

70-532 Exam Prep

Session 2 of 5

Design and Implement a Storage Data Strategy

Agenda

- Azure Storage Overview
- Storage Blobs
- Controlling Access to Storage Blobs and Containers
- Configuring Azure Storage Accounts
- Azure Files
- Azure Storage Tables Overview
- Cosmos DB
- Application Design Practices for Highly Available Applications
- Application Analytics
- Building High Performance Applications by Using ASP.NET
- Common Cloud Application Patterns
- Caching Application Data

Scenario

- You need a place to store the Word documents that are generated by the Contoso Events application. You decide store the generated Word documents in blobs. You also decide to create a protected container so that the Word documents are not accessed by anonymous users. Finally, you want to create the logic to generate SAS tokens for temporary access to one of the Word documents.

Azure Storage

- The Azure Storage services allow you to store records, files, and simple requests in a flexible, managed, and scalable solution
- Separating the storage of your data from your application allows more flexibility when planning and scaling different aspects of your cloud application scenario(s)
- Storage services are massively scalable which allow you to store large sets of data without being forced to plan partitioning and sharding of your data

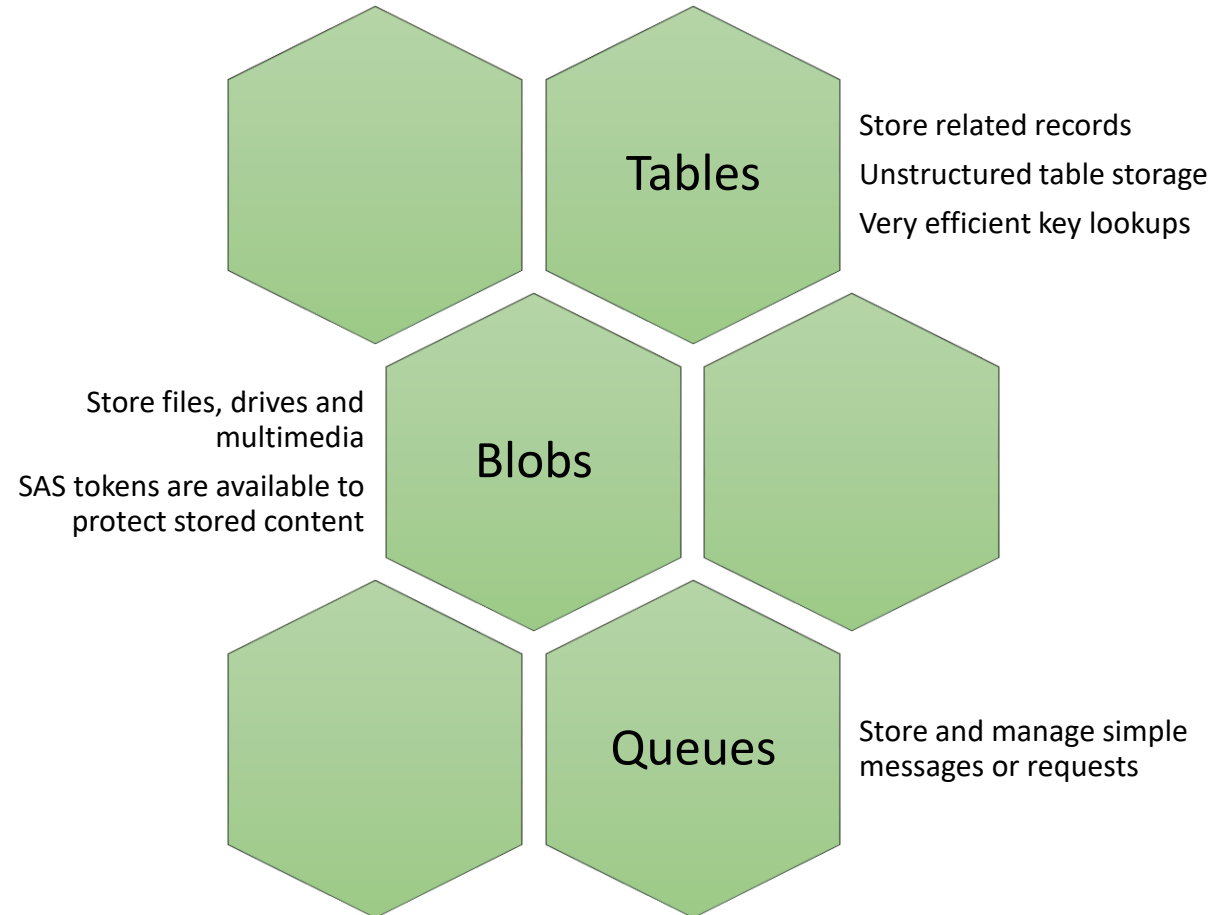
| Types of Storage

Tables

Blobs

Queues

Types of Storage (cont.)



Geo-Replication in Azure Storage

Locally Redundant Storage (LRS)

- Default option
- Data is replicated to three different nodes within the same data center

Zone Redundant Storage (ZRS)

- Data is replicated to three different data centers within the same region
- It is possible for data to be replicated across region boundaries if there are not enough data centers for storage within the region

Geo-Replication in Azure Storage (cont.)

Geo Redundant Storage (GRS)

- Data is replicated to three different nodes within the primary data center
- Data is replicated asynchronously to three nodes within a fixed redundant data center
- Data is “eventually consistent”

Read Access - Geo Redundant Storage (RA-GRS)

- Similar to GRS, data is replicated to a redundant data center
- Access to the secondary data source is read-only

Accessing Storage Data

Access to resources are authenticated by using a shared key for your storage account

You can configure blobs to allow anonymous access

You can generate shared access signatures to give a client controlled, temporary access to a storage resource

Storage accounts are given two keys that can be regenerated at any time

Allows you to rotate keys and regenerate keys on a scheduled basis without your application losing access to the resources

Storage Blobs

01

You can use Blob storage to store large amounts of unstructured text or binary data.

