca a serem carregadas a partir de partes em cache de uma CDN como a Amazon. CloudFront

Para gerenciar dependências de tempo de execução, os microfrontends podem usar mapas de importação (ou bibliotecas como`System.js`) para especificar de onde cada módulo é carregado em tempo de execução. O webpack Module Federation é outra abordagem para apontar para uma versão hospedada de um módulo remoto e resolver dependências comuns em microfrontends independentes.

Outra abordagem é facilitar o carregamento dinâmico de mapas de importação com uma solicitação inicial para um endpoint de descoberta.

Estado compartilhado

Para reduzir o acoplamento de microfront-ends, é importante evitar um gerenciamento de estado global acessível a partir de todos os microfront-ends na mesma visualização, semelhante às arquiteturas monolíticas. Por exemplo, ter uma loja Redux global acessível a partir de todos os microfront-ends aumenta o acoplamento.

Um padrão para eliminar o estado compartilhado é encapsulá-lo em microfront-ends e comunicar-se com mensagens assíncronas, conforme discutido anteriormente.

Quando for absolutamente necessário, introduza interfaces bem definidas para o estado global e opte pelo compartilhamento somente leitura para evitar comportamentos inesperados:

- Quando há uma divisão vertical, você pode usar componentes de URL e armazenamento do navegador para acessar informações do ambiente do host.
- Quando você tem uma divisão mista, também pode usar os eventos ou JavaScript bibliotecas personalizados padrão DOM, como emissores de eventos ou fluxos bidirecionais, para passar informações para microfrontends.

Se você precisar compartilhar várias informações entre microfront-ends, recomendamos revisitar os limites do microfrontend. A necessidade de compartilhar pode ser causada pela evolução dos negócios ou por um design inicial abaixo da média.

Também é possível usar sessões do lado do servidor, em que cada microfrontend busca os dados necessários usando um identificador de sessão. Para reduzir o acoplamento, é importante eliminar o estado compartilhado e manter separados os dados específicos da sessão do microfrontend.

Estruturas e ferramentas

Não faltam estruturas de front-end, como Angular e Next.js, mas a maioria delas não é criada pensando em microfront-ends. Portanto, às vezes faltam mecanismos para enfrentar os desafios da arquitetura de microfrontend.

Considerações gerais sobre o enquadramento

Este guia não tem como objetivo recomendar ou comparar estruturas individuais. Como vários microfront-ends geralmente são executados na mesma página do aplicativo web, o carregamento e o desempenho do tempo de execução são as principais preocupações. É importante escolher uma estrutura que apresente o mínimo de sobrecarga possível.

As estruturas são divididas com base na camada de renderização:

- Renderização do lado do cliente (CSR)
- Renderização do lado do servidor (SSR)

As arquiteturas de front-end incluem outros recursos, como geração estática de sites (SSG). No entanto, o SSG é executado apenas uma vez. Os microfront-ends são compostos principalmente em tempo de execução, então CSR e SSR são as principais opções.

Renderização do lado do cliente

Para CSR, há duas opções populares:

- Estrutura única de SPA
- Federação de módulos

O Single SPA é uma opção leve para compor micro-front-ends. Ele resolve os desafios mais comuns em arquiteturas de microfrontend, como compor vários microfront-ends na mesma página e evitar conflitos de dependências.

O Module Federation começou como um plug-in, oferecido pelo webpack 5, e resolve a grande maioria dos desafios em arquiteturas de microfront-end, incluindo gerenciamento de dependências em diferentes artefatos. O Module Federation 2.0 funciona nativamente com Rspack, webpack, esbuild e agora com. JavaScript

Considere não usar nenhuma estrutura. Os navegadores modernos, com uma participação de mercado geral de 98%, de acordo com caniuse.com, oferecem recursos como elementos personalizados de forma nativa e são adequados para um aplicativo de microfront-ends. Quando necessário, combine elementos personalizados com bibliotecas leves para propagação de eventos, internacionalização ou outras preocupações específicas.

Renderização do lado do servidor

No lado do SSR, as duas opções principais são mais complicadas:

- Adote uma estrutura existente, como o Next.js, e aplique um princípio de microfront-ends que usa a Federação de Módulos.
- Use HTML- over-the-wire para trocar fragmentos HTML que representam microfront-ends e compor esses fragmentos dentro de um modelo em tempo de execução. Um exemplo dessa abordagem é o Podium.

Integração de API – Backend para front-end

O padrão Backends for Frontends (BFF) é normalmente usado em ambientes de microsserviços. No contexto de microfrontends, um BFF é um serviço do lado do servidor que pertence a um microfrontend. Nem todos os microfront-ends precisam ter um melhor amigo. No entanto, se você estiver usando um BFF, ele deverá ser executado dentro do mesmo contexto limitado e não ser compartilhado em outros contextos limitados.

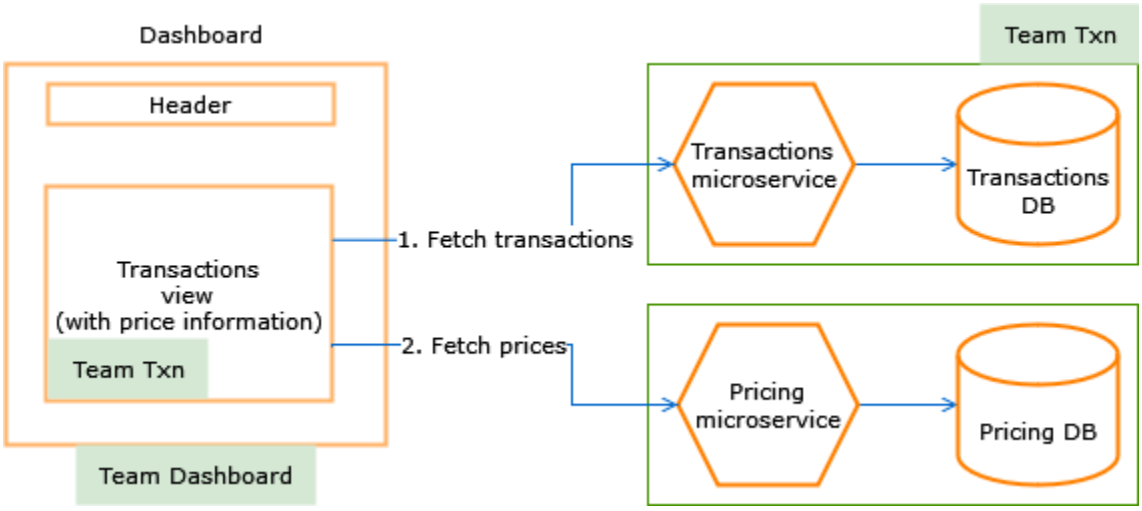
Ao contrário de um serviço tradicional, um BFF não segue um modelo de domínio. Em vez disso, é uma camada de API para o microfrontend pré-processar os dados antes que eles cheguem ao cliente. As áreas em que isso é útil incluem o seguinte:

- Autorização para uso privado APIs
- Agregação de dados de diferentes fontes
- Transformação de dados para reduzir a carga da rede e facilitar o consumo de dados pelo cliente

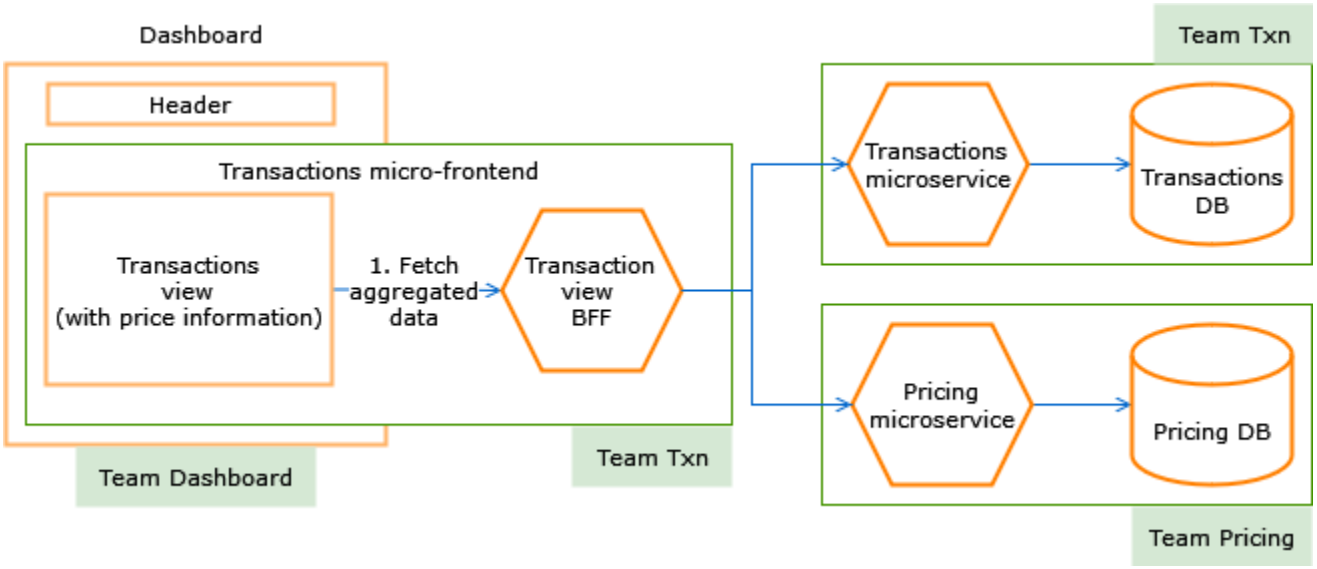
Dessa forma, um BFF é de propriedade do microfrontend, não do nível de serviço do domínio. BFFs pode ser implantado usando o seguinte:

- AWS AppSync GraphQL APIs
- Um conjunto de funções do AWS Lambda
- Como um contêiner em execução no Amazon ECS, Amazon EKS ou AWS AppRunner

O diagrama a seguir mostra que, sem o padrão BFF, os microfront-ends devem se conectar a endpoints individuais da API de microsserviço para buscar e agregar dados.



Em vez disso, com o padrão BFF no diagrama a seguir, os microfront-ends podem se comunicar com seu próprio back-end e buscar dados agregados.



As equipes podem desenvolver BFFs para diferentes canais, como dispositivos móveis, web ou visualizações específicas, com requisitos para otimizar as interações de back-end reduzindo o bate-papo.

Estilo e CSS

Cascading Style Sheets (CSS) é uma linguagem para determinar centralmente a apresentação de um documento em vez da formatação codificada para texto e objetos. O recurso em cascata da linguagem foi projetado para controlar as prioridades entre os estilos usando a herança. Quando você trabalha em microfront-ends e cria uma estratégia para gerenciar dependências, o recurso em cascata da linguagem pode ser um desafio.

Por exemplo, dois microfrontends coexistem na mesma página, cada um definindo seu próprio estilo para o elemento HTML. `body` Se cada um buscar seu próprio arquivo CSS e anexá-lo ao DOM usando uma `style` tag, o arquivo CSS substituirá o primeiro se ambos tiverem definição para elementos HTML, nomes de classes ou elementos comuns. IDs Existem diferentes estratégias para lidar com esses problemas, dependendo da estratégia de dependência que você escolher para gerenciar estilos.

Atualmente, a abordagem mais popular para equilibrar desempenho, consistência e experiência do desenvolvedor consiste em desenvolver e manter um sistema de design.

Sistemas de design – Uma abordagem de compartilhar algo

Essa abordagem usa um sistema para compartilhar estilos quando apropriado, ao mesmo tempo em que oferece suporte a divergências ocasionais para equilibrar consistência, desempenho e experiência do desenvolvedor. Um sistema de design é uma coleção de componentes reutilizáveis, guiados por padrões claros. O desenvolvimento do sistema de design geralmente é conduzido por uma equipe com contribuições e contribuições de várias equipes. Em termos práticos, um sistema de design é uma forma de compartilhar elementos de baixo nível que podem ser exportados como uma JavaScript biblioteca. Os desenvolvedores de microfront-end podem usar a biblioteca como uma dependência para criar interfaces simples, compondo recursos pré-fabricados disponíveis e como ponto de partida para criar novas interfaces.

Considere o exemplo de um microfrontend que precisa de um formulário. A experiência típica do desenvolvedor consiste em usar componentes pré-fabricados disponíveis no sistema de design para compor caixas de texto, botões, listas suspensas e outros elementos da interface do usuário. O desenvolvedor não precisa escrever nenhum estilo para os componentes reais, apenas para ver como eles se parecem juntos. O sistema a ser construído e lançado pode usar o webpack Module Federation ou uma abordagem semelhante para declarar o sistema de design como uma

dependência externa, de forma que a lógica do formulário seja empacotada sem incluir o sistema de design.

Vários microfront-ends podem então fazer o mesmo para cuidar de preocupações compartilhadas. Quando as equipes desenvolvem novos componentes que podem ser compartilhados entre vários microfront-ends, esses componentes são adicionados ao sistema de design após atingirem a maturidade.

A principal vantagem da abordagem do sistema de design é o alto nível de consistência. Embora os microfront-ends possam escrever estilos e, ocasionalmente, substituir os do sistema de design, há muito pouca necessidade disso. Os principais elementos de baixo nível não mudam com frequência e oferecem funcionalidades básicas que podem ser estendidas por padrão. Outra vantagem é o desempenho. Com uma boa estratégia para criar e lançar, você pode produzir pacotes compartilhados mínimos que são instrumentados pelo shell do aplicativo. Você pode melhorar ainda mais quando vários pacotes específicos de microfront-end são carregados de forma assíncrona sob demanda, com espaço mínimo em termos de largura de banda de rede. Por último, mas não menos importante, a experiência do desenvolvedor é ideal porque as pessoas podem se concentrar na criação de interfaces avançadas sem reinventar a roda (como escrever JavaScript e CSS toda vez que um botão precisa ser adicionado a uma página).

A desvantagem é que um sistema de design de qualquer tipo é uma dependência, portanto, ele deve ser mantido e, às vezes, atualizado. Se vários microfront-ends exigirem uma nova versão de uma dependência compartilhada, você poderá usar uma das seguintes opções:

- Um mecanismo de orquestração que pode, ocasionalmente, buscar várias versões dessa dependência compartilhada sem conflitos
- Uma estratégia compartilhada para fazer com que todos os dependentes usem uma nova versão

Por exemplo, se todos os microfrontends dependerem da versão 3.0 de um sistema de design e houver uma nova versão chamada 3.1 para ser usada de forma compartilhada, você poderá implementar sinalizadores de recursos para que todos os microfrontends migrem com risco mínimo. Para obter mais informações, consulte a seção Sinalizadores de recursos. Outra desvantagem potencial é que os sistemas de design geralmente abordam mais do que estilo. Eles também incluem JavaScript práticas e ferramentas. Esses aspectos exigem chegar a um consenso por meio de debate e colaboração.

Implementar um sistema de design é um bom investimento a longo prazo. É uma abordagem popular e deve ser considerada por qualquer pessoa que trabalhe em uma arquitetura de front-

end complexa. Normalmente, exige que engenheiros de front-end e equipes de produto e design colaborem e definam mecanismos para interagir uns com os outros. É importante agendar um horário para alcançar o estado desejado. Também é importante ter o patrocínio da liderança para que as pessoas possam criar algo confiável, bem mantido e com bom desempenho a longo prazo.

CSS totalmente encapsulado – Uma abordagem de não compartilhar nada

Cada microfrontend usa convenções e ferramentas para superar o recurso de cascata do CSS. Um exemplo é garantir que o estilo de cada elemento esteja sempre associado a um nome de classe em vez do ID do elemento, e que os nomes das classes sejam sempre exclusivos. Dessa forma, tudo é direcionado para microfront-ends individuais e o risco de conflitos indesejados é minimizado. O shell do aplicativo normalmente é responsável por carregar os estilos dos micro-front-ends depois que eles são carregados no DOM, embora algumas ferramentas agrupem os estilos usando. JavaScript

A principal vantagem de não compartilhar nada é o risco reduzido de introduzir conflitos entre microfront-ends. Outra vantagem é a experiência do desenvolvedor. Cada microfrontend não compartilha nada com outros microfrontends. Liberar e testar isoladamente é mais simples e rápido.

