Azure SQL - Basics

Agenda

- What is Azure SQL?
- Deployment Models
- Setup - SQL Server, Azure SQL Database, Firewall
- Connect through Azure Data Studio
- Database Security Features

Azure SQL

- Fully-managed, database services in Azure
- Create only databases in the cloud
- Provides latest stable release of SQL Server Database
- Scalable and highly available
- Supports geographic replication and failover

Deployment Models

- **SQL Server on Azure VM**
  - Full control on SQL Server, databases, and underlying infrastructure
  - Deployed inside Virtual Network
- **Azure SQL databases**
  - Most features of on-prem SQL Server are supported (some are not)
  - Cheaper, and highly-scalable
  - Supports SQL Server, MySQL and PostgreSQL databases
- **Azure SQL Managed Instance**
  - Nearly compatible with on-prem SQL Server
  - Managed service; deployed inside Virtual Network

Azure SQL - Advanced

Agenda

- Deployment Models
- Purchasing Models
- Resource Sharing Options
- Service Tiers
- Security Features
- Geo replication & Failover groups
- High Availability

Azure SQL

- Fully-managed, database services in Azure
- Create only databases in the cloud
- Provides latest stable release of SQL Server Database
- Scalable and highly available
- Supports geographic replication and failover

Deployment Models

- **SQL Server on Azure VM**
  - Full control on SQL Server, databases, and underlying infrastructure
  - Deployed inside Virtual Network
- **Azure SQL databases**
  - Most features of on-prem SQL Server are supported (some are not)
  - Cheaper, and highly-scalable
  - Supports SQL Server, MySQL and PostgreSQL databases
- **Azure SQL Managed Instance**
  - Nearly compatible with on-prem SQL Server
  - Managed service; deployed inside Virtual Network

Purchasing Models

- **DTU based Model**
  - DTU stands for Database Transaction Unit
  - Combination of compute, storage and IOPS
  - Pre-configured resource option - Pay only for DTUs
- **vCore based Model**
  - Select & scale independently - compute, storage and IOPS
  - More flexible and transparent
  - Azure Hybrid Benefit – use existing licenses to save cost

Resource Sharing Options

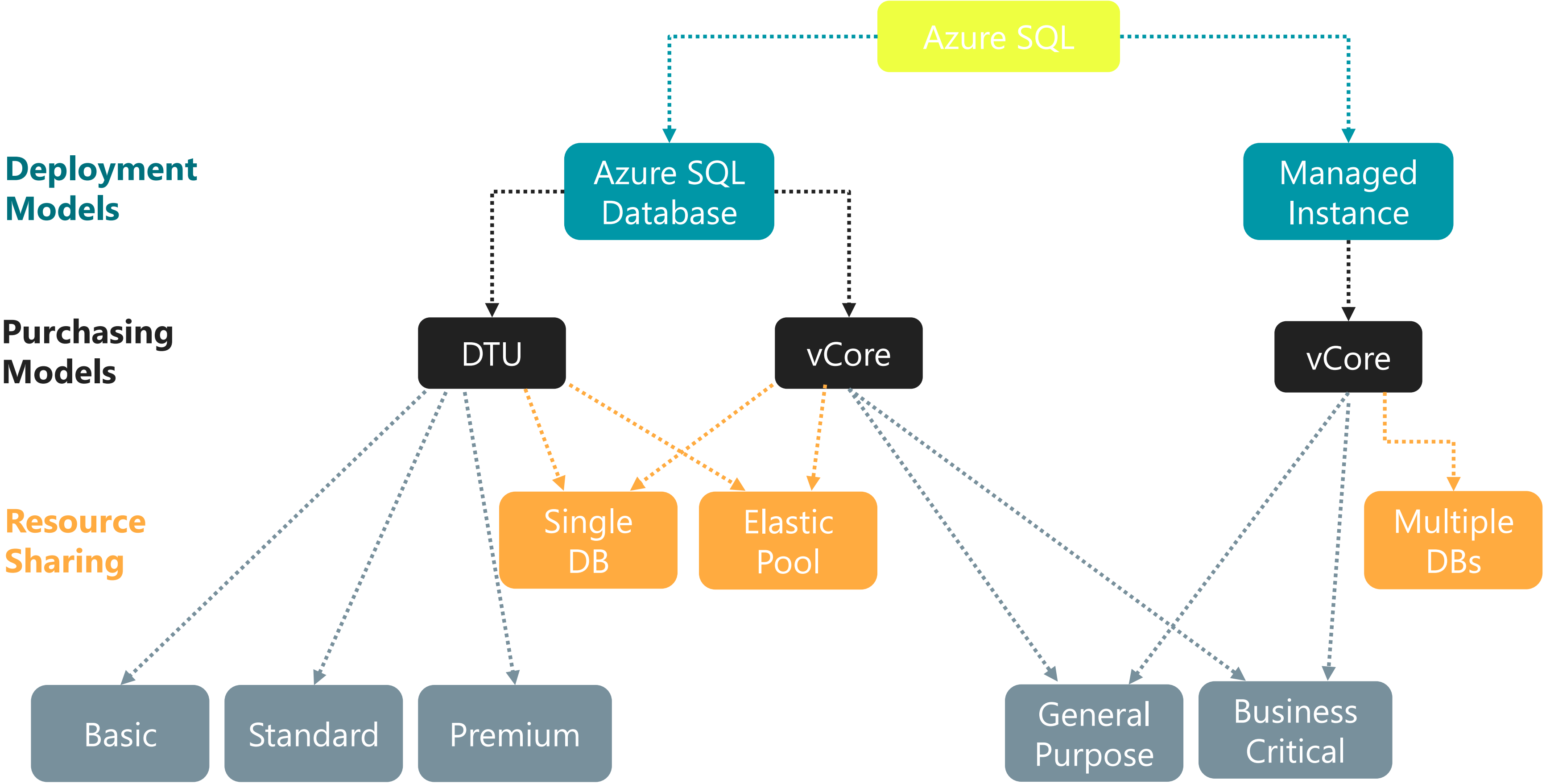
- Only available in Azure SQL database
- Available for both DTU and vCore purchase models