02

You can use Blob storage to store files such as:

- Virtual hard disk drives
- Videos
- Images
- Log text files

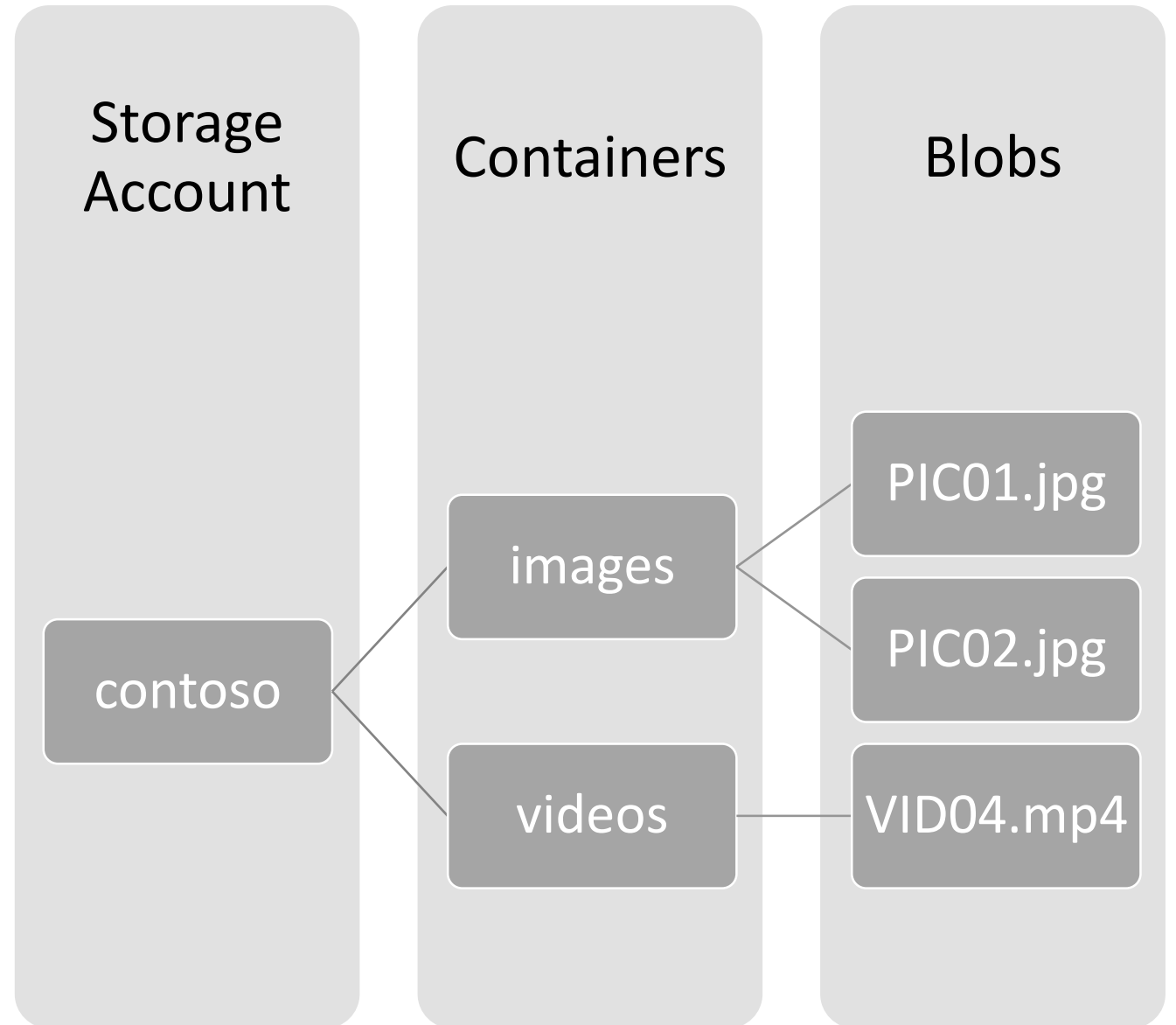
03

You can use Blob storage to group blobs into logical, hierarchical containers

04

You can secure blobs and make them available for anonymous access

Storage Blobs (cont.)



Blob Types

- There are two primary types of blobs:

Page blobs

Block
blobs

Page Blobs

Page Blobs
are a
collection of
512-byte
pages

Optimized for Random Access and Frequent Updates

Used as persistent disks for VMs in Azure.

Highly performant, durable and reliable

Can grow and shrink in size by adding or removing pages

Modifications
of a page
blob can
overwrite
one or more
pages.

Changes are in-place and immediately committed.

A page blob
can be no
larger than 1
TB.

Block Blobs

Block Blobs
are
comprised
of a series of
blocks

Blocks can be uploaded in parallel sets to speed up the ingress of a large file.

You can use an MD5 hash to verify that each block is uploaded successfully and retry failed blocks.

You can also track the progress of block upload.

While uploading, the blob is considered uncommitted.

When all blocks are uploaded, you can determine the order and then “commit” the blob.

Block Blobs
are
optimized
for
multimedia
streaming
scenarios.

Block Blobs
can be no
larger than
200 GB.

REST API for Storage Blobs

- Blobs have the simplest of all of the Storage REST endpoints
 - GET BLOB
 - [https://\[account\].blob.core.windows.net/\[container\]/\[blob\]](https://[account].blob.core.windows.net/[container]/[blob])
 - POST, PUT, and DELETE is available on the same endpoint
- You can access containers for operations by using the restype query string parameter
 - [https://\[account\].blob.core.windows.net/\[container\]?restype=container](https://[account].blob.core.windows.net/[container]?restype=container)
- You can append Shared Access Signature (SAS) tokens to the end of a URL to access protected blobs

Content Delivery Network

You can create CDN endpoints for an existing storage account

Storage content is cached to edge servers that is closer to your users

You can also create CDN endpoints for a cloud service

CDN content can be configured to be served from a custom domain

Cross-Origin Resource Sharing

Cross-Origin
Resource Sharing
(CORS) allows
client code to
make requests
across domains.

Browsers and security policies typically restrict these type of requests.

CORS is an
extension of the
HTTP
request/response
spec.

Composed of two requests (preflight and actual).

CORS is enabled
using rules at the
Storage Account
level.

CORS is opt-in.

The rules serve as a policy list of
allowed origins, methods and headers.

Scenario

- Even though event registrations could be stored in SQL, you have a unique need. Each event requires a different registration form that can be changed at any time. Essentially, registrations could be of any schema. A relational database such as SQL requires a well-defined schema. Because of your business requirement, you require a database that can store items with flexible structures (or schemas). To facilitate this you have elected to use Azure Table Storage for your event registrations.

Storage Tables

Table storage allows you to store flexible datasets that are not constrained by a schema or a model

- Client applications can dictate the model of the data stored
- Complex objects of different schema(s) can be stored in the same table

Built for massive scale (very large datasets)

Data is partitioned by using a partition key to support load balancing across different nodes

Data is indexed by a combination of the partition key and a row key for very fast lookup

NoSQL Data in Storage Tables

Table storage is a NoSQL database

- Implemented as a key-value store

Tables contain entities that are partitioned across multiple nodes by using the partition key

Entities have an index, which is a combination of the partition and row keys

Properties of a particular entity are implemented as a collection of key-value pairs

- Key is the name of the property
- Value is the value for the property

NoSQL Data in Storage Tables (cont.)

