

Semana 4

Semana 4

- Teoria – 30 min
- Exercícios – 30 min

Semana 4

Domínio 2: Design de arquiteturas resilientes

Declaração de tarefa 2.1: Projetar arquiteturas dimensionáveis e com acoplamento fraco.

Conhecimento sobre:

- Criação e gerenciamento de APIs (por exemplo, Amazon API Gateway, API REST).
- AWS Managed Services com casos de uso apropriados (por exemplo, AWS Transfer Family, Amazon Simple Queue Service [Amazon SQS], Secrets Manager).
- Estratégias de armazenamento em cache.
- Princípios de design para microsserviços (por exemplo, cargas de trabalho stateless em comparação com cargas de trabalho stateful).
- Arquiteturas orientadas por eventos.
- Scaling horizontal e vertical.
- Como usar adequadamente os aceleradores de borda (por exemplo, rede de entrega de conteúdo [CDN]).
- Como migrar aplicações para contêineres.
- Conceitos de balanceamento de carga (por exemplo, Application Load Balancer).
- Arquiteturas multicamadas.
- Conceitos de enfileiramento e sistema de mensagens (por exemplo, publicar/assinar).
- Tecnologias e padrões sem servidor (por exemplo, AWS Fargate, AWS Lambda).
- Tipos de armazenamento com características associadas (por exemplo, objeto, arquivo, bloco).
- A orquestração de contêineres (por exemplo, Amazon Elastic Container Service [Amazon ECS], Amazon Elastic Kubernetes Service [Amazon EKS]).
- Quando usar réplicas de leitura.
- Orquestração de fluxo de trabalho (por exemplo, AWS Step Functions).

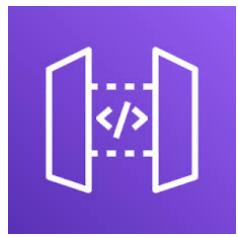
Semana 4

Habilidades em:

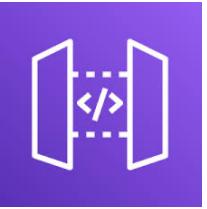
- Projetar arquiteturas orientadas por eventos, microsserviços e/ou multicamadas com base em requisitos.
- Determinar estratégias de scaling para componentes usados em um projeto de arquitetura.
- Determinar os serviços da AWS necessários para obter um acomplamento fraco com base em requisitos.
- Determinar quando usar contêineres.
- Determinar quando usar tecnologias e padrões sem servidor.
- Recomendar tecnologias apropriadas de computação, armazenamento, redes e banco de dados com base em requisitos.
- Usar serviços da AWS com propósito específico para cargas de trabalho.

Semana 4

- API Gateway
- SQS
- Cache
- Design para microsserviços
- Scaling horizontal vs vertical
- Aceleradores de borda
- Migrar aplicações para containers
- Balanceamento de carga (ELB)
- Serverless
- Orquestração de Containers (ECS e EKS)
- Réplicas de leitura
- Orquestração de fluxos de trabalho (Step Functions)



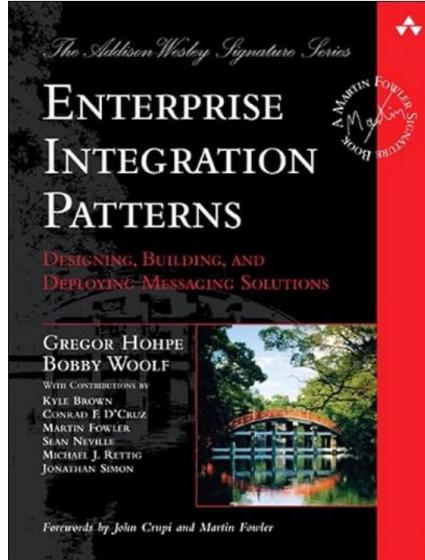
API Gateway



API Gateway

- Segurança: integra facilmente com o WAF
- Curva de aprendizado rápida para uso inicial
- Porta de entrada para REST APIs
- Porta de entrada para WebSocket API
- Pode ser utilizado com diversos serviços da AWS
- Para o exame de nível associate precisamos apenas de localização inicial do que é um api gateway (porta de entrada) e seus casos de uso. Não há necessidade de saber todos os detalhes.

Antes de falar de SQS e SNS é importante ressaltar a importância da temática de integrações de aplicações, desafio super atual e que tem muitos desafios a serem vencidos. Dica de leitura:



Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions 1st Edition

by [Gregor Hohpe](#) (Author), [Bobby Woolf](#) (Author)

4.6 532 ratings

[See all formats and editions](#)

Enterprise Integration Patterns provides an invaluable catalog of sixty-five patterns, with real-world solutions that demonstrate the formidable of messaging and help you to design effective messaging solutions for your enterprise.

The authors also include examples covering a variety of different integration technologies, such as JMS, MSMQ, TIBCO ActiveEnterprise, Microsoft BizTalk, SOAP, and XSL. A case study describing a bond trading system illustrates the patterns in practice, and the book offers a look at emerging standards, as well as insights into what the future of enterprise integration might hold.

This book provides a consistent vocabulary and visual notation framework to describe large-scale integration solutions across many technologies. It also explores in detail the advantages and limitations of asynchronous messaging architectures. The authors present practical advice on designing code that connects an application to a messaging system, and provide extensive information to help you determine when to send a message, how to route it to the proper destination, and how to monitor the health of a messaging system. If you want to know how to manage, monitor, and maintain a messaging system once it is in use, get this book.

[Report an issue with this product or seller](#)

ISBN-10

ISBN-13

Edition

Publisher

Publication date



#



October 10, 2003

9780321200686

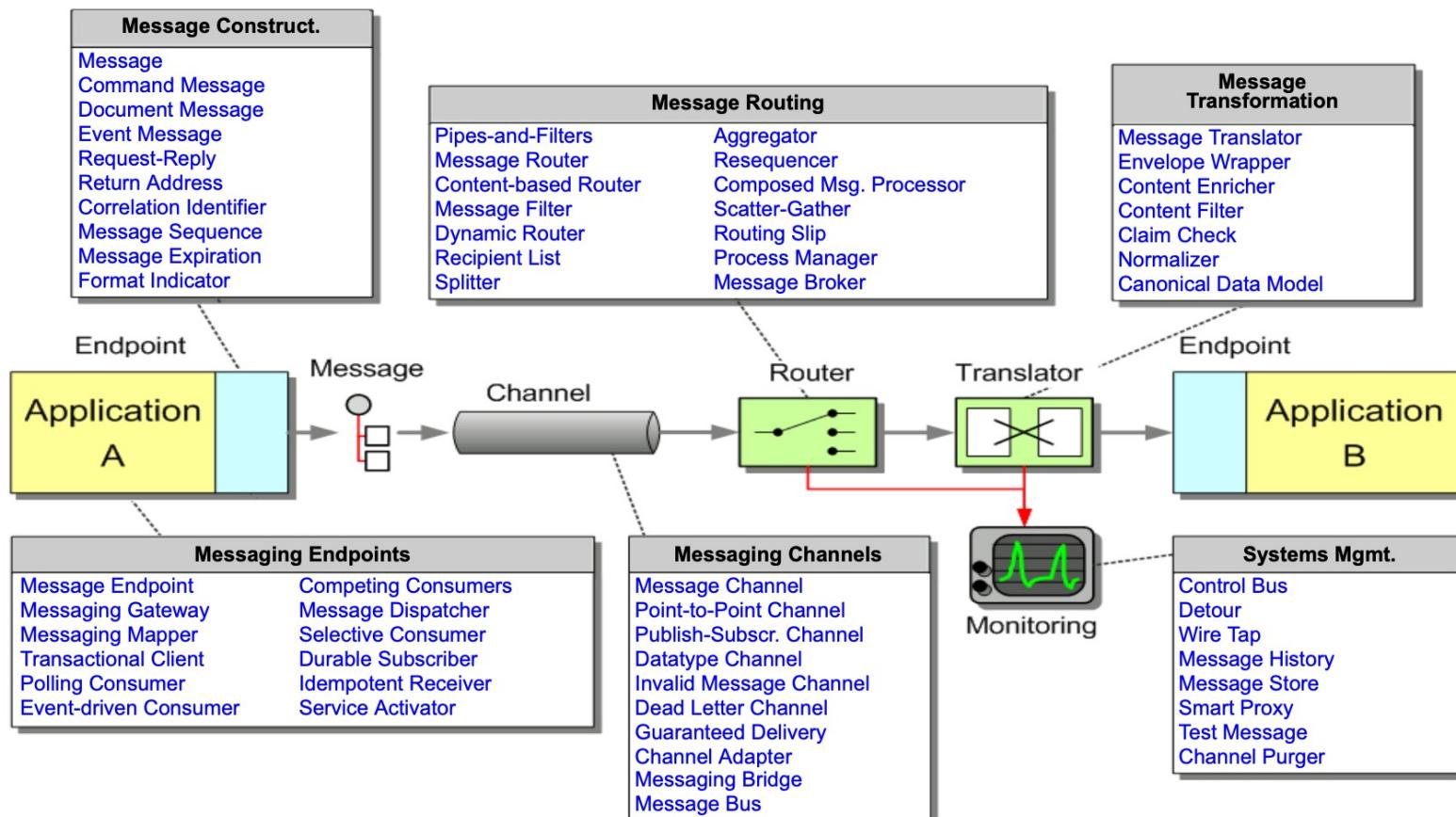
978-0321200686

1st

Addison-Wesley Professional

Site do livro - <https://www.enterpriseintegrationpatterns.com/patterns/messaging/>:

Integration Pattern Language



AWS re:Invent – Dica de conteúdo no YouTube

The image displays two YouTube video thumbnails side-by-side, both from the AWS re:Invent channel.

Top Video (API308):

- Title:** AWS re:Invent 2021 - Building modern cloud applications? Think integration
- Uploader:** AWS Events
- Views:** 10 mil visualizações
- Date:** há 2 anos
- Description:** Modern cloud applications based on serverless or microservices architectures are typically fine-grained, distributed...
- Length:** 55:59
- Thumbnail Content:** A dark blue background with white text: "Building modern cloud applications? Think integration". Below it, smaller text: "Gregor Hohpe Enterprise Strategist Amazon Web Services". At the bottom, the AWS logo and "55:59".

Bottom Video (API309):

- Title:** AWS re:Invent 2023 - Advanced integration patterns & trade-offs for loosely coupled systems (API309)
- Uploader:** AWS Events
- Views:** 21 mil visualizações
- Date:** há 8 meses
- Description:** Modern applications rarely live in isolation: They expose APIs, publish events, call third-party services, and externalize states.
- Length:** 59:12
- Thumbnail Content:** A dark blue background with white text: "Advanced integration patterns & trade-offs for loosely coupled systems". Below it, smaller text: "Dirk Fröhner Principal Solutions Architect Amazon Web Services" and "Gregor Hohpe Senior Principal Evangelist, Serverless Amazon Web Services". At the bottom, the AWS logo and "59:12".

- 1) https://youtu.be/ttJAIQf7cTw?si=rVM9u_e45Ms4rrqg
- 2) <https://youtu.be/FGKGdUiZKto?si=p2IYjgkrxbB6PHQO>

SQS

Simple Queue Service



Curiosidade: filas são estrutura de dados muito trabalhadas na computação ->
<https://www.ime.usp.br/~pf/estruturas-de-dados/aulas/queue.html>

Nessa descrição, o autor está falando de uma fila FIFO – First In First Out

Fila (= queue) e sua API

- Uma **fila** (= queue) é um **ADT** que consiste em uma coleção de coisas munida de duas operações: **enqueue**, que insere uma coisa na coleção, e **dequeue**, que remove a coisa mais antiga da coleção. (Portanto, a primeira coisa a entrar é a primeira a sair.)
- As coisas de que uma fila é feita serão chamadas *itens*. Os itens podem ser números, strings, structs, etc., etc.
- Exemplo: fila de pessoas esperando por atendimento no caixa de um banco.
- Eis a **API** de nossa fila de itens (além das operações fundamentais enqueue e dequeue, temos algumas operações auxiliares):

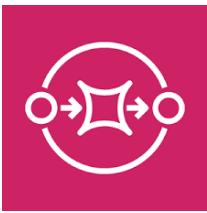
```
public class Queue<Item>
{
    Queue()
    void enqueue(Item item)
    Item dequeue()
    boolean isEmpty()
    int size()

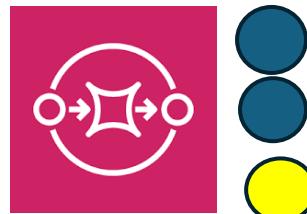
    construtor: cria uma fila de Items vazia
    coloca item nesta fila
    remove o item mais antigo desta fila
    esta fila está vazia?
    número de Items nesta fila
```

Perguntas iniciais e características importantes

- O fluxo é assícrono ou síncrono?
- A ordem de recebimento da mensagem importa?
- Quais são os limites de cada serviço que iremos abordar hj?
- SQS guarda mensagens por no máximo 14 dias
- Bom saber os valores que vem definidos por padrão







DLQ

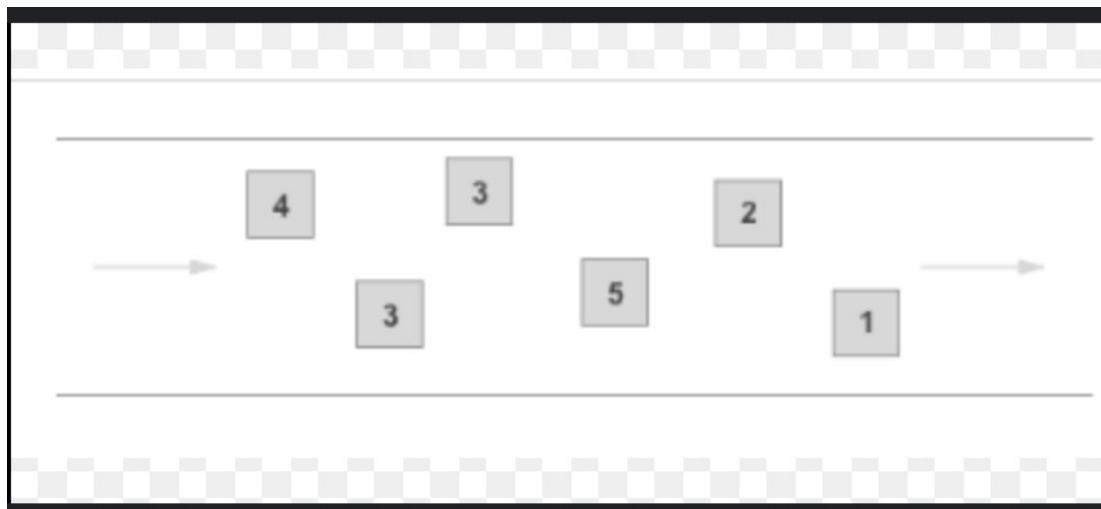
Lambda de reprocessamento

FIFO: First In First Out



<https://aws.amazon.com/blogs/compute/solving-complex-ordering-challenges-with-amazon-sqs-fifo-queues/>

Standard



Processamento em lote

Ações em lote do Amazon SQS

[PDF](#) | [RSS](#)

O Amazon SQS fornece ações em lote para ajudar você a reduzir custos e manipular até 10 mensagens com uma única ação. Essas ações em lote incluem:

- [SendMessageBatch](#)
- [DeleteMessageBatch](#)
- [ChangeMessageVisibilityBatch](#)

Usando ações em lote, você pode realizar várias operações em uma única chamada de API, o que ajuda a otimizar o desempenho e reduzir custos. Você pode aproveitar a funcionalidade em lote usando a API de consulta ou qualquer AWS SDK que ofereça suporte às ações em lote do Amazon SQS.

