Cosmos Db

Azure Distributed Document Database

About the Presenter

- Clarke D. Bowers
- Clarke Bowers Consulting, LLC
- http://www.cbsoftwareengineering.com/
- mailto: clarke@cbsoftwareengineering.com
- 35 years of industry experience
- He has architected and developed embedded systems, desktop applications, enterprise data warehouses, web sites and web services; cloud bases and locally hosted solutions
- He holds six patents

You can find this presentation and all the examples at: https://1drv.ms/f/s!Ar3pO7_GhJY9hPoKTTVe-Hn-gBpuow

Agenda

- Main focus: Document Databases and No SQL

- Define Cosmos DB
- Compare to SQL Server
- Investigate MongoDb features & API
- Learn to query & store
- Compare document and relational models
- Learn about sharding & geo-distribution
- See demo of IoT application
- Review Azure function from demo

COSMOS DB Application: Tidal Probe

- Mount Raspberry Pi, GPS, sensors and batteries on buoy
- Measure location, linear acceleration, angular acceleration, temperature and barometric pressure
- Transmit to Azure & store in Cosmos DB

Application for harbor pilots. It tracks the cargo ships location and view latest buoy data with PowerBI for nearby buoys. Displays on map the microclimate marine conditions.

What is Cosmos DB?

- I have not validate these claims and features
  - Availability and latency guaranteed
  - Automatic indexing tuning
  - Encryption
  - Backups
  - First-Class Citizen of Azure
    - Subscription maybe based on it

- Formerly known as DocumentDb
- No Schema (schema-less)
- Sharded
- Globally Distributed
- Document Database
- API (MongoDb API in my examples)
- No SQL (SQL-less) (mostly…)
- Server-less
- Replicated

Storage Hierarchy

SQL Server

- Windows Server or Cluster
- SQL Server Instance
- Database
- Schema (namespace)
- Table (Indexes, Keys, Relationships)
- Tuple (row)

Cosmos DB as MongoDb

- Azure Subscription (billing)
- DB account (like a server but not singular in location or compute power) (mostly defines storage)
- Database (mostly a namespace)
- Collection (a cross between a table and a star schema, also the unit of computing) (a.ka. Container)
- Document (more than a tuple)

Container

- When you create the Cosmos DB account, you must select the contents of the containers.
- All container hold the same type of data repository for the Cosmos DB account

- MongoDB for document data
  - This is what I will demo in this presentation
- Gremlin for graph data
- Azure Table
- Cassandra
- Core SQL (preview)

MongoDb API

- Cosmos DB is wire compatible with the MongoDb protocol
- MongoDb API can be found here (http://api.mongodb.com/)
- Supports many languages including: C++, C#, JavaScript and Python

- Cosmos DB supports other APIs
  - Choice is made at instance creation to match containers
- Cosmos DB has it's own .NET API
  - Version 3.0 preview
    - Compatible with .NET Core
    - Does not work yet
  - Version 2.0
    - Requires full framework

Python & MongoDB

- Cosmos DB supports the wire protocol for MongoDb API
- Pymongo is a package for Python that generate the wire protocol for MongoDB API
  - Install with: python -m pip install pymongo –user
  - Latest version 3.7.2
- Connect to a Cosmos DB account
- Access the "wave" database
- Access the "Sensor" document collection
- Count the documents
- Select the most recent three documents
- Equivalent SQL statement
  - SELECT TOP 3 * FROM [wave].[Sensor] ORDER BY [When] DESC

```
import pymongo;
import pprint;

#connect to the server
mongoClient = pymongo.MongoClient('mongodb://...');

#grab the wave database
db = mongoClient.wave;

#grab  the sensor document collection
sensorDocs = db.Sensor;
print(sensorDocs.count());

 #find the most recent three documents and display them
cursor = sensorDocs.find().sort([('When', -1)]).limit(3)
for doc in cursor : pprint.pprint(doc);
```

Document Creation

- Check for testdb
  - Access creates the DB when a document is added to a collection
  - Access creates the collection when a document is added
- Add a document
- All documents receive an ID field automatically
  - The name is _id
- Warning: default RU/s is 1,000
  - Request Units
  - This cost $60 per month for one collection!

```
#check if the DB exits and access it
dblist = mongoClient.list_database_names()
if "testdb" in dblist:
   print("The testdb database exists.")
testDb = mongoClient["testdb"];


testCollection = testDb["testCollection"]
print(testCollection.count());


testDocument = { "name": "John", "address": "Highway 37" }
id = testCollection.insert_one(testDocument)
print(id)
```

Criterion

- Find method on the collection object first parameter is the criteria
- Criteria is an JSON object
- Case sensitive equality
- Equivalent SQL statement
  - SELECT TOP 1 * FROM [wave].[Sensor] WHERE MachineName = “MINIWINPC”

```
mongoClient = pymongo.MongoClient('mongodb://...);

db = mongoClient.wave;

sensorDocs = db.Sensor;


doc = sensorDocs.find(

{ "MachineName" : "MINWINPC" }

).limit(1)[0]

pprint.pprint(doc);
```

