

Module 14:

Azure Cosmos DB



Cosmos DB

A fully-managed, highly-scalable, NoSQL document database service.



Schema free storage,
indexing and query
of JSON documents



Transaction aware
service side
programmability
with JavaScript



Write optimized, SSD
backed and tuneable
via indexing and
consistency

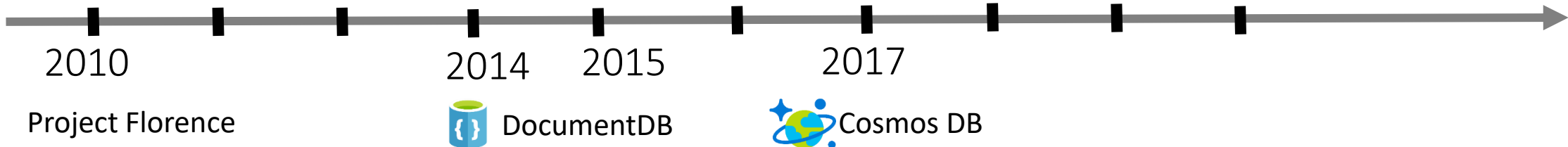


Built to be delivered
as a service. Pay as
you go. Achieve
faster time to value.

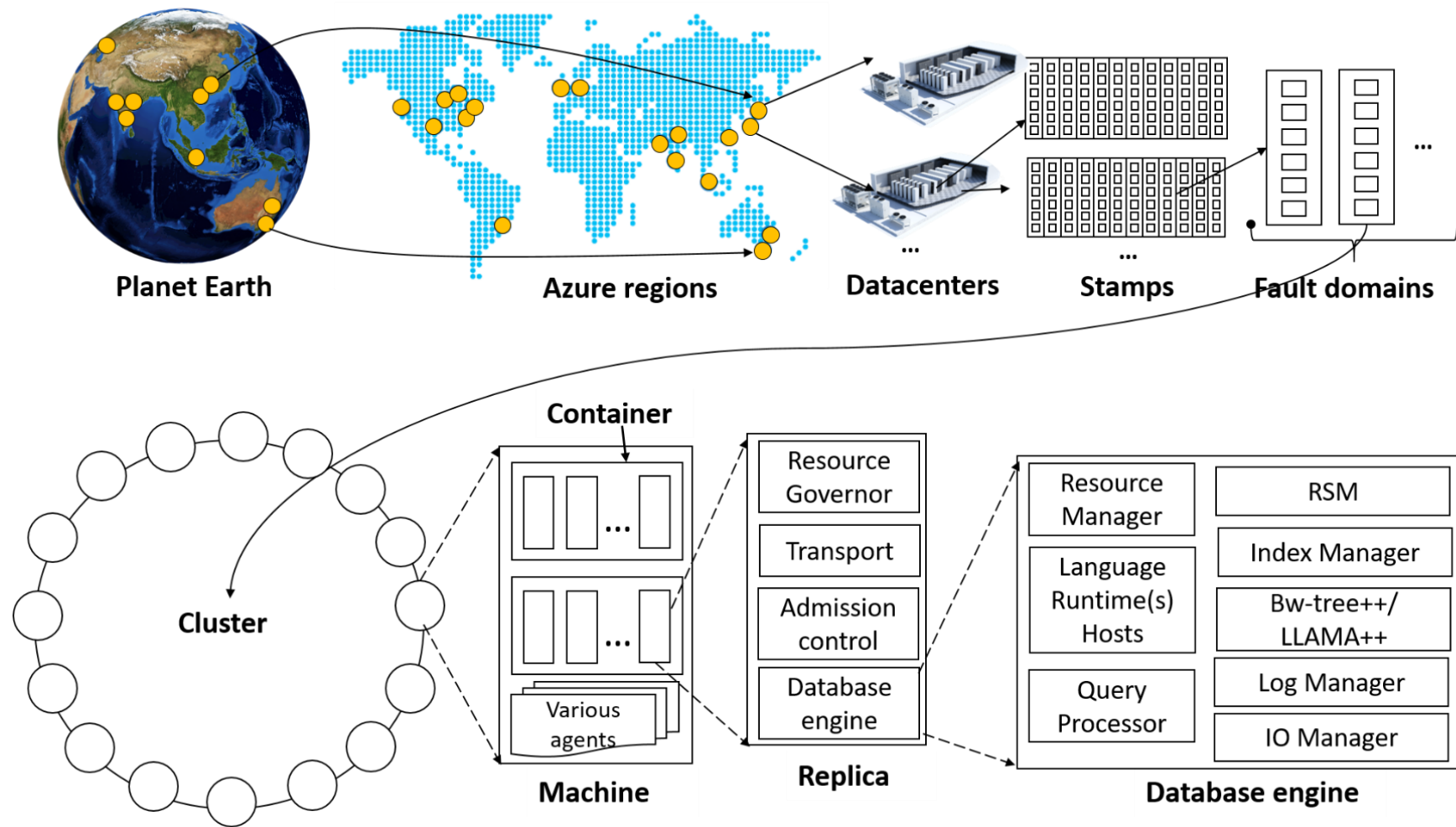
Azure Cosmos DB Evolution



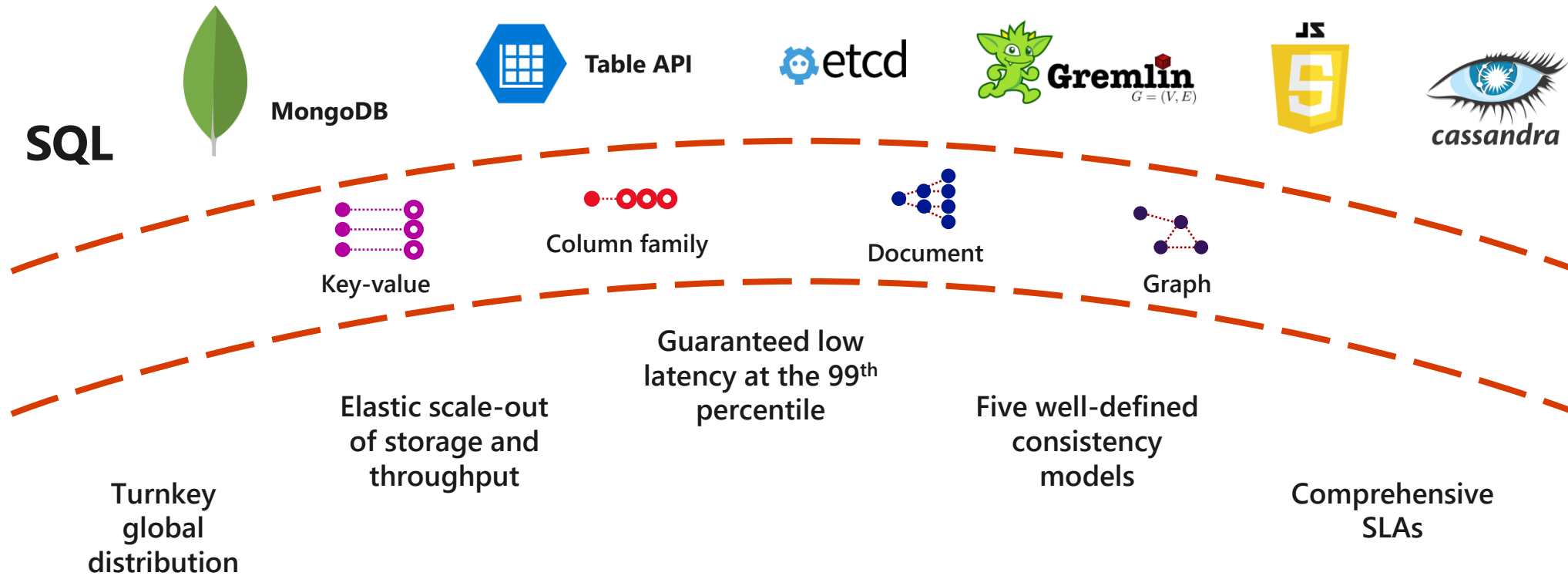
- Originally started to address the problems faced by large scale apps inside Microsoft
- Built from the ground up for the cloud
- Used extensively inside Microsoft
- One of the fastest growing services on Azure



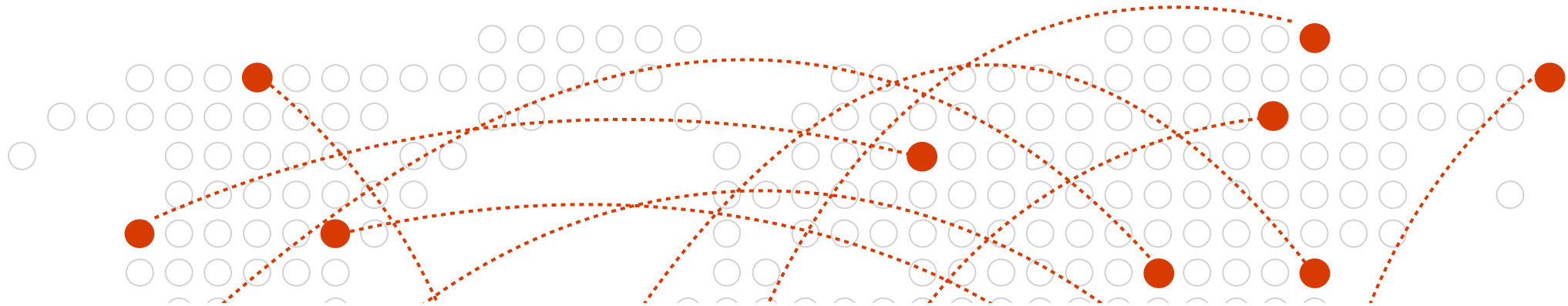
System topology (behind the scenes)



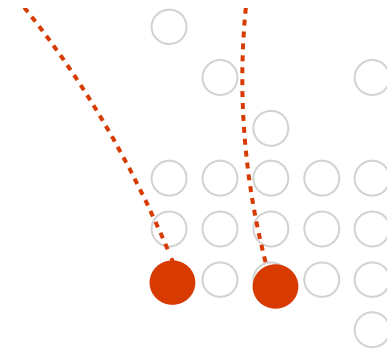
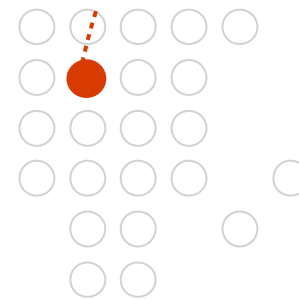