- **Single Database**
  - Each database gets it own set of dedicated resources

- **Elastic Pool of Databases**
  - Creates a pool of resources (compute, IOPS etc.)
  - Multiple databases can run on the same pool, sharing the resources

Service Tiers

- **DTU model**
  - Basic
  - Standard
  - Premium
- **vCore model**
  - General purpose
  - Business critical
  - Hyperscale


Azure SQL
Deployment Models
Azure SQL Database
Managed Instance
Purchasing Models
DTU
vCore
vCore
Resource Sharing
Single DB
Elastic Pool
Multiple DBs
Basic
Standard
Premium
General Purpose
Business Critical

High Availability

- Standard Availability Model
- Premium Availability Model

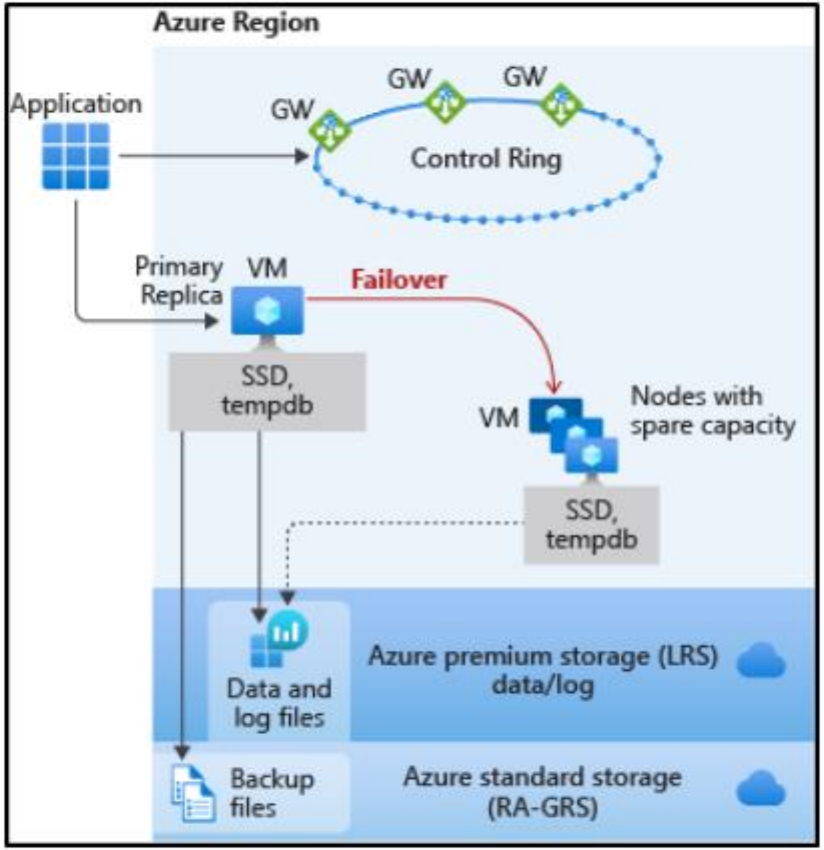
Standard Availability Model

Stateless Compute Layer

- Runs in a VM
- Only contains cached & transient data in SSD
- On crash, auto failover is performed by using spare capacity

Stateful Data Layer

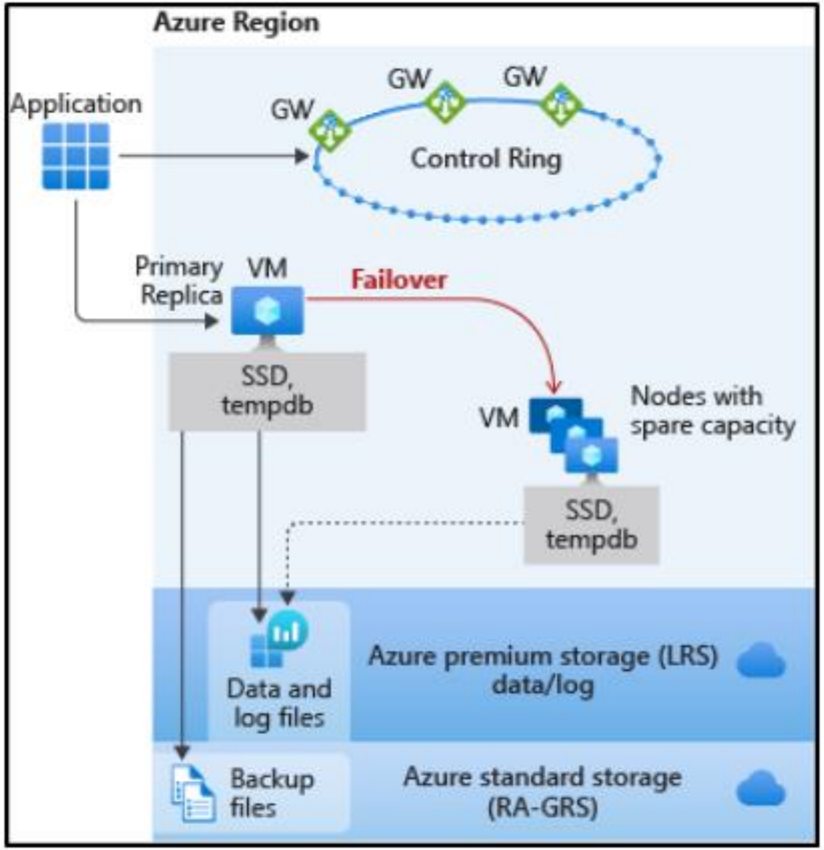
- Data is stored in Premium Storage (mdf/ldf files)
- Data is safe even if machine with SQL engine crashes