Embedded Document Query

- Return document from field nested to any depth
- Dot canonical notation
- Operators are abbreviations and not symbols
- find_one returns the first matching document
- WITH forces AS (
  SELECT [ForceId] FROM [wave].[Sensor] WHERE Mean > 0.1)
  SELECT TOP 1 * FROM [wave].[Sensor]
  WHERE ForceId In (SELECT ForceId FROM force)

```
#find the one document with a
# force mean greater than 0.1
doc = sensorDocs.find_one(
   { "Force.Mean": { "$gt": 0.1 } })
pprint.pprint(doc);
```

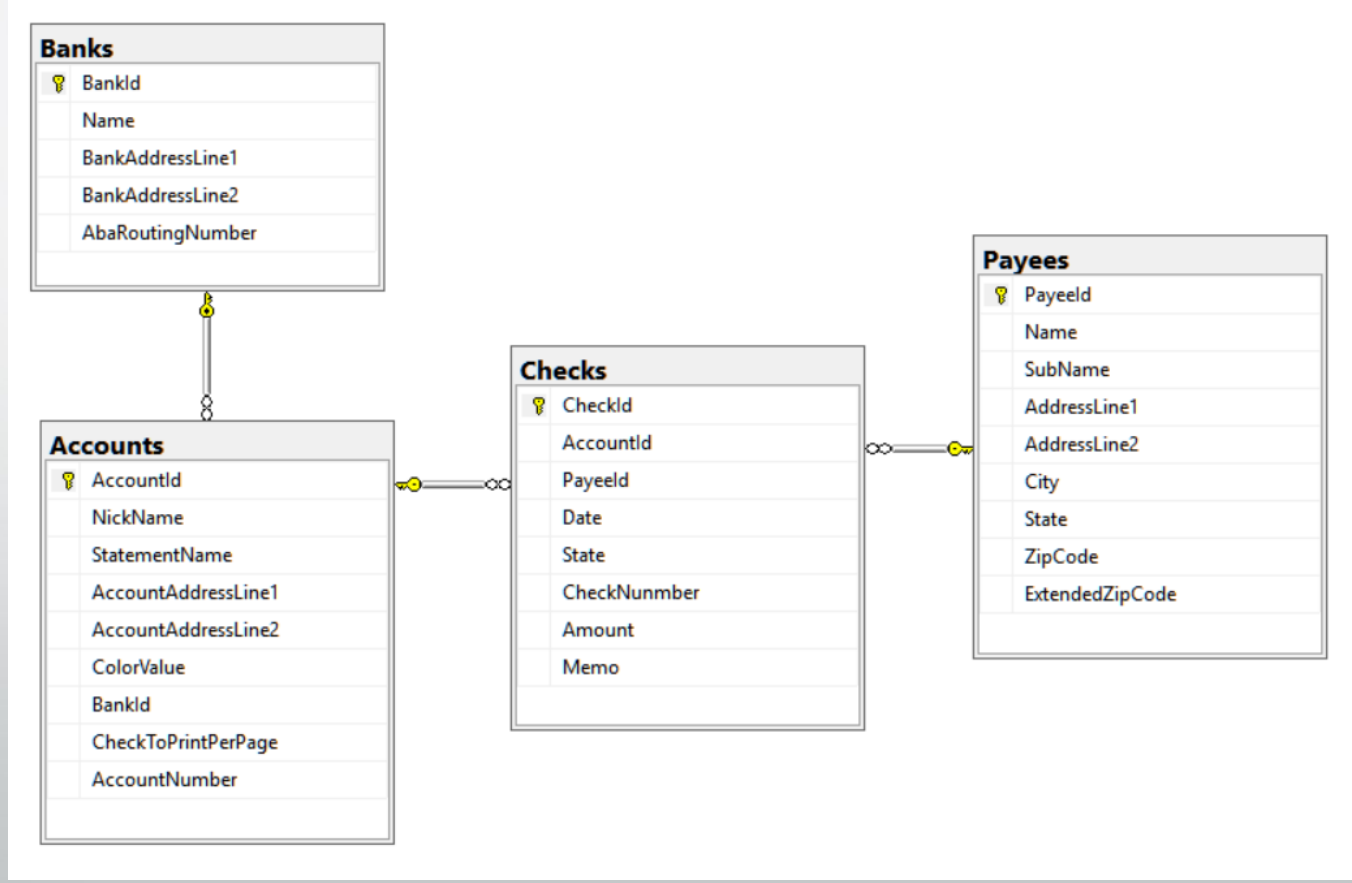
Projections

- Selection of fields is done with the second parameter to the find method
- The project is another JSON object
- WITH forces AS (
  SELECT [ForceId] FROM [wave].[Sensor] WHERE Mean > 0.1)
  SELECT TOP 1 [MachineName], [When] FROM [wave].[Sensor]
  WHERE ForceId In (SELECT ForceId FROM force)

```
#find the one document with a
# force mean greater than 0.1
doc = sensorDocs.find_one(
   { "Force.Mean": { "$gt": 0.1 } },
   { "MachineName", "When"})
pprint.pprint(doc);
```

