Serverless and Storage

Alex Pshul

Software Architect & Consultant

@AlexPshul

alex@pshul.com

http://pshul.com

http://codevalue.net

https://www.meetup.com/Code-Digest/

Code.Digest();

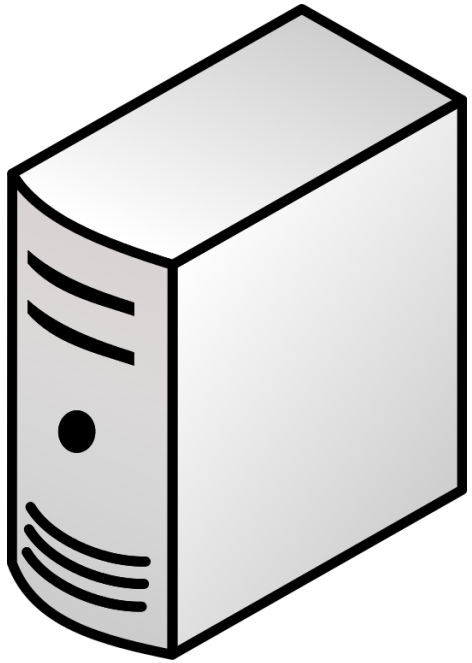
About Me

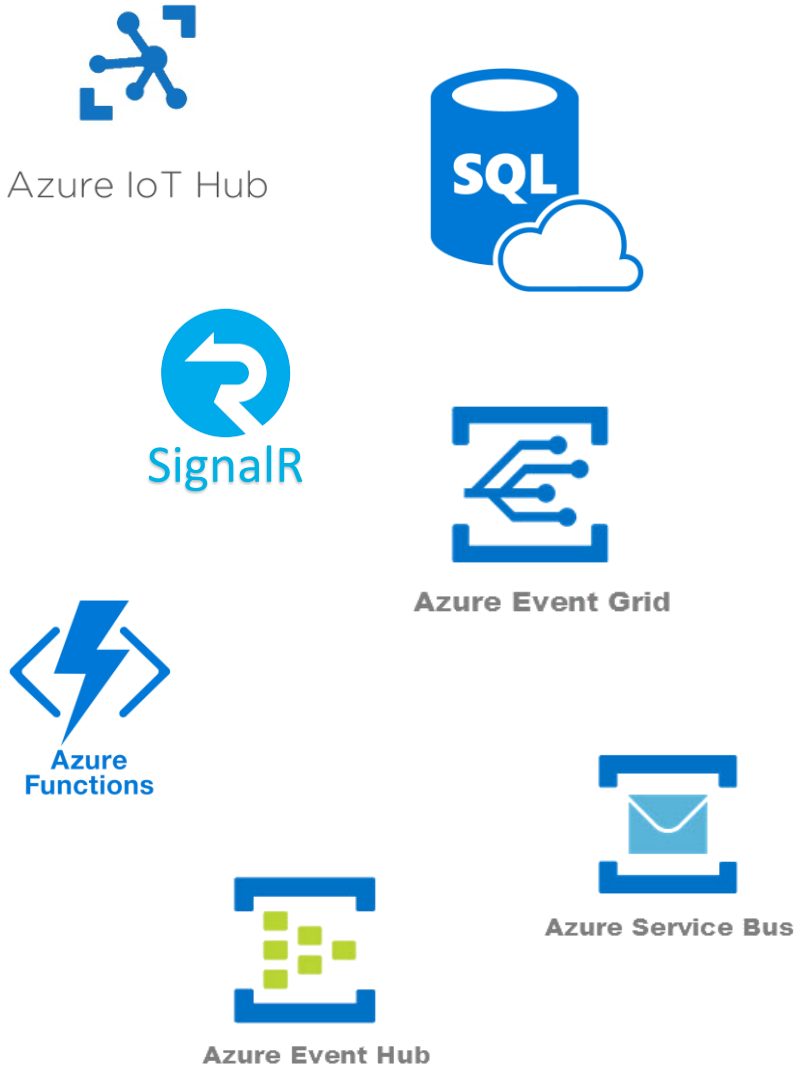
Code.Digest();

Alex Pshul

- Architect, Consultant and lecturer
- More than 9 years of hands on experience
- Co-organizer of the Code.Digest Meetup
  - https://www.meetup.com/Code-Digest/
- Talk to me about:
  - Software Development
  - Hardware and Gadgets
  - Gaming
  - Animals

What is serverless?





Benefits of serverless

- “Pinnacle of PaaS compute”
- Not just hardware “servers”, but software servers are also managed for you
- Focus on business logic, not solving technical problems not core to business
- Lower effort to get started makes it easier to experiment (bots, etc.)

When to go serverless?

- Stateless → Scale
- Too complicated to deploy a traditional backend
- Workload is sporadic (very low & high scale)
- (Human) Operational costs need to stay low
- Lots of different services involved

Suggestions for getting started

- For existing services, start small. Replace 1 API or background processing item
- Integration is a great place to introduce serverless, because it is often a new layer on top of old layers
- For new services, establish a pattern early and stick with it. Lack of tooling/established patterns mean you pay an early adopter tax. Build automation asap

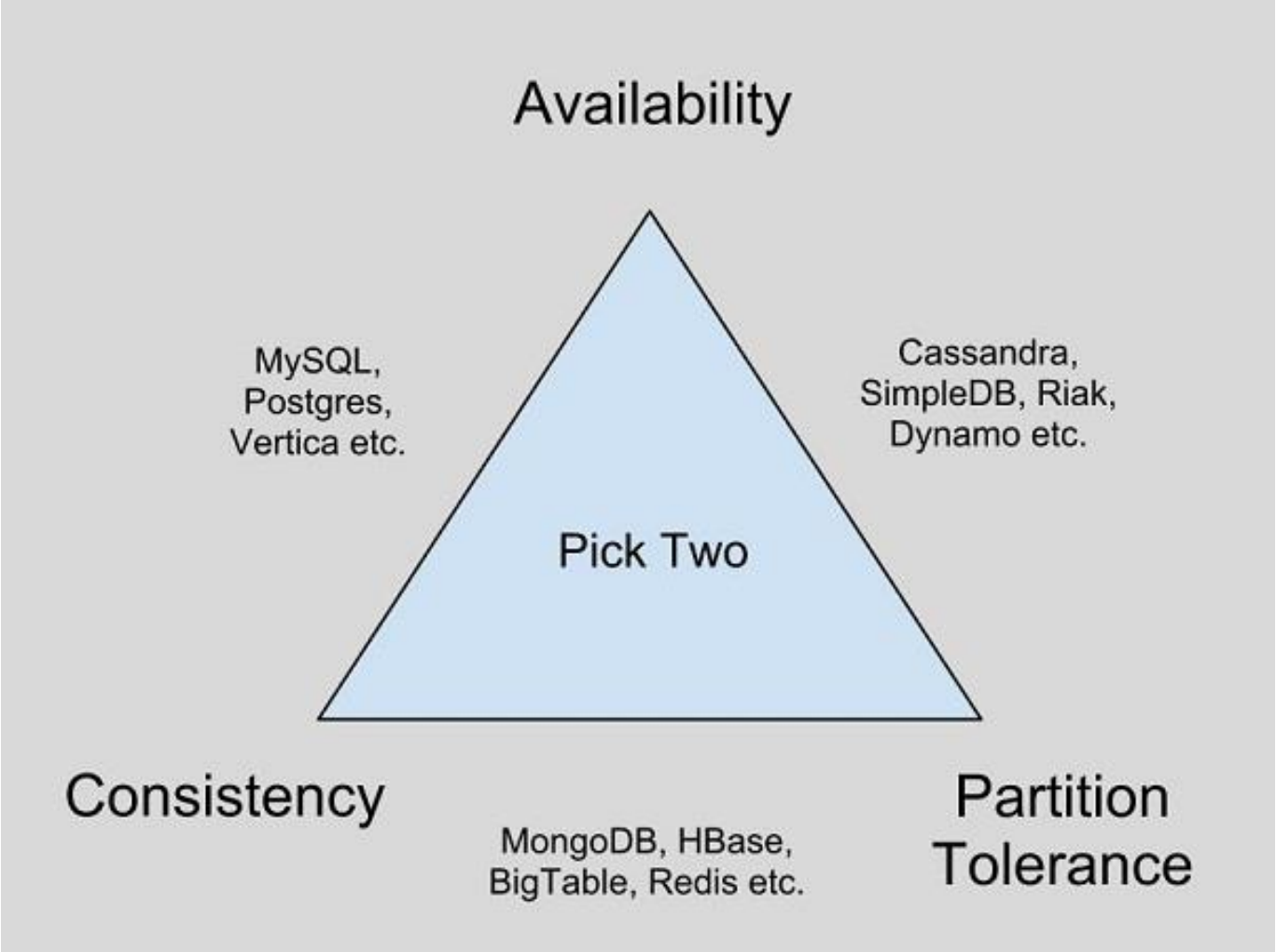
Azure Storage

Azure Storage Options

Relational	Key-Value	Column Family	Document	Graph	Files	Search
• Azure SQL Server • SQL Server • Postgres • MySQL • Oracle • SQLite	• Azure Blob Storage • Azure Table Storage • CosmosDB • Redis • Memcached • Riak	• Cassandra • HBase	• Cosmos DB (previously Document DB) • MongoDB • RavenDB • CouchDB	• Cosmos DB • Neo4J	• Azure Blob • Azure File Storage (SMB)	• Elasticearch • Azure Search

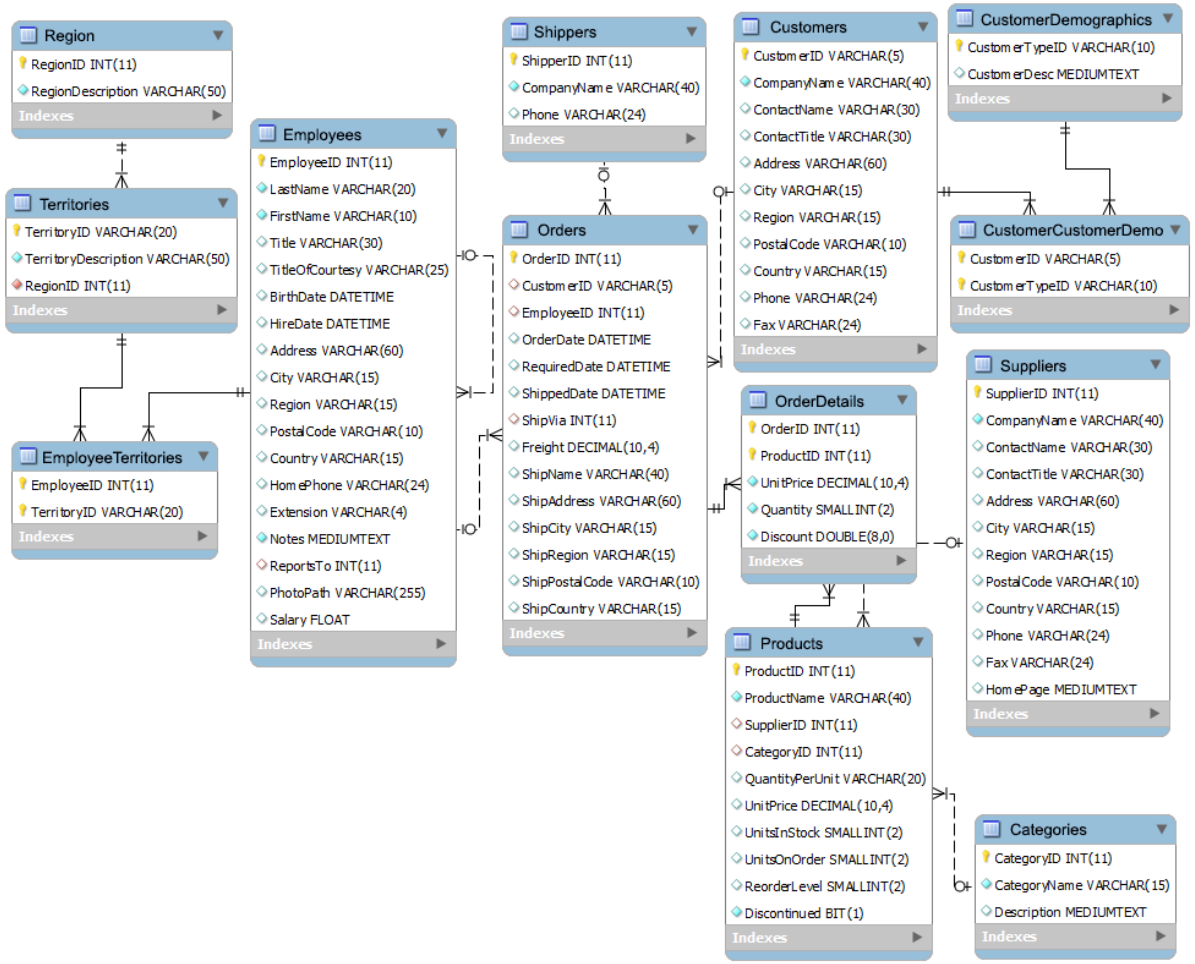
+ **Azure Queues**, which are part of the Azure storage infrastructure

CAP Theorem


Availability
MySQL,
Postgres,
Vertica etc.
Cassandra,
SimpleDB, Riak,
Dynamo etc.
Pick Two
Consistency
MongoDB, HBase,
BigTable, Redis etc.
Partition
Tolerance

RDBMS

- Relational model exist for many years
- Tables, columns, relationships and constraints
- Azure PaaS solutions:
  - MS SQL Server
  - PostgreSQL
  - MySQL



Azure SQL - Basics

SQL Database

SQL Server database technology as a service

Fully Managed

Enterprise-ready with automatic support for HA

Designed to scale out elastically with demand

Ideal for simple and complex applications

Feature comparison with SQL Server –
https://docs.microsoft.com/en-us/azure/sql-database/sql-database-features

Server Provisioning

Server Defined

Service head that contains databases

Connect via automatically generated FQDN (xxx.database.windows.net)

Initially contains only a **master** database

Provision Servers Interactively

Log on to Windows Azure Management Portal

Create a SQL Database server

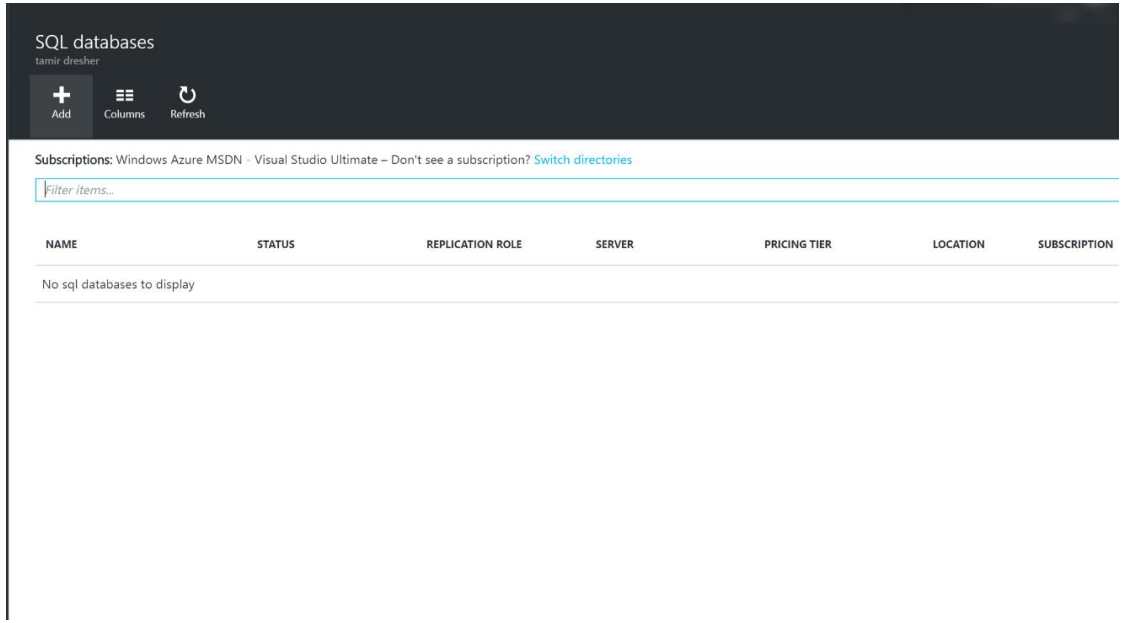
Specify admin login credentials

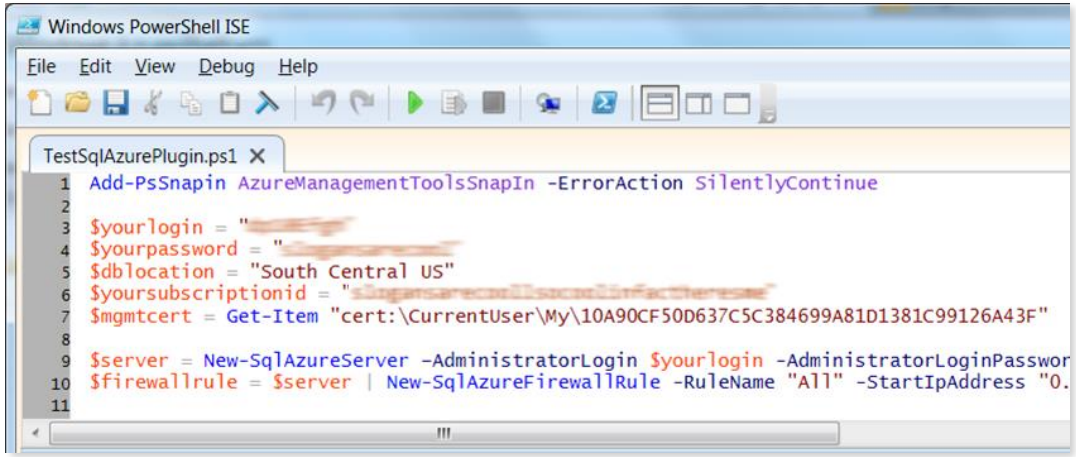
Add firewall rules and enable service access