Source: https://docs.microsoft.com/en-us/azure/azure-sql/database/high-availability-sla

Standard Availability Model

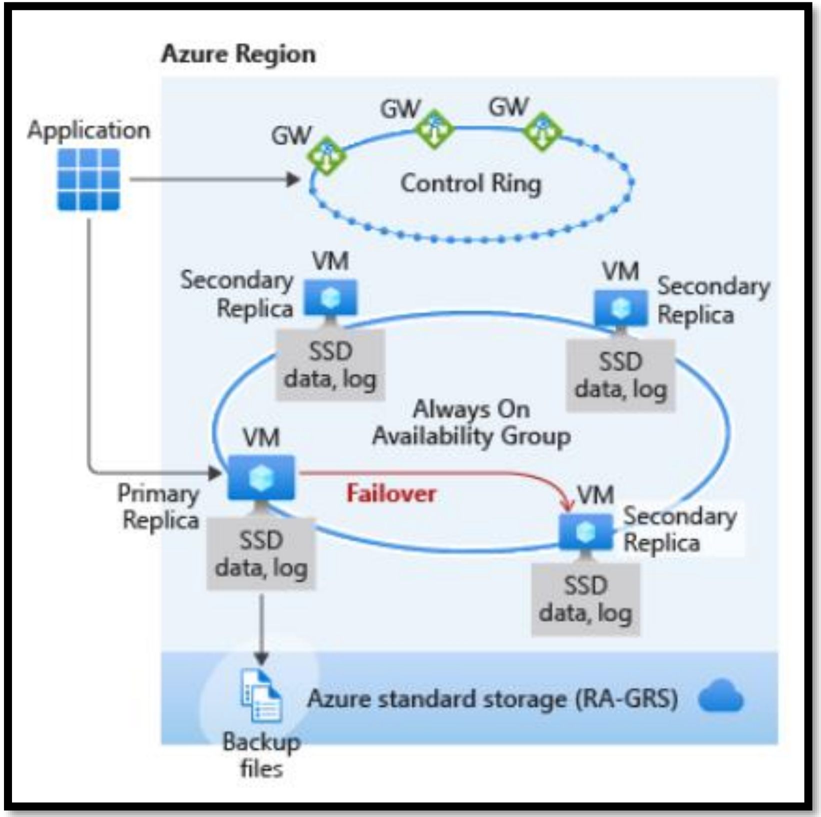
- Cheaper option
- Less performant than Premium availability model
- Service Tiers supported
  - Basic
  - Standard
  - General Purpose



Source: https://docs.microsoft.com/en-us/azure/azure-sql/database/high-availability-sla

Premium Availability Model

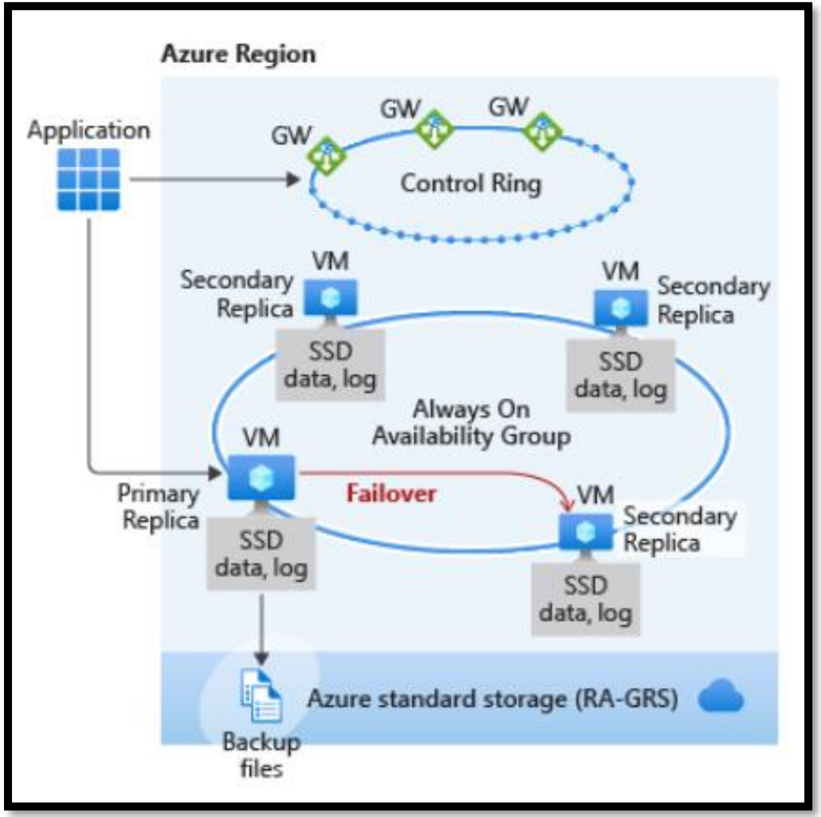
- Compute & data are on same node
- Data is stored on attached SSDs (mdf/ldf files)
- Compute & data are replicated on multiple secondary nodes in cluster
- Uses Always On Availability Groups to provide high availability
- If primary node crashes, failover is performed



Source: https://docs.microsoft.com/en-us/azure/azure-sql/database/high-availability-sla