PartitionKey: Student
RowKey: 145A

FirstName

- Chad

LastName

- Drayton

Age

- 8

Grade

- 3

PartitionKey: Student
RowKey: 287C

FirstName

- Joann

LastName

- Chambers

Grade

- 3

PartitionKey: Teacher
RowKey: 945FT

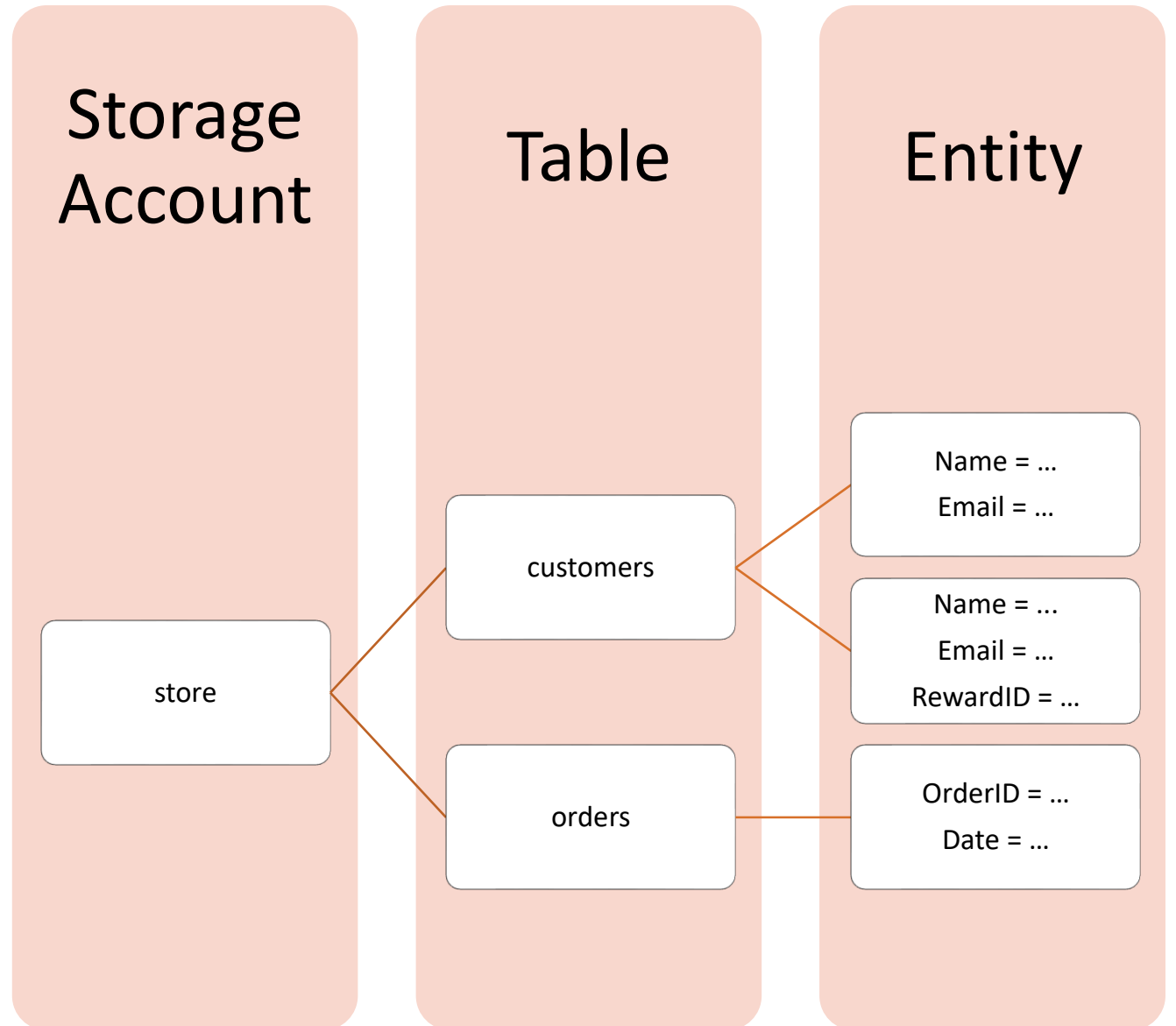
LastName

- Summers

Grade

- 5

NoSQL Data in Storage Tables (cont.)



Demo: Implementing Azure Storage Tables

In this demo, you will
see how to:

- Create a table by using the Azure Storage client libraries for .NET
- Add an entity to the table
- Query the table's entities by the row key

Common Transactions

Tables support common transactions for the specified entities:

- Create
- Read
- Update
- Delete

Entities behave like Dictionaries when updated

- Any key-value pairs with a new key or added to the collection of properties for an entity
- Any key-value pairs that re-use an existing key replaces the corresponding value

OData Queries

- Storage tables can be queried by using an HTTP endpoint and the OData protocol
 - OData is a REST protocol that standardizes a method for querying a data API
 - Because it is built on top of the REST protocol, the standard Create, Read, Update, Delete (CRUD) operations are still available

OData Queries (cont.)

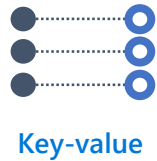
- Table storage OData examples:
 - Base URL:
 - `https://[account].table.core.windows.net`
 - Get an entity by a partition and row key:
 - `[GET] /[table]([PartitionKey],[RowKey])`
 - Query a table for entities that match an expression:
 - `[GET] /[table]()?$filter=[query expression]`
 - Delete an entity:
 - `[DELETE] /[table]([PartitionKey],[RowKey])`
 - Insert or replace an entity:
 - `[PUT] /[table]([PartitionKey],[RowKey])`

Azure Cosmos DB

A globally distributed, massively scalable, multi-model database service



MongoDB API



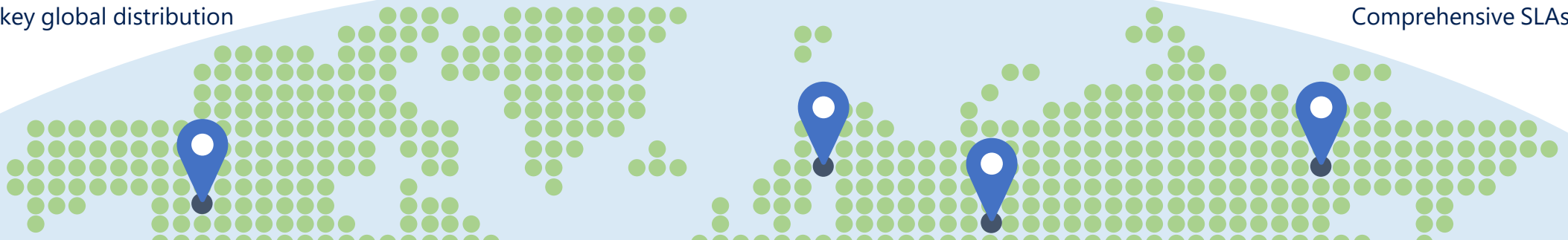
Elastic scale out
of storage & throughput