Automate Server Provisioning

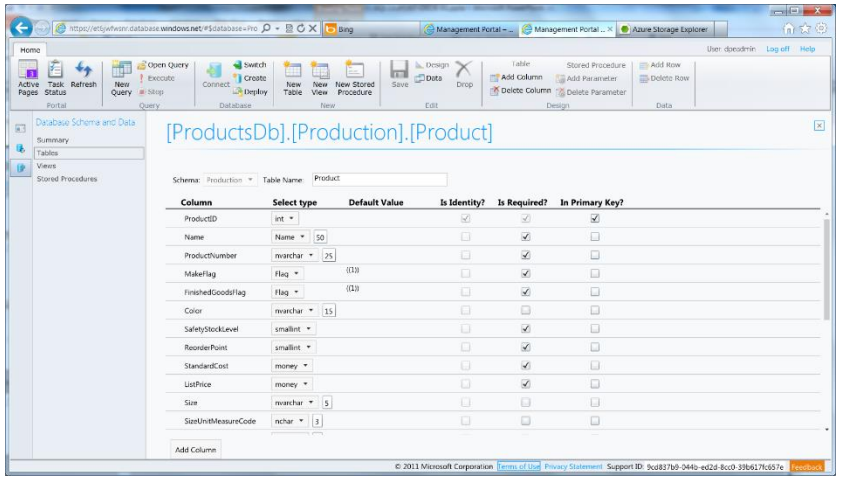
Use Windows Azure Platform PowerShell cmdlets
(or use REST API directly)

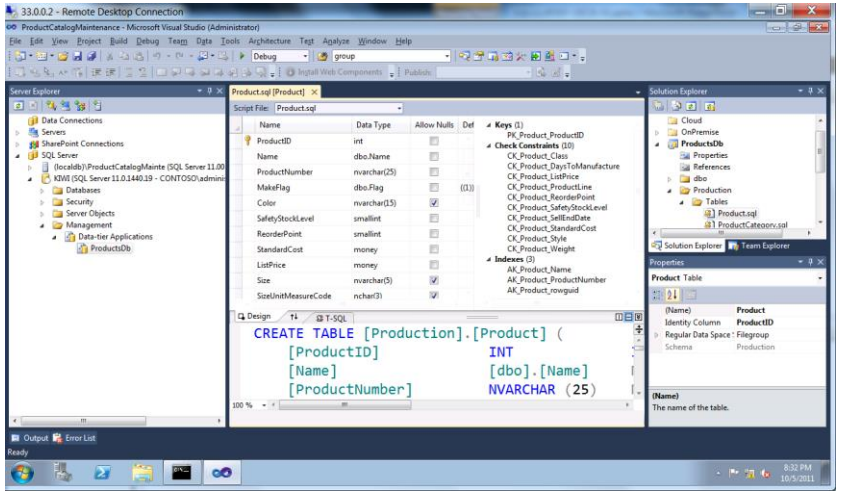
wappowershell.codeplex.com





Enhanced Tooling





SQL Database Management Portal

Web designers for tables, views, stored procs

Interactive query editing and execution

SQL Server Data Tools (SSDT)

Visual Studio IDE for database development

Includes modern designers and projects with declarative, model-driven development

Develop and test in both connected and disconnected states

Platform targeting for both SQL Server (2005 and above) and SQL Database

Get it free with Web PI, with SQL Server 2012 and with Visual Studio 11

SQL Database editions

Service Tier	Performance Level	Common App Pattern	Performance			Business Continuity	
			Max DB Size	Transaction Perf. Objective	DTU	PITR	DR / GEO-Rep
Basic	Basic	Small DB, SQL opp	2 GB	Reliability / Hr.	5	7 Days	DB Copy + Manual Export
Standard	S0 S1 S2	Wrkgp/cloud app, multiple concurrent operations	250 GB	Reliability / Min.	10 20 50	14 Days	DB Copy + Manual Export
Premium	P1 P2 P3	Mission Critical, High volume, Many concurrent Users	500 GB	Reliability / sec.	100 200 800	35 Days	Active Geo-replication

http://dtucalculator.azurewebsites.net/

Azure Storage Account

Azure Storage Account Services

Blobs

Highly scalable, REST based cloud object store

Block Blobs: Sequential file I/O

Page Blobs: Random-write pattern data

Cool Blob Storage

Tables

Massive auto-scaling NoSQL store

Dynamic scaling based on load

Scale to PBs of table data

Fast key/value lookups

Queues

Reliable queues at scale for cloud services

Decouple and scale components

Message visibility timeout and update message to protect against unreliable dequeuers

Disks

Persistent disks for Azure IaaS VMs

Built on page blobs

Premium Storage Disks: SSD based, high IOPS, low latency

Files

Fully Managed File Shares in the Cloud

Map to file share, standard file system semantics

"Lift and shift" legacy apps

Code against (REST API)

Use on Windows & Linux VMs

Real Worlds Examples:
XBOX – Cloud Game Save, Halo 4, Music, Kinect data collection
OneDrive
Bing – stores raw data from Twitter and Facebook to digest later
Skype – Video Messaging

Azure Storage Account types

- General-purpose Storage Accounts
  - Tables, Queues, Files, Blobs and Azure virtual machine
  - performance tiers:
    - **Standard storage performance tier** allows you to store Tables, Queues, Files, Blobs and Azure virtual machine disks.
    - **Premium storage performance tier** provides High-Performance Storage for Azure Virtual Machine Workloads
- Blob Storage Accounts
  - specialized storage for unstructured data as blobs (objects)
    - Only block and append blobs
  - Access tiers:
    - **Hot access tier** indicates that the objects in the storage account will be more frequently accessed.
    - **Cool access tier** indicates that the objects in the storage account will be less frequently accessed.

Microsoft Azure

Search resources, services, and docs (G+/)

alex@pshul.com
DEFAULT DIRECTORY

Home > New

New

Search the Marketplace

Azure Marketplace See all

- Get started
- Recently created
- AI + Machine Learning
- Analytics
- Blockchain
- Compute
- Containers
- Databases
- Developer Tools
- DevOps
- Identity
- Integration
- Internet of Things
- Media
- Mixed Reality
- IT & Management Tools
- Networking
- Software as a Service (SaaS)
- Security
- Storage
- Web

Featured See all

Storage account - blob, file, table, queue
Quickstarts + tutorials

Azure Stack Edge / Data Box Gateway
Learn more

Data Lake Storage Gen1
Quickstarts + tutorials

Azure Data Box
Learn more

Backup and Site Recovery
Quickstarts + tutorials

PREVIEW
AltaVault AVA-c4, version 4.4.1 (preview)
Learn more

PREVIEW
Cloudian HyperCloud for Azure (preview)
Learn more

PREVIEW
Veeam Cloud Connect for the Enterprise (preview)
Learn more

Storage account

Microsoft Azure

Search resources, services, and docs (G+/)

alex@pshul.com
DEFAULT DIRECTORY

Home > authtest123 | Access keys

authtest123 | Access keys

Storage account

Search (Ctrl+/)

- Overview
- Activity log
- Access control (IAM)
- Tags
- Diagnose and solve problems
- Data transfer
- Storage Explorer (preview)

Settings

- Access keys
- Geo-replication
- CORS
- Configuration
- Encryption
- Shared access signature
- Firewalls and virtual networks
- Advanced security
- Properties
- Locks
- Export template

Blob service

- Containers
- Custom domain
- Data protection
- Azure CDN

Use access keys to authenticate your applications when making requests to this Azure storage account. Store your access keys securely - for example, using Azure Key Vault - and don't share them. We recommend regenerating your access keys regularly. You are provided two access keys so that you can maintain connections using one key while regenerating the other.

When you regenerate your access keys, you must update any Azure resources and applications that access this storage account to use the new keys. This action will not interrupt access to disks from your virtual machines. Learn more about regenerating storage access keys

Storage account name

authtest123

key1

Key

Connection string

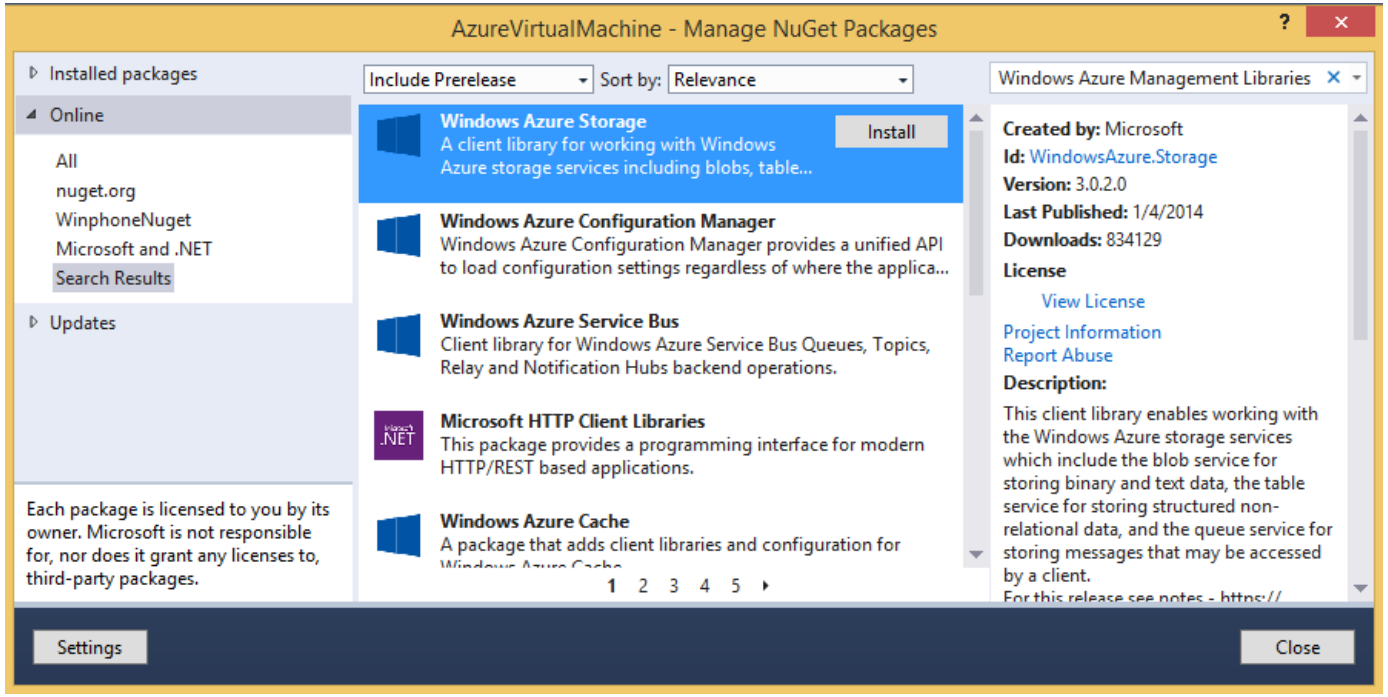
key2

Key

Connection string

Storage API

- REST
- Client API from SDK: WindowsAzure.Storage namespace
  - A wrapper around the REST API
  - Hides many of the complexities of the service + Auto retries



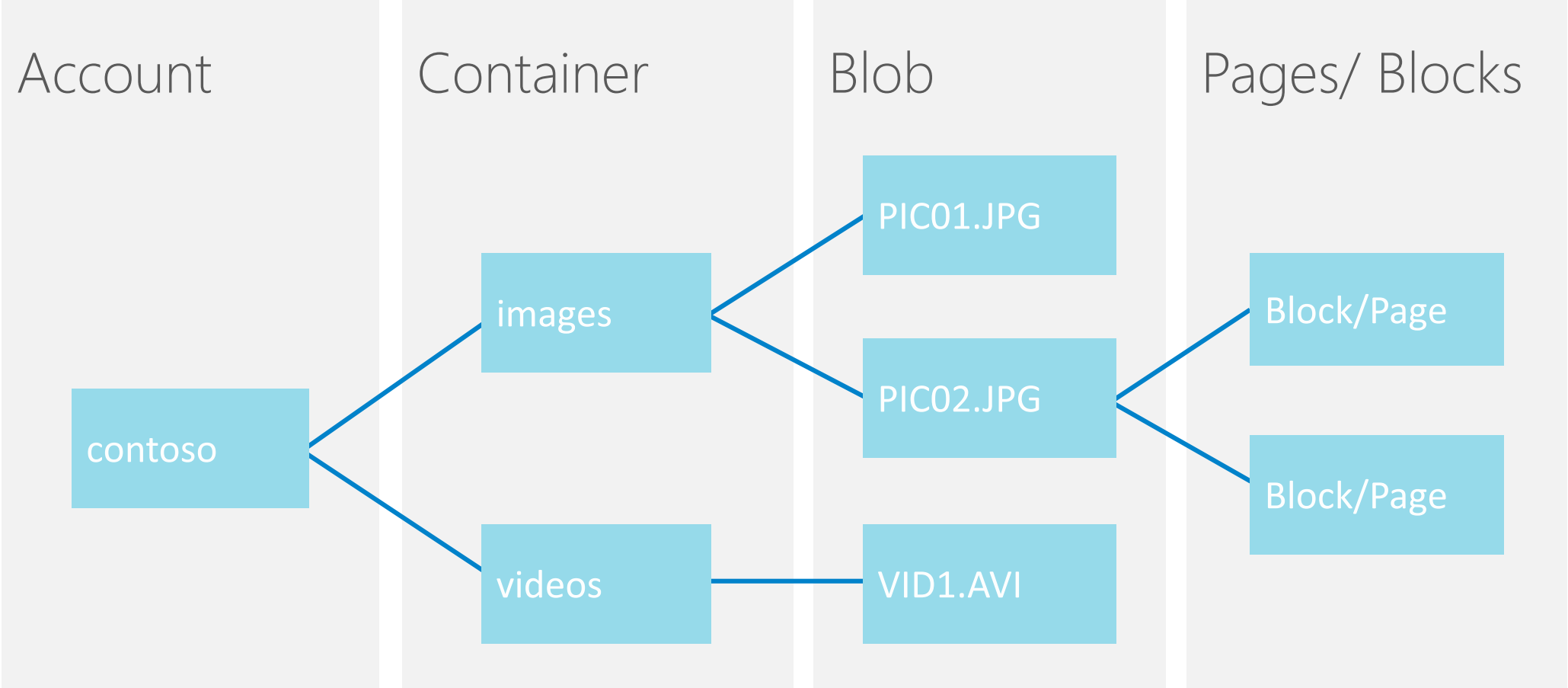
Cloud Storage - Azure BLOB Storage

- BLOB – Binary Large OBject
- Storage for any type of entity such as binary files and text documents
- Distributed File Service (DFS)
  - Scalability and High availability
- BLOB file is distributed between multiple server and replicated at least 3 times

- Get Started with Storage Account
- Get Started with Blob Storage

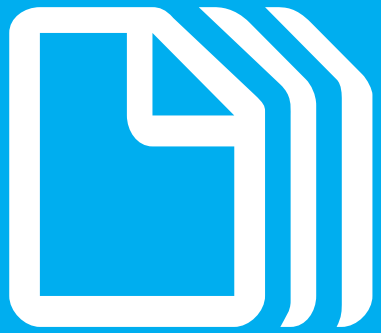
Azure Blob Storage Concepts

```
http://<account>.blob.core.windows.net/<container>/<blobname>
```

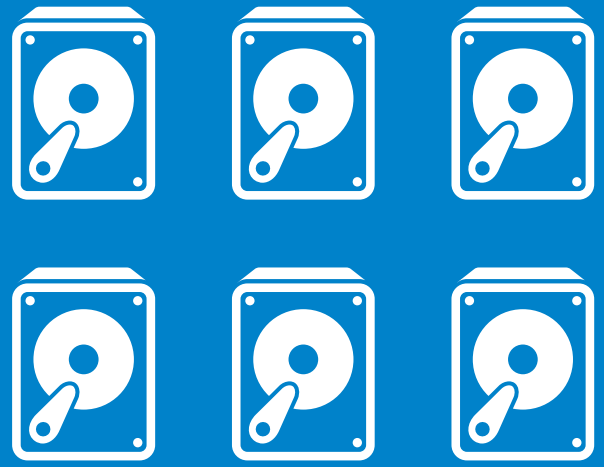


Blob Operations

PutBlob
GetBlob
DeleteBlob
CopyBlob
SnapshotBlob
LeaseBlob



REST



Windows Azure Storage

Page Blob in code

- Creating

```
using Microsoft.WindowsAzure.StorageClient ;

CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
                $"DefaultEndpointsProtocol=https;AccountName={accountName};AccountKey={accountKey}");
CloudBlobClient blobClient = storageAccount.CreateCloudBlobClient();
CloudBlobContainer container = blobClient.GetContainerReference(containerName);
container.CreateIfNotExist();
CloudPageBlob pageBlob = container.GetPageBlobReference(blobName);
pageBlob.Create(blobSize);
```