Detalhes importantes

- **Limite de tamanho de mensagem:** O tamanho total de todas as mensagens enviadas em uma única `SendMessageBatch` chamada não pode exceder 262.144 bytes (256 KiB).
- **Permissões:** você não pode definir permissões explicitamente para `SendMessageBatch` `DeleteMessageBatch`, ou `ChangeMessageVisibilityBatch`. Em vez disso, defina permissões para `SendMessageBatch` `DeleteMessageBatch`, ou `ChangeMessageVisibilityBatch` defina permissões para as versões em lote correspondentes das ações.
- **Supporte do console:** O console do Amazon SQS não oferece suporte a ações em lote. Você deve usar a API de consulta ou um AWS SDK para realizar operações em lote.

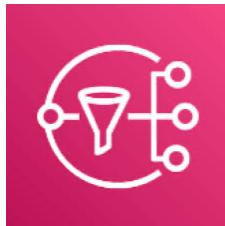
SNS

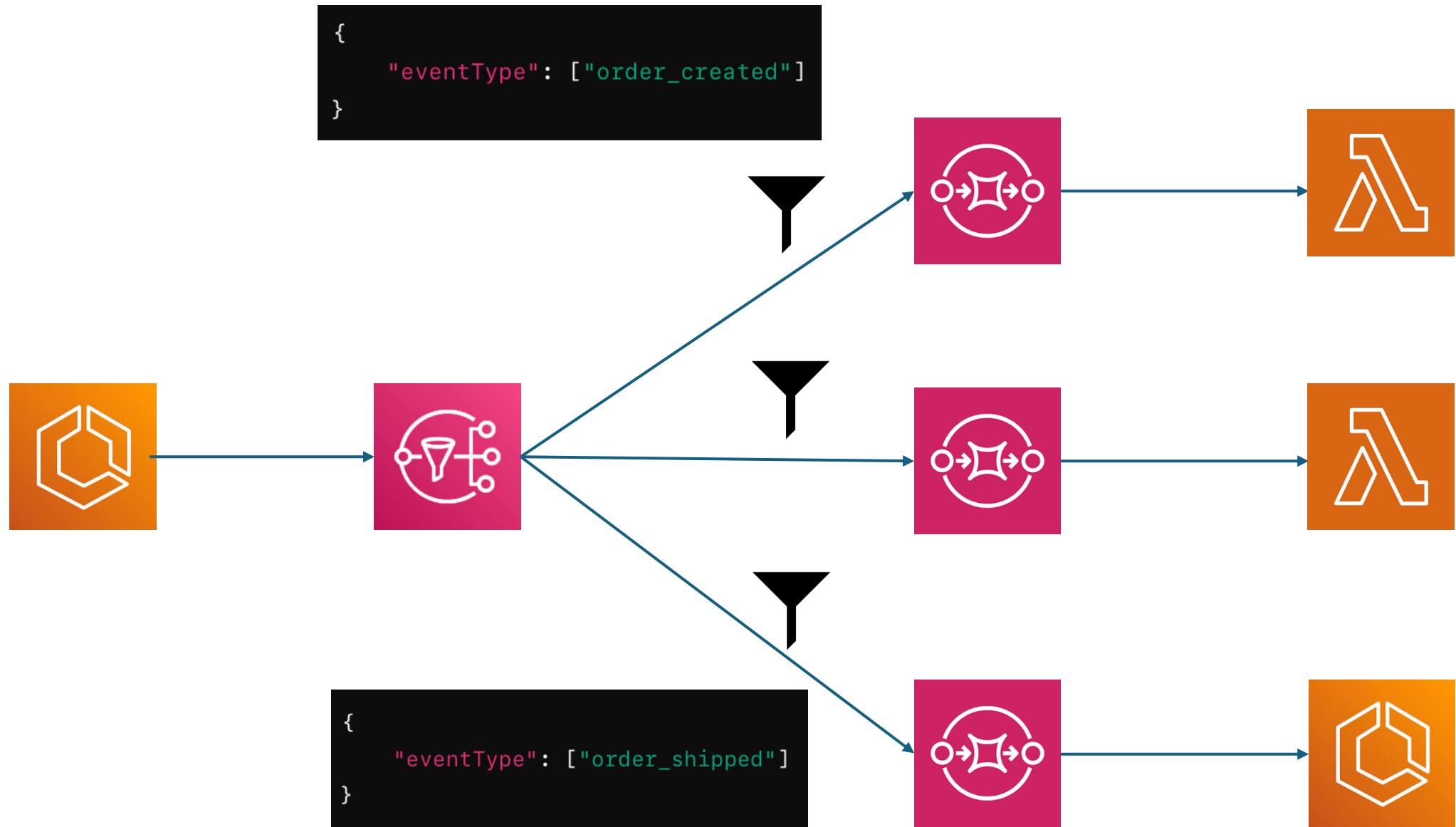
Simple Notification Service

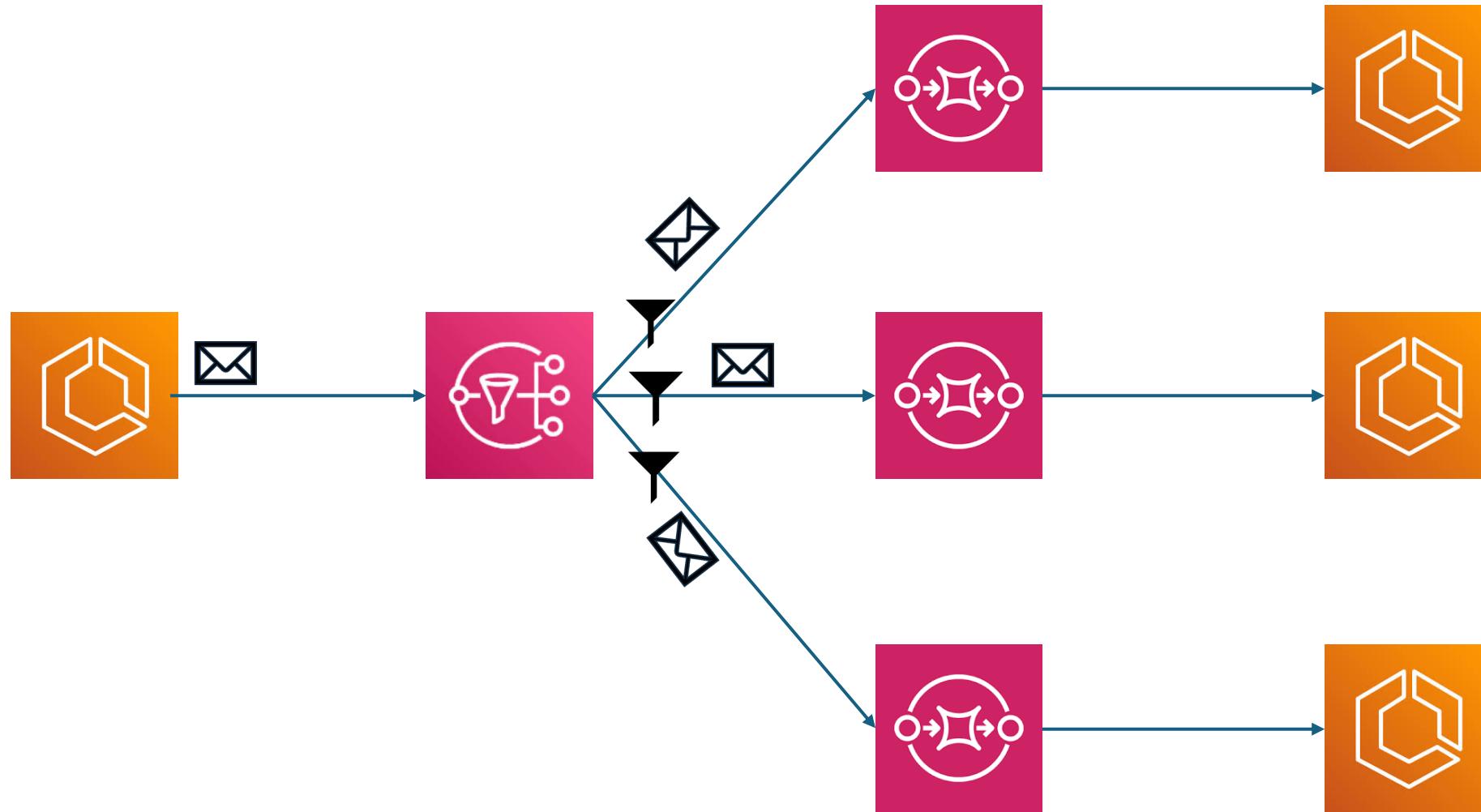


Padrão Arquitetural Comum: Fanout

- 1) https://docs.aws.amazon.com/pt_br/sns/latest/dg/sns-sqs-as-subscriber.html
- 2) https://docs.aws.amazon.com/pt_br/sns/latest/dg/sns-lambda-as-subscriber.html
- 3) https://docs.aws.amazon.com/pt_br/sns/latest/dg/sns-firehose-as-subscriber.html









- Email, texto e qualquer tipo de notificação do tipo Push
- SNS + SQS
- CloudWatch + SNS
- Lambda + SNS

EKS

Elastic Kubernetes Service





Kubernetes é uma plataforma de código-fonte aberto para a execução de aplicações em contêineres. O nome se origina da palavra grega para timoneiro ou piloto. A abreviação de Kubernetes, K8s, deriva da contagem de oito letras entre "K" e "s".

O Google abriu o código-fonte do projeto Kubernetes em 2014. O Kubernetes combina [mais de 15 anos de experiência do Google](#) executando cargas de trabalho de produção em escala com as melhores ideias e práticas da comunidade.

Antes da popularização da computação em nuvem, as organizações executavam aplicações em servidores físicos. Não havia como definir limites de recursos para aplicações em um servidor físico, o que causou problemas de alocação de recursos.

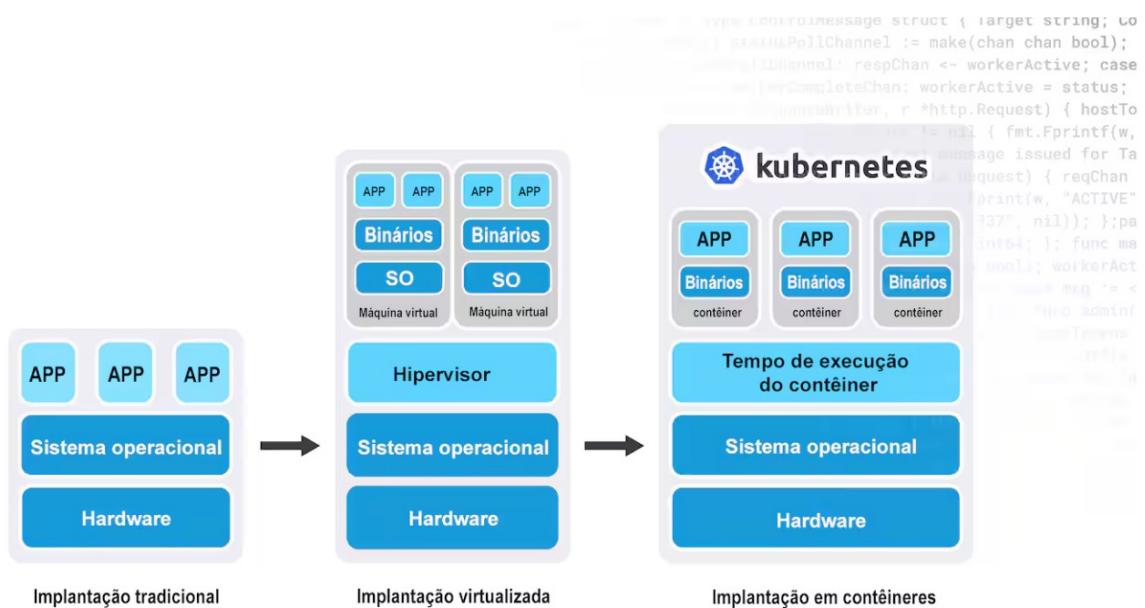


Tabela de Atributos Principais do AWS EKS

Atributo	Descrição
Gerenciamento de Kubernetes	AWS EKS é um serviço gerenciado que facilita a execução do Kubernetes na AWS, cuidando da operação e manutenção do plano de controle.
Integração com Serviços AWS	EKS se integra com outros serviços AWS, como IAM, VPC, ECR (Elastic Container Registry), CloudWatch, e ALB (Application Load Balancer).
Escalabilidade Automática	Suporta escalabilidade automática de pods e clusters com o Kubernetes Cluster Autoscaler e AWS Auto Scaling.
Segurança Avançada	Integra-se com IAM para autenticação e autorização, e suporta Network Policies para segurança de rede, além de AWS PrivateLink para clusters privados.
Suporte Multi-Zonas de Disponibilidade	Clusters podem ser distribuídos por múltiplas zonas de disponibilidade, aumentando a resiliência e disponibilidade dos serviços.
Atualizações Automáticas	Suporte para atualizações automáticas de versões do Kubernetes, permitindo que você mantenha seu ambiente sempre atualizado com os patches de segurança mais recentes.
Networking Flexível	Suporte para CNI (Container Network Interface) do Kubernetes, permitindo a escolha de plugins de rede e integração com serviços de rede da AWS.
Monitoramento e Logging	Integração com CloudWatch para monitoramento e logging, permitindo a coleta de métricas e logs para análise e diagnóstico.
Compatibilidade com Workloads On-Premises	Suporte para a execução de workloads híbridos com AWS Outposts, permitindo a execução de Kubernetes tanto na AWS quanto em ambientes on-premises.