Guaranteed low latency at the 99th percentile

Five well-defined consistency models

Turnkey global distribution

Comprehensive SLAs



Azure Cosmos DB



Azure Cosmos DB

Key-Value



Column-family



Documents



Graph



Global distribution

Elastic scale out

Guaranteed low latency

Five consistency models

Comprehensive SLAs



- Globally distributed
- Multi-model database service

Key Capabilities – Cosmos DB

Turnkey Global Distribution

- distribute your data to any number of Azure regions, with the click of a button

Multiple data models and popular APIs for accessing and querying data

- SQL API, Mongo DB API, Cassandra API, Graph (Gremlin) API, Table API

Elastically and independently scale throughput and storage on demand and worldwide

- scale database throughput at a per-second granularity

Build highly responsive and mission-critical applications

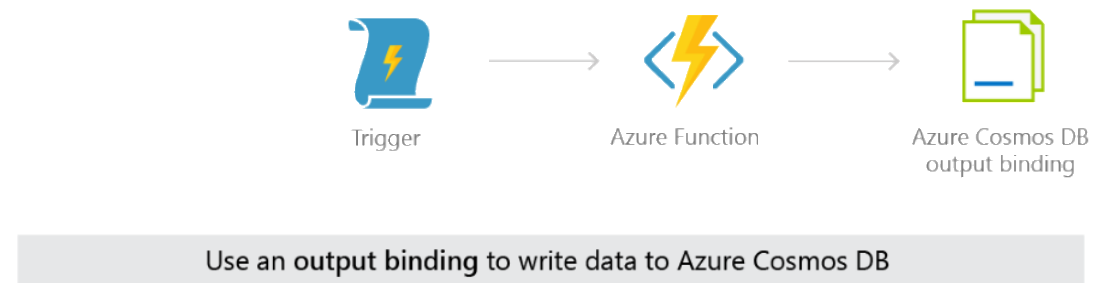
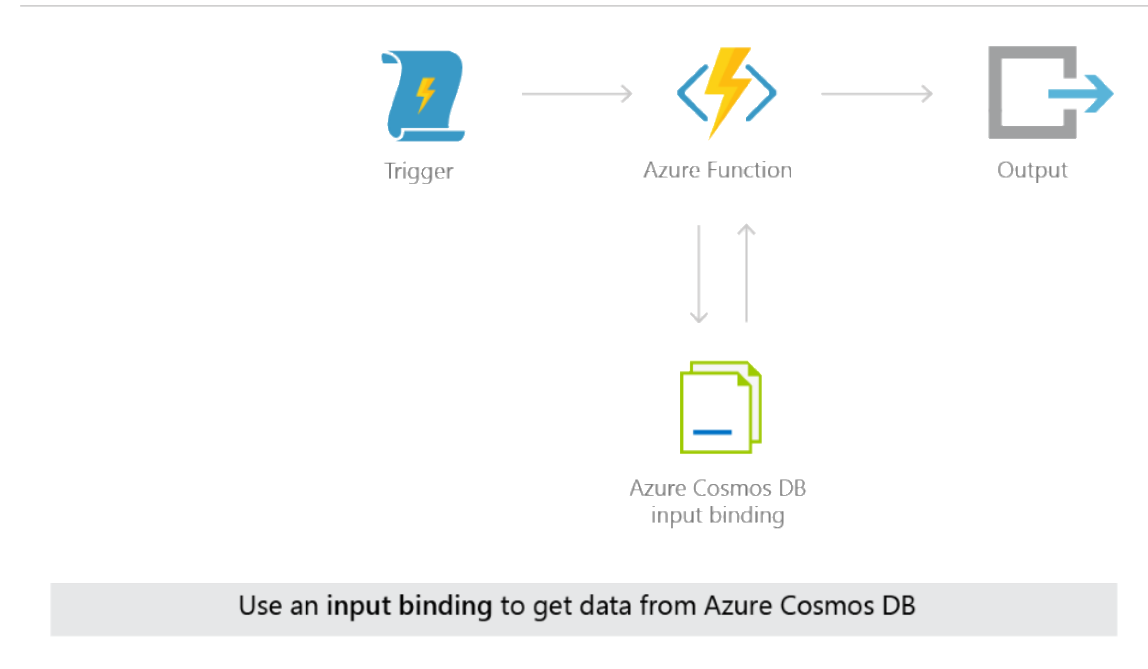
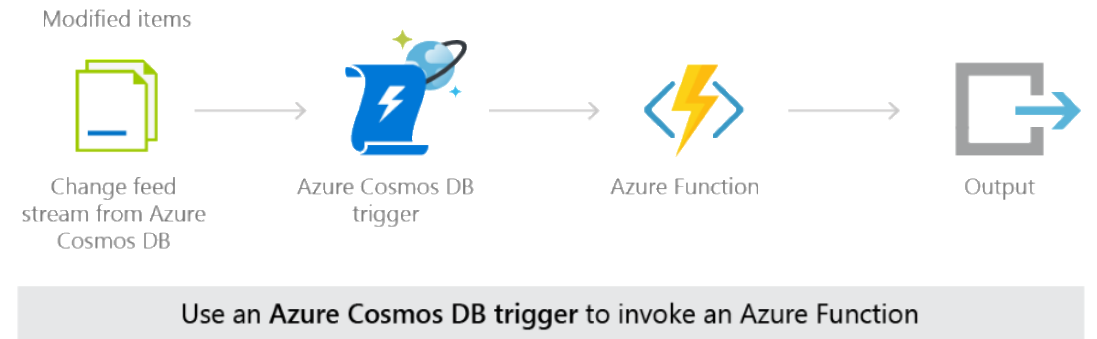
- guarantees end-to-end low latency at the 99th percentile to its customers

Key Capabilities (contd.)