- Writing

```
pageBlob.WritePages(dataStream, startingOffset) ;
```

Page Blob in code

- Reading

```
BlobStream blobStream = pageBlob.OpenRead();
byte[] buffer = new byte[rangeSize] ;
blobStream.Seek(blobOffset, SeekOrigin.Begin) ;
int numBytesRead = blobStream.Read(buffer, bufferOffset, rangeSize);
```

- Clear Pages

```
pageBlob.ClearPages(startOffset, length)
```

Concurrency

- Optimistic concurrency – Timestamps/ETags
  - Timestamp based – If-Modified-Since and If-UnModified-Since
  - ETag based – If-Match and If-None-Match (can force update with *)
  - Conditional update with supplied Timestamp or ETag will fail if conditions not met
- Pessimistic Concurrency - Leases
  - Lease Blob for exclusive write and delete access
  - 15-60s lease duration (can be renewed) or infinite lease (locks)
  - Can change lease id to acquire ownership in a chain/workflow
  - Can also acquire on containers to prevent container deletion
- Last Writer wins
  - https://azure.microsoft.com/en-us/blog/managing-concurrency-in-microsoft-azure-storage-2/
  - https://msdn.microsoft.com/en-us/library/dd179371.aspx

Optimistic Concurrency

```
string orignalETag = blockBlob.Properties.ETag;
try
{
    blockBlob.UploadText(helloText,
        accessCondition: AccessCondition.GenerateIfMatchCondition(orignalETag));
}
catch (StorageException ex)
{
    if (ex.RequestInformation.HttpStatusCode == (int)HttpStatusCode.PreconditionFailed)
    {
        Console.WriteLine("Blob's orignal etag no longer matches");

    }
    else
        throw;
}
```

Passimistic Concurrency

```
string lease = blockBlob.AcquireLease(TimeSpan.FromSeconds(15), null);

// Update blob using lease. This operation will succeed
var accessCondition = AccessCondition.GenerateLeaseCondition(lease);
blockBlob.UploadText("update", accessCondition: accessCondition);

try
{
    // Below operation will fail as no valid lease provided
    blockBlob.UploadText("Update without lease, will fail");
}
catch (StorageException ex)
{
    if (ex.RequestInformation.HttpStatusCode == (int)HttpStatusCode.PreconditionFailed)
        Console.WriteLine("Blob's lease does not match");
    else
        throw;
}
```

Transient Faults

- " **transient fault** is a fault that is no longer present if power is disconnected for a short time and then restored." (http://en.wikipedia.org/wiki/Transient_fault#Transient_fault)
- Many faults in connectivity to cloud are transient by nature
- Commonly occur when connecting to service or database

Transient Faults handling

- Retry Logic
  - Linear – every fixed amount of time
  - Exponential – if the server is heavy-used (throttling) we don't want to flood it
    immediate….1 sec….5 seconds….etc.

- Idempotency
  - operations in mathematics and computer science, that can be applied multiple times without changing the result beyond the initial application (wikipedia)
  - Same messages could be sent more than once or out of sequence
  - Design for idempotency

Retry Policy Application

- Microsoft.WindowsAzure.Storage.RetryPolicies.IRetryPolicy Interface
- ExponentialRetry
- LinearRetry
- NoRetry

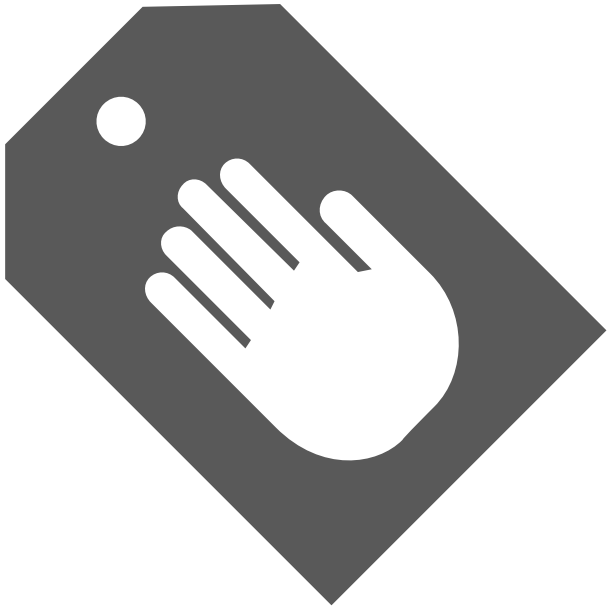
- Default is exponential – if you don’t want any retry logic then you must override

Cloud Storage - Table Storage

- Not RDBMS
  - No relationships between entities
  - NoSql
- Entity can have up to 255 properties - Up to 1MB per entity
- Mandatory Properties for every entity
  - PartitionKey & RowKey (only indexed properties)
    - Uniquely identifies an entity
    - Same RowKey can be used in different PartitionKey
    - Defines the sort order
  - Timestamp - Optimistic Concurrency
- Strongly consistent
- Get Started with Table Storage

Shared Access Signatures

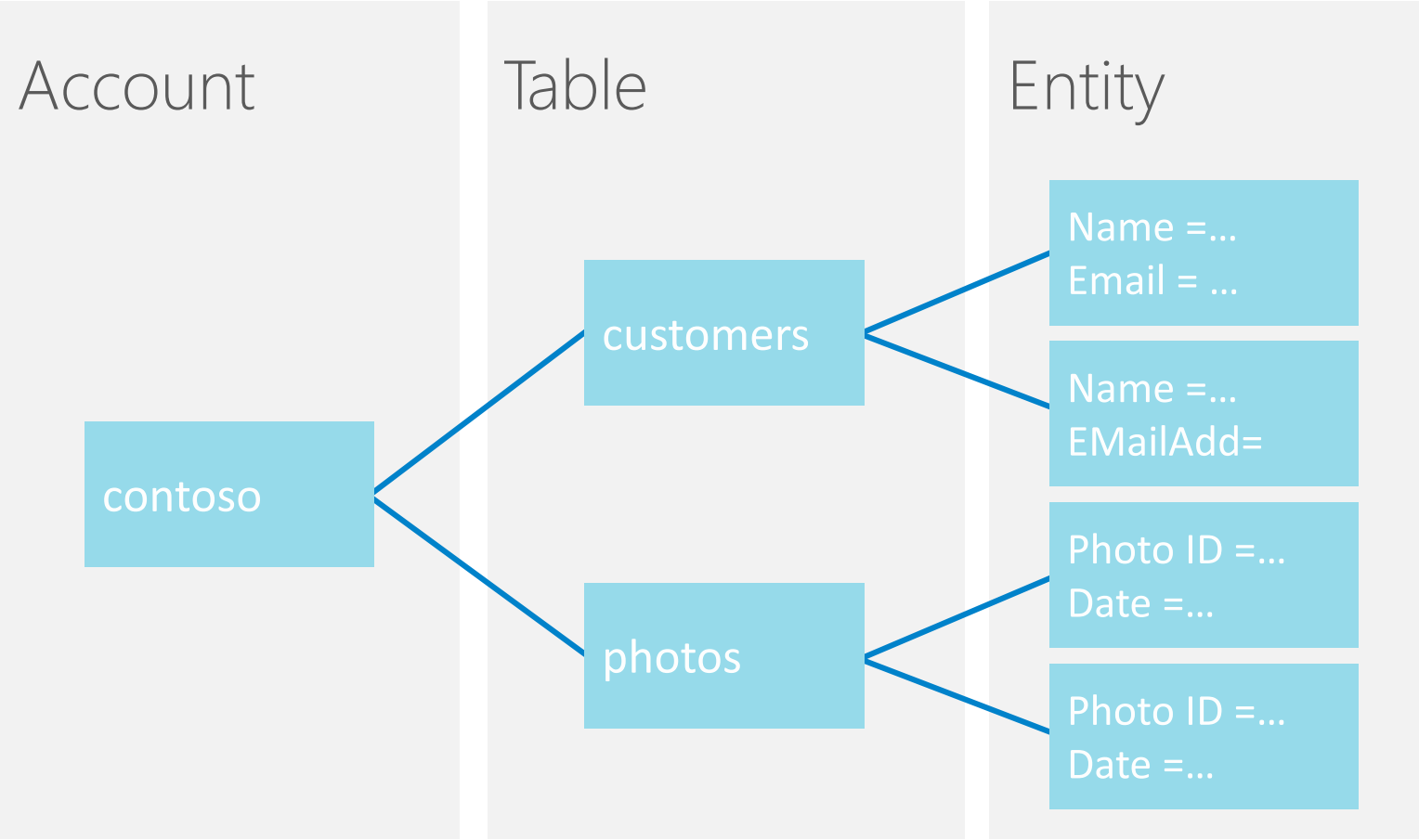
- Fine grain access rights to blobs and containers
- Sign URL with storage key – permit elevated rights
- Revocation
  - Use short time periods and re-issue
  - Use container level policy that can be deleted



- Two broad approaches
  - Ad-hoc
  - Policy based

Tables

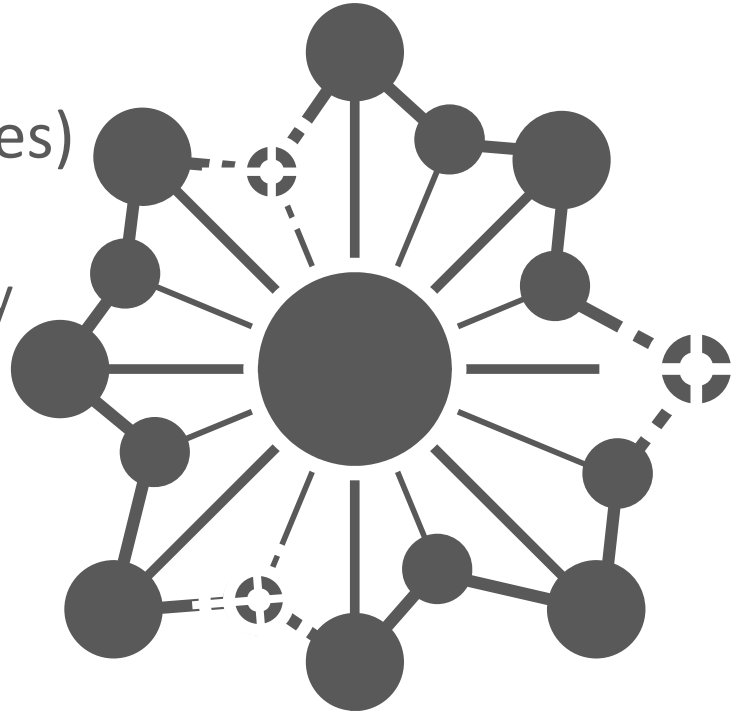
Table Storage Concepts



Entity Properties

- Entity can have up to 255 properties
  - Up to 1MB per entity

- Mandatory Properties for every entity
  - PartitionKey & RowKey (only indexed properties)
    - Uniquely identifies an entity
    - Same RowKey can be used in different PartitionKey
    - Defines the sort order
  - Timestamp
    - Optimistic Concurrency
    - Exposed as an HTTP Etag



Sample – Inserting an Entity into a Table

```
// You will need the following using statements
using Microsoft.WindowsAzure.Storage;
using Microsoft.WindowsAzure.Storage.Table;

// Create the table client.
CloudTableClient tableClient = storageAccount.CreateCloudTableClient();
CloudTable peopleTable = tableClient.GetTableReference("people");
peopleTable.CreateIfNotExists();

// Create a new customer entity.
CustomerEntity customer1 = new CustomerEntity("Harp", "Walter");
customer1.Email = "Walter@contoso.com";
customer1.PhoneNumber = "425-555-0101";

// Create an operation to add the new customer to the people table.
TableOperation insertCustomer1 = TableOperation.Insert(customer1);

// Submit the operation to the table service.
peopleTable.Execute(insertCustomer1);
```

Table Object Model

- *ITableEntity* interface –PartitionKey, RowKey, Timestamp, and Etag properties
  - Implemented by *TableEntity* and *DynamicTableEntity*

```
// This class defines one additional property of integer type,
// since it derives from TableEntity it will be automatically
// serialized and deserialized.
public class SampleEntity : TableEntity
{
    public int SampleProperty { get; set; }
}
```

Querying

- Retrieve(PartitionKey, RowKey) – retrieve single entity that satisfy the arguments
- TableQuery - lightweight object that represents a query for a given set of entities
- IQueryable (not efficient)

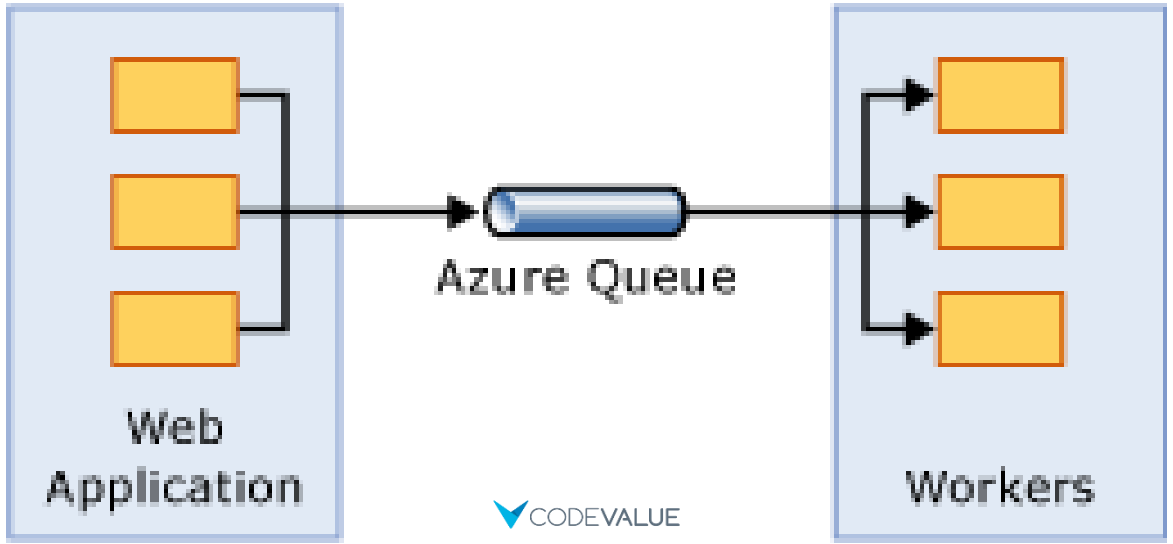
```
IQueryable<Footwear> query = table.CreateQuery<Footwear>()
            .Where(f => f.Gender == "Male" && (f.Size > 4 && f.Size < 7));

IEnumerable<Footwear> shoes = query.ToList();
```

full table scan will be performed.
because no Partition key was specified, the query will be sent to every Partition Server.

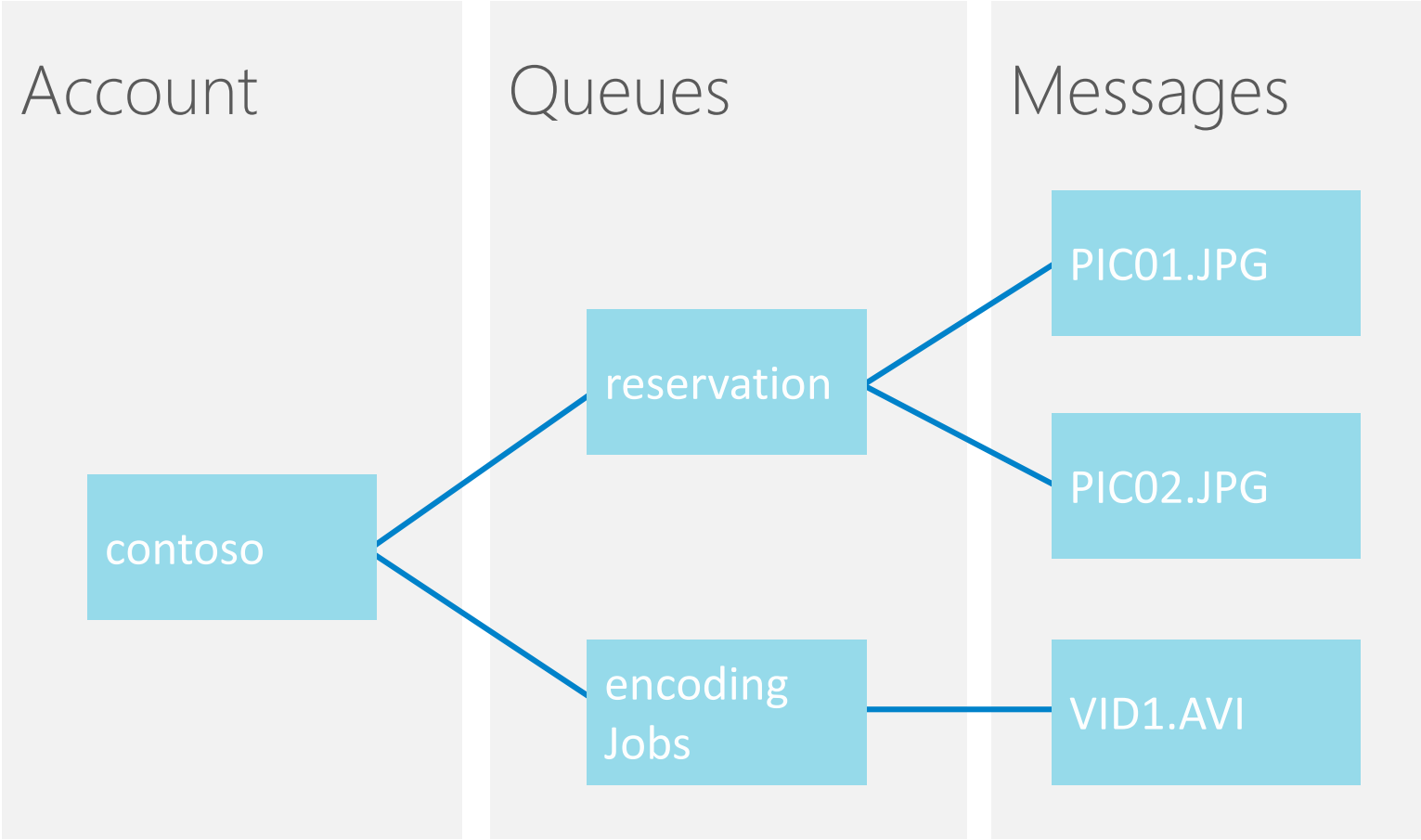
Queues

Windows Azure Queues

- Queue – First In First Out (FIFO) – Not guaranteed
- Queue are performance efficient, highly available and provide reliable message delivery
  - Simple, asynchronous work dispatch
  - Programming semantics ensure that a message can be processed at least once
- Decouple Producers and Consumers