Lambda





Atributo	Descrição
Execução Sem Servidor (Serverless)	AWS Lambda permite executar código sem a necessidade de gerenciar servidores, escalando automaticamente com base na carga de trabalho.
Eventos que Disparam Funções	Lambda pode ser disparado por vários eventos de serviços AWS, como S3, DynamoDB, API Gateway, SNS, SQS, CloudWatch, entre outros.
Escalabilidade Automática	Lambda escala automaticamente em resposta ao número de eventos recebidos, permitindo que milhões de solicitações sejam atendidas simultaneamente.
Pagamentos por Execução	Você só paga pelo tempo de execução das funções e pela quantidade de solicitações, tornando-o altamente econômico para muitas aplicações.
Gerenciamento de Estado (Stateless)	Cada execução de função Lambda é independente, o que significa que o serviço é stateless, necessitando de armazenamento externo para manter estado.
Segurança e Controle de Acesso	Lambda integra-se com AWS IAM para controle refinado de permissões e com AWS KMS para criptografia de dados sensíveis em trânsito e em repouso.
Tempo de Execução e Memória	As funções Lambda podem ser configuradas com até 15 minutos de tempo de execução e entre 128 MB e 10 GB de memória.
Limites de Conexões e Tamanho de Payload	O tamanho máximo do payload de invocação síncrona é 6 MB (payload de evento) e 256 KB para invocação assíncrona; conexões de rede são permitidas apenas de saída.
Supporte a Vários Idiomas	Lambda suporta várias linguagens, incluindo Python, Node.js, Java, Go, Ruby, .NET Core, e pode executar executáveis personalizados através do uso de layers ou container images.
Criação de APIs	Combinado com o API Gateway, Lambda permite a criação de APIs RESTful e WebSocket, facilitando o desenvolvimento de aplicações serverless completas.



Limites

Limites do AWS Lambda

Limite	Valor
Tempo Máximo de Execução	15 minutos
Memória Disponível	128 MB a 10 GB
Tamanho Máximo do Pacote de Deploy (ZIP)	50 MB (direto) ou 250 MB (através de S3)
Tamanho Máximo de Payload de Requisição	6 MB para invocação síncrona, 256 KB para invocação assíncrona
Conexões de Rede	Apenas conexões de saída são permitidas (não pode aceitar conexões de entrada)
Tamanho Máximo de Logs	Logs gerados pela função Lambda são limitados a 16 KB por invocação
Número Máximo de Funções Simultâneas	1.000 invocações simultâneas por padrão (pode ser aumentado sob solicitação)



Otimização

Estratégia	Descrição
Aumento de Memória (Aumenta CPU)	No AWS Lambda, a CPU é proporcional à memória alocada. Para aumentar a CPU, você deve aumentar a memória da função.
Otimização do Código	Simplifique algoritmos, minimize loops e utilize estruturas de dados eficientes.
Gerenciamento de Conexões	Reutilize conexões de rede e minimize chamadas externas, combinando requisições quando possível.
Caching de Resultados	Implemente cache em memória para resultados frequentes e utilize DAX para acelerar acessos ao DynamoDB.
Pacote de Implantação Otimizado	Reduza o tamanho do pacote para minimizar o tempo de carregamento e inicialização.
Lambda Layers	Use Layers para compartilhar bibliotecas e dependências comuns entre funções, reduzindo o tamanho do pacote de implantação.
Configuração de VPC	Evite colocar a função em uma VPC, a menos que seja necessário, para reduzir a latência e o overhead.
Parallelismo e Processamento Assíncrono	Utilize processamento paralelo dentro da função e delegue trabalho a outros serviços, como Step Functions.
Dividir Funções em Módulos Menores	Crie microserviços dividindo a lógica em funções menores e independentes para reduzir a carga em cada função.
Pré-Compilação e Otimizações	Pré-processe ou pré-compile tarefas que podem ser feitas antes da execução da função para economizar tempo.
Provisioned Concurrency	Configure concurrency provisionada para reduzir o impacto de cold starts e melhorar a latência de funções críticas.
Avaliação de Alternativas	Considere o uso de EC2 ou Fargate para tarefas que exigem muita CPU ou são inadequadas para Lambda.



Otimização



IMPORTANTE Não controlamos diretamente a CPU dedicada a uma função lambda!

Memory and computing power

Memory is the principal lever available to Lambda developers for controlling the performance of a function. You can configure the amount of memory allocated to a Lambda function, between 128 MB and 10,240 MB. The Lambda console defaults new functions to the smallest setting and many developers also choose 128 MB for their functions.

However, 128 MB should usually only be used for the simplest of Lambda functions, such as those that transform and route events to other AWS services. If the function imports libraries or Lambda layers, or interacts with data loaded from S3 or EFS, it's likely to be more performant with a higher memory allocation.

The amount of memory also determines the amount of virtual CPU available to a function. Adding more memory proportionally increases the amount of CPU, increasing the overall computational power available. If a function is CPU-, network- or memory-bound, then changing the memory setting can dramatically improve its performance.

https://docs.aws.amazon.com/pt_br/lambda/latest/operatorguide/computing-power.html



Lambda: Nome tem uso comum na matemática e é comumente usado com sinônimo de função na computação

The screenshot shows the Stanford Encyclopedia of Philosophy website. The header features the SEP logo (a stylized Greek letter phi) and the text "Stanford Encyclopedia of Philosophy". Below the header are navigation links: "Browse", "About", and "Support SEP". A search bar is also present. On the left, a sidebar contains links to "Entry Contents", "Bibliography", "Academic Tools", "Friends PDF Preview" (with a red "X" icon), "Author and Citation Info" (with a red "X" icon), and "Back to Top" (with a red "X" icon). The main content area is titled "The Lambda Calculus" and includes a timestamp: "First published Wed Dec 12, 2012; substantive revision Tue Jul 25, 2023". The text describes the λ-calculus as a notation for functions and application, highlighting its sparse syntax, abstraction, and application rules.

Entry Contents
Bibliography
Academic Tools
Friends PDF Preview
Author and Citation Info
Back to Top

The Lambda Calculus

First published Wed Dec 12, 2012; substantive revision Tue Jul 25, 2023

The λ -calculus is, at heart, a simple notation for functions and application. The main ideas are *applying* a function to an argument and forming functions by *abstraction*. The syntax of basic λ -calculus is quite sparse, making it an elegant, focused notation for representing functions. Functions and arguments are on a par with one another. The result is a non-extensional theory of functions as rules of computation, contrasting with an extensional theory of functions as sets of ordered pairs. Despite its sparse syntax, the expressiveness and flexibility of the λ -calculus make it a cornucopia of logic and mathematics. This entry develops some of the central highlights of the field and prepares the reader for further study of the subject and its applications in philosophy, linguistics, computer science, and logic.

<https://plato.stanford.edu/entries/lambda-calculus/>

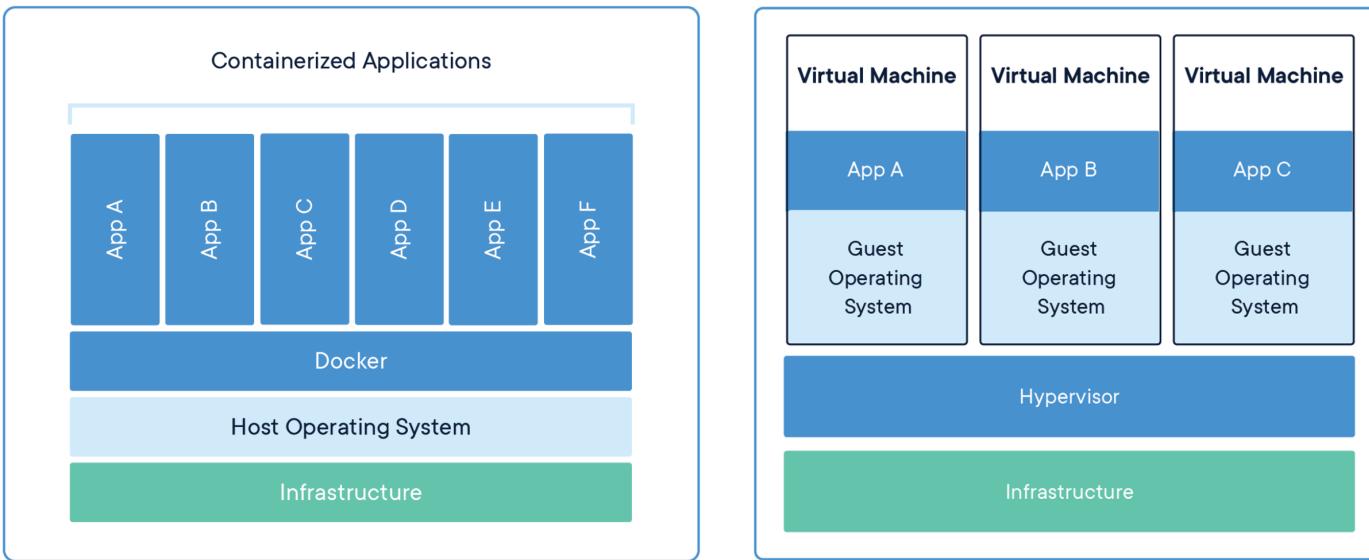
ECS

Elastic Container Service





O que é um container?



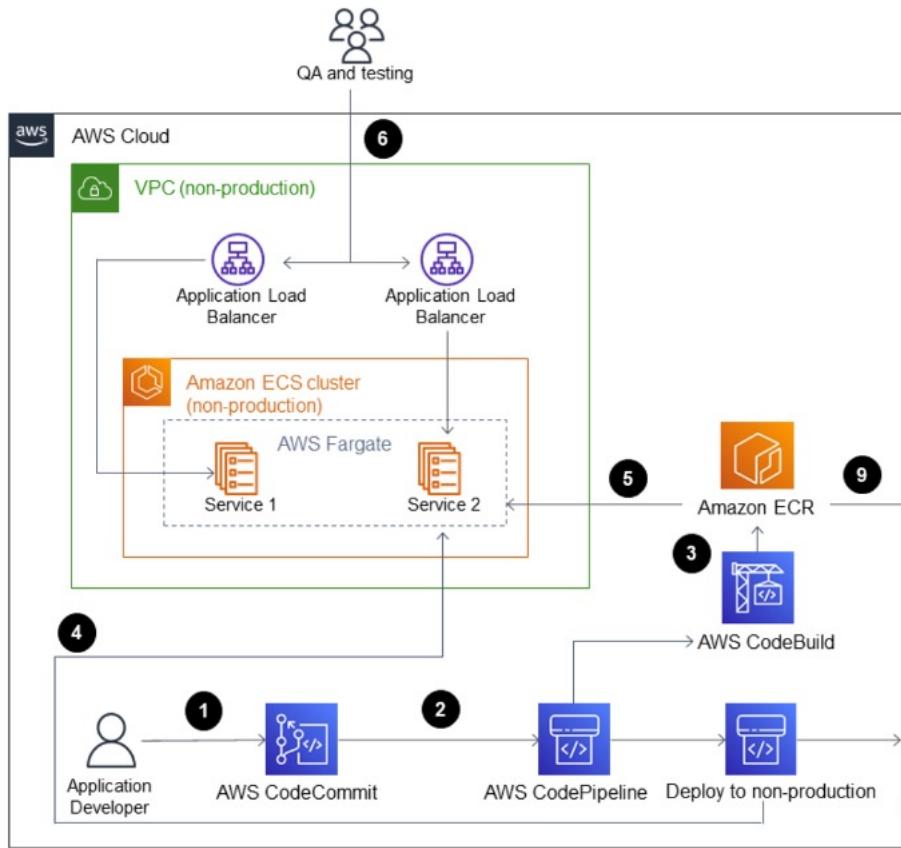
CONTAINERS

Containers are an abstraction at the app layer that packages code and dependencies together. Multiple containers can run on the same machine and share the OS kernel with other containers, each running as isolated processes in user space. Containers take up less space than VMs (container images are typically tens of MBs in size), can handle more applications and require fewer VMs and Operating systems.

