

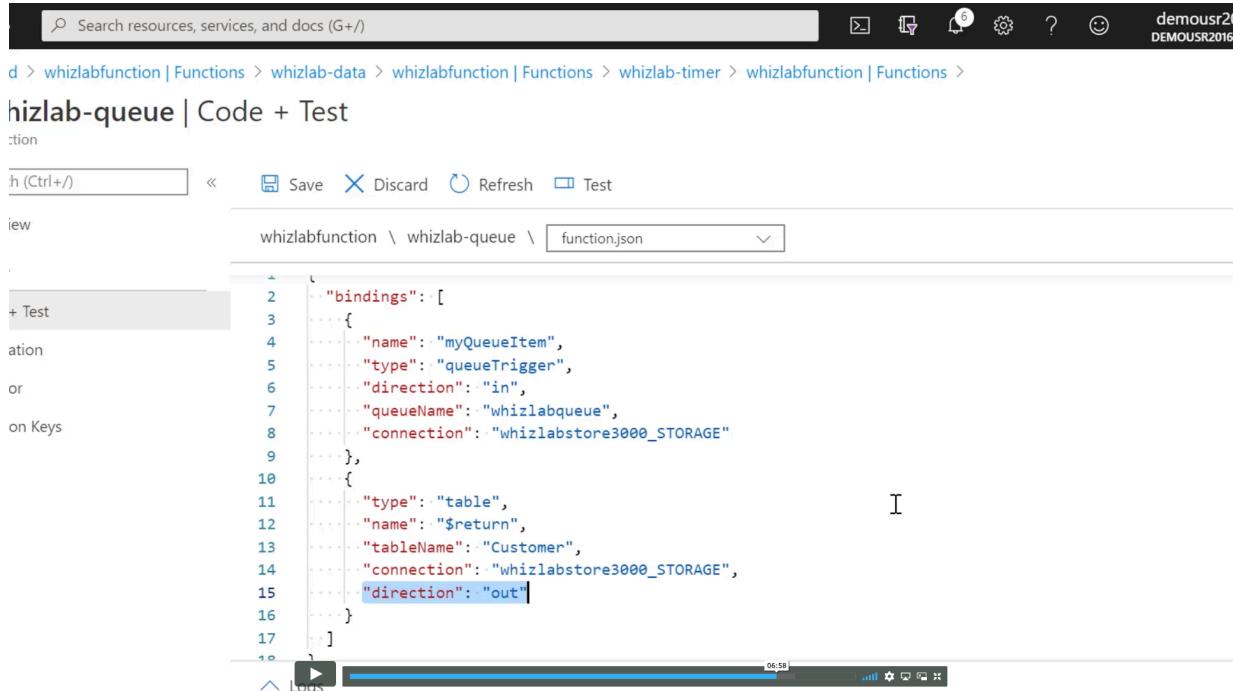
# 2020.07.12 Whizlabs Microsoft Azure Exam AZ-204 Certification

<https://www.whizlabs.com/learn/course/microsoft-azure-az-204>

Originally started on Saturday, July 11th, 2020

IMPORTANT: This exam uses C# for all examples

## Azure Functions



The screenshot shows the Azure Functions code editor interface. The top navigation bar includes a search bar, icons for file operations, and user information ('demousr2 DEMOUSR2016'). Below the navigation is a breadcrumb trail: 'd > whizlabfunction | Functions > whizlab-data > whizlabfunction | Functions > whizlab-timer > whizlabfunction | Functions > hizlab-queue | Code + Test'. The main area shows a code editor with a JSON file named 'function.json'. The code defines two bindings: one for a queue trigger named 'myQueueItem' and one for a table output named '\$return'. Both are connected to the 'whizlabstore3000\_STORAGE' connection.

```
2   "bindings": [
3     {
4       "name": "myQueueItem",
5       "type": "queueTrigger",
6       "direction": "in",
7       "queueName": "whizlabqueue",
8       "connection": "whizlabstore3000_STORAGE"
9     },
10    {
11      "type": "table",
12      "name": "$return",
13      "tableName": "Customer",
14      "connection": "whizlabstore3000_STORAGE",
15      "direction": "out"
16    }
17  ]
```

Note that we have "\$return" in this example because our trigger is going to return a JSON object that should be written to the Customer table/

## Lab – Azure Durable Functions

### Understanding Azure Durable Functions

#### What are Azure Durable Functions

- This is an extension of Azure Functions.
- This allows you to write stateful functions.

Function Chaining



Function1

Function2

Function3

Function4



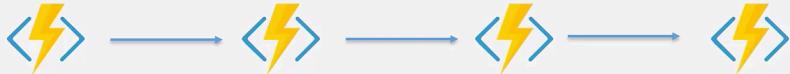
## Lab – Azure Durable Functions

### Understanding Azure Durable Functions

#### What are Azure Durable Functions

- This is an extension of Azure Functions.
- This allows you to write stateful functions.

Function Chaining

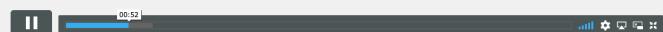


Function1

Function2

Function3

Function4



## Lab – Azure Durable Functions

### Understanding Azure Durable Functions

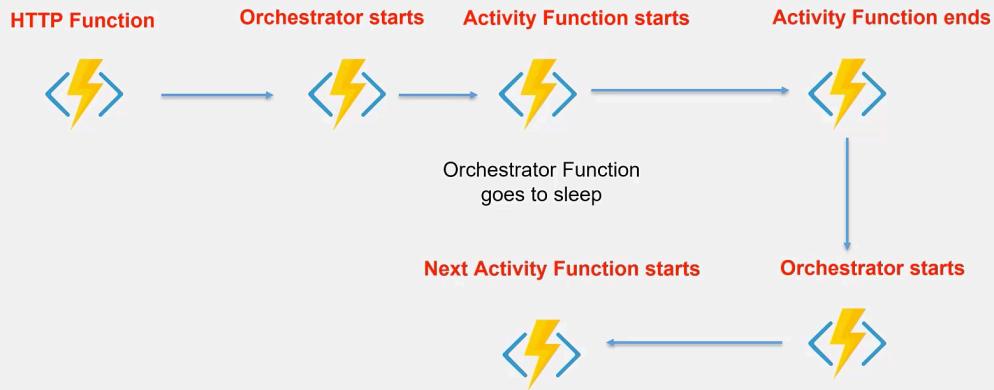
#### What are Azure Durable Functions



## Lab – Azure Durable Functions

### Understanding Azure Durable Functions

#### What are Azure Durable Functions



The screenshot shows the Postman application interface. In the top navigation bar, there are links for File, Edit, View, Help, New, Import, Runner, My Workspace, Invite, Upgrade, and a gear icon for settings. The main workspace is titled "Launchpad" and contains an "Untitled Request". The request type is set to "POST" and the URL is "whizlabfunction.azurewebsite...". Below the URL, there is a parameter "code=rjmRmOEjvhiMZ2C032QgKcmERBgceEjN42WUGHBCF3NIRVSSpwLpA==". The "Params" tab is selected, showing a table with one row for "code". The table has columns for KEY, VALUE, and DESCRIPTION. The KEY column contains "code" with a checked checkbox, and the VALUE column contains "rjmRmOEjvhiMZ2C032QgKcmERBgceEjN42WUGHBC". The DESCRIPTION column contains "Description". Other tabs include Authorization, Headers (8), Body, Pre-request Script, Tests, and Settings. On the right side of the request panel, there are "Send" and "Save" buttons, along with "Comments" and "Code" links. The left sidebar shows a history of requests, including several GET requests to "https://demoapi2000.azure-api.net/api/Courses" and one POST request to the same endpoint. The bottom of the interface features a toolbar with icons for search, refresh, and other functions, and a status bar with "Bootcamp", "Build", "Browse", and a question mark icon.

## Azure CosmosDB

### Lab – Creating a Cosmos DB Account

#### Working with the SQL API

##### Choosing the right partition key

- The partition keys should be chosen in such a way that read operations minimize cross-partition look ups.
- The partition keys should be chosen in such a way that write operations are evenly distributed across different partition key values.
- Also the partition key must be a field present when querying for the data in the database.
- Also note that each item in a container in a database contains an item ID. This is unique within a logical partition. The combination of the partition key and the item ID helps to create the item's index which helps to uniquely identify the item.



## Azure Cosmos DB

Understanding consistency levels

Knowing more on consistency levels

Strong

Bounded Staleness

Session

Consistent prefix

Eventual



Stronger Consistency

Weaker Consistency

Lower throughput

Higher throughput



## Azure Cosmos DB

Understanding consistency levels

Knowing more on consistency levels

- **Bounded staleness** – Here the reads can lag behind the writes by at most "K" versions of an item or by "T" time interval.
- **Session** – Here within a single client session, the reads are guaranteed to honor the consistent-prefix, monotonic reads, monotonic writes, read-your-writes, and write-follows-reads guarantees
- **Consistent prefix** – Here the updates will be seen after a gap, but the client will never see out-of-order writes.



Screenshot of Visual Studio showing the `Customer.cs` file in the editor. The code defines a class `Customer` with properties `customername`, `id`, and `name`, and a constructor that initializes these properties. A yellow lightbulb icon is visible next to the constructor, indicating a warning or suggestion.

```
public class Customer
{
    public string customername { get; set; }

    public Customer()
    {
    }

    public Customer(string city, string id, string name)
    {
        PartitionKey = city;
        RowKey = id;
        customername = name;
    }
}
```

Screenshot of Visual Studio showing the `Program.cs` file in the editor. The code contains an asynchronous method `GetItem()` that retrieves a `Customer` entity from a Azure Cosmos DB table named "customer". It uses CloudStorageAccount and CloudTableClient to interact with the database.

```
static async Task GetItem()
{
    string p_partitionKey = "Chicago";
    string p_rowKey = "1";

    // We are creating a connection to the Azure Cosmos DB account
    CloudStorageAccount whizlabaccount = CloudStorageAccount.Parse(whizlabconnstring);
    // We are creating a table client
    CloudTableClient whizlabtableclient = whizlabaccount.CreateCloudTableClient();
    // We are getting a reference to an existing table
    CloudTable table = whizlabtableclient.GetTableReference("customer");
    // Here we are retrieving an entity based on the partition and row key
    TableOperation whizlaboperation = TableOperation.Retrieve<Customer>(p_partitionKey, p_r
    // We execute the operation
    TableResult result = await table.ExecuteAsync(whizlaboperation);

    // Displaying the properties of the returned object
    Customer obj = result.Result as Customer;
    if (obj != null)
    {
    }
}
```

The screenshot shows the Microsoft Web Function App Portal interface. The left sidebar lists various function types: In Apps, abfunction, actions, +, whizlabtrigger (selected), integrate, manage, monitor, proxies, and vts. The main area displays the code for 'run.csx' under the 'whizlabtrigger' function. The code is as follows:

```
1 #r "Microsoft.Azure.DocumentDB.Core"
2 using System;
3 using System.Collections.Generic;
4 using Microsoft.Azure.Documents;
5
6 public static void Run(IReadOnlyList<Document> input, ILogger log)
7 {
8     if (input != null && input.Count > 0)
9     {
10         log.LogInformation("Documents modified " + input.Count);
11         foreach(var obj in input)
12         {
13             log.LogInformation(JsonConvert.SerializeObject(obj, Formatting.Indented));
14         }
15     }
16 }
17
18
19
20
21
22
```

Below the code editor, there are tabs for Log and Console, and a status bar showing 01:46.

## Lab – Azure Cosmos DB

### Working with stored procedures

#### A quick understanding on stored procedures

- A stored procedure can be used to perform an operation on a Cosmos DB collection.
- Instead of performing that operation directly from a program, you can invoke a stored procedure to carry out the operation.
- The stored procedure in Cosmos DB is written in JavaScript.
- The stored procedure is registered and executed against a collection and runs as a single transaction.



A screenshot of a Notepad window titled "code1 - Notepad". The code is a single-line function:

```
function HelloWorld() {  
    var context = getContext();  
    var response = context.getResponse();  
    response.setBody("Hello, World");  
}
```

The line "response.setBody("Hello, World");" is highlighted with a blue selection bar.

A screenshot of a Visual Studio code editor window titled "Program.cs". The code is a C# program that calls a stored procedure:

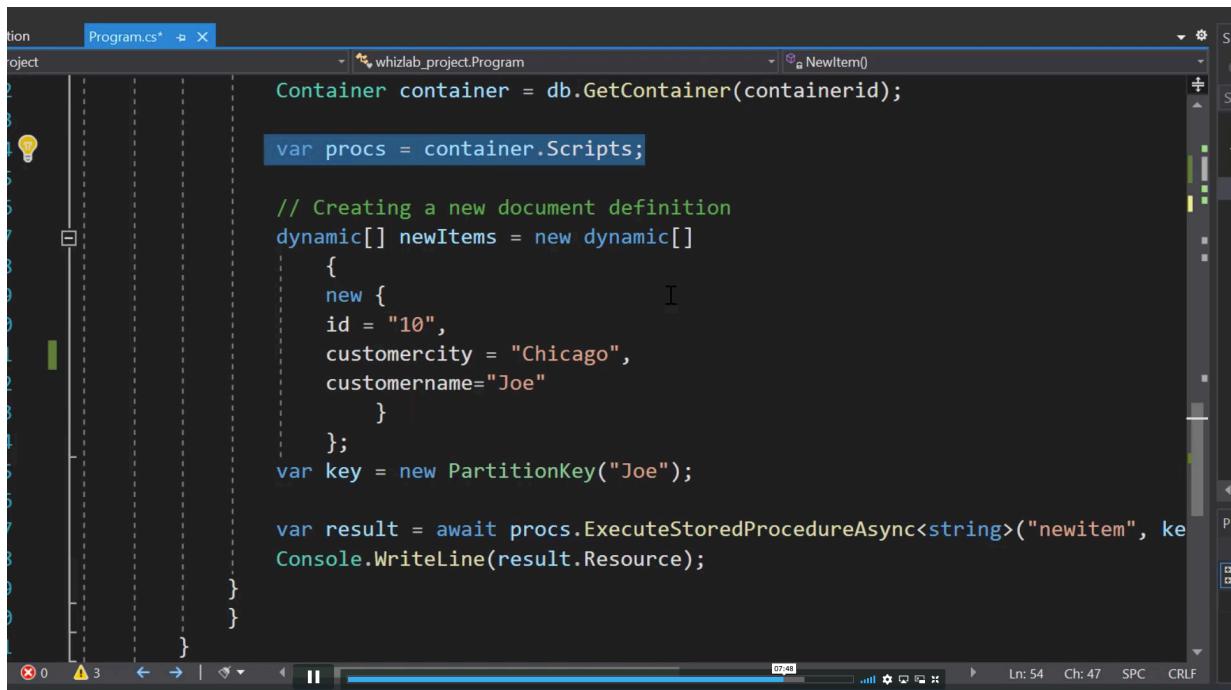
```
private static async Task CallProcedure()  
{  
    // Here we create a connection to the Cosmos DB account using the endpoint and t  
    using (CosmosClient client = new CosmosClient(endpoint, accountkeys))  
    {  
        // Here we get a handle to the database  
        Database db = client.GetDatabase(databaseid);  
        // Here we get a handle to the container  
        Container container = db.GetContainer(containerid);  
  
        // Then we get a link to the scripts for the container  
        var procs = container.Scripts;  
        // Then we create a [ (local variable) Container container  
        var key = new PartitionKey(string.Empty);  
        // We finally call the stored porcedure  
        var result=await procs_ExecuteStoredoredProcedureAsync<string>("starter",key, nu  
  
        Console.WriteLine(result.Resource);  
    }  
}
```