A principal desvantagem da abordagem de não compartilhar nada é a potencial falta de consistência. Não há nenhum sistema para avaliar a consistência. Mesmo que a meta seja duplicar o que é compartilhado, isso se torna um desafio equilibrar a velocidade de lançamento e a colaboração. Uma mitigação comum é criar ferramentas para medir a consistência. Por exemplo, você pode criar um sistema para fazer capturas de tela automatizadas de vários microfront-ends renderizados em uma página com um navegador sem cabeçalho. Em seguida, você pode revisar manualmente as capturas de tela antes do lançamento. No entanto, isso requer disciplina e governança. Para obter mais informações, consulte a seção Equilibrando autonomia com alinhamento.

Dependendo do caso de uso, outra desvantagem potencial é o desempenho. Se uma grande quantidade de estilo for usada por todos os microfront-ends, o cliente deverá baixar muitos códigos duplicados. Isso afetará negativamente a experiência do usuário.

Essa abordagem de não compartilhar nada deve ser considerada somente para arquiteturas de microfront-end que envolvem apenas algumas equipes ou microfront-ends que podem tolerar baixa consistência. Também pode ser uma etapa inicial natural enquanto uma organização está trabalhando em um sistema de design.

CSS global compartilhado – Uma abordagem de compartilhamento total

Com essa abordagem, todo o código relacionado ao estilo é armazenado em um repositório central onde os colaboradores escrevem CSS para todos os microfrontends trabalhando em arquivos CSS ou usando pré-processadores como o Sass. Quando as alterações são feitas, um sistema de compilação cria um único pacote CSS que pode ser hospedado em uma CDN e incluído em cada microfrontend pelo shell do aplicativo. Os desenvolvedores de microfront-end podem projetar e criar seus aplicativos executando seu código por meio de um shell de aplicativo hospedado localmente.

Além da vantagem óbvia de reduzir o risco de conflitos entre microfront-ends, as vantagens dessa abordagem são consistência e desempenho. No entanto, desacoplar estilos da marcação e da lógica torna mais difícil para os desenvolvedores entenderem como os estilos são usados, como podem evoluir e como podem ser descontinuados. Por exemplo, pode ser mais rápido introduzir um novo nome de classe do que aprender sobre a classe existente e as consequências da edição de suas propriedades. As desvantagens de criar um novo nome de classe são o crescimento do tamanho do pacote, que afeta o desempenho, e a possível introdução de inconsistências na experiência do usuário.

Embora um CSS global compartilhado possa ser o ponto de partida de uma monolith-to-micro-frontends migração, raramente é benéfico para arquiteturas de microfront-end que envolvem mais de uma ou duas equipes colaborando juntas. Recomendamos investir em um sistema de design o mais rápido possível e implementar uma abordagem de não compartilhar nada enquanto o sistema de design estiver em desenvolvimento.

Organização e formas de trabalhar

Como acontece com todas as estratégias de arquitetura, os microfrontends têm implicações muito além da tecnologia que uma organização escolhe implementar. A decisão de criar aplicativos de microfront-end deve estar alinhada aos negócios, produtos, organização, operações e até mesmo à cultura (por exemplo, capacitando equipes e descentralizando a tomada de decisões). Em troca, esse tipo de arquitetura de microfrontend oferece suporte ao desenvolvimento verdadeiramente ágil e orientado ao produto, pois reduz significativamente a sobrecarga de comunicação entre equipes independentes.

Desenvolvimento ágil

A ideia de desenvolvimento ágil de software se tornou tão universal nos últimos anos que praticamente todas as organizações afirmam trabalhar com agilidade. Embora uma definição conclusiva de ágil esteja além do escopo dessa estratégia, vale a pena revisar os principais elementos que são relevantes para o desenvolvimento de microfrontend.

A base do paradigma ágil é o Manifesto Ágil (2001), que postulou quatro princípios principais (por exemplo, “Indivíduos e interações acima de processos e ferramentas”) e doze princípios. Estruturas de processo como o Scrum e o Scaled Agile Framework (SAFe) surgiram em torno do Manifesto Ágil e chegaram às práticas diárias. No entanto, a filosofia por trás deles é amplamente mal compreendida ou ignorada.

No contexto da arquitetura de microfrontend, é importante adotar os seguintes princípios ágeis:

- “Forneça software funcional com frequência, de algumas semanas a alguns meses, com preferência por prazos mais curtos.”

  Esse princípio enfatiza a importância de trabalhar em incrementos e entregar software para produção com a maior regularidade e frequência possível. Do ponto de vista tecnológico, isso se refere à integração e entrega contínuas (CI/CD). In CI/CDas ferramentas e os processos de construção, teste e implantação são partes integrantes de cada projeto de software. O princípio também implica que a infraestrutura de tempo de execução e as responsabilidades operacionais devem pertencer à equipe. Essa propriedade é especialmente importante em sistemas distribuídos, nos quais subsistemas independentes podem ter requisitos significativamente diferentes de infraestrutura e operações.

- “Crie projetos em torno de indivíduos motivados. Ofereça a eles o ambiente e o suporte de que precisam e confie neles para realizar o trabalho.”

  “As melhores arquiteturas, requisitos e projetos surgem de equipes auto-organizadas.”

  Ambos os princípios enfatizam os benefícios da propriedade, independência e end-to-end responsabilidade. Uma arquitetura de microfrontend será bem-sucedida quando (e somente quando) as equipes realmente possuírem seus microfront-ends. End-to-enda responsabilidade desde a concepção, passando pelo design e implementação, até a entrega e operação, garante que a equipe possa realmente exercer a propriedade. Essa independência é necessária, tanto técnica quanto organizacionalmente, para que a equipe tenha autonomia sobre a direção estratégica. Não recomendamos o uso de uma plataforma de microfrontend em uma organização centralizada que usa um modelo de desenvolvimento em cascata.

Composição e tamanho da equipe

Para que uma equipe de software exerça a propriedade, ela deve governar a si mesma, incluindo como e o que a equipe entrega, dentro dos limites impostos pela organização.

Para serem eficazes, as equipes devem ser capazes de fornecer software de forma independente e ter autoridade para decidir a melhor maneira de entregá-lo. Uma equipe que recebe requisitos de recursos de um gerente de produto externo ou projetos de interface de usuário de um designer externo, sem estar envolvida no planejamento desses itens, não pode ser considerada autônoma. Os recursos podem violar contratos ou funcionalidades existentes. Essas violações exigirão mais discussões e negociações, com o risco de atrasar a entrega e introduzir conflitos desnecessários entre as equipes.

Ao mesmo tempo, as equipes não devem se tornar muito grandes. Embora uma equipe maior tenha mais recursos e possa acomodar ausências individuais, a complexidade da comunicação cresce exponencialmente com cada novo membro. Não é possível declarar um tamanho máximo de equipe universalmente válido. O número de pessoas necessárias para um projeto depende de fatores como maturidade da equipe, complexidade técnica, ritmo de inovação e infraestrutura. Por exemplo, a Amazon segue a regra das duas pizzas: uma equipe grande demais para ser alimentada com duas pizzas deve ser dividida em equipes menores. Isso pode ser um desafio. A divisão deve ocorrer de acordo com limites naturais e deve dar a cada equipe autonomia e propriedade sobre seu trabalho.

DevOps cultura

DevOps refere-se a uma prática de engenharia de software em que as etapas do ciclo de vida de desenvolvimento são fortemente integradas do ponto de vista organizacional e técnico. Ao contrário da crença popular, DevOps tem muito a ver com cultura e mentalidade, e muito pouco com papéis e ferramentas.

Tradicionalmente, uma organização de software teria equipes de especialistas, como para design, implementação, teste, implantação e operações. Sempre que uma equipe concluía seu trabalho, ela entregava o projeto para a próxima equipe. No entanto, a entrega de software por meio de silos de equipes especializadas causa atrito durante as entregas. Ao mesmo tempo, quando especialistas são forçados a trabalhar com um foco estreito, eles carecem de conhecimento em domínios vizinhos e não têm uma visão sistêmica do produto. Esses déficits podem levar à baixa coerência do produto de software.

Por exemplo, quando um arquiteto de software projeta uma solução que deve ser implementada por alguém de uma equipe diferente, ele pode ignorar aspectos inerentes da implementação (como uma incompatibilidade de dependências). Os desenvolvedores então usam atalhos (como um monkey patch) ou uma formalização back-and-forth é iniciada entre o arquiteto e a equipe de desenvolvimento. Devido à sobrecarga de gerenciar esses processos, o desenvolvimento não é mais ágil (no sentido de flexível, adaptativo, incremental e informal).

Embora o termo se refira DevOps principalmente à cultura, ele implica as tecnologias e processos que DevOps possibilitam na prática. DevOps está intimamente relacionado aCI/CD. When a developer has finished implementing an increment of software, they commit it to a version-control system such as Git. Traditionally, a build system would then build and integrate the software, and have it tested and released in a more or less unified and centralized process. With CI/CD: a construção, a integração, o teste e o lançamento do software são inerentes e automatizados. Idealmente, o processo faz parte do próprio projeto de software por meio de arquivos de configuração personalizados especificamente para o projeto em questão.

O maior número possível de etapas é automatizado. Por exemplo, as práticas de testes manuais devem ser reduzidas, porque quase todos os tipos de testes podem ser automatizados. Quando o projeto é configurado dessa forma, as atualizações de um produto de software podem ser entregues várias vezes ao dia com alta confiança. Outra tecnologia que oferece suporte DevOps é a infraestrutura como código (IaC).

Tradicionalmente, configurar e manter a infraestrutura de TI exigiria o trabalho manual de instalação e manutenção de hardware (configuração de cabos e servidores em um data center) e software

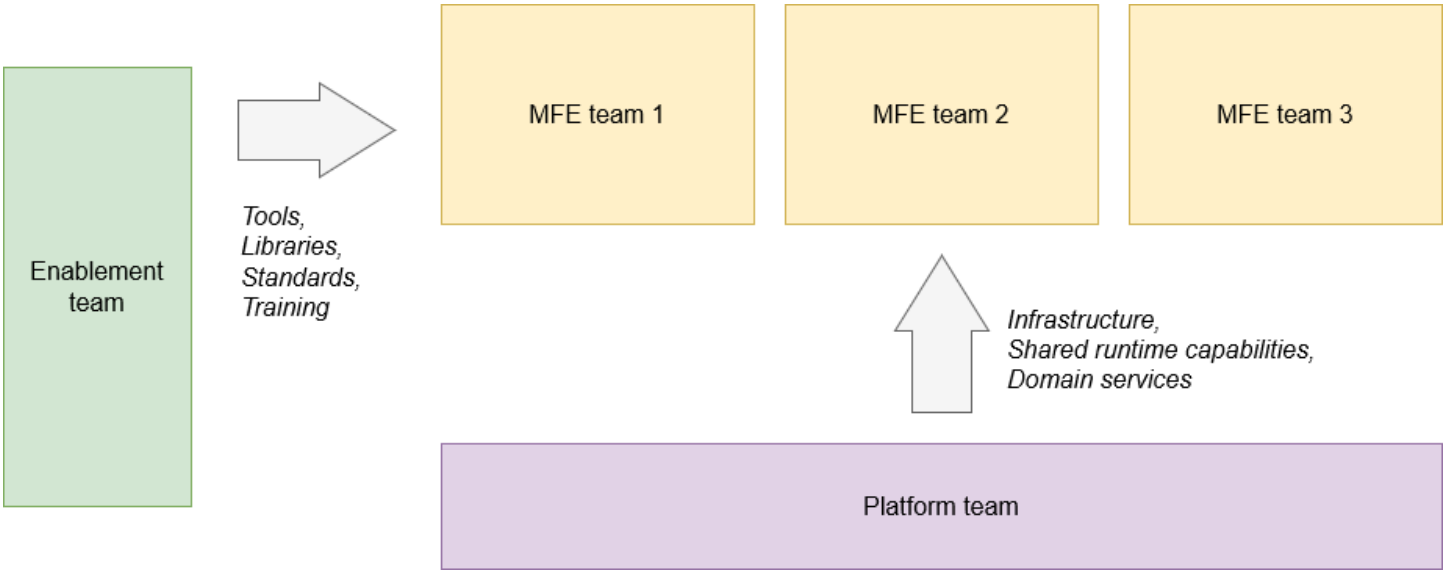
operacional. Isso era necessário, mas tinha vários inconvenientes. A configuração era demorada e propensa a erros. O hardware geralmente era superprovisionado ou subprovisionado, resultando em gastos excessivos ou diminuição do desempenho. Ao usar o IaC, você pode descrever os requisitos de infraestrutura de um sistema de TI por meio de um arquivo de configuração a partir do qual os serviços em nuvem podem ser implantados e atualizados automaticamente.

O que tudo isso tem a ver com microfront-ends? DevOps, CI/CD e IaC são complementos ideais para uma arquitetura de microfrontend. Os benefícios dos microfront-ends dependem de processos de entrega rápidos e sem atrito. Uma DevOps cultura só pode prosperar em ambientes em que as equipes possuem projetos de software com end-to-end responsabilidade.

Orquestrando o desenvolvimento de microfront-end em várias equipes

Ao escalar o desenvolvimento de microfront-end em várias equipes multifuncionais, surgem dois problemas: primeiro, as equipes começam a desenvolver sua própria interpretação do paradigma, fazer escolhas de estrutura e biblioteca e criar suas próprias ferramentas e bibliotecas auxiliares. Em segundo lugar, equipes totalmente autônomas devem assumir a responsabilidade por recursos genéricos, como gerenciamento de infraestrutura de baixo nível. Portanto, faz sentido introduzir duas equipes adicionais em uma organização de micro-frontend com várias equipes: uma equipe de capacitação e uma equipe de plataforma. Esses conceitos são amplamente adotados em organizações de TI modernas com sistemas distribuídos e estão bem documentados em Team Topologies.

O diagrama a seguir mostra a equipe de capacitação fornecendo ferramentas, bibliotecas, padrões e testes para três equipes de microfrontend. A equipe da plataforma fornece infraestrutura, recursos de tempo de execução compartilhados e serviços de domínio para essas mesmas três equipes de microfront-end.



A equipe da plataforma apoia as equipes de microfront-end, libertando-as do trabalho pesado indiferenciado. Esse suporte pode incluir serviços de infraestrutura, como tempos de execução de contêineres, CI/CD pipelines, ferramentas de colaboração e monitoramento. No entanto, a criação de uma equipe de plataforma não deve levar a uma organização na qual o desenvolvimento esteja separado das operações. O oposto é verdadeiro: a equipe da plataforma oferece um produto de engenharia, e as equipes de microfrontend têm a propriedade e a responsabilidade pelo tempo de execução de seus serviços na plataforma.

A equipe de capacitação fornece suporte concentrando-se na governança e garantindo a consistência entre as equipes de microfront-end. (A equipe da plataforma não deve se envolver com isso.) A equipe de capacitação mantém recursos compartilhados, como uma biblioteca de interface do usuário, e cria padrões como escolha de estrutura, orçamentos de desempenho e convenções de interoperabilidade. Ao mesmo tempo, fornece treinamento para novas equipes ou membros da equipe na aplicação de padrões e ferramentas, conforme definido pela governança.

Implantação

O norte da autonomia em equipes de microfront-end é ter um pipeline automatizado com um caminho de produção independente de outras equipes de microfrontend. Equipes que seguem o princípio de não compartilhar nada podem implementar pipelines independentes. As equipes que compartilham bibliotecas ou dependem de uma equipe de plataforma devem decidir como gerenciar dependências nos pipelines de implantação.

Normalmente, cada pipeline faz o seguinte:

- Cria ativos de front-end
- Implanta os ativos na hospedagem para consumo
- Garante que os registros e caches sejam atualizados para que as novas versões possam ser entregues aos clientes

As etapas reais do pipeline variam de acordo com a pilha de tecnologia e a abordagem de composição da página.

Para composição do lado do cliente, isso significa fazer upload de pacotes de aplicativos para buckets de hospedagem e liberá-los para consumo por meio de armazenamento em cache em uma CDN. Os aplicativos que usam o cache do navegador com os trabalhadores de serviços também devem implementar formas de atualizar os caches dos trabalhadores de serviços.

Para composição do lado do servidor, isso geralmente significa implantar a nova versão do componente do servidor e atualizar o registro do microfront-end para tornar a nova versão detectável. Você pode usar blue/green nossos padrões de implantação canários para lançar gradualmente novas versões.

Governança

Várias personas normalmente trabalham em microfront-ends, e cada uma trabalha sob restrições diferentes em direção a objetivos comerciais comuns. Embora a comunicação e a colaboração entre as pessoas sejam fundamentais para o sucesso, a comunicação excessiva e a implementação de processos excessivamente complicados retardam o ciclo de desenvolvimento. Isso resulta em diminuição do moral e diminui o nível de qualidade.