Relational Entity Modeling

- ERD for a checking application
- Checks table is an intersection between Accounts and Payees
- State is an enumeration including PENDING, WRITTEN and CLEARED



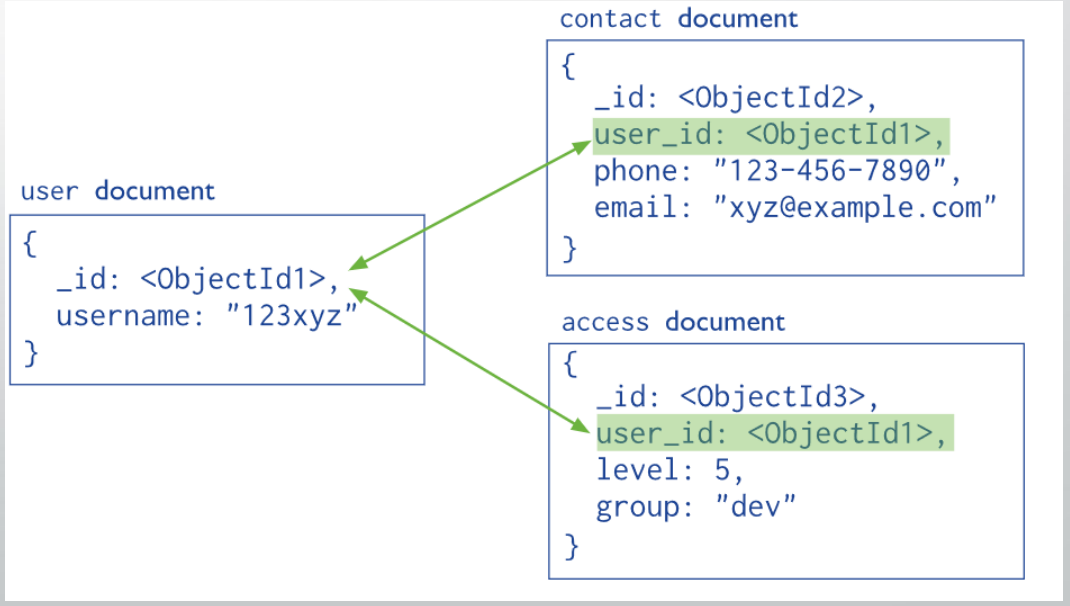
Document Database

- MongoDB API works with BSON documents
- BSON is MongoDB's binary-encoded-version of JSON
- BSON extends the JSON model with additional language feature support
  - E.g. dates
- Cosmos DB only supports standard JSON
- Note that one document collection encompasses four relational tables: Bank, Account, Payee and Check.
- Documents do not need to have consistent structure within the collection
- ISO8601 dates are not supported


```
{  "Date": "2019-04-11T12:17:05.5152519-04:00",
 "State": "Pending",
 "CheckNunmber": 101.0,
 "Amount": 100.02,
 "Memo": "Testing",
 "Payee": {  "Name": "Fred Smith"  },
 "Account": {
  "NickName": "BOA Joint Checking",
  "StatementName": "John and Jane Doe",
  "AccountAddressLine1": "100 West St.",
  "AccountNumber": "57C0A50A08",
  "Bank": {
   "Name": "Bank Of America, NA",
   "BankAddressLine1": "Baltimore, MD 21215",
  "AbaRoutingNumber": "F7CC0F065"
 } } }
```

Embedded & Referenced

- MongoDb supports both embedded and referenced data
- Relational databases only support referenced
  - XML columns are an outlier
- Embedded is great for low cardinality of relationship
  - Customers to phone numbers
- Referenced is best for high cardinality
  - Products intersects Customers via an Orders tables.
- When historic information is important embedded is better
  - If you need the product description at the time of purchase, then embed product in order



.NET Insert

- Add nugget package MongoDb.Driver
- Connect to Cosmos Db using the MongoClient
- Get database and then collection
- Insert a document
  - Document is a hierarchy of POCO objects
  - Similar to Entity Framework

```
//connect to mongo DB
string connectionString =
  GetEnvironmentVariable("MongoDb");
var settings = MongoClientSettings.FromUrl(
  new MongoUrl(connectionString));
settings.SslSettings = new SslSettings()
 { EnabledSslProtocols = SslProtocols.Tls12 };
var mongoClient = new MongoClient(settings);

//get the database and do some mappings
var checkbook = mongoClient.
    GetDatabase(DatabaseId);

var documentCollection = checkbook.
   GetCollection<Check>(CollectionId);
documentCollection.InsertOne(newCheck);
```

Object ID

- Primary Key
- Mongo DB BSON requires a field named *_id*
- C# attribute [BsonId] is valid declaration
- Cross collection projects (joins) can only be done with object ID