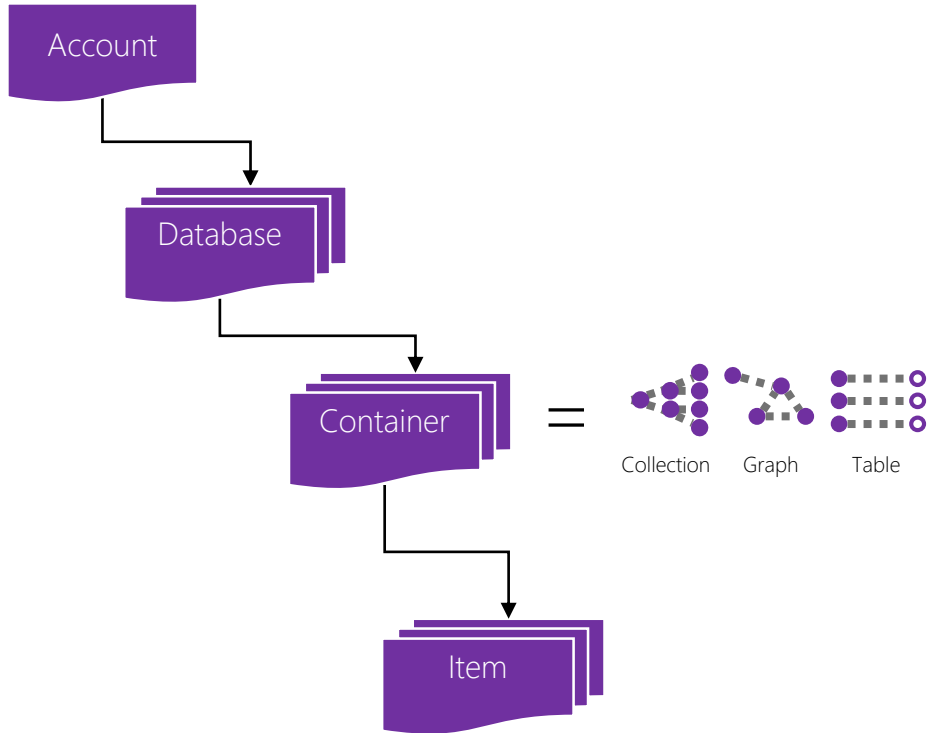
Ensure "always on" availability	99.99% availability SLA for all single region database accounts, and all 99.999% read availability on all multi-region database accounts
Write globally distributed applications, the right way	5 well defined consistency models
Money back guarantees	financially backed, comprehensive service level agreements for availability, latency, throughput, and consistency.
No database schema/index management	fully schema-agnostic – it automatically indexes all the data it ingests
Low cost of ownership	Five to ten times more cost effective than a non-managed solution or an on-prem NoSQL solution.

Serverless database computing using Azure Functions with Cosmos DB

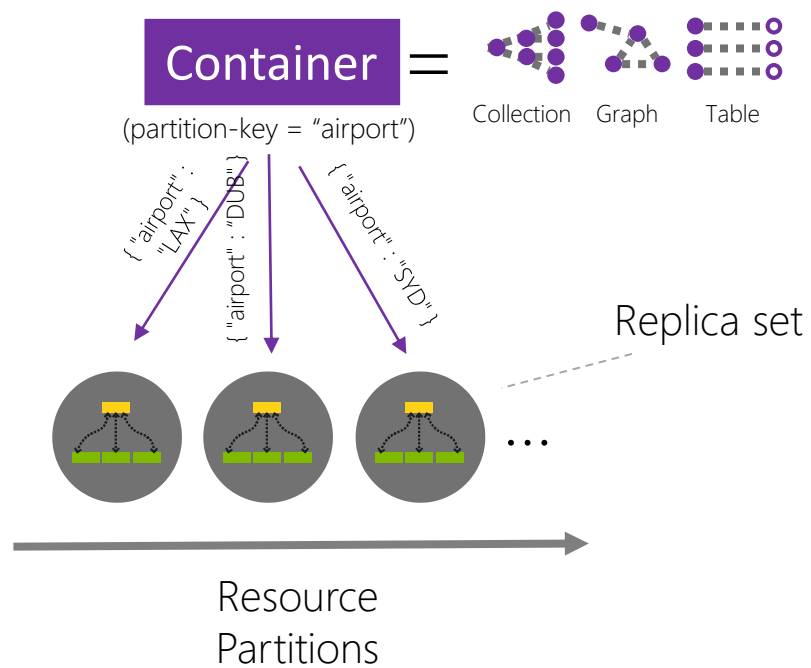
- Create an event-driven Azure Cosmos DB trigger in an Azure Function
- bind an Azure Function to an Azure Cosmos DB collection using an input binding.
- Bind a function to an Azure Cosmos DB collection using an output binding
- Use Case – IoT, Gaming, Retail



Resource Model



- Resources identified by their logical and stable URI
- Hierarchical overlay over horizontally partitioned entities; spanning machines, clusters and regions
- Extensible custom projections based on specific type of API interface
- Stateless interaction (HTTP and TCP)



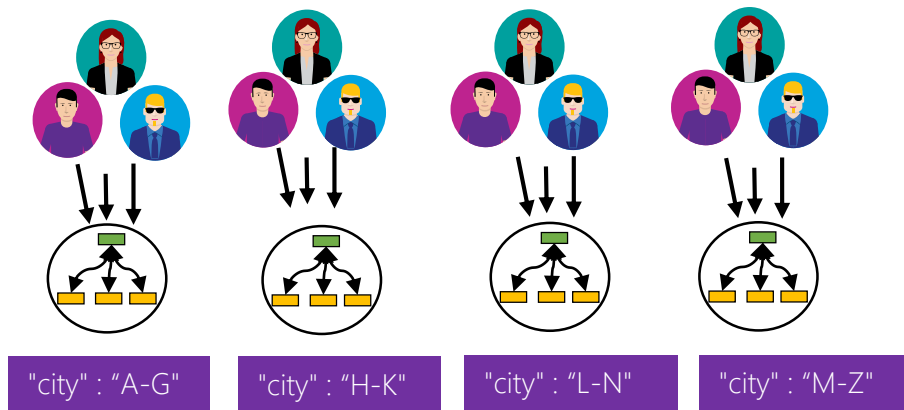
Horizontal Partitioning

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"