The line "var procs = container.Scripts;" is highlighted with a blue selection bar. A tooltip for "procs" appears above the line, showing "(local variable) Container container".



```
code2 - Notepad
File Edit Format View Help
function createToDoItem(itemToCreate) {
    var context = getContext();
    var container = context.getCollection();

    var accepted = container.createDocument(container.getSelfLink(),
        itemToCreate,
        function (err, itemCreated) {
            if (err) throw new Error('Error' + err.message);
            context.getResponse().setBody(itemCreated.id)
        });
    if (!accepted) return;
}
```



```
Program.cs  whizlab_project.Program  NewItem0
Container container = db.GetContainer(containerid);

var procs = container.Scripts;

// Creating a new document definition
dynamic[] newItems = new dynamic[]
{
    new {
        id = "10",
        customercity = "Chicago",
        customername="Joe"
    }
};
var key = new PartitionKey("Joe");

var result = await procs.ExecuteStoredProcedureAsync<string>("newitem", key);
Console.WriteLine(result.Resource);
}
```

## Lab – Azure Cosmos DB

### Working with Triggers

A quick understanding on the triggers feature

- Triggers allow you to perform an action either before or after an operation occurs on the Cosmos DB collection.
- The operation can include the creation, deletion and replacement of a document.
- The triggers are written in JavaScript.
- You can create both pre-triggers and post-triggers.



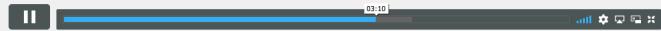
## Azure Storage Accounts

### Azure storage accounts

#### Understanding Azure storage accounts

Quick understanding on the different services in storage accounts

- **General Purpose v2 account – Blob service**
- This is used for storing objects on the cloud.
- This is used for storing large amounts of unstructured data.
- If your application has a need to store data such as images, video or audio files, log files , then consider using the Blob service.

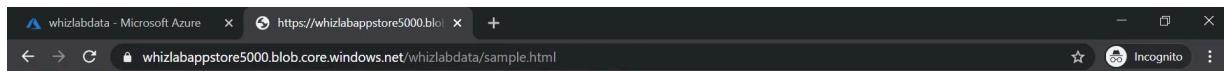


## Azure storage accounts

### Understanding Azure storage accounts

Quick understanding on the different services in storage accounts

- **General Purpose v2 account – Table service**
- This is a service that can be used to store structured NoSQL data in the cloud.
- It's ideal for storage accessed by web applications.
- It's ideal for datasets that don't require complex joins, foreign keys or stored procedures.



[https://whizlabappstore5000.blob.core.windows.net/  
whizlabdata/sample.html](https://whizlabappstore5000.blob.core.windows.net/whizlabdata/sample.html)

```
commands - Notepad
File Edit Format View Help
az storage account create --name whizlab1000 --resource-group whizlabs-rg --location EastUS --sku Standard_LRS

az storage container create --account-name whizlab1000 --name data

echo This is a sample html file > sample.html

az storage blob upload --account-name whizlab1000 --container-name data --name sample.html --file sample.html

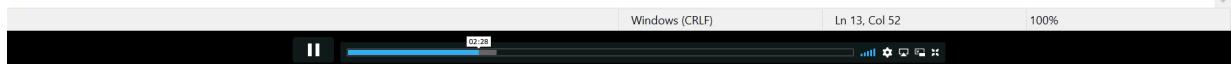
az storage container set-permission --account-name whizlab1000 --name data --public-access blob

az storage blob list --account-name whizlab1000 --container-name data --output table

az storage blob download --account-name whizlab1000 --container-name data --name sample.html --file sample2.html
```

I

All of these commands are available  
as a resource that can be downloaded



```
commands - Notepad
File Edit Format View Help
azcopy login --tenant-id=97ab3455-f094-48ff-b855-10d642093cf2

azcopy make "https://whizlab1000.blob.core.windows.net/newdata"

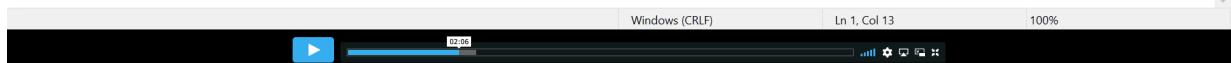
Remember to give Azure Storage Blob Contributor Role Based access to the storage account

azcopy copy "sample.html" "https://whizlab1000.blob.core.windows.net/newdata/sample.html"

azcopy copy "https://whizlab1000.blob.core.windows.net/newdata/sample.html" "sample3.html"
```

I

All of these commands are available  
as a resource that can be downloaded



The screenshot shows a Microsoft Visual Studio interface with the following details:

- Title Bar:** Program.cs\*
- Code Editor:** Displays the following C# code for a static async Task named UploadFile().

```
        }
    }

1 reference
static async Task UploadFile()
{
    // Create a connection to the storage account
    BlobServiceClient w_blobServiceClient = new BlobServiceClient(whizlabconnstring);
    // Get a handle to an existing container
    BlobContainerClient w_containerClient = w_blobServiceClient.GetBlobContainerClient();
    // Create a new reference to a new blob in the container
    BlobClient w_blobClient = w_containerClient.GetBlobClient(whizlabfilename);

    // Upload the local file to a file stream
    using FileStream w_uploadFileStream = File.OpenRead(filepath);
    // Upload the blob to the container
    await w_blobClient.UploadAsync(w_uploadFileStream, true);
    w_uploadFileStream.Close();
}
```
- Status Bar:** Shows 0.0, 5, 06:32, Ln: 44, Ch: 56, SP.

## Implement Azure Security - Implement secure cloud solution

## Lab – Accessing a secret - .Net

How to use a .Net program to access a secret

What are the steps we are going to follow

- o We are going to create a service principal.
- o We are then going to take the values of the service principal and define environment variables on our client system.
- o We will then set the access policy for the key vault to give access for the service principal to secrets in the key vault.
- o We will then use a .Net program to access the value of the secret.



The screenshot shows a Microsoft Notepad window titled "commands - Notepad". The content of the file is as follows:

```
az ad sp create-for-rbac -n whizlabapplication --skip-assignment
Define the following environment variables first
AZURE_CLIENT_ID="38ab1ada-1c41-4bc6-af9e-440db377a2d8"
AZURE_CLIENT_SECRET="cae68966-471e-4714-bc0d-3168cb3d3c3a"
AZURE_TENANT_ID="97ab3455-f094-48ff-b855-10d642093cf2"

az keyvault set-policy --name whizlabkeyvault2000 --spn "38ab1ada-1c41-4bc6-af9e-440db377a2d8" --
secret-permissions backup delete get list set
```

The status bar at the bottom of the Notepad window indicates "Ln 1, Col 47" and "100% Windows (CRLF) UTF-8".

9:43 PM Sat Jul 11

X

## Lab – Accessing the Key Vault

How to access a secret in the Key Vault via an API call

What are the steps we are going to follow

- o We are going to register an application in Azure AD.
- o We will give permissions for the application to invoke the Azure Key vault service.
- o We will generate a secret for the application.
- o We will set the access policy for the key vault to give permissions for our application.
- o We will use the POSTMAN tool to get a bearer token.
- o We will then use the bearer token to fetch the value of a secret in the key vault.

WPS Office

9:46 PM Sat Jul 11 68%

portal.azure.com/#blade/Microsoft\_AAD\_RegisteredApps/ApplicationMenuBlade/CallAnAPI/quickStartType//sourceType/Microsoft\_AAD\_IAM/appId/fcab...

Microsoft Azure

Dashboard > demouser2016@gmail (Default Directory) | App registrations > postman | API permissions

postman | API permissions

Refresh

Successfully granted admin consent for the requested permissions.

Configured permissions

Applications are authorized to call APIs when they are granted permissions by users/admins as part of the consent process. The list of configured permissions should include all the permissions the application needs. [Learn more about permissions and consent](#)

+ Add a permission    Grant admin consent for demouser2016@gmail (Default Directory)

API / Permissions name	Type	Description	Admin consent req...	Status
✓ Azure Key Vault (1)	Delegated	Have full access to the Azure Key Vault service	-	<span>Granted for demouser2016@gmail (Default Directory)</span>

Branding    Authentication    Certificates & secrets    Token configuration    API permissions    Expose an API    Owners    Roles and administrators (Pr...    Manifest    Support + Troubleshooting

## Lab - Role based access control

Working with role-based access control

### What is role-based access control

- Role-based access control helps you manage access to Azure resources.
- This is the authorization system that is used to provide fine-grained access control for Azure resources.
- When you consider role-based access control , you have to consider 3 aspects.
- First is the security principal which can either be a User, Group, Service principal or a Managed identity.
- **User** – This is a profile that you create for an individual in Azure Active Directory.
- **Group** – This is a set of users created in Azure Active Directory.
- **Service principal** – This is a security identity which is created for applications or services.
- **Managed Identity** – This is an identity that is automatically managed by Azure.



## Connect to and consume Azure services and third-party services

## Lab – Azure Service Bus Queue

### Understanding the Message properties

#### The properties of a message in the Service Bus

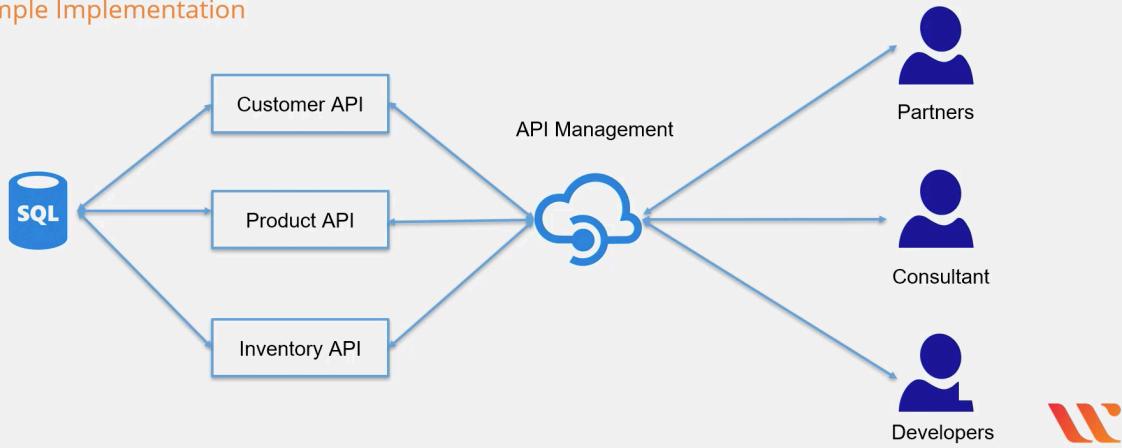
- o Each message that is sent to the Service Bus consists of three elements.
- o One is the body of the message that is in binary format.
- o Next is the broker properties that are defined by the system.
- o Next is the user properties that can be set by the publisher of the message.



## Azure API Management

### Introduction

#### Simple Implementation



# Azure API Management

## Introduction

### The different elements

- o API Gateway.
- o Used to route requests to your backend API.
- o Can cache backend responses.
- o Enforce usage quotas and rate limits.
- o Verify API credentials such as API keys.



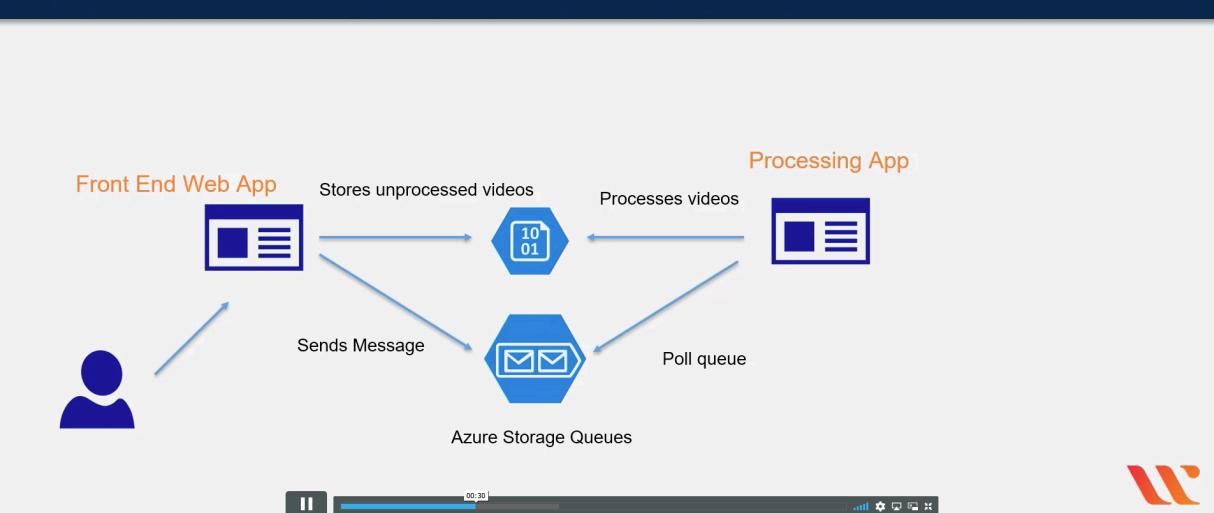
The screenshot shows the Microsoft Azure portal interface for managing APIs. The left sidebar navigation includes 'Overview', 'Activity log', 'Access control (IAM)', 'Tags', 'Diagnose and solve problems', 'General' (with 'Quickstart' and 'Properties' options), and 'APIs' (with 'APIs', 'Named values', 'Subscriptions', and 'Products' sub-options). The main content area displays the 'whizlab | APIs' service. At the top, there are links for 'Publisher portal (legacy)', 'Developer portal (legacy)', and 'Developer portal'. Below this, a 'REVISION 1' section indicates it was 'CREATED May 13, 2020, 4:51:14 PM'. The 'Design' tab is selected, showing the 'All operations' list, which contains a single item: 'GET whizlab'. The 'Definitions' tab is also visible. On the right, the 'Policies' section shows XML code for a policy:

```
10 - Policies are applied in the order of their appearance, from the top down.
11 - Comments within policy elements are not supported and may disappear. Place your
12 -->
13 <!--
14   <policies>
15     <inbound>
16       <ip-filter action="forbid">
17         <address>92.98.36.4</address>
18       </ip-filter>
19     </inbound>
20     <backend>
21       <base />
22     </backend>
23     <outbound>
24       <base />
25     </outbound>
26   </policies>
27 -->
28 </!--
```

Buttons at the bottom include 'Save', 'Discard', 'Reset to default', and 'Calculate effective policy'.