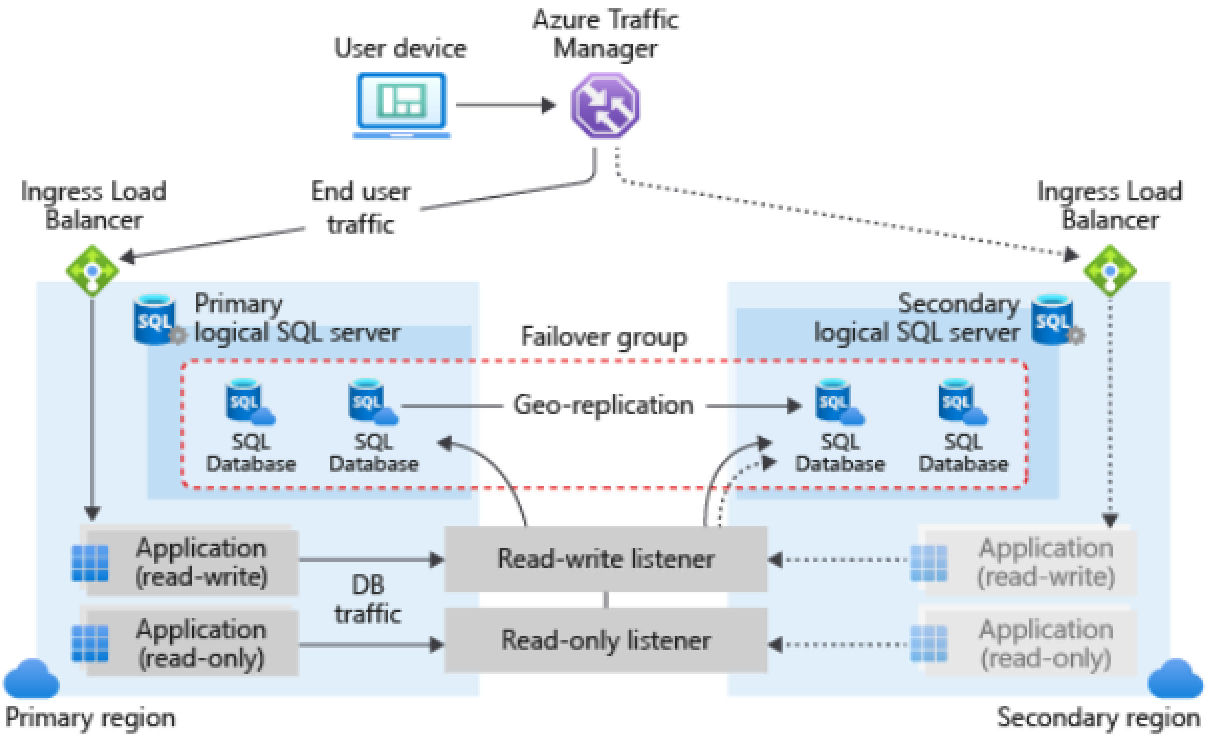
Premium Availability Model

- Expensive option
- More performant that Standard availability model
- Service Tiers supported
  - Premium
  - Business critical



Source: https://docs.microsoft.com/en-us/azure/azure-sql/database/high-availability-sla

Geo-Replication and Failover Groups



Source: https://docs.microsoft.com/en-us/azure/azure-sql/database/auto-failover-group-overview?tabs=azure-powershell

Azure Cosmos DB - Basics

Agenda

- NoSQL Concepts
- NoSQL Database Types
- Introduction to Azure Cosmos DB
- Cosmos DB APIs
- Resource Model
- Working with Core SQL API
- Connecting to Azure function

NoSQL Concepts

- Non-relational / non-SQL
- Schema-free / Flexible schema
- Distributed – Data is locally distributed amongst multiple nodes
- Replicated – Multiple copies of the data are created
- Horizontal scaling – Sharding across servers
- Provides
    - High Availability
    - High Throughput
    - Low Latency
- Maintaining consistency is challenging – provides eventual consistency

Use Cases

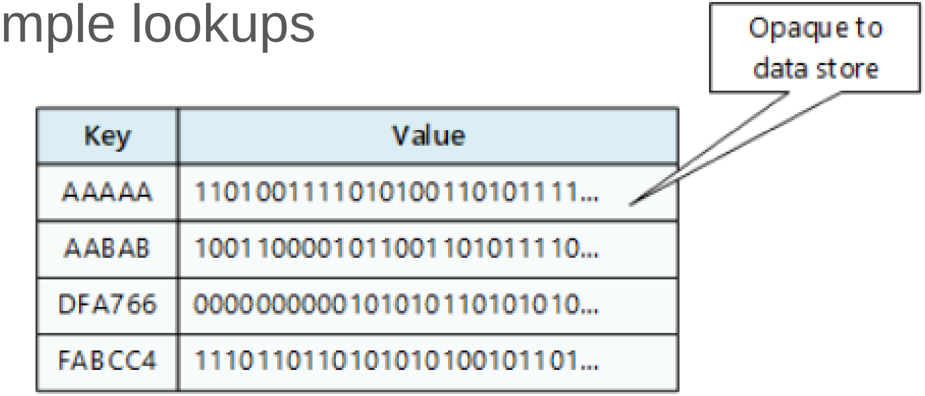
- Content management
- Personalization
- Build networks of entities
- Social media data

NoSQL Database Types

- Key-Value stores
- Document stores
- Wide-column / Column-family / Columnar stores
- Graph stores

Key-Value Store

- Data is associated with unique key
- Value is considered binary
- Query only on keys
- Optimized for simple lookups