As empresas mais bem-sucedidas que implementam microfrontends usando várias equipes criam mecanismos para equilibrar autonomia e alinhamento. Eles capacitam os tomadores de decisão a agir localmente e a escalar hierarquicamente somente quando necessário. Os mecanismos incluem o seguinte:

- Contratos de API
- Interatividade cruzada usando eventos
- Equilibrando autonomia com alinhamento
- Sinalizadores de destaque
- Descoberta de serviço

Contratos de API

Cada microfrontend é um sistema capaz de encapsular opiniões, lógica e complexidade. As preocupações transversais geralmente incluem o seguinte:

- Sistemas de design – Ferramentas para desenvolver UIs distribuídas como bibliotecas
- Composição – A forma como um microfrontend precisa interagir com um shell de aplicativo para renderizar e herdar seu contexto
- Manipulação lógica – Interação com APIs para lidar com o estado persistente
- Interatividade com outros microfront-ends – Cenários como publicar e consumir eventos ou navegar de um microfrontend para outro

Para acelerar o consumo e a solução de problemas, é comum investir na padronização da forma como essas interfaces são declaradas e documentadas, incluindo dependências de microfront-end. Wikis organizados por humanos são um bom começo. Uma abordagem mais escalável é armazenar essas informações como metadados estruturados no código. Em seguida, você pode centralizá-lo

para consumo usando a automação para rastrear alterações históricas e fornecer pesquisa de texto completo.

Quando os microfrontends envolvem um grande número de equipes, você precisa de uma estratégia para coordenar as equipes. Compartilhar contratos de API de forma unificada se torna essencial porque reduz a sobrecarga de comunicação e melhora a experiência do desenvolvedor.

O OpenAPI é uma linguagem de especificação para HTTP APIs que suporta a definição de interfaces e contratos de API de forma unificada. Você pode implementar o REST APIs usando a OpenAPI no Amazon API Gateway. Você também pode usar uma grande variedade de estruturas de código aberto que podem ser hospedadas em contêineres ou máquinas virtuais. Uma vantagem significativa é que a OpenAPI pode gerar automaticamente documentação em um formato consistente, para que várias equipes possam compartilhar conhecimento com um investimento inicial mínimo.

Quando várias equipes trabalham em microfront-ends, elas geralmente formam grupos. Nesses grupos, as pessoas podem se encontrar e aprender umas com as outras enquanto pensam e contribuem para o panorama geral. Essas iniciativas geralmente definem e documentam os limites de propriedade, discutem questões transversais e identificam precocemente qualquer duplicação de esforços para resolver problemas comuns.

Interatividade cruzada usando eventos

Em alguns cenários, vários microfront-ends podem precisar interagir entre si para reagir às mudanças de estado ou às ações do usuário. Por exemplo, vários microfront-ends na página podem incluir menus dobráveis. Um menu aparece quando o usuário escolhe um botão. O menu fica oculto quando o usuário clica em qualquer outro lugar, incluindo outro menu que é renderizado em um micro-frontend diferente.

Tecnicamente, uma biblioteca estadual compartilhada, como a Redux, pode ser usada por vários microfront-ends e coordenada por um shell. No entanto, isso cria um acoplamento significativo entre os aplicativos, resultando em um código que é mais difícil de testar e pode diminuir o desempenho durante a renderização.

Uma abordagem comum e eficaz é desenvolver um barramento de eventos distribuído como uma biblioteca, orquestrado pelo shell do aplicativo e usado por vários microfront-ends. Dessa forma, cada microfrontend publica e escuta eventos específicos de forma assíncrona, baseando seu comportamento apenas em seu próprio estado interno. Em seguida, várias equipes podem manter uma página wiki compartilhada que descreve eventos e documenta comportamentos que foram acordados pelos designers de experiência do usuário.

Em uma implementação do exemplo de barramento de eventos, um componente suspenso usa o barramento compartilhado para publicar um evento chamado `drop-down-open-menu` com uma carga útil de. `{"id": "homepage-aboutus-button"}` O componente adiciona um ouvinte ao `drop-down-open-menu` evento para garantir que, se um evento for acionado para uma nova ID, o componente suspenso seja renderizado para ocultar sua seção dobrável. Dessa forma, o microfrontend pode reagir às mudanças de forma assíncrona com maior desempenho e melhor encapsulamento, facilitando que várias equipes projetem e testem comportamentos.

Recomendamos usar o padrão APIs implementado nativamente por navegadores modernos para melhorar a simplicidade e a capacidade de manutenção. A referência de eventos do MDN fornece informações sobre o uso de eventos com aplicativos renderizados do lado do cliente.

Equilibrando autonomia com alinhamento

As arquiteturas de microfront-end são fortemente orientadas para a autonomia da equipe. No entanto, é importante distinguir entre áreas que podem oferecer flexibilidade e abordagens diversas para resolver problemas e áreas em que a padronização é necessária para alcançar o alinhamento. Líderes e arquitetos seniores devem identificar essas áreas desde o início e priorizar os investimentos para equilibrar segurança, desempenho, excelência operacional e confiabilidade dos microfront-ends. Encontrar esse equilíbrio envolve o seguinte: criação, teste, lançamento e registro de microfront-ends, monitoramento e alertas.

Criação de microfront-ends

Idealmente, todas as equipes estão fortemente alinhadas para maximizar os benefícios em termos de desempenho do usuário final. Na prática, isso pode ser difícil e exigir mais esforço. Recomendamos começar com algumas diretrizes escritas com as quais várias equipes podem contribuir por meio de um debate aberto e transparente. As equipes podem então adotar gradualmente o padrão de software Cookiecutter, que suporta a criação de ferramentas que fornecem uma maneira unificada de estruturar um projeto.

Usando essa abordagem, você pode incluir opiniões e restrições. A desvantagem é que essas ferramentas exigem investimentos significativos para criação e manutenção e para garantir que os bloqueadores sejam resolvidos rapidamente sem afetar a produtividade do desenvolvedor.

End-to-end testes para microfront-ends

O teste unitário pode ser deixado para os proprietários. Recomendamos implementar uma estratégia desde o início para fazer testes cruzados de microfront-ends executados em um shell exclusivo.

A estratégia inclui a capacidade de testar aplicativos antes e depois de uma versão de produção. Recomendamos o desenvolvimento de processos e documentação para que pessoas técnicas e não técnicas testem manualmente as funcionalidades críticas.

É importante garantir que as mudanças não prejudiquem a experiência funcional ou não funcional do cliente. Uma estratégia ideal é investir gradualmente em testes automatizados, tanto para os principais recursos quanto para as características da arquitetura, como segurança e desempenho.

Lançamento de microfront-ends

Cada equipe pode ter sua própria maneira de implantar seu código, formar opiniões e ter sua própria infraestrutura. O custo da complexidade para manter esses sistemas geralmente é um impedimento. Em vez disso, recomendamos investir desde o início para implementar uma estratégia compartilhada que possa ser aplicada por ferramentas compartilhadas.

Desenvolva modelos com a plataforma CI/CD de sua escolha. As equipes podem então usar os modelos pré-aprovados e a infraestrutura compartilhada para liberar as alterações na produção. Você pode começar a investir nesse trabalho de desenvolvimento mais cedo, pois esses sistemas raramente precisam de atualizações significativas após um período inicial de teste e consolidação.

Registro em log e monitoramento

Cada equipe pode ter diferentes métricas de negócios e de sistema que deseja monitorar para fins operacionais ou analíticos. O padrão de software Cookiecutter também pode ser aplicado aqui. A entrega de eventos pode ser resumida e disponibilizada como uma biblioteca que vários microfront-ends podem consumir. Para equilibrar flexibilidade e fornecer autonomia, desenvolva ferramentas para registrar métricas personalizadas e criar painéis ou relatórios personalizados. Os relatórios promovem uma estreita colaboração com os proprietários do produto e reduzem o ciclo de feedback do cliente final.

Ao padronizar a entrega, várias equipes podem colaborar para monitorar as métricas. Por exemplo, um site de comércio eletrônico pode acompanhar a jornada do usuário desde o microfrontend “Detalhes do produto” até o microfrontend “Carrinho” e o microfrontend “Compra” para medir o engajamento, a rotatividade e os problemas. Se cada microfrontend registrar eventos usando uma única biblioteca, você poderá consumir esses dados como um todo, explorá-los de forma holística e identificar tendências interessantes.

Geração de alertas

Semelhante ao registro e ao monitoramento, o alerta se beneficia da padronização, com espaço para um certo grau de flexibilidade. Equipes diferentes podem reagir de forma diferente a alertas funcionais e não funcionais. No entanto, se todas as equipes tiverem uma forma consolidada de iniciar alertas com base em métricas coletadas e analisadas em uma plataforma compartilhada, a empresa poderá identificar problemas entre equipes. Esse recurso é útil durante eventos de gerenciamento de incidentes. Por exemplo, os alertas podem ser iniciados da seguinte forma:

- Número elevado de exceções do JavaScript lado do cliente em uma versão específica do navegador
- Tempo de renderização significativamente degradado acima de um determinado limite
- Número elevado de códigos de status 5xx ao consumir uma API específica

Dependendo da maturidade do seu sistema, você pode equilibrar seus esforços em diferentes partes da sua infraestrutura, conforme mostrado na tabela a seguir.

Adoção	Pesquisa e desenvolvimento	Ascensão	maturidade
Crie microfrontends.	Experimente, documente e compartilhe aprendizados.	Invista em ferramentas para criar novos microfrontends. Evangelize a adoção.	Consolide as ferramentas para andaimes. Pressione pela adoção.
Teste os microfrontends de ponta a ponta.	Implemente mecanismos para testar manualmente todos os microfrontends relacionados.	Invista em ferramentas para testes automatizados de segurança e desempenho. Investigue os sinalizadores de recursos e a descoberta de serviços.	Consolide ferramentas para descoberta de serviços, testes em produção e testes automatizados end-to-end.
Libere microfrontends.	Invista em uma infraestrutura compartilhada	Consolide as ferramentas para a infraestrutura de CI/CD Implemente	Crie mecanismos para iniciar reversões automatizadas com base

Adoção	Pesquisa e desenvolvimento	Ascensão	maturidade
	de CI/CD e em lançamentos automatizados em vários ambientes. Evangelize a adoção.	mecanismos manuais de reversão. Pressione pela adoção.	nas métricas e alertas do sistema e da empresa.
Observe o desempe o do microfron tend.	Invista em uma infraestrutura e biblioteca de monitoramento compartilhadas para registrar de forma consistente os eventos do sistema e da empresa.	Consolide as ferramentas para monitoramento e alertas. Implemente painéis entre equipes para monitorar a saúde geral e melhorar o gerenciamento de incidentes.	Padronize os esquemas de registro. Otimize o custo. Implemente alertas com base em métricas comerciais complexas.

Sinalizadores de atributo

Os sinalizadores de recursos podem ser implementados em microfront-ends para facilitar a coordenação de testes e lançamento de recursos em vários ambientes. A técnica de sinalização de recursos consiste em centralizar as decisões em uma loja baseada em booleanos e direcionar o comportamento com base nisso. Geralmente é usado para propagar silenciosamente mudanças que podem ser mantidas ocultas até um momento específico, ao mesmo tempo em que desbloqueia novas versões de novos recursos que, de outra forma, seriam bloqueados, reduzindo a velocidade da equipe.

Considere o exemplo de equipes trabalhando em um recurso de microfrontend que será lançado em uma data específica. O recurso está pronto, mas precisa ser lançado junto com uma alteração em outro microfrontend que seja lançado de forma independente. Bloquear a liberação de ambos os microfront-ends seria considerado um antipadrão e aumentaria o risco quando implantado.

Em vez disso, as equipes podem criar um sinalizador de recurso booleano em um banco de dados que ambas consomem durante o tempo de renderização (talvez por meio de uma chamada HTTP para uma API de sinalizadores de recursos compartilhada). As equipes podem até mesmo lançar a alteração em um ambiente de teste em que o valor booleano é definido `True` para verificar os requisitos funcionais e não funcionais de vários projetos antes de iniciar a produção.

Outro exemplo de uso do sinalizador de recurso é implementar um mecanismo para substituir o valor de um sinalizador definindo um valor específico por meio do `QueryString` parâmetro ou armazenando uma string de teste específica em um cookie. Os proprietários do produto podem iterar os recursos sem bloquear o lançamento de outros recursos ou corrigir erros até a data de lançamento. Na data especificada, alterar o valor do sinalizador no banco de dados torna instantaneamente a alteração visível na produção, sem a necessidade de lançamentos coordenados entre equipes. Depois que um recurso é lançado, as equipes de desenvolvimento limpam o código para remover o comportamento antigo.

Outros casos de uso incluem o lançamento de um sistema de sinalização de recursos baseado em contexto. Por exemplo, se um único site atende clientes em vários idiomas, um recurso pode estar disponível somente para visitantes de um determinado país. O sistema de sinalização de recursos pode depender do consumidor que envia o contexto do país (por exemplo, usando o cabeçalho `Accept-Language` HTTP), e pode haver um comportamento diferente dependendo desse contexto.

Embora os sinalizadores de recursos sejam uma ferramenta poderosa para facilitar a colaboração entre desenvolvedores e proprietários de produtos, eles dependem da diligência das pessoas para evitar uma degradação significativa da base de código. Manter os sinalizadores ativos em vários recursos pode aumentar a complexidade na solução de problemas, aumentar o tamanho do JavaScript pacote e, por fim, acumular dívidas técnicas. As atividades comuns de mitigação incluem o seguinte:

- Teste unitário de cada recurso por trás de um sinalizador para reduzir a probabilidade de bugs, o que pode introduzir ciclos de feedback mais longos nos pipelines automatizados de CI/CD que executam os testes
- Criação de ferramentas para medir o aumento do tamanho do pacote durante as alterações de código, o que pode ser mitigado durante as revisões de código

AWS oferece uma variedade de soluções para otimizar os testes A/B na borda usando as CloudFront funções da Amazon ou o Lambda @Edge. Essas abordagens ajudam a reduzir a complexidade da integração de uma solução ou do produto SaaS existente que você está usando para afirmar suas suposições. Para obter mais informações, consulte Teste A/B.

Descoberta de serviço

O padrão de descoberta de front-ends melhora a experiência de desenvolvimento ao desenvolver, testar e fornecer microfront-ends. O padrão usa uma configuração compartilhável que descreve o ponto de entrada dos microfront-ends. A configuração compartilhável também inclui metadados adicionais que são usados para implantações seguras em cada ambiente usando versões canary.

O desenvolvimento moderno de front-end envolve o uso de uma ampla variedade de ferramentas e bibliotecas para oferecer suporte à modularidade durante o desenvolvimento. Tradicionalmente, esse processo consistia em agrupar código em arquivos individuais que poderiam ser hospedados em uma CDN com o objetivo de manter as chamadas de rede no mínimo durante o tempo de execução, incluindo o carregamento inicial (quando um aplicativo é aberto em um navegador) e o uso (quando um cliente realiza ações como escolher botões ou inserir informações).

Dividindo pacotes

As arquiteturas de microfront-end resolvem os problemas de desempenho causados por pacotes muito grandes gerados pelo agrupamento individual de um grande conjunto de funcionalidades. Por exemplo, um site de comércio eletrônico muito grande pode ser agrupado em um JavaScript arquivo de 6 MB. Apesar da compactação, o tamanho desse arquivo pode afetar negativamente a experiência do usuário ao carregar o aplicativo e baixar o arquivo de uma CDN otimizada para borda.

Se você dividir o aplicativo em página inicial, detalhes do produto e microfront-ends do carrinho, poderá usar um mecanismo de agrupamento para produzir três pacotes individuais de 2 MB. Essa alteração pode melhorar o desempenho do primeiro carregamento em 300% quando os usuários consomem a página inicial. Os pacotes de micro-front-ends do produto ou carrinho são carregados de forma assíncrona somente se e quando o usuário visita a página do produto de um item e decide comprá-lo.

Muitas estruturas e bibliotecas estão disponíveis com base nessa abordagem, e há vantagens para clientes e desenvolvedores. Para identificar limites comerciais que podem resultar na dissociação de dependências no código, você pode mapear diferentes funções de negócios para várias equipes. A propriedade distribuída introduz independência e agilidade.

