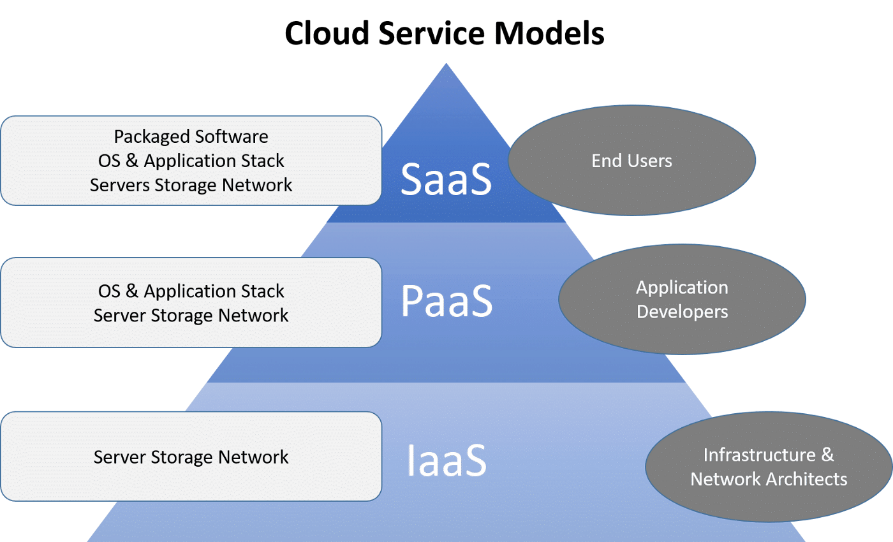
**Sistemas de Computação em Cloud**

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



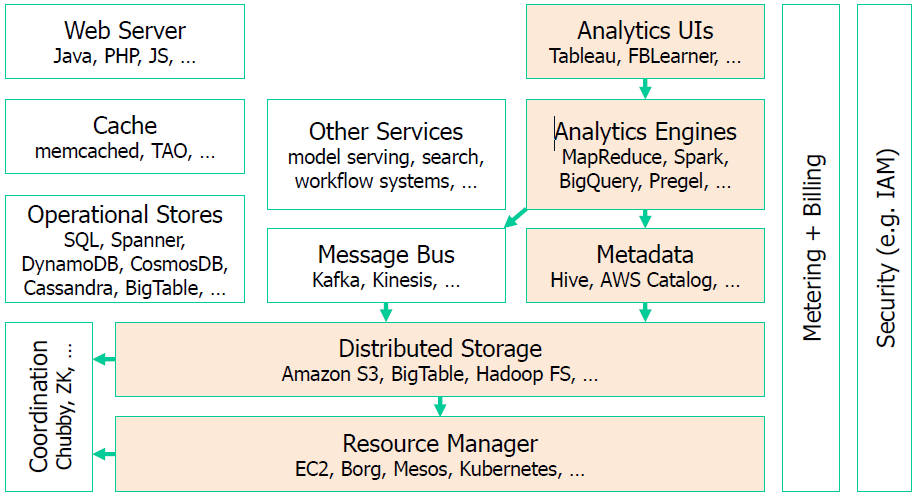
Provides processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software

The capability provided to the consumer is to deploy onto the cloud Infrastructure applications created using programming languages