Storage Queue Concepts



Queue Operations

- Queue
  - Create Queue
  - Delete Queue
  - List Queues
  - Get/Set Queue Metadata
- Messages
  - Add Message (i.e. Enqueue Message)
  - Get Message(s) (i.e. Dequeue Message)
  - Peek Message(s)
  - Delete Message

Queue API

```
// Retrieve storage account from connection string
CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
    CloudConfigurationManager.GetSetting("StorageConnectionString"));

// Create the queue client
CloudQueueClient queueClient = storageAccount.CreateCloudQueueClient();

// Retrieve a reference to a queue
CloudQueue queue = queueClient.GetQueueReference("myqueue");

// Create the queue if it doesn't already exist
queue.CreateIfNotExists();

// Create a message and add it to the queue.
CloudQueueMessage message = new CloudQueueMessage("Hello, World");
queue.AddMessage(message);
```

Queue API

```
// Peek at the next message
CloudQueueMessage peekedMessage = queue.PeekMessage();



// Get the next message
CloudQueueMessage retrievedMessage = queue.GetMessage();
//Process the message in less than 30 seconds, and then delete the message
queue.DeleteMessage(retrievedMessage);


// Get the message from the queue and update the message contents.
CloudQueueMessage message = queue.GetMessage();
message.SetMessageContent("Updated contents.") ;
queue.UpdateMessage(message,
       TimeSpan.FromSeconds(0.0), // Make it visible immediately.
       MessageUpdateFields.Content | MessageUpdateFields.Visibility);
```

Message Visibility

- By default, after dequeuing, messages are invisible for 30 seconds
- While invisible, no other consumer can dequeue the message
- You can set the visibility-timeout when getting the message from the queue

```
GetMessageAsync                    Task<CloudQueueMessage>
(TimeSpan? visibilityTimeout, QueueRequestOptions options, OperationContext operationContext):
Task<CloudQueueMessage>
```

- You can extend the visibility-timeout by executing the `UpdateMessageAsync` method

```
UpdateMessageAsync                 Task
(CloudQueueMessage message, TimeSpan visibilityTimeout, MessageUpdateFields updateFields):Task
```

- Call the `DeleteMessageAsync` method to remove the message from the queue
- Use the `DequeueCount` property to validate the amount of times the message was dequeued

Poison Messages

- Message can cause the consumer to crash
- find “poison” messages when dequeuing by examining the DequeueCount property of the message.
- If DequeueCount is above a given threshold it is a potential “poison” message
- Two options
  1. Delete the message
  2. Store in Poison Queue/Table

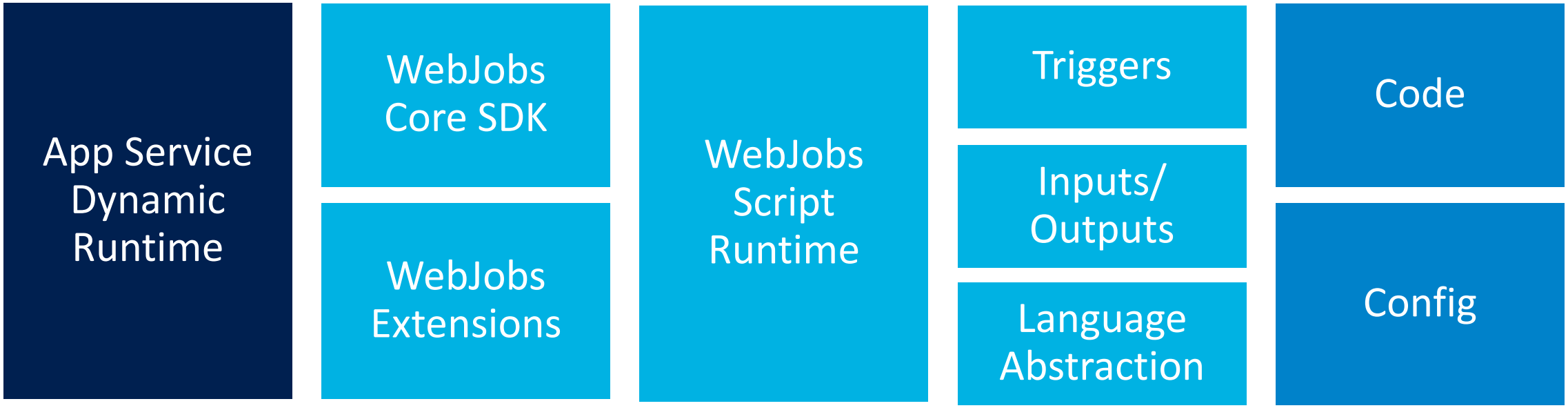
Azure Functions

Azure Functions

- Azure Functions is an event driven, compute-on-demand experience
- Azure Functions scale based on demand and you pay only for the resources you consume.
- Azure Functions can be built with .Net, Java, Node.js, PHP, and Python.
- The runtime, otherwise known as the script host, is the underlying WebJobs SDK host which listens for events, gathers and sends data, and ultimately runs your code.

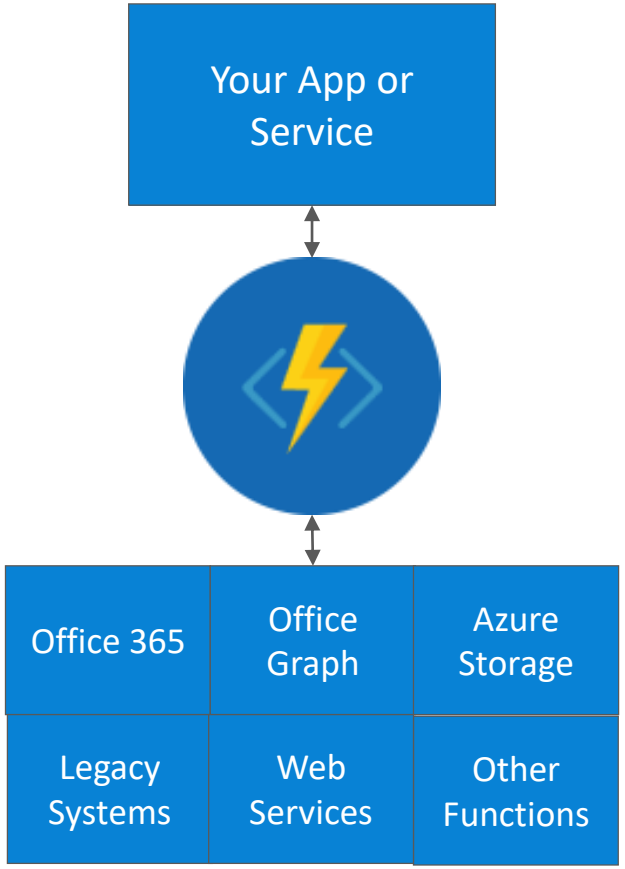
Azure Functions architecture

- Azure Functions is built around the WebJobs SDK runtime. The WebJobs SDK makes it easy to react to events and work with data in a consistent abstracted fashion.



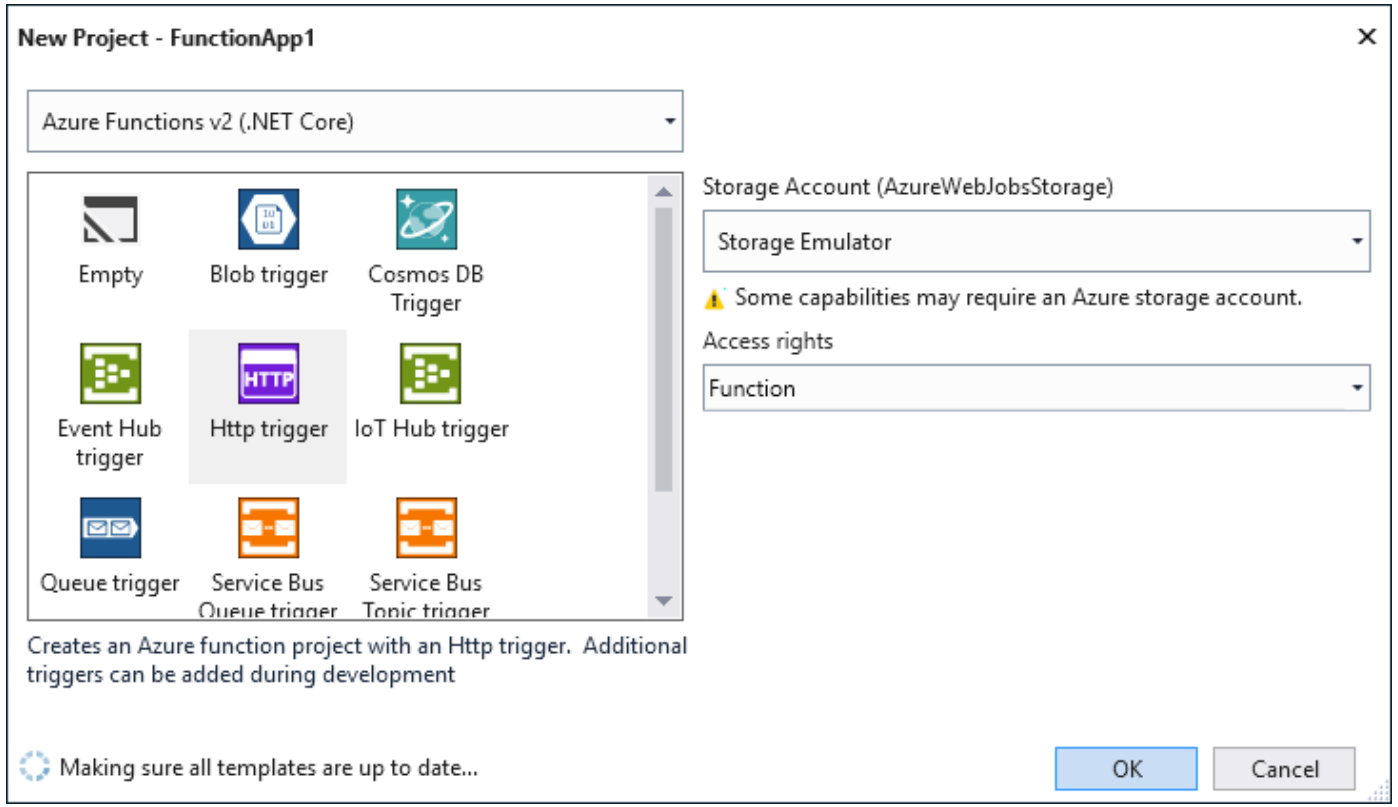
Common Scenarios

- Timer-based processing
- Azure service event processing
- SaaS event processing
- Serverless web application architectures
- Serverless mobile backends
- Real-time stream processing
- Real-time bot messaging



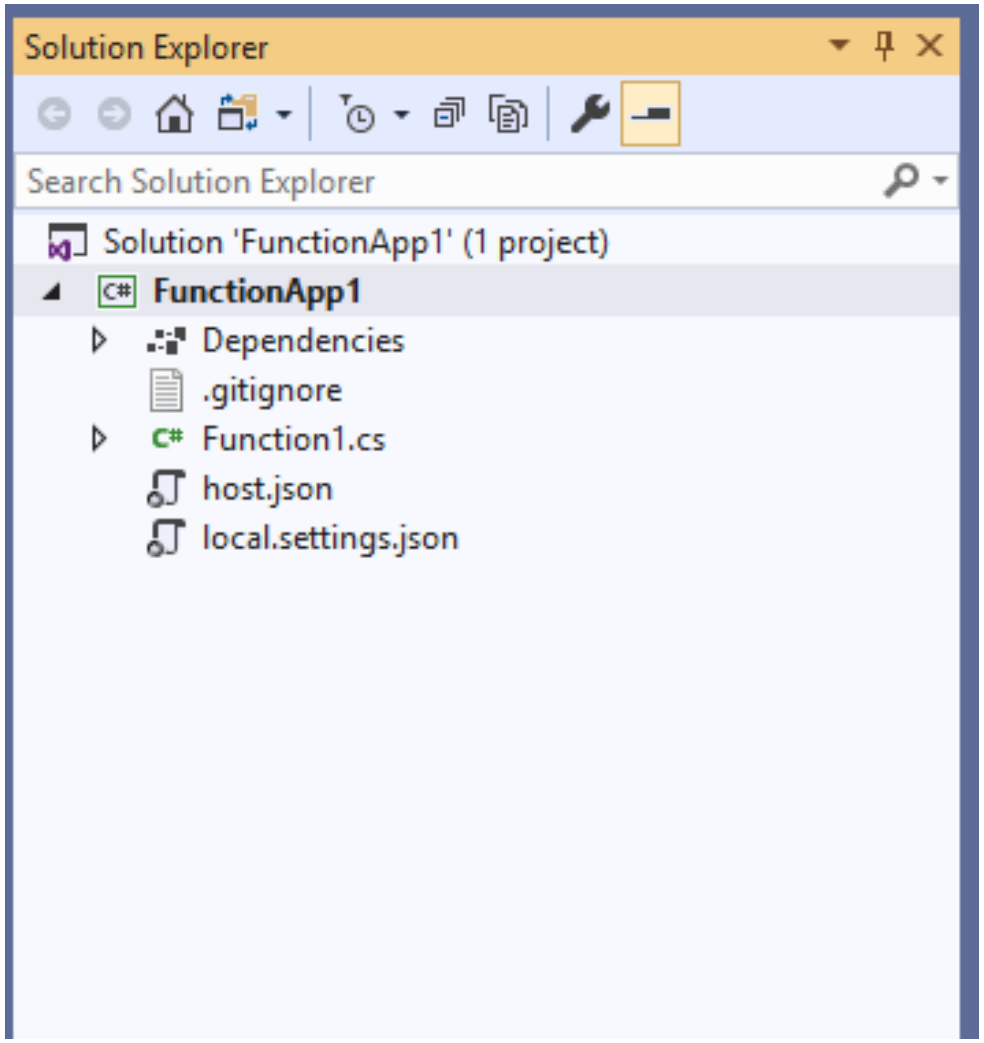
Function App Templates

Function App templates are categorized into general areas of Timer, Data Processing, and Webhook & API



- BlobTrigger
- EventHubTrigger
- Generic webhook
- GitHub webhook
- HTTPTrigger
- QueueTrigger
- ServiceBusQueueTrigger
- ServiceBusTopicTrigger
- TimerTrigger
- Blank & Experimental

Azure Functions folder structure



Function1.cs

```
[FunctionName("Function1")]
public static async Task<string> Run(
    [HttpTrigger(AuthorizationLevel.Function, "get", "post", Route = null)] HttpRequest req,
    ILogger log)
{
    log.LogInformation("C# HTTP trigger function processed a request.");

    // Login Goes Here

    return "Hello World";
}
```

- Different binding to trigger the function. For example, *HttpTrigger*.
- Regular method semantics
- Bindings - different parameters that can be injected to allow easy use of other azure services, i.e. SignalR bindings.

