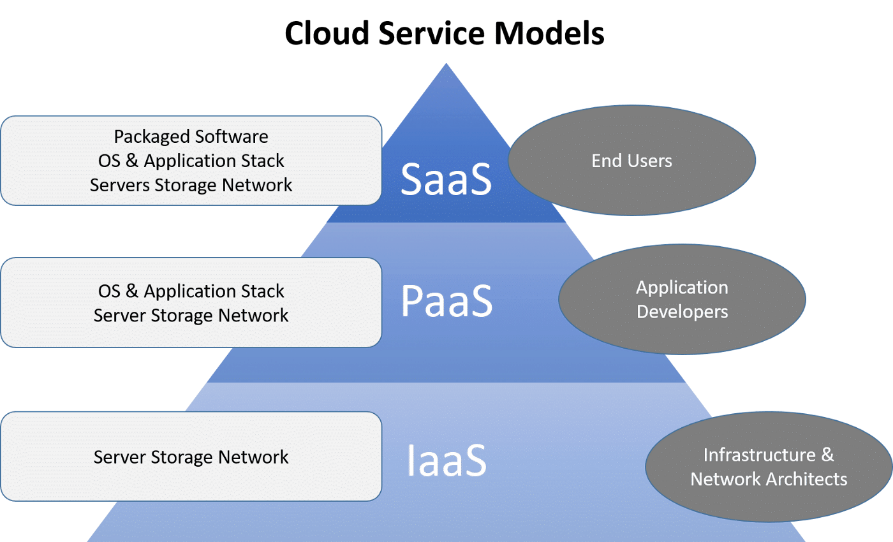
**Sistemas de Computação em Cloud**

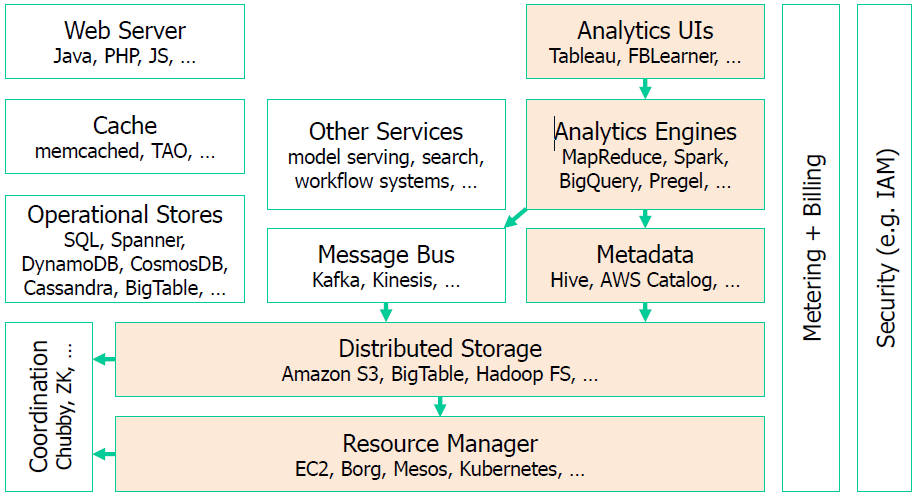
A computação em nuvem refere-se tanto aos aplicativos entregues, como serviços pela Internet e o hardware e os sistemas software nos “data centers” que fornecem esses serviços.