VIRTUAL MACHINES

Virtual machines (VMs) are an abstraction of physical hardware turning one server into many servers. The hypervisor allows multiple VMs to run on a single machine. Each VM includes a full copy of an operating system, the application, necessary binaries and libraries – taking up tens of GBs. VMs can also be slow to boot.





- A imagem docker do app é registrada no ECR: Elastic Container Registry.
- O ECS utilizará essa imagem para fazer deploy de tarefas no seu serviço.
- Legal guardarmos já alguns nomes como serviço, cluster e tarefa no contexto do ECS.

<https://aws.amazon.com/blogs/architecture/its-adopts-microservices-architecture-for-improved-air-travel-search-engine/>

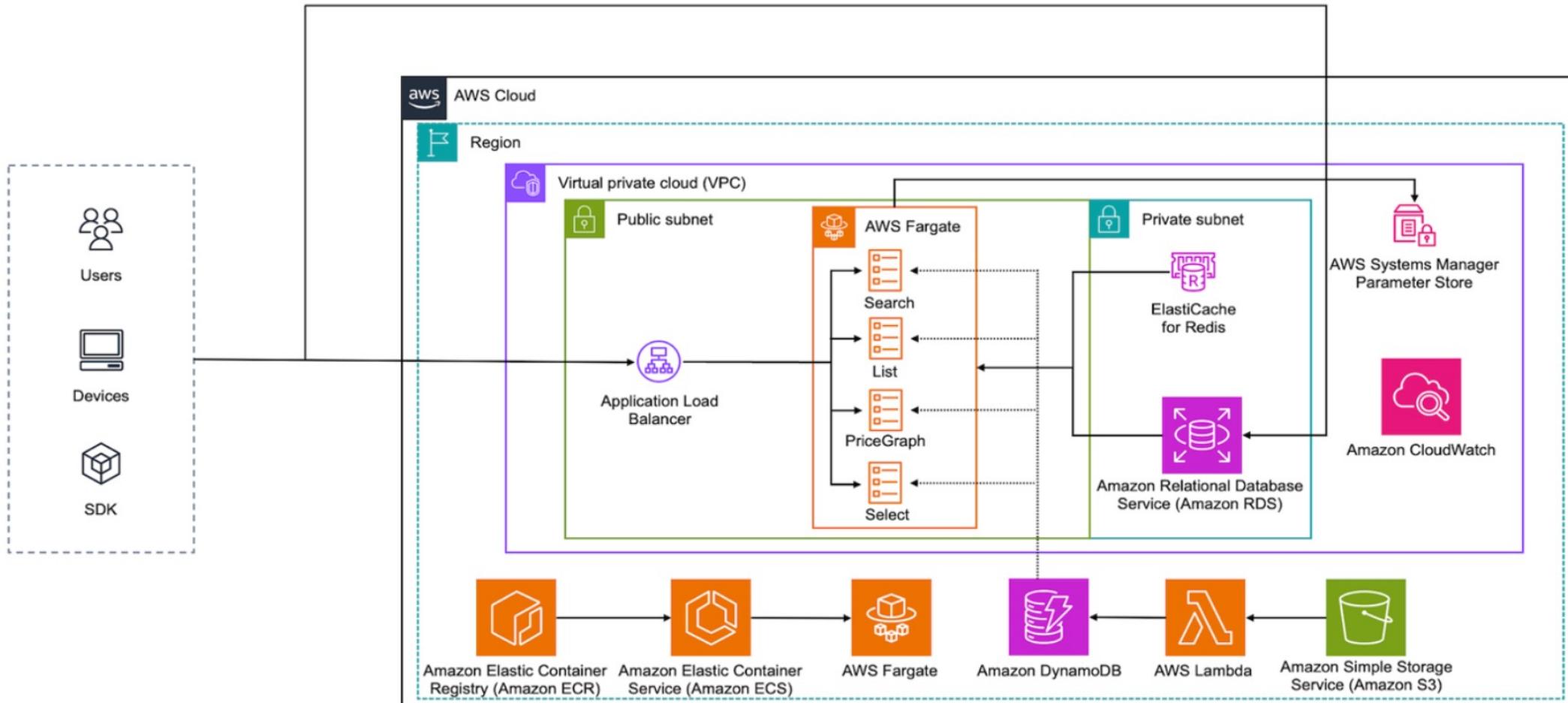


Figure 1. ITS' microservice architecture, using AWS

1. Cluster

- **Definição:** Um cluster é um agrupamento lógico de recursos de infraestrutura, que pode incluir instâncias EC2 (no modo EC2) ou servidores gerenciados pela AWS (no modo Fargate), onde os containers são executados.
- **Como Funciona:** Um cluster agrupa todos os recursos necessários para executar suas tarefas e serviços. Pode ser gerenciado diretamente pelo ECS e pode conter várias instâncias EC2 ou recursos do Fargate, onde seus containers serão implantados.

2. Task Definition

- **Definição:** Uma task definition (definição de tarefa) é uma especificação de como um conjunto de containers deve ser executado. Ela define parâmetros como a imagem do Docker, as variáveis de ambiente, a quantidade de CPU e memória alocada, além de volumes e políticas de reinício.
- **Como Funciona:** A task definition é o "plano" para uma aplicação containerizada. Quando você executa uma tarefa ou serviço, o ECS usa a definição de tarefa para iniciar os containers especificados.

3. Task

- **Definição:** Uma task (tarefa) é uma instância em execução de uma task definition. É o processo de execução dos containers conforme definido na task definition.
- **Como Funciona:** Quando você lança uma tarefa, o ECS cria uma instância de task usando a definição de tarefa especificada. Essa task pode ser executada até ser concluída ou ser encerrada manualmente.

4. Service

- **Definição:** Um service (serviço) é uma maneira de executar e manter um número especificado de tasks em execução de forma contínua dentro de um cluster.
- **Como Funciona:** O ECS garante que o número desejado de tasks esteja sempre em execução para um serviço. Se uma task falhar ou for encerrada, o ECS inicia automaticamente outra task para substituí-la, garantindo alta disponibilidade.

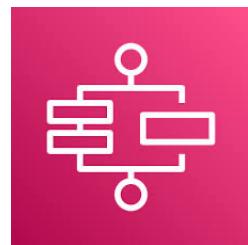


Launch Type

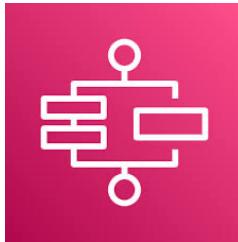
Fargate Launch Type: menor
gerenciamento

EC2 Launch Type

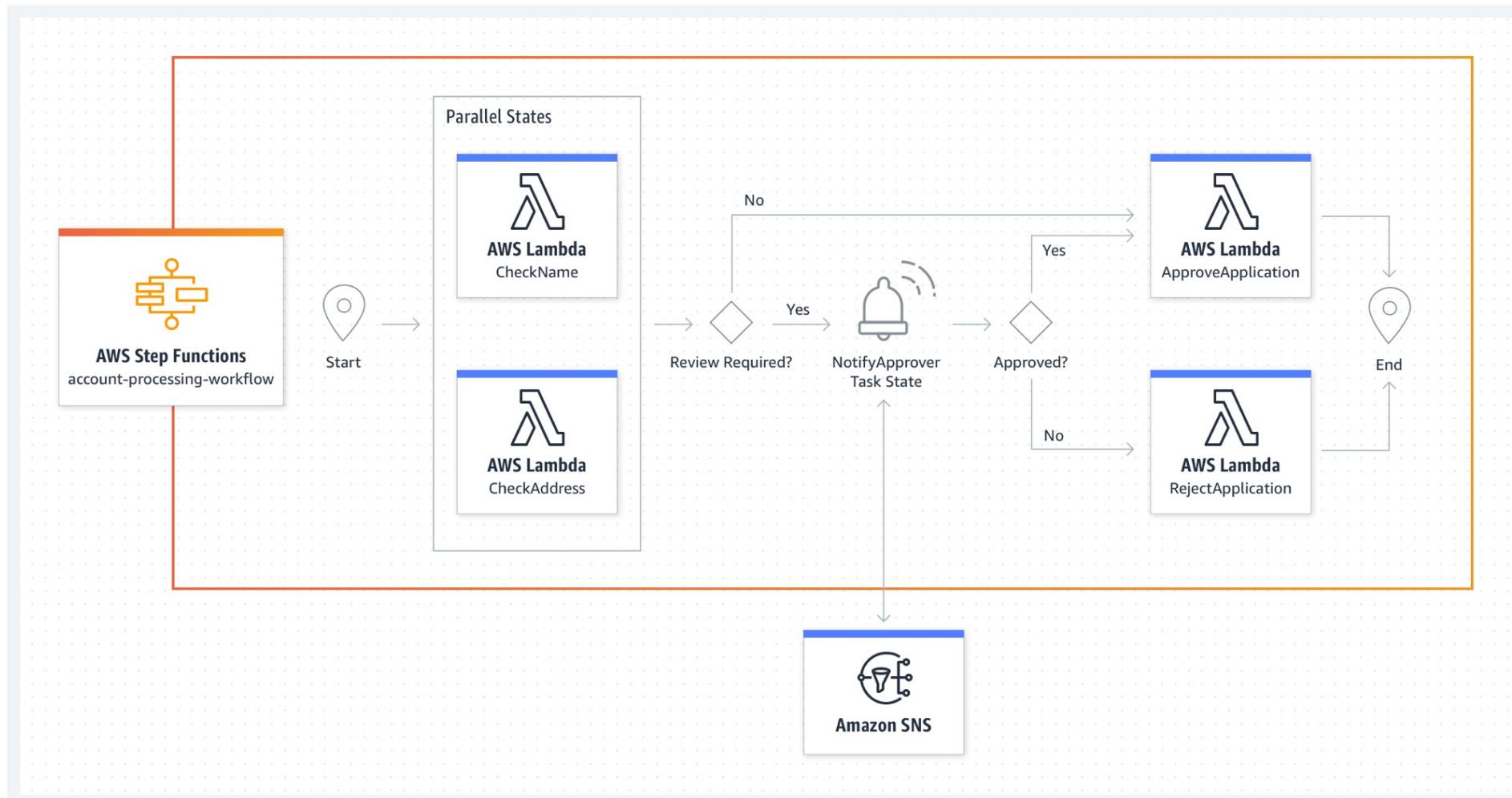
AWS Step Functions

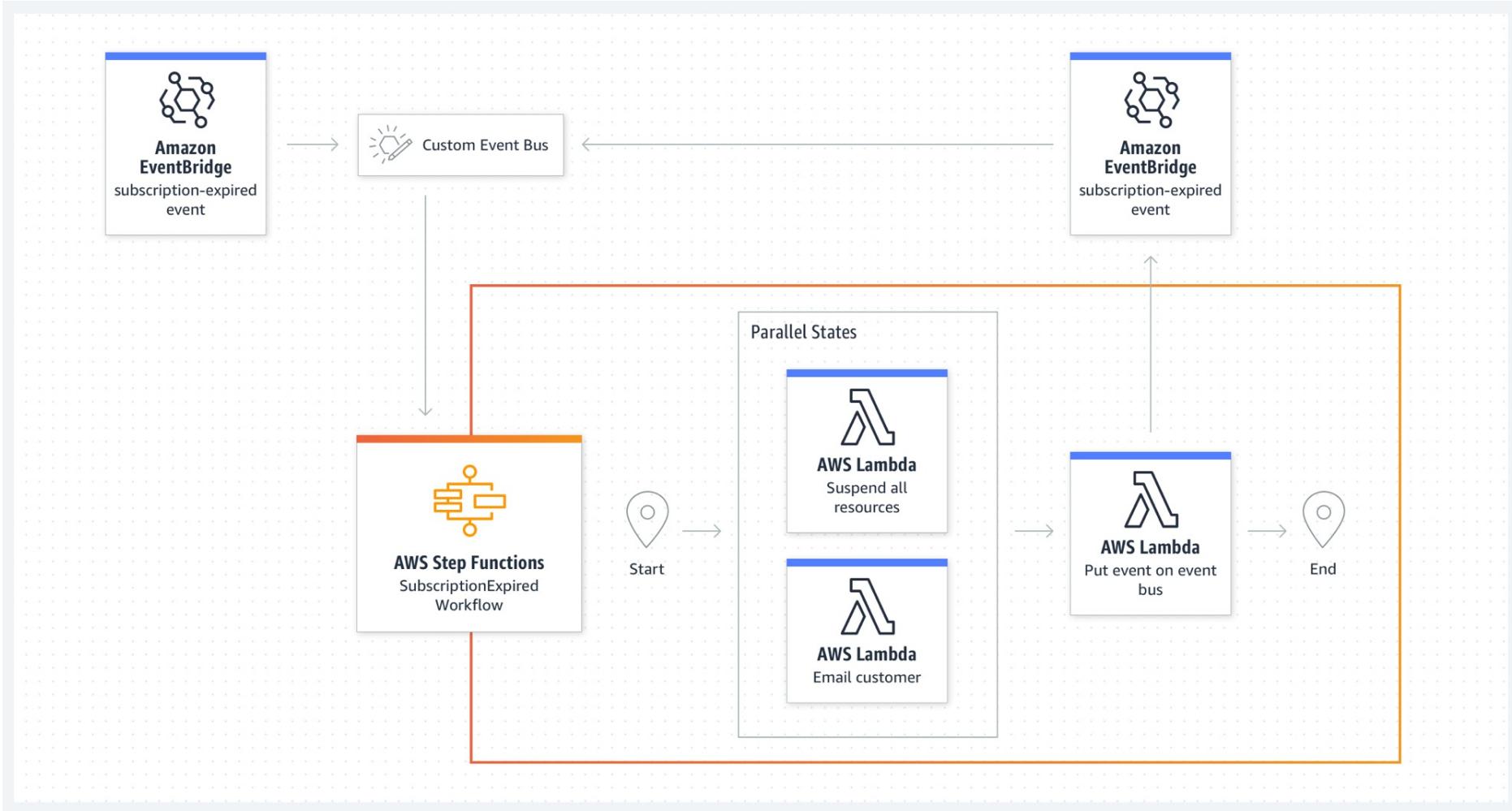
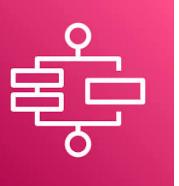