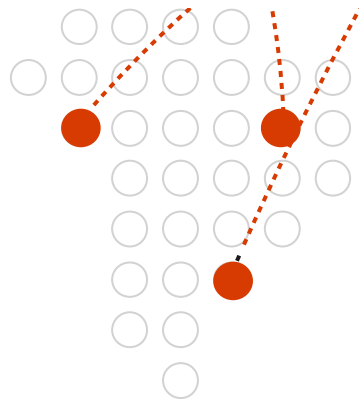
Azure Cosmos DB



Global Replication



Turnkey global distribution automatically replicates data to other Azure datacenters across the globe without the need to manually write code or build a replication infrastructure



APIs



- MongoDB API

- Acts as a massively scalable MongoDB service powered by the Azure Cosmos DB platform
- Compatible with existing MongoDB libraries, drivers, tools, and applications



- Table API

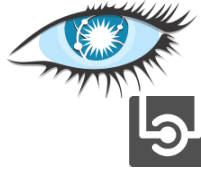
- A key-value database service built to provide premium capabilities to existing Azure Table storage applications without making any app changes



- Gremlin API

- A fully managed, horizontally scalable graph database service
- Easy-to-build and run applications that work with highly connected datasets supporting Open Graph APIs (based on the Apache TinkerPop specification, Apache Gremlin)

APIs (cont.)