Logging

- To log output to your streaming logs in C#, you can include a TraceWriter typed argument. We recommend that you name it **log** or **logger**. It's recommend to avoid using Console.Write in Azure Functions.

```
[FunctionName("Function1")]
public static async Task<string> Run(
    [HttpTrigger(AuthorizationLevel.Function, "get", "post", Route = null)] HttpRequest req,
    ILogger log)
{
    log.LogInformation("C# HTTP trigger function processed a request.");

    // Login Goes Here

    return "Hello World";
}
```

Dynamic tier pricing

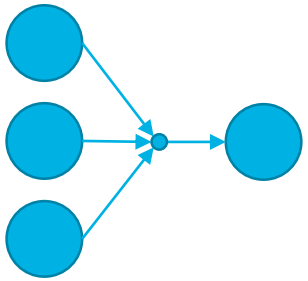
- Pay per execution model - two meters, three units
- Number of executions
- Duration of execution x reserved memory

Advanced Messaging

Communication Patterns

Synchronous Request-Reply

- A.K.A RPC – Remote Procedure Call
- Client synchronously wait for the server response
- Connection remains open -> Increase load on server
- Sent message is not durable

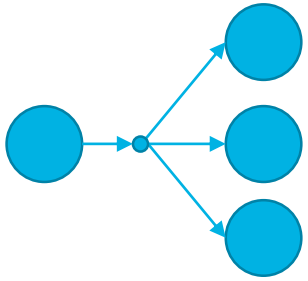


Fan in

- Server receives messages asynchronously from multiple producers
- Decoupling of client and server

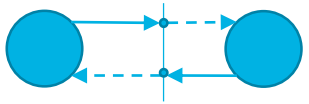
Fire and Forget

- After the server acknowledge, client continues without waiting for response (not even for operation completion)
- Sent message is not durable



Fan out

- A.K.A Publish-Subscribe (PubSub)
- The producer broadcasts a message
- Decoupling of client and server

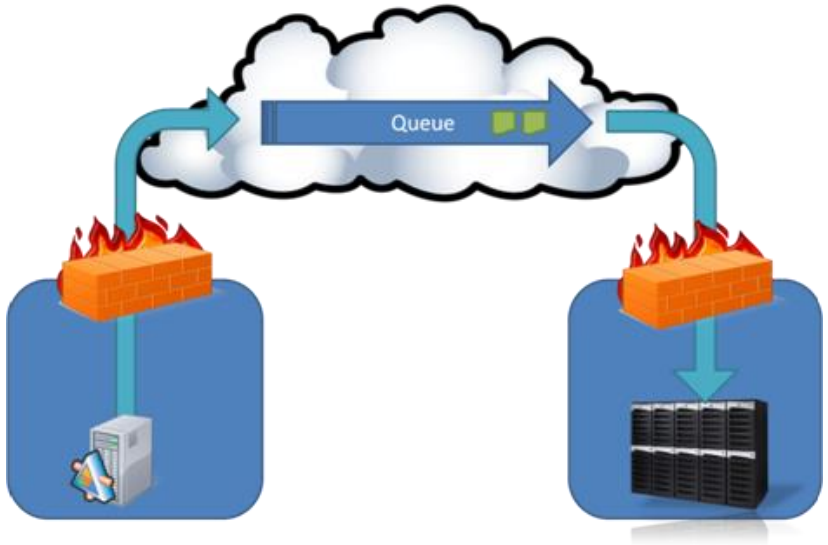


Asynchronous Request-Reply

- Decoupling of client and server
- The server asynchronously process the message and post a response
- The client asynchronously process the response

Service Bus/Queue Scenarios

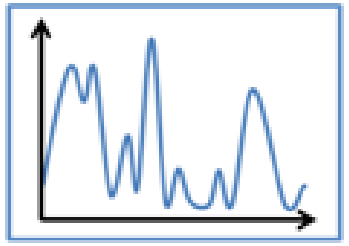
- **Secure Network Traversal**

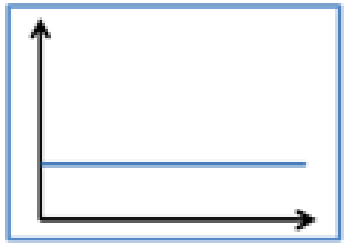


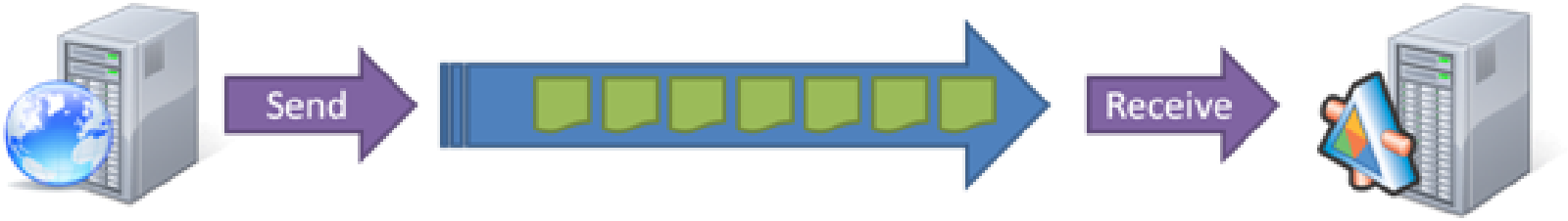
http://www.cloudcasts.net/devguide

Service Bus/Queue Scenarios

- **Load Leveling**



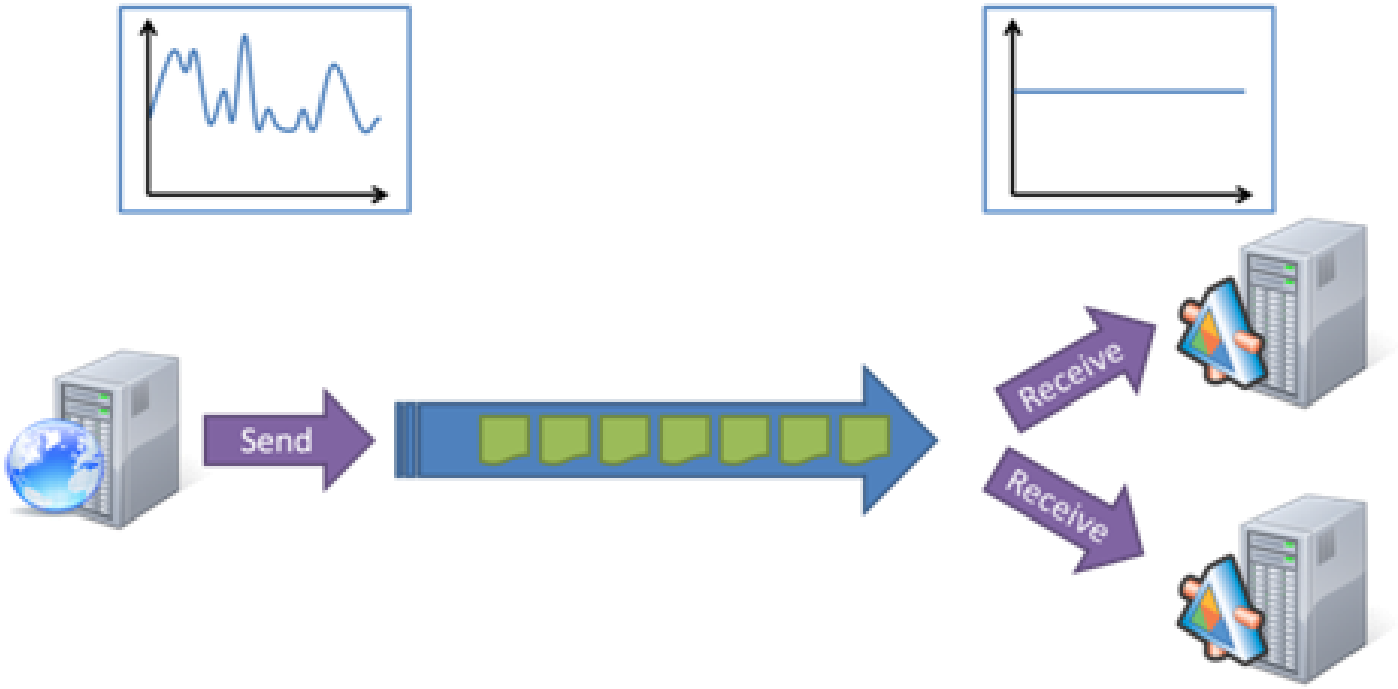




http://www.cloudcasts.net/devguide

Service Bus/Queue Scenarios

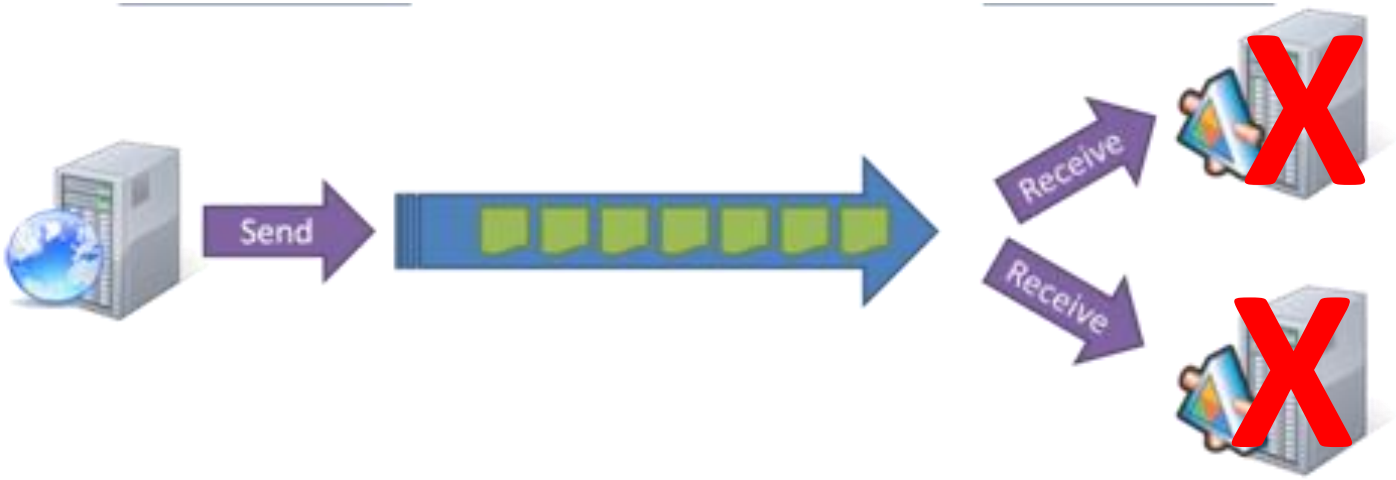
- **Load Balancing**



http://www.cloudcasts.net/devguide

Service Bus/Queue Scenarios

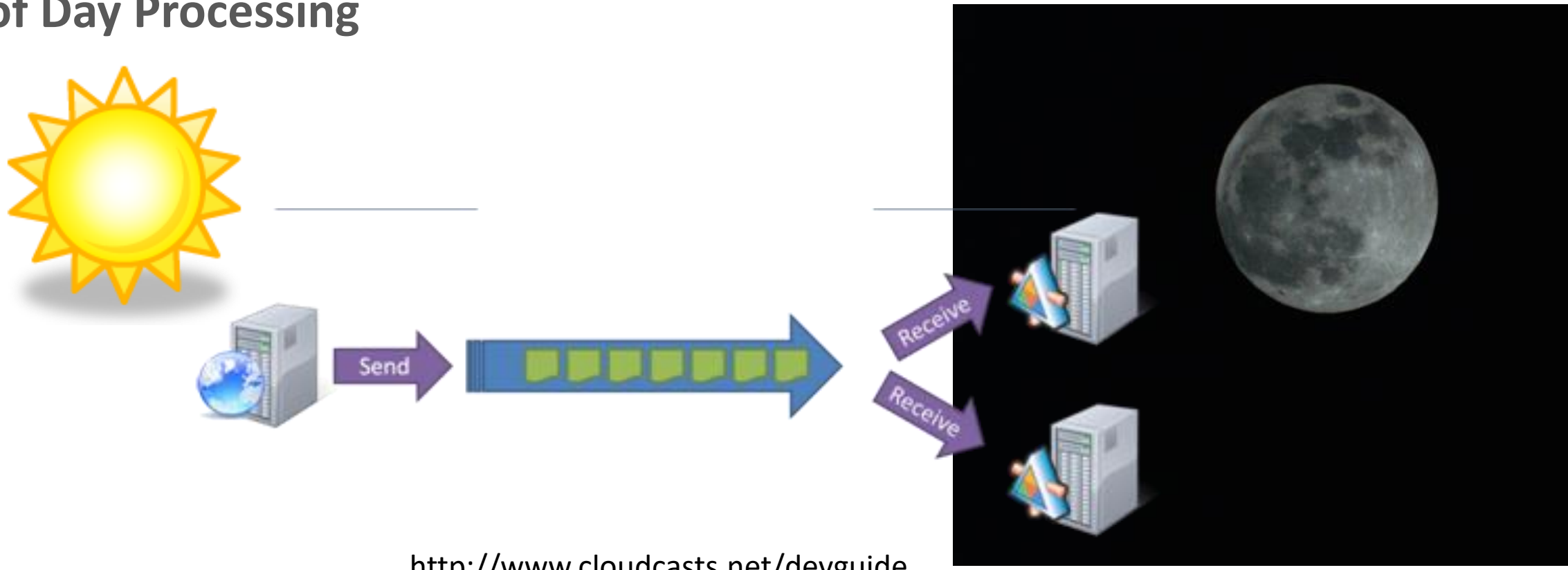
- **Resilience against Service Failure**