AWS Step Functions



Palavra-chave: Orquestração





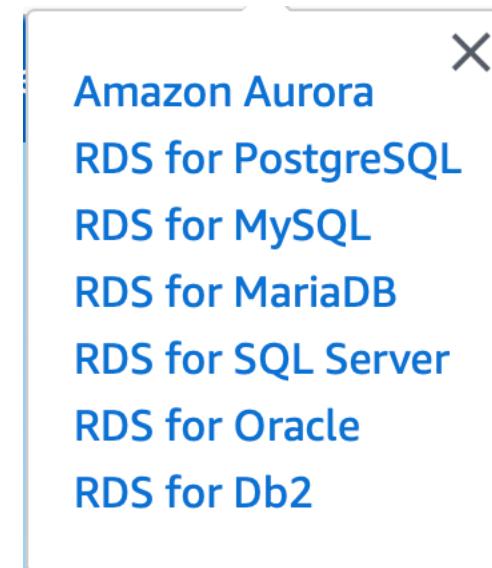
RDS

Relational Database Service





Engines suportadas

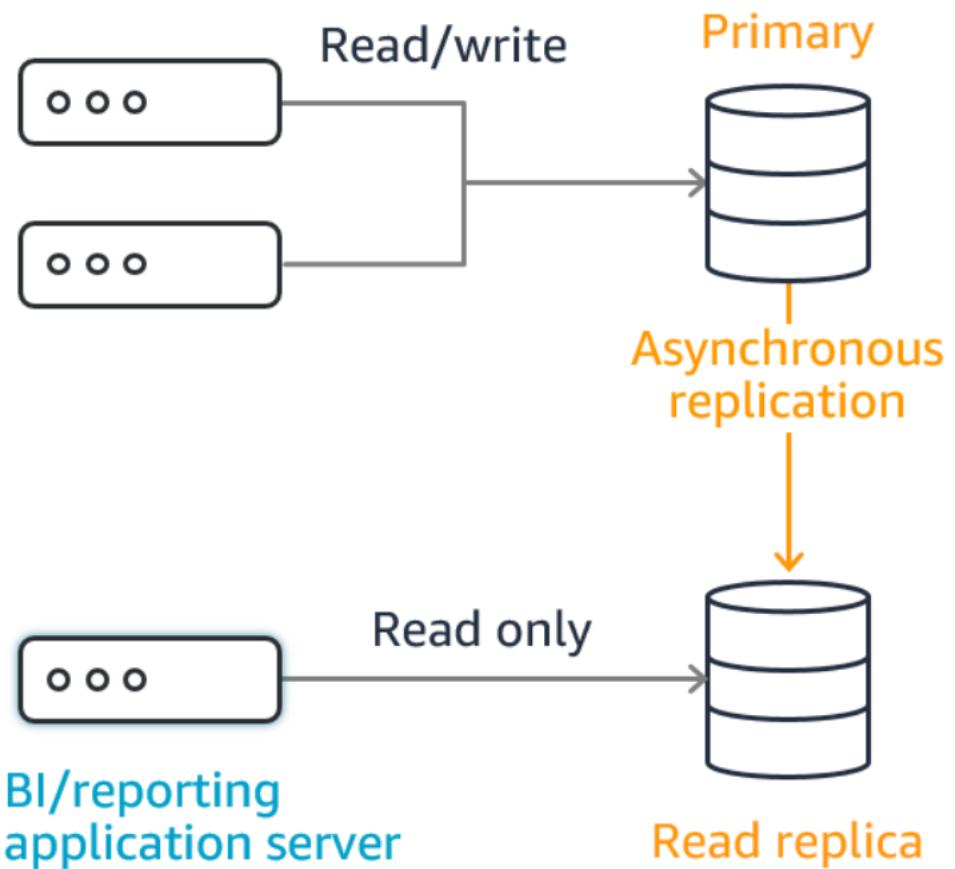




Read Réplica

Replicação assíncrona

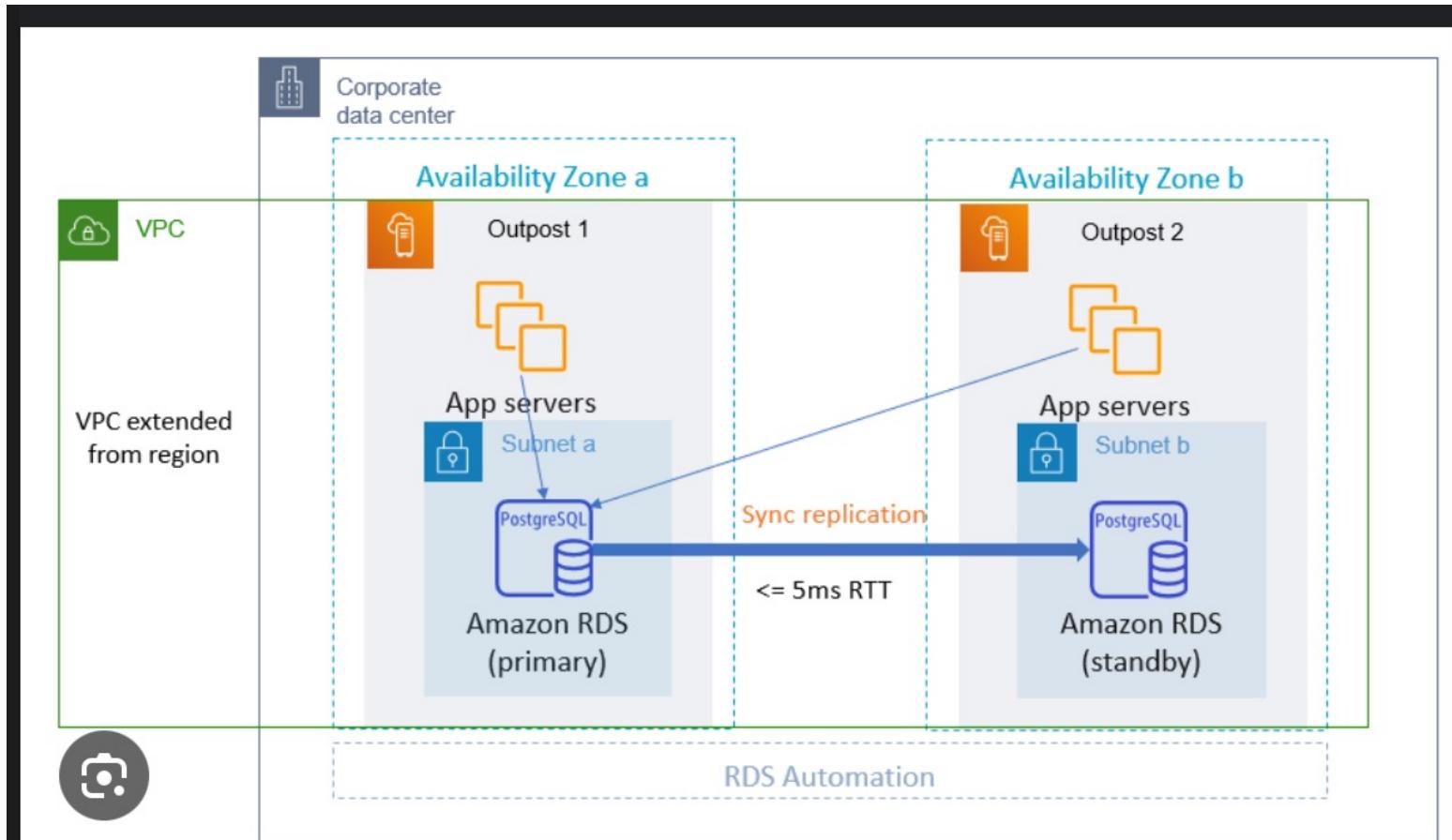
Application servers Database server





Multi-AZ

Replicação síncrona





IMPORTANTE

Read Réplica

Replicação assíncrona

vs

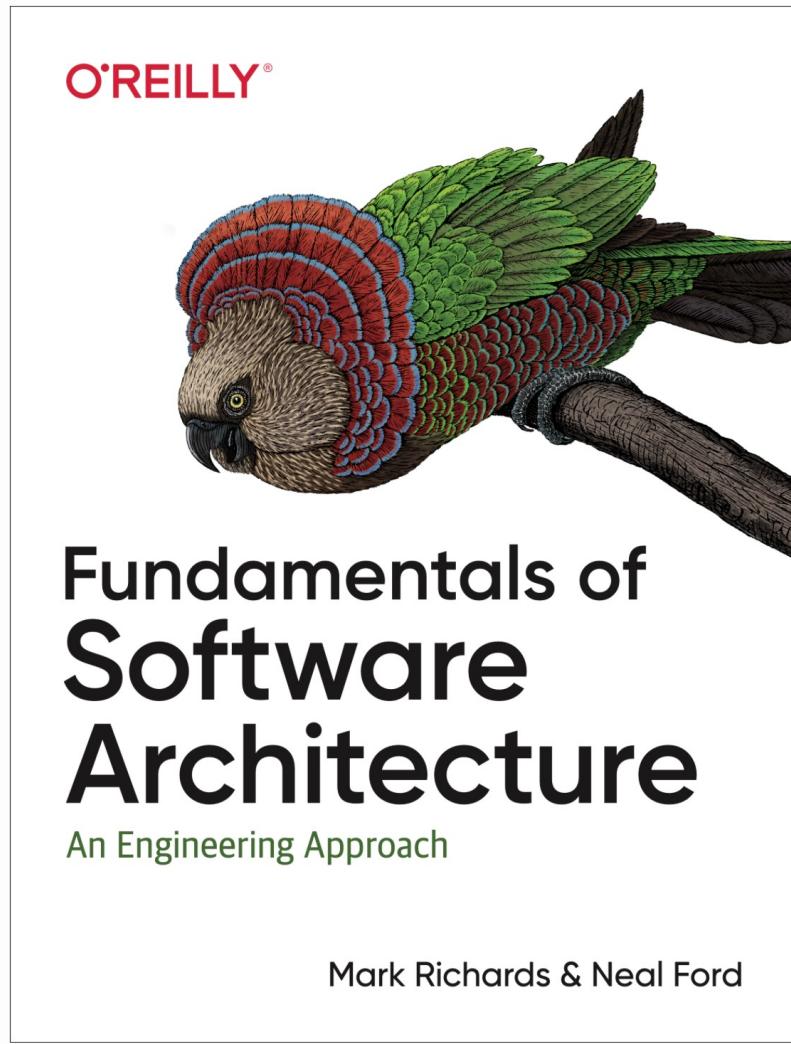
Multi-AZ

Replicação síncrona



- Quais as vantagens de utilizar o RDS? Podemos pensar em uma das ideias centrais da computação em nuvem que é reduzir a carga de gerenciamento da infraestrutura de algum serviço. Vamos pensar num cenário on-premise em que eu precisaria disponibilizar uma máquina para hospedar um banco de dados. O trabalho e custo com gerenciamento certamente é bem maior. Isso não quer dizer que uma solução é melhor que a outra, apenas que algumas situações podem demandar um serviço gerenciado ou não e cada contexto terá seus prós e contras.

Mark Richards & Neal Ford



- “Everything in software architecture is a trade-off”

ElastiCache





- 1- Redis
- 2- MemCached



1- Redis

Atributo	Descrição
Persistência de Dados	Com Redis no ElastiCache, você pode optar por persistir dados em disco usando snapshots (RDB) e logs de operações (AOF), garantindo a recuperação após falhas.
Replicação e Failover Automático	Redis no ElastiCache suporta replicação entre várias zonas de disponibilidade (Multi-AZ) com failover automático, garantindo alta disponibilidade dos dados.
Supporte a Clustering	Redis permite o uso de clusters com sharding, o que permite escalabilidade horizontal e distribuição de dados em diferentes nós, melhorando a capacidade e a performance.
Tipos de Dados Avançados	Redis suporta tipos de dados complexos como strings, listas, conjuntos, hashes, e sorted sets, permitindo soluções sofisticadas de armazenamento em cache e processamento de dados.
Pub/Sub	Redis no ElastiCache oferece suporte para o padrão de publicação/assinatura (Pub/Sub), permitindo a criação de sistemas de mensagens em tempo real para comunicação entre serviços.
Backup e Restauração	Você pode criar backups automatizados e manuais dos seus clusters Redis, permitindo a restauração rápida dos dados em caso de necessidade.
Monitoramento e Segurança	Com integração ao Amazon CloudWatch, Redis no ElastiCache permite o monitoramento detalhado de métricas, além de oferecer suporte a criptografia de dados em trânsito e em repouso.
Controle de Acesso (IAM e VPC)	Redis no ElastiCache integra-se com IAM e VPC para oferecer controle refinado sobre quem pode acessar os clusters e de onde, garantindo segurança no acesso aos dados.
Auto Scaling	O ElastiCache com Redis pode ser configurado para auto scaling, ajustando automaticamente a capacidade do cluster com base na carga de trabalho.