Ao dividir pacotes de compilação, você pode usar uma configuração para mapear microfront-ends e conduzir a orquestração para o carregamento inicial e para a navegação pós-carregamento. Então, a configuração pode ser consumida durante o tempo de execução e não durante o tempo de construção. Por exemplo, o código de front-end do lado do cliente ou o código de back-end do lado

do servidor podem fazer uma chamada de rede inicial para uma API para buscar dinamicamente a lista de microfrontends. Ele também busca os metadados necessários para composição e integração. Você pode configurar estratégias de failover e armazenamento em cache para garantir confiabilidade e desempenho. O mapeamento dos microfront-ends ajuda a fazer com que implantações individuais de microfront-ends sejam detectáveis por microfront-ends implantados anteriormente que são orquestrados por um aplicativo shell.

Lançamentos do Canary

Uma versão canary é um padrão bem estabelecido e popular para a implantação de microsserviços. As versões do Canary agrupam os usuários-alvo de uma versão em vários grupos, e a versão muda gradualmente, em vez de uma substituição imediata (também conhecida como implantação azul/verde). Um exemplo de uma estratégia de lançamento rápido é implementar uma nova alteração em 10% dos usuários-alvo e adicionar 10% a cada minuto, com uma duração total de 10 minutos para chegar a 100%.

O objetivo de uma versão canary é obter feedback antecipado sobre as mudanças, monitorando o sistema para reduzir o impacto de quaisquer problemas. Quando a automação está em vigor, as métricas do negócio ou do sistema podem ser monitoradas por um sistema interno que pode interromper a implantação ou iniciar uma reversão.

Por exemplo, uma mudança pode introduzir um bug que, nos primeiros dois minutos de uma versão, resulta em perda de receita ou degradação do desempenho. O monitoramento automatizado pode iniciar um alarme. Com o padrão de descoberta de serviços, esse alarme pode interromper a implantação e reverter imediatamente, afetando apenas 20% dos usuários em vez de 100%. A empresa se beneficia da redução do escopo do problema.

Para ver um exemplo de arquitetura que usa o DynamoDB como armazenamento para implementar uma API REST Admin, consulte a solução Frontend Service Discovery on AWS em. GitHub Use o AWS CloudFormation modelo para integrar a arquitetura em seus próprios pipelines de CI/CD. A solução inclui uma API REST Consumer para integrar a solução com seus aplicativos de front-end.

Você precisa de uma equipe de plataforma?

Algumas empresas têm uma equipe responsável por possuir e manter o código, a infraestrutura e os processos que são adotados por outras equipes para trabalhar em microfront-ends. As responsabilidades comuns incluem:

- Crie e mantenha um pipeline de CI/CD que possa ser usado com repositórios contendo microfront-ends. Crie e teste alterações no código e libere-as em vários ambientes.
- Crie e mantenha ferramentas relacionadas à observabilidade, como painéis compartilhados, mecanismos de alerta e sistemas para reagir aos problemas.
- Crie e mantenha bibliotecas compartilhadas para tratamento de eventos, consumo de serviços compartilhados e dependências de terceiros.
- Crie e mantenha ferramentas que monitorem continuamente qualidades não funcionais, como desempenho, segurança e confiabilidade do sistema.
- Crie e mantenha sistemas de design.
- Crie, mantenha e ofereça suporte ao shell do aplicativo para o sistema de microfrontend.

Dependendo da escala do projeto, você pode gerenciar essas responsabilidades usando uma das seguintes abordagens:

- Crie uma equipe de plataforma dedicada cuja única responsabilidade é trabalhar em ferramentas compartilhadas.
- Crie um grupo composto por membros de várias equipes. Os membros do grupo dividem seu tempo entre trabalhar em microfront-ends e trabalhar em ferramentas compartilhadas. Isso também é conhecido como equipe de tigres.

Embora a abordagem da equipe tigre seja uma forma eficaz de manter o foco no cliente, uma equipe tigre geralmente evolui para uma equipe de plataforma se o projeto ganhar força e responsabilidades. Tanto para equipes de plataforma quanto para equipes gigantes, as empresas mais bem-sucedidas que trabalham com microfront-ends formam essas equipes para que várias pessoas com várias experiências e habilidades possam contribuir. Os membros da equipe podem incluir engenheiros de back-end, engenheiros de front-end, designers de experiência do usuário (UX) e gerentes técnicos de produtos. Essa diversidade leva as pessoas a se envolverem continuamente em debates e designs saudáveis com a simplicidade em mente.

Próximas etapas

Este guia abordou padrões arquitetônicos e organizacionais, compensações para decisões importantes e questões de governança relacionadas a microfrontends. As tabelas resumem as vantagens e desvantagens das práticas discutidas neste documento em termos das seguintes dimensões:

- Autonomia – A capacidade de cada equipe de microfrontend de desenvolver de forma independente sua implementação e lançamento para os usuários finais.
- Consistência – A experiência geral do aplicativo em que cada microfrontend se comporta conforme o esperado. Alta consistência significa que os microfront-ends são consistentes com o resto do aplicativo e não prejudicam a experiência do usuário no aplicativo geral.
- Complexidade – A quantidade de infraestrutura, código e esforço necessários para implementar e testar microfront-ends, o aplicativo geral e os controles de governança.

Prática	Autonomia	Consistência	Complexidade
Construindo com microfront-ends em vez de aplicativos monolíticos	Alta	Médio	Alta

Práticas de compartilhamento de código	Autonomia	Consistência	Complexidade
Não compartilhe nada	Alta	Baixo	Baixo
Compartilhe preocupações transversais	Médio	Alta	Médio
Compartilhe a lógica de negócios	Baixo	Alta	Médio
Compartilhe por meio de bibliotecas na hora da construção	Médio	Alta	Baixo

Práticas de compartilhamento de código	Autonomia	Consistência	Complexidade
Compartilhe em tempo de execução	Alta	Alta	Alta

Práticas de descoberta de microfront-end	Autonomia	Consistência	Complexidade
Configurar durante a criação do aplicativo	Baixo	Alta	Baixo
Descoberta do lado do servidor	Alta	Alta	Médio

Práticas de descobe a de microfror t-end	Autonomia	Consistência	Complexidade
Descobe a do lado do cliente (tempo de execuçã	Alta	Alta	Médio

Veja as práticas de composi o	Autonomia	Consistência	Complexidade
Composi o do servidor	Alta	Médio	Alta
Composi o lateral	Médio	Médio	Alta
Composi o do cliente	Alta	Médio	Médio

Para saber mais sobre os conceitos apresentados nesta orientação, consulte a seção Recursos.

Recursos

- Microfront-ends em contexto
- Design orientado por domínio
- EDA Visuals
- Descoberta de front-end
- Descoberta de serviços de front-end em AWS
- Manifesto Ágil
- Referência do evento MDN
- OpenAPI

Colaboradores

As seguintes pessoas contribuíram para este guia.

- Matteo Figus, arquiteto principal de soluções, AWS
- Alexander Guensche, arquiteto sênior de soluções, AWS
- Harun Hasdal, arquiteto sênior de soluções, AWS
- Luca Mezzalira, diretor, arquiteto de soluções especializado em Go to Market, Serverless UK, AWS

Histórico do documento

A tabela a seguir descreve alterações significativas feitas neste guia. Se desejar receber notificações sobre futuras atualizações, inscreva-se em um feed RSS.

Alteração	Descrição	Data
Publicação inicial	—	12 de julho de 2024

AWS Glossário de orientação prescritiva

A seguir estão os termos comumente usados em estratégias, guias e padrões fornecidos pela Orientação AWS Prescritiva. Para sugerir entradas, use o link Fornecer feedback no final do glossário.

Números

7 Rs

Sete estratégias comuns de migração para mover aplicações para a nuvem. Essas estratégias baseiam-se nos 5 Rs identificados pela Gartner em 2011 e consistem em:

- Refatorar/rearquitetar: mova uma aplicação e modifique sua arquitetura aproveitando ao máximo os recursos nativos de nuvem para melhorar a agilidade, a performance e a escalabilidade. Isso normalmente envolve a portabilidade do sistema operacional e do banco de dados. Exemplo: migrar seu banco de dados Oracle on-premises para o Amazon Aurora Edição Compatível com PostgreSQL.
- Redefinir a plataforma (mover e redefinir [mover e redefinir (lift-and-reshape]): mova uma aplicação para a nuvem e introduza algum nível de otimização a fim de aproveitar os recursos da nuvem. Exemplo: migrar seu banco de dados Oracle on-premises para o Amazon Relational Database Service (Amazon RDS) para Oracle na Nuvem AWS.
- Recomprar (drop and shop): mude para um produto diferente, normalmente migrando de uma licença tradicional para um modelo SaaS. Exemplo: migrar seu sistema de gerenciamento de relacionamento com o cliente (CRM) para o Salesforce.com.
- Redefinir a hospedagem (mover sem alterações [lift-and-shift])mover uma aplicação para a nuvem sem fazer nenhuma alteração a fim de aproveitar os recursos da nuvem. Exemplo: migrar seu banco de dados Oracle on-premises para o Oracle em uma instância do EC2 na Nuvem AWS.
- Realocar (mover o hipervisor sem alterações [hypervisor-level lift-and-shift]): mover a infraestrutura para a nuvem sem comprar novo hardware, reescrever aplicações ou modificar suas operações existentes. Você migra servidores de uma plataforma on-premises para um serviço de nuvem para a mesma plataforma. Exemplo: migrar um Microsoft Hyper-V aplicativo para o. AWS
- Reter (revisitar): mantenha as aplicações em seu ambiente de origem. Isso pode incluir aplicações que exigem grande refatoração, e você deseja adiar esse trabalho para um

momento posterior, e aplicações antigas que você deseja manter porque não há justificativa comercial para migrá-las.

- Retirar: desative ou remova aplicações que não são mais necessárias em seu ambiente de origem.

A

ABAC

Consulte controle de acesso baseado em atributo.

serviços abstraídos

Veja serviços gerenciados.

ACID

Veja atomicidade, consistência, isolamento, durabilidade.

migração ativa-ativa

Um método de migração de banco de dados no qual os bancos de dados de origem e de destino são mantidos em sincronia (por meio de uma ferramenta de replicação bidirecional ou operações de gravação dupla), e ambos os bancos de dados lidam com transações de aplicações conectadas durante a migração. Esse método oferece suporte à migração em lotes pequenos e controlados, em vez de exigir uma substituição única. É mais flexível, mas exige mais trabalho do que a migração ativa-passiva.

migração ativa-passiva

Um método de migração de banco de dados em que os bancos de dados de origem e de destino são mantidos em sincronia, mas somente o banco de dados de origem manipula as transações das aplicações conectadas, enquanto os dados são replicados no banco de dados de destino. O banco de dados de destino não aceita nenhuma transação durante a migração.

AGGREGATE FUNCTION

Uma função SQL que opera em um grupo de linhas e calcula um único valor de retorno para o grupo. Exemplos de funções agregadas incluem `SUM` e `MAX`.

AI

Veja inteligência artificial.

AIOps

Veja operações de inteligência artificial.

anonimização

O processo de excluir permanentemente informações pessoais em um conjunto de dados. A anonimização pode ajudar a proteger a privacidade pessoal. Dados anônimos não são mais considerados dados pessoais.

antipadrões

Uma solução frequentemente usada para um problema recorrente em que a solução é contraproducente, ineficaz ou menos eficaz do que uma alternativa.

controle de aplicações

Uma abordagem de segurança que permite o uso somente de aplicações aprovadas para ajudar a proteger um sistema contra malware.

portfólio de aplicações

Uma coleção de informações detalhadas sobre cada aplicação usada por uma organização, incluindo o custo para criar e manter a aplicação e seu valor comercial. Essas informações são fundamentais para o processo de descoberta e análise de portfólio e ajudam a identificar e priorizar as aplicações a serem migradas, modernizadas e otimizadas.

inteligência artificial (IA)

O campo da ciência da computação que se dedica ao uso de tecnologias de computação para desempenhar funções cognitivas normalmente associadas aos humanos, como aprender, resolver problemas e reconhecer padrões. Para obter mais informações, consulte O que é inteligência artificial?

operações de inteligência artificial (AIOps)

O processo de usar técnicas de machine learning para resolver problemas operacionais, reduzir incidentes operacionais e intervenção humana e aumentar a qualidade do serviço. Para obter mais informações sobre como AIOps é usado na estratégia de AWS migração, consulte o guia de integração de operações.

criptografia assimétrica

Um algoritmo de criptografia que usa um par de chaves, uma chave pública para criptografia e uma chave privada para descriptografia. É possível compartilhar a chave pública porque ela não é usada na descriptografia, mas o acesso à chave privada deve ser altamente restrito.

atomicidade, consistência, isolamento, durabilidade (ACID)

Um conjunto de propriedades de software que garantem a validade dos dados e a confiabilidade operacional de um banco de dados, mesmo no caso de erros, falhas de energia ou outros problemas.

controle de acesso por atributo (ABAC)

A prática de criar permissões minuciosas com base nos atributos do usuário, como departamento, cargo e nome da equipe. Para obter mais informações, consulte ABAC AWS na documentação AWS Identity and Access Management (IAM).

fonte de dados autorizada

Um local onde você armazena a versão principal dos dados, que é considerada a fonte de informações mais confiável. Você pode copiar dados da fonte de dados autorizada para outros locais com o objetivo de processar ou modificar os dados, como anonimizá-los, redigi-los ou pseudonimizá-los.

Zona de disponibilidade

Um local distinto dentro de um Região da AWS que está isolado de falhas em outras zonas de disponibilidade e fornece conectividade de rede barata e de baixa latência a outras zonas de disponibilidade na mesma região.

AWS Estrutura de adoção da nuvem (AWS CAF)

Uma estrutura de diretrizes e melhores práticas AWS para ajudar as organizações a desenvolver um plano eficiente e eficaz para migrar com sucesso para a nuvem. AWS O CAF organiza a orientação em seis áreas de foco chamadas perspectivas: negócios, pessoas, governança, plataforma, segurança e operações. As perspectivas de negócios, pessoas e governança têm como foco habilidades e processos de negócios; as perspectivas de plataforma, segurança e operações concentram-se em habilidades e processos técnicos. Por exemplo, a perspectiva das pessoas tem como alvo as partes interessadas que lidam com recursos humanos (RH), funções de pessoal e gerenciamento de pessoal. Nessa perspectiva, o AWS CAF fornece orientação para desenvolvimento, treinamento e comunicação de pessoas para ajudar a preparar a organização para a adoção bem-sucedida da nuvem. Para obter mais informações, consulte o site da AWS CAF e o whitepaper da AWS CAF.

AWS Estrutura de qualificação da carga de trabalho (AWS WQF)

Uma ferramenta que avalia as cargas de trabalho de migração do banco de dados, recomenda estratégias de migração e fornece estimativas de trabalho. AWS O WQF está incluído com AWS

Schema Conversion Tool ()AWS SCT. Ela analisa esquemas de banco de dados e objetos de código, código de aplicações, dependências e características de performance, além de fornecer relatórios de avaliação.

B

bot malicioso

Um bot destinado a causar disrupção ou danos a indivíduos ou organizações.

BCP

Veja planejamento de continuidade de negócios

gráfico de comportamento

Uma visualização unificada e interativa do comportamento e das interações de recursos ao longo do tempo. É possível usar um gráfico de comportamento com o Amazon Detective para examinar tentativas de login malsucedidas, chamadas de API suspeitas e ações similares. Para obter mais informações, consulte Dados em um gráfico de comportamento na documentação do Detective.

sistema big-endian

Um sistema que armazena o byte mais significativo antes. Veja também endianness.

classificação binária

Um processo que prevê um resultado binário (uma de duas classes possíveis). Por exemplo, seu modelo de ML pode precisar prever problemas como “Este e-mail é ou não é spam?” ou “Este produto é um livro ou um carro?”

filtro de bloom

Uma estrutura de dados probabilística e eficiente em termos de memória que é usada para testar se um elemento é membro de um conjunto.

blue/green deployment (implantação azul/verde)

Uma estratégia de implantação em que você cria dois ambientes separados, mas idênticos. Você executa a versão atual da aplicação em um ambiente (azul) e a nova versão da aplicação no outro ambiente (verde). Essa estratégia ajuda você a reverter rapidamente com o mínimo de impacto.

bot

Uma aplicação de software que executa tarefas automatizadas na internet e simula a atividade ou interação humana. Alguns bots são úteis ou benéficos, como crawlers da web que indexam informações na internet. Outros bots, conhecidos como bots maliciosos, têm como objetivo causar disrupção ou danos a indivíduos ou organizações.

botnet

Redes de bots infectadas por malware e sob o controle de uma única parte, conhecidas como bot herder ou operador de bots. Os botnets são o mecanismo mais conhecido para escalar bots e seu impacto.

ramo

Uma área contida de um repositório de código. A primeira ramificação criada em um repositório é a ramificação principal. Você pode criar uma nova ramificação a partir de uma ramificação existente e, em seguida, desenvolver recursos ou corrigir bugs na nova ramificação. Uma ramificação que você cria para gerar um recurso é comumente chamada de ramificação de recurso. Quando o recurso estiver pronto para lançamento, você mesclará a ramificação do recurso de volta com a ramificação principal. Para obter mais informações, consulte Sobre filiais (GitHub documentação).