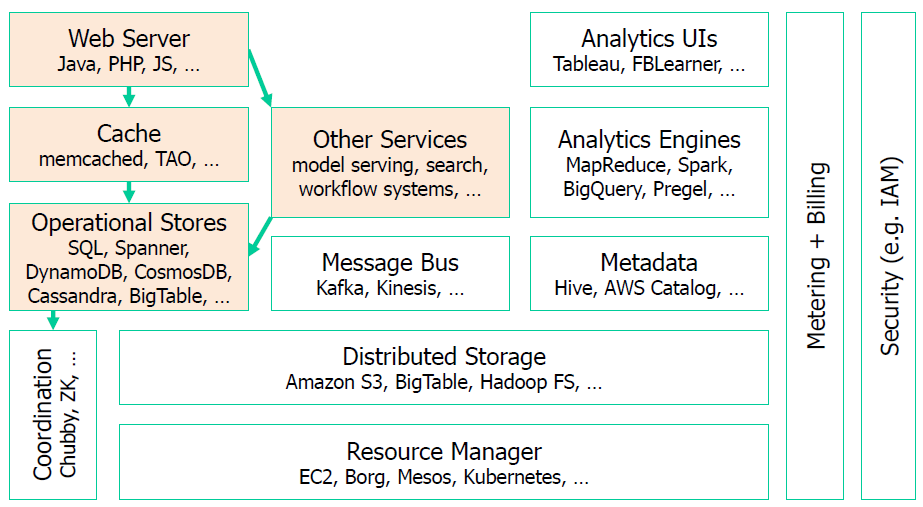
The capability provided to the consumer is to use the provider’s applications running on a cloud infrastructure

# 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

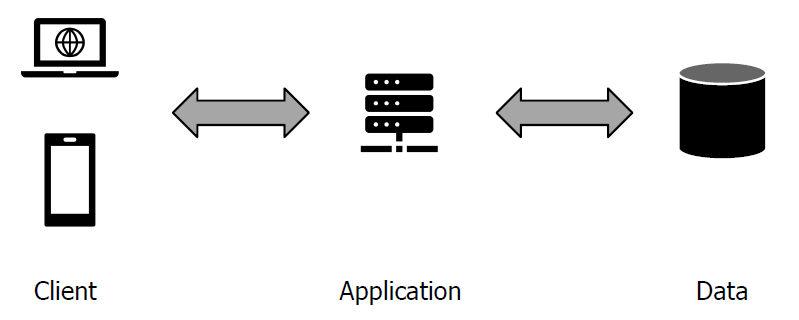
* Os nós de rede geralmente são switches e roteadores.
* SDN, NICs inteligentes, acesso remoto direto à memória.

## 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**.

## Deploying novas versões

Cria e implanta automaticamente novas versões de um repositório de origem, sem tempo de inatividade

**Staging**

1. Mantenha a versão antiga em execução;
2. Implantar a nova versão em um slot de teste (usando a terminologia do Azure) – as instâncias são criadas, mas as solicitações dos clientes continuam sendo processadas pela versão antiga;
3. (Opcional) aplicação de aquecimento;
4. Hotswap da versão antiga para a nova, iniciando o encaminhamento de solicitações do cliente para a nova versão.

**A/B deployment**

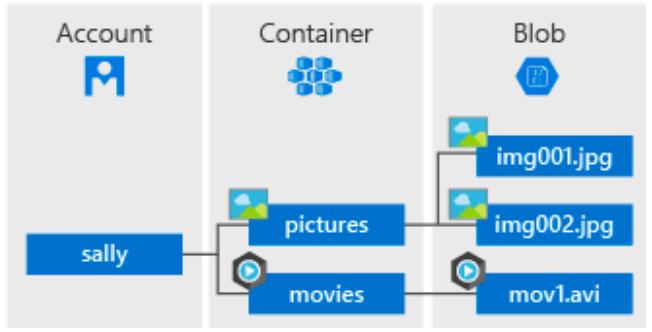
1. Ter duas versões rodando ao mesmo tempo;
2. Encaminhe uma pequena porcentagem de solicitações para a nova versão.

### 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.
* **Append blobs** são compostos de blocos otimizados para operações de acréscimo (logging scenarios).
* **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
  + latência de armazenamento consistente de um dígito (transformação de dados)

**Standard**

* Otimizado para alta:
  + capacidade
  + rendimento (processamento de dados em massa)

## Access tiers (Azure)

**Hot access tier**

* Para acesso frequente
* + custo de armazenamento

**Cool access tier**

* Para acesso pouco frequente
* Para grandes quantidades
* Armazenados por >= 30d
* + custo de acessamento

**Archive tier**

* Tolera várias horas de latência de recuperação
* Permanecerão por >= 180d

## Redundancy levels (Azure)

Redundância oferece: fault tolerance e rápido acesso à informação

**Locally-redundant storage (LRS)**

* Tolerar falhas de máquina numa região.
* Os dados são replicados 3 vezes de forma síncrona na região primária.