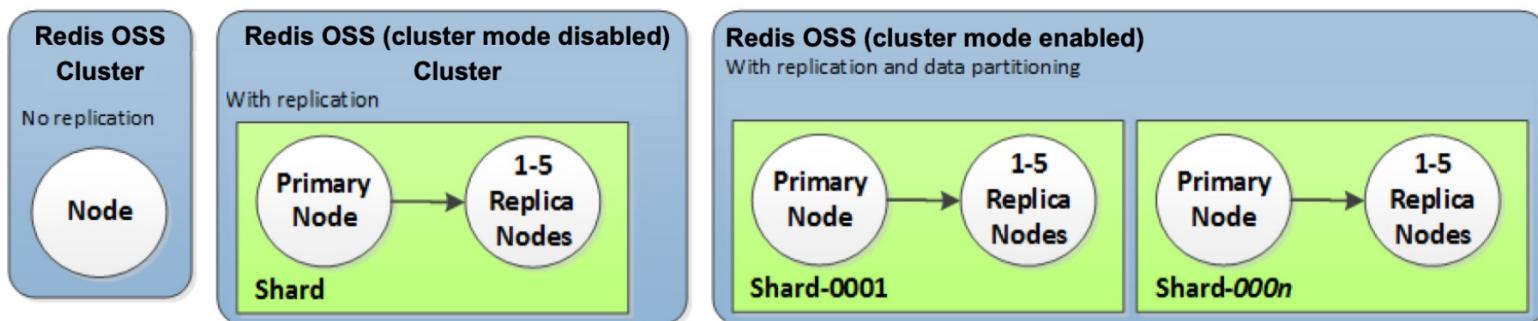


1- Redis

- TTL: Time to Live -> tempo até “limpar” algumas chaves de entrada no cache
- Podemos pensar no Redis como um pouco mais robusto que o MemCached
- Redis tem suporte a Backups
- Pode ser utilizado em Cluster (grupo)

Quando você possui múltiplos nós em um estilhaço, um dos nós é um nó primário de leitura/gravação. Todos os outros nós no estilhaço são réplicas somente leitura.

Os clusters Redis OSS típicos têm a seguinte aparência.





2- MemCached

Atributo	Descrição
Alto Desempenho e Baixa Latência	Memcached é extremamente rápido e leve, oferecendo latência muito baixa para operações de leitura e gravação, ideal para caching de dados frequentemente acessados.
Escalabilidade Horizontal	Memcached permite a adição e remoção de nós facilmente, suportando escalabilidade horizontal e balanceamento de carga sem interrupção do serviço.
Simplicidade e Eficiência	Com uma arquitetura simples e eficiente, Memcached é fácil de configurar e gerenciar, focado exclusivamente em caching de pares chave-valor.
Suporte para Cache Distribuído	Memcached distribui automaticamente os dados entre múltiplos nós, permitindo o cache de grandes volumes de dados com consistência e balanceamento.
Sem Persistência de Dados	Memcached é voltado para caching em memória volátil, sem suporte para persistência, o que é útil para aplicações onde a durabilidade dos dados não é crítica.
Gerenciamento de Memória	Memcached utiliza uma estratégia de gerenciamento de memória baseada em slabs, otimizando a utilização de memória e minimizando a fragmentação.
Compatibilidade Multi-thread	Memcached suporta multi-threading, o que permite o uso eficiente de recursos em sistemas multi-core, aumentando a capacidade de processamento simultâneo.
Solução Open-Source	Sendo uma solução amplamente adotada e open-source, Memcached é suportado por uma grande comunidade, com um ecossistema de ferramentas e bibliotecas.
Monitoramento e Segurança	Integrado com o Amazon CloudWatch para monitoramento de métricas e VPC para isolamento de rede, garantindo segurança e visibilidade do desempenho do cache.
Controle de Acesso	Utiliza políticas de IAM e VPC para controlar o acesso aos clusters, oferecendo segurança e controle refinado sobre quem pode acessar os dados cacheados.



2- MemCached

- Um cache mais simples, com o foco em ser apenas um cache básico mesmo



Estratégias de cache

Write-through

The write-through strategy adds data or updates data in the cache whenever data is written to the database.

Advantages and disadvantages of write-through

The advantages of write-through are as follows:

- Data in the cache is never stale.

Because the data in the cache is updated every time it's written to the database, the data in the cache is always current.

- Write penalty vs. read penalty.

Every write involves two trips:

1. A write to the cache
2. A write to the database

Which adds latency to the process. That said, end users are generally more tolerant of latency when updating data than when retrieving data. There is an inherent sense that updates are more work and thus take longer.

The disadvantages of write-through are as follows:

- Missing data.

If you spin up a new node, whether due to a node failure or scaling out, there is missing data. This data continues to be missing until it's added or updated on the database. You can minimize this by implementing [lazy loading](#) with write-through.

- Cache churn.

Most data is never read, which is a waste of resources. By [adding a time to live \(TTL\) value](#), you can minimize wasted space.



Estratégias de cache

Lazy loading

As the name implies, *lazy loading* is a caching strategy that loads data into the cache only when necessary. It works as described following.

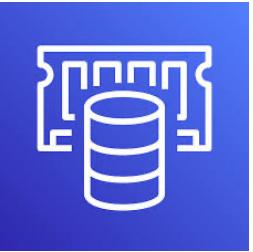
Amazon ElastiCache is an in-memory key-value store that sits between your application and the data store (database) that it accesses. Whenever your application requests data, it first makes the request to the ElastiCache cache. If the data exists in the cache and is current, ElastiCache returns the data to your application. If the data doesn't exist in the cache or has expired, your application requests the data from your data store. Your data store then returns the data to your application. Your application next writes the data received from the store to the cache. This way, it can be more quickly retrieved the next time it's requested.

A *cache hit* occurs when data is in the cache and isn't expired:

1. Your application requests data from the cache.
2. The cache returns the data to the application.

A *cache miss* occurs when data isn't in the cache or is expired:

1. Your application requests data from the cache.
2. The cache doesn't have the requested data, so returns a `null`.
3. Your application requests and receives the data from the database.
4. Your application updates the cache with the new data.



Estratégias de cache



IMPORTANTE

Adding TTL

Lazy loading allows for stale data but doesn't fail with empty nodes. Write-through ensures that data is always fresh, but can fail with empty nodes and can populate the cache with superfluous data. By adding a time to live (TTL) value to each write, you can have the advantages of each strategy. At the same time, you can largely avoid cluttering up the cache with extra data.

Time to live (TTL) is an integer value that specifies the number of seconds until the key expires. Memcached specifies this value in seconds. When an application attempts to read an expired key, it is treated as though the key is not found. The database is queried for the key and the cache is updated. This approach doesn't guarantee that a value isn't stale. However, it keeps data from getting too stale and requires that values in the cache are occasionally refreshed from the database.

For more information, see the [Memcached set command](#).

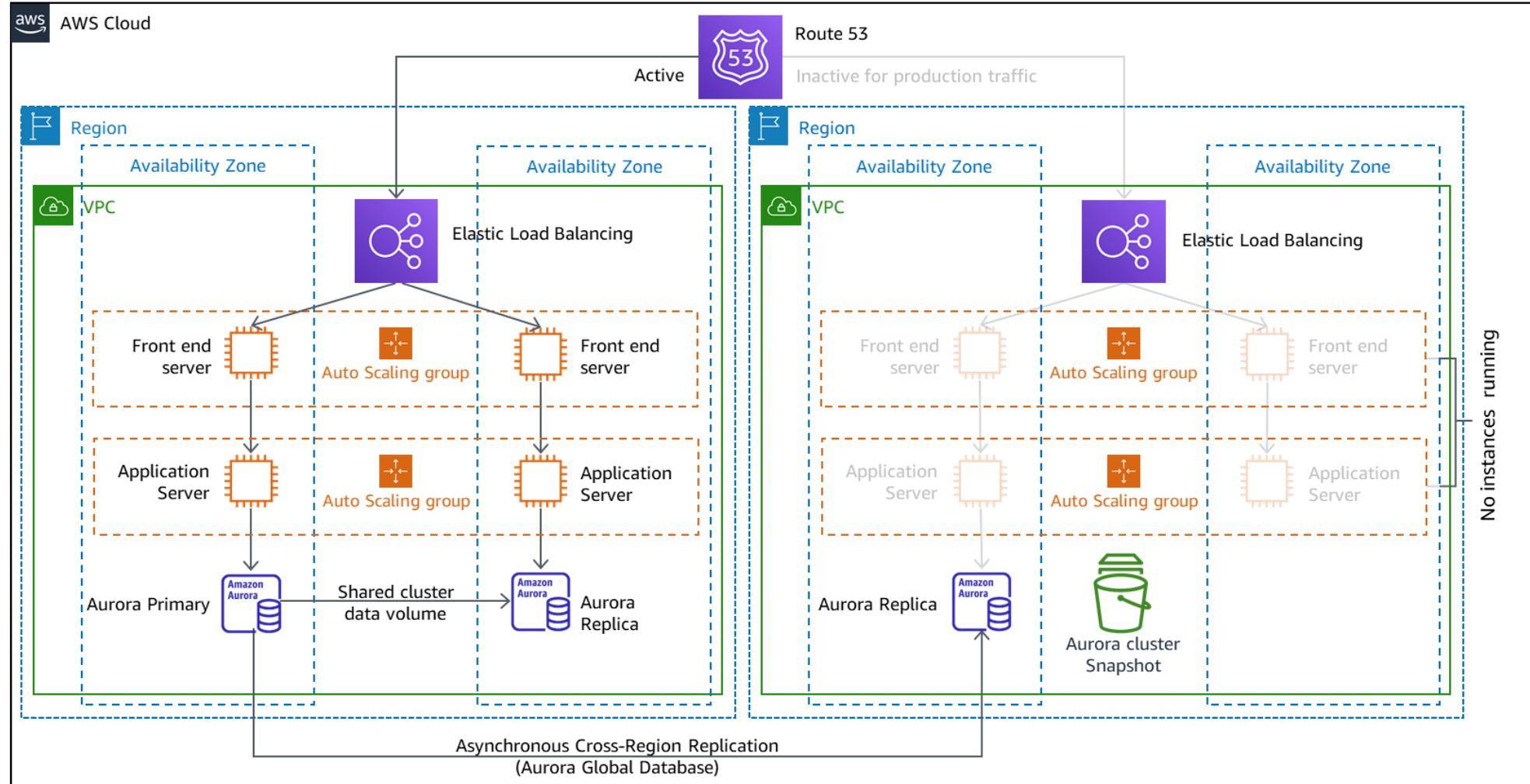
Elastic Load Balancer

Balanceador de carga





<https://aws.amazon.com/blogs/architecture/disaster-recovery-dr-architecture-on-aws-part-iii-pilot-light-and-warm-standby/>





ALB

Tipo de ELB	Atributos	Limitações	Conceitos Principais
Application Load Balancer (ALB)	<ul style="list-style-type: none">- Opera na Camada 7 (Aplicação) do modelo OSI.- Suporte a roteamento baseado em conteúdo (URL, cabeçalhos, campos de consulta).- Suporte a WebSockets e HTTP/2.- Suporte a autenticação integrada com IAM, Cognito ou OpenID Connect.- Integração com serviços como ECS e Lambda.- Regras de roteamento avançadas (URL path, host-based, etc.).	<ul style="list-style-type: none">- Máximo de 100 listeners por ALB.- Máximo de 25 regras por listener.- Limite de 5 grupos de destino por regra.- Tamanho máximo do corpo da solicitação é de 16 KB.	<p>Regras: Define o comportamento de roteamento com base em condições como URL, cabeçalhos, e métodos HTTP.</p> <p>Targets: Pode ser instâncias EC2, IPs, funções Lambda, ou containers (ECS).</p> <p>Listeners: Porta e protocolo que o ALB usa para escutar o tráfego.</p> <p>Sticky Sessions: Implementado com cookies gerados pelo ALB para garantir que as requisições subsequentes do cliente sejam enviadas para a mesma instância.</p> <p>Health Checks: Monitora a saúde das instâncias de destino através de verificações HTTP/HTTPS.</p>



NLB

Network Load Balancer (NLB)	<ul style="list-style-type: none">- Opera na Camada 4 (Transporte) do modelo OSI.- Suporte a TCP, UDP, TLS.- Capacidade de lidar com milhões de requisições por segundo com baixa latência.- Suporte para manutenção do IP do cliente original.- Integração com VPC PrivateLink.- Suporte a endereços IP estáticos.	<ul style="list-style-type: none">- Máximo de 100 listeners por NLB.- Limite de 3000 grupos de destino por NLB.- Não suporta Sticky Sessions via cookies.- Regras de roteamento mais simples em comparação com ALB.	<p>Regras: Muito básicas, principalmente baseadas em portas e protocolos.</p> <p>Targets: Pode ser instâncias EC2, IPs, containers.</p> <p>Listeners: Define porta e protocolo (TCP/UDP/TLS) que o NLB escuta.</p> <p>Sticky Sessions: Não suportado nativamente via cookies, mas pode ser implementado via outras técnicas de rede.</p> <p>Health Checks: Monitora a saúde dos destinos com verificações TCP ou HTTP simples.</p>
------------------------------------	--	--	---



Gateway Load Balancer

Gateway Load Balancer (GWLB)	<ul style="list-style-type: none">- Opera na Camada 3 (Rede) do modelo OSI.- Usado principalmente para integrar appliances de rede, como firewalls e sistemas de detecção de intrusão (IDS/IPS).- Combina roteamento e balanceamento de carga com NAT.- Gerencia tráfego entre VPCs e entre diferentes regiões AWS.	<ul style="list-style-type: none">- Máximo de 100 listeners por GWLB.- Limite de 10,000 endpoints por GWLB.- Não suporta regras de roteamento avançadas.- Não suporta Sticky Sessions.- Regras de roteamento simples, baseadas em portas e protocolos.	<p>Regras: Muito básicas, focadas em roteamento de tráfego para appliances de rede.</p> <p>Targets: Pode ser instâncias EC2, appliances de rede.</p> <p>Listeners: Porta e protocolo que o GWLB escuta.</p> <p>Sticky Sessions: Não aplicável.</p> <p>Health Checks: Suporte a verificações básicas de saúde para appliances de rede.</p>
-------------------------------------	--	--	--



Classic Load Balancer

Classic Load Balancer (CLB)	<ul style="list-style-type: none">- Opera nas Camadas 4 (Transporte) e 7 (Aplicação) do modelo OSI.- Suporte para HTTP, HTTPS, TCP, SSL.- Funcionalidade básica com suporte legado.- Integração com instâncias EC2 via Auto Scaling.- Pode平衡ear carga em várias zonas de disponibilidade.	<ul style="list-style-type: none">- Máximo de 20 listeners por CLB.- Limite de 10 grupos de destino por CLB.- Não suporta regras de roteamento avançadas como ALB.- Regras de roteamento mais limitadas.	<p>Regras: Simples, baseadas em portas e protocolos (sem suporte para condições avançadas como ALB).</p> <p>Targets: Principalmente instâncias EC2.</p> <p>Listeners: Define portas e protocolos (HTTP/HTTPS/TCP/SSL) que o CLB escuta.</p> <p>Sticky Sessions: Implementado via cookies gerados pelo load balancer (HTTP/HTTPS) ou via IP hash (TCP/SSL).</p> <p>Health Checks: Suporte para verificações de saúde simples (HTTP/HTTPS/TCP).</p>
------------------------------------	---	---	--

Semana 4

Exercícios

Exercício 1 - Revisão

Uma empresa de e-commerce está migrando sua infraestrutura para a AWS. Eles precisam configurar uma rede segura na nuvem que permita que suas instâncias EC2 em sub-redes privadas acessem a internet para baixar atualizações de software, enquanto ainda mantêm uma comunicação segura com outras instâncias EC2 em sub-redes públicas.