- Cassandra API

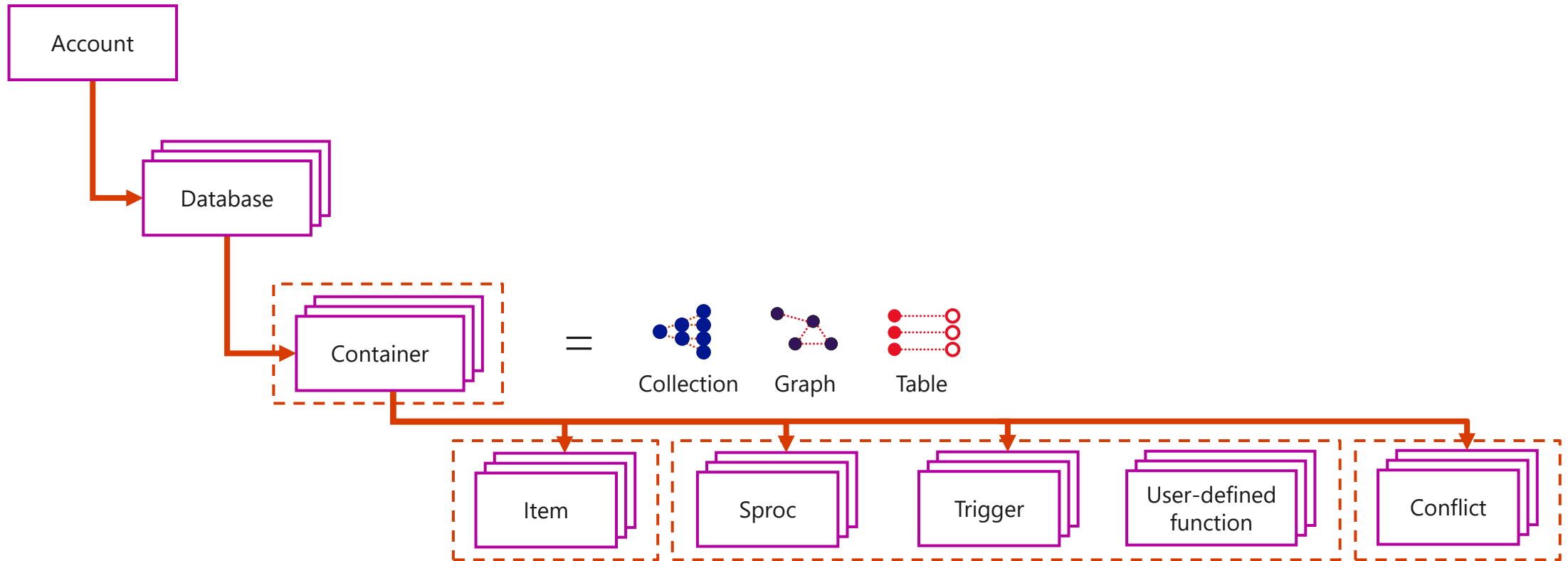
- Globally distributed Apache Cassandra service powered by the Azure Cosmos DB platform
- Compatible with existing Apache Cassandra libraries, drivers, tools, and applications

- SQL API

- JavaScript and JavaScript Object Notation (JSON) native API based on the Azure Cosmos DB database engine
- Provides query capabilities rooted in SQL
- Query for documents based on their identifiers or make deeper queries based on properties of the document, complex objects, or the existence of specific properties
- Supports the execution of JavaScript logic within the database in the form of stored procedures, triggers, and user-defined functions