- A byte array
- GUID work just fine
- Defaults to 12-bytes if not specified
  - a 4-byte value representing the seconds since the Unix epoch,
  - a 5-byte random value, and
  - a 3-byte counter, starting with a random value

```
{
"_id": ObjectId("58f65e1198f3a12c7090e68c"),
"id": "WakefieldFamily",
"parents": [{
        "familyName": "Wakefield",
        "givenName": "Robin"
},
```

Upsert

- Replace() entire document
- Update() portions of document
- Upsert is option on both commands
  - Performs insert if no match
- Equivalent SQL DML

```
UPDATE [testdb].[Checkbook].[Checks]
  SET [State] = 4
 WHERE [CheckId] = 5
```

```
var documentCollection = checkbook.
  GetCollection<Check>(CollectionId);

var filter = Builders<Check>.
  Filter.Eq(c => c.Id, clearedCheck.Id);
var update = Builders<Check>.
  Update.Set(c => c.State, clearedCheck.State);

var result = documentCollection.
    UpdateOne(filter, update);
```

Geo-distributed Database

Buoy Customer and Data Center Location

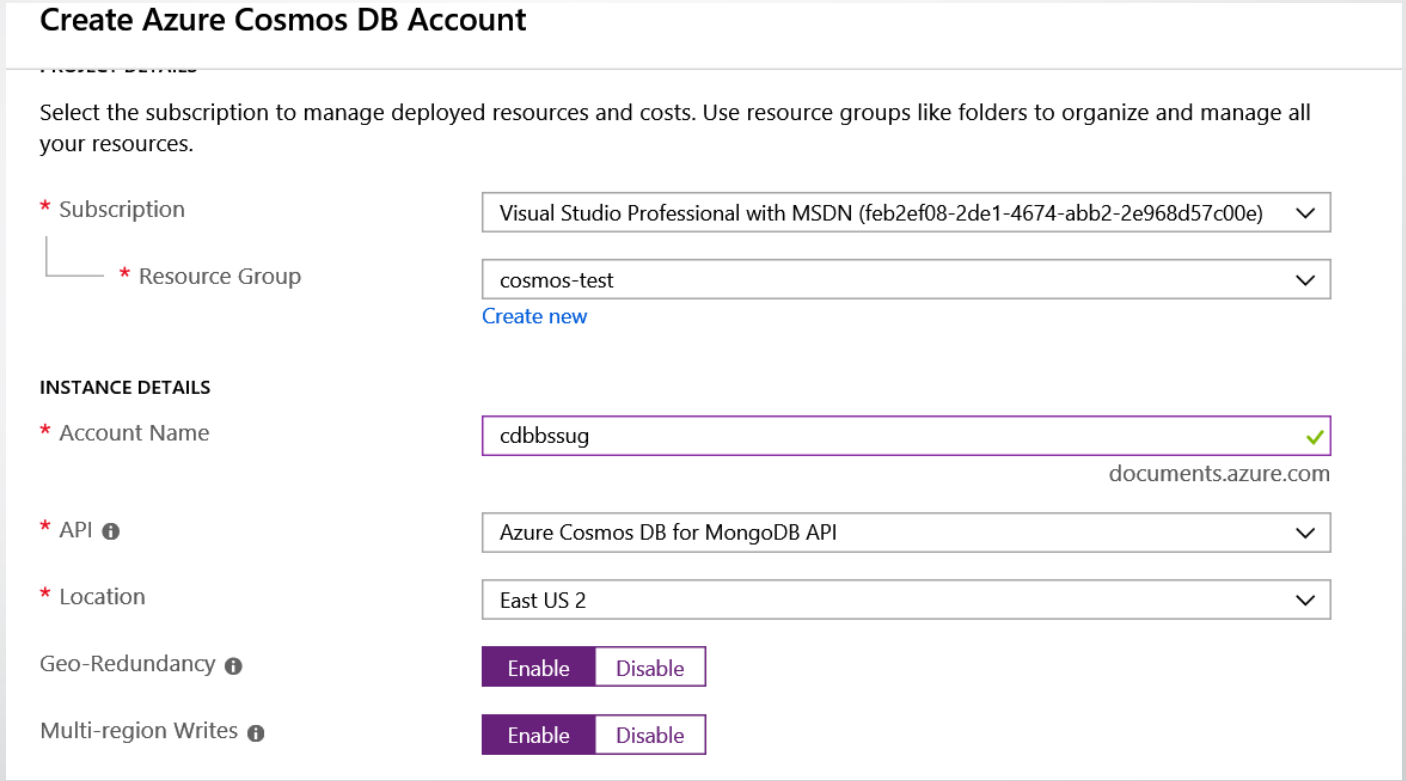
Chesapeake Bay, East Coast 2

Tokyo Bay, Japan East

Manilla Bay, South East Asia

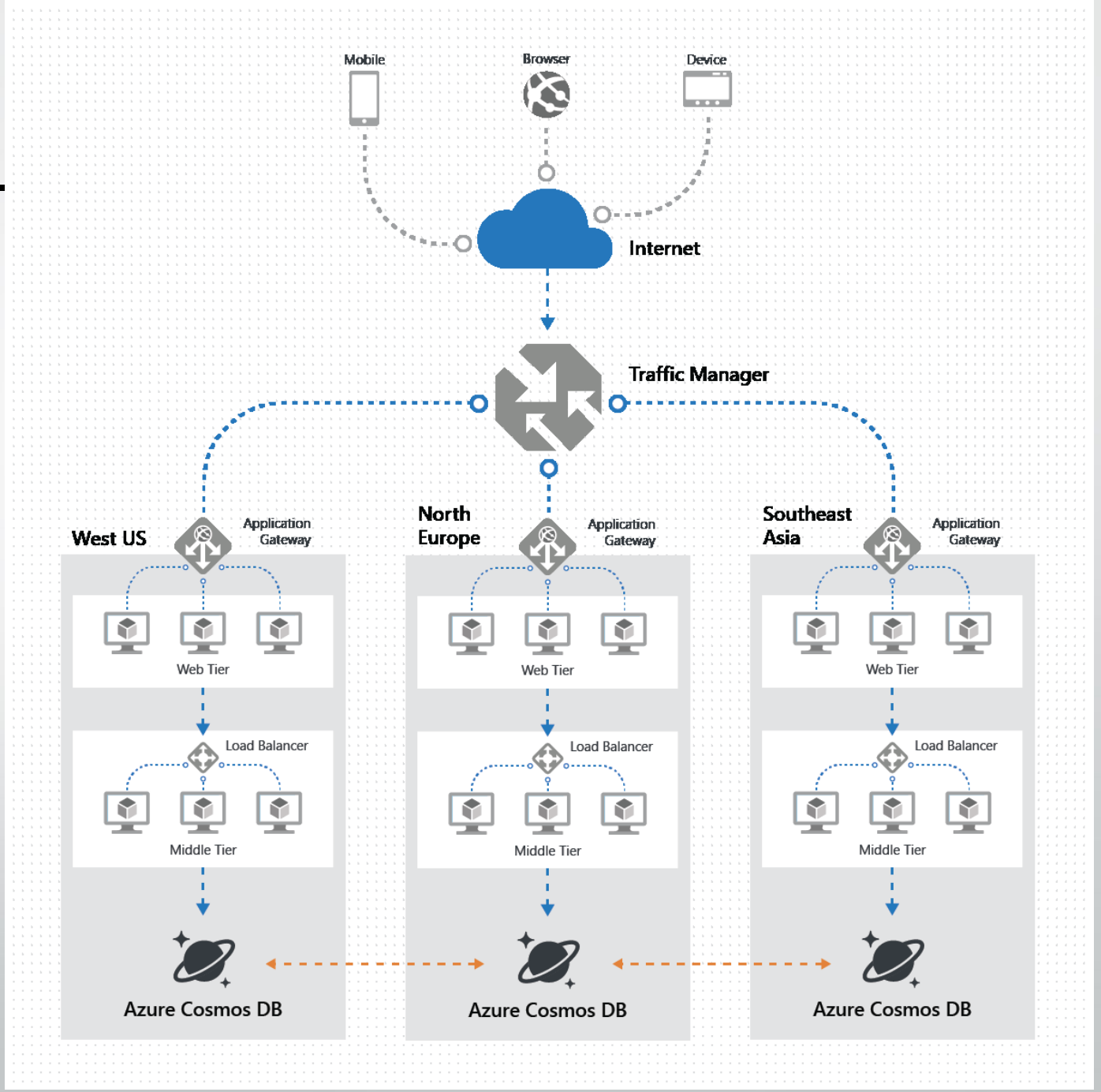
DB Account Creation

- Add a resource to your Azure subscription
- Name must be unique
- Must choose the container content & API
- Select storage methodology