Acesso de emergência

Em circunstâncias excepcionais e por meio de um processo aprovado, um meio rápido para um usuário obter acesso a um Conta da AWS que ele normalmente não tem permissão para acessar. Para obter mais informações, consulte o indicador Implement break-glass procedures nas orientações do AWS Well-Architected.

estratégia brownfield

A infraestrutura existente em seu ambiente. Ao adotar uma estratégia brownfield para uma arquitetura de sistema, você desenvolve a arquitetura de acordo com as restrições dos sistemas e da infraestrutura atuais. Se estiver expandindo a infraestrutura existente, poderá combinar as estratégias brownfield e greenfield.

cache do buffer

A área da memória em que os dados acessados com mais frequência são armazenados.

capacidade de negócios

O que uma empresa faz para gerar valor (por exemplo, vendas, atendimento ao cliente ou marketing). As arquiteturas de microsserviços e as decisões de desenvolvimento podem

ser orientadas por recursos de negócios. Para obter mais informações, consulte a seção Organizados de acordo com as capacidades de negócios do whitepaper Executar microsserviços conteinerizados na AWS.

planejamento de continuidade de negócios (BCP)

Um plano que aborda o impacto potencial de um evento disruptivo, como uma migração em grande escala, nas operações e permite que uma empresa retome as operações rapidamente.

C

CAF

Veja AWS Cloud Adoption Framework.

implantação canário

O lançamento lento e incremental de uma versão para usuários finais. Quando estiver confiante, você implanta a nova versão e substitui a versão atual por completo.

CCoE

Veja Centro de Excelência da Nuvem.

CDC

Veja captura de dados de alteração.

captura de dados de alterações (CDC)

O processo de rastrear alterações em uma fonte de dados, como uma tabela de banco de dados, e registrar metadados sobre a alteração. É possível usar o CDC para várias finalidades, como auditar ou replicar alterações em um sistema de destino para manter a sincronização.

engenharia do caos

Introduzir intencionalmente falhas ou eventos disruptivos para testar a resiliência de um sistema. Você pode usar AWS Fault Injection Service (AWS FIS) para realizar experimentos que estressam suas AWS cargas de trabalho e avaliar sua resposta.

CI/CD

Veja integração e entrega contínuas.

classificação

Um processo de categorização que ajuda a gerar previsões. Os modelos de ML para problemas de classificação predizem um valor discreto. Os valores discretos são sempre diferentes uns dos outros. Por exemplo, um modelo pode precisar avaliar se há ou não um carro em uma imagem.

criptografia no lado do cliente

Criptografia de dados localmente, antes que o alvo os AWS service (Serviço da AWS) receba.

Centro de excelência em nuvem (CCoE)

Uma equipe multidisciplinar que impulsiona os esforços de adoção da nuvem em toda a organização, incluindo o desenvolvimento de práticas recomendas de nuvem, a mobilização de recursos, o estabelecimento de cronogramas de migração e a liderança da organização em transformações em grande escala. Para obter mais informações, consulte as publicações CCo E no blog de estratégia Nuvem AWS corporativa.

computação em nuvem

A tecnologia de nuvem normalmente usada para armazenamento de dados remoto e gerenciamento de dispositivos de IoT. A computação em nuvem é normalmente conectada à tecnologia de computação de borda.

modelo operacional em nuvem

Em uma organização de TI, o modelo operacional usado para criar, amadurecer e otimizar um ou mais ambientes de nuvem. Para obter mais informações, consulte Criar seu modelo operacional de nuvem.

estágios de adoção da nuvem

As quatro fases pelas quais as organizações normalmente passam ao migrar para a Nuvem AWS:

- Projeto: executar alguns projetos relacionados à nuvem para fins de prova de conceito e aprendizado
- Fundação — Fazer investimentos fundamentais para escalar sua adoção da nuvem (por exemplo, criar uma landing zone, definir um CCo E, estabelecer um modelo de operações)
- Migração: migrar aplicações individuais
- Reinvenção: otimizar produtos e serviços e inovar na nuvem

Esses estágios foram definidos por Stephen Orban na postagem do blog The Journey Toward Cloud-First & the Stages of Adoption no blog de estratégia Nuvem AWS empresarial. Para obter

informações sobre como eles se relacionam com a estratégia de AWS migração, consulte o guia de preparação para migração.

CMDB

Veja banco de dados de gerenciamento de configuração.

repositório de código

Um local onde o código-fonte e outros ativos, como documentação, amostras e scripts, são armazenados e atualizados por meio de processos de controle de versão. Os repositórios de nuvem comuns incluem o GitHub ou o Bitbucket Cloud. Cada versão do código é chamada de ramificação. Em uma estrutura de microsserviços, cada repositório é dedicado a uma única peça de funcionalidade. Um único pipeline de CI/CD pode usar vários repositórios.

cache frio

Um cache de buffer que está vazio, não está bem preenchido ou contém dados obsoletos ou irrelevantes. Isso afeta a performance porque a instância do banco de dados deve ler da memória principal ou do disco, um processo que é mais lento do que a leitura do cache do buffer.

dados frios

Dados que raramente são acessados e geralmente são históricos. Ao consultar esse tipo de dados, consultas lentas geralmente são aceitáveis. Mover esses dados para níveis ou classes de armazenamento de baixo desempenho e menos caros pode reduzir os custos.

visão computacional (CV)

Um campo de IA que usa machine learning para analisar e extrair informações de formatos visuais, como vídeos e imagens digitais. Por exemplo, a Amazon SageMaker AI fornece algoritmos de processamento de imagem para CV.

desvio de configuração

Em uma workload, uma alteração de configuração em relação ao estado esperado. Isso pode fazer com que a workload se torne incompatível e, normalmente, é gradual e não intencional.

banco de dados de gerenciamento de configuração (CMDB)

Um repositório que armazena e gerencia informações sobre um banco de dados e seu ambiente de TI, incluindo componentes de hardware e software e suas configurações. Normalmente, os dados de um CMDB são usados no estágio de descoberta e análise do portfólio da migração.

pacote de conformidade

Uma coleção de AWS Config regras e ações de remediação que você pode montar para personalizar suas verificações de conformidade e segurança. Você pode implantar um pacote de conformidade como uma entidade única em uma Conta da AWS região ou em uma organização usando um modelo YAML. Para obter mais informações, consulte Pacotes de conformidade na documentação. AWS Config

integração contínua e entrega contínua (CI/CD)

O processo de automatizar os estágios de origem, criação, teste, preparação e produção do processo de lançamento do software. CI/CD é comumente descrito como um pipeline. CI/CD pode ajudá-lo a automatizar processos, melhorar a produtividade, melhorar a qualidade do código e entregar com mais rapidez. Para obter mais informações, consulte Benefícios da entrega contínua. CD também pode significar implantação contínua. Para obter mais informações, consulte Entrega contínua versus implantação contínua.

CV

Veja visão computacional.

D

dados em repouso

Dados estacionários em sua rede, por exemplo, dados que estão em um armazenamento.

classificação de dados

Um processo para identificar e categorizar os dados em sua rede com base em criticalidade e confidencialidade. É um componente crítico de qualquer estratégia de gerenciamento de riscos de segurança cibernética, pois ajuda a determinar os controles adequados de proteção e retenção para os dados. A classificação de dados é um componente do pilar de segurança no AWS Well-Architected Framework. Para obter mais informações, consulte Classificação de dados.

desvio de dados

Uma variação significativa entre os dados de produção e os dados usados para treinar um modelo de ML ou uma alteração significativa nos dados de entrada ao longo do tempo. O desvio de dados pode reduzir a qualidade geral, a precisão e a imparcialidade das previsões do modelo de ML.

dados em trânsito

Dados que estão se movendo ativamente pela sua rede, como entre os recursos da rede.

data mesh

Um framework de arquitetura que fornece propriedade de dados distribuída e descentralizada com gerenciamento e governança centralizados.

minimização de dados

O princípio de coletar e processar apenas os dados estritamente necessários. Praticar a minimização de dados no Nuvem AWS pode reduzir os riscos de privacidade, os custos e a pegada de carbono de sua análise.

perímetro de dados

Um conjunto de proteções preventivas em seu AWS ambiente que ajudam a garantir que somente identidades confiáveis acessem recursos confiáveis das redes esperadas. Para obter mais informações, consulte Construindo um perímetro de dados em. AWS

pré-processamento de dados

A transformação de dados brutos em um formato que seja facilmente analisado por seu modelo de ML. O pré-processamento de dados pode significar a remoção de determinadas colunas ou linhas e o tratamento de valores ausentes, inconsistentes ou duplicados.

proveniência dos dados

O processo de rastrear a origem e o histórico dos dados ao longo de seu ciclo de vida, por exemplo, como os dados foram gerados, transmitidos e armazenados.

titular dos dados

Um indivíduo cujos dados estão sendo coletados e processados.

data warehouse

Um sistema de gerenciamento de dados compatível com business intelligence, como analytics. Os data warehouses geralmente contêm grandes quantidades de dados históricos e geralmente são usados para consultas e análises.

linguagem de definição de dados (DDL)

Instruções ou comandos para criar ou modificar a estrutura de tabelas e objetos em um banco de dados.

linguagem de manipulação de dados (DML)

Instruções ou comandos para modificar (inserir, atualizar e excluir) informações em um banco de dados.

DDL

Veja linguagem de definição de banco de dados.

deep ensemble

A combinação de vários modelos de aprendizado profundo para gerar previsões. Os deep ensembles podem ser usados para produzir uma previsão mais precisa ou para estimar a incerteza nas previsões.

Aprendizado profundo

Um subcampo do ML que usa várias camadas de redes neurais artificiais para identificar o mapeamento entre os dados de entrada e as variáveis-alvo de interesse.

defense-in-depth

Uma abordagem de segurança da informação na qual uma série de mecanismos e controles de segurança são cuidadosamente distribuídos por toda a rede de computadores para proteger a confidencialidade, a integridade e a disponibilidade da rede e dos dados nela contidos. Ao adotar essa estratégia AWS, você adiciona vários controles em diferentes camadas da AWS Organizations estrutura para ajudar a proteger os recursos. Por exemplo, uma defense-in-depth abordagem pode combinar autenticação multifatorial, segmentação de rede e criptografia.

administrador delegado

Em AWS Organizations, um serviço compatível pode registrar uma conta de AWS membro para administrar as contas da organização e gerenciar as permissões desse serviço. Essa conta é chamada de administrador delegado para esse serviço Para obter mais informações e uma lista de serviços compatíveis, consulte Serviços que funcionam com o AWS Organizations na documentação do AWS Organizations .

implantação

O processo de criar uma aplicação, novos recursos ou correções de código disponíveis no ambiente de destino. A implantação envolve a implementação de mudanças em uma base de código e, em seguida, a criação e execução dessa base de código nos ambientes da aplicação

ambiente de desenvolvimento

Veja ambiente.

controle detectivo

Um controle de segurança projetado para detectar, registrar e alertar após a ocorrência de um evento. Esses controles são uma segunda linha de defesa, alertando você sobre eventos de segurança que contornaram os controles preventivos em vigor. Para obter mais informações, consulte Controles detectivos em Como implementar controles de segurança na AWS.

mapeamento do fluxo de valor de desenvolvimento (DVSM)

Um processo usado para identificar e priorizar restrições que afetam negativamente a velocidade e a qualidade em um ciclo de vida de desenvolvimento de software. O DVSM estende o processo de mapeamento do fluxo de valor originalmente projetado para práticas de manufatura enxuta. Ele se concentra nas etapas e equipes necessárias para criar e movimentar valor por meio do processo de desenvolvimento de software.

gêmeo digital

Uma representação virtual de um sistema real, como um prédio, fábrica, equipamento industrial ou linha de produção. Os gêmeos digitais oferecem suporte à manutenção preditiva, ao monitoramento remoto e à otimização da produção.

tabela de dimensões

Em um esquema em estrela, uma tabela menor que contém atributos de dados sobre dados quantitativos em uma tabela de fatos. Os atributos da tabela de dimensões geralmente são campos de texto ou números discretos que se comportam como texto. Esses atributos normalmente são usados para restringir consultas, filtrar e rotular conjuntos de resultados.

desastre

Um evento que impede que uma workload ou sistema cumpra seus objetivos de negócios em seu local principal de implantação. Esses eventos podem ser desastres naturais, falhas técnicas ou o resultado de ações humanas, como configuração incorreta não intencional ou ataque de malware.

Recuperação de desastres (RD)

A estratégia e o processo que você usa para minimizar o tempo de inatividade e a perda de dados causados por um disastre. Para obter mais informações, consulte Recuperação de desastres de cargas de trabalho em AWS: Recuperação na nuvem no AWS Well-Architected Framework.

DML

Veja linguagem de manipulação de banco de dados.

design orientado por domínio

Uma abordagem ao desenvolvimento de um sistema de software complexo conectando seus componentes aos domínios em evolução, ou principais metas de negócios, atendidos por cada componente. Esse conceito foi introduzido por Eric Evans em seu livro, Design orientado por domínio: lidando com a complexidade no coração do software (Boston: Addison-Wesley Professional, 2003). Para obter informações sobre como usar o design orientado por domínio com o padrão strangler fig, consulte Modernizar incrementalmente os serviços web herdados do Microsoft ASP.NET (ASMX) usando contêineres e o Amazon API Gateway.

DR

Veja recuperação de desastres.

Detecção da oscilação

Rastreamento de desvios de uma configuração de linha de base. Por exemplo, você pode usar AWS CloudFormation para detectar desvios nos recursos do sistema ou AWS Control Tower para detectar mudanças em seu landing zone que possam afetar a conformidade com os requisitos de governança.

DVSM

Veja mapeamento do fluxo de valor de desenvolvimento.

E

EDA

Veja análise exploratória de dados.

EDI

Veja intercâmbio eletrônico de dados.

computação de borda

A tecnologia que aumenta o poder computacional de dispositivos inteligentes nas bordas de uma rede de IoT. Quando comparada com a computação em nuvem, a computação de borda pode reduzir a latência da comunicação e melhorar o tempo de resposta.

intercâmbio eletrônico de dados (EDI)

A troca automatizada de documentos comerciais entre organizações. Para obter mais informações, consulte O que é EDI (Intercâmbio eletrônico de dados)?.

criptografia

Um processo de computação que transforma dados de texto simples, legíveis por humanos, em texto cifrado.

chave de criptografia

Uma sequência criptográfica de bits aleatórios que é gerada por um algoritmo de criptografia. As chaves podem variar em tamanho, e cada chave foi projetada para ser imprevisível e exclusiva.

endianismo

A ordem na qual os bytes são armazenados na memória do computador. Os sistemas big-endian armazenam o byte mais significativo antes. Os sistemas little-endian armazenam o byte menos significativo antes.

endpoint

Veja endpoint de serviço.

serviço de endpoint

Um serviço que pode ser hospedado em uma nuvem privada virtual (VPC) para ser compartilhado com outros usuários. Você pode criar um serviço de endpoint com AWS PrivateLink e conceder permissões a outros diretores Contas da AWS ou a AWS Identity and Access Management (IAM). Essas contas ou entidades principais podem se conectar ao serviço de endpoint de maneira privada criando endpoints da VPC de interface. Para obter mais informações, consulte Criar um serviço de endpoint na documentação do Amazon Virtual Private Cloud (Amazon VPC).

planejamento de recursos empresariais (ERP)

Um sistema que automatiza e gerencia os principais processos de negócios (como contabilidade, MES e gerenciamento de projetos) para uma empresa.

criptografia envelopada

O processo de criptografar uma chave de criptografia com outra chave de criptografia. Para obter mais informações, consulte Criptografia de envelope na documentação AWS Key Management Service (AWS KMS).

ambiente

Uma instância de uma aplicação em execução. Estes são tipos comuns de ambientes na computação em nuvem:

- ambiente de desenvolvimento: uma instância de uma aplicação em execução que está disponível somente para a equipe principal responsável pela manutenção da aplicação. Ambientes de desenvolvimento são usados para testar mudanças antes de promovê-las para ambientes superiores. Esse tipo de ambiente às vezes é chamado de ambiente de teste.
- ambientes inferiores: todos os ambientes de desenvolvimento para uma aplicação, como aqueles usados para compilações e testes iniciais.
- ambiente de produção: uma instância de uma aplicação em execução que os usuários finais podem acessar. Em um CI/CD pipeline, o ambiente de produção é o último ambiente de implantação.
- ambientes superiores: todos os ambientes que podem ser acessados por usuários que não sejam a equipe principal de desenvolvimento. Isso pode incluir um ambiente de produção, ambientes de pré-produção e ambientes para testes de aceitação do usuário.

epic

Em metodologias ágeis, categorias funcionais que ajudam a organizar e priorizar seu trabalho. Os epics fornecem uma descrição de alto nível dos requisitos e das tarefas de implementação. Por exemplo, os épicos de segurança AWS da CAF incluem gerenciamento de identidade e acesso, controles de detetive, segurança de infraestrutura, proteção de dados e resposta a incidentes. Para obter mais informações sobre epics na estratégia de migração da AWS , consulte o guia de implementação do programa.