# Azure Storage

## Queues



# Azure Storage

## Queues

### Differences



Azure Storage Queues

- Storage of more than 80GB worth of messages
- Track progress for processing a message inside of the queue
- Server side logs of all of the transactions



Azure Service Bus Queues

- First-in-first-out ordered delivery
- Support automatic duplicate detection
- Publish and consume batches of messages

The screenshot shows the Visual Studio IDE interface with the 'queue-send' project selected in the Solution Explorer. The code editor displays Program.cs with the following code:

```
4
5     queue_send
6
7     Program
8
9     private static string whizlab_connstring = "DefaultEndpointsProtocol=https;AccountName=whizlab;AccountKey=..."; // Replace with your own connection string
10    static void Main(string[] args)
11    {
12        // Create a connection to the storage account
13        CloudStorageAccount w_account = CloudStorageAccount.Parse(whizlab_connstring);
14        // Create a queue client
15        CloudQueueClient w_client = w_account.CreateCloudQueueClient();
16        // Get a reference to an existing client
17        CloudQueue w_queue = w_client.GetQueueReference("whizlabqueue");
18
19        for (int i = 1; i < 10; i++)
20        {
21            string message = $"Sending message {i}";
22            CloudQueueMessage w_message = new CloudQueueMessage(message);
23            w_queue.AddMessage(w_message);
24        }
25    }

```

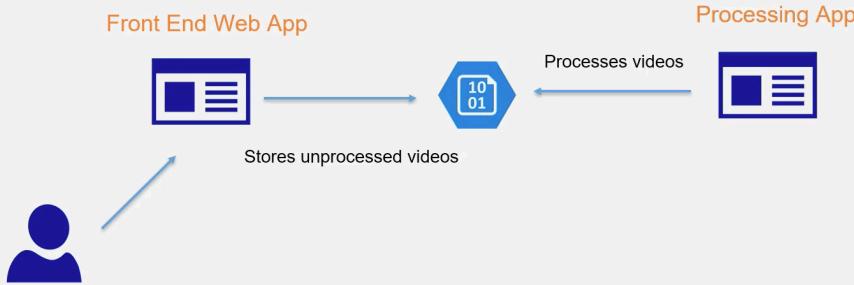
The screenshot shows the Visual Studio IDE interface with the 'queue\_receive' project selected in the Solution Explorer. The code editor displays Program.cs with the following code:

```
0
1     queue_receive
2
3     Program
4
5     static void Main(string[] args)
6     {
7         // Create a connection to the storage account
8         CloudStorageAccount w_account = CloudStorageAccount.Parse(whizlab_connstring);
9         // Create a queue client
10        CloudQueueClient w_client = w_account.CreateCloudQueueClient();
11        // Get a reference to an existing client
12        CloudQueue w_queue = w_client.GetQueueReference("whizlabqueue");
13
14        // Get the attributes of the queue
15        w_queue.FetchAttributes();
16
17        // Get the count of messages in the queue
18        int? count = w_queue.ApproximateMessageCount;
19
20        for (int i = 0; i < count; i++)
21        {
22            // First get the message.
23            CloudQueueMessage w_message = w_queue.GetMessage();
24            // This message will not be visible to consumers for 30 seconds
25            w_message.VisibilityTimeout = TimeSpan.FromSeconds(30);
26            w_queue.UpdateMessage(w_message, MessageUpdateType.Replace);
27        }
28    }

```

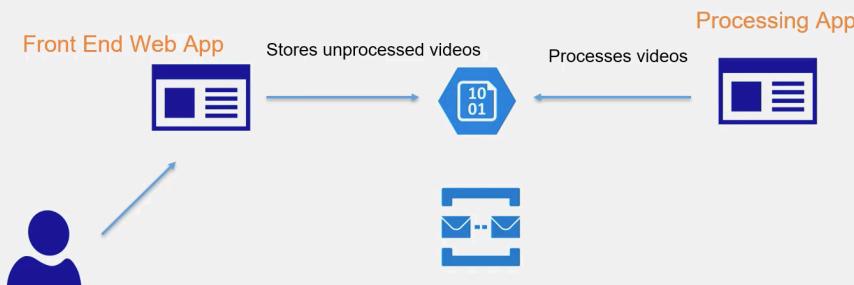
## Azure Service Bus

What's the purpose?



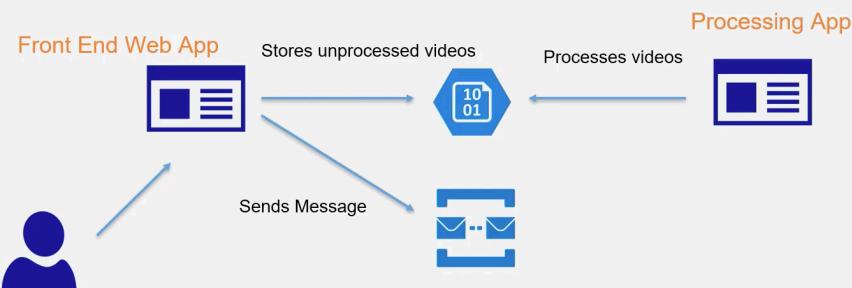
## Azure Service Bus

What's the purpose?



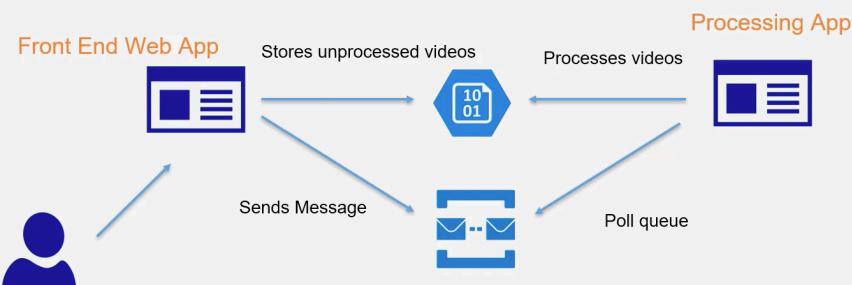
## Azure Service Bus

What's the purpose?



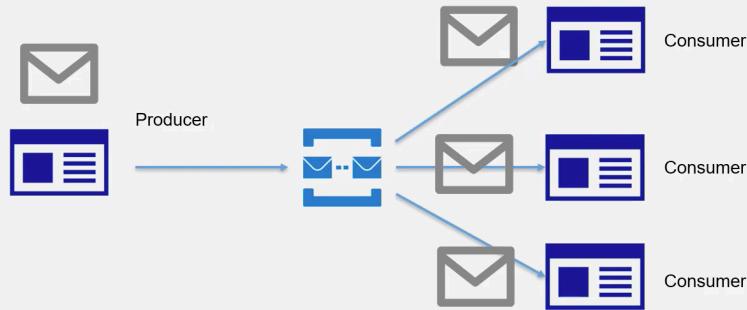
## Azure Service Bus

What's the purpose?



# Azure Service Bus

## Topic



Choose your pricing tier - Microsoft Azure | whizlabfunction4000 - whizlabtrial | +

portal.azure.com/#create/Microsoft.ServiceBus

Microsoft Azure | Search resources, services, and docs (G+)

Dashboard > All resources > New > Service Bus > Create namespace

Create namespace Service Bus

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription \* Staging

Resource group \* whizlabs-rg

Create new

INSTANCE DETAILS

Enter required settings for this namespace.

Namespace name \* whizlab.servicebus.windows.net

Location \* (US) Central US

Pricing tier (View full pricing details) \* Standard

Choose your pricing tier

Browse the available plans and their features

Basic

- Shared capacity
- Max message size 256 KB
- Queues
- Topics
- Variable pricing

0.05 USD/MILLION/MONTH (ESTIMATED)

Standard

- Shared capacity
- Max message size 256 KB
- Queues
- Topics
- Messaging operations 12.5 Ops/Month
- Variable pricing

10.00 USD/12.5MILLION/MONTH (ESTIMATED)

Premium

- Dedicated capacity
- Max message size 1024 KB

Select

Review + create < Previous Next: Tags >

Create queue - Microsoft Azure whizlabfunction4000 - whizlabtri... portal.azure.com/#@demousr2016gmail.onmicrosoft.com/resource/subscriptions/baaa99b3-1d19-4c5e-90e1-39d55de5fc6e/resourceGroups/whizlabs-r... Incognito demousr2016@gmail.c... DEMOUSR2016GMAIL (DEFAULT ...)

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

Dashboard > NoMarketplace | Overview > whizlab | Queues

whizlab | Queues Service Bus Namespace

+ Queue Refresh

Search to filter items...

Name	Status	Max size
No results.		

Create queue

Name \* whizlabqueue

Max queue size 1 GB

Message time to live 14 Days, 0 Hours, 0 Minutes, 0 Seconds

Lock duration 0 Days, 0 Hours, 0 Minutes, 30 Seconds

Enable duplicate detection

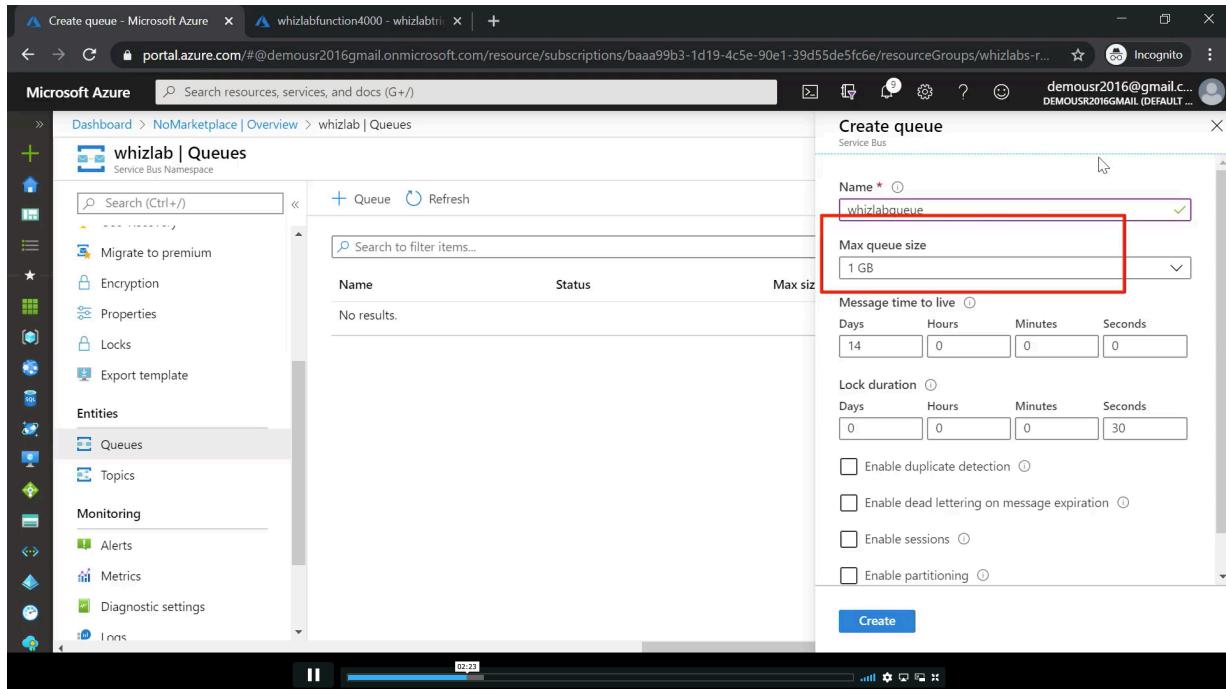
Enable dead lettering on message expiration

Enable sessions

Enable partitioning

Create

02:23



Add SAS Policy - Microsoft Azure whizlabfunction4000 - whizlabtri... portal.azure.com/#@demousr2016gmail.onmicrosoft.com/resource/subscriptions/baaa99b3-1d19-4c5e-90e1-39d55de5fc6e/resourceGroups/whizlabs-r... Incognito demousr2016@gmail.c... DEMOUSR2016GMAIL (DEFAULT ...)

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

Dashboard > NoMarketplace | Overview > whizlab | Queues > whizlabqueue (whizlab/whizlabqueue) | Shared access policies

whizlabqueue (whizlab/whizlabqueue) | Shared access policies

Overview Access control (IAM) Diagnose and solve problems Settings Shared access policies Metrics (preview) Properties Locks Export template Support + troubleshooting New support request

+ Add

Search to filter items...

Policy	Claims
no policies have been set up yet.	