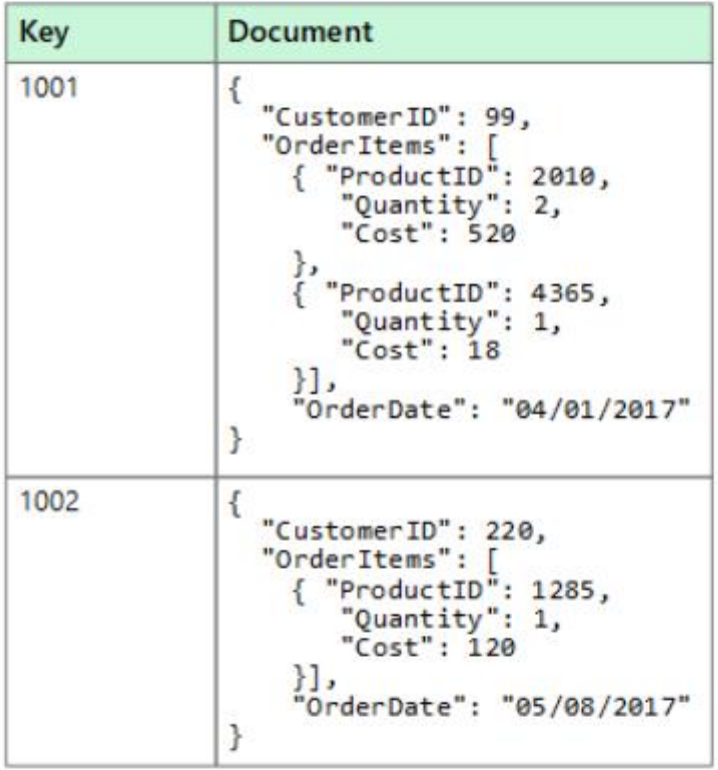
Opaque to data store

Key	Value
AAAAA	11010011110101001101011111...
AABAB	1001100001011001101011110...
DFA766	0000000000101010110101010...
FABCC4	1110110110101010100101101...

Source: *https://docs.microsoft.com/en-in/azure/architecture/data-guide/big-data/non-relational-data*

Document Store

- Data stored is known as Document
- Document is associated with unique key
- Document has flexible schema
- Document can be different formats - XML, JSON, YAML, BSON etc.
- Query on keys or on fields inside document



Key	Document

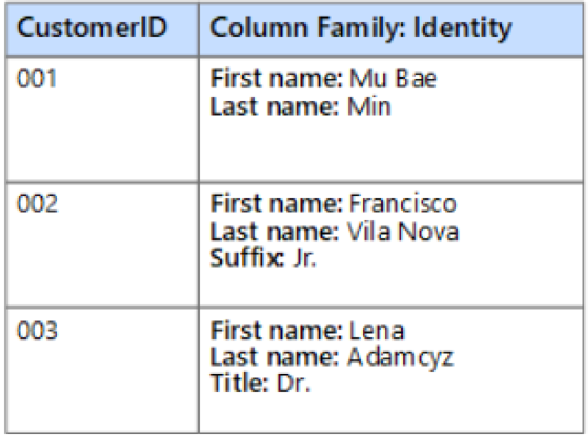
1001	{
	"CustomerID": 99,
	"OrderItems": [
	{ "ProductID": 2010,
	"Quantity": 2,
	"Cost": 520
	},
	{ "ProductID": 4365,
	"Quantity": 1,
	"Cost": 18
	}],
	"OrderDate": "04/01/2017"
	}

1002	{
	"CustomerID": 220,
	"OrderItems": [
	{ "ProductID": 1285,
	"Quantity": 1,
	"Cost": 120
	}],
	"OrderDate": "05/08/2017"
	}

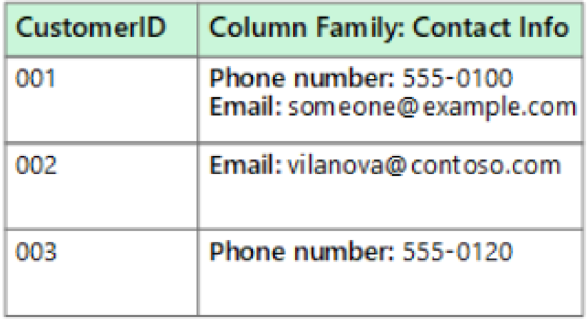
Source: https://docs.microsoft.com/en-in/azure/architecture/data-guide/big-data/non-relational-data

Columnar Store

- Uses column-oriented format and denormalized approach
- Looks very similar to relational structure
- Columns are grouped into column families, which are retrieved together
- Key is ' stored togetl



CustomerID	Column Family: Identity
001	First name: Mu Bae Last name: Min
002	First name: Francisco Last name: Vila Nova Suffix: Jr.
003	First name: Lena Last name: Adamcyz Title: Dr.