ERP

Veja planejamento de recursos empresariais.

análise exploratória de dados (EDA)

O processo de analisar um conjunto de dados para entender suas principais características. Você coleta ou agrega dados e, em seguida, realiza investigações iniciais para encontrar padrões, detectar anomalias e verificar suposições. O EDA é realizado por meio do cálculo de estatísticas resumidas e da criação de visualizações de dados.

F

tabela de fatos

A tabela central em um esquema em estrela. Ela armazena dados quantitativos sobre as operações comerciais. Normalmente, uma tabela de fatos contém dois tipos de colunas: as que contêm medidas e as que contêm uma chave externa para uma tabela de dimensões.

Antecipar-se à falha

Uma filosofia que usa testes frequentes e incrementais para reduzir o ciclo de vida do desenvolvimento. É uma parte essencial de uma abordagem ágil.

delimitação de isolamento contra falhas

No Nuvem AWS, um limite, como uma zona de disponibilidade, Região da AWS um plano de controle ou um plano de dados, que limita o efeito de uma falha e ajuda a melhorar a resiliência das cargas de trabalho. Para obter mais informações, consulte AWS Fault Isolation Boundaries.

ramificação de recursos

Veja ramificação.

recursos

Os dados de entrada usados para fazer uma previsão. Por exemplo, em um contexto de manufatura, os recursos podem ser imagens capturadas periodicamente na linha de fabricação.

importância do recurso

O quanto um recurso é importante para as previsões de um modelo. Isso geralmente é expresso como uma pontuação numérica que pode ser calculada por meio de várias técnicas, como Shapley Additive Explanations (SHAP) e gradientes integrados. Para obter mais informações, consulte Interpretabilidade do modelo de aprendizado de máquina com AWS.

transformação de recursos

O processo de otimizar dados para o processo de ML, incluindo enriquecer dados com fontes adicionais, escalar valores ou extrair vários conjuntos de informações de um único campo de dados. Isso permite que o modelo de ML se beneficie dos dados. Por exemplo, se a data “2021-05-27 00:15:37” for dividida em “2021”, “maio”, “quinta” e “15”, isso poderá ajudar o algoritmo de aprendizado a aprender padrões diferenciados associados a diferentes componentes de dados.

prompt few shot

Fornecer a um LLM um pequeno número de exemplos que demonstram a tarefa e o resultado desejado antes de solicitar que ele execute uma tarefa semelhante. Essa técnica é uma aplicação do aprendizado em contexto, em que os modelos aprendem com exemplos (shots) incorporados aos prompts. Prompts few-shot podem ser eficazes para tarefas que exigem formatação, raciocínio ou conhecimento de domínio específicos. Veja também prompts zero-shot.

FGAC

Veja controle de acesso refinado.

Controle de acesso refinado (FGAC)

O uso de várias condições para permitir ou negar uma solicitação de acesso.

migração flash-cut

Um método de migração de banco de dados que usa replicação contínua de dados via captura de dados de alteração para migrar os dados no menor tempo possível, em vez de usar uma abordagem em fases. O objetivo é reduzir ao mínimo o tempo de inatividade.

FM

Veja modelo de base.

modelo de base (FM)

Uma grande rede neural de aprendizado profundo que vem treinando em grandes conjuntos de dados generalizados e não rotulados. FMs são capazes de realizar uma ampla variedade de tarefas gerais, como entender a linguagem, gerar texto e imagens e conversar em linguagem natural. Para obter mais informações, consulte O que são modelos de base?.

G

IA generativa

Um subconjunto de modelos de IA que foram treinados em grandes quantidades de dados e que podem usar um simples prompt de texto para criar novos artefatos e conteúdo, como imagens, vídeos, texto e áudio. Para obter mais informações, consulte O que é IA generativa?.

bloqueio geográfico

Veja restrições geográficas.

restrições geográficas (bloqueio geográfico)

Na Amazon CloudFront, uma opção para impedir que usuários em países específicos acessem distribuições de conteúdo. É possível usar uma lista de permissões ou uma lista de bloqueios para especificar países aprovados e banidos. Para obter mais informações, consulte Restringir a distribuição geográfica do seu conteúdo na CloudFront documentação.

Fluxo de trabalho do GitFlow

Uma abordagem na qual ambientes inferiores e superiores usam ramificações diferentes em um repositório de código-fonte. O fluxo de trabalho do Gitflow é considerado legado, e o fluxo de trabalho trunk-based é a abordagem moderna e preferencial.

golden image

Um snapshot de um sistema ou software usado como modelo para implantar novas instâncias desse sistema ou software. Por exemplo, na manufatura, uma golden image pode ser usada para provisionar software em vários dispositivos e ajudar a melhorar a velocidade, a escalabilidade e a produtividade nas operações de fabricação de dispositivos.

estratégia greenfield

A ausência de infraestrutura existente em um novo ambiente. Ao adotar uma estratégia greenfield para uma arquitetura de sistema, é possível selecionar todas as novas tecnologias sem a restrição da compatibilidade com a infraestrutura existente, também conhecida como brownfield. Se estiver expandindo a infraestrutura existente, poderá combinar as estratégias brownfield e greenfield.

barreira de proteção

Uma regra de alto nível que ajuda a governar recursos, políticas e conformidade em todas as unidades organizacionais ()OUs. Barreiras de proteção preventivas impõem políticas para garantir o alinhamento a padrões de conformidade. Elas são implementadas usando políticas de controle de serviço e limites de permissões do IAM. Barreiras de proteção detectivas detectam violações de políticas e problemas de conformidade e geram alertas para remediação. Eles são implementados usando AWS Config, AWS Security Hub CSPM, Amazon GuardDuty AWS Trusted Advisor, Amazon Inspector e verificações personalizadas AWS Lambda .

H

HA

Veja alta disponibilidade.

migração heterogênea de bancos de dados

Migrar seu banco de dados de origem para um banco de dados de destino que usa um mecanismo de banco de dados diferente (por exemplo, Oracle para Amazon Aurora). A migração heterogênea geralmente faz parte de um esforço de redefinição da arquitetura, e converter

o esquema pode ser uma tarefa complexa. O AWS fornece o AWS SCT para ajudar nas conversões de esquemas.

alta disponibilidade (HA)

A capacidade de uma workload operar continuamente, sem intervenção, em caso de desafios ou desastres. Os sistemas AH são projetados para realizar o failover automático, oferecer consistentemente desempenho de alta qualidade e lidar com diferentes cargas e falhas com impacto mínimo no desempenho.

modernização de historiador

Uma abordagem usada para modernizar e atualizar os sistemas de tecnologia operacional (OT) para melhor atender às necessidades do setor de manufatura. Um historiador é um tipo de banco de dados usado para coletar e armazenar dados de várias fontes em uma fábrica.

dados de hold-out

Uma parte dos dados históricos rotulados que são retidos de um conjunto de dados usado para treinar um modelo de machine learning. Você pode usar dados de hold-out para avaliar a performance do modelo comparando as predições do modelo com os dados de retenção.

migração homogênea de bancos de dados

Migrar seu banco de dados de origem para um banco de dados de destino que compartilha o mesmo mecanismo de banco de dados (por exemplo, Microsoft SQL Server para Amazon RDS para SQL Server). A migração homogênea geralmente faz parte de um esforço de redefinição da hospedagem ou da plataforma. É possível usar utilitários de banco de dados nativos para migrar o esquema.

dados quentes

Dados acessados com frequência, como dados em tempo real ou dados translacionais recentes. Esses dados normalmente exigem uma camada ou classe de armazenamento de alto desempenho para fornecer respostas rápidas às consultas.

hotfix

Uma correção urgente para um problema crítico em um ambiente de produção. Devido à sua urgência, um hotfix geralmente é feito fora do fluxo de trabalho normal de DevOps lançamento.

período de hipercuidados

Imediatamente após a substituição, o período em que uma equipe de migração gerencia e monitora as aplicações migradas na nuvem para resolver quaisquer problemas. Normalmente,

a duração desse período é de 1 a 4 dias. No final do período de hipercuidados, a equipe de migração normalmente transfere a responsabilidade pelas aplicações para a equipe de operações de nuvem.

eu

IaC

Veja infraestrutura como código.

Política baseada em identidade

Uma política anexada a um ou mais diretores do IAM que define suas permissões no Nuvem AWS ambiente.

aplicação ociosa

Uma aplicação que tem um uso médio de CPU e memória entre 5 e 20% em um período de 90 dias. Em um projeto de migração, é comum retirar essas aplicações ou retê-las on-premises.

IIoT

Veja Internet das Coisas Industrial.

infraestrutura imutável

Um modelo que implanta uma nova infraestrutura para workloads de produção em vez de atualizar, aplicar patches ou modificar a infraestrutura existente. Infraestruturas imutáveis são inerentemente mais consistentes, confiáveis e preditivas do que infraestruturas mutáveis. Para obter mais informações, consulte a prática recomendada Implantar usando infraestrutura imutável no AWS Well-Architected Framework.

VPC de entrada (admissão)

Em uma arquitetura de AWS várias contas, uma VPC que aceita, inspeciona e roteia conexões de rede de fora de um aplicativo. A Arquitetura de Referência de AWS Segurança recomenda configurar sua conta de rede com entrada, saída e inspeção VPCs para proteger a interface bidirecional entre seu aplicativo e a Internet em geral.

migração incremental

Uma estratégia de substituição na qual você migra a aplicação em pequenas partes, em vez de realizar uma única substituição completa. Por exemplo, é possível mover inicialmente

apenas alguns microsserviços ou usuários para o novo sistema. Depois de verificar se tudo está funcionando corretamente, mova os microsserviços ou usuários adicionais de forma incremental até poder descomissionar seu sistema herdado. Essa estratégia reduz os riscos associados a migrações de grande porte.

Indústria 4.0

Um termo que foi introduzido por Klaus Schwab em 2016 para se referir à modernização dos processos de manufatura por meio de avanços em conectividade, dados em tempo real, automação, analytics e IA/ML.

infraestrutura

Todos os recursos e ativos contidos no ambiente de uma aplicação.

Infraestrutura como código (IaC)

O processo de provisionamento e gerenciamento da infraestrutura de uma aplicação por meio de um conjunto de arquivos de configuração. A IaC foi projetada para ajudar você a centralizar o gerenciamento da infraestrutura, padronizar recursos e escalar rapidamente para que novos ambientes sejam reproduzíveis, confiáveis e consistentes.

Internet industrial das coisas (IIoT)

O uso de sensores e dispositivos conectados à Internet nos setores industriais, como manufatura, energia, automotivo, saúde, ciências biológicas e agricultura. Para obter mais informações, consulte Criando uma estratégia de transformação digital industrial da Internet das Coisas (IIoT).

VPC de inspeção

Em uma arquitetura de AWS várias contas, uma VPC centralizada que gerencia as inspeções do tráfego de rede entre VPCs (na mesma ou em diferentes Regiões da AWS) a Internet e as redes locais. A Arquitetura de Referência de AWS Segurança recomenda configurar sua conta de rede com entrada, saída e inspeção VPCs para proteger a interface bidirecional entre seu aplicativo e a Internet em geral.

Internet das coisas (IoT)

A rede de objetos físicos conectados com sensores ou processadores incorporados que se comunicam com outros dispositivos e sistemas pela Internet ou por uma rede de comunicação local. Para obter mais informações, consulte O que é IoT?

interpretabilidade

Uma característica de um modelo de machine learning que descreve o grau em que um ser humano pode entender como as previsões do modelo dependem de suas entradas. Para obter mais informações, consulte Interpretabilidade do modelo de aprendizado de máquina com AWS.

IoT

Veja Internet das Coisas.

Biblioteca de informações de TI (ITIL)

Um conjunto de práticas recomendadas para fornecer serviços de TI e alinhar esses serviços a requisitos de negócios. A ITIL fornece a base para o ITSM.

Gerenciamento de serviços de TI (ITSM)

Atividades associadas a design, implementação, gerenciamento e suporte de serviços de TI para uma organização. Para obter informações sobre a integração de operações em nuvem com ferramentas de ITSM, consulte o guia de integração de operações.

ITIL

Veja biblioteca de informações de TI.

ITSM

Veja gerenciamento de serviços de TI.

L

controle de acesso baseado em etiqueta (LBAC)

Uma implementação do controle de acesso obrigatório (MAC) em que os usuários e os dados em si recebem explicitamente um valor de etiqueta de segurança. A interseção entre a etiqueta de segurança do usuário e a etiqueta de segurança dos dados determina quais linhas e colunas podem ser vistas pelo usuário.

zona de pouso

Uma landing zone é um AWS ambiente bem arquitetado, com várias contas, escalável e seguro. Um ponto a partir do qual suas organizações podem iniciar e implantar rapidamente workloads e aplicações com confiança em seu ambiente de segurança e infraestrutura. Para obter mais

informações sobre zonas de pouso, consulte Configurar um ambiente da AWS com várias contas seguro e escalável.

grande modelo de linguagem (LLM)

Um modelo de IA de aprendizado profundo pré-treinado em uma grande quantidade de dados. Um LLM pode realizar várias tarefas, como responder a perguntas, resumir documentos, traduzir texto para outros idiomas e completar frases. Para obter mais informações, consulte O que são LLMs.

migração de grande porte

Uma migração de 300 servidores ou mais.

LBAC

Veja controle de acesso baseado em rótulo.

privilégio mínimo

A prática recomendada de segurança de conceder as permissões mínimas necessárias para executar uma tarefa. Para obter mais informações, consulte Aplicar permissões de privilégios mínimos na documentação do IAM.

mover sem alterações (lift-and-shift)

Veja 7 Rs.

sistema little-endian

Um sistema que armazena o byte menos significativo antes. Veja também endianness.

LLM

Veja grande modelo de linguagem.

ambientes inferiores

Veja ambiente.

M

machine learning (ML)

Um tipo de inteligência artificial que usa algoritmos e técnicas para reconhecimento e aprendizado de padrões. O ML analisa e aprende com dados gravados, por exemplo, dados da

Internet das Coisas (IoT), para gerar um modelo estatístico baseado em padrões. Para obter mais informações, consulte Machine learning.

ramificação principal

Veja ramificação.

Malware

Software projetado para comprometer a segurança ou a privacidade do computador. O malware pode interromper os sistemas do computador, vazar informações sensíveis ou obter acesso não autorizado. Exemplos de malware incluem vírus, worms, ransomware, cavalos de Troia, spyware e keyloggers.

Serviços gerenciados

Serviços da AWS para o qual AWS opera a camada de infraestrutura, o sistema operacional e as plataformas, e você acessa os endpoints para armazenar e recuperar dados. O Amazon Simple Storage Service (Amazon S3) e o Amazon DynamoDB são exemplos de serviços gerenciados. Eles também são conhecidos como serviços abstraídos.

sistema de execução de manufatura (MES)

Um sistema de software para rastrear, monitorar, documentar e controlar processos de produção que convertem matérias-primas em produtos acabados no chão de fábrica.

MAP

Veja Programa de Aceleração da Migração.

mecanismo

Um processo completo em que você cria uma ferramenta, impulsiona a adoção da ferramenta e, em seguida, inspeciona os resultados para fazer ajustes. Um mecanismo é um ciclo que se reforça e se aprimora à medida que opera. Para obter mais informações, consulte Construindo mecanismos no AWS Well-Architected Framework.

conta de membro

Todos, Contas da AWS exceto a conta de gerenciamento, que fazem parte de uma organização em AWS Organizations. Uma conta só pode ser membro de uma organização de cada vez.

