WASHINGTON LUIZ PERONI

MACHINE LEARNING ENGINEERING

BIG DATA CLOUD PLATFORMS

AULA 03

SUMÁRIO

O QUE VEM POR AÍ?

Olá, pessoal! Boas-vindas à aula de AWS SQS, um serviço de fila de mensagens que ajuda na arquitetura de sistemas desacoplados.

O Amazon Simple Queue Service (SQS) é um serviço de mensagens totalmente gerenciado pela AWS que permite a comunicação entre diferentes partes de uma aplicação ou sistemas distribuídos. Ele segue o modelo de fila de mensagens, no qual mensagens são enviadas para uma fila e consumidores recuperam e processam essas mensagens. O SQS oferece uma solução eficaz para comunicação assíncrona entre componentes distribuídos, sendo uma opção valiosa em arquiteturas na nuvem.

Demonstraremos alguns casos de uso em conjunto com outros serviços, como Glue, Lambda e S3, para que vocês possam aplicar os conhecimentos adquiridos no mundo real.

HANDS ON

Demonstraremos na prática, dentro do console da AWS, o uso do AWS SQS, mostrando desde a tela inicial e os conceitos básicos até um caso de uso real de integração com Glue, Lambda e S3. Veremos como o uso do SQS é importante para limitar o escopo entre aplicações e trazer robustez e escala para a arquitetura desacoplada em Big Data. O conteúdo será dividido nas seguintes etapas:

- AWS SQS via console: configuração inicial.
- AWS SQS: tipos de filas, Standart e FIFO.
- Serviço AWS como Produtor escrevendo na Fila SQS.
- Serviço AWS como Consumidor lendo da Fila SQS.

SAIBA MAIS

Histórico AWS SQS:

O **Amazon Simple Queue Service (SQS)** foi lançado pela Amazon Web Services (AWS) em 2004 para atender à crescente demanda por uma solução eficaz de filas de mensagens em sistemas distribuídos. Confira um resumo do histórico de criação do AWS SQS.

Lançamento inicial (2004)

O SQS foi lançado para fornecer uma solução simples e altamente escalável para sistemas distribuídos na nuvem. O serviço foi projetado para desacoplar componentes de aplicativos distribuídos, permitindo que eles se comuniquem de maneira assíncrona e confiável.

Modelo de fila de mensagens

O SQS introduziu o modelo de fila de mensagens, no qual os produtores podem enviar mensagens para uma fila e os consumidores podem buscar e processar essas mensagens. Isso proporcionou uma forma eficiente de comunicação assíncrona entre diferentes partes de uma aplicação ou sistema distribuído.

Características de alta disponibilidade e escalabilidade:

Desde o início, o SQS foi projetado para ser totalmente gerenciado pela AWS, proporcionando alta disponibilidade e escalabilidade automática. Isso permitiu que os desenvolvedores se concentrassem na lógica de negócios, sem se preocupar com a infraestrutura subjacente.

Melhorias ao longo do tempo:

Ao longo dos anos, a AWS continuou aprimorando o SQS, introduzindo novos recursos e melhorias de desempenho. Isso incluiu a adição de filas FIFO (First-In-First-Out) para garantir a ordem exata de processamento de mensagens.

Integração com outros serviços AWS

O SQS foi integrado a outros serviços da AWS, como AWS Lambda, Amazon EC2 e Amazon S3, proporcionando uma maior flexibilidade e funcionalidade aos desenvolvedores.

Adoção global

Devido à sua confiabilidade, escalabilidade e facilidade de uso, o SQS foi adotado globalmente entre empresas de diversos setores. Tornou-se uma peça fundamental em arquiteturas de microsserviços, sistemas de processamento em lote e outras aplicações distribuídas.

O SQS continua a ser uma solução popular para a comunicação assíncrona na nuvem, refletindo a evolução contínua das demandas e necessidades dos desenvolvedores em ambientes distribuídos.

Arquitetura desacoplada com filas

Uma arquitetura desacoplada com filas é um design em que os componentes de um sistema podem se comunicar de maneira assíncrona e desacoplada por meio do uso de filas de mensagens. Este modelo promove a escalabilidade, a resiliência e a flexibilidade, pois permite que partes independentes de um sistema interajam sem conhecimento direto umas das outras**.**

Principais benefícios:

Desacoplamento: componentes podem operar independentemente, sem conhecimento direto uns dos outros.

Escalabilidade: facilita o dimensionamento horizontal, pois os componentes podem ser adicionados ou removidos sem afetar outros.

Resiliência: mensagens podem ser armazenadas temporariamente em caso de falhas, evitando perda de dados.

Flexibilidade: diferentes componentes podem ser desenvolvidos e evoluídos independentemente.

Exemplos Práticos

Processamento de pedidos em comércio eletrônico: uma aplicação de comércio eletrônico pode usar filas para processar pedidos. Após a finalização do pedido, uma mensagem é colocada na fila, indicando que o pedido precisa ser

processado. Diversos serviços (consumidores) podem então processar essas mensagens, atualizando inventários, gerando faturas e notificando clientes.

Integração de microsserviços: em uma arquitetura de microsserviços, filas são usadas para comunicação assíncrona entre serviços. Isso permite que os serviços publiquem eventos ou solicitações na fila e que outros serviços os consumam conforme necessário. Isso desacopla os serviços, permitindo que evoluam independentemente.