Add SAS Policy

Service Bus

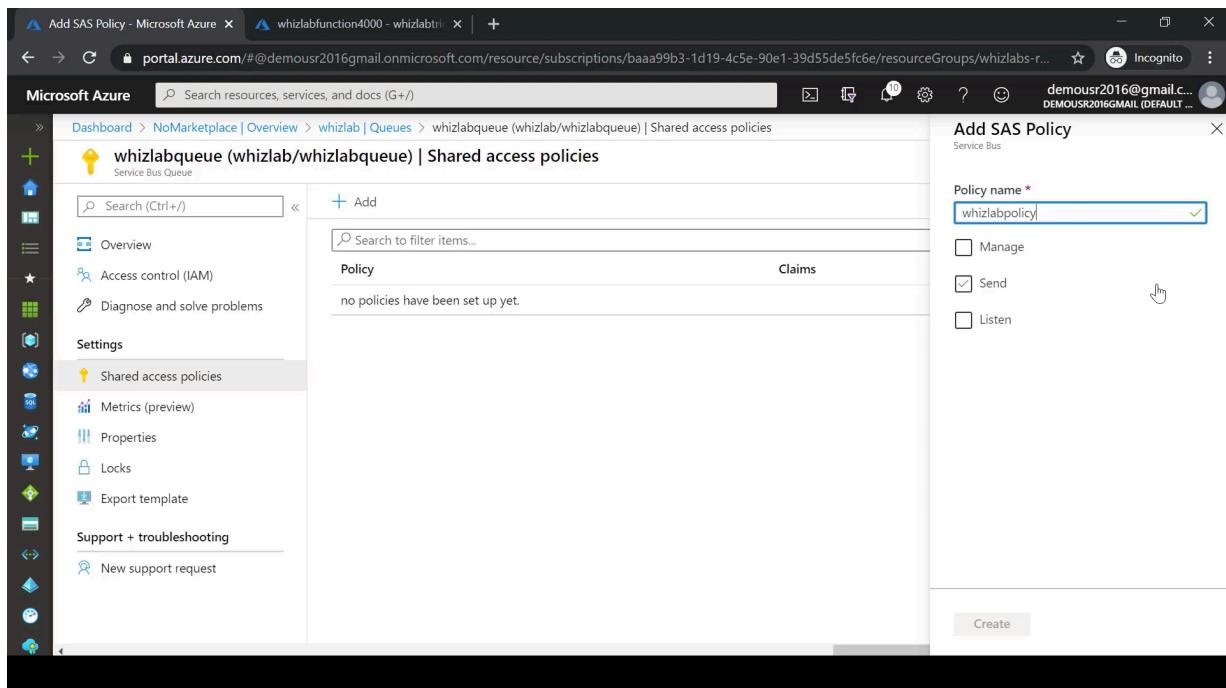
Policy name \* whizlabpolicy

Manage

Send

Listen

Create



The screenshot shows the Microsoft Azure portal interface. The left sidebar has a 'Shared access policies' section selected under 'Settings'. The main content area displays a table for a policy named 'whizlabpolicy'. The table has two columns: 'Policy' and 'Claims'. Under 'Policy', it lists 'whizlabpolicy'. Under 'Claims', it shows 'Send' and 'Listen'. On the right, there are buttons for 'Save', 'Discard', 'Delete', and a 'Manage' button. Below these are four text fields: 'Primary Key' (containing a long hex string), 'Secondary Key' (containing another hex string), 'Primary Connection String' (containing an Endpoint URL), and 'Secondary Connection String' (containing another Endpoint URL). A red box highlights the connection string fields.

Policy	Claims
whizlabpolicy	Send , Listen

Primary Key: GqjS0ZXkDSH9YGVXDvZw2Xvexg9zZu...  
Secondary Key: pOobcqz6V1R2emCSB0090u1Qb9fNn...  
Primary Connection String: Endpoint=sb://whizlab.servicebus.windows.net/  
Secondary Connection String: Endpoint=sb://whizlab.servicebus.windows.net/

The screenshot shows a Visual Studio code editor with a dark theme. The file is named 'queue\_send.Program.cs'. The code defines a static async Main method that creates a QueueClient, sends 20 messages to a queue named 'whizlabqueue', and prints each message's ID to the console. The 'whizlab\_queue' variable is highlighted in blue.

```
private static string whizlab_connection_string = "Endpoint=sb://whizlab.servicebus.windows.net/";  
private static string whizlab_queue = "whizlabqueue";  
  
static async Task Main(string[] args)  
{  
    IQueueClient w_queueclient;  
    // Create a client connection to the queue  
    w_queueclient = new QueueClient(whizlab_connection_string, whizlab_queue);  
    Console.WriteLine("Sending Messages");  
    for (int i = 1; i <=20; i++)  
    {  
        // Create a new message  
        var whizlab_message = new Message(Encoding.UTF8.GetBytes($"Message {i}"));  
        // Send a message to the queue  
        await w_queueclient.SendAsync(whizlab_message);  
        Console.WriteLine($"Message : {i} sent");  
    }  
}
```

The screenshot shows the 'queue\_send' program in Visual Studio. The code is as follows:

```
private static string whizlab_connection_string = "Endpoint=sb://whizlab.servicebus.windows.net";
private static string whizlab_queue = "whizlabqueue";
0 references
static async Task Main(string[] args)
{
    IQueueClient w_queueclient;
    // Create a client connection to the queue
    w_queueclient = new QueueClient(whizlab_connection_string, whizlab_queue);
    Console.WriteLine("Sending Messages");
    for (int i = 1; i <=20; i++)
    {
        // Create a new message
        var whizlab_message = new Message(Encoding.UTF8.GetBytes($"Message {i}"));

        // Send a message to the queue
        await w_queueclient.SendAsync(whizlab_message);
        Console.WriteLine($"Message : {i} sent");
    }
}
```

The screenshot shows the 'queue\_receive' program in Visual Studio. The code is as follows:

```
Whizlab_receive();
}

1 reference
static async Task Whizlab_receive()
{
    // Create a connection to the queue
    whizlab_client = new QueueClient(whizlab_connection_string, whizlab_queue);

    // Specify the options on how you will receive a message
    var _options = new MessageHandlerOptions(whizlab_exception)
    {
        MaxConcurrentCalls = 1,
        AutoComplete = false
    };

    // Register the function and that will receive the messages
    whizlab_client.RegisterMessageHandler(whizlab_message, _options);
    Console.ReadKey();
}
```

A screenshot of a Visual Studio code editor window titled "Program.cs". The code is written in C# and defines a class with a static method "queue\_receive". The method contains logic to register a message handler and read a key from the console. It also includes two delegate definitions: "whizlab\_message" and "whizlab\_exception".

```
    AutoComplete = false
};

// Register the function and that will receive the messages
whizlab_client.RegisterMessageHandler(whizlab_message, _options);
Console.ReadKey();

}

1 reference
static async Task whizlab_message(Message whizlab_message, CancellationToken _token)
{
    Console.WriteLine(Encoding.UTF8.GetString(whizlab_message.Body));
    // Receive a message from the queue
    await whizlab_client.CompleteAsync(whizlab_message.SystemProperties.LockToken);
}

1 reference
static Task whizlab_exception(ExceptionReceivedEventArgs args)
{
```

A screenshot of a Visual Studio code editor window titled "Program.cs". The code is identical to the one in the first screenshot, but the "whizlab\_exception" delegate now contains a body of code that writes the exception to the console and returns a completed task.

```
}

1 reference
static async Task whizlab_message(Message whizlab_message, CancellationToken _token)
{
    Console.WriteLine(Encoding.UTF8.GetString(whizlab_message.Body));
    // Receive a message from the queue
    await whizlab_client.CompleteAsync(whizlab_message.SystemProperties.LockToken);
}

1 reference
static Task whizlab_exception(ExceptionReceivedEventArgs args)
{
    Console.WriteLine(args.Exception);
    return Task.CompletedTask;
}
```

## Lab – Azure Service Bus Queue

### Understanding the Message properties

#### The properties of a message in the Service Bus

- Some of the system-defined broker properties are
- **ContentType** – This describes the payload of the message. An example is application/json.
- **MessageId** – This is an application-defined value that uniquely identifies the message and the payload.
- **CorrelationId** – This is used when a message is being replied to, based on the MessageId.
- **ReplyTo** – This is used to express a reply path to the receiver of the message.
- **SequenceNumber** – This is a 64-bit integer assigned to the message.



## Lab – Azure Service Bus Queue

### Understanding the Message properties

#### The properties of a message in the Service Bus

- Some of the system-defined broker properties are
- **TimeToLive** – This is used to specify a duration after which the message would expire.
- **SessionId** – This can be used for session based applications.
- **LockToken** – This is used to reference the lock that is held by the broker in the peek-lock receive mode.



queue\_send.Program

```
private static string whizlab_connection_string = "Endpoint=sb://whizlab.servicebus.windows.net/";  
private static string whizlab_queue = "whizlabqueue";  
0 references  
static async Task Main(string[] args)  
{  
    IQueueClient w_queueclient;  
    // Create a client connection to the queue  
    w_queueclient = new QueueClient(whizlab_connection_string, whizlab_queue);  
    Console.WriteLine("Sending Messages");  
    for (int i = 1; i <= 20; i++)  
    {  
        // Create a new message  
        var whizlab_message = new Message();  
        whizlab_message.MessageId = $"Message {i}";  
  
        // Send a message to the queue  
        await w_queueclient.SendAsync(whizlab_message);  
        Console.WriteLine($"Message : {i} sent");  
    }  
}
```

Properties

queue\_receive.Program

```
AutoComplete = false  
};  
  
// Register the function and that will receive the messages  
whizlab_client.RegisterMessageHandler(whizlab_message, _options);  
Console.ReadKey();  
  
static async Task whizlab_message(Message whizlab_message, CancellationToken _token)  
{  
    // Receive a message from the queue  
    Console.WriteLine($"Message Id : {whizlab_message.MessageId}");  
    Console.WriteLine($"Sequence number : {whizlab_message.SystemProperties.SequenceNumber}");  
    await whizlab_client.CompleteAsync(whizlab_message.SystemProperties.LockToken);  
}  
  
static Task whizlab_exception(ExceptionReceivedEventArgs args)
```

Properties

Create subscription - Microsoft A whizlabfunction4000 - whizlabtri

portal.azure.com/#@demousr2016gmail.onmicrosoft.com/resource/subscriptions/baaa99b3-1d19-4c5e-90e1-39d55de5fc6e/resourceGroups/whizlabs-r...

Microsoft Azure

Dashboard > NoMarketplace | Overview > whizlab | Topics > whizlabtopic (whizlab/whizlabtopic) > Create subscription

### Create subscription

Service Bus

**SUBSCRIPTION SETTINGS**

Name \*  ✓

Max delivery count \*  ✓

Auto-delete after idle for  Days  Hours  Minutes  Seconds

Never auto-delete

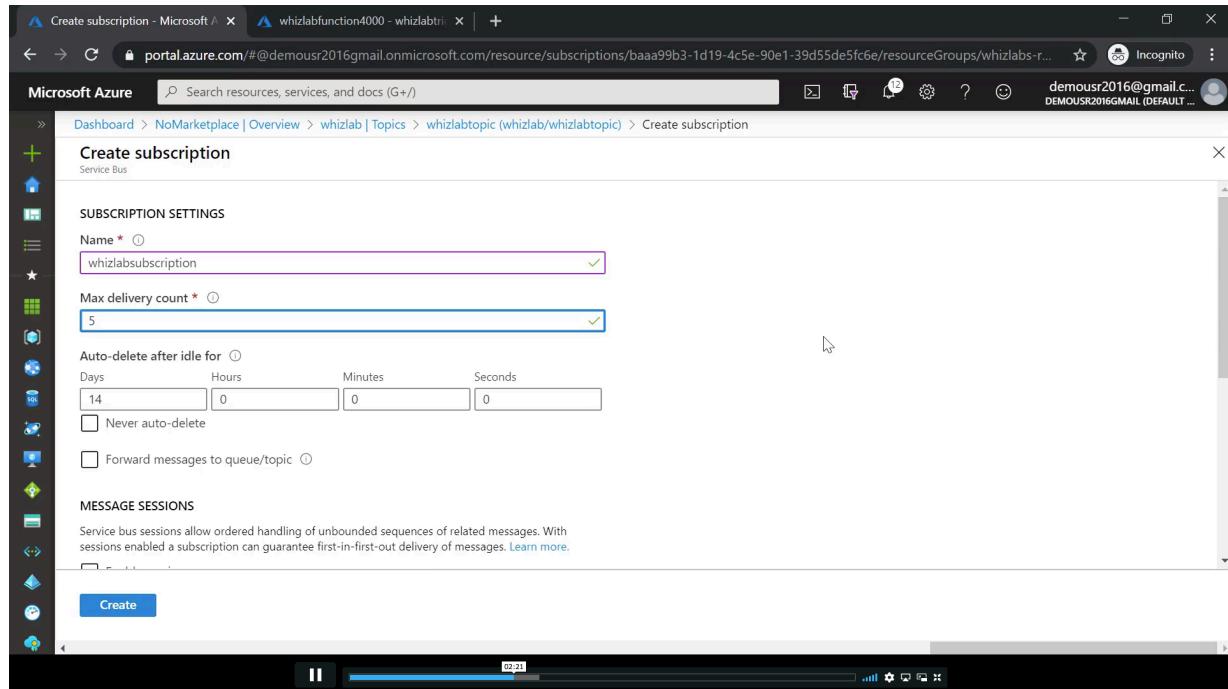
Forward messages to queue/topic

**MESSAGE SESSIONS**

Service bus sessions allow ordered handling of unbounded sequences of related messages. With sessions enabled a subscription can guarantee first-in-first-out delivery of messages. [Learn more.](#)

Enable sessions

**Create**



Create subscription - Microsoft A whizlabfunction4000 - whizlabtri

portal.azure.com/#@demousr2016gmail.onmicrosoft.com/resource/subscriptions/baaa99b3-1d19-4c5e-90e1-39d55de5fc6e/resourceGroups/whizlabs-r...

Microsoft Azure

Dashboard > NoMarketplace | Overview > whizlab | Topics > whizlabtopic (whizlab/whizlabtopic) > Create subscription

### Create subscription

Service Bus

**SUBSCRIPTION SETTINGS**

Name \*  ✓

Max delivery count \*  ✓

Auto-delete after idle for  Days  Hours  Minutes  Seconds

Never auto-delete

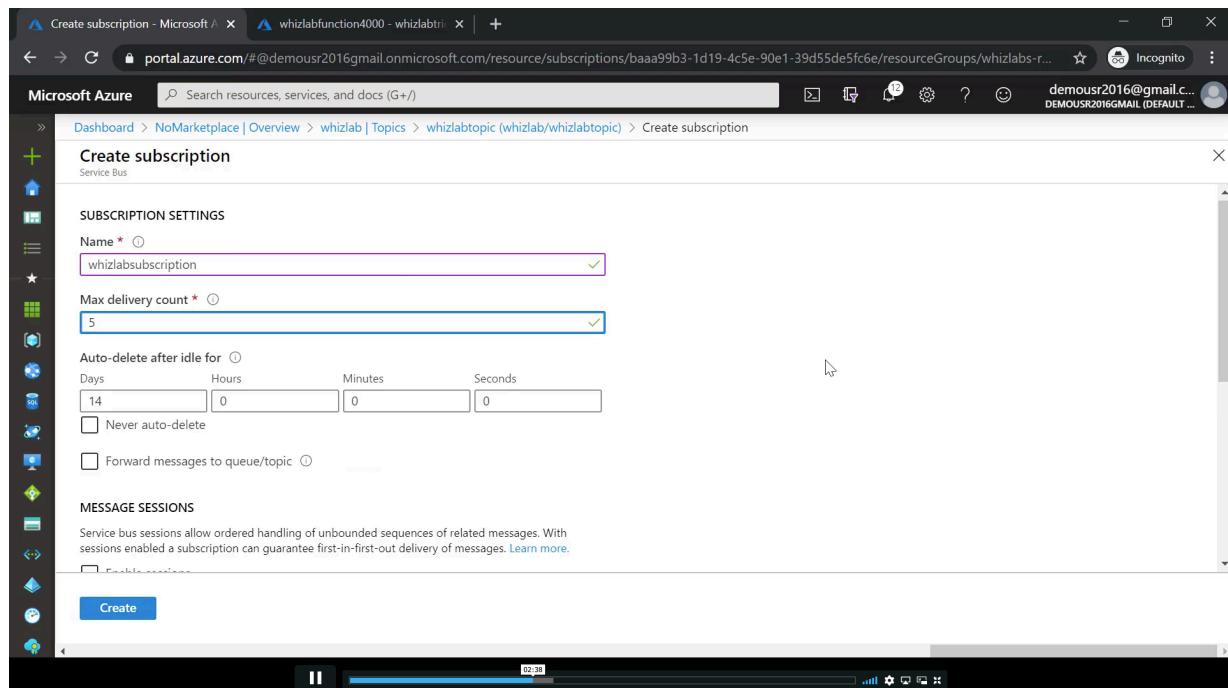
Forward messages to queue/topic

**MESSAGE SESSIONS**

Service bus sessions allow ordered handling of unbounded sequences of related messages. With sessions enabled a subscription can guarantee first-in-first-out delivery of messages. [Learn more.](#)