Resource hierarchy



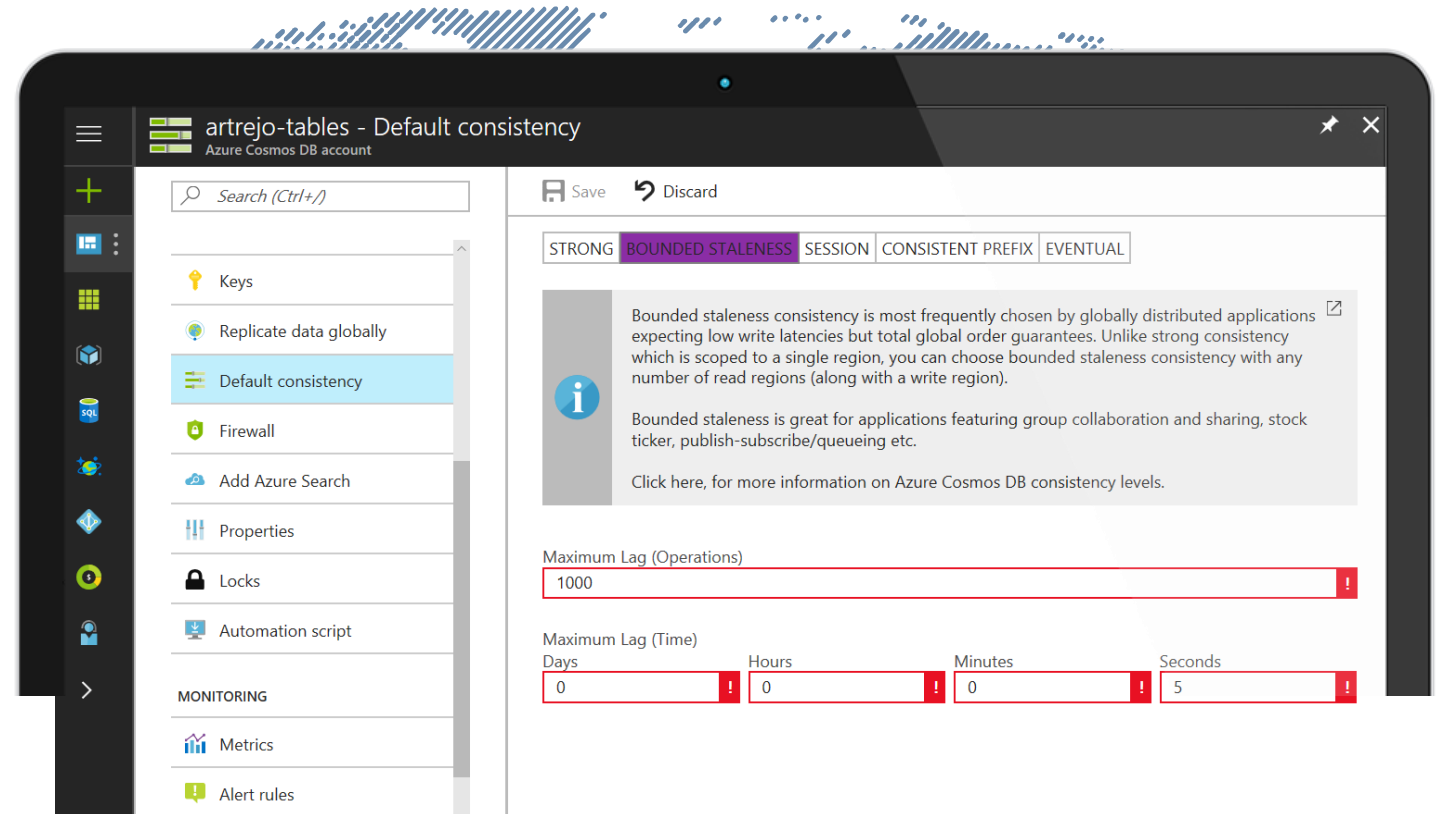
Resource hierarchy (continued)

Resource	Description
Account	A set of databases
Database	Logical container for containers that can (optionally) share throughput across the containers
Collection (container)	A group of Items and programmatic resources usually related in some way
Document (item)	An arbitrary unit of content In many cases, this would be a JSON document
Stored procedure (sproc)	Application logic written in JavaScript executed within the database engine as a transaction
Trigger	Application logic written in JavaScript executed before or after either an insert, replace, or delete operation
User-defined function	Application logic written in JavaScript to extend the SQL API query language

Well-defined consistency models

- Intuitive programming model
- 5 Well-defined, consistency models
- Overridable on a per-request basis

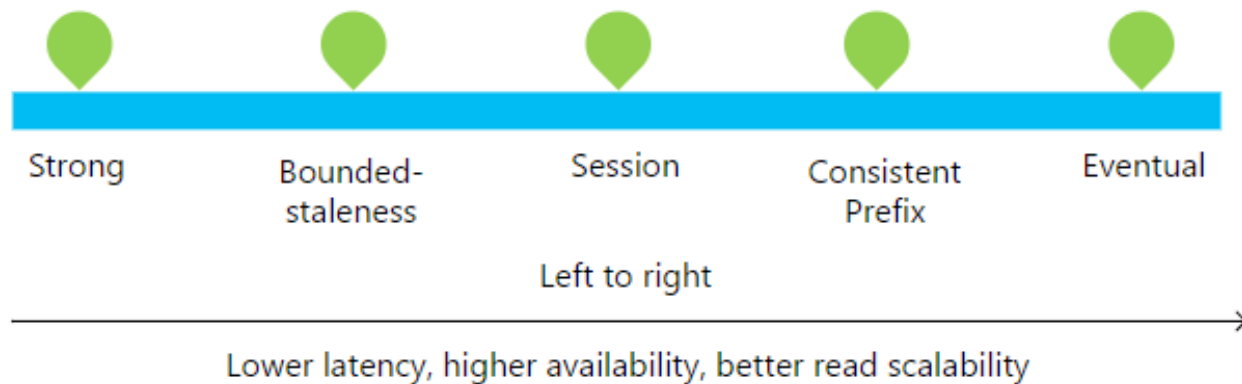
- Clear tradeoffs
- Latency
- Availability
- Throughput



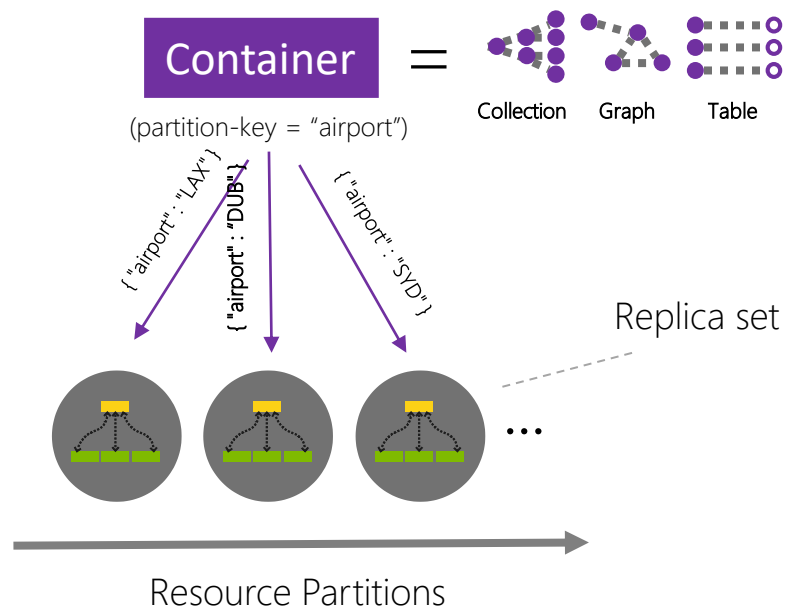
Consistency levels

Azure Cosmos DB provides five consistency levels:





Consistency Level	Guarantees
Strong	Linearizability (once operation is complete, it will be visible to all)
Bounded Staleness	Consistent Prefix. Reads lag behind writes by at most k prefixes or t interval Similar properties to strong consistency (except within staleness window), while preserving 99.99% availability and low latency.
Session	Consistent Prefix. Within a session: monotonic reads, monotonic writes, read-your-writes, write-follows-reads Predictable consistency for a session, high read throughput + low latency
Consistent Prefix	Reads will never see out of order writes (no gaps).
Eventual	Potential for out of order reads. Lowest cost for reads of all consistency levels.



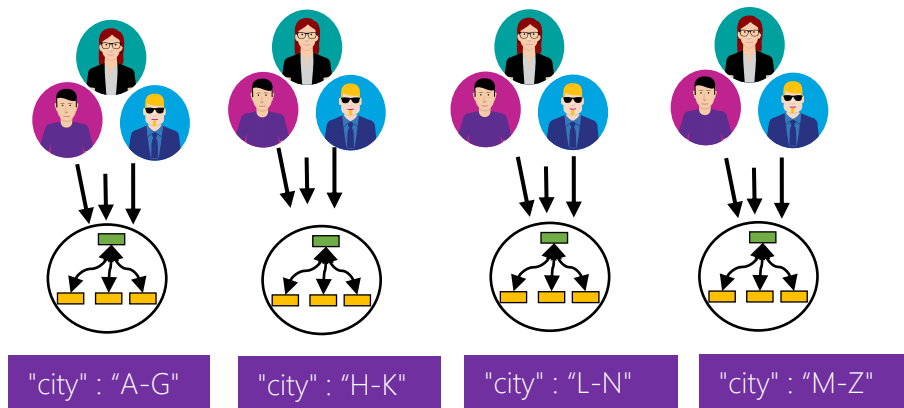
Horizontal Partitioning

Containers are horizontally partitioned

Each partition made highly available via a replica set

Partition management is transparent and highly responsive

Partitioning scheme is dictated by a "partition-key"



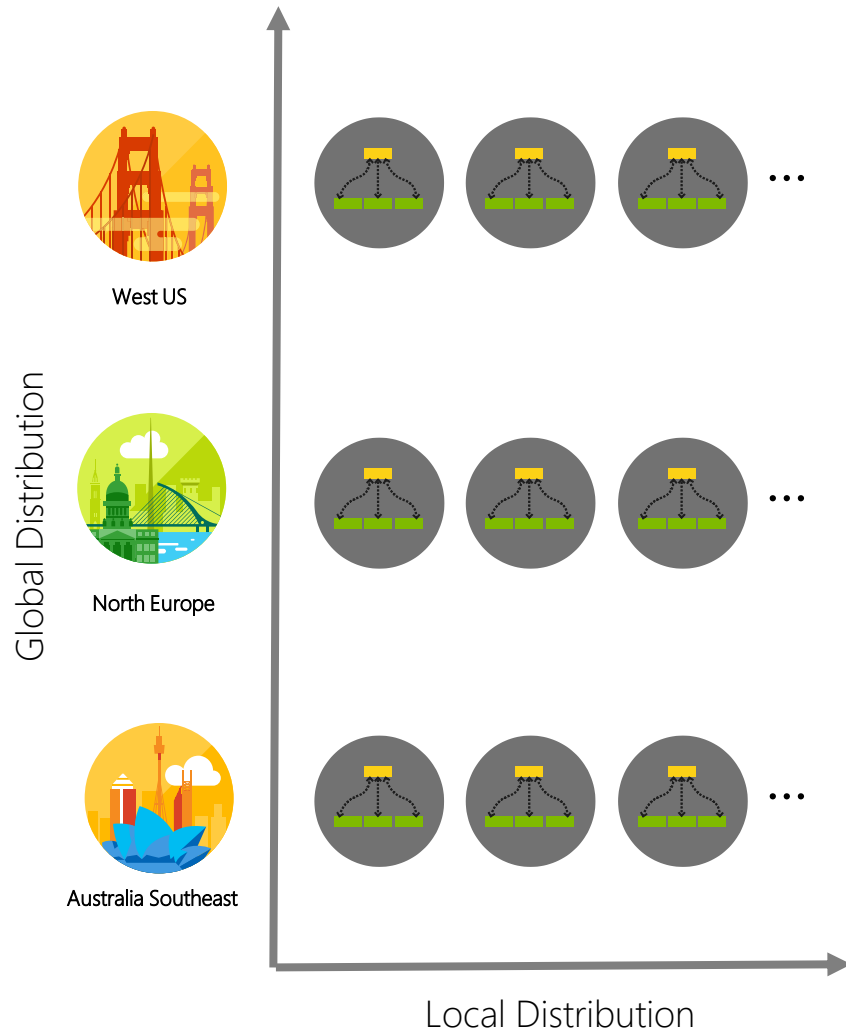
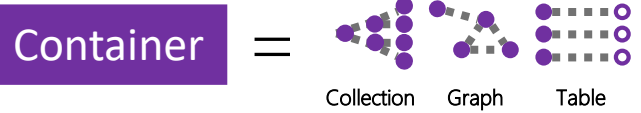
Best Practices: Partitioning

All items with the same partition key will be stored in the same partition

Multiple partition keys may share the same partition using hash-based partitioning

Select a partition key which provides even distribution of storage and throughput (req/sec) at any given time to avoid storage and performance bottlenecks