Processamento em lote: aplicações que envolvem processamento em lote, como análise de big data, podem usar filas para enfileirar tarefas de processamento. Cada trabalhador pode pegar uma mensagem da fila e processar os dados, garantindo escalabilidade e resiliência.

Arquitetura de software desacoplada: a arquitetura de software desacoplada refere-se a um design em que os diferentes componentes de um sistema são projetados para operar independentemente, minimizando as dependências entre eles. Isso promove a flexibilidade, a escalabilidade e facilita a manutenção, pois as alterações em um componente têm menos impacto nos outros.

Histórico

O conceito de desacoplamento em arquitetura de software tem suas raízes na busca por designs mais flexíveis e adaptáveis. Com a evolução das práticas de engenharia de software, arquiteturas desacopladas tornaram-se fundamentais em ambientes distribuídos, microsserviços e sistemas altamente escaláveis.

Exemplos Práticos

Microsserviços: arquiteturas de microsserviços são um exemplo proeminente de desacoplamento. Cada microsserviço é independente, executando uma única função específica e se comunicando com outros serviços por meio de interfaces bem definidas.

Mensageria assíncrona: o uso de sistemas de mensageria assíncrona, como filas de mensagens (por exemplo, AWS SQS, RabbitMQ), permite a comunicação

entre componentes sem conhecimento direto uns dos outros. Isso desacopla a produção e o consumo de mensagens.

Injeção de Dependência: princípios como Injeção de Dependência (DI) permitem que componentes sejam construídos sem depender explicitamente de implementações específicas, aumentando a flexibilidade e facilitando a substituição de partes do sistema.

Vantagens:

Flexibilidade e manutenção: facilita a introdução de alterações sem impactar outros componentes, permitindo a evolução contínua do software.

Escalabilidade: componentes podem ser escalados independentemente, proporcionando melhor aproveitamento dos recursos.

Reusabilidade: componentes desacoplados são frequentemente mais reutilizáveis, pois não têm dependências fortes de contexto.

Desvantagens:

Complexidade adicional: pode introduzir complexidade adicional, especialmente em sistemas muito distribuídos.

Comunicação assíncrona: em alguns casos, a comunicação assíncrona pode ser mais difícil de entender e depurar.

A arquitetura desacoplada é uma estratégia poderosa quando bem aplicada, proporcionando sistemas mais flexíveis e resilientes. No entanto, é essencial equilibrar o desacoplamento com a complexidade adicional que pode surgir.

Comparação entre arquiteturas de software desacoplada e monolítica

Definição de arquitetura monolítica: uma aplicação monolítica é desenvolvida como uma unidade, onde todos os componentes são empacotados e implantados juntos.

Comunicação interna: os componentes se comunicam diretamente dentro da aplicação monolítica.

Escalabilidade: geralmente escala verticalmente, adicionando mais recursos à instância existente.

Manutenção: mudanças frequentes podem exigir a reimplantação da aplicação inteira.

Exemplo prático: uma aplicação de blog onde o frontend, backend e banco de dados estão em um único código base.

Definição de arquitetura desacoplada: uma aplicação desacoplada consiste em componentes independentes que se comunicam entre si.

Comunicação interna: os componentes podem se comunicar por meio de APIs, filas de mensagens ou outros mecanismos assíncronos.

Escalabilidade: pode escalar horizontalmente, adicionando instâncias independentes de componentes específicos.

Manutenção: permite atualizações independentes de componentes, facilitando a evolução contínua.

Exemplo prático: uma aplicação de comércio eletrônico que utiliza microsserviços independentes para gerenciar catálogo de produtos, processamento de pedidos e autenticação de usuários.

Comparação

Complexidade:

- Monolítica: menos complexa para configuração e implantação.
- Desacoplada: pode introduzir complexidade adicional devido à comunicação distribuída.

Escalabilidade:

- Monolítica: escala verticalmente, adicionando mais recursos à instância existente.

- Desacoplada: escala horizontalmente, adicionando mais instâncias independentes.

Manutenção e evolução:

- Monolítica: alterações podem exigir reimplantação da aplicação inteira.
- Desacoplada: permite atualizações independentes de componentes.

QUE VOCÊ VIU NESTA AULA?

Nessa aula vimos como o AWS SQS nos ajuda no desacoplamento de sistemas, ou seja, a dividir escopo e responsabilidade de aplicações para executar tarefas de forma assíncronas, tornando os sistemas mais robustos e escaláveis. Embora isso aumente a complexidade inicial, é importante discutir e rever a arquitetura, para que ela sempre esteja em evolução. Revejam a aula, pratiquem em casa e, em caso de dúvidas, entrem em contato conosco. Bons estudos!

REFERÊNCIAS

ALLAMARAJU, S. **RESTful Web Services Cookbook: Solutions for Improving Scalability and Simplicity**. Sebastopol: O'Reilly Media, 2016.

FOWLER, M. **Microservices: A Definition of This New Architectural Term.** Sebastopol: O'Reilly Media, 2014.

FREEMAN, E; PRYCE, N. **Growing Object-Oriented Software, Guided by Tests.** Boston: Addison-Wesley, 2004.

NEWMAN, S. **Building Microservices: Designing Fine-Grained Systems**. Sebastopol: O'Reilly Media, 2015.

RICHARDSON, C. **Microservices Patterns: With examples in Java.** Shelter Island: Manning Publications, 2018.

PALAVRAS-CHAVE

Palavras-chave: Big Data. Microsserviços. Desacoplamento de Software, Engenharia de Software. AWS SQS.