Enable sessions

**Create**



The screenshot shows the Visual Studio IDE interface with the 'topic\_send' project selected. The 'Program.cs' file is open, displaying C# code for sending messages to a Service Bus topic. The code defines a connection string and a topic, creates a TopicClient, and sends 20 messages. The code editor has syntax highlighting and a status bar at the bottom.

```
private static string whizlab_connection_string = "Endpoint=sb://whizlab.servicebus.windows.net/";  
private static string whizlab_topic = "whizlabtopic";  
private static ITopicClient whizlab_client;  
  
static async Task Main(string[] args)  
{  
    // Create a connection to the topic  
    whizlab_client = new TopicClient(whizlab_connection_string, whizlab_topic);  
  
    for (int i = 1; i <= 20; i++)  
    {  
        // Create a message  
        var whizlab_message = new Message(Encoding.UTF8.GetBytes($"Message {i}"));  
        Console.WriteLine($"Message : {i} sent");  
        // Send the message to the topic  
        await whizlab_client.SendAsync(whizlab_message);  
    }  
}
```

The screenshot shows the Visual Studio IDE interface with the 'topic\_receive' project selected. The 'Program.cs' file is open, displaying C# code for receiving messages from a Service Bus subscription. It creates a connection string builder, a SubscriptionClient, and registers a message handler. The code editor has syntax highlighting and a status bar at the bottom.

```
ServiceBusConnectionStringBuilder w_builder = new ServiceBusConnectionStringBuilder();  
whizlab_client = new SubscriptionClient(w_builder, whizlab_subscription);  
  
var _options = new MessageHandlerOptions(whizlab_exception)  
{  
    MaxConcurrentCalls = 1,  
    AutoComplete = false  
};  
  
whizlab_client.RegisterMessageHandler(whizlab_message, _options);
```

## Lab – Azure Service Bus Topic

### Working with Azure Service Bus topics filters

#### What are Service Bus topic filters

- Here a subscriber can choose to define which messages they want to receive.
- You can define a filter condition to specify which messages that need to be received.
- There are three types of filter conditions
- **Boolean filters** – This can be a TrueFilter or FalseFilter. Here the subscriber can receive all of the messages if the filter is mentioned as TrueFilter.
- **SQL Filters** – Here you can define SqlFilter to define what type of messages to receive.
- **Correlation Filters** – Here you can compare against the CorrelationId property or other user and system defined properties.



The screenshot shows the Microsoft Azure portal interface for managing a Service Bus Topic. The top navigation bar includes links for 'whizlabsubscription (whizlab/whizlabtopic)' and 'whizlabfunction4000 - whizlabtopic'. The main title is 'whizlabsubscription (whizlab/whizlabtopic/whizlabsubscription)'. The left sidebar has sections for 'Overview', 'Diagnose and solve problems', 'Settings' (with 'Export template'), and 'Support + troubleshooting'. The main content area displays various configuration settings: Max delivery count (5), Message time to live (14 DAYS), Message lock duration (30 SECONDS), Auto-delete after idle for (14 DAYS), Active message count (0 MESSAGES), Dead-letter message count (0 MESSAGES), Transfer dead-letter messages (0 MESSAGES), Scheduled message count (0 MESSAGES), and Transfer message count (0 MESSAGES). A 'FILTERS' section shows a single entry named '\$Default' of type 'SqlFilter'. A large panel on the right is titled '\$Default' and describes it as a SQL-like expression for filtering messages based on message properties.

Microsoft Azure | portal.azure.com/#@demouser2016@gmail.onmicrosoft.com/resource/subscriptions/baaa99b3-1d19-4c5e-90e1-39d55de5fc6e/resourceGroups/whizlabs-r...

whizlabsubscription (whizlab/whizlabtopic) > whizlabsubscription (whizlab/whizlabtopic/whizlabsubscription)

whizlabsubscription (whizlab/whizlabtopic/whizlabsubscription)

Service Bus Subscription

Search (Ctrl+ /) Delete Refresh

FILTERS

+ Add filter

Name	Filter Type
whizlabfilter	SqlFilter

whizlabfilter

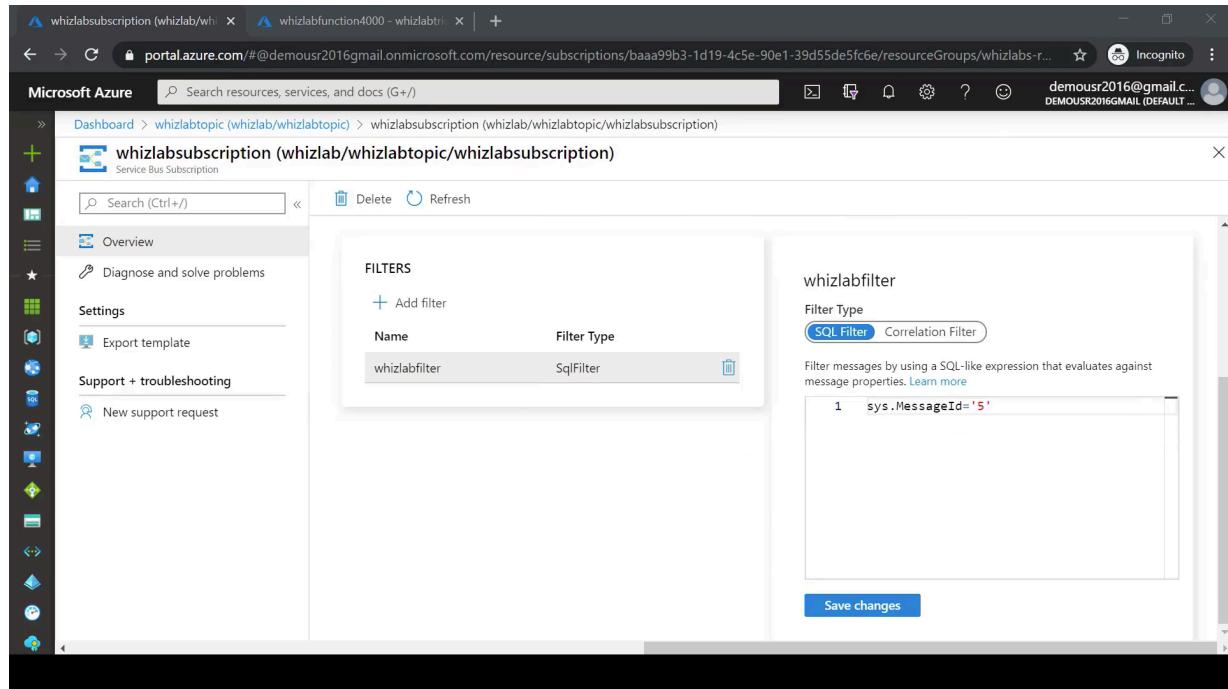
Filter Type

SQL Filter Correlation Filter

Filter messages by using a SQL-like expression that evaluates against message properties. [Learn more](#)

```
1 sys.MessageId = '5'
```

Save changes



Microsoft Azure | portal.azure.com/#@demouser2016@gmail.onmicrosoft.com/resource/subscriptions/baaa99b3-1d19-4c5e-90e1-39d55de5fc6e/resourceGroups/whizlabs-r...

whizlabtopic (whizlab/whizlabtopic) > whizlabtopic (whizlab/whizlabtopic)

whizlabtopic (whizlab/whizlabtopic)

Service Bus Topic

Search (Ctrl+ /) + Subscription Delete

Namespace: whizlab

Topic URL: <https://whizlab.servicebus.windows.net/whizlabtopic>

Active message count: 0 MESSAGES Max size: 1 GB

Scheduled message count: 0 MESSAGES CURRENT: 0.2 kB

Dead-letter message count: 0 MESSAGES

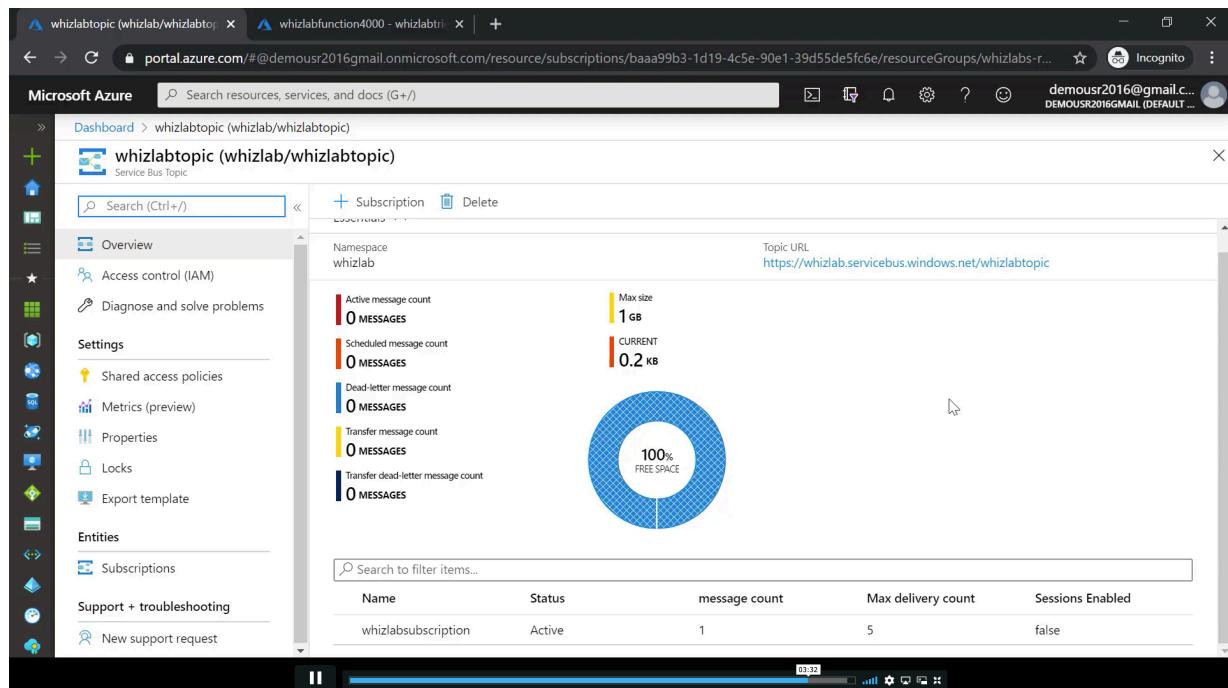
Transfer message count: 0 MESSAGES

Transfer dead-letter message count: 0 MESSAGES

100% FREE SPACE

Search to filter items...

Name	Status	message count	Max delivery count	Sessions Enabled
whizlabsubscription	Active	1	5	false



whizlabweb6000 | Scale out (App Service plan)

whizlabqueue - Microsoft Azure

Updating autoscale configuration  
Updating autoscale configuration for 'ASP-whizlabsrg-8fb1'.

Save Discard Refresh Provide feedback

Delete warning

The very last or default recurrence rule cannot be deleted. Instead, you can disable autoscale to turn off autoscale.

Scale mode

Scale based on a metric  Scale to a specific instance count

Rules

Scale out

When whizlabqueue ApproximateMessageCount > 5 Increase count by 1

Scale in

When whizlabqueue ApproximateMessageCount < 5 Decrease count by 1

+ Add a rule

Instance limits

Minimum  Maximum  Default   
1 3 1

Schedule

This scale condition is executed when none of the other scale condition(s) match

+ Add a scale condition

## Lab – Azure Event Grid

### Working with Azure Event Grid

#### What is the Event Grid Service

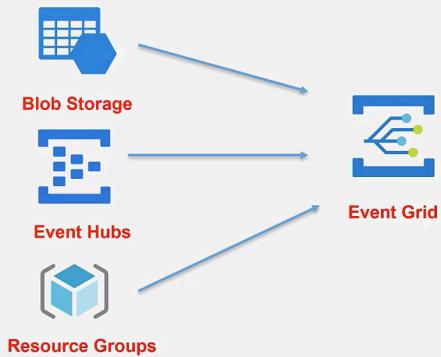
- You can take advantage of the Event Grid Service to listen to events emitted by Azure services.
- You can subscribe to the various events published by Azure services.
- You can then define an Event Handler that can be used to process the event.
- You can also define your own custom events.
- This service is a highly available service because it is spread across multiple fault domains and availability zones in every region.



## Lab – Azure Event Grid

### Working with Azure Event Grid

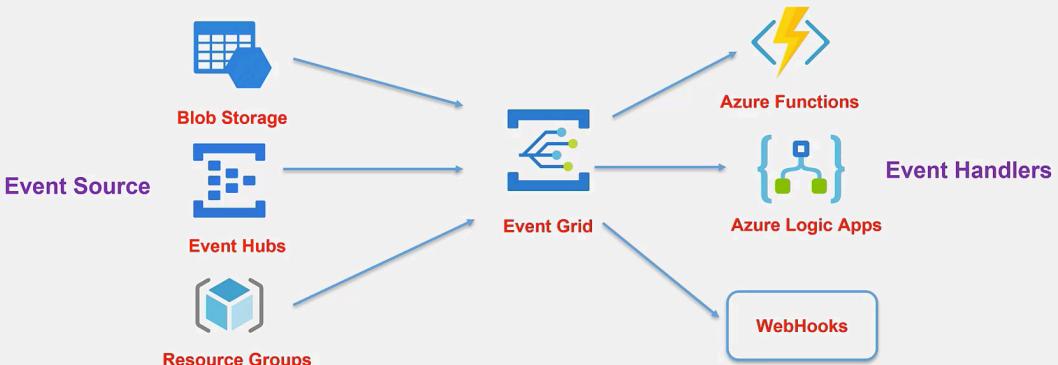
What is the Event Grid Service



## Lab – Azure Event Grid

### Working with Azure Event Grid

What is the Event Grid Service



## Lab – Azure Event Grid

### Working with Azure Event Grid

#### What is the Event Grid Service

- The five important concepts when it comes to the Azure Event Grid
- **Events** – This describes what happened.
- **Event sources** – This is the place where the event took place.
- **Topics** – This is the endpoint where publishers send events.
- **Event subscriptions** – This is the endpoint to route events to handlers.
- **Event handlers** – The application or service that will receive the event.