# Tipos de serviços

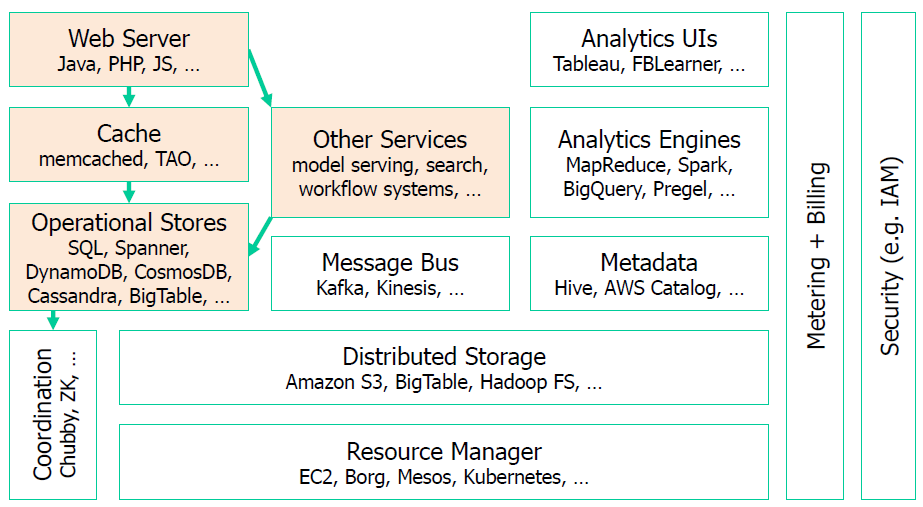


# Cloud Software Stack

## Analiytics Warehouse



## Web application



# Building blocks

## Compute

* Basicamente um computador: processadores, memória (RAM), canais para rede e nós de armazenamento.
* Em um DC, um nó de computação é um servidor, geralmente montado em um rack.

## Networking

Permite que as comunicações fluam entre as partes

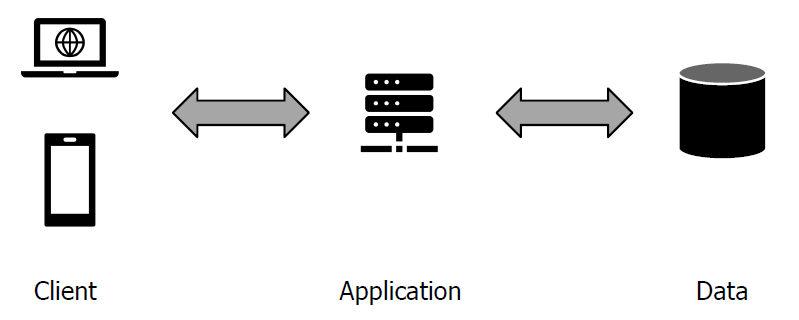
* Os nós de rede geralmente são switches e roteadores.
* SDN, NICs inteligentes, acesso remoto direto à memória.

## Storage

Armazena “dados” persistentemente em dispositivos de disco.

* SSD, NVM Flash e memória não volátil

### Web applications



Razões para ter **multiplas instâncias** do servidor de aplicativos:

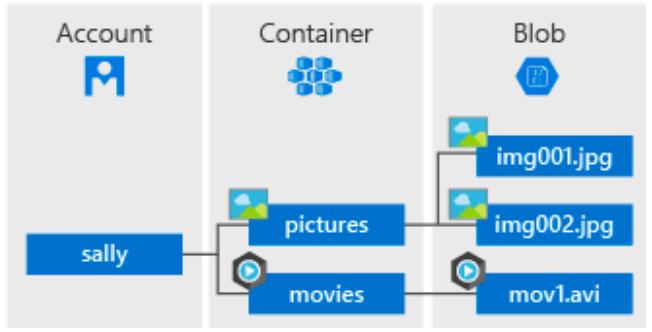
1. Um único servidor pode ser **insuficiente para lidar com todos os pedidos**;
2. Um único servidor é um **único ponto de falha** tornando o serviço indisponível;
3. Um único servidor pode estar muito longe de alguns clientes: **latência muito alta**.

### Blob storage services

Os armazenamentos de blob são projetados para armazenar grandes quantidades de **dados não estruturados**.