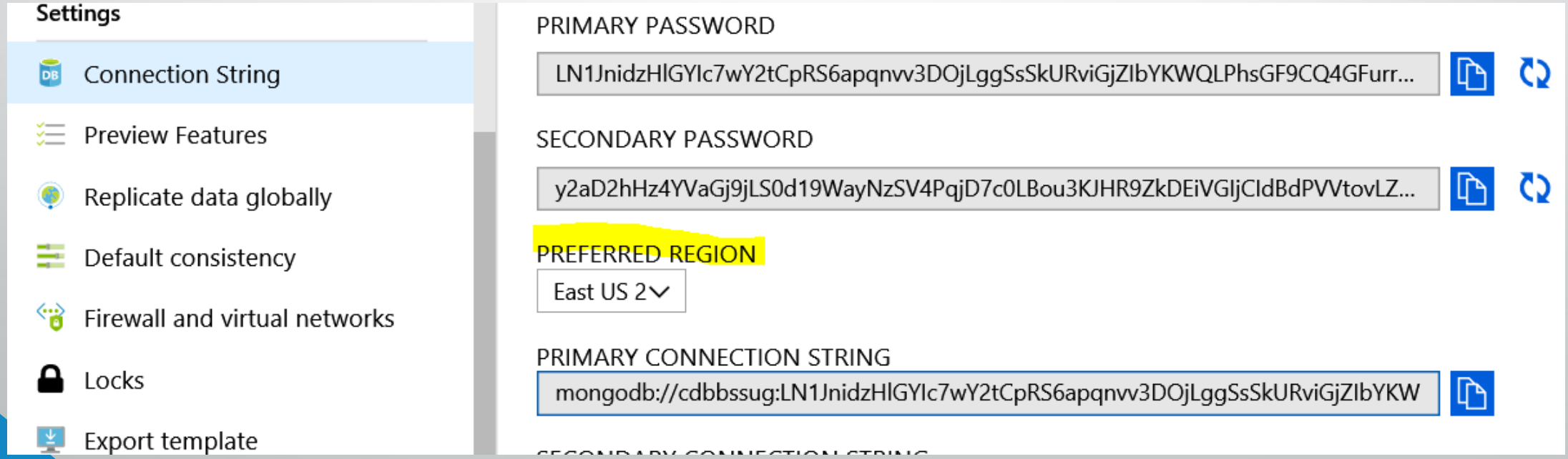
Execution Geo-distribute

- Traffic routing to single DNS across data centers and computational machines
- Traffic is routed to closest data center automatically
- Claim: 99.999% read and write availability all around the world