A arquitetura proposta inclui:

- Uma VPC com duas sub-redes: uma sub-rede pública e uma sub-rede privada.
- Instâncias EC2 na sub-rede pública que precisam ser acessíveis via SSH a partir da internet.
- Instâncias EC2 na sub-rede privada que não devem ser acessíveis diretamente da internet.
- Requisitos de conformidade exigem que todas as comunicações de entrada e saída sejam restritas ao mínimo necessário.

Pergunta:

Com base no cenário acima, qual é a melhor configuração de componentes de segurança para atender aos requisitos da empresa? Considere o uso de grupos de segurança, tabelas de rotas, listas de controle de acesso de rede (NACLs) e gateways NAT.

Exercício 1 - Revisão

- A)** Configurar um gateway de internet para a VPC e associar uma tabela de rotas que direcione todo o tráfego de saída das instâncias da sub-rede privada para o gateway de internet. Configurar um grupo de segurança que permita SSH (porta 22) para as instâncias na sub-rede pública e permitir todo o tráfego de entrada e saída na sub-rede privada.
- B)** Configurar um gateway de internet para a VPC e um gateway NAT na sub-rede pública. Associar uma tabela de rotas para a sub-rede privada que direcione o tráfego de saída para o gateway NAT. Configurar grupos de segurança para permitir SSH (porta 22) para as instâncias na sub-rede pública e permitir apenas tráfego necessário de saída para a internet nas instâncias na sub-rede privada. Usar NACLs para permitir o tráfego de retorno na sub-rede privada.
- C)** Configurar um gateway NAT na sub-rede privada e associar uma tabela de rotas que direcione o tráfego de entrada e saída para a sub-rede pública. Configurar um grupo de segurança que permita todo o tráfego de entrada e saída para ambas as sub-redes e usar NACLs para bloquear todo o tráfego de entrada na sub-rede privada.
- D)** Configurar um gateway de internet para a VPC e associar uma tabela de rotas para direcionar o tráfego de saída da sub-rede privada diretamente para o gateway de internet. Configurar grupos de segurança que permitam todo o tráfego de entrada e saída, e usar NACLs para bloquear todo o tráfego de entrada na sub-rede pública.

Exercício 1 - Revisão

Resposta Correta: B

Explicação:

- **Gateway NAT:** É necessário para que as instâncias na sub-rede privada possam acessar a internet sem serem diretamente acessíveis da internet.
- **Tabela de Rotas:** A tabela de rotas da sub-rede privada deve direcionar o tráfego de saída para o gateway NAT, enquanto a sub-rede pública usa o gateway de internet.
- **Grupos de Segurança:** Devem ser configurados para permitir apenas o tráfego necessário, como SSH para as instâncias na sub-rede pública, e comunicação seletiva para as instâncias na sub-rede privada.
- **NACLs:** Podem ser usados para adicionar uma camada adicional de segurança, controlando o tráfego de entrada e saída nas sub-redes.

Exercício 2 - Revisão

Cenário:

Uma grande organização possui várias equipes de desenvolvimento, cada uma com seu próprio ambiente na AWS. A organização deseja adotar uma estratégia de segurança para gerenciar centralmente suas contas AWS, garantindo que cada equipe tenha autonomia sobre suas respectivas contas, mas dentro de políticas de segurança rigorosas definidas pela organização.

A organização também quer garantir que todas as contas AWS sigam práticas de segurança recomendadas, como a ativação de logs do CloudTrail, proteção contra exclusão acidental de recursos críticos e gerenciamento centralizado de permissões.

Pergunta:

Qual é a melhor abordagem para implementar essa estratégia de segurança para várias contas AWS, considerando o uso do AWS Control Tower e políticas de controle de serviço (SCPs)?

Exercício 2 - Revisão

- A)** Criar uma conta principal para a organização e usar o AWS Control Tower para provisionar e gerenciar todas as contas filhas. Configurar políticas de controle de serviço (SCPs) no AWS Organizations para aplicar restrições a todas as contas, como a obrigatoriedade de usar criptografia em S3 e proibir a exclusão de logs do CloudTrail. Conceder permissões amplas para que cada equipe gerencie seus recursos sem interferir nas configurações de segurança centralizadas.
- B)** Criar uma conta principal e usar o AWS Control Tower para provisionar todas as contas filhas. Permitir que cada equipe configure suas próprias políticas de segurança e logs do CloudTrail, sem impor restrições globais, para que possam adaptar as configurações às suas necessidades específicas.
- C)** Usar o AWS Organizations para centralizar a gestão de contas, mas permitir que cada equipe configure seus próprios SCPs e políticas de segurança para melhor atender às necessidades individuais de cada equipe, sem interferência da organização central.
- D)** Criar uma única conta AWS com várias VPCs para diferentes equipes e usar o AWS IAM para gerenciar permissões dentro dessa conta única, usando SCPs apenas para restringir o uso de certos serviços.

Exercício 2 - Revisão

Resposta Correta: A

Explicação:

- **AWS Control Tower:** É ideal para provisionar e gerenciar múltiplas contas AWS, garantindo conformidade com políticas de segurança centralizadas.
- **SCPs (Políticas de Controle de Serviço):** São usadas para aplicar políticas de segurança obrigatórias a todas as contas dentro da organização, garantindo que práticas recomendadas sejam seguidas em todas as equipes.
- **Permissões Centralizadas:** A organização pode garantir que práticas de segurança cruciais sejam seguidas enquanto ainda permite que as equipes individuais tenham autonomia sobre seus próprios recursos dentro dos limites estabelecidos.

Exercício 3 - Revisão

Cenário:

Uma empresa de tecnologia está expandindo rapidamente e precisa implementar um modelo de autorização que permita o gerenciamento flexível de permissões para seus funcionários. A empresa tem diferentes equipes (desenvolvimento, operações, segurança) com necessidades distintas de acesso. Além disso, certos funcionários precisam acessar múltiplas contas AWS para diferentes projetos.

A empresa quer garantir que as permissões sejam gerenciadas de forma centralizada, mas que sejam flexíveis o suficiente para permitir atribuições temporárias de permissões para projetos específicos. A empresa também deseja minimizar o risco de permissões excessivas e garantir que todas as atividades críticas sejam monitoradas.

Pergunta:

Qual é a melhor estratégia para implementar esse modelo de autorização flexível usando usuários, grupos, funções e políticas do IAM?

Exercício 3 - Revisão

Cenário:

Uma empresa de tecnologia está expandindo rapidamente e precisa implementar um modelo de autorização que permita o gerenciamento flexível de permissões para seus funcionários. A empresa tem diferentes equipes (desenvolvimento, operações, segurança) com necessidades distintas de acesso. Além disso, certos funcionários precisam acessar múltiplas contas AWS para diferentes projetos.

A empresa quer garantir que as permissões sejam gerenciadas de forma centralizada, mas que sejam flexíveis o suficiente para permitir atribuições temporárias de permissões para projetos específicos. A empresa também deseja minimizar o risco de permissões excessivas e garantir que todas as atividades críticas sejam monitoradas.

Pergunta:

Qual é a melhor estratégia para implementar esse modelo de autorização flexível usando usuários, grupos, funções e políticas do IAM?

Exercício 3 - Revisão

- A)** Criar usuários do IAM para cada funcionário com permissões específicas atribuídas diretamente a eles. Usar grupos do IAM para conceder permissões adicionais conforme necessário para diferentes projetos. Conceder acesso a múltiplas contas usando funções do IAM que possam ser assumidas conforme necessário.
- B)** Criar funções do IAM para cada função de trabalho específica (por exemplo, desenvolvedor, administrador) e permitir que os usuários do IAM assumam essas funções conforme necessário para diferentes projetos. Usar grupos do IAM para gerenciar o acesso a recursos globais da empresa.
- C)** Criar um usuário do IAM para cada funcionário e atribuir permissões diretamente a esses usuários para cada projeto em que estejam envolvidos. Usar políticas gerenciadas pelo cliente para personalizar permissões individuais conforme necessário.
- D)** Criar grupos do IAM para cada equipe (desenvolvimento, operações, segurança) e atribuir permissões diretamente a esses grupos. Usar funções do IAM para conceder acesso temporário a recursos em diferentes contas AWS e usar políticas de sessão para restringir o escopo dessas funções.

Exercício 3 - Revisão

Resposta Correta: D

Explicação:

- **Grupos do IAM:** São ideais para conceder permissões baseadas em equipes, simplificando o gerenciamento de permissões comuns para membros da equipe.
- **Funções do IAM:** Permitem um modelo de autorização flexível, permitindo que os usuários assumam funções específicas com permissões definidas, especialmente útil para acessar recursos em diferentes contas ou para atribuições temporárias.
- **Políticas de Sessão:** Fornecem um mecanismo adicional para restringir o escopo das permissões concedidas ao assumir uma função, aumentando a segurança ao limitar o que pode ser feito durante a sessão da função.

Exercício 4

Uma empresa de serviços financeiros está implementando um sistema de processamento de transações em tempo real. Para garantir que as transações sejam processadas de forma confiável e escalável, eles decidiram usar o Amazon SQS para enfileirar as transações e o AWS Lambda para processá-las.

O volume de transações pode variar significativamente ao longo do dia, com picos que podem atingir milhares de transações por segundo. A empresa precisa garantir que o sistema possa lidar com esses picos sem perder transações e manter a latência baixa.

Pergunta:

Com base nos limites do SQS e do Lambda, qual das opções a seguir é a melhor estratégia para configurar este sistema para processar eficientemente as transações e manter o desacoplamento dos componentes?

Exercício 4

- A)** Configurar uma única fila SQS e um único Lambda para processar as mensagens. Aumentar a memória alocada para a função Lambda para maximizar a CPU e permitir o processamento de mais transações simultaneamente. Usar o mecanismo de redrive para mover as mensagens não processadas para uma fila DLQ (Dead Letter Queue).
- B)** Configurar uma fila SQS com mensagens configuradas para serem processadas em lotes (batch). Configurar a função Lambda para consumir esses lotes e definir o número máximo de invocações simultâneas para a função Lambda para evitar sobrecarga. Monitorar a fila SQS e ajustar a concorrência provisionada do Lambda para lidar com picos de carga.
- C)** Configurar várias filas SQS, uma para cada tipo de transação, e criar funções Lambda separadas para cada fila. Definir a função Lambda para consumir as mensagens imediatamente após serem adicionadas à fila. Configurar a concorrência provisionada para garantir que a função Lambda esteja sempre pronta para processar novas mensagens.
- D)** Configurar uma única fila SQS com um tempo de visibilidade curto para garantir que as mensagens sejam reprocessadas rapidamente em caso de falha. Configurar a função Lambda para processar as mensagens individualmente com a menor quantidade de memória possível para economizar custos.

Exercício 4

Resposta Correta: B

Explicação:

- **Fila SQS com Lotes (Batch Processing):** O processamento em lotes permite que o Lambda consuma várias mensagens ao mesmo tempo, o que é mais eficiente em termos de custo e pode reduzir a latência total, especialmente em volumes altos.
- **Limites do Lambda:** Lambda tem limites de concorrência e tempo de execução. Configurar o número máximo de invocações simultâneas ajuda a gerenciar picos de carga e evita que o sistema fique sobrecarregado.
- **Concorrência Provisionada:** Monitorar e ajustar a concorrência provisionada do Lambda permite que a função seja invocada rapidamente durante picos de demanda, garantindo que as mensagens sejam processadas de forma oportuna sem exceder os limites.
- **Limites do SQS:** SQS pode manipular um volume muito grande de mensagens, mas o tempo de visibilidade e o tamanho dos lotes devem ser gerenciados para otimizar o processamento e evitar que mensagens sejam processadas múltiplas vezes ou fiquem presas na fila.
Essa abordagem garante que o sistema possa lidar com grandes volumes de transações de maneira eficiente e confiável, respeitando os limites do SQS e do Lambda, e mantendo o desacoplamento necessário para escalabilidade.

Exercício 5

Cenário:

Uma empresa de mídia digital está desenvolvendo uma aplicação web que exibe conteúdo personalizado para os usuários. Para melhorar o desempenho e reduzir a carga no banco de dados, eles decidiram implementar uma camada de cache usando AWS ElastiCache. Eles precisam decidir entre usar Redis ou Memcached, e também precisam escolher a estratégia de caching mais adequada entre Lazy Loading, TTL (Time-to-Live) e Write-Through.