* Servindo imagens ou documentos diretamente para um navegador.
* Armazenamento de arquivos para acesso distribuído.
* Streaming de vídeo e áudio.
* Gravação em arquivos de log.
* Armazenamento de dados para backup e restauração, recuperação de desastres e arquivamento

## O Blob storage oferece 3 tipos de recursos



## Tipos de Blobs (Azure)

* **Block blobs** armazenam texto e dados binários, até cerca de 190 TB.
* Os **Append blobs** são compostos de blocos otimizados para operações de acréscimo (login).
* Os **Page blobs** armazenam arquivos de acesso aleatório de até 8 TB de tamanho, armazenando em arquivos de disco rígido virtual (VHD) e servem como discos para MV do Azure.

## Performance tiers (Azure)

**Premium**

* Otimizado para altas taxas de transação e latência de armazenamento consistente de um dígito (transformação de dados)

**Standard**

* Otimizado para alta capacidade e alto rendimento (processamento de dados em massa)

## Access tiers (Azure)

**Hot access tier**

* Para acesso frequente de objetos.
* Custo de armazenamento são maiores.

**Cool access tier**

* Para armazenar grandes quantidades de dados
* Acessados com pouca frequência e armazenados por pelo menos 30 dias.
* Custo mais caro do que acessar dados na Hot tier.

**Archive tier**

* Dados que podem tolerar várias horas de latência de recuperação e permanecerão na **Archive tier** por pelo menos 180 dias.

## Redundancy levels (Azure)

**Locally-redundant storage (LRS)**

* Tolerar falhas de máquina em uma região.
* Os dados são replicados de forma síncrona na região primária.

**Zone-redundant storage (ZRS)**

* Tolera falhas de data center em uma determinada região.
* Os dados são replicados de forma síncrona nas zonas de disponibilidade na região primária.

**Geo-redundant storage (GRS)**

* Tolera falhas de regiões completas
* Os dados são replicados de forma síncrona na região primária e, em seguida, replicados de forma assíncrona na região secundária.

**Geo-zone-redundant storage (GZRS)**

* Tolera falhas do data center e falhas completas da região
* Os dados são replicados de forma síncrona nas zonas de disponibilidade na região primária e, em seguida, replicados de forma assíncrona na região secundária.

****



## Concurrencia

Por padrão, o Azure usa uma política de **last-writer-wins**

**Alternativas:**

* Simultaneidade otimista com atualizações condicionais - Um cliente pode especificar alguma condição que deve ser mantida para que uma atualização seja aceita
* Simultaneidade pessimista - Um cliente pode bloquear um bloco para acesso exclusivo.

## Protection

**Soft Delete**

* A exclusão reversível de blob protege um blob individual (ou contêiner) contra exclusões ou substituições acidentais.
* Mantém os dados excluídos por um período de tempo especificado,
* durante o qual é possível “recuperar” o blob.

**Versioning**

* Permitir manter várias versões de um blob.

### Cloud databases

SQL databases fornecem uma forte consistência.

Databases replicados com consistência forte fornecem a ilusão de que existe uma única BD, isto exige coordenação entre vários servidores.

# Cloud databases – First Gen. (Amazon Dynamo)

* Replicação geográfica com alta disponibilidade e desempenho

## Data Model (Amazon Dynamo)

* (Key,Value) pairs
* Easy to scale
* Consistent hashing

## Eventual consistency model

Alta disponibilidade e baixa latência exigem a aceitação de operações sem coordenação com outras réplicas, surgindo o modelo de consistência eventual (**Eventual consistency model**):

* As réplicas podem sempre aceitar atualizações;
* As atualizações são eventualmente propagadas para todas as réplicas;
* As réplicas convergem para o mesmo estado

Quando ocorrem atualizações simultâneas, o sistema mantém as várias versões. O aplicativo deve mesclar versões aplicando uma nova atualização.

**Problemas:**

* Nem todos os aplicativos/dados funcionam corretamente sob consistência eventual

## Amazon DynamoDB

Estende o design inicial do Dynamo em várias direções.

* O modelo de dados suporta documentos (como documentos JSON).
* Geo-replicação: possível selecionar regiões para replicar dados.
* Maior consistência com operação de atualização condicional.
* Execute a atualização se alguma condição do estado se mantiver.