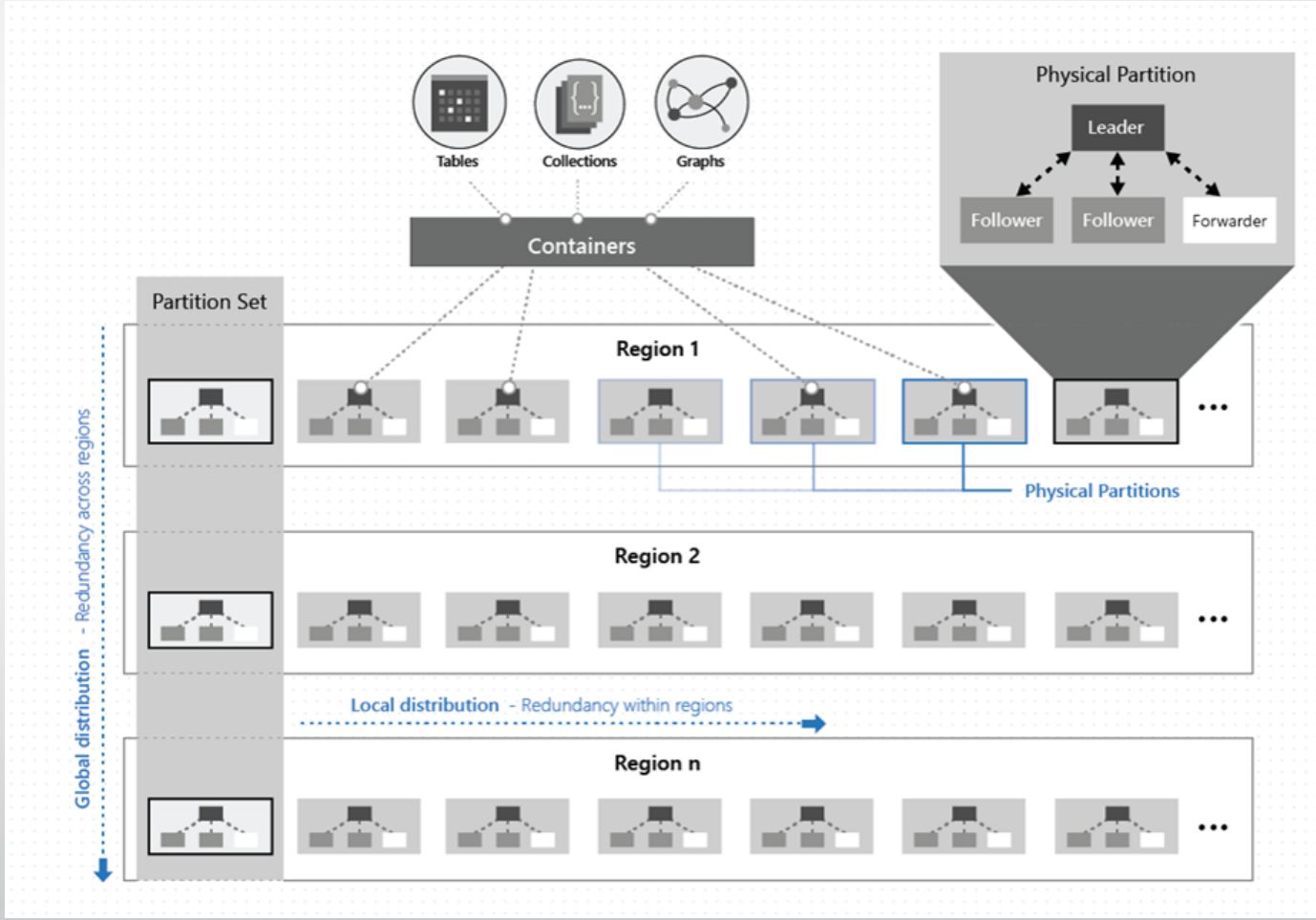
Data Center

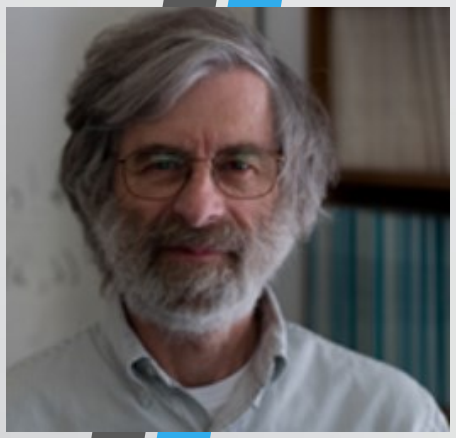
- Can embed data center in connection string
- Local Distribution: In a given region, data within a container is distributed by using a *partition-key*, which you provide and is transparently managed by the underlying physical partitions
- Global Distribution: Each physical partition is also replicated across geographical regions



Replica Distribution

- Replica-sets, a modular Lego block for coordination
- Partition-sets, a dynamic overlay of one or more geographically distributed physical partitions
- Claim: Unlimited elastic write and read scalability
- Claim Guaranteed reads and writes served in less than 10 milliseconds at the 99th percentile





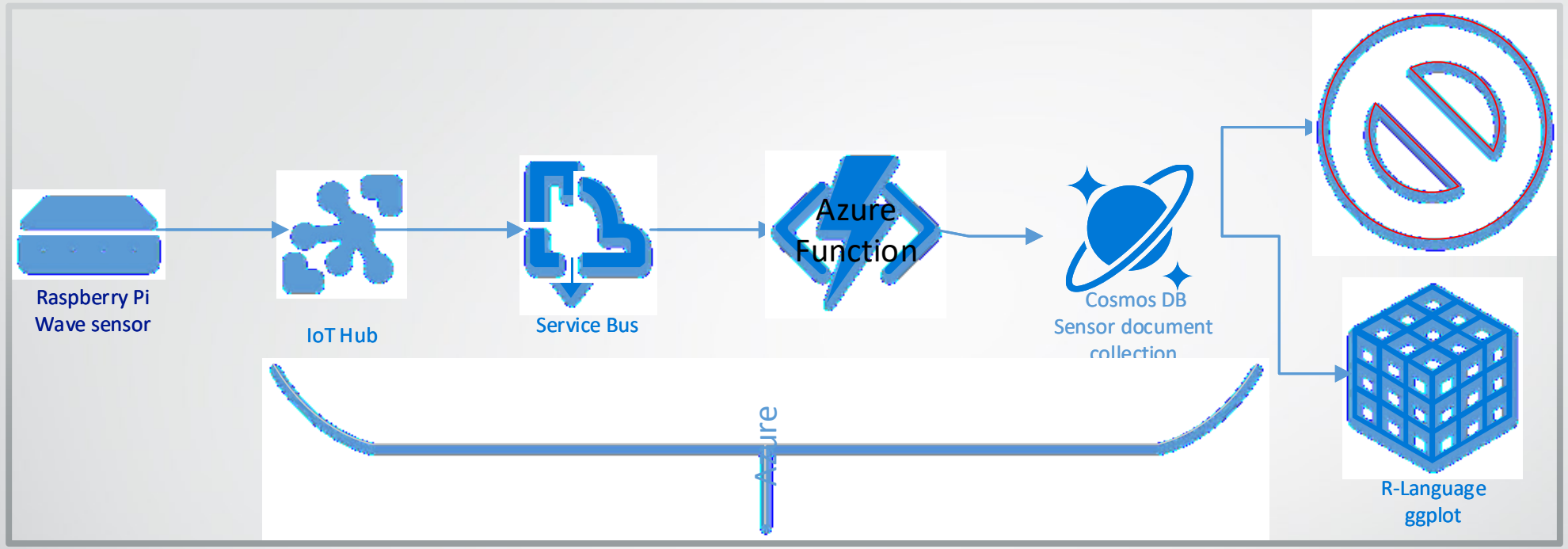
Concurrency Models

- Similar to SQL Server
  - TRANSACTION ISOLATION LEVEL
- Affects replica's communications
- Works across all container types
- Conflict Resolution: Last Write Wins (by time)
- Five consistency models
  - Only directly selectable via Cosmos DB API
  - MongoDb read-write-concerns are mapped to consistency model

Consistency Model	Read	Write
Strong	Complete latest data	In order across sessions
Bounded Staleness	Read can max k-versions back and T-time back	Force writes before point
Session	Session latest data	In order for session
Consistent prefix	Nothing out of order, but as-of certain point in the past	Force writes before point
Eventual	Catch-as-catch-can	Eventually replicate are the same

Data Source: Bobble Head Santa

- Mount Raspberry Pi on a bobble head Santa.
  - Place an inertial measurement unit in Santa's head.
  - Send data to IoT Hub
- Use gyroscope & accelerometer to determine when the head is bobbing.
- Simulate wave motion with bobble head.