**Zone-redundant storage (ZRS)**

* Tolera falhas de data center numa região.
* Os dados são replicados de forma síncrona em 3 zonas na região primária.

**Geo-redundant storage (GRS)**

* Tolera falhas de regiões
* Os dados são replicados de 3 vezes 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 da região
* Os dados são replicados de forma síncrona em 3 zonas na região primária e, em seguida, replicados de forma assíncrona na região secundária.

“Under the hood”

1. Três vezes a replicação dentro de um DC, os dados são replicados em diferentes racks
2. Replicação síncrona vs. Assíncrona:

* Replicação síncrona na zona primária para estabelecer o estado “oficial” do blob.
* Replicação assíncrona para zona secundária, permitindo retornar o resultado de uma operação de gravação ao cliente sem incorrer em latência entre regiões.

## Protection

**Soft Delete**

* Delete reversivél dum blob indiv. ou container
* Os dados excluidos ficam guardados por um periodo de tempo

**Versioning**

* Mantém várias versões de um blob.

## Security

Data stored is encrypted using AES.

* Shared account keys, access key, and have active directory.

Communications always uses secure channels.

Role-based access control

## Concurrencia

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

**Alternativas:**

* Simultaneidade otimista - Um cliente especifica uma condição para que uma atualização seja aceite
* Simultaneidade pessimista - Um cliente pode bloquear um bloco para acesso exclusivo.

### Cloud databases

SQL databases fornecem uma forte consistência.

Databases replicados + consistência 🡪 ilusão de uma única BD (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

* + disponibilidade / - latência

🡪 exige que as réplicas aceitem operações sem coordenação com outras réplicas:

* + 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.

Nem todos os aplicativos/dados funcionam corretamente sob consistência eventual

## Amazon DynamoDB

Estende o design inicial do Dynamo em várias direções.

* 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**

* DB compostas 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)

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

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

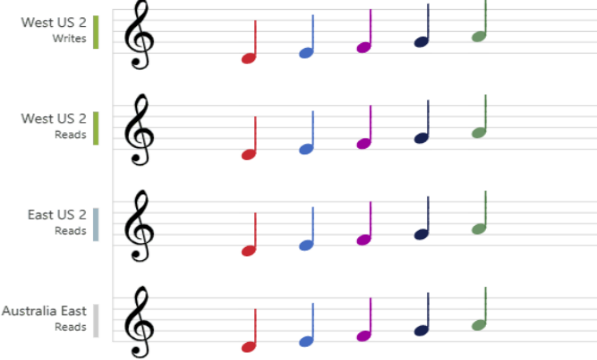
Tem suporte para distribuição global:

* Replicação **Multi-master** (réplicas aceitão op. de atualização, sem coordenar com outras)
* Taxa de transferência escalável

## Consistency Model



Nota: apps podem escolher o modelo de consistência adequado



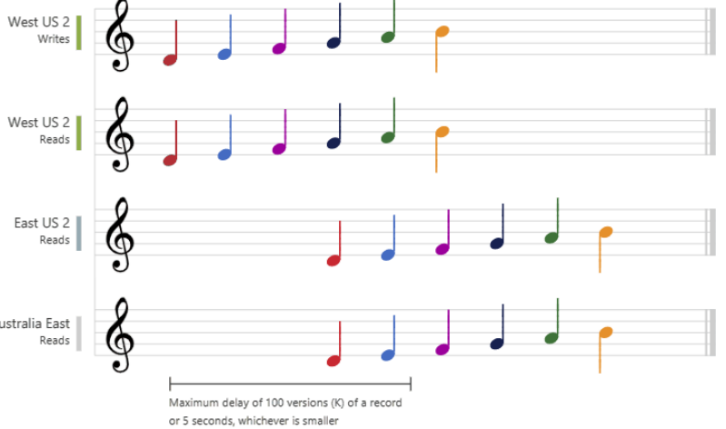
#### Strong consistency

Dá a ilusão de que existe uma única BD

Oferece:

* Numa leitura 🡪 garante 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 durante **staleness window**:

1. Quando só existe uma região de gravação:

* Para clientes:
  + dessa região 🡪 as atualizações são totalmente solicitadas imediatamente
  + noutras regiões 🡪 podem perder atualizações na **staleness window**, mas verão-as em ordem

1. Quando existem várias regiões de gravação:

* Para clientes que gravam:
  + Numa única região 🡪 observarão essas atualizações de atualizações em ordem
  + Em diferentes regiões 🡪 as atualizações são aplicadas localmente, mas podem ser reordenadas posteriormente (consistência eventual)

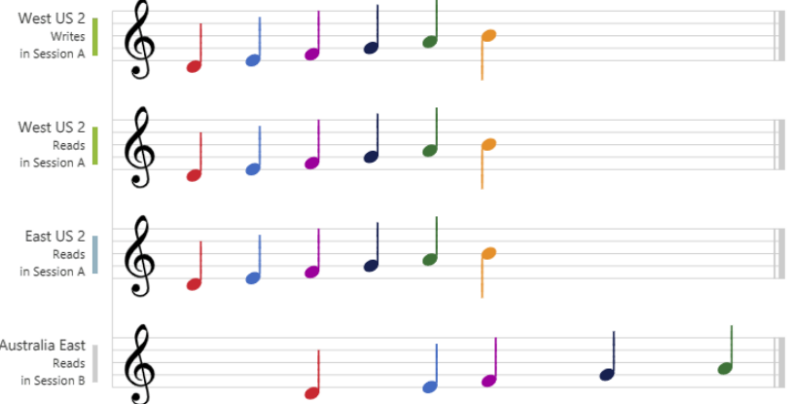
**Garante**

* Consistent-prefix - as atualizações são ordenadas, exceto as na **staleness window**
* Monotonic read - clientes observam uma versão posterior à lida antes
* A **staleness window** pode ser em tempo ou o nº de versões atrás da versão currente

**Implementação**

* A Master region ordena e propagação de atualizações para outras regiões.
* Uma região pode receber op. 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, mas com condições limitadas 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 - as atualizações são 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 🡪 atualizações em ordem;
* em outras regiões 🡪 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.

# Conflit resolution

Conflitos de . . .

* **Inserção** - insere simultaneamente dois ou mais itens com igual índice exclusivo em duas ou mais regiões.
* **Substituição** - atualiza o mesmo item simultaneamente em duas ou mais regiões.
* **Exclusão** - exclui simultaneamente um item em uma região e o atualiza noutra

Default: Last-Write-Wins (LWW):

# Durability

* **Recovery time objective(RTO)** - tempo até que um aplicativo se recupere de um evento disruptivo.
* **Recovery point objective(RPO)** - período de tempo em que as atualizações podem ser perdidas numa falha.

# Sharding

Um contêiner particionado horizontalmente em várias máquinas de acordo com uma partition key.

**Replication in the machines**

* Dentro de cada região, cada partição é replicada em um conjunto de réplicas. Todas as gravações são replicadas e confirmadas de forma durável pela maioria das réplicas.
* Cada partição é replicada entre regiões. Cada região contém todas as partições de dados de um contêiner Cosmos.
* Para uma conta distribuída em N regiões, haverá pelo menos N x 4 cópias dos dados.

# Change feed

* O Cosmos DB permite ouvir alterações em um contêiner do Azure Cosmos.
* O feed de alterações gera uma lista de docs alterados, na ordem em que foram modificados.
* As alterações são persistentes, podem ser processadas de forma assíncrona e incremental e a saída pode ser distribuída por um ou mais consumidores para processamento paralelo.

O Azure dá suporte a running computations na réplica local em cada região.