MES

Veja sistema de execução de manufatura.

Transporte de Telemetria de Enfileiramento de Mensagens (MQTT)

Um protocolo de comunicação leve machine-to-machine (M2M), baseado no padrão de publicação/assinatura, para dispositivos de IoT com recursos limitados.

microsserviço

Um serviço pequeno e independente que se comunica de forma bem definida APIs e normalmente é de propriedade de equipes pequenas e independentes. Por exemplo, um sistema de seguradora pode incluir microsserviços que mapeiam as capacidades comerciais, como vendas ou marketing, ou subdomínios, como compras, reclamações ou análises. Os benefícios dos microsserviços incluem agilidade, escalabilidade flexível, fácil implantação, código reutilizável e resiliência. Para obter mais informações, consulte Integração de microsserviços usando serviços sem AWS servidor.

arquitetura de microsserviços

Uma abordagem à criação de aplicações com componentes independentes que executam cada processo de aplicação como um microsserviço. Esses microsserviços se comunicam por meio de uma interface bem definida usando leveza. APIs Cada microsserviço nessa arquitetura pode ser atualizado, implantado e escalado para atender à demanda por funções específicas de uma aplicação. Para obter mais informações, consulte Implementação de microsserviços em. AWS

Programa de Aceleração da Migração (MAP)

Um AWS programa que fornece suporte de consultoria, treinamento e serviços para ajudar as organizações a criar uma base operacional sólida para migrar para a nuvem e ajudar a compensar o custo inicial das migrações. O MAP inclui uma metodologia de migração para executar migrações legadas de forma metódica e um conjunto de ferramentas para automatizar e acelerar cenários comuns de migração.

migração em escala

O processo de mover a maior parte do portfólio de aplicações para a nuvem em ondas, com mais aplicações sendo movidas em um ritmo mais rápido a cada onda. Essa fase usa as práticas recomendadas e lições aprendidas nas fases anteriores para implementar uma fábrica de migração de equipes, ferramentas e processos para agilizar a migração de workloads por meio de automação e entrega ágeis. Esta é a terceira fase da estratégia de migração para a AWS.

fábrica de migração

Equipes multifuncionais que simplificam a migração de workloads por meio de abordagens automatizadas e ágeis. As equipes da fábrica de migração geralmente incluem operações,

analistas e proprietários de negócios, engenheiros de migração, desenvolvedores e DevOps profissionais que trabalham em sprints. Entre 20 e 50% de um portfólio de aplicações corporativas consiste em padrões repetidos que podem ser otimizados por meio de uma abordagem de fábrica. Para obter mais informações, consulte discussão sobre fábricas de migração e o guia do Cloud Migration Factory neste conjunto de conteúdo.

metadados de migração

As informações sobre a aplicação e o servidor necessárias para concluir a migração. Cada padrão de migração exige um conjunto de metadados de migração diferente. Exemplos de metadados de migração incluem a sub-rede, o grupo de segurança e AWS a conta de destino.

padrão de migração

Uma tarefa de migração repetível que detalha a estratégia de migração, o destino da migração e a aplicação ou o serviço de migração usado. Exemplo: rehospede a migração para o Amazon EC2 AWS com o Application Migration Service.

Avaliação de Portfólio para Migração (MPA)

Uma ferramenta on-line que fornece informações para validar o caso de negócios para migrar para a Nuvem AWS. O MPA fornece avaliação detalhada do portfólio (dimensionamento correto do servidor, preços, comparações de TCO, análise de custos de migração), bem como planejamento de migração (análise e coleta de dados de aplicações, agrupamento de aplicações, priorização de migração e planejamento de ondas). A ferramenta MPA (requer login) está disponível gratuitamente para todos os AWS consultores e consultores parceiros da APN.

Avaliação de Preparação para Migração (MRA)

O processo de obter insights sobre o status de prontidão de uma organização para a nuvem, identificar pontos fortes e fracos e criar um plano de ação para fechar as lacunas identificadas, usando o CAF. AWS Para mais informações, consulte o guia de preparação para migração. A MRA é a primeira fase da estratégia de migração para a AWS.

estratégia de migração

A abordagem usada para migrar uma workload para a Nuvem AWS. Para obter mais informações, veja a entrada 7 Rs neste glossário e consulte Mobilize sua organização para acelerar migrações em grande escala.

ML

Veja machine learning.

modernização

Transformar uma aplicação desatualizada (herdada ou monolítica) e sua infraestrutura em um sistema ágil, elástico e altamente disponível na nuvem para reduzir custos, ganhar eficiência e aproveitar as inovações. Para obter mais informações, consulte Strategy for modernizing applications in the Nuvem AWS.

avaliação de preparação para modernização

Uma avaliação que ajuda a determinar a preparação para modernização das aplicações de uma organização. Ela identifica benefícios, riscos e dependências e determina o quão bem a organização pode acomodar o estado futuro dessas aplicações. O resultado da avaliação é um esquema da arquitetura de destino, um roteiro que detalha as fases de desenvolvimento e os marcos do processo de modernização e um plano de ação para abordar as lacunas identificadas. Para obter mais informações, consulte Evaluating modernization readiness for applications in the Nuvem AWS.

aplicações monolíticas (monólitos)

Aplicações que são executadas como um único serviço com processos fortemente acoplados. As aplicações monolíticas apresentam várias desvantagens. Se um recurso da aplicação apresentar um aumento na demanda, toda a arquitetura deverá ser escalada. Adicionar ou melhorar os recursos de uma aplicação monolítica também se torna mais complexo quando a base de código cresce. Para resolver esses problemas, é possível criar uma arquitetura de microsserviços. Para obter mais informações, consulte Decompor monólitos em microsserviços.

MPA

Veja Avaliação do Portfólio para Migração.

MQTT

Veja Transporte de Telemetria de Enfileiramento de Mensagens.

classificação multiclasse

Um processo que ajuda a gerar previsões para várias classes (prevendo um ou mais de dois resultados). Por exemplo, um modelo de ML pode perguntar “Este produto é um livro, um carro ou um telefone?” ou “Qual categoria de produtos é mais interessante para este cliente?”

infraestrutura mutável

Um modelo que atualiza e modifica a infraestrutura existente para workloads de produção. Para melhorar a consistência, confiabilidade e previsibilidade, o AWS Well-Architected Framework recomenda o uso de infraestrutura imutável como uma prática recomendada.

O

OAC

Veja controle de acesso de origem.

OAI

Veja identidade de acesso de origem.

OCM

Veja gerenciamento de alterações organizacionais.

migração offline

Um método de migração no qual a workload de origem é desativada durante o processo de migração. Esse método envolve tempo de inatividade prolongado e geralmente é usado para workloads pequenas e não críticas.

OI

Veja integração de operações.

Ola

Veja acordo de nível operacional.

migração online

Um método de migração no qual a workload de origem é copiada para o sistema de destino sem ser colocada offline. As aplicações conectadas à workload podem continuar funcionando durante a migração. Esse método envolve um tempo de inatividade nulo ou mínimo e normalmente é usado para workloads essenciais para a produção.

OPC-UA

Veja Open Process Communications - Unified Architecture.

Open Process Communications - Unified Architecture (OPC-UA)

Um protocolo de comunicação machine-to-machine (M2M) para automação industrial. O OPC-UA fornece um padrão de interoperabilidade com esquemas de criptografia, autenticação e autorização de dados.

acordo de nível operacional (OLA)

Um acordo que esclarece o que os grupos funcionais de TI prometem oferecer uns aos outros para apoiar um acordo de serviço (SLA).

análise de prontidão operacional (ORR)

Uma lista de verificação de perguntas e práticas recomendadas associadas que ajudam você a entender, avaliar, prevenir ou reduzir o escopo de incidentes e possíveis falhas. Para obter mais informações, consulte Operational Readiness Reviews (ORR) no AWS Well-Architected Framework.

tecnologia operacional (TO)

Sistemas de hardware e software que trabalham com o ambiente físico para controlar operações, equipamentos e infraestrutura industriais. Na manufatura, a integração dos sistemas de tecnologia da informação (TI) e tecnologia operacional (TO) é o foco principal das transformações da Indústria 4.0.

integração de operações (OI)

O processo de modernização das operações na nuvem, que envolve planejamento de preparação, automação e integração. Para obter mais informações, consulte o guia de integração de operações.

trilha organizacional

Uma trilha criada por ela AWS CloudTrail registra todos os eventos de todos Contas da AWS em uma organização em AWS Organizations. Essa trilha é criada em cada Conta da AWS que faz parte da organização e monitora a atividade em cada conta. Para obter mais informações, consulte Criação de uma trilha para uma organização na CloudTrail documentação.

gerenciamento de alterações organizacionais (OCM)

Uma estrutura para gerenciar grandes transformações de negócios disruptivas de uma perspectiva de pessoas, cultura e liderança. O OCM ajuda as organizações a se prepararem e fazerem a transição para novos sistemas e estratégias, acelerando a adoção de alterações, abordando questões de transição e promovendo mudanças culturais e organizacionais. Na estratégia de AWS migração, essa estrutura é chamada de aceleração de pessoas, devido à velocidade de mudança exigida nos projetos de adoção da nuvem. Para obter mais informações, consulte o guia do OCM.

controle de acesso de origem (OAC)

Em CloudFront, uma opção aprimorada para restringir o acesso para proteger seu conteúdo do Amazon Simple Storage Service (Amazon S3). O OAC oferece suporte a todos os buckets S3 Regiões da AWS, criptografia do lado do servidor com AWS KMS (SSE-KMS) e solicitações dinâmicas ao bucket S3. `PUT` `DELETE`

Identidade do acesso de origem (OAI)

Em CloudFront, uma opção para restringir o acesso para proteger seu conteúdo do Amazon S3. Quando você usa o OAI, CloudFront cria um principal com o qual o Amazon S3 pode se autenticar. Os diretores autenticados podem acessar o conteúdo em um bucket do S3 somente por meio de uma distribuição específica. CloudFront Veja também OAC, que fornece um controle de acesso mais granular e aprimorado.

ORR

Veja análise de prontidão operacional.

OT

Veja tecnologia operacional.

VPC de saída (egresso)

Em uma arquitetura de AWS várias contas, uma VPC que gerencia conexões de rede que são iniciadas de dentro de um aplicativo. A Arquitetura de Referência de AWS Segurança recomenda configurar sua conta de rede com entrada, saída e inspeção VPCs para proteger a interface bidirecional entre seu aplicativo e a Internet em geral.

P

limite de permissões

Uma política de gerenciamento do IAM anexada a entidades principais do IAM para definir as permissões máximas que o usuário ou perfil podem ter. Para obter mais informações, consulte Limites de permissões na documentação do IAM.

Informações de identificação pessoal (PII)

Informações que, quando visualizadas diretamente ou combinadas com outros dados relacionados, podem ser usadas para inferir razoavelmente a identidade de um indivíduo. Exemplos de PII incluem nomes, endereços e informações de contato.

PII

Veja informações de identificação pessoal.

manual

Um conjunto de etapas predefinidas que capturam o trabalho associado às migrações, como a entrega das principais funções operacionais na nuvem. Um manual pode assumir a forma de scripts, runbooks automatizados ou um resumo dos processos ou etapas necessários para operar seu ambiente modernizado.

PLC

Veja controlador lógico programável.

PLM

Veja gerenciamento do ciclo de vida do produto.

política

Um objeto que pode definir permissões (veja política baseada em identidade), especificar condições de acesso (veja políta baseada em recurso) ou definir as permissões máximas para todas as contas em uma organização no AWS Organizations (veja política de controle de serviços).

persistência poliglota

Escolher de forma independente a tecnologia de armazenamento de dados de um microsserviço com base em padrões de acesso a dados e outros requisitos. Se seus microsserviços tiverem a mesma tecnologia de armazenamento de dados, eles poderão enfrentar desafios de implementação ou apresentar baixa performance. Os microsserviços serão implementados com mais facilidade e alcançarão performance e escalabilidade melhores se usarem o armazenamento de dados mais bem adaptado às suas necessidades.

avaliação do portfólio

Um processo de descobrir, analisar e priorizar o portfólio de aplicações para planejar a migração. Para obter mais informações, consulte Avaliar a preparação para a migração.

predicado

Uma condição de consulta que retorna `true` ou `false`, normalmente localizada em uma cláusula `WHERE`.

pushdown de predicados

Uma técnica de otimização de consultas de banco de dados que filtra os dados na consulta antes da transferência. Isso reduz a quantidade de dados que devem ser recuperados e processados do banco de dados relacional e melhora a performance das consultas.

controle preventivo

Um controle de segurança projetado para evitar que um evento ocorra. Esses controles são a primeira linha de defesa para ajudar a evitar acesso não autorizado ou alterações indesejadas em sua rede. Para obter mais informações, consulte Controles preventivos em Como implementar controles de segurança na AWS.

principal (entidade principal)

Uma entidade AWS que pode realizar ações e acessar recursos. Essa entidade geralmente é um usuário raiz para um Conta da AWS, uma função do IAM ou um usuário. Para obter mais informações, consulte Entidade principal em Termos e conceitos de perfis na documentação do IAM.

Privacidade por design

Uma abordagem em engenharia de sistemas que leva em consideração a privacidade em todo o processo de desenvolvimento.

zonas hospedadas privadas

Um contêiner que contém informações sobre como você deseja que o Amazon Route 53 responda às consultas de DNS para um domínio e seus subdomínios em um ou mais. VPCs Para obter mais informações, consulte Como trabalhar com zonas hospedadas privadas na documentação do Route 53.

controle proativo

Um controle de segurança desenvolvido para evitar a implantação de recursos não conformes. Esses controles verificam os recursos antes de serem provisionados. Se o recurso não estiver em conformidade com o controle, ele não será provisionado. Para obter mais informações, consulte o guia de referência de controles na AWS Control Tower documentação e consulte Controles proativos em Implementação de controles de segurança em AWS.

gerenciamento do ciclo de vida do produto (PLM)

O gerenciamento de dados e processos de um produto em todo o seu ciclo de vida, desde a concepção, o desenvolvimento e o lançamento, passando pelo crescimento e maturidade, até o declínio e a remoção.

ambiente de produção

Veja ambiente.

controlador lógico programável (PLC)

Na manufatura, um computador altamente confiável e adaptável que monitora as máquinas e automatiza os processos de fabricação.

encadeamento de prompts

Uso da saída de um prompt do LLM como entrada para o próximo prompt para gerar respostas melhores. Essa técnica é usada para dividir uma tarefa complexa em subtarefas, ou para refinar ou expandir iterativamente uma resposta preliminar. Isso ajuda a melhorar a precisão e a relevância das respostas de um modelo e permite resultados mais granulares e personalizados.

pseudonimização

O processo de substituir identificadores pessoais em um conjunto de dados por valores de espaço reservado. A pseudonimização pode ajudar a proteger a privacidade pessoal. Os dados pseudonimizados ainda são considerados dados pessoais.

publish/subscribe (pub/sub)

Um padrão que permite comunicações assíncronas entre microsserviços para melhorar a escalabilidade e a capacidade de resposta. Por exemplo, em um MES baseado em microsserviços, um microsserviço pode publicar mensagens de eventos em um canal em que outros microsserviços possam assinar. O sistema pode adicionar novos microsserviços sem alterar o serviço de publicação.

Q

plano de consulta

Uma série de etapas, como instruções, usadas para acessar os dados em um sistema de banco de dados relacional SQL.

regressão de planos de consultas

Quando um otimizador de serviço de banco de dados escolhe um plano menos adequado do que escolhia antes de uma determinada alteração no ambiente de banco de dados ocorrer. Isso pode ser causado por alterações em estatísticas, restrições, configurações do ambiente, associações de parâmetros de consulta e atualizações do mecanismo de banco de dados.

R

Matriz RACI

Veja responsável, aprovador, consultado, informado (RACI).

RAG

Veja geração aumentada via recuperação.

ransomware

Um software mal-intencionado desenvolvido para bloquear o acesso a um sistema ou dados de computador até que um pagamento seja feito.

Matriz RASCI

Veja responsável, aprovador, consultado, informado (RACI).

RCAC

Veja controle de acesso por linha e coluna.

réplica de leitura

Uma cópia de um banco de dados usada somente para leitura. É possível encaminhar consultas para a réplica de leitura e reduzir a carga no banco de dados principal.