http://www.cloudcasts.net/devguide

Service Bus/Queue Scenarios

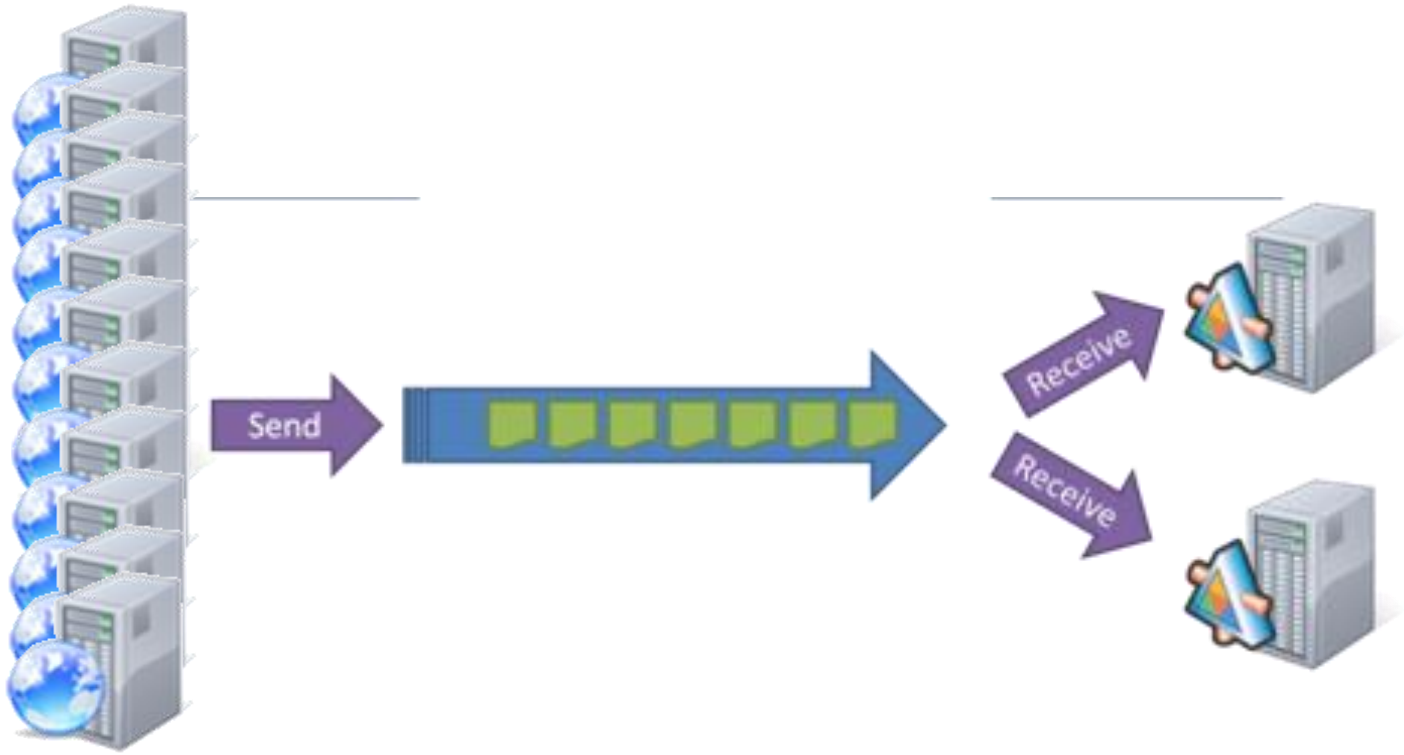
- **End of Day Processing**



http://www.cloudcasts.net/devguide

Service Bus/Queue Scenarios

- **Hyper scale data ingress (Event Hub)**

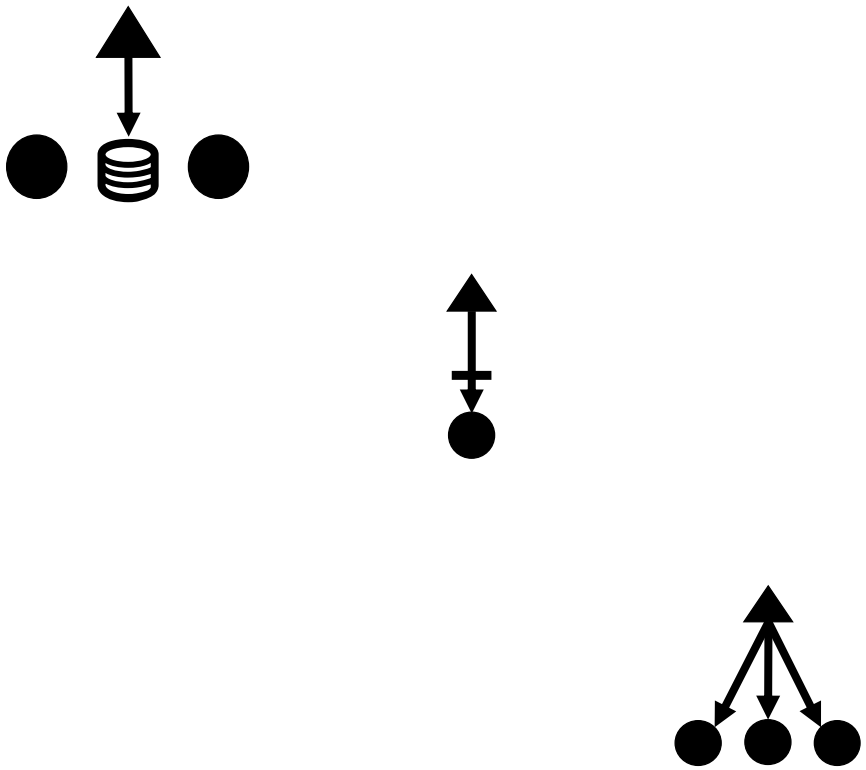


Azure Service Bus

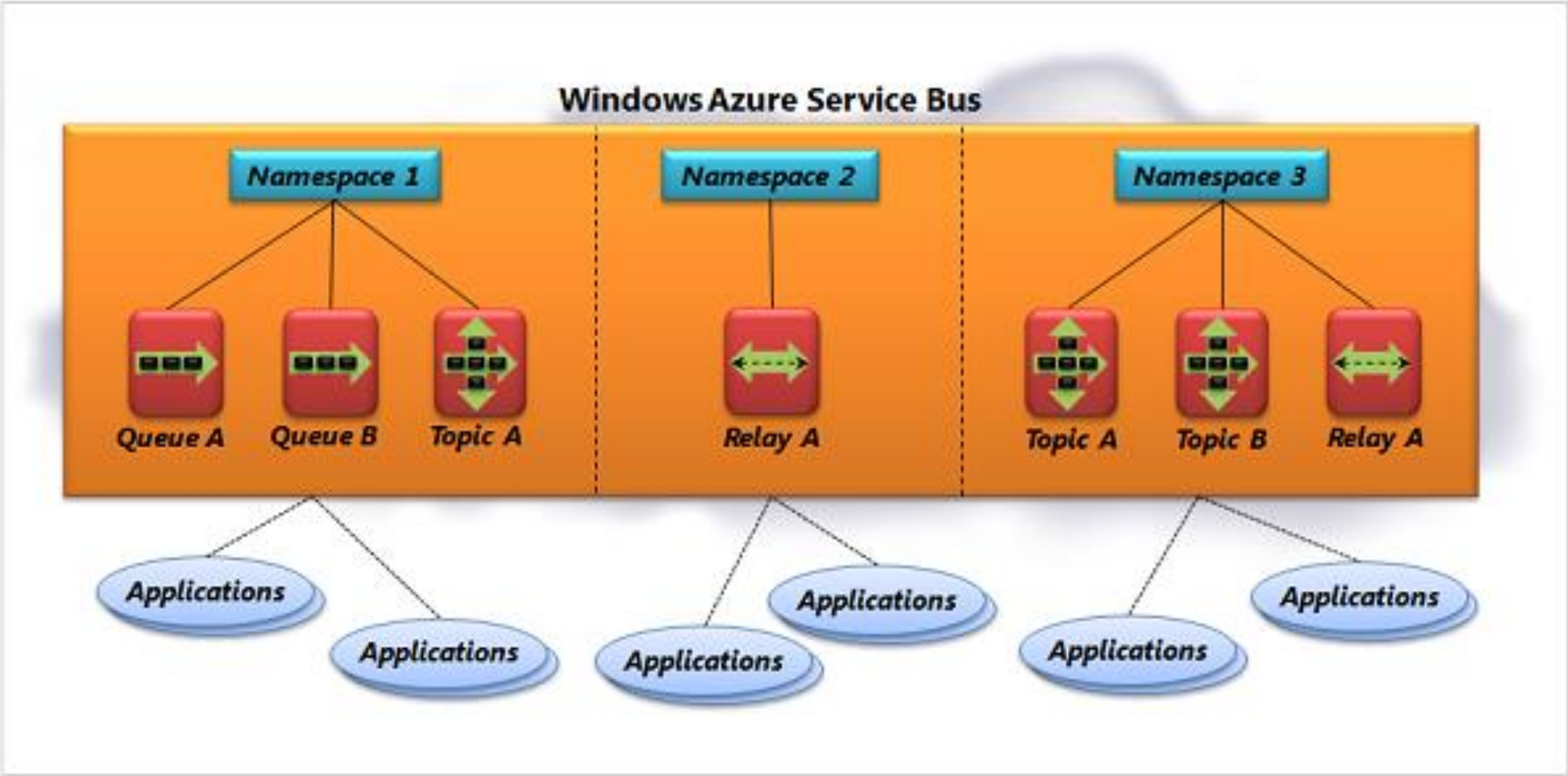
Communication Patterns

Azure Service Bus

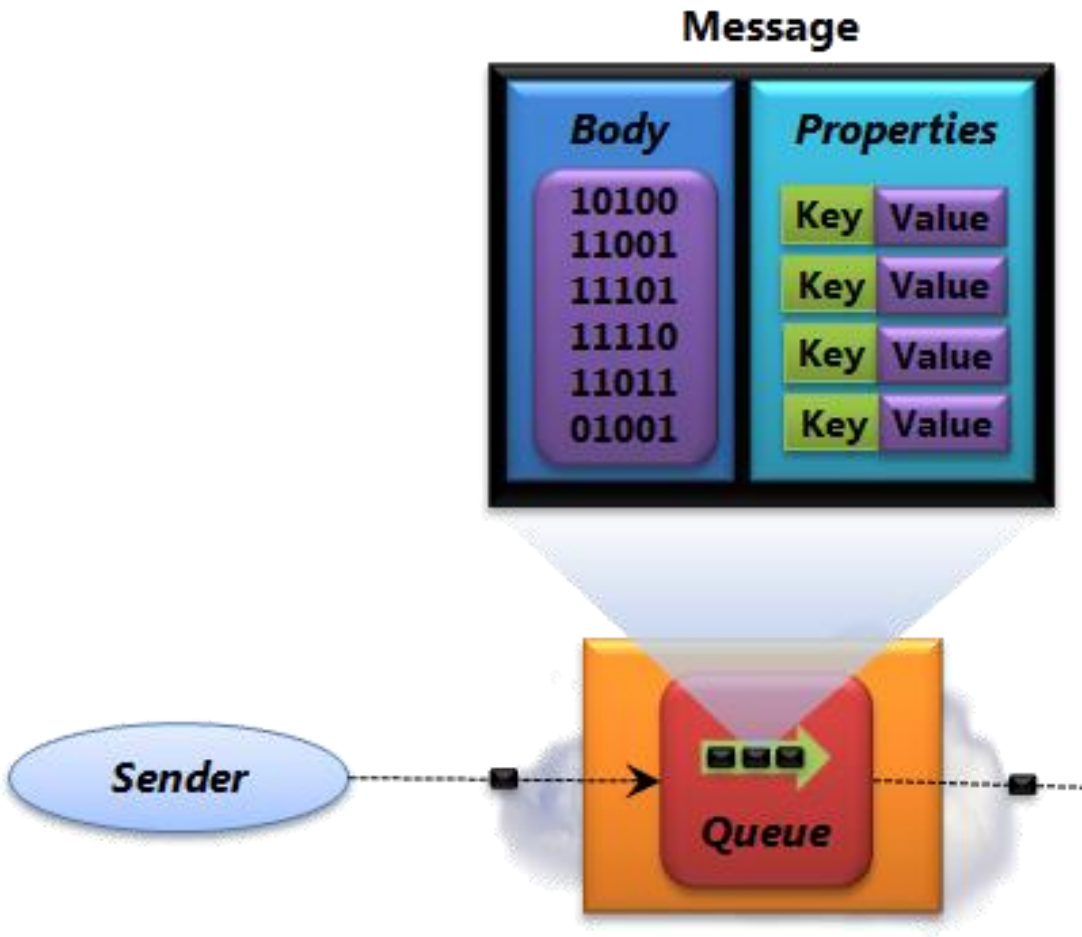
- Messaging
  - Queuing
  - Topics (Pub/Sub)
  - Reliable Transfer
- Connectivity
  - Service Relay
  - Protocol tunneling
- Hyper scale data ingestion
- Notification Hub
  - Scalable Push notifications
  - Multi platform



Service Bus Messaging



Service Bus Messaging - Queue



two parts:

1. key/value properties
2. binary message body.

Example: a sales application sends message

<u>properties:</u>

**Seller="Ava"*

**Amount=10000.*

<u>body</u> :

The sale's signed contract scanned image

Service Bus Messaging - Queue

```
Uri managementUri =
       ServiceBusEnvironment.CreateServiceUri("sb", "ServiceBusNamespace", string.Empty);
var sharedSecretTokenProvider =
   TokenProvider.CreateSharedSecretTokenProvider("[ServiceBusIssuerName]",
                                                 "[ServiceBusIssuerKey]");
var namespaceManager = new NamespaceManager(managementUri, sharedSecretTokenProvider);

var queuDescription=namespaceManager.CreateQueue("[QueuePath]");
var queueClient = QueueClient.Create(queuDescription.Path);

var someSerializableObject = new SomeSerializableType();
var brokeredMessageToSend = new BrokeredMessage(someSerializableObject);
brokeredMessageToSend.Properties["key"] = "val";
queueClient.Send(brokeredMessageToSend);

var recievedBrokerdMessage = queueClient.Receive();
var someSerializableType = recievedBrokerdMessage.GetBody<SomeSerializableType>();
var peekedBrokeredMessage = queueClient.Peek();
```

Service Bus Explorer

- https://github.com/paolosalvatori/ServiceBusExplorer

Service Bus Explorer 3.0.4

File Edit Actions View Help

Microsoft Azure

Service Bus Namespace

sb://paolosalvatori.servicebus.windows.net/
- Queues
  - bts2013
  - calculator
    - authorizationrules (0, 0)
    - notpartitioned (0, 0)
    - requestqueue (0, 0)
    - responsequeue (0, 0)
    - sessionqueue (0, 0)
    - testqueue (0, 0)
- Topics
  - bts2013
  - calculator
    - filtertopic
    - multiplerules
    - partitioned
    - requesttopic
    - stocks
    - syslabeltopic
- Event Hubs
  - baboeventhub
  - ioteventhub
- Notification Hubs
- Relays

View Queue: testqueue

Description | Authorization Rules | Metrics

Path
Relative URI: testqueue

Auto Delete On Idle

Days:	Hours:	Minutes:	Seconds:	Millisecs:
106751	2	48	5	477

Duplicate Detection History Time Window

Days:	Hours:	Minutes:	Seconds:	Millisecs:
0	0	10	0	0

Default Message Time To Live

Days:	Hours:	Minutes:	Seconds:	Millisecs:
106751	2	48	5	477

Queue Properties
Max Queue Size In GB: 1 GB
Max Delivery Count: 10
User Description:
Forward To:
Forward Dead Lettered Messages To:

Lock Duration

Days:	Hours:	Minutes:	Seconds:	Millisecs:
0	0	1	0	0

Queue Settings
- [x] Enable Batched Operations
- [x] Enable Dead Lettering On Message Expiration
- [x] Enable Partitioning
- [ ] Enable Express
- [ ] Requires Duplicate Detection
- [ ] Requires Session

Queue Information

Name	Value
Status	Active
Is ReadOnly	False
Size In Bytes	0
Created At	10/5/2015 9:2...
Accessed At	10/6/2015 7:4...
Updated At	10/5/2015 9:2...
Active Message Count	0
DeadLetter Message Count	0
Scheduled Message Count	0
Transfer Message Count	0
Transfer DL Message Count	0
Message Count	0

Get Metrics | Close Tabs | Messages | Deadletter | Refresh | Disable | Delete | Update

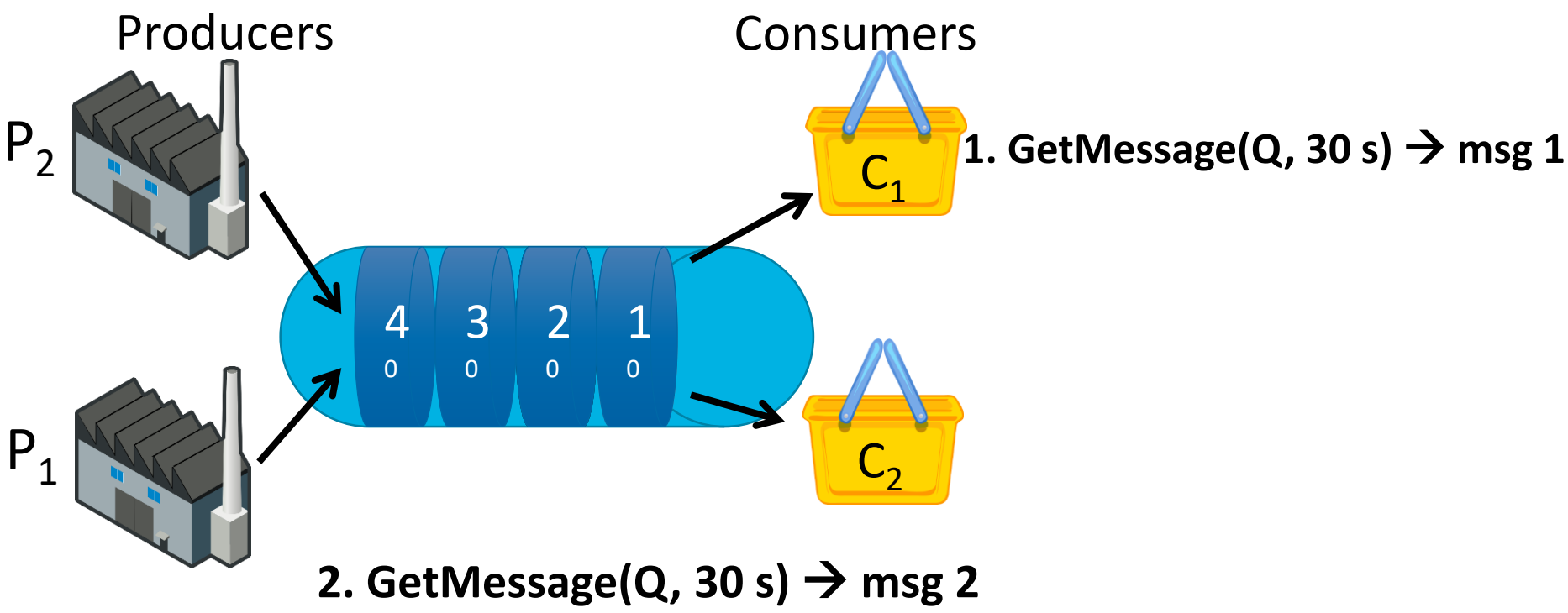
Log

```
<12:10:28> The queue authorizationrules has been successfully retrieved.
<12:10:28> The queue bts2013/requestqueue has been successfully retrieved.
<12:10:28> The queue bts2013/responsequeue has been successfully retrieved.
<12:10:28> The queue calculator/requestqueue has been successfully retrieved.
<12:10:28> The queue calculator/responsequeue has been successfully retrieved.
<12:10:28> The queue notpartitioned has been successfully retrieved.
<12:10:28> The queue requestqueue has been successfully retrieved.
<12:10:28> The queue responsequeue has been successfully retrieved.
<12:10:28> The queue sessionqueue has been successfully retrieved.
<12:10:28> The queue testqueue has been successfully retrieved.
```

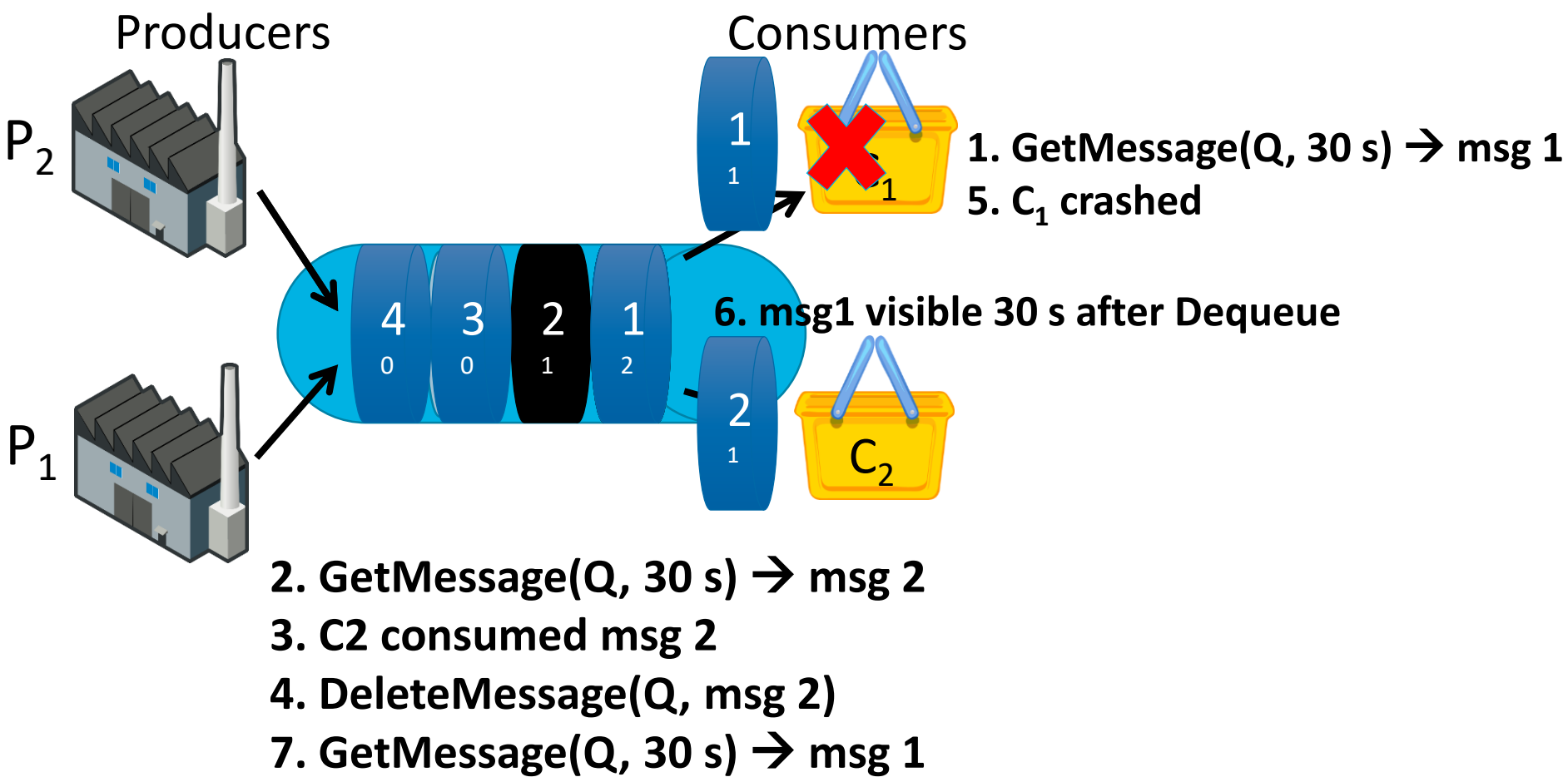
Service Bus Messaging – Queue – Event Driven

```
var eventDrivenMessagingOptions = new OnMessageOptions
{
    AutoComplete = true,
    MaxConcurrentCalls = 5
};
eventDrivenMessagingOptions.ExceptionReceived += OnExceptionReceived;
queueClient.OnMessage(OnMessageArrived, eventDrivenMessagingOptions);

private void OnMessageArrived(BrokeredMessage obj)
{
    //do something
}

private void OnExceptionReceived(object sender, ExceptionReceivedEventArgs e)
{
   //do something
}
```