Replicação primária/de backup

Fragmentação com transações entre estilhaços

### 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 🡪 dados serão armazenados em caches de 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
* - tempo de processamento
* - carga do servidor

**Cache-Aside**

* Cache faz parte da DB para acesso mais rápido
* Modifica a cache ao modificar os dados de back-end
* - carga do servidor

**User session caching**

* Armazena informações associadas a uma sessão de usuário na cache
* Interação mais rápida
* 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
* conhecem a fun. hash para atribuir uma chave a um servidor
* enviam op. para o servidor que armazenará uma chave diretamente

Os servidores . . .

* mantêm um armazenamento de “valor-chave”

## Limitações do Memcached

**THUNDERING HERDS**

* Em cache miss 🡪 vários clientes podem acabar tentando definir o valor de uma entrada da 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.

**SEM REPLICAÇÃO**

* Em faso de falha - os clientes simplesmente vão para o disco.
* Se preciso recuperação - a cache é preenchida como consequência da execução normal dos clientes
* 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.

## Propriedades

* Armazenamento de valor-chave distribuído (tabela de hash)
* Valores são tipos: lista, set, sorted set, hashes, bit array e hyperloglogs

O Redis pode ser usado como DB, cache ou agente de mensagens.

* Defenições do Redis como cache:
  + quantidade limitada de memória;
  + algoritmo LRU para remoção de cache.
  + Possível definir um TTL para despejo;
  + Definir que somente chaves com um TTL podem ser despejadas
* O Redis oferece suporte a transações, incluindo uma sequência de operações.

## Arquitetura

* Cada servidor mantém pares chave-valor e executa operações nos valores.
* Possível criar um cluster de servidores Redis, com dados particionados.

Estratégias de particionamento disponíveis:

* + Particionamento de intervalo;
  + Hashing (alguns clientes implementam hash consistente).
* O Redis oferece suporte à replicação, mas a replicação normalmente não é usada para armazenamento em cache.
* Backup primário, com replicação assíncrona.
* Replicação multimestre com resolução automática de conflitos (com base em CRDTs).

## Persistencia

Várias alternativas:

1. **Sem persistência**
2. **RDB persistence**: executa point-in-time snapshots da database em certos intervalos
3. **Append-only fashion**: registra todas as op. de gravação recebidas pelo servidor que serão reproduzidas na inicialização do servidor

# Caching and consistency

Usar um cache distribuído apresenta problemas de consistência para dados na cache e na DB.

* Com CosmosDB e Redis, uma solução possível é usar o timestamp associado ao documento para garantir que a versão mais recente seja mantida no cache.
* Ao definir o valor na cache, o valor antigo pode ser retornado, permitindo que o cliente verifique se ele sobrescreveu um valor mais recente.

### CDN

Uma rede de entrega de conteúdo (CDN) é uma rede distribuída de edge servers que armazenam conteúdo em cache em locais point-of-presence (POP) próximos aos usuários finais, para minimizar a latência.

* Menor latência para clientes
* Grande escala para lidar melhor com altas cargas instantâneas
* Reduza o tráfego enviado ao servidor de origem

## Caching rules

Resultados da tradução:

* Todos os pedidos
* Solicitação de determinados caminhos, extensões
* Solicitações com determinada string de consulta
* Possíveis incluem definir se uma página deve ser armazenada em cache ou não e o TTL para armazenamento em cache.

## Advanced cache features

**Route Optimization**

Escolhe a rota desda origem até aos utilizadores finais, para que a entrega seja feita pela rota mais rápida e confiável possível

**TCP Optimizations**

Várias otimizações para TCP para acelerar a comunicação.

1. Eliminando o início lento do TCP.
2. Aproveitando conexões persistentes.
3. Ajustando os parâmetros do pacote TCP.

**Object prefetch**

Quando o cliente faz as requisições dos ativos vinculados, o servidor de edge CDN já possui os objetos solicitados e pode atendê-los imediatamente sem ida e volta até a origem

**Adaptive image compression**

Monitora a qualidade da rede e emprega métodos de compactação JPEG padrão quando as velocidades da rede são mais lentas para melhorar o tempo de entrega

**Zone-based restriction**

Permite restringir o acesso ao conteúdo por país/região

**DDoS protection**

Uma rede de entrega de conteúdo fornece proteção contra DDoS por design, sendo capaz de absorver ataques volumétricos.