# Cloud databases – Current Gen. (Azure CosmosDB)

* Distribuido globalmente
* Taxa de transferência e armazenamento fáceis de dimensionar
* Acesso a dados de um dígito em milissegundos usando APIs que incluem SQL, MongoDB, Cassandra, Tables, etc.
* Fornece acordos de nível de serviço (SLAs) abrangentes para garantias de taxa de transferência, latência, disponibilidade e consistência

## Database API

1. **Relational data model**

* Banco de dados composto por tabelas e documentos.

1. **Document data model**

* Um documento pode ser qualquer objeto JSON.

1. **Graph data model**

* Representam dados que podem ser modelados como um gráfico (por exemplo, rede social relações, etc)

Mecanismo da Database é baseado em sequência de registro de átomos (ARS):

* **Atoms** consistem em um pequeno conjunto de tipos primitivos
* **Records** são estruturas de pares (key, value)
* **Sequences** são matrizes que consistem em Atoms, Records ou Sequences.

Tem suporte para distribuição global:

* Replicação **Multi-master** (várias réplicas aceitarão operações de atualização, sem coordenação com outras réplicas).
* Nível de consistência bem definido.
* Taxa de transferência escalável de leitura e gravação com SLAs.

## Consistency Model



Nota: apps podem escolher o modelo de consistência adequado

#### Strong consistency

Dá a ilusão de que existe uma única BD

Oferece:

* Uma leitura é garantida para retornar a versão mais recente de um item.
* Um cliente nunca vê uma gravação parcial ou não confirmada.

#### Bounded Staleness

Na BD todas as atualizações fora da **staleness window** são aplicadas na mesma ordem total em todas as réplicas.

Para atualizações dentro da **staleness window**:

1. Quando há uma única região de gravação:

* Para clientes dessa região, as atualizações são totalmente solicitadas imediatamente
* Para clientes em outras regiões, os clientes podem perder atualizações dentro da janela de inatividade, mas as verão em ordem

1. Quando houver várias regiões de gravação:

* Para clientes que gravam em uma única região, os clientes observarão essas atualizações de atualizações em ordem
* Quando o cliente grava em diferentes regiões, as atualizações são aplicadas localmente, mas podem ser reordenadas posteriormente (consistência eventual)

**Garante**

* Consistent-prefix - todas as atualizações são totalmente ordenadas, exceto as na **staleness window**
* Monotonic read - clientes observam uma versão posterior à lida antes
* Uma leitura pode retornar um valor antigo do item de dados

**Implementação**

* A Master region ordena e propagação de atualizações para outras regiões.
* Uma região pode receber operações que se propagam para a região mestre a ser ordenada – enquanto a ordem não for estabelecida pelo mestre, essas atualizações ficam visíveis fora de ordem na região local.
* A leitura pode ser realizada na região local, desde que as condições limitadas possam ser estabelecidas localmente – por exemplo. a partir da última mensagem recebida do mestre, uma réplica conhece a potencial obsolescência.

#### Session

O cliente tem uma session, onde vê a BD evoluindo como se fosse uma única réplica. As atualizações de outros clientes/regiões são integradas na visualização da sessão.

Nota: diferentes clintes podem ver diferentes estados da BD.

**Garante**

* Consistent-prefix - todas as atualizações são totalmente ordenadas, exceto as na **staleness window**
* Monotonic read - clientes observam uma versão posterior à lida antes
* Monotonic writes - as gravações são propagadas após as gravações que as precedem logicamente.
* Write-follows-reads - uma gravação é propagada sempre após as atualizações de leitura.
* Read your writes - uma leitura sempre reflete as gravações executadas na sessão.