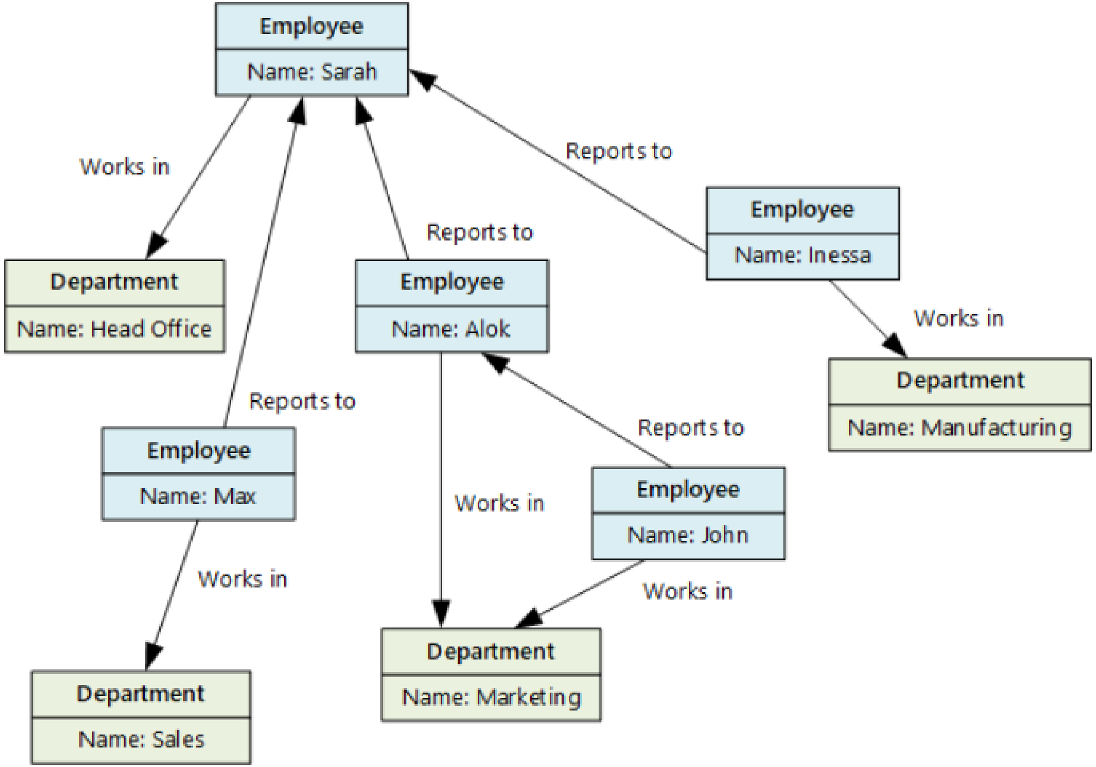


CustomerID	Column Family: Contact Info
001	Phone number: 555-0100 Email: someone@example.com
002	Email: vilanova@contoso.com
003	Phone number: 555-0120

Source: https://docs.microsoft.com/en-in/azure/architecture/data-guide/big-data/non-relational-data

Graph Store

- Based on nodes and edges
- Maintains relationship between entities
- Queries traverses nodes by using relationships



Source: https://docs.microsoft.com/en-in/azure/architecture/data-guide/big-data/non-relational-data

Azure Cosmos DB

- Fully managed NoSQL platform
- Supports typical NoSQL features
  - Distribution, replication, horizontal scaling, high throughput, low latency etc.
- Multi-API support
  - Create any type of NoSQL store
  - Supports five APIs
- Supports global distribution
  - Multiple read regions support with one write region
  - Multiple write regions support
- Automatic expiration of documents using Time-to-Live feature

Cosmos DB APIs

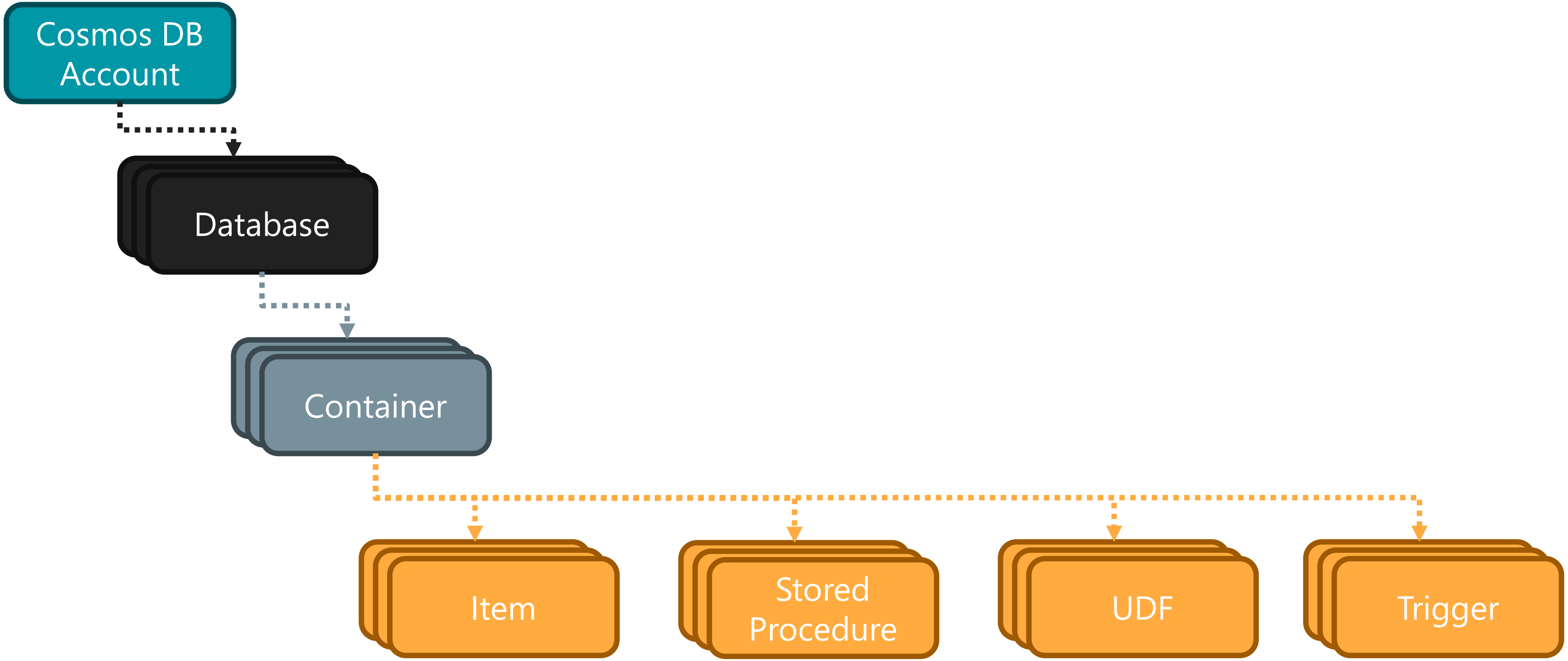
- **Table API**
  - Key-value store
  - Similar to Azure Table Storage
- **Core (SQL) API**
  - Document store
  - Earlier known as Azure Document DB
  - Developed by Microsoft
  - Store data in JSON format
  - Supports SQL-like syntax to query the data
  - Build stored procedures, user-defined functions and triggers in JavaScript