### Serverless

# Function-as-a-Service

As **cloud functions** são pequenos pedaços de código, escritos em uma linguagem de alto nível. Uma **cloud function** é invocada quando ocorre algum evento (carregar img na cloud).

O **serverless system** é responsável pelas tarefas de provisionamento e administração. A **serverless computing** é feita em servidores, mas as tarefas de provisionamento e administração de servidores são feitas pelo cloud provider.

Um **serverless service** deve ser dimensionado automaticamente sem necessidade de provisionamento explícito e ser cobrado com base no uso.

Outras **serverless frameworks** incluem ofertas de BaaS (Backend as a Service).

## Serverless (vs. Serverfull)

**Computação e armazenamento desacoplados.**

* O armazenamento e computação são dimensionados separadamente
* São provisionados e precificados de forma independente.
* O armazenamento é fornecido por um serviço na cloud separado e a computação não tem estado.

**Alocação automática de recursos.**

* Em vez de solicitar recursos, o usuário fornece um pedaço de código e a nuvem provisiona automaticamente recursos para executar esse código.

**Pagamento proporcional aos recursos utilizados em vez dos recursos alocados.**

* O faturamento é por alguma dimensão associada à execução, como o tempo de execução, e não por uma dimensão da plataforma de nuvem básica, como tamanho e número de VMs alocadas.

## FaaS

O FaaS deve ser provisionado rapidamente e oferecer segurança forte. VM e contêineres são a chave para tornar o FaaS possível.

Para agilizar a criação dos ambientes de execução para execução da função, várias otimizações são feitas pelos usuários:

* **“warm pool”** de instâncias de VM que só precisam ser atribuídas a um locatário;
* **“pool ativo”** de instâncias que foram usadas para executar uma função antes e são mantidas para atender a chamadas futuras.

# Azure Functions

O **Azure Functions** é uma solução para executar facilmente pequenos pedaços de código, ou "funções", na nuvem.

Pode se escrever apenas o código necessário para o problema em questão, sem se preocupar com o aplicativo completo ou com a infraestrutura para executá-lo

## Function triggers

**HTTPTrigger -** função executada em uma solicitação HTTP.

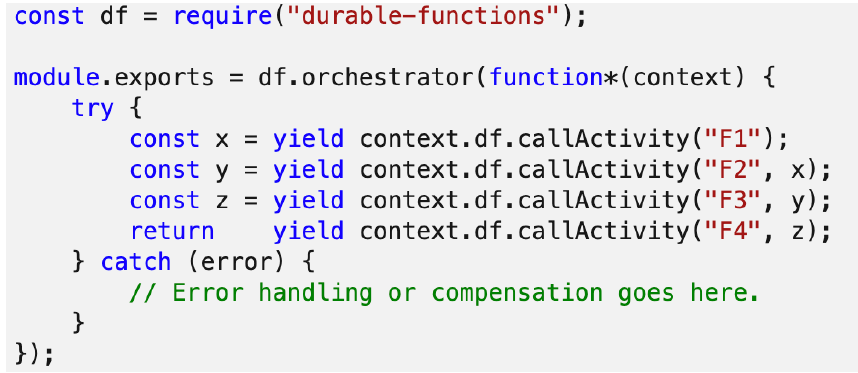
**TimerTrigger -** execute a limpeza ou outras tarefas em lote em uma programação predefinida.

**CosmosDBTrigger -** Processe documentos do Azure Cosmos DB quando eles forem adicionados ou atualizados.

**BlobTrigger -** processa blobs de armazenamento do Azure quando eles são adicionados a contêineres.

**EventHubTrigger -** Responda a eventos entregues a um Azure

## Durable Functions

Durable Functions é uma extensão do Azure Functions para escrever funções com estado.