## Lab – Azure Event Grid

### Working with Azure Event Grid

#### The Event Grid Schema

- The Event Grid schema that is common for all events

```
[  
  {  
    "topic": string,  
    "subject": string,  
    "id": string,  
    "eventType": string,  
    "eventTime": string,  
    "data":{  
      object-unique-to-each-publisher  
    },  
    "dataVersion": string,  
    "metadataVersion": string  
  }  
]
```

Reference - <https://docs.microsoft.com/en-us/azure/event-grid/event-schema>



## Lab – Azure Event Grid

### Working with Azure Event Grid

#### What are we going to do in our lab

- We are going to create an Azure Function based on the Event Grid trigger.
- We will make the Azure Function subscribe to the events emitted by a storage account.
- We will perform an action on the storage account.

The screenshot shows the Microsoft Azure portal interface. The left sidebar navigation bar includes 'Dashboard', 'Storage', 'Compute', 'Networking', 'Machine Learning', 'Monitoring', 'Logs', 'Metrics', 'Logs Analytics', 'Logs Analytics Workspaces', 'Logs Analytics Metrics', and 'Logs Analytics Metrics'. The main content area displays the 'whizlabfunction4000 - whizlabtrigger' blade. The blade title is 'whizlabfunction4000 - whizlabtrigger' and the sub-title is 'Function Apps'. The 'Functions' section shows a single function named 'whizlabtrigger'. The 'run.csx' code editor contains the following C# code:

```
1 #r "Microsoft.Azure.EventGrid"
2 using Microsoft.Azure.EventGrid.Models;
3
4 public static void Run(EventGridEvent eventGridEvent, ILogger log)
5 {
6     log.LogInformation(eventGridEvent.Data.ToString());
7 }
8
```

Below the code editor are 'Save' and 'Run' buttons, and a link to 'Add Event Grid subscription'. At the bottom of the blade are 'Logs' and 'Console' tabs, and a media control bar with a play/pause button and a progress bar showing '04:51'.

data - Microsoft Azure  Create Event Subscription - Microsoft Azure

portal.azure.com/#blade/WebsitesExtension/FunctionsFrameBlade/id/%2Fsubscriptions%2Fbaaa99b3-1d19-4c5e-90e1-39d55de5fc6e%2Fresourcegroup...

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

Incognito DEMOUSR2016GMAIL (DEFAULT ...)

Dashboard > Microsoft.Web.FunctionApp-Portal-e2aa5a65-845d | Overview > whizlabfunction4000 - whizlabtrigger > Create Event Subscription

## Create Event Subscription

Event Grid

Basic Filters Additional Features Advanced Editor

Event Subscriptions listen for events emitted by the topic resource and send them to the endpoint resource. [Learn more](#)

**EVENT SUBSCRIPTION DETAILS**

Name: whizlab

Event Schema: Event Grid Schema

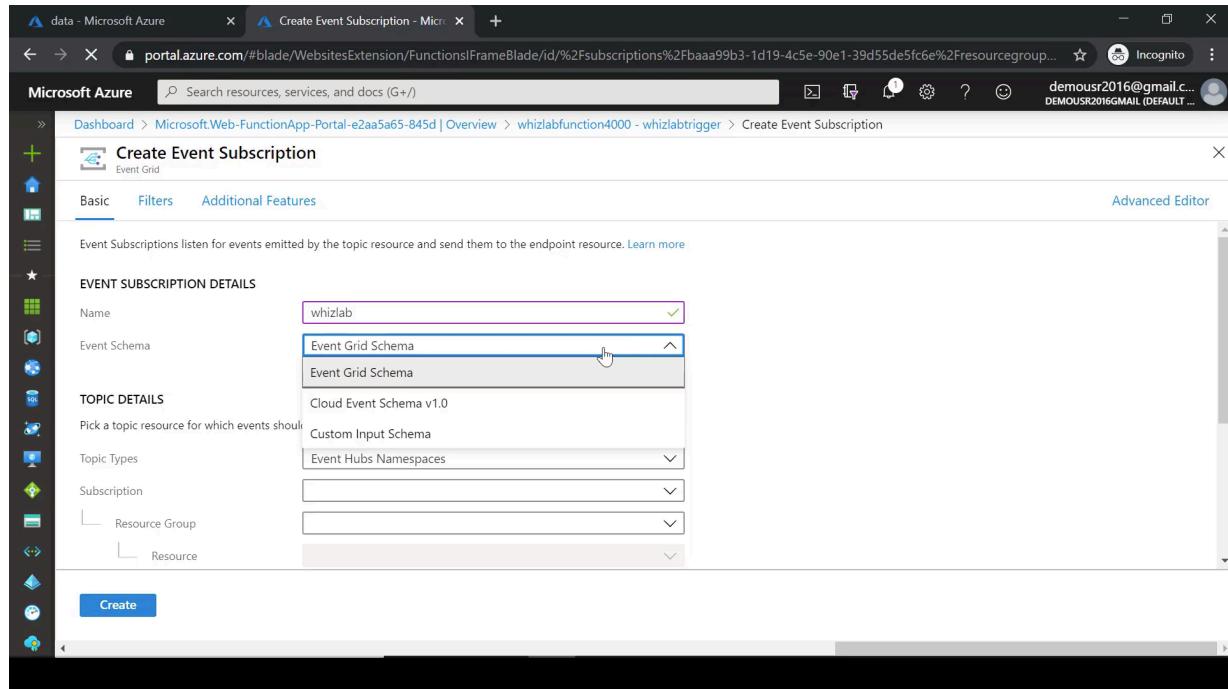
**TOPIC DETAILS**

Pick a topic resource for which events should be pushed to your destination. [Learn more](#)

Topic Types: Event Hubs Namespaces

Subscription: Resource Group: whizlabs-rg, Resource: Loading...

**Create**



data - Microsoft Azure  Create Event Subscription - Microsoft Azure

portal.azure.com/#blade/WebsitesExtension/FunctionsFrameBlade/id/%2Fsubscriptions%2Fbaaa99b3-1d19-4c5e-90e1-39d55de5fc6e%2Fresourcegroup...

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

Incognito DEMOUSR2016GMAIL (DEFAULT ...)

Dashboard > Microsoft.Web.FunctionApp-Portal-e2aa5a65-845d | Overview > whizlabfunction4000 - whizlabtrigger > Create Event Subscription

## Create Event Subscription

Event Grid

Basic Filters Additional Features Advanced Editor

Pick a topic resource for which events should be pushed to your destination. [Learn more](#)

Topic Types: Storage Accounts (Blob & GPv2)

Subscription: Staging

Resource Group: whizlabs-rg

Resource: Loading...

**EVENT TYPES**

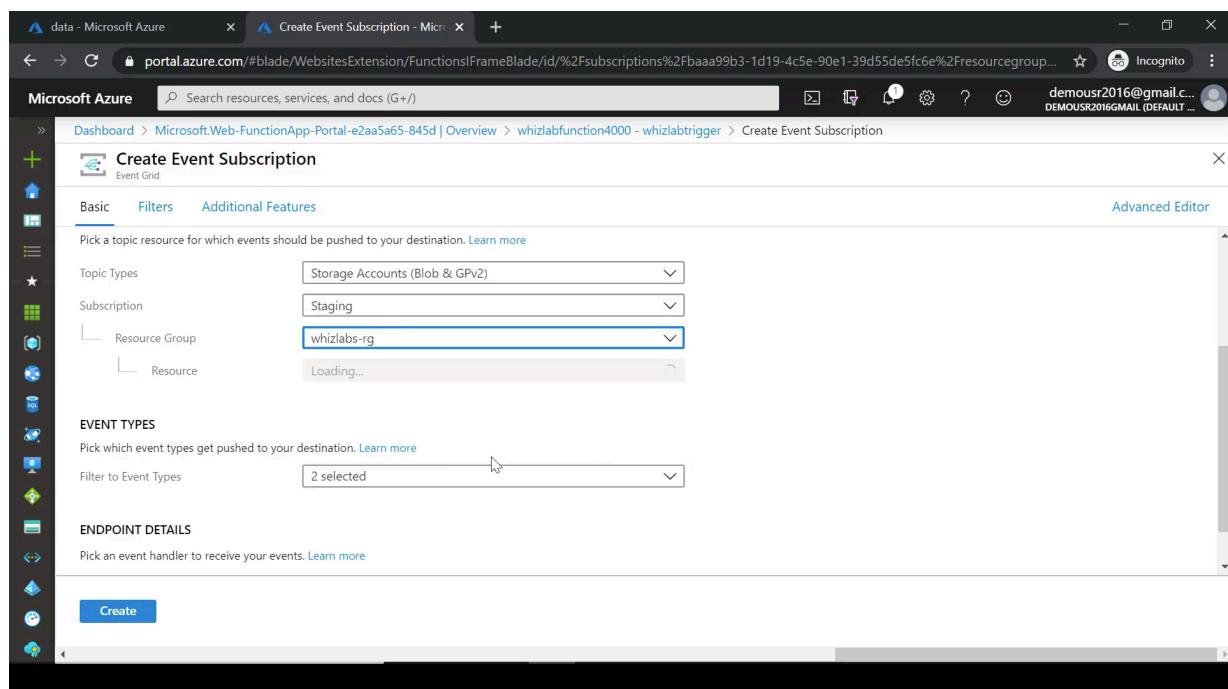
Pick which event types get pushed to your destination. [Learn more](#)

Filter to Event Types: 2 selected

**ENDPOINT DETAILS**

Pick an event handler to receive your events. [Learn more](#)

**Create**



[data - Microsoft Azure](#) [Create Event Subscription - Micro](#)

portal.azure.com/#blade/WebsitesExtension/FunctionsFrameBlade/id/%2Fsubscriptions%2Fbaaa99b3-1d19-4c5e-90e1-39d55de5fc6e%2Fresourcegroup...

Microsoft Azure  Incognito demousr2016@gmail.c... DEMOUSR2016GMAIL (DEFAULT ...)

Dashboard > Microsoft.Web.FunctionApp-Portal-e2aa5a65-845d | Overview > whizlabfunction4000 - whizlabtrigger > Create Event Subscription

## Create Event Subscription

Event Grid

Basic Filters Additional Features Advanced Editor

### TOPIC DETAILS

Pick a topic resource for which events should be pushed to your destination. [Learn more](#)

Topic Type: Storage account  
Topic Resource: whizlabstore4000 (change)

### EVENT TYPES

Pick which event types get pushed to your destination. [Learn more](#)

Filter to Event Types: 2 selected

- Blob Created
- Blob Deleted
- Directory Created
- Directory Deleted
- Blob Renamed

Endpoint Type: Endpoint: [Create](#)

[whizlabstore4000 | Events - Microsoft Azure](#) [whizlabfunction4000 - whizlabtrig...](#)

portal.azure.com/#@demousr2016gmail.onmicrosoft.com/resource/subscriptions/baaa99b3-1d19-4c5e-90e1-39d55de5fc6e/resourceGroups/whizlabs-r...

Microsoft Azure  Incognito demousr2016@gmail.c... DEMOUSR2016GMAIL (DEFAULT ...)

Dashboard > Storage accounts > whizlabstore4000 | Events

## whizlabstore4000 | Events

Storage account

Search (Ctrl+/  
)

Event Subscription Refresh

100  
90  
80  
70  
60  
50  
40  
30  
20  
10  
0

2:30 PM 2:45 PM May 13 2:53 PM 3 PM 3:15 PM UTC+04:00

Published Events (Sum) whizlabstore4000/mic... Publish Failed Event... whizlabstore4000/mic... Unmatched Events (Sum) whizlabstore4000/mic...

Search to filter items...

Name	Endpoint	Prefix Filter	Suffix Filter	Event Types
whizlab	AzureFunction			Microsoft.Storage.BlobCreated,Microsoft.Sto...

## Lab – Azure Event Hubs

### Working with Azure Event Hubs

#### What are Event Hubs

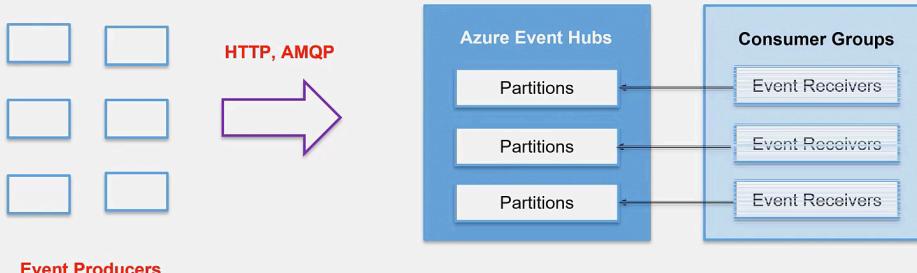
- This is a big data streaming platform and event ingestion service.
- This service can be used to receive and process millions of events per second.
- The data that is sent to the Event Hub can then be transformed and stored in other data stores for data processing.
- Some of the areas where you can use Event Hubs , is for Anomaly detection, Application logging, clickstreams, telemetry processing.



## Lab – Azure Event Hubs

### Working with Azure Event Hubs

#### What are the main components of Event Hubs



## Lab – Azure Event Hubs

### Working with Azure Event Hubs

#### What are the main components of Event Hubs

- **Event producers** – This is the entity that is used to send data to the event hub.
- **Partition** – Here each consumer reads a specific subset, or partition of the message stream.
- **Consumer groups** – This is a view of the entire event hub.
- **Throughput units** – This controls the capacity of the event hub.
- **Event receivers** – This is an entity that reads event data from the event hub.



## Lab – Azure Event Hubs

### Working with Azure Event Hubs

#### What are the main components of Event Hubs

- **Throughput units** – These are pre-purchases units of capacity. One throughput unit gives you
  - Ingress – Up to 1 MB per second or 1000 events per second.
  - Egress - Up to 2 MB per second or 4096 events per second.