Cosmos DB APIs

- **MongoDB API**
    - Document store
    - MongoDB service in Azure, powered by Cosmos DB
    - Use existing MongoDB libraries, tools and applications
- **Cassandra API**
    - Columnar store
    - Cassandra service in Azure, powered by Cosmos DB
    - Use existing Cassandra libraries, tools and applications
- **Gremlin API**
    - Graph store
    - Based on open-source Apache Gremlin

greatlearning
Learning for Life

Resource Model



Resource Model

Azure Cosmos entity	SQL API	Cassandra API	MongoDB API	Gremlin API	Table API
Cosmos database	Database	Keyspace	Database	Database	NA
Cosmos container	Container	Table	Collection	Graph	Table
Cosmos item	Document	Row	Document	Node or edge	Item

Azure Cosmos DB - Advanced

Agenda

- Partitioning
- Throughput
- Global Distribution

Azure Cosmos DB

- Fully managed NoSQL platform
- Supports typical NoSQL features
    - Distribution, replication, horizontal scaling, high throughput, low latency etc.
- Multi-API support
    - Create any type of NoSQL store
    - Supports five APIs
- Supports global distribution
    - Multiple read regions support with one write region

Partitioning

- Every container has a user-defined Partition Key
- Data is stored in separate partitions, based on Partition Key
- All data with same partition key is stored in same partition
- Records can be retrieved efficiently using Partition Key
- One partition has max size of 20 GB

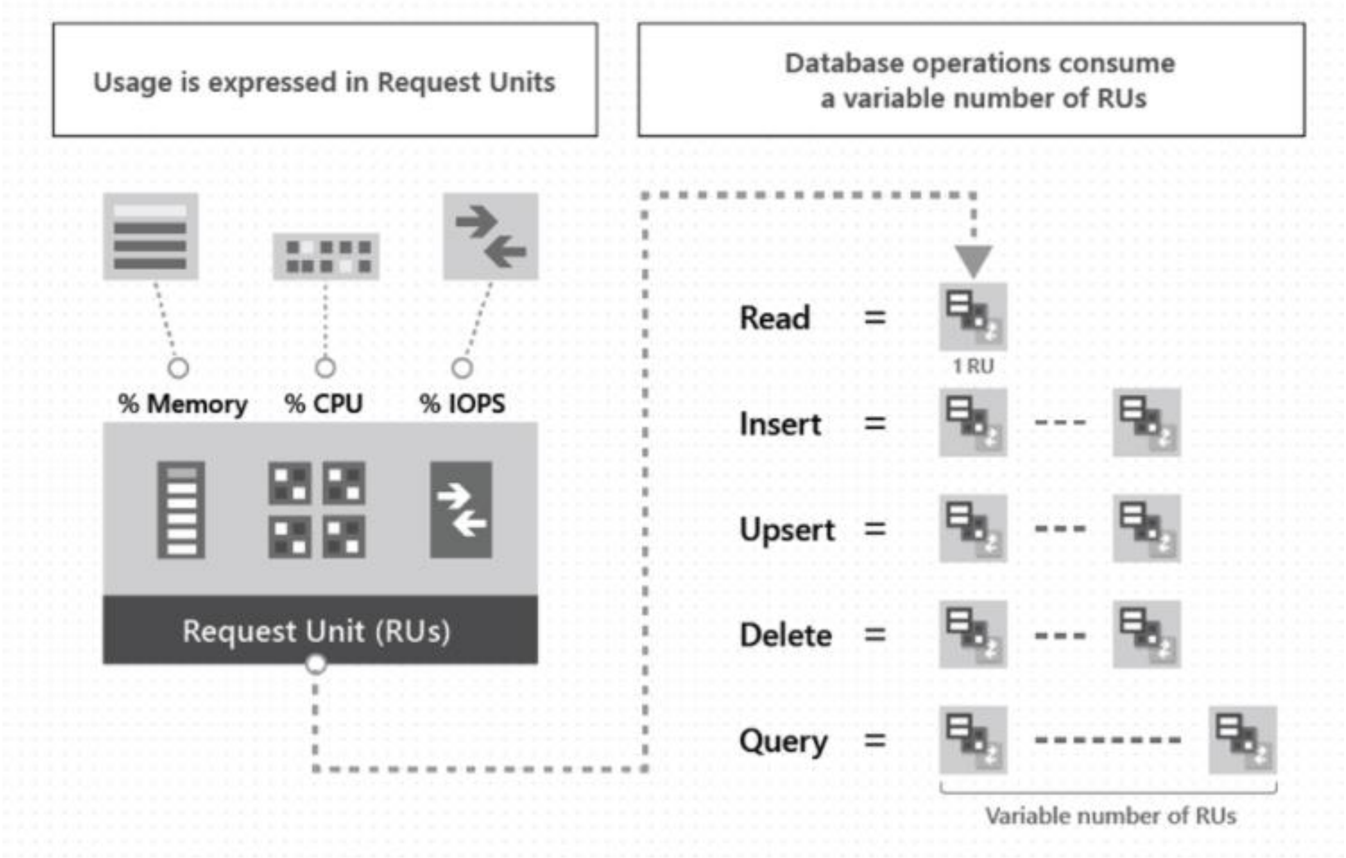
Partition Key – Design Considerations

- Partition key should have high cardinality – many different values
- Should be decided based on workload
  - Read-heavy vs write-heavy
- **Write-heavy**
  - Try to write data to as many partitions as possible
  - Choose a partition key that maximizes the parallelizability of writes
  - Avoid hot-partitions
- **Read-heavy**
  - Try to read from one partition as much as possible
  - Choose a partition key that optimizes the most common queries
  - Avoid fan-out