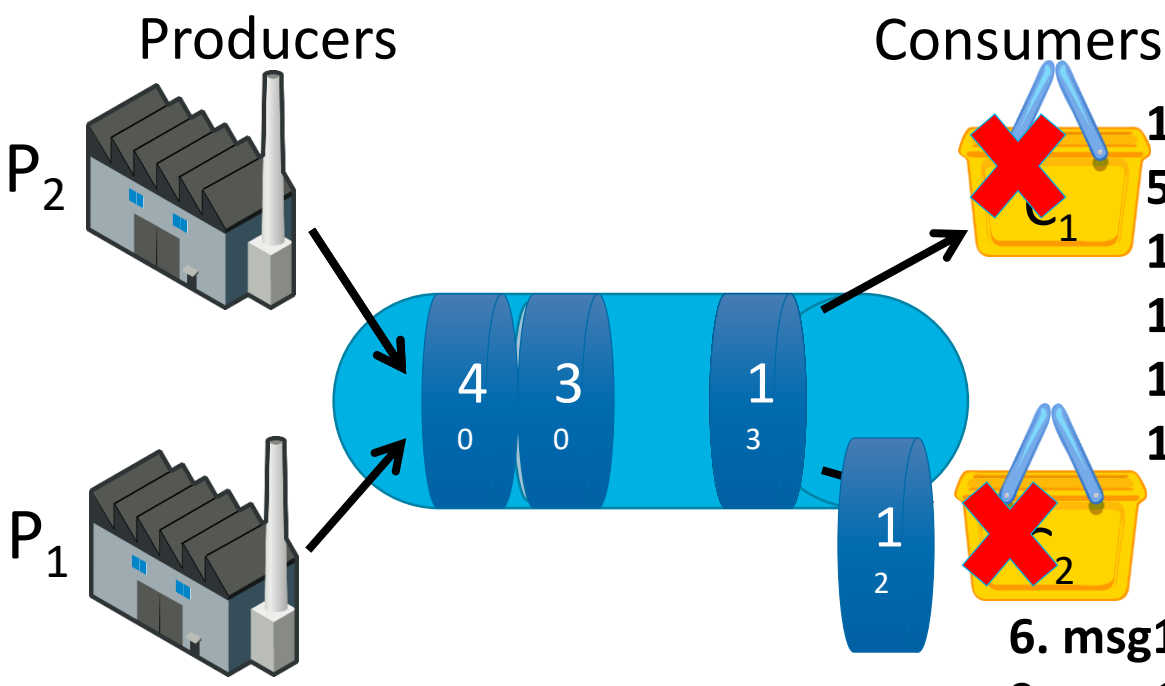
Removing Poison Messages



Removing Poison Messages



Removing Poison Messages



1. Dequeue(Q, 30 sec) → msg 1
5. C_1 crashed
10. C1 restarted
11. Dequeue(Q, 30 sec) → msg 1
12. DequeueCount > 2
13. Delete (Q, msg1)

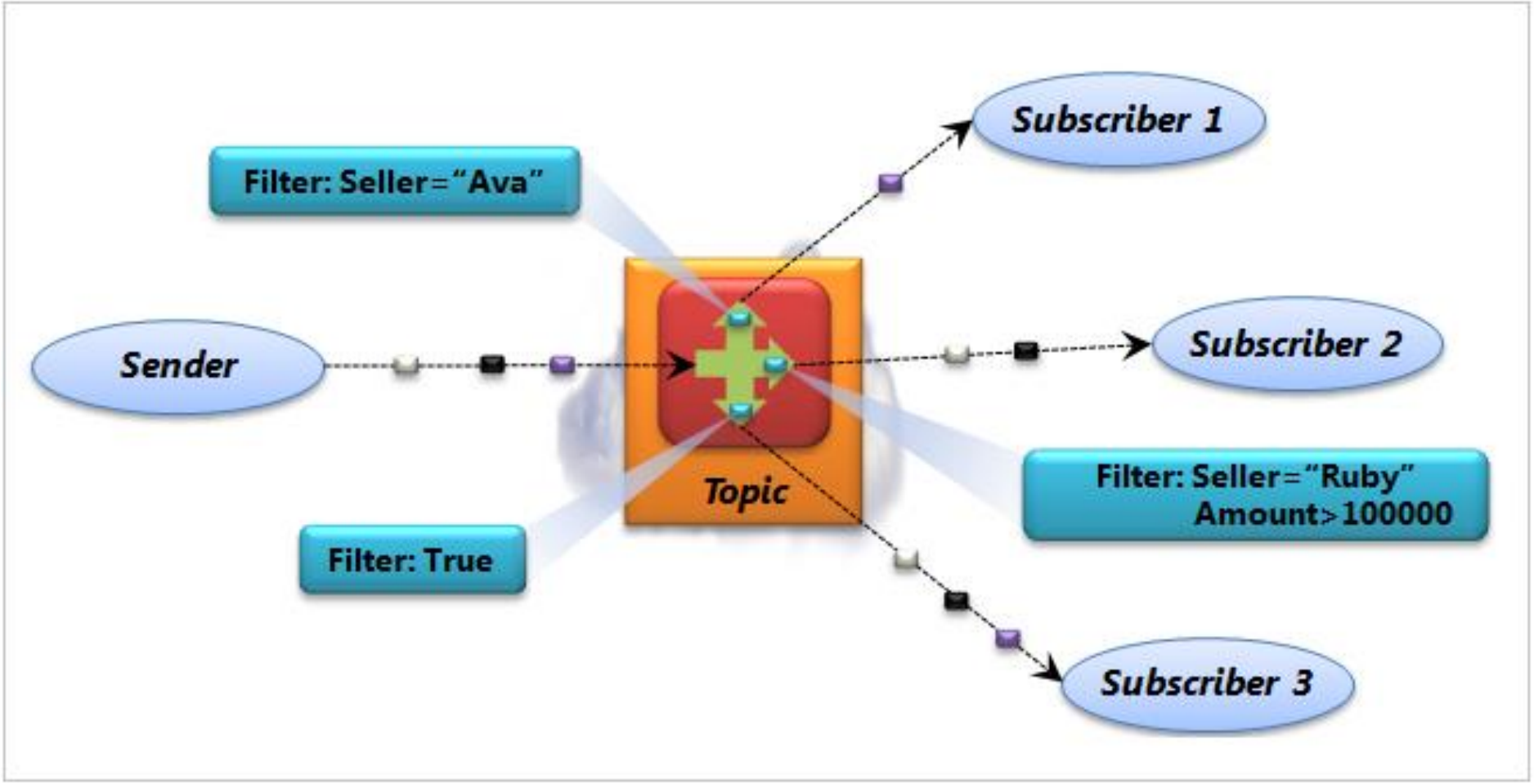
6. msg1 visible 30s after Dequeue
9. msg1 visible 30s after Dequeue

2. Dequeue(Q, 30 sec) → msg 2
3. C2 consumed msg 2
4. Delete(Q, msg 2)
7. Dequeue(Q, 30 sec) → msg 1
8. C2 crashed

Poison Messages

- Message can cause the consumer to crash
- Detecting “Poison Messages”
  - For Storage Queues - examine the DequeueCount property of the message.
    - Two options
      - Delete the message
      - Store in Poison Queue/Table
  - Azure Service bus
    - automatically done by setting the QueueDescription.MaxDeliveryCount and SubscriptionDescription.MaxDeliveryCount properties
    - Explicitily calling the DeadLetter() method

Service Bus Messaging – Topics



Service Bus Messaging – Topics

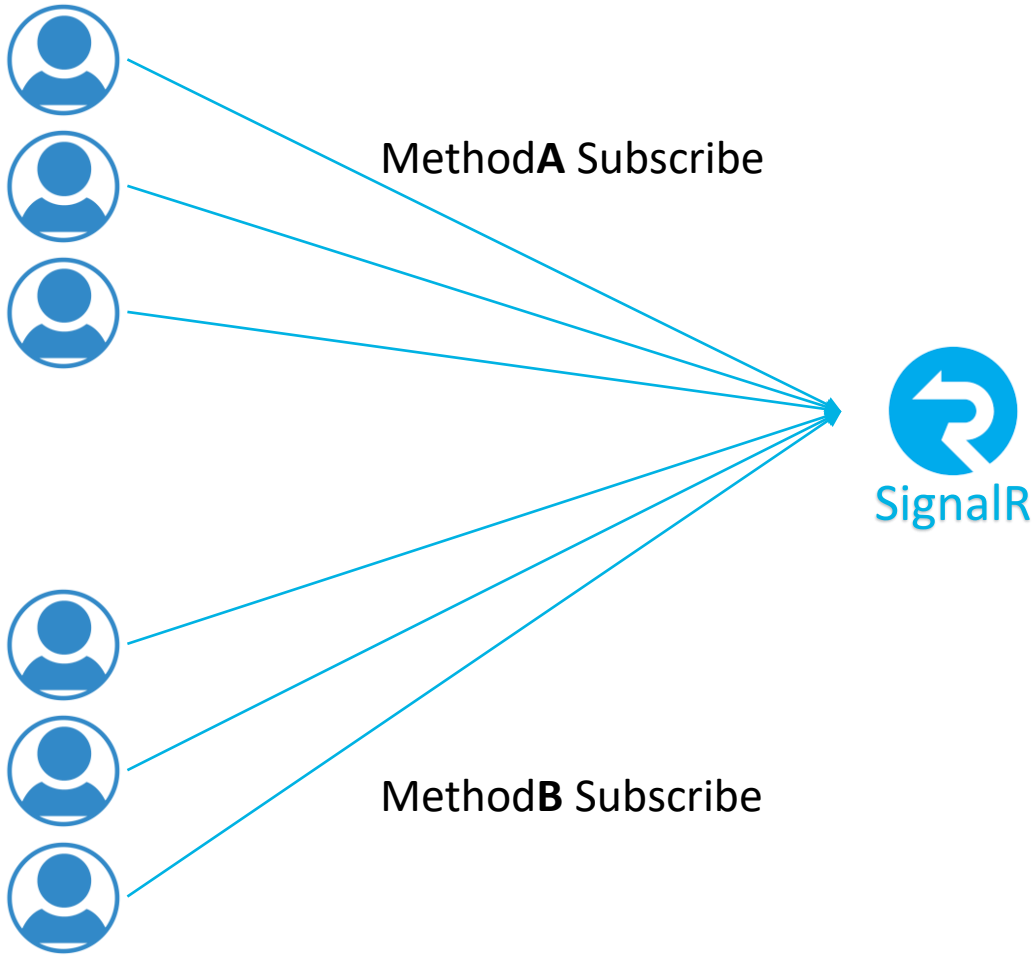
- A topic is similar in many ways to a queue.
- Topics let each receiving application create its own subscription by defining a *filter*.
- A subscriber will then see only the messages that match that filter.
- Unlike queues, however, a single message sent to a topic can be received by multiple subscribers.
- ***publish and subscribe***

SignalR Service

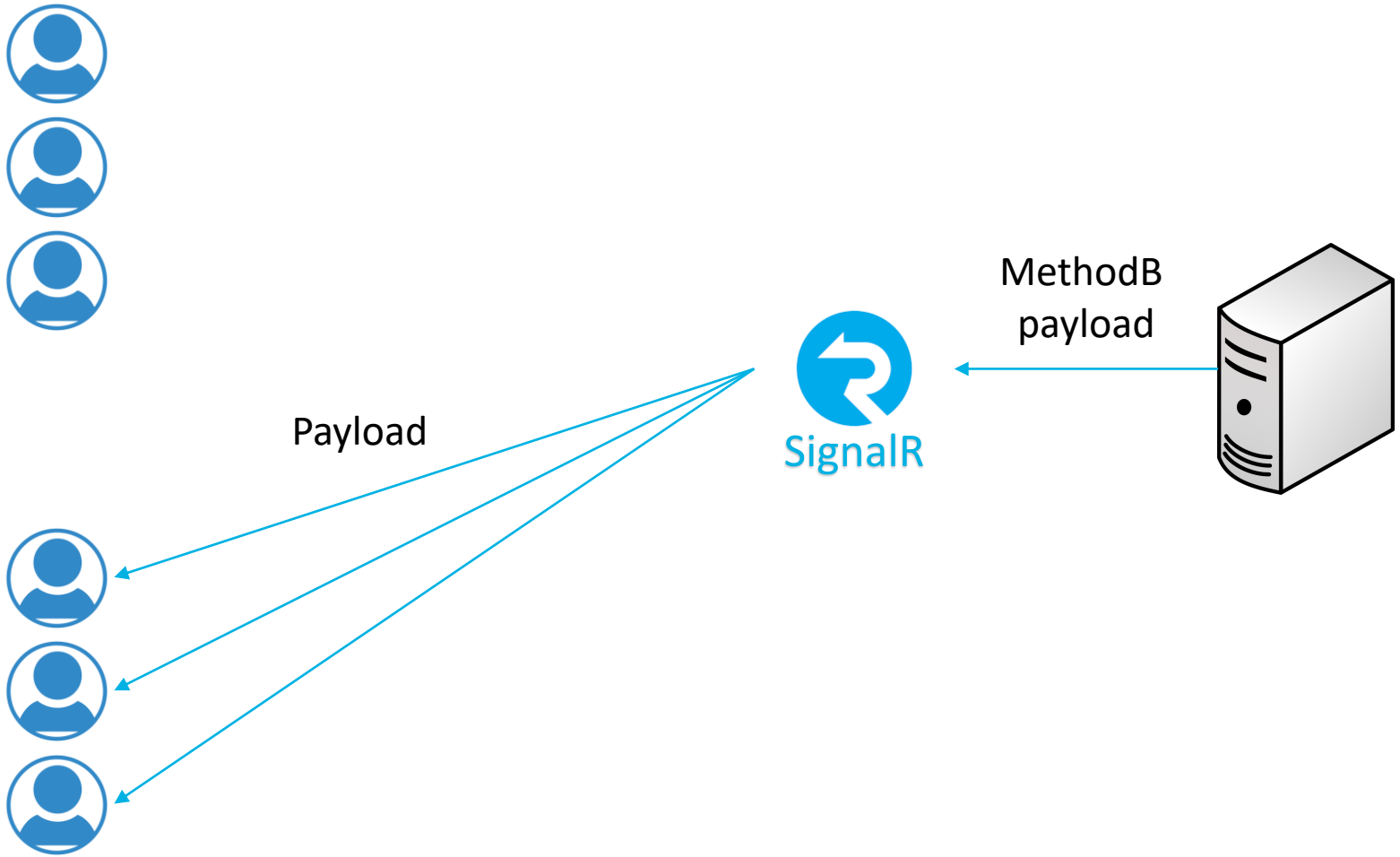
What is SignaR?