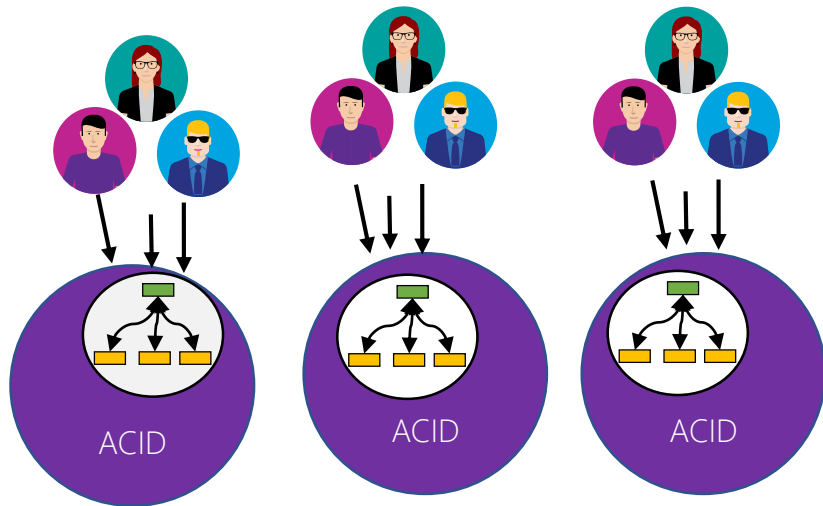
Best Practices: Partitioning

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



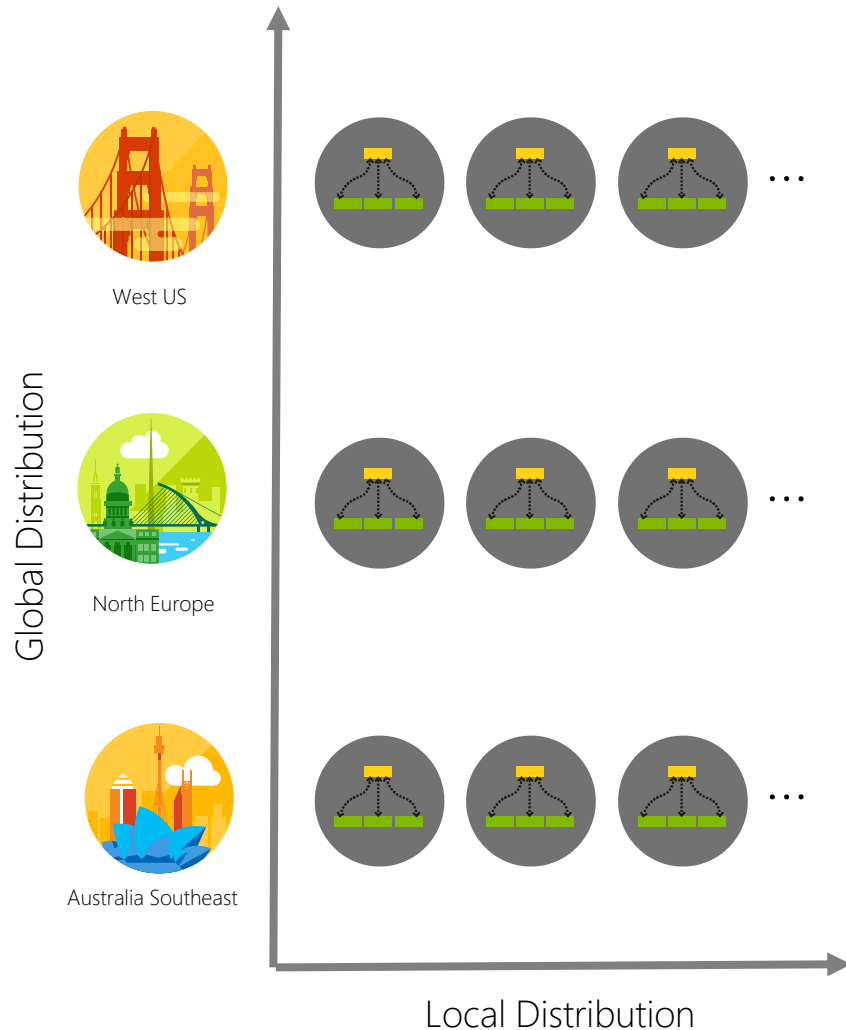
Best Practices: Partitioning

The service handles routing query requests to the right partition using the partition key

Partition key should be represented in the bulk of queries for read heavy scenarios to avoid excessive fan-out.

Partition key is the boundary for cross item transactions. Select a partition key which can be a transaction scope.

An ideal partition key enables you to use efficient queries and has sufficient cardinality to ensure solution is scalable