Throughput

- Number of requests that can be served in an instance of time
- Options
    - Provisioned throughput
    - Autoscaling throughput
    - Serverless throughput
- Throughput in Cosmos DB is defined using Request Units (RUs)
    - RUs is not equal to number of Requests
    - One read or write request can consume more than 1 RU
    - Each RU is a combination of CPU, memory and IOPS
    - Depending on type of query, RUs are consumed
- To handle more requests, provision more RUs

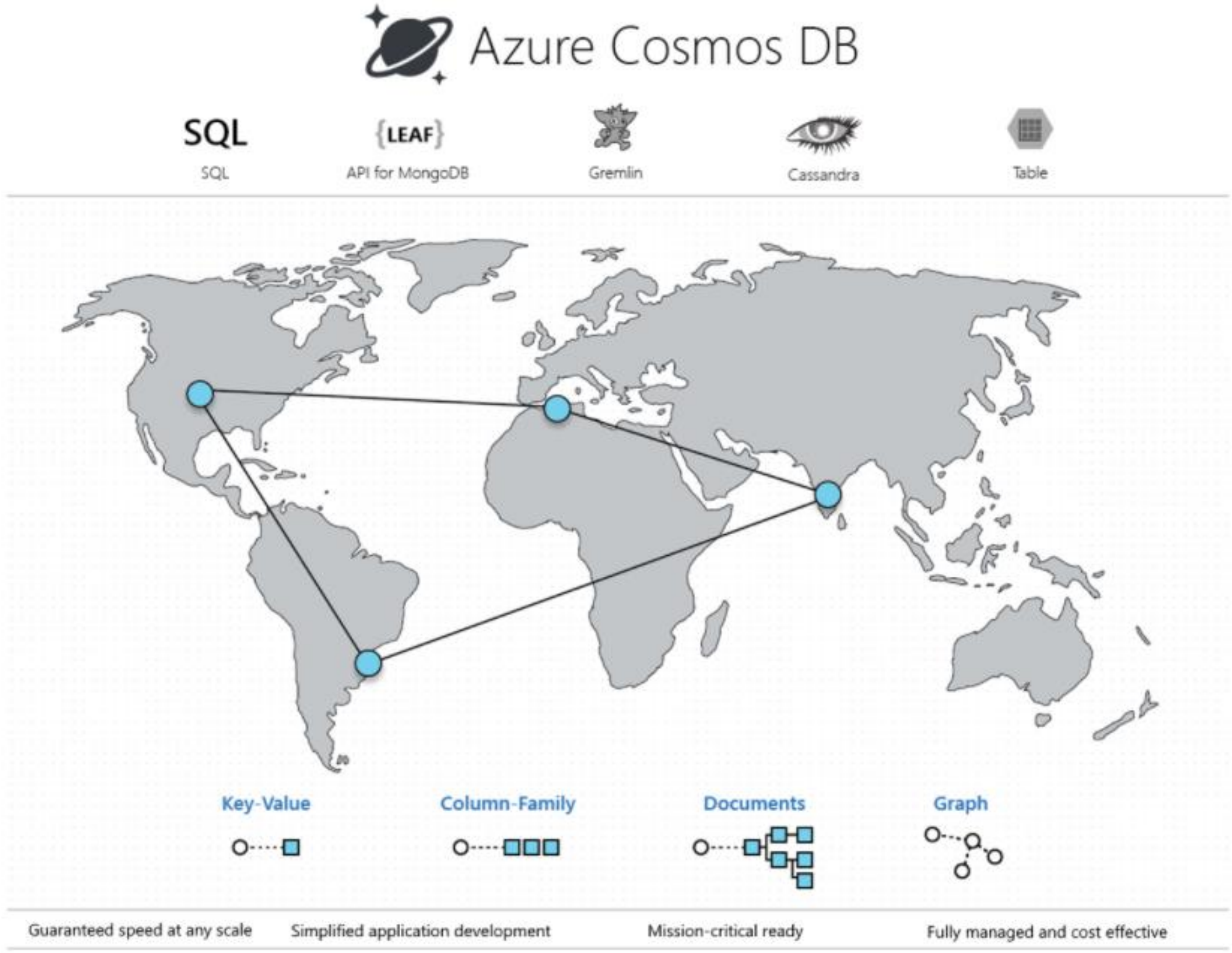
Throughput



Source: https://docs.microsoft.com/en-us/azure/cosmos-db/request-units

Factors affecting Request Units

- Item size
- Item indexing – by default all attributes in a document are indexed
- Attributes within a document – if indexing is enabled
- How data is distributed in partitions
- Type of query
  - Read or write query
  - Searching within a document consumes higher RUs
  - Querying on partition key consumes less RUs
  - Applying filters on attributes of document increases RUs etc.



Source: https://docs.microsoft.com/en-us/azure/cosmos-db/introduction

Global Distribution

- Supports single read-write and multiple read regions
- Supports multiple read-write regions
- Provides global low latency, high availability and high throughput
- Handles conflict management
  - LWW (Last-Write-Wins) algorithm – depends on timestamp
  - Custom-defined algorithm