The screenshot shows the Visual Studio IDE interface with the 'whizlab-send' project selected. The code editor displays the 'Program.cs' file, which contains C# code for sending events to an Event Hub. The code uses the `EventHubProducerClient` to create batches and send individual events. A break point is set at line 22. The status bar at the bottom indicates the code is ready.

```
17
18     }
19
20     static private async Task SendEvents()
21     {
22         EventHubProducerClient whizlab_client = new EventHubProducerClient(whizlab_connstring);
23         while (true)
24         {
25             Random generator = new Random();
26             int price = generator.Next(1000);
27             int quantity = generator.Next(100);
28             Order obj = new Order(Guid.NewGuid().ToString(), quantity, price);
29             using EventDataBatch whizlab_batch = await whizlab_client.CreateBatchAsync();
30             whizlab_batch.TryAdd(new EventData(Encoding.UTF8.GetBytes(obj.ToString())));
31             await whizlab_client.SendAsync(whizlab_batch);
32             Console.WriteLine("Sending Message - {0}", obj.ToString());
33         }
34     }
35
36 }
```

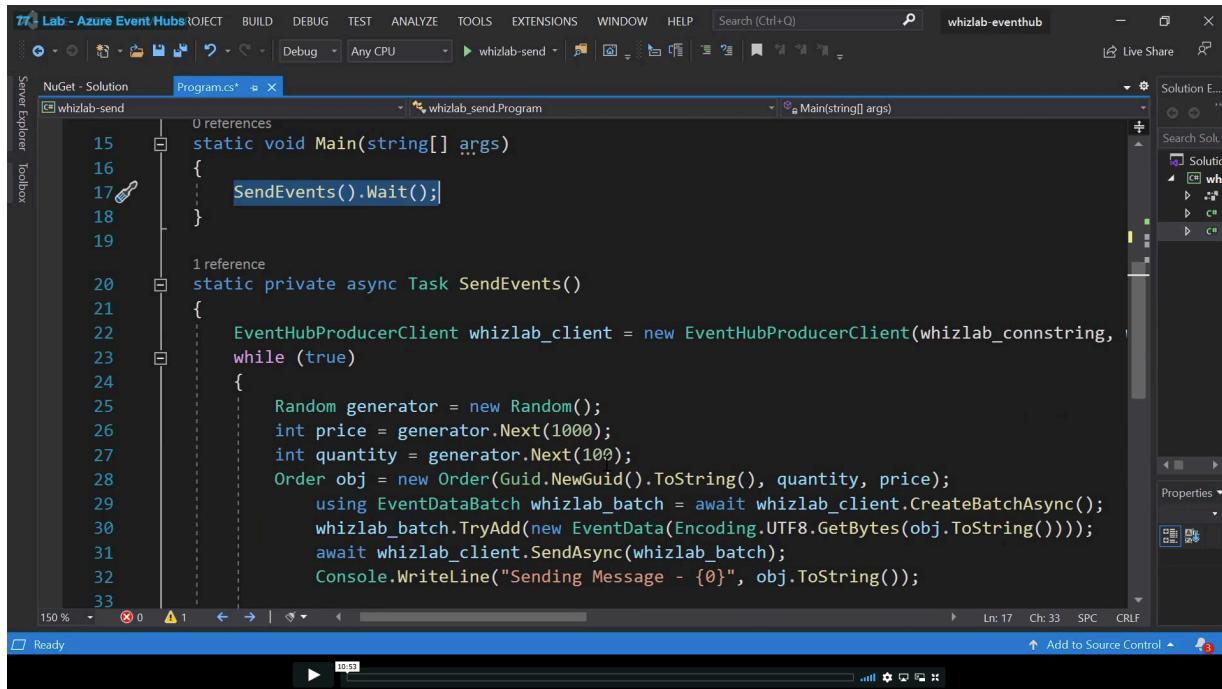
The screenshot shows the Visual Studio IDE interface with the 'whizlab-receive' project selected. The code editor displays the 'Program.cs' file, which contains C# code for receiving events from an Event Hub. It uses the `EventHubConsumerClient` to read events. A break point is set at line 22. The status bar at the bottom indicates the code is ready.

```
15
16
17
18
19
20     static void Main(string[] args)
21     {
22         EventHubConsumerClient client = new EventHubConsumerClient("$Default", whizlab_connstring, whizlab_hubname);
23         CancellationTokenSource cancellationToken = new CancellationTokenSource();
24         string consumerGroup = "Test Group";
25         client.AcceptMessageHandler((e) => {
26             Console.WriteLine($"Received event: {e.Data}");
27             foreach (var message in e.Data)
28             {
29                 var data = message.Body;
30                 var bodyString = Encoding.UTF8.GetString(data.Body.ToArray());
31             }
32         });
33         client.ReadEventsAsync(cancellationToken.Token);
```

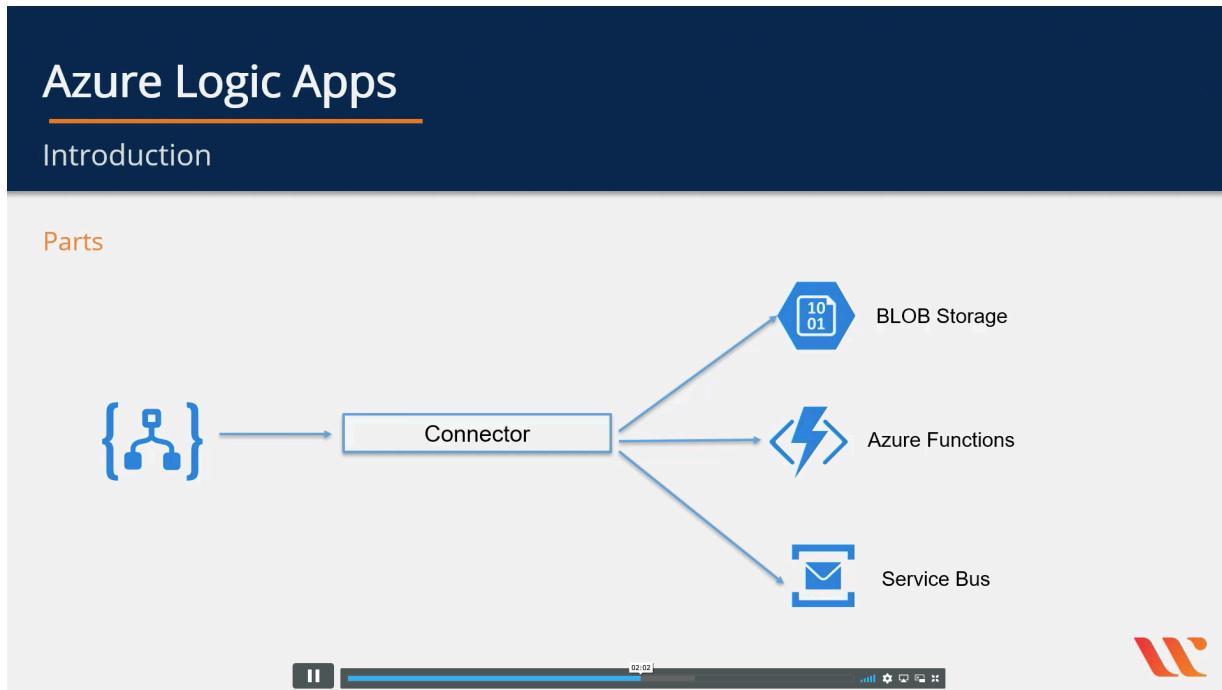
The screenshot shows the Microsoft Azure portal interface. The top navigation bar includes tabs for 'whizlabhub (whizlab2000/whizlabhub)' and 'data - Microsoft Azure'. The main content area is titled 'whizlabhub (whizlab2000/whizlabhub) | Consumer groups'. On the left, a sidebar menu lists options like Overview, Access control (IAM), Diagnose and solve problems, Settings (Shared access policies, Properties, Locks, Export template), Entities (Consumer groups), and Features (Capture, Process data). The central pane displays a table of consumer groups with one entry: '\$Default' located in Central US.

The screenshot shows the Visual Studio IDE interface. The title bar reads 'whizlab-eventhub-1'. The main window displays the 'Program.cs' file of a C# project named 'whizlab-receive'. The code implements an asynchronous task to read events from an Event Hub:

```
18 }  
19 }  
20 1 reference  
21 static private async Task<> ReceiveEvents()  
22 {  
23     EventHubConsumerClient client = new EventHubConsumerClient("$Default", whizlab_connstr);  
24     var cancellation = new CancellationToken();  
25  
26     Console.WriteLine("Receiving Events");  
27     await foreach (PartitionEvent whizlab_event in client.ReadEventsAsync(cancellation))  
28     {  
29         EventData data = whizlab_event.Data;  
30         Console.WriteLine(Encoding.UTF8.GetString(data.Body.ToArray()));  
31     }  
32 }  
33 }  
34 }  
35 }  
36 }
```



```
15 static void Main(string[] args)
16 {
17     SendEvents().Wait();
18 }
19
20 static private async Task SendEvents()
21 {
22     EventHubProducerClient whizlab_client = new EventHubProducerClient(whizlab_connstring,
23     while (true)
24     {
25         Random generator = new Random();
26         int price = generator.Next(1000);
27         int quantity = generator.Next(100);
28         Order obj = new Order(Guid.NewGuid().ToString(), quantity, price);
29         using EventDataBatch whizlab_batch = await whizlab_client.CreateBatchAsync();
30         whizlab_batch.TryAdd(new EventData(Encoding.UTF8.GetBytes(obj.ToString())));
31         await whizlab_client.SendAsync(whizlab_batch);
32         Console.WriteLine("Sending Message - {0}", obj.ToString());
33     }
34 }
```



# Azure Logic Apps

## Introduction

### Parts

- Triggers – Some of the connectors like Azure BLOB storage have the facility to trigger the Azure Logic App
- Example - If an object is placed in Azure BLOB storage, it will trigger an Azure Logic App Workflow.
- Each trigger to the Azure Logic App will create a new instance of the Workflow.



# Azure Logic Apps

## Introduction

### Parts

- Action – What to do when an event is triggered.
- Do you want to send an email to the IT administrator on the event?
- Do you want to call an Azure Function?

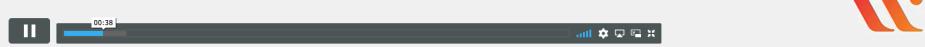


## Lab – Azure Logic Apps

### Lab – Azure Logic Apps

#### What are Azure Logic Apps

- This is a cloud service that helps you schedule, automate and orchestrate tasks , business processes and workflows.
- It helps you integrate applications, data, systems and services across your enterprise.
- This workflow system is completely serverless.
- This service has integration with other Azure services and third party services as well.

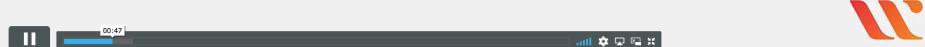


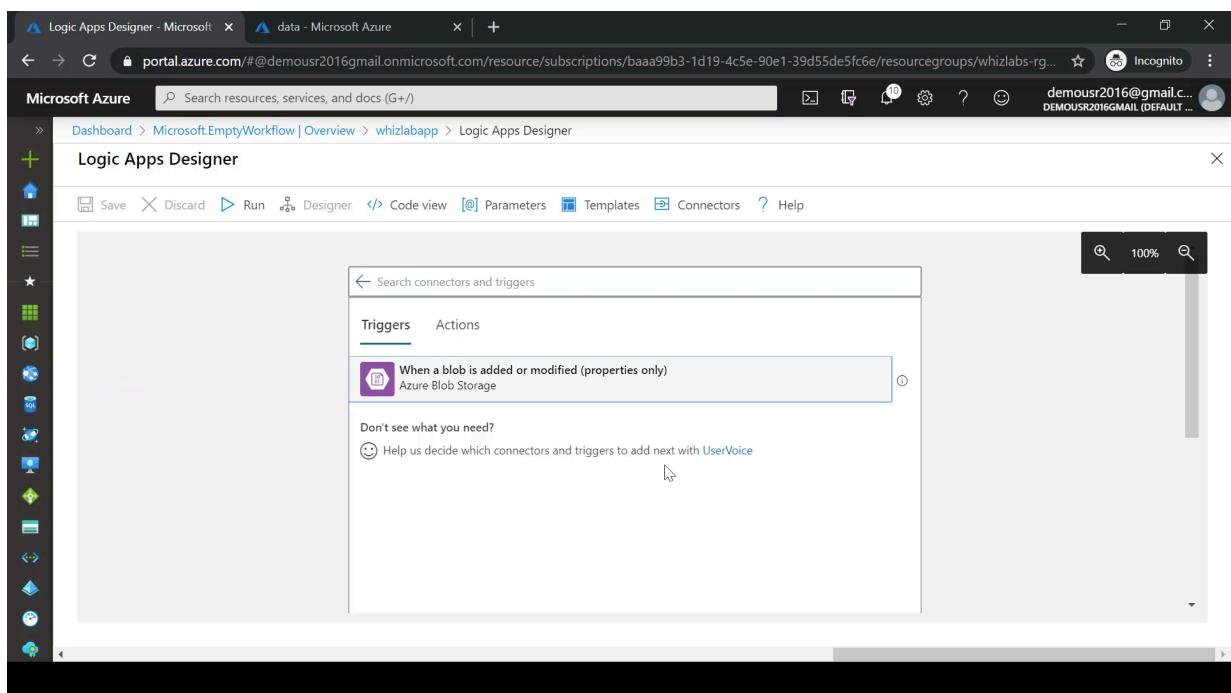
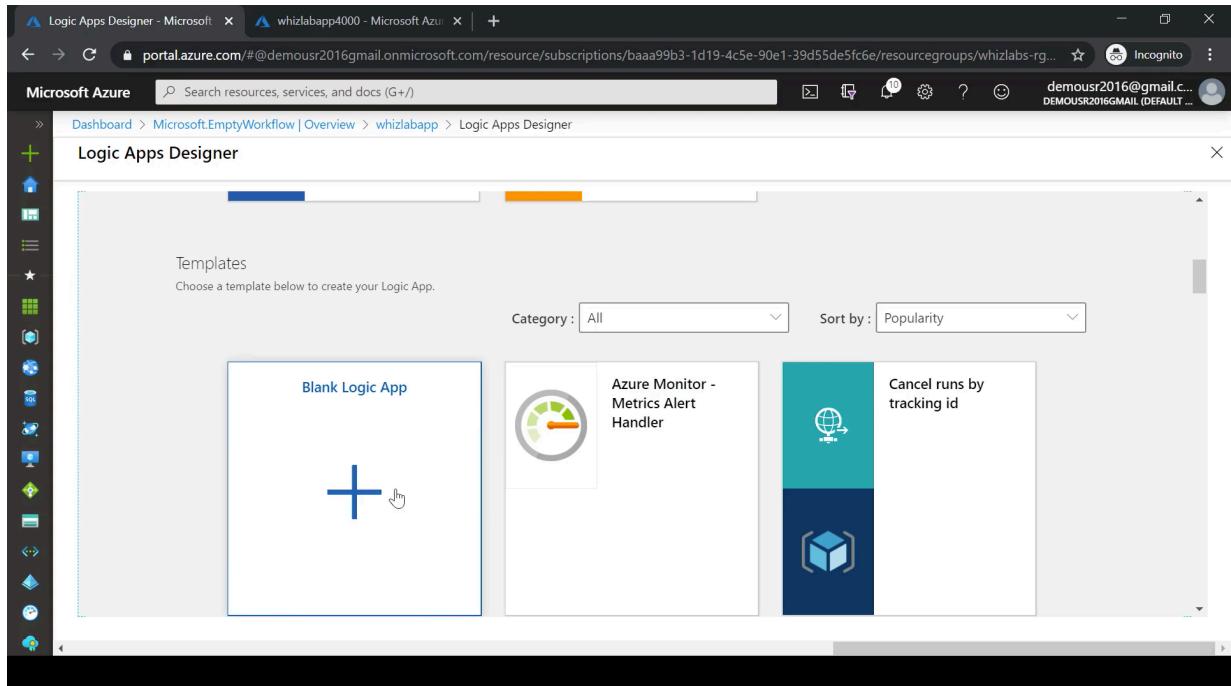
## Lab – Azure Logic Apps