Data Flow

From device to Cosmos DB and reporting
PowerBI adapter is still beta and does not work

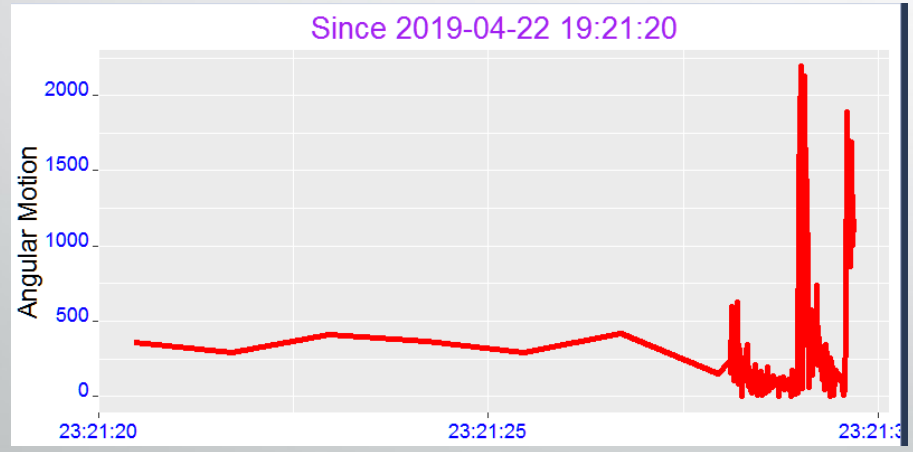
Azure Function

- Automatically invoked on blocks of service bus data
- Performs batch insert of MongoDb BSON documents into Cosmos DB
- Beware of exceeding RU/s limit
  - Receive uncategorized error

```
[FunctionName("HubServiceBus")]
public static void Run(
   [EventHubTrigger("wavehub",  Connection = "WaveHub")]
  string hubMessage,   TraceWriter log)
{…}
…
/*  Look for too much load
   A write operation resulted in an error.
   Message: {   "Errors":["Request rate is large"]}
   ActivityId: …,  */
if (ex.WriteError != null &&
   ex.WriteError.Code == Uncatorgized &&
   ex.WriteError.Message.Contains("Request rate is large"))
{
   log.Warning($"Request rate is large. Sleeping: {sleepTime}.");
…
```

R Language

- Mongolite is an excellent R-library that works with Cosmos DB
- Ggplot2 is a library that plots data
- install.packages('mongolite') & include.packages('ggplot2') is VS
- MongoDb uses periods for nested fields
- R language uses dollar sign for nested fields




```
library(mongolite)
library(ggplot2)

mgo <- mongo(db = "wave", collection = "Sensor",
    url="mongodb://...")

lastMin = Sys.time() - (60 * 1) #latest minute of data

query = paste('{ "When" : { "$gte" : { "$date" : "',
      format(lastMin, "%Y-%m-%dT%H:%M:%SZ", 'UTC'), '"} } }',
      sep = "")
recent = mgo$find(query,
       field = '{ "When": true, "MachineName" : true,
"Angles.Absolute.X" : true }' ,
      limit = 100)

ggplot(recent, aes(x = When, y = Angles$Absolute$X)) +
   geom_line(aes(group = 1), size = 2, color = "red") +
   scale_x_datetime(date_labels = "%H:%M:%S")
```

Conclusions

- Best Uses for Cosmos DB Today
  - Geographically distributed applications
  - Low cost to start acquiring document data
  - When data format will evolve
  - Retrieval speed across very large data is important
- Must be able to handle high cost of data mining development

Useful Links

- Cosmos DB documentation home https://docs.microsoft.com/en-us/azure/cosmos-db/
- Cosmos DB API https://docs.microsoft.com/en-us/dotnet/api/overview/azure/cosmosdb?view=azure-dotnet
- Theory of Cosmos DB Engine https://www.vldb.org/pvldb/vol8/p1668-shukla.pdf
- Comparison of mongodb to Cosmos DB https://db-engines.com/en/system/Microsoft+Azure+Cosmos+DB%3BMongoDB
- MongoDb manual https://docs.mongodb.com/manual/
- Getting started with Cosmos DB https://www.sqlshack.com/getting-started-with-azure-cosmos-db-and-mongodb-api/
- JSON/BSON Dates: https://jira.mongodb.org/browse/CSHARP-2233
- Cosmos DB support level of MongoDb API https://docs.microsoft.com/en-us/azure/cosmos-db/mongodb-feature-support