Heloa Vitoria Silva - RM 343686

Solução de Arquitetura do Instagram

Funcional Requerimentos

1. Usuários podem carregar fotos e vídeos
2. Usuários podem visualizar fotos e vídeos
3. Usuários podem seguir e deixar de seguir outros usuários
4. Usuários podem procurar outros usuários na barra de procura
5. Criar um *feed* para cada usuário com as postagens dos usuários que ele segue

Requerimentos não funcionais

1. Escalável
2. Consistente
3. Alta disponibilidade
4. Dados do usuário devem ser persistidos
5. Latência máxima do *feed* deve ser de 120s

Quantidade de dados

Usuários

Vamos assumir que teremos 500 milhões de usuários registrados, em média 25% são usuários ativos por hora sendo 125 milhões, multiplicando por 100, para obter número de requisições de leitura e uma média de trafico tendo então 12.5B de requests, multiplicando por 4 para ter o trafego durante o pico 50B de requests.

Posts

Tendo a quantidade media de usuários ativos em 125 milhões conforme mencionado anteriormente, considerando que um usuário padrão realize 3 publicações por semana, cada uma com 1 MB (imagem + texto), uma media de 100 seguidores visualizam, com pelo menos 10 likes e 2-3 comentários e atualize o *feed* 2 vezes ao dia.

Estimativa de Capacidade

1. Carregamento de fotos diaria = 5 milhões

2. Armazenamento diario = 5 milhões * 150KB = 716GB
3. Manter os dados por 10 anos, 716 GB*365*10=2553TB ≈ 2.6 PB
4. Requisições do Feed = 10 milhões, sendo 2800 RPS*
5. Considerando uma media de 1 busca por dia seria 10 milhões totalizando 115 RPS*.

* (requests per second)

Solução

Armazenamento fotos

A ideia é ter 2 servidores para escrita e leitura respectivamente. Além disso, separar as solicitações de leitura e gravação das fotos nos permitirá dimensionar e otimizar cada processo de forma independente.

Para essa parte da solução, a sugestão seria utilizar o **S3** combinado com o **Athena** para armazenar e recuperar as informações rapidamente.

Feed

O feed de notícias pré-gerado é adotado. Criamos um servidor dedicado a gerar o feed de notícias exclusivo para cada usuário e armazená-lo em uma tabela de feed de notícias separada. Com essa abordagem, quando o usuário clicar na atualização, o feed de notícias do banco de dados será exibido ao usuário.

O feed então será um serviço executado sob demanda com o **Lambda** para processar o algoritmo e retornar para um banco de metadados em **Redshift**.

Todos os eventos que o usuário disparar serão executados de forma assíncrona através do **Lambda**.

Tráfego

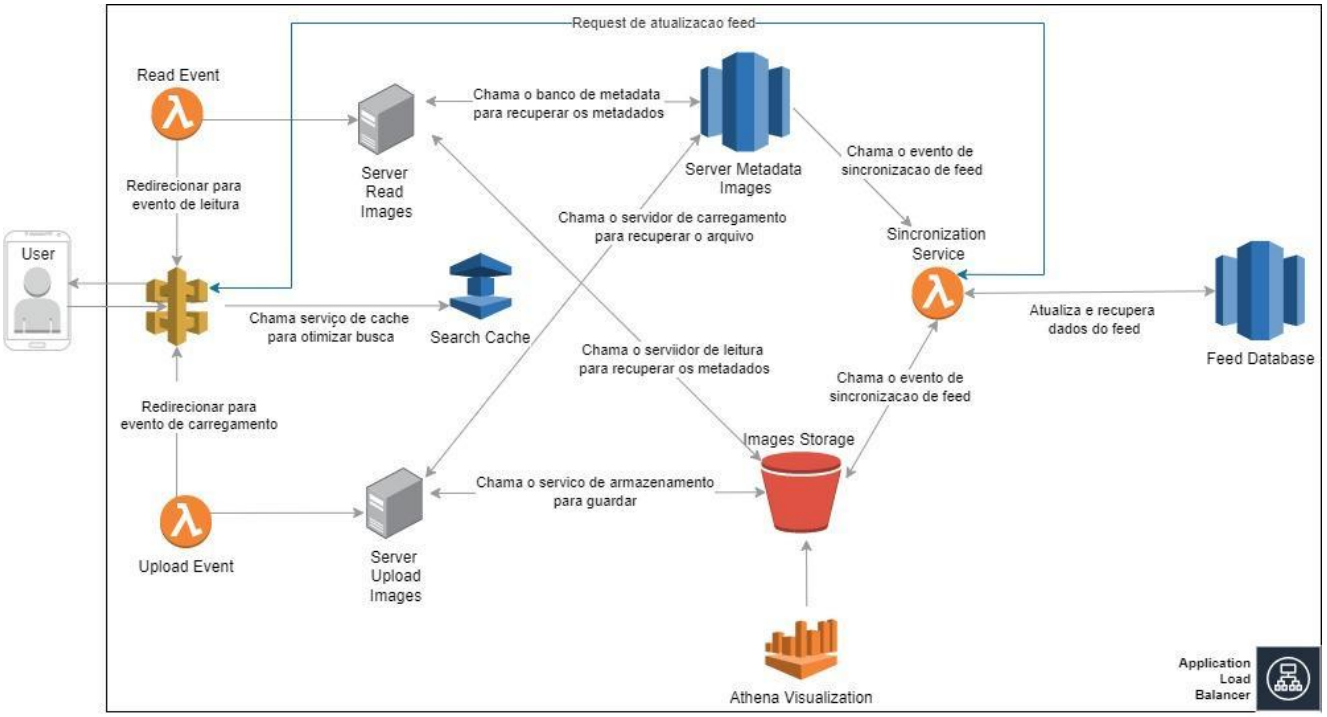
Os usuários irão gerar feed de notícias, alguns vão carregar fotos, alguns visualizarão as fotos, etc. Precisamos ter um único ponto de entrada para todos os clientes. Esse ponto de entrada é o **API Gateway**.

Ele lidará com todas as solicitações enviando-as para vários serviços. E para algumas solicitações, ele apenas roteará para o servidor específico. O gateway de API também pode implementar segurança, como verificar a permissão do cliente para realizar a solicitação.

Escalabilidade

Para garantir que a solução vai responder em tempo hábil as requisições do usuário e para distribuir o tráfego de usuários para os servidores de forma eficiente, utilizaremos **Load Balancer**.

Diagrama


Request de atualizacao feed
Read Event
Chama o banco de metadata para recuperar os metadados
Server Metadata Images
Chama o evento de sincronizacao de feed
Server Read Images
Redirecionar para evento de leitura
Chama o servidor de carregamento para recuperar o arquivo
Sincronization Service
User
Chama serviço de cache para otimizar busca
Search Cache
Atualiza e recupera dados do feed
Feed Database
Chama o serviidor de leitura para recuperar os metadados
Chama o evento de sincronizacao de feed
Redirecionar para evento de carregamento
Images Storage
Chama o servico de armazenamento para guardar
Upload Event
Server Upload Images
Athena Visualization
Application Load Balancer