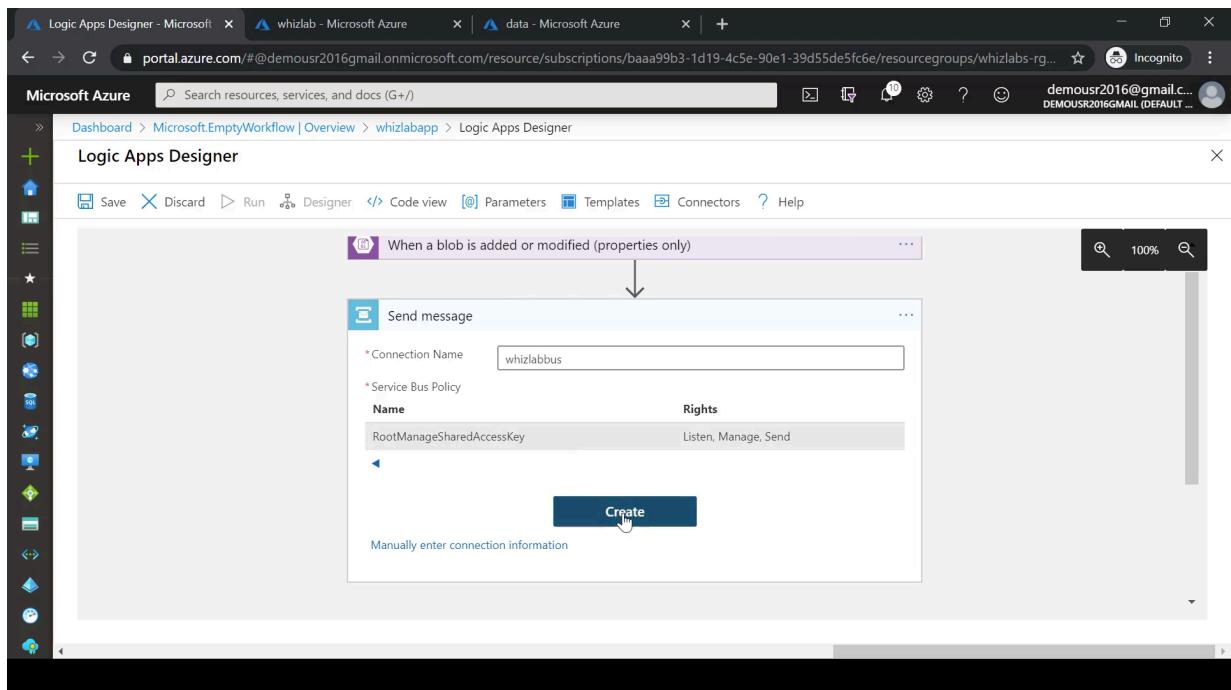
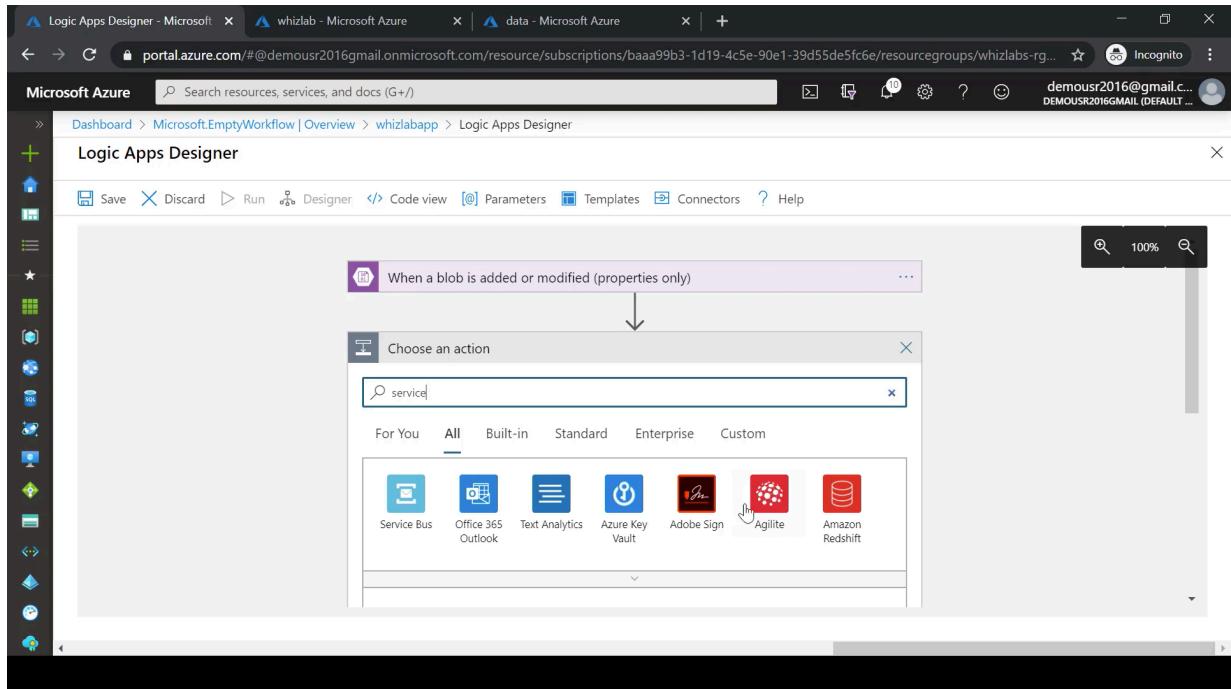
### Lab – Azure Logic Apps

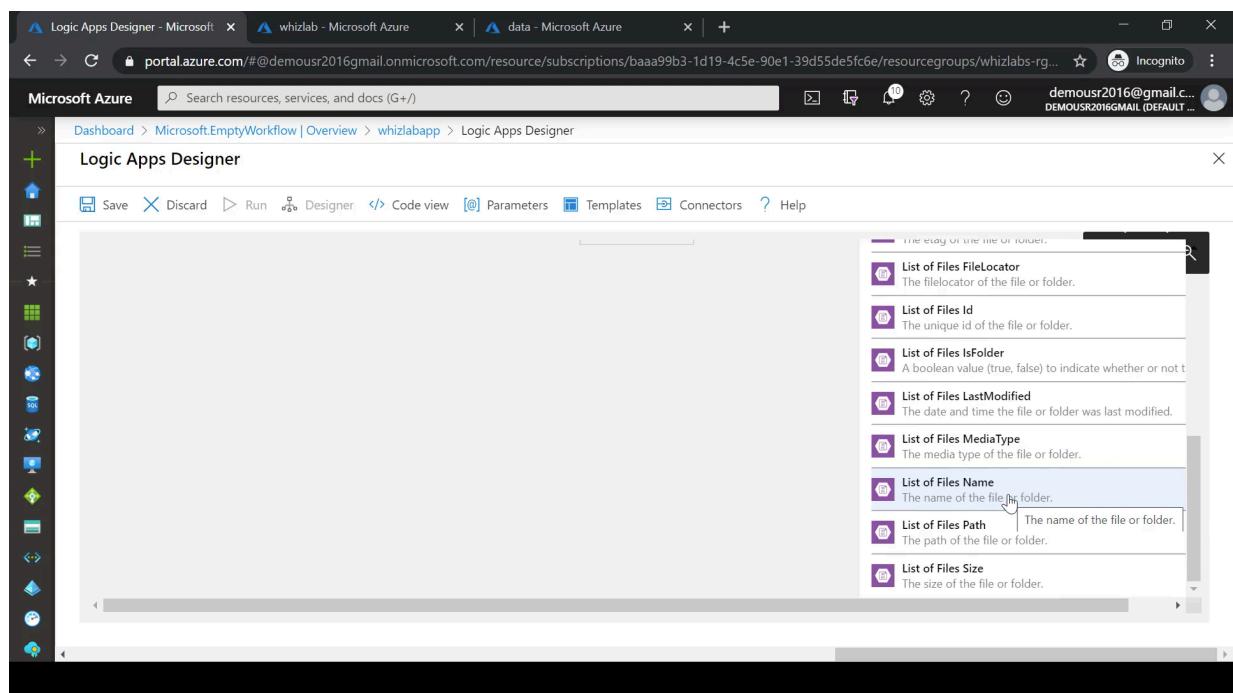
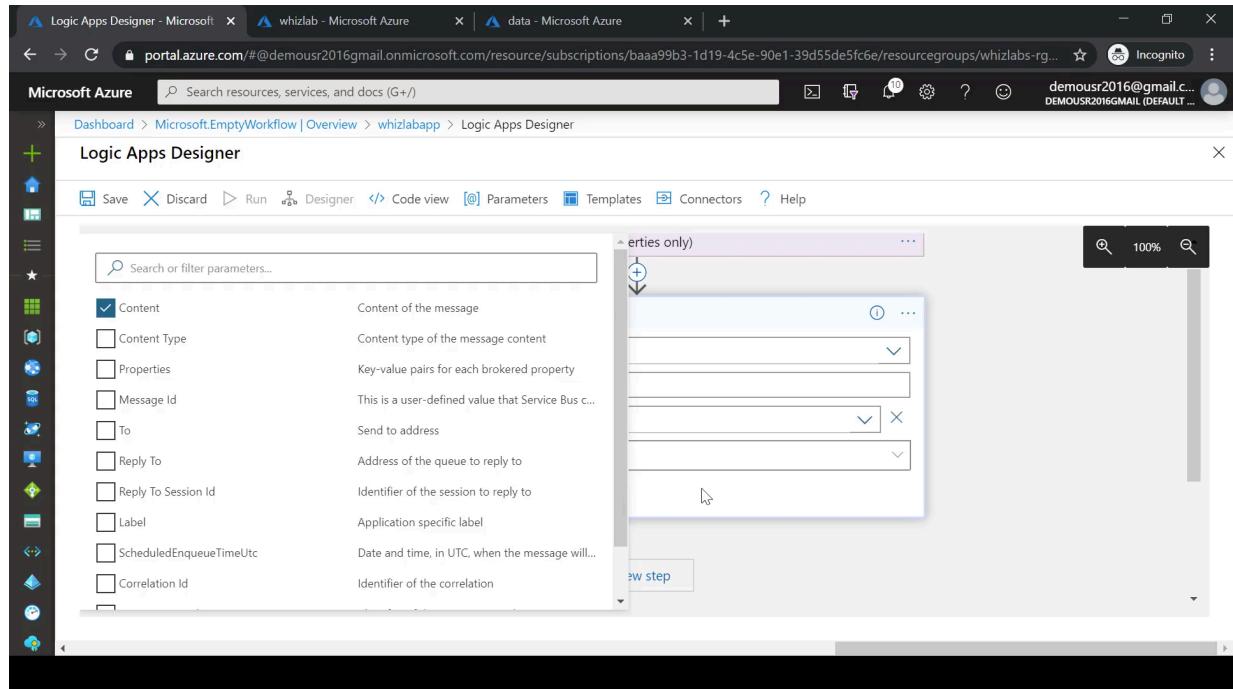
#### What are Azure Logic Apps

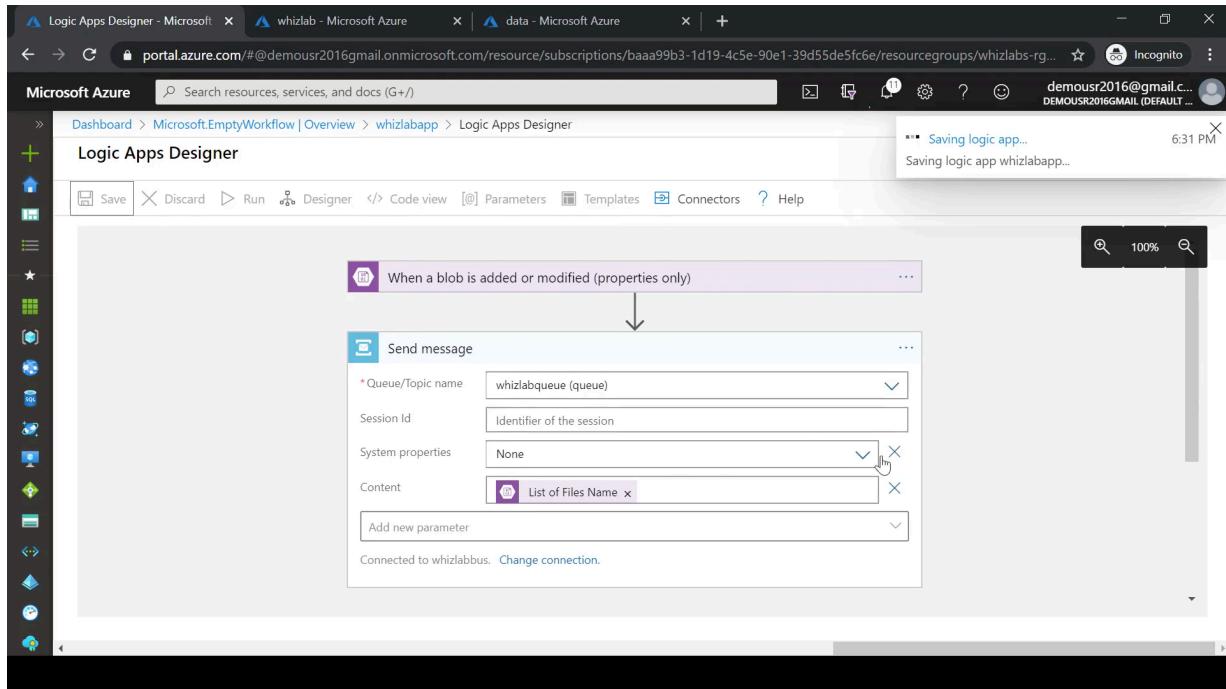
- This is a cloud service that helps you schedule, automate and orchestrate tasks , business processes and workflows.
- It helps you integrate applications, data, systems and services across your enterprise