- **Lazy Loading:** Carrega os dados no cache apenas quando são solicitados, deixando os dados não requisitados fora do cache até que sejam necessários.
- **TTL (Time-to-Live):** Define um tempo limite para que os dados permaneçam no cache antes de serem automaticamente removidos.
- **Write-Through:** Atualiza o cache e o banco de dados ao mesmo tempo, garantindo que o cache sempre esteja sincronizado com o banco de dados.
A empresa está preocupada com a consistência dos dados, especialmente em cenários onde os dados mudam frequentemente. Eles também precisam garantir que o cache possa lidar com cargas de trabalho pesadas e oferecer baixa latência.

Exercício 5

Pergunta:

Qual das opções a seguir representa a melhor combinação de estratégia de caching e escolha de ElastiCache (Redis ou Memcached) para atender às necessidades da empresa?

- A)** Usar Memcached com Lazy Loading e TTL para minimizar a carga no banco de dados e permitir que os dados sejam automaticamente removidos do cache após um determinado período.
- B)** Usar Redis com Write-Through e TTL para garantir que o cache esteja sempre atualizado e que os dados sejam removidos do cache após um tempo definido, enquanto oferece suporte para persistência e estruturas de dados complexas.
- C)** Usar Redis com Lazy Loading e sem TTL para garantir baixa latência, sem necessidade de remover automaticamente os dados do cache, aproveitando os recursos de persistência do Redis.
- D)** Usar Memcached com Write-Through e sem TTL para garantir que os dados no cache sejam consistentes com o banco de dados, sem a necessidade de persistência ou remoção automática de dados.

Exercício 5

Resposta Correta: B

Explicação:

- **Redis:** Suporta estruturas de dados complexas e persistência, tornando-o mais adequado para casos onde a consistência e a durabilidade dos dados são importantes.
- **Write-Through com TTL:** Garante que os dados no cache estejam sempre atualizados e que dados obsoletos sejam removidos após um tempo definido, reduzindo a possibilidade de servir dados desatualizados.
- **Memcached:** É uma opção mais simples e rápida, mas não oferece persistência ou suporte para estruturas de dados complexas, tornando-o menos adequado para cenários que exigem alta consistência dos dados.

Exercício 6

Uma empresa de serviços financeiros está projetando uma aplicação que deve processar um grande volume de transações financeiras em tempo real. A aplicação é dividida em dois componentes principais: um frontend web que serve uma interface de usuário e uma API backend que processa as transações. A empresa precisa garantir alta disponibilidade e baixa latência para ambos os componentes, mas cada um tem requisitos diferentes:

- **Frontend Web:** Precisa lidar com HTTP/HTTPS, permitir roteamento baseado em URL e fornecer autenticação de usuário.
- **API Backend:** Precisa lidar com conexões TCP de baixa latência e alto volume de transações em tempo real, sem a necessidade de processamento de camada de aplicação (como roteamento HTTP).

Pergunta:

Com base nos requisitos acima, qual das opções a seguir representa a melhor configuração de平衡amento de carga para essa aplicação, utilizando Application Load Balancer (ALB) e Network Load Balancer (NLB)?

Exercício 6

- A)** Usar um ALB para ambos o frontend web e a API backend, aproveitando as funcionalidades avançadas de roteamento e autenticação do ALB para todas as requisições.
- B)** Usar um NLB para o frontend web devido à sua baixa latência, e um ALB para a API backend para aproveitar o roteamento baseado em URL.
- C)** Usar um ALB para o frontend web, aproveitando o suporte para HTTP/HTTPS e autenticação de usuário, e um NLB para a API backend, garantindo baixa latência e alta taxa de transferência para as transações em tempo real.
- D)** Usar dois ALBs, um para o frontend web e outro para a API backend, configurando roteamento baseado em URL para ambos, garantindo consistência na arquitetura.

Exercício 6

Resposta Correta: C

Explicação:

- **ALB:** É ideal para o frontend web porque oferece suporte para HTTP/HTTPS, roteamento baseado em URL, e autenticação, que são todos requisitos importantes para a interface de usuário.
- **NLB:** É mais adequado para a API backend devido à sua baixa latência, capacidade de lidar com altos volumes de tráfego TCP, e simplicidade, o que é crucial para transações financeiras em tempo real.
- **Combinação de ALB e NLB:** Permite otimizar cada componente da aplicação de acordo com seus requisitos específicos, garantindo desempenho e escalabilidade.

Exercício 7

Uma empresa de e-commerce está desenvolvendo um sistema automatizado para processar pedidos de clientes. Quando um pedido é recebido, ele passa por várias etapas, incluindo verificação de pagamento, confirmação de estoque, preparação para envio e notificação ao cliente. A empresa quer garantir que cada etapa seja executada de forma confiável, mesmo em caso de falhas temporárias. Para implementar este fluxo de trabalho, a empresa está considerando o uso de AWS Step Functions para orquestrar as etapas do processo. No entanto, eles também precisam notificar diferentes sistemas e partes interessadas (como o departamento de logística) sobre o status de cada pedido, o que pode ser feito usando o Amazon SNS.

Exercício 7

Pergunta:

Qual das abordagens a seguir seria a mais adequada para implementar esse sistema, considerando as limitações e capacidades dos AWS Step Functions e SNS?

- A)** Usar Step Functions para orquestrar todas as etapas do processo, incluindo a lógica de verificação de pagamento, confirmação de estoque e preparação para envio. Em cada etapa crítica, configurar Step Functions para publicar uma mensagem no SNS, que então notificará as partes interessadas relevantes.
- B)** Usar SNS para publicar uma mensagem sempre que um novo pedido for recebido, e configurar diferentes assinaturas SNS para iniciar as verificações de pagamento, confirmação de estoque e outras etapas. Cada assinatura SNS será configurada para chamar uma função Lambda que executa a etapa correspondente.
- C)** Usar Step Functions para orquestrar as etapas principais do processo (verificação de pagamento, confirmação de estoque, etc.), e no final de cada fluxo, configurar SNS para enviar notificações sobre o status final do pedido.
- D)** Usar Step Functions para orquestrar as etapas principais e SNS para processar etapas paralelas, como notificações de status, de forma independente das funções principais. SNS será responsável por iniciar as etapas de processamento, enquanto Step Functions coordena as atividades principais.

Exercício 7

Resposta Correta: A

Explicação:

- **Step Functions:** É ideal para orquestrar fluxos de trabalho complexos com múltiplas etapas e garante a resiliência ao lidar com falhas, oferecendo monitoramento e retries automáticos.
- **SNS:** Integrado com Step Functions, permite notificar diferentes sistemas ou partes interessadas de forma assíncrona durante o processo, garantindo que os sistemas externos sejam informados sem interromper o fluxo principal.
- **Abordagem A:** Garante que o processo principal seja gerenciado de forma confiável com Step Functions, enquanto o SNS lida com notificações paralelas e assíncronas para os sistemas envolvidos.

Exercício 8

Uma empresa de marketing digital precisa processar grandes volumes de dados de eventos gerados a partir de diversas campanhas publicitárias em diferentes plataformas. Os eventos incluem cliques em anúncios, visualizações de páginas e conversões. Esses dados são usados para análises em tempo real e para acionar notificações para diferentes sistemas internos e parceiros externos.

A arquitetura proposta deve garantir que os eventos sejam entregues de forma confiável e que o processamento possa escalar conforme a demanda aumenta. A empresa está considerando usar SNS para enviar os eventos para diferentes sistemas e SQS para garantir que todos os eventos sejam processados, mesmo que haja picos repentinos no volume de dados.

Pergunta:

Qual das opções a seguir representa a melhor configuração para desacoplar e processar os eventos gerados pelas campanhas, considerando as características e limitações do SNS e do SQS?

Exercício 8

Uma empresa de marketing digital precisa processar grandes volumes de dados de eventos gerados a partir de diversas campanhas publicitárias em diferentes plataformas. Os eventos incluem cliques em anúncios, visualizações de páginas e conversões. Esses dados são usados para análises em tempo real e para acionar notificações para diferentes sistemas internos e parceiros externos.

A arquitetura proposta deve garantir que os eventos sejam entregues de forma confiável e que o processamento possa escalar conforme a demanda aumenta. A empresa está considerando usar SNS para enviar os eventos para diferentes sistemas e SQS para garantir que todos os eventos sejam processados, mesmo que haja picos repentinos no volume de dados.

Pergunta:

Qual das opções a seguir representa a melhor configuração para desacoplar e processar os eventos gerados pelas campanhas, considerando as características e limitações do SNS e do SQS?

Exercício 8

- A)** Usar SNS para publicar os eventos gerados pelas campanhas e configurar múltiplas filas SQS como assinaturas. Cada fila SQS será consumida por diferentes sistemas internos ou parceiros, garantindo que os eventos sejam processados independentemente do volume de dados.
- B)** Usar SQS como o principal mecanismo de publicação para eventos, com múltiplas filas para diferentes tipos de eventos. Cada fila será consumida por diferentes sistemas, e SNS será usado apenas para notificar sistemas externos sobre eventos críticos.
- C)** Usar SNS para publicar os eventos e notificar diretamente os sistemas internos e parceiros, sem usar SQS. Cada sistema será responsável por processar os eventos imediatamente após recebê-los.
- D)** Usar SQS para enfileirar os eventos e configurar uma única assinatura SNS que publique um resumo dos eventos processados para sistemas externos no final de cada dia.

Exercício 8

Resposta Correta: A

Explicação:

- **SNS com SQS:** Usar SNS para publicar eventos e SQS como assinaturas garante que os eventos sejam distribuídos de forma confiável para múltiplos sistemas. Cada fila SQS pode lidar com picos de carga de forma independente, desacoplando o processamento e garantindo que nenhum evento seja perdido.
- **SNS:** Ideal para fan-out de mensagens, onde uma única publicação precisa ser distribuída para múltiplos destinos (neste caso, múltiplas filas SQS).
- **SQS:** Garante que as mensagens sejam processadas de maneira confiável e escalável, suportando filas que podem crescer conforme a demanda, sem perda de mensagens.

Exercício 9

Uma empresa de tecnologia está desenvolvendo uma aplicação crítica que depende de um banco de dados relacional hospedado no Amazon RDS. A aplicação deve ser altamente disponível e resiliente a falhas, com a capacidade de se recuperar rapidamente em caso de problemas com o banco de dados principal. A empresa também está considerando como melhorar o desempenho de leitura em escala global, pois espera um grande volume de tráfego de leitura de diferentes regiões.

A empresa está avaliando as opções de Multi-AZ (Alta Disponibilidade) e Read Replicas (Desempenho de Leitura), mas está confusa sobre as diferenças entre a replicação síncrona e assíncrona, e como cada uma impacta a disponibilidade, latência e consistência dos dados.

Pergunta:

Qual das seguintes opções representa a melhor configuração para a empresa, considerando a necessidade de alta disponibilidade e desempenho de leitura, e quais são as diferenças principais entre a replicação síncrona e assíncrona no contexto do Amazon RDS?

Exercício 9

- A)** Usar Multi-AZ para garantir alta disponibilidade e failover automático, pois a replicação síncrona garante que os dados estejam completamente atualizados no banco de dados de réplica antes de confirmar uma transação. Além disso, configurar Read Replicas para escalar a leitura globalmente, aceitando o risco de eventual inconsistência de dados devido à replicação assíncrona.
- B)** Usar Read Replicas em múltiplas regiões para alta disponibilidade e baixa latência de leitura, pois a replicação assíncrona garante que as réplicas recebam atualizações com a menor latência possível. Configurar Multi-AZ para melhorar a escalabilidade de leitura em uma única região, com a garantia de consistência dos dados.
- C)** Usar apenas Read Replicas para alcançar alta disponibilidade, pois a replicação assíncrona permite que a aplicação continue funcionando, mesmo se a réplica estiver ligeiramente atrasada em relação ao banco de dados principal. Não configurar Multi-AZ, já que a alta disponibilidade pode ser gerenciada manualmente por failover para uma réplica de leitura.
- D)** Usar Multi-AZ para todas as regiões onde a aplicação opera, pois a replicação síncrona é necessária para garantir consistência dos dados em escala global. Evitar Read Replicas, pois a latência de leitura é menos importante que a consistência dos dados.

Exercício 9

Resposta Correta: A

Explicação:

- **Multi-AZ (Replicação Síncrona):** Garante alta disponibilidade e consistência dos dados ao replicar de forma síncrona as transações para uma instância de standby em outra zona de disponibilidade. Em caso de falha da instância principal, o failover automático garante continuidade sem perda de dados.
- **Read Replicas (Replicação Assíncrona):** São usadas para escalar leituras, especialmente em diferentes regiões. No entanto, como a replicação é assíncrona, pode haver um pequeno atraso na propagação dos dados, o que significa que as leituras podem não refletir imediatamente as últimas atualizações.
- **Diferenças Principais:** A replicação síncrona (Multi-AZ) prioriza a consistência e disponibilidade dos dados, enquanto a replicação assíncrona (Read Replicas) prioriza o desempenho de leitura, permitindo que os dados sejam lidos em múltiplas regiões, com potencial risco de latência e inconsistência momentânea. Essa configuração oferece uma combinação robusta de alta disponibilidade com Multi-AZ e escalabilidade de leitura com Read Replicas, atendendo às necessidades de disponibilidade e desempenho global da empresa.



Bônus: Atividade Desafio (extra, demanda bastante tempo)

- 1) Escolher alguma plataforma com simulados e resolver um simulado do início ao fim.
- 2) Ao final, anotar maiores pontos de dúvidas.
- 3) Separar em tópicos todas as dúvidas e fazer documento breve explicando cada um dos pontos.
- 4) Se possível e quiser, compartilhar aqui com a turma o documento de anotações e resolução das dúvidas.
- 5) Fazer outro simulado e comparar o desempenho.
- 6) Se quiser, trazer relato no próximo encontro! (Dia 20/08/2024)