Do not use current timestamp as partition key for write heavy workloads



Global Distribution

All resources are horizontally partitioned and vertically distributed

Distribution can be within a cluster, x-cluster, x-DC or x-region

Replication topology is dynamic based on consistency level and network conditions

Internet of Things – Telemetry & Sensor Data

- **Business Needs:**
 - High scalability to ingest large # of events coming from many devices
 - Low latency queries and changes feeds for responding quickly to anomalies
 - Schema-agnostic storage and automatic indexing to support dynamic data coming from many different generations of devices
 - High availability across multiple data centers

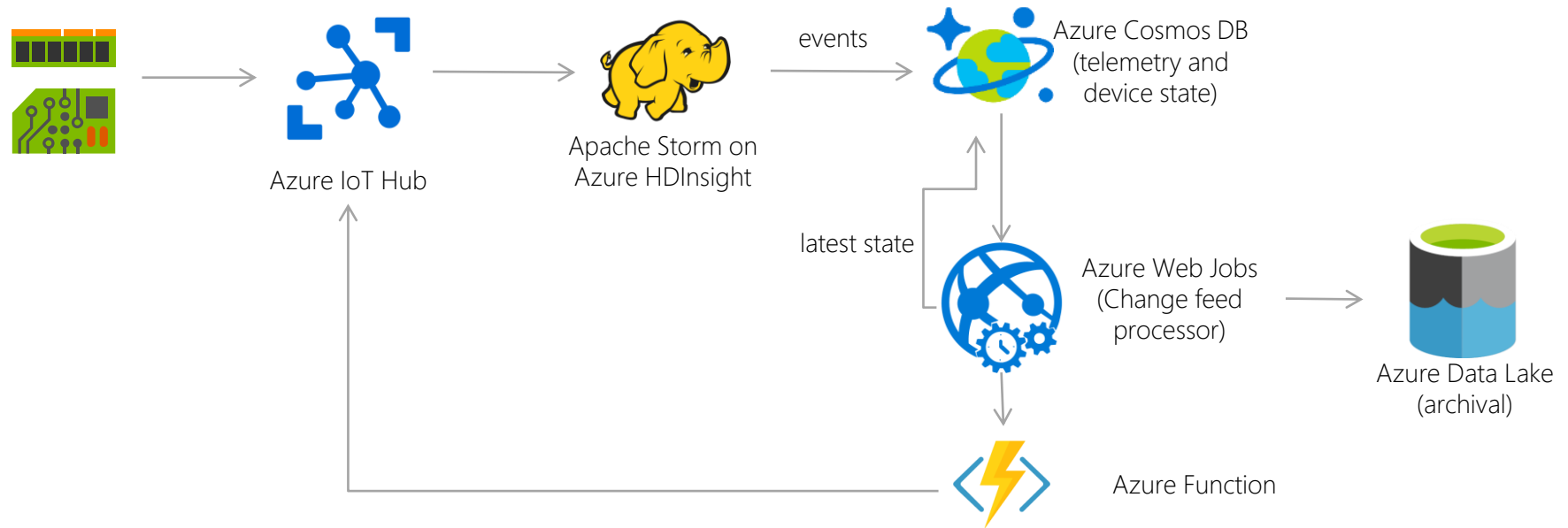
Microsoft Azure



Honeywell



Internet of Things – Telemetry & Sensor Data

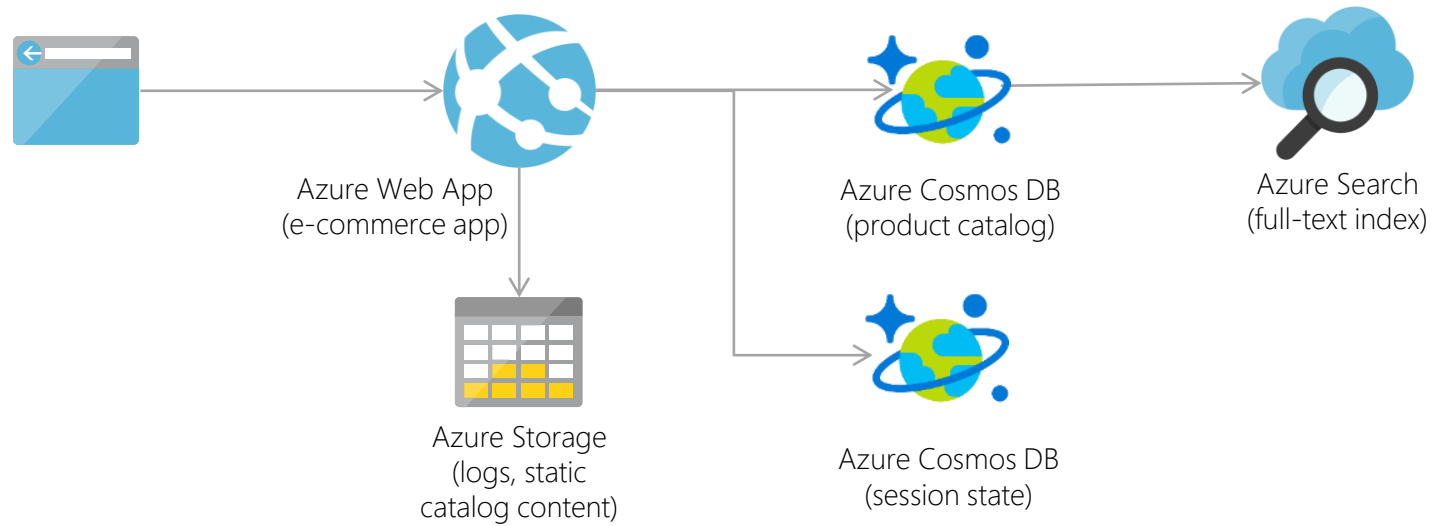


Retail – Product Catalog & Order Processing

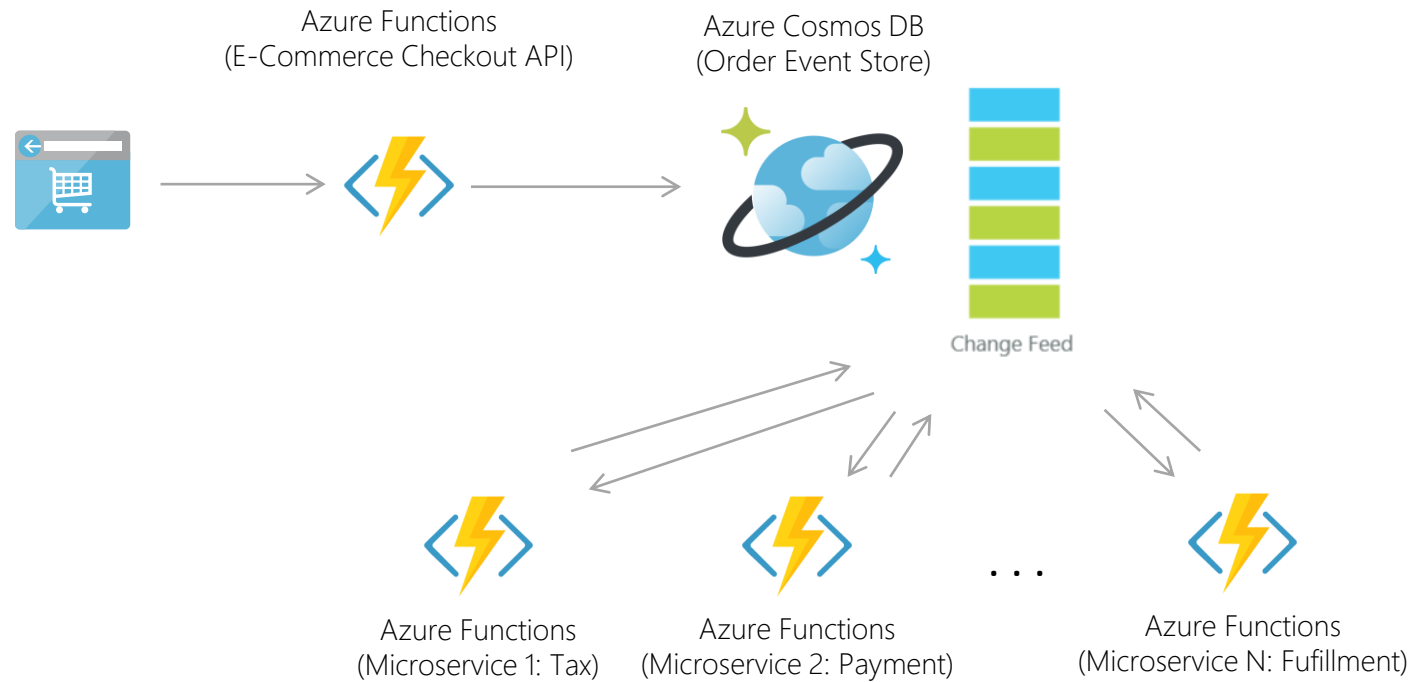
- **Business Needs:**
 - Elastic scale to handle seasonal traffic (e.g. Black Friday)
 - Low-latency access across multiple geographies to support a global user-base and latency sensitive workloads (e.g. real-time personalization)
 - Schema-agnostic storage and automatic indexing to handle diverse product catalogs, orders, and events
 - High availability across multiple data centers