Os clientes fora da session que executam gravações verão:

* na mesma região as atualizações em ordem;
* em outras regiões, com atualizações realizadas em uma única região, a atualização em ordem;
* escrevendo em outras regiões a atualização sendo reordenada em relação às gravações de outras regiões

**Implementação**

* Cliente mantém vetor de versão (token, contexto) com resumo das operações observadas;
* Leituras solicitam um estado que seja pelo menos tão recente quanto o vetor;
* As atualizações em cada região são ordenadas e essa ordem é respeitada ao aplicá-las a outras réplicas.

#### Consistent Prefix

O cliente vê um prefixo das atualizações de cada região, mas pode perder atualizações recentes de diferentes regiões.

**Garante**

* Os resultados retornados contêm algum prefixo de todas as atualizações, sem lacunas;
* As leituras nunca veem gravações fora de ordem de uma região (ou seja, observa sempre as atualizações executadas na região na mesma ordem).

**Implementação**

* A região solicita atualizações e as propaga para as réplicas;
* Lê vê as atualizações recebidas pelo mestre (em ordem) mais pedido local.

#### Eventual

O cliente pode ver um estado que reflete qualquer subconjunto das atualizações.

Sob consistência eventual, não há garantia de ordenação para leituras. Na ausência de outras gravações, as réplicas eventualmente convergem.

### Application cache at the data-center

**Dados de cache na máquina do servidor de aplicativos. Problemas?**

* O tamanho do cache é limitado à memória de cada máquina.
* Distribuir a carga significa que os mesmos dados serão armazenados em cache em várias máquinas.

## Compartilhar cache entre várias máquinas

**Vantagens:**

* Tamanho de cache muito maior do que o cache somente na memória local.
* O acesso ao cache é mais barato que ao database.

**Desvantagens:**

* Mais lento quando comparado ao acesso à memória local.

## Usos de caches

**Content Caching**

* Armazene dados que mudam com pouca frequência.
* Reduz o tempo de processamento e a carga do servidor.

**Cache-Aside**

* Cache parte do database para acesso mais rápido.
* Modifique o cache ao modificar os dados de back-end.
* Reduz a carga do servidor.

**User session caching**

* Armazene informações associadas a uma sessão de usuário no cache em vez de no banco de dados.
* Permite interação mais rápida;
* As solicitações do cliente podem ser processadas por qualquer servidor.

# Memcached

Cache distribuído básico, onde os objetos são tratados como BLOBs.

## Propriedades

* Armazenamento de valor-chave distribuído (tabela de hash)
* Tamanho limitado – pares de chave e valor são descartados quando o cache está cheio
* Política de remoção de cache: LRU (usado menos recentemente).
* Mais detalhes: cache dividido em HOT, WARM e COLD (novos itens entram no HOT; remoção de cache move o objeto para o nível inferior)
* **Desenhado para:**
  + Servidores de alto rendimento – o acesso à memória é muito mais rápido que o disco.
  + Consultas de alta latência – evite repetir consultas caras.

## Arquitetura

**One-hop DHT**

* Os clientes conhecem todos os servidores.
* Os clientes conhecem a função hash para atribuir uma chave a um servidor(pode usar hash consistente)
* Os clientes enviam operações para o servidor que armazenará uma determinada chave diretamente.
* Os servidores mantêm um armazenamento de “valor-chave”.

## Limitações do Memcached

**THUNDERING HERDS**

* Em uma falta de cache, vários clientes podem acabar tentando definir o valor de uma entrada de cache.
* Para um valor atualizado com muita frequência, as gravações podem ser reordenadas e o valor antigo é gravado no cache.

Solução [do Facebook]: Os clientes obtêm concessões para escrever o valor de uma chave. Isso garante que um único cliente grave por vez.

**SEM REPLICAÇÃO**

* Projetado para dados voláteis
  + Falha: Os clientes simplesmente vão para o disco.
  + Recuperação: O Cache é preenchido como consequência da execução normal dos clientes.
* Precisa de redundância?
  + Necessidade de criar redundância acima do Memcached.

## Recursos adicionais

* Suporte para contador

# Redis

Modelo de dados avançado, com suporte para tipos como List, Set, etc.