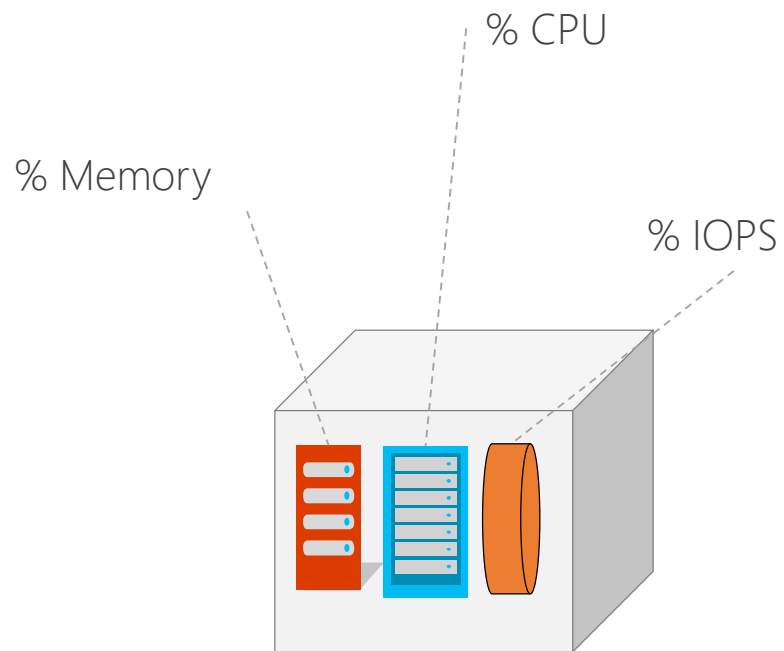


Global Distribution

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



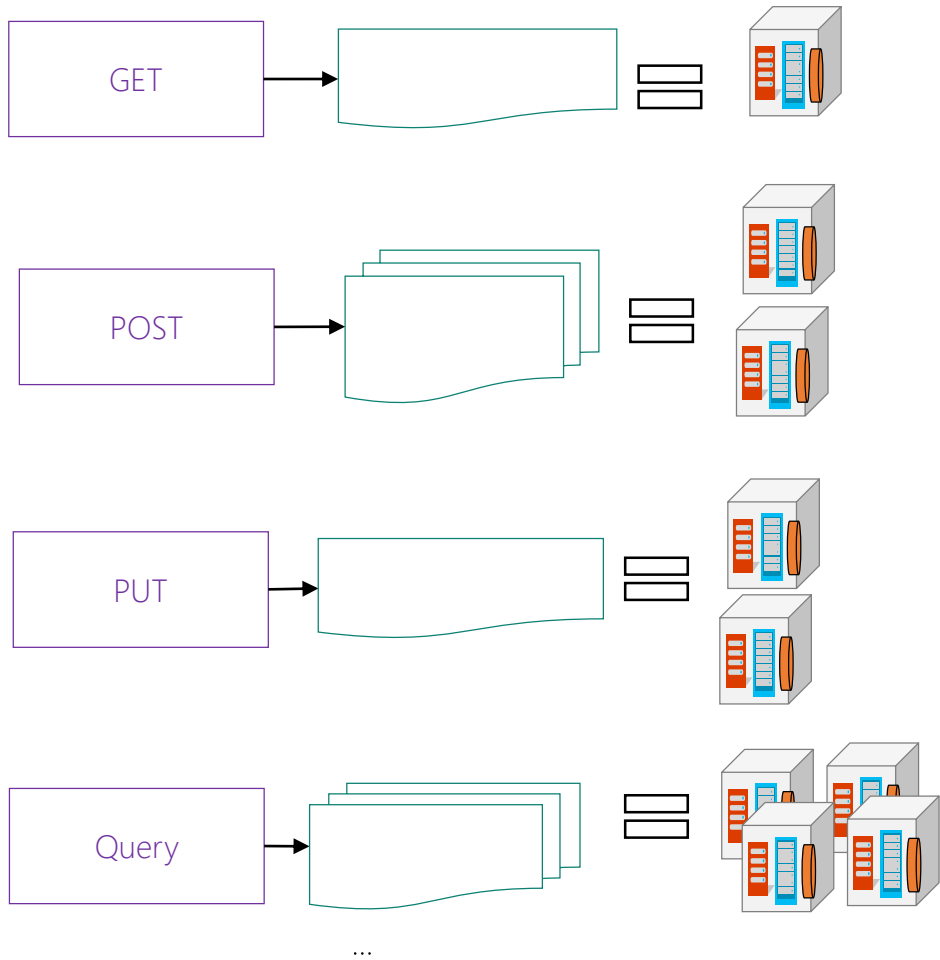
Request Units

Request Units (RU) is a rate-based currency

Abstracts physical resources for performing requests

Key to multi-tenancy, SLAs, and COGS efficiency

Foreground and background activities



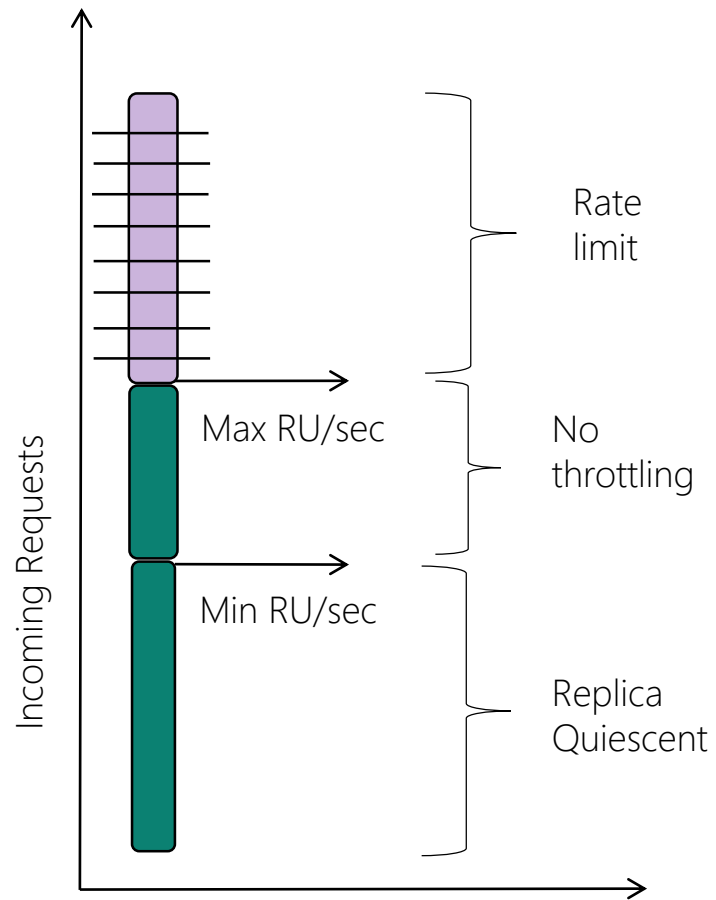
Request Units

Normalized across various access methods

1 RU = 1 read of 1 KB document

Each request consumes fixed RUs

Applies to reads, writes, queries, and stored procedure execution



Request Units

Provisioned in terms of RU/sec and RU/min granularities

Rate limiting based on amount of throughput provisioned

Can be increased or decreased instantaneously

Metered Hourly

Background processes like TTL expiration, index transformations scheduled when quiescent

Azure Cosmos DB Table API

Azure Cosmos DB provides the Table API for applications that are written for Azure Table storage and that need premium capabilities like:

- Turnkey global distribution.
- Dedicated throughput worldwide.
- Single-digit millisecond latencies at the 99th percentile.
- Guaranteed high availability.
- Automatic secondary indexing.



Applications written for Azure Table storage can migrate to Azure Cosmos DB by using the Table API with no code changes and take advantage of premium capabilities. The Table API has SDK available for .NET, Java, Node.js & Python.

Application Design Practices for Highly Available Applications

Partitioning Workloads

When designing web applications, split your business processes into Partitioning Workloads

Partitioning Workloads:

- Can be handled in modular websites, cloud services, or virtual machines
- Provides the ability to scale the different components of your application in isolation

Partitioning Workloads (cont.)

- Example Photo Sharing Application

Virtual Machine

Web Front-End

SignalR Hub

Image Processor

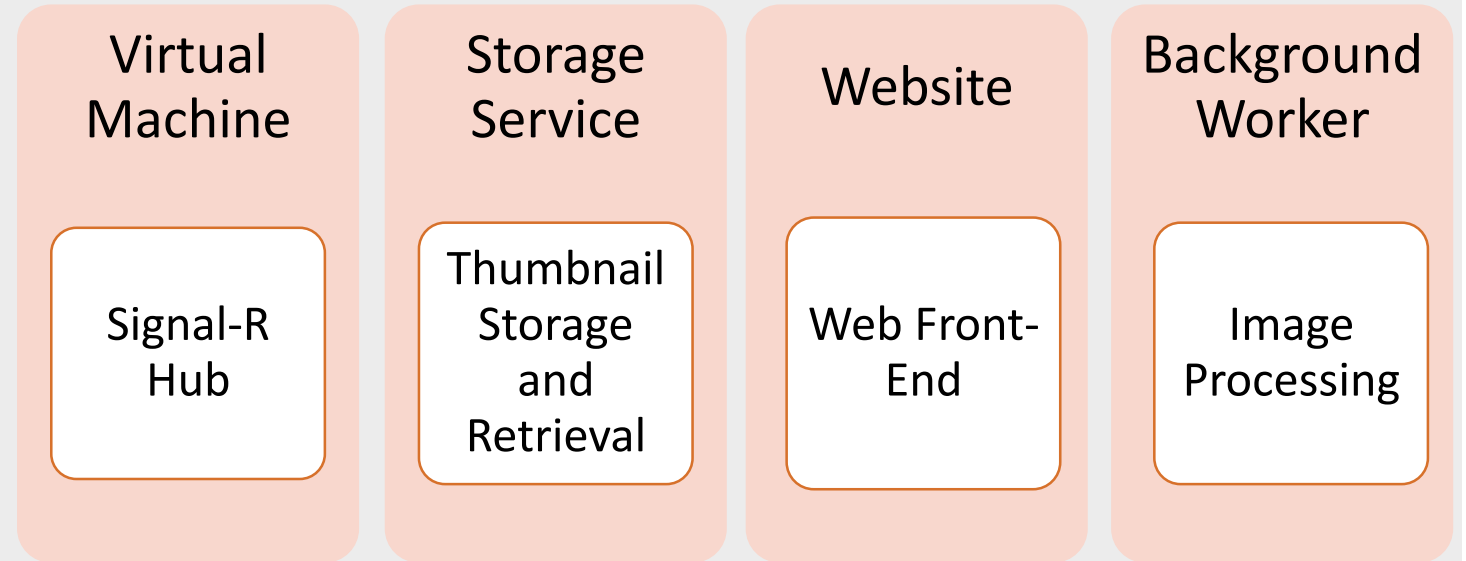
Thumbnail Storage

Partitioning Workloads (cont.)