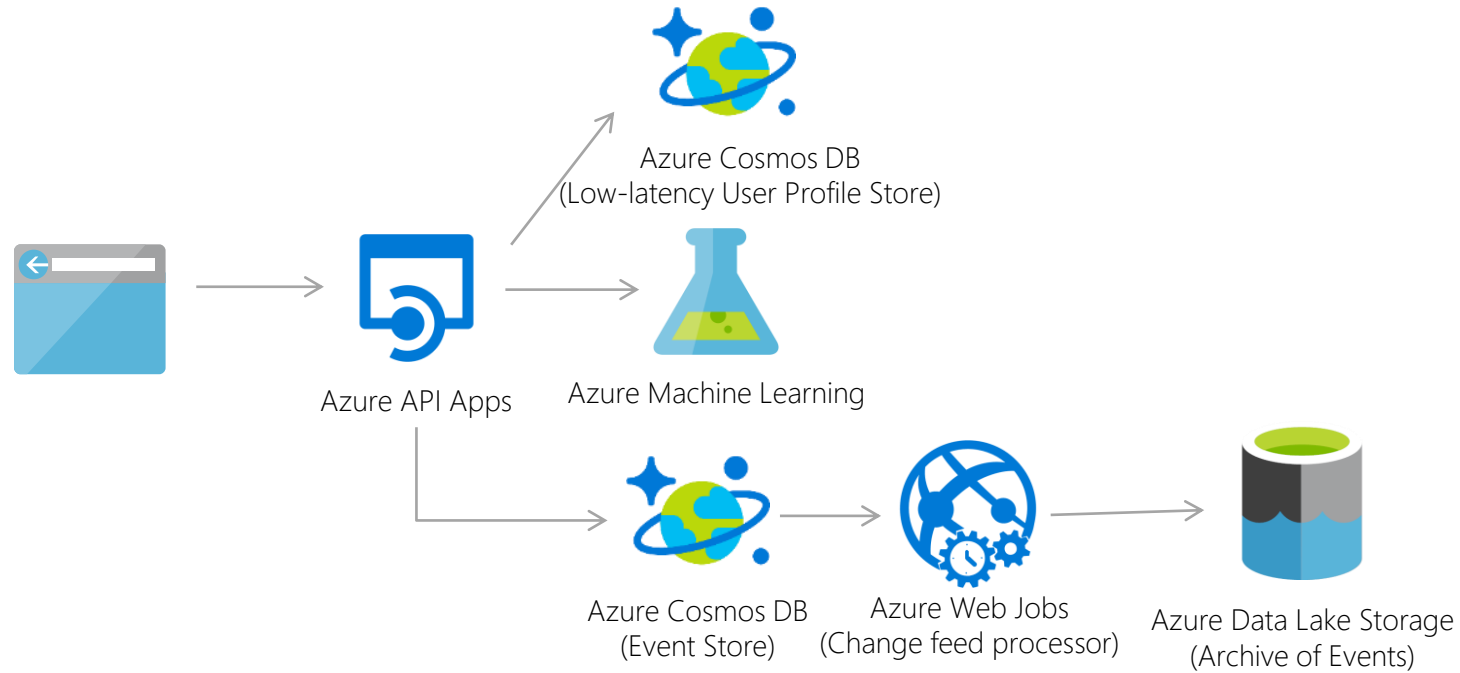
Retail Product Catalogs



Retail Order Processing Pipelines



Real-time Personalization / Recommendations

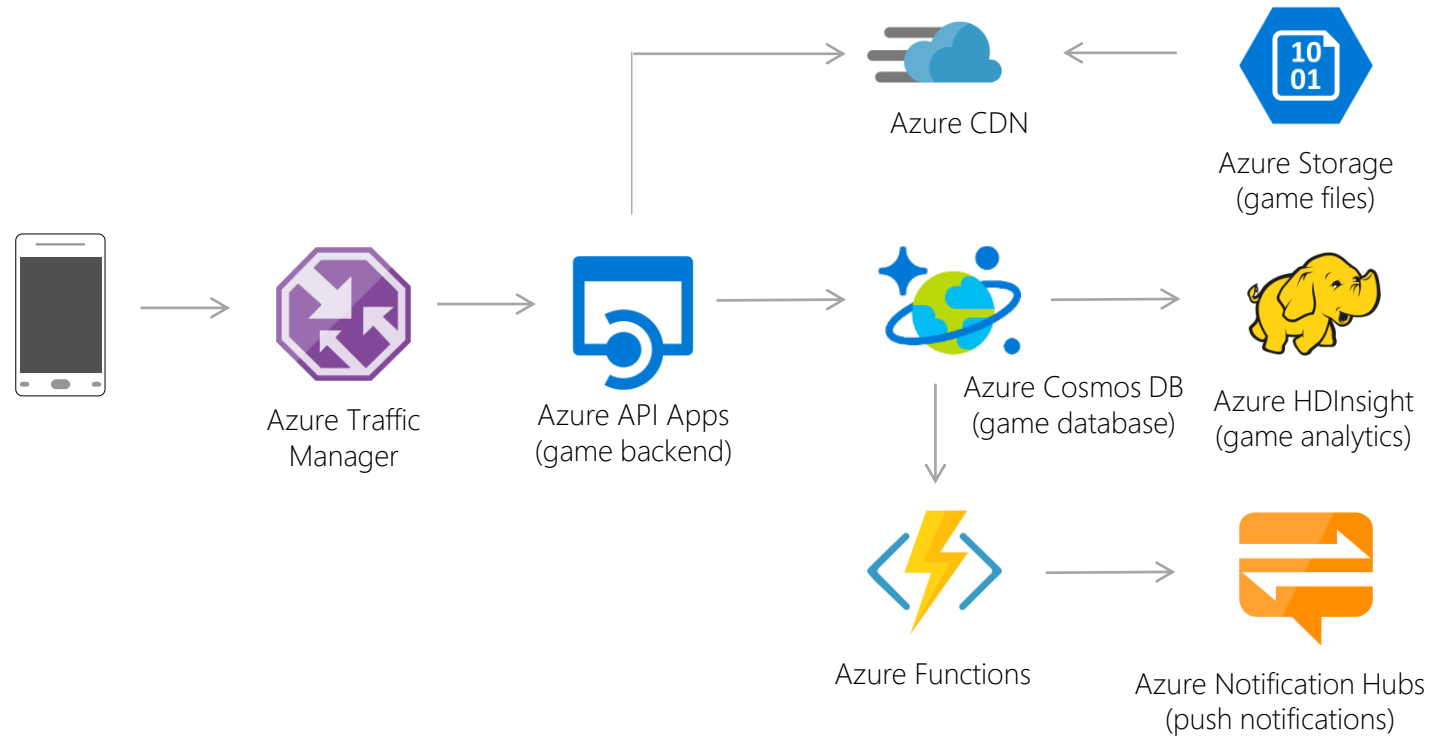


Multiplayer Gaming

- **Business Needs:**
 - Elastic scale to handle bursty traffic on day
 - Low-latency queries to support responsive gameplay for a global user-base
 - Schema-agnostic storage and indexing allows teams to iterate quickly to fit a demanding ship schedule
 - Change-feeds to support leaderboards and social gameplay



Multiplayer Gaming



	Azure Table Storage	Azure Cosmos DB
Throughput	Up to 20K operations per second	No upper limit, supports > 10M operations per second
Redundancy	One optional secondary read-only region	Multiple configurable worldwide regions
Latency	No upper bounds	Single-digit milliseconds
Consistency	Strong and Eventual consistency models	Five defined consistency models
Query	Optimized for query on primary key	Improved query performance because all fields are indexed
Failover	Can't initiate failover	Automatic and manual failovers
Billing	Storage-based	Throughput-based

Demo & Lab:

- Create Cosmos DB
- Working with SDK