Redefinir arquitetura

Veja 7 Rs.

objetivo de ponto de recuperação (RPO).

O máximo período de tempo aceitável desde o último ponto de recuperação de dados. Isso determina o que é considerado uma perda aceitável de dados entre o último ponto de recuperação e a interrupção do serviço.

objetivo de tempo de recuperação (RTO)

O máximo atraso aceitável entre a interrupção e a restauração do serviço.

refatorar

Veja 7 Rs.

Região

Uma coleção de AWS recursos em uma área geográfica. Cada um Região da AWS é isolado e independente dos outros para fornecer tolerância a falhas, estabilidade e resiliência. Para obter informações, consulte Specify which Regiões da AWS your account can use.

regressão

Uma técnica de ML que prevê um valor numérico. Por exemplo, para resolver o problema de "Por qual preço esta casa será vendida?" um modelo de ML pode usar um modelo de regressão linear para prever o preço de venda de uma casa com base em fatos conhecidos sobre a casa (por exemplo, a metragem quadrada).

redefinir a hospedagem

Veja 7 Rs.

versão

Em um processo de implantação, o ato de promover mudanças em um ambiente de produção.

realocar

Veja 7 Rs.

redefinir a plataforma

Veja 7 Rs.

recomprar

Veja 7 Rs.

resiliência

A capacidade de uma aplicação de resistir ou se recuperar de interrupções. Alta disponibilidade e recuperação de desastres são considerações comuns ao planejar a resiliência na Nuvem AWS. Para obter mais informações, consulte Nuvem AWS Resilience.

política baseada em recurso

Uma política associada a um recurso, como um bucket do Amazon S3, um endpoint ou uma chave de criptografia. Esse tipo de política especifica quais entidades principais têm acesso permitido, ações válidas e quaisquer outras condições que devem ser atendidas.

matriz responsável, accountable, consultada, informada (RACI)

Uma matriz que define as funções e responsabilidades de todas as partes envolvidas nas atividades de migração e nas operações de nuvem. O nome da matriz é derivado dos tipos de responsabilidade definidos na matriz: responsável (R), responsabilizável (A), consultado (C) e informado (I). O tipo de suporte (S) é opcional. Se você incluir suporte, a matriz será chamada de matriz RASCI e, se excluir, será chamada de matriz RACI.

controle responsivo

Um controle de segurança desenvolvido para conduzir a remediação de eventos adversos ou desvios em relação à linha de base de segurança. Para obter mais informações, consulte Controles responsivos em Como implementar controles de segurança na AWS.

reter

Veja 7 Rs.

Retirada

Veja 7 Rs.

Geração Aumentada de Recuperação (RAG)

Uma tecnologia de IA generativa em que um LLM faz referência a uma fonte de dados autorizada que está fora de suas fontes de dados de treinamento antes de gerar uma resposta. Por exemplo, um modelo RAG pode realizar uma pesquisa semântica na base de conhecimento ou nos dados personalizados de uma organização. Para obter mais informações, consulte O que é RAG (geração aumentada via recuperação)?.

alternância

O processo de atualizar periodicamente um segredo para dificultar o acesso de um invasor às credenciais.

controle de acesso por linha e coluna (RCAC)

O uso de expressões SQL básicas e flexíveis que tenham regras de acesso definidas. O RCAC consiste em permissões de linha e máscaras de coluna.

RPO

Veja objetivo de ponto de recuperação.

RTO

Veja objetivo de tempo de recuperação.

runbook

Um conjunto de procedimentos manuais ou automatizados necessários para realizar uma tarefa específica. Eles são normalmente criados para agilizar operações ou procedimentos repetitivos com altas taxas de erro.

S

SAML 2.0

Um padrão aberto que muitos provedores de identidade (IdPs) usam. Esse recurso permite o login único federado (SSO), para que os usuários possam fazer login Console de gerenciamento da AWS ou chamar as operações da AWS API sem que você precise criar um usuário no IAM para todos em sua organização. Para obter mais informações sobre a federação baseada em SAML 2.0, consulte Sobre a federação baseada em SAML 2.0 na documentação do IAM.

SCADA

Veja controle de supervisão e aquisição de dados.

SCP

Veja política de controle de serviço.

secret

Em AWS Secrets Manager, informações confidenciais ou restritas, como uma senha ou credenciais de usuário, que você armazena de forma criptografada. Consiste no valor secreto e em seus metadados. O valor secreto pode ser binário, uma única string ou várias strings. Para obter mais informações, consulte What's in a Secrets Manager secret? na documentação do Secrets Manager.

segurança desde a concepção

Uma abordagem em engenharia de sistemas que leva em consideração a segurança em todo o processo de desenvolvimento.

controle de segurança

Uma barreira de proteção técnica ou administrativa que impede, detecta ou reduz a capacidade de uma ameaça explorar uma vulnerabilidade de segurança. Existem quatro tipos primários de controles de segurança: preventivos, detectivos, responsivos e proativos.

hardening da segurança

O processo de reduzir a superfície de ataque para torná-la mais resistente a ataques. Isso pode incluir ações como remover recursos que não são mais necessários, implementar a prática recomendada de segurança de conceder privilégios mínimos ou desativar recursos desnecessários em arquivos de configuração.

sistema de gerenciamento de eventos e informações de segurança (SIEM)

Ferramentas e serviços que combinam sistemas de gerenciamento de informações de segurança (SIM) e gerenciamento de eventos de segurança (SEM). Um sistema SIEM coleta, monitora e analisa dados de servidores, redes, dispositivos e outras fontes para detectar ameaças e violações de segurança e gerar alertas.

automação de resposta de segurança

Uma ação predefinida e programada projetada para responder ou remediar automaticamente um evento de segurança. Essas automações servem como controles de segurança responsivos ou detectivos que ajudam você a implementar as melhores práticas AWS de segurança. Exemplos de ações de resposta automatizada incluem a modificação de um grupo de segurança da VPC, a aplicação de patches em uma instância do Amazon EC2 ou a alternância de credenciais.

Criptografia do lado do servidor

Criptografia dos dados em seu destino, por AWS service (Serviço da AWS) quem os recebe.

política de controle de serviços (SCP)

Uma política que fornece controle centralizado sobre as permissões de todas as contas em uma organização em AWS Organizations. SCPs defina barreiras ou estabeleça limites nas ações que um administrador pode delegar a usuários ou funções. Você pode usar SCPs como listas de permissão ou listas de negação para especificar quais serviços ou ações são permitidos ou proibidos. Para obter mais informações, consulte Políticas de controle de serviço na AWS Organizations documentação.

service endpoint (endpoint de serviço)

O URL do ponto de entrada para um AWS service (Serviço da AWS). Você pode usar o endpoint para se conectar programaticamente ao serviço de destino. Para obter mais informações, consulte Endpoints do AWS service (Serviço da AWS) na Referência geral da AWS.

acordo de serviço (SLA)

Um acordo que esclarece o que uma equipe de TI promete fornecer aos clientes, como tempo de atividade e performance do serviço.

indicador de nível de serviço (SLI)

Uma avaliação de um aspecto de performance de um serviço, como taxa de erro, disponibilidade ou throughput.

objetivo de nível de serviço (SLO)

Uma métrica alvo que representa a integridade de um serviço, conforme avaliado por um indicador de nível de serviço.

modelo de responsabilidade compartilhada

Um modelo que descreve a responsabilidade com a qual você compartilha AWS pela segurança e conformidade na nuvem. AWS é responsável pela segurança da nuvem, enquanto você é responsável pela segurança na nuvem. Para obter mais informações, consulte o Modelo de responsabilidade compartilhada.

SIEM

Veja sistema de gerenciamento de eventos e informações de segurança.

ponto único de falha (SPOF)

Uma falha em um único componente crítico de uma aplicação que pode interromper o sistema.

SLA

Veja cacordo de serviço.

SLI

Veja indicador de nível de serviço.

SLO

Veja objetivo de nível de serviço.

split-and-seed modelo

Um padrão para escalar e acelerar projetos de modernização. À medida que novos recursos e lançamentos de produtos são definidos, a equipe principal se divide para criar novas equipes de produtos. Isso ajuda a escalar os recursos e os serviços da sua organização, melhora a produtividade do desenvolvedor e possibilita inovações rápidas. Para obter mais informações, consulte Phased approach to modernizing applications in the Nuvem AWS.

SPOF

Veja ponto único de falha.

esquema em estrela

Uma estrutura organizacional de banco de dados que usa uma grande tabela de fatos para armazenar dados transacionais ou medidos e usa uma ou mais tabelas dimensionais menores para armazenar atributos de dados. Essa estrutura foi projetada para ser usada em um data warehouse ou para fins de inteligência comercial.

padrão strangler fig

Uma abordagem à modernização de sistemas monolíticos que consiste em reescrever e substituir incrementalmente a funcionalidade do sistema até que o sistema herdado possa ser desativado. Esse padrão usa a analogia de uma videira que cresce e se torna uma árvore estabelecida e, eventualmente, supera e substitui sua hospedeira. O padrão foi apresentado por Martin Fowler como forma de gerenciar riscos ao reescrever sistemas monolíticos. Para ver um exemplo de como aplicar esse padrão, consulte Modernizar incrementalmente os serviços Web herdados do Microsoft ASP.NET (ASMX) usando contêineres e o Amazon API Gateway.

sub-rede

Um intervalo de endereços IP na VPC. Cada sub-rede fica alocada em uma única zona de disponibilidade.

controle supervisório e aquisição de dados (SCADA)

Na manufatura, um sistema que usa hardware e software para monitorar ativos físicos e operações de produção.

symmetric encryption (criptografia simétrica)

Um algoritmo de criptografia que usa a mesma chave para criptografar e descriptografar dados.

testes sintéticos

Testar um sistema de forma que simule as interações do usuário para detectar possíveis problemas ou monitorar a performance. Você pode usar o Amazon CloudWatch Synthetics para criar esses testes.

prompt do sistema

Uma técnica para fornecer contexto, instruções ou orientações a um LLM a fim de direcionar seu comportamento. Os prompts do sistema ajudam a definir o contexto e a estabelecer regras para interações com os usuários.

T

tags

Pares de valores-chave que atuam como metadados para organizar seus recursos. AWS As tags podem ajudar você a gerenciar, identificar, organizar, pesquisar e filtrar recursos da . Para obter mais informações, consulte Marcar seus recursos do AWS.

variável-alvo

O valor que você está tentando prever no ML supervisionado. Ela também é conhecida como variável de resultado. Por exemplo, em uma configuração de fabricação, a variável-alvo pode ser um defeito do produto.

lista de tarefas

Uma ferramenta usada para monitorar o progresso por meio de um runbook. Uma lista de tarefas contém uma visão geral do runbook e uma lista de tarefas gerais a serem concluídas. Para cada tarefa geral, ela inclui o tempo estimado necessário, o proprietário e o progresso.

ambiente de teste

Veja ambiente.

treinamento

O processo de fornecer dados para que seu modelo de ML aprenda. Os dados de treinamento devem conter a resposta correta. O algoritmo de aprendizado descobre padrões nos dados de treinamento que mapeiam os atributos dos dados de entrada no destino (a resposta que você deseja prever). Ele gera um modelo de ML que captura esses padrões. Você pode usar o modelo de ML para obter previsões de novos dados cujo destino você não conhece.

gateway de trânsito

Um hub de trânsito de rede que você pode usar para interconectar sua rede com VPCs a rede local. Para obter mais informações, consulte O que é um gateway de trânsito na AWS Transit Gateway documentação.

fluxo de trabalho baseado em troncos

Uma abordagem na qual os desenvolvedores criam e testam recursos localmente em uma ramificação de recursos e, em seguida, mesclam essas alterações na ramificação principal. A ramificação principal é então criada para os ambientes de desenvolvimento, pré-produção e produção, sequencialmente.

Acesso confiável

Conceder permissões a um serviço que você especifica para realizar tarefas em sua organização AWS Organizations e em suas contas em seu nome. O serviço confiável cria um perfil vinculado ao serviço em cada conta, quando esse perfil é necessário, para realizar tarefas de gerenciamento para você. Para obter mais informações, consulte Usando AWS Organizations com outros AWS serviços na AWS Organizations documentação.

tuning (ajustar)

Alterar aspectos do processo de treinamento para melhorar a precisão do modelo de ML. Por exemplo, você pode treinar o modelo de ML gerando um conjunto de rótulos, adicionando rótulos e repetindo essas etapas várias vezes em configurações diferentes para otimizar o modelo.

equipe de duas pizzas

Uma pequena DevOps equipe que você pode alimentar com duas pizzas. Uma equipe de duas pizzas garante a melhor oportunidade possível de colaboração no desenvolvimento de software.

U

incerteza

Um conceito que se refere a informações imprecisas, incompletas ou desconhecidas que podem minar a confiabilidade dos modelos preditivos de ML. Há dois tipos de incertezas: a incerteza epistêmica é causada por dados limitados e incompletos, enquanto a incerteza aleatória é causada pelo ruído e pela aleatoriedade inerentes aos dados. Para obter mais informações, consulte o guia Como quantificar a incerteza em sistemas de aprendizado profundo.

tarefas indiferenciadas

Também conhecido como trabalho pesado, trabalho necessário para criar e operar um aplicativo, mas que não fornece valor direto ao usuário final nem oferece vantagem competitiva. Exemplos de tarefas indiferenciadas incluem aquisição, manutenção e planejamento de capacidade.

ambientes superiores

Veja ambiente.

V

aspiração

Uma operação de manutenção de banco de dados que envolve limpeza após atualizações incrementais para recuperar armazenamento e melhorar a performance.

controle de versões

Processos e ferramentas que rastreiam mudanças, como alterações no código-fonte em um repositório.

emparelhamento da VPC

Uma conexão entre duas VPCs que permite rotear o tráfego usando endereços IP privados. Para ter mais informações, consulte O que é emparelhamento de VPC? na documentação da Amazon VPC.

Vulnerabilidade

Uma falha de software ou hardware que compromete a segurança do sistema.

W

cache quente

Um cache de buffer que contém dados atuais e relevantes que são acessados com frequência. A instância do banco de dados pode ler do cache do buffer, o que é mais rápido do que ler da memória principal ou do disco.

dados mornos

Dados acessados raramente. Ao consultar esse tipo de dados, consultas moderadamente lentas geralmente são aceitáveis.

função de janela

Uma função SQL que executa um cálculo em um grupo de linhas que se relacionam de alguma forma com o registro atual. As funções de janela são úteis para processar tarefas, como calcular uma média móvel ou acessar o valor das linhas com base na posição relativa da linha atual.

workload

Uma coleção de códigos e recursos que geram valor empresarial, como uma aplicação voltada para o cliente ou um processo de backend.

workstreams

Grupos funcionais em um projeto de migração que são responsáveis por um conjunto específico de tarefas. Cada workstream é independente, mas oferece suporte aos outros workstreams do projeto. Por exemplo, o workstream de portfólio é responsável por priorizar aplicações, planejar ondas e coletar metadados de migração. O workstream de portfólio entrega esses ativos ao workstream de migração, que então migra os servidores e as aplicações.

WORM

Veja gravação única e várias leituras.

WQF

Veja AWS Workload Qualification Framework.

gravação única e várias leituras (WORM)

Um modelo de armazenamento que grava dados uma única vez e evita que os dados sejam excluídos ou modificados. Os usuários autorizados podem ler os dados quantas vezes forem necessárias, mas não podem alterá-los. Essa infraestrutura de armazenamento de dados é considerada imutável.

Z

exploração de dia zero

Um ataque, normalmente malware, que tira proveito de uma vulnerabilidade zero-day.

vulnerabilidade de dia zero

Uma falha ou vulnerabilidade não mitigada em um sistema de produção. Os agentes de ameaças podem usar esse tipo de vulnerabilidade para atacar o sistema. Os desenvolvedores frequentemente ficam cientes da vulnerabilidade como resultado do ataque.

prompt zero shot

Fornecer a um LLM instruções para realizar uma tarefa, mas sem exemplos (shots) que possam ajudar a orientá-lo. O LLM deve usar seu conhecimento pré-treinado para lidar com a tarefa. A

eficácia dos prompts zero-shot depende da complexidade da tarefa e da qualidade do prompt. Veja também prompts few-shot.

aplicação zumbi

Uma aplicação que tem um uso médio de CPU e memória inferior a 5%. Em um projeto de migração, é comum retirar essas aplicações.

As traduções são geradas por tradução automática. Em caso de conflito entre o conteúdo da tradução e da versão original em inglês, a versão em inglês prevalecerá.