- Example Photo Sharing Application
 - The Web Front-End shows thumbnails of images users have uploaded today
 - The application code takes an image that a user uploads, processes it into a thumbnail and then stores the thumbnail.
 - The SignalR hub notifies the client web browser that the image is finished processing

Partitioning Workloads (cont.)

- Example Photo Sharing Application



- Each component of the application can be scaled in isolation to meet its specific demand

| Load Balancing

Provide the same service from multiple instances and use a load balancer to distribute requests across all of the instances

Considerations for selecting a load balancing strategy:

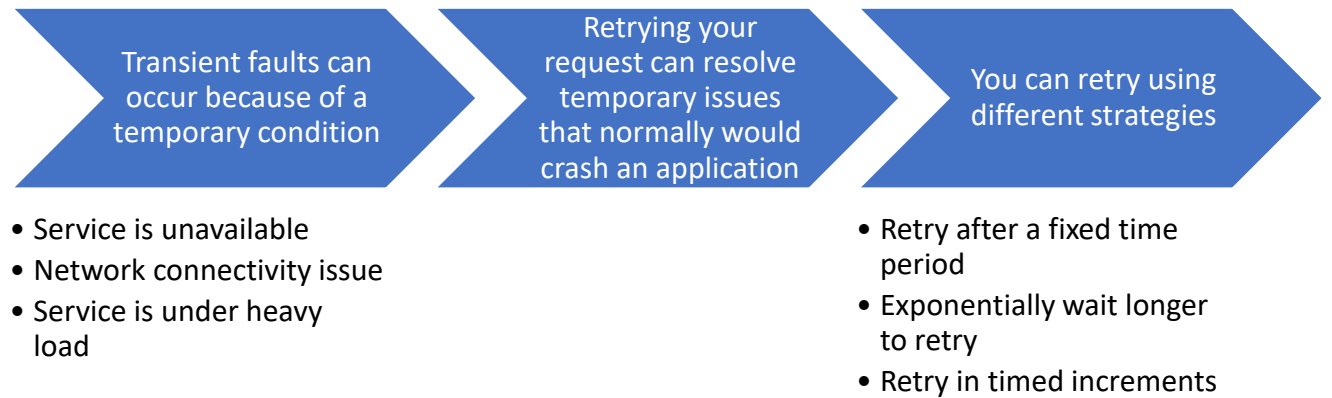
- Hardware or software load balancers
- Load balancing algorithms (round robin)
- Load balancer stickiness

Load balancing becomes critical even if you have a single service instance as it offers the capability to scale seamlessly

Load Balancing and Geographic Resiliency

- Load balancing can be combined with geographic redundancy to help achieve high availability
 - You can use a load balancer to direct client requests to their closest data center
 - Traffic Manager is often used for this task
 - A load balancer can be used to implement a failover scenario
 - When a data center or compute instance is down, clients can be directed to the next desirable instance that is available
- Designing a load balancing strategy along with distributing your application across data centers is key to high availability across the globe

Transient Fault Handling



Transient Fault Handling (cont.)

The Transient Fault Handling application block is a part of the Enterprise Library

The Enterprise library contains a lot of code that is necessary to implement the pattern and the retry strategies

Retry strategies are prebuilt for common scenarios including accessing Azure services

You can build custom strategies and extend the library for the unique needs of your application

Queues

A modular web application can behave like a monolithic application if each component relies on a direct two-way communication with a persistent connection

Persistent queue messages allow your application to handle requests if one of your application components fail or is temporary unavailable

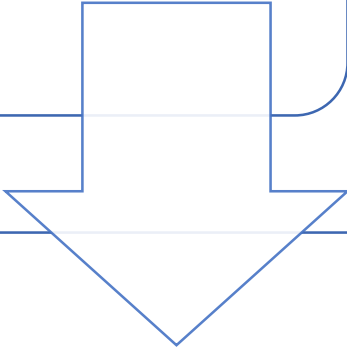
An external queue allows your application to audit requests or measure your load without adding any overhead to the code of your primary application



| Queues (cont.)

Queues can be used to communicate in a disconnected manner between the different components of your application

- If an instance of your application module fails, another instance can process the queue messages
- Messages that fail to be processed can either be reinserted into the queue or not marked as complete by the client module
- Messages can be measured or audited in isolation from your application



Queues become very important when dealing with background processing (Worker roles and WebJobs)

Application Insights

Application Insights is a analytics and monitoring service available for your applications

- View exception stack traces
- Monitor CPU and resource usage
- Periodically test URLs from worldwide data centers
- Monitor usage of your application and most popular requests



It can be used with .NET or Java



Applications do not specifically need to be hosted in Azure

Application Insights (cont.)

- The application insights blade provides a rich dashboard of data (tiles) and supports “drilling down” into more specific metrics.



Demo: Monitoring a Web Application

In this demo, you will see how to:

- Create a new ASP.NET Web Application project with Application Insights referenced
- Use the Application Insights service and dashboard to monitor the metadata about your application's requests

Azure Redis Cache

- Based on the open-source Redis platform
 - Multiple tiers are available that offer different numbers of nodes
 - Basic – sizes upto 53 GB
 - Standard
 - Premium – sizes upto 530 GB
 - Supports transactions
 - Supports message aggregation using a publish subscribe model
 - Considered a key-value store where the keys can be simple or complex values
 - Massive Redis ecosystem already exists with many different clients

Azure Redis Cache Premium Tier

Better performance

Redis Data Persistence

Redis cluster

Enhanced Security and Isolation

Import/Export

Reboot

Schedule Updates

Geo-Replication

What is Azure Search?

- Azure Search is a search-as-a-service cloud solution that gives developers APIs and tools for adding a rich search experience over your content in web, mobile, and enterprise applications.
- Features
 - Full text search and text analysis
 - Data Integration
 - Linguistic Analysis
 - Geo-Search
 - User Experience Features
 - Relevance
 - Monitoring and Reporting
- Supported Document Formats – pdf, office files, HTML, XML, ZIP, EML, RTF, Plain text, JSON, CSV

Synonyms in Azure Search



Synonyms in search engines associate equivalent terms that implicitly expand the scope of a query, without the user having to actually provide the term.

For example, given the term "dog" and synonym associations of "canine" and "puppy", any documents containing "dog

synonym expansion is done at query time

2 step process

Add a synonym map to your search service through the APIs below.

Configure a searchable field to use the synonym map in the index definition.



Student Reference

<https://tinyurl.com/532S02STRG>