- A simple two way communication service
  - Uses web sockets to send messages
  - Falls back to regular HTTP polling if web socket is not available
- Light-weight
- Scalable
- Supports only real time communication
  - No messages storing – you snooze, you lose
- Receive messages based on the method names the server invokes
- Supports broadcasting to all connected users
- Supports direct messages to specific users/groups

Broadcasting Method calls



Broadcasting Method calls



SignalR direct users/groups calls

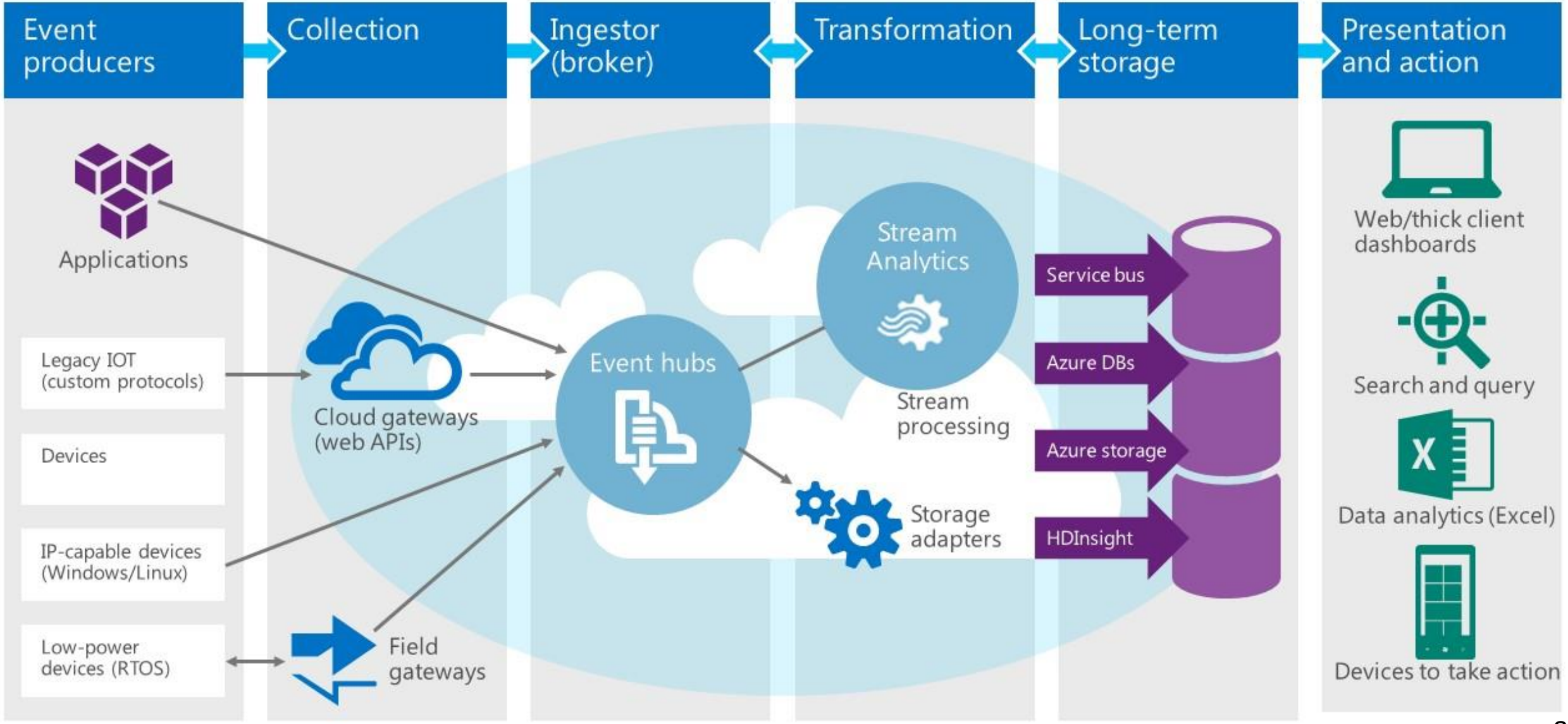
- Server can decide to add a user to a group or remove them from the group
- The messages are still sent via a method call to the clients/groups
  - Similar to broadcasting a message, but narrowing down the audience
- A disconnected user will be automatically removed from a group
  - The disconnection occurs only after a few seconds, to avoid “losing” a group user due to a temporary disconnection

Event Hubs

Event Hubs

- Highly scalable data ingress service
- Can ingest millions of events per second
- Act as the "front door" for an event pipeline
  - Once data is collected into an Event Hub, it can be transformed and stored using any real-time analytics provider or batching/storage adapters.
- Decouples the production of a stream of events from the consumption of those events
  - Event consumers can access the events on their own schedule
- Different from traditional queues
  - Journal Logging
  - Similar to Apache Kafka

Azure Based High-Throughput Ingest Architecture



Event Hubs

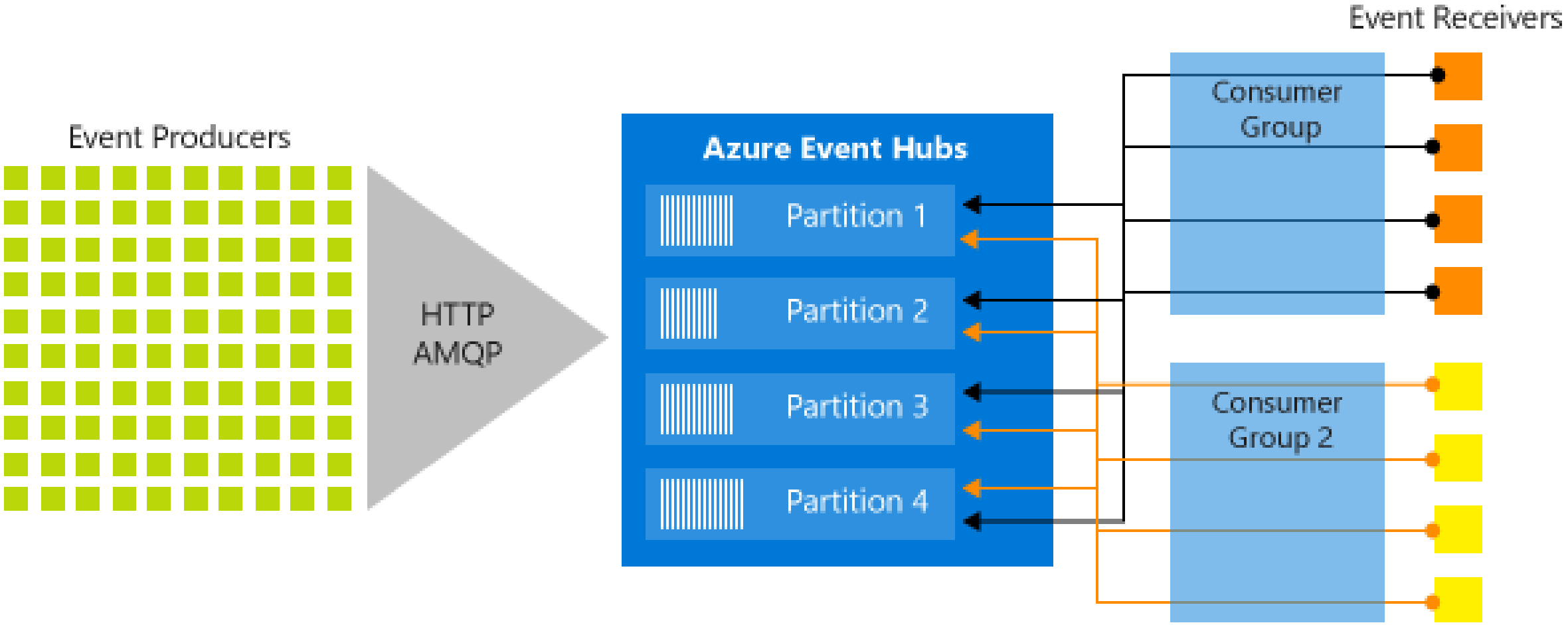
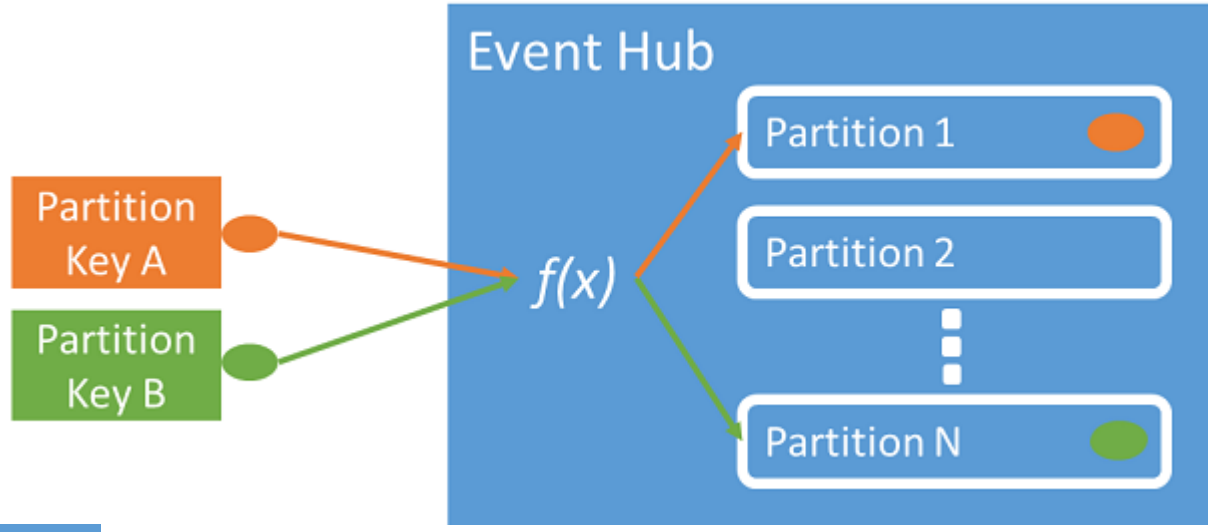
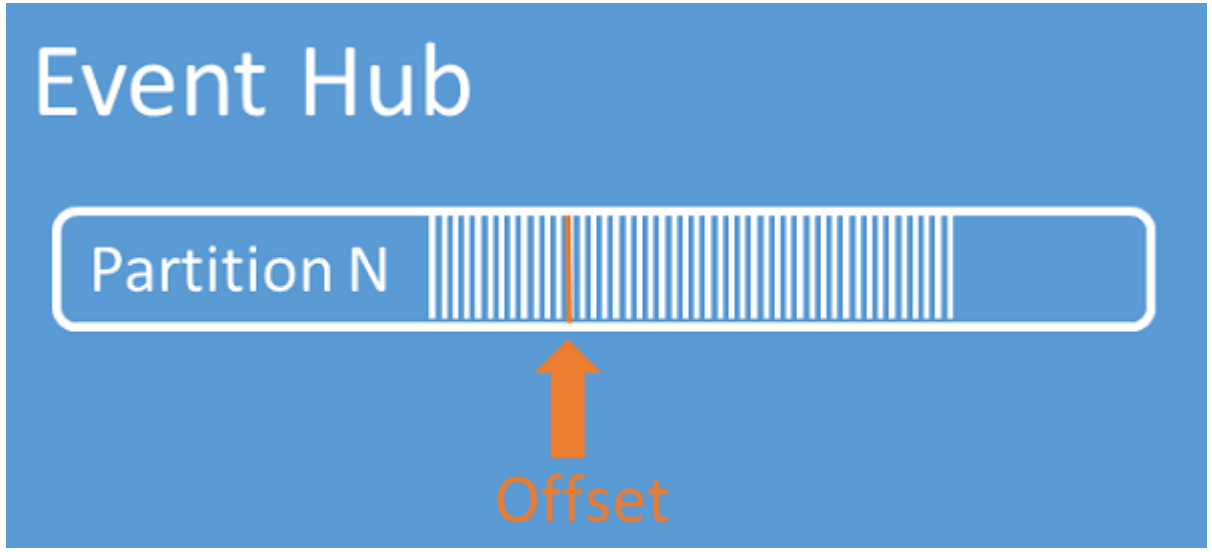


Image from https://docs.microsoft.com/en-us/azure/event-hubs/event-hubs-features

Event Hub Partition





Basic Programming model

Creating

```
var manager = new Microsoft.ServiceBus.NamespaceManager("mynamespace.servicebus.windows.net");
var description = manager.CreateEventHub("MyEventHub");
var client = EventHubClient.Create(description.Path);
```

Sending

```
var partitionedSender = client.CreatePartitionedSender();
var partitionedSender = client.CreatePartitionedSender(description.PartitionIds[0]);
```

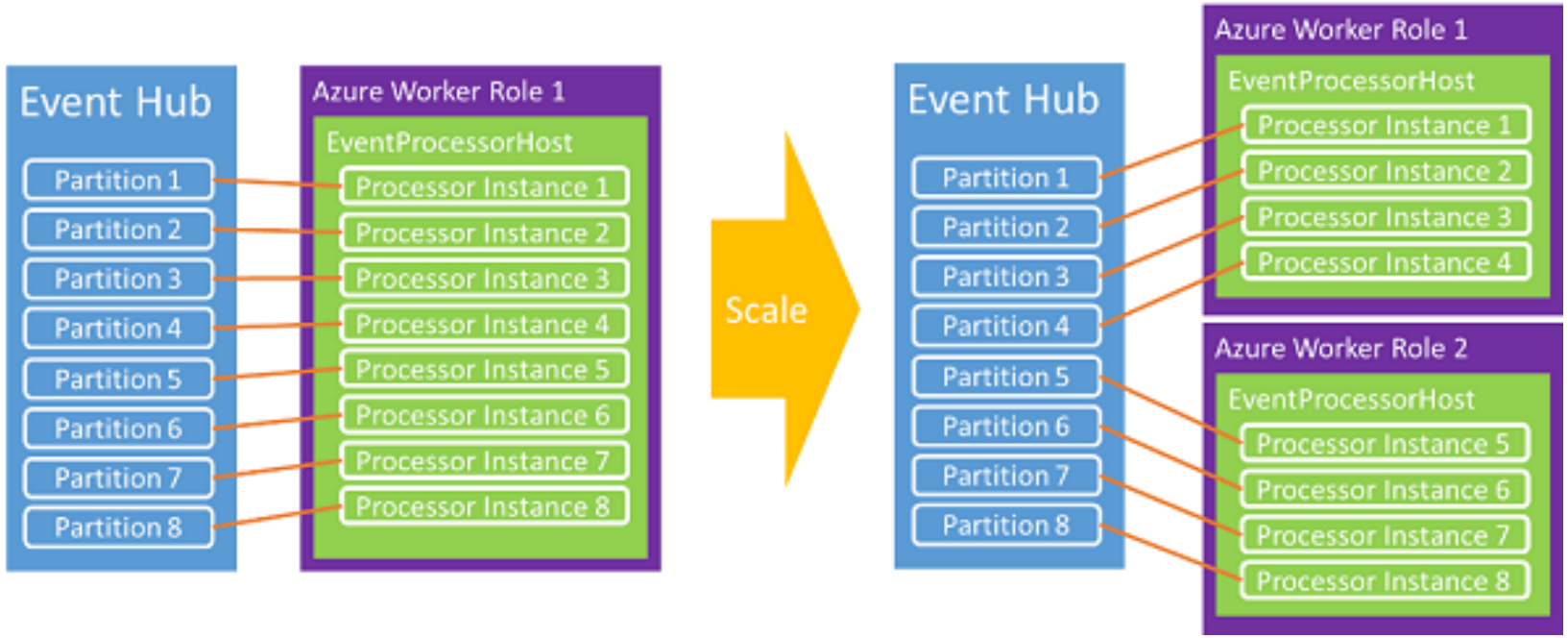
Receving

```
EventHubConsumerGroup group = client.GetDefaultConsumerGroup();
var receiver = group.CreateReceiver(client.GetRuntimeInformation().PartitionIds[0]);
```

Event Processor Host

- EventProcessorHost provides a thread-safe, multi-process, safe runtime environment for event processor with checkpointing and partition lease
- Reside in `Microsoft Azure Service Bus Event Hub – EventProcessorHost` nuget package
- How it works:
  - Implement IEventProcessor with the logic of your message-processing:
  - Use EventProcessorHost .RegisterEventProcessorAsync to register IEventProcessor
  - The host will attempt to acquire a lease on every partition in the event hub using a "greedy" algorithm.
  - As new nodes,(worker instances), come online, they place lease reservations and over time the load shifts between nodes as each attempts to acquire more leases.

Event Processor Host



- The EventProcessorHost class also implements an Azure storage-based checkpointing mechanism.
- This mechanism stores the offset on a per partition basis, so that each consumer can determine what the last checkpoint from the previous consumer was.

Event Processor Host - IEventProcessor

```
class SimpleEventProcessor : IEventProcessor
{
    async Task IEventProcessor.CloseAsync(PartitionContext context, CloseReason reason)
    {
        Console.WriteLine("Processor Shutting Down. Partition '{context.Lease.PartitionId}'."
        if (reason == CloseReason.Shutdown){
            await context.CheckpointAsync();
        }
    }
    Task IEventProcessor.OpenAsync(PartitionContext context){
        Console.WriteLine($"SimpleEventProcessor initialized. Partition:'{context.Lease.PartitionId}’”);
        return Task.CompletedTask;
    }
    async Task IEventProcessor.ProcessEventsAsync(PartitionContext context,
                                                  IEnumerable<EventData> messages){
        foreach (EventData eventData in messages) {
            string data = Encoding.UTF8.GetString(eventData.GetBytes());
            Console.WriteLine($"Message received.
                Partition: '{context.Lease.PartitionId}', Data: '{data}'"));
        }
    }
}
```

Event Processor Host – Register IEventProcessor

```
string eventProcessorHostName = Guid.NewGuid().ToString();
EventProcessorHost eventProcessorHost =
       new EventProcessorHost(eventProcessorHostName,
                              eventHubName,
                              EventHubConsumerGroup.DefaultGroupName,
                              eventHubConnectionString,
                              storageConnectionString);
Console.WriteLine("Registering EventProcessor...");
var options = new EventProcessorOptions();
options.ExceptionReceived += (sender, e) => { Console.WriteLine(e.Exception); };
eventProcessorHost.RegisterEventProcessorAsync<SimpleEventProcessor>(options).Wait();

Console.WriteLine("Receiving. Press enter key to stop worker.");
Console.ReadLine();
eventProcessorHost.UnregisterEventProcessorAsync().Wait();
```

Summary

Summary

- The world is moving to serverless architecture
  - Saves money (Most of the time)
  - Saves time
  - Easy to start using
- The amount of serverless services is rapidly growing
- Messaging is the key for communicating between parts
- Azure functions is your main glue for calculations