NOTA: **Lambda@Edge** é um serviço que permite executar funções Lambda nos pontos de presença da Amazon (CDN). Como outros serviços FaaS, os clientes pagam apenas o tempo de computação que consomem.

## Exemplos

* Adicione cabeçalhos de segurança HTTP em todas as respostas de origem.
* Execute aplicativos da Web na edge, em vez de executá-los nos data centers.
* Transforme imagens em tempo real para personalizar a experiência dos usuários com base nas características dos dispositivos dos usuários.

## FaaS challenges

**Baixo desempenho para padrões de comunicação padrão** - Broadcast, agregação e embaralhamento são algumas das primitivas de comunicação mais comuns em sistemas distribuídos.

**Desempenho previsível** - Os atrasos incorridos ao iniciar novas instâncias podem ser altos para alguns aplicativos e são imprevisíveis

### Serviços para micro-service based applications

Diferentes funcionalidades são separadas em pequenos serviços, e esses serviços são deployed de forma independente podendo ter multiplas instâncias deles em servidores separados

**Vantagens**

Faceis de programar e manter´

Facil de evoluir o sistema e adaptar novas tecnologias

Possivel escalar cada micro serviço independentemente

Microserviços tem falhas independentes

**Desvantagens**

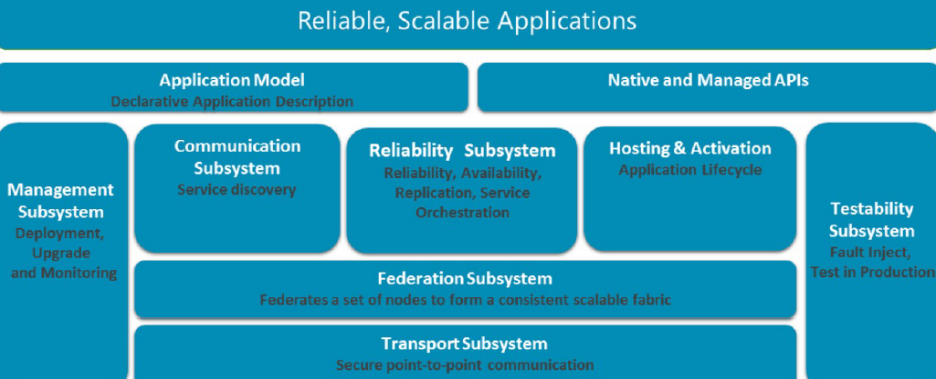
Monitorização e management

Configuarção e deploying

Aplication support

Debugging e teste

# Azure Service Fabric



## Transport Subsystem

Usado para comunicação dentro do cluster e com clientes

## Federation Subsystem

Serve para gerenciar um cluster. Fornece: detecção de falhas, eleição de líderes e roteamento consistente.

Construido sobre uma DHT

## Reliability subsystem

Fornece o mecanismo para tornar o estado de um serviço do Service Fabric altamente disponível

**Replicator –** replica o stado do micro serviço

**Failover manager –** reconfigura o cluster

**Resource manager –** manages the resources

## Managment subsystem

Fornece serviço de ponta a ponta e gerenciamento do ciclo de vida do aplicativo

**Cluster manager -** gerencia o ciclo de vida dos aplicativos desde o provisionamento até o desprovisionamento.

**Health manager –** serviço que permite o monitoramento da integridade de aplicativos, serviços e entidades de cluster.

**Image Store -** Este serviço fornece armazenamento e distribuição dos binários do aplicativo

## Comunication subsystem

fornece mensagens confiáveis dentro do cluster e descoberta de serviço por meio do serviço de nomenclatura. O serviço de nomenclatura resolve nomes de serviço para um local no cluster

## Programimg models

**Reliable services**

O Reliable Services é uma estrutura leve para escrever serviços que se integram à plataforma Service Fabric e se beneficiam do conjunto completo de recursos da plataforma

**Reliable actors**

Construído sobre o Reliable Services, o framework Reliable Actor implementa o padrão Virtual Actor, baseado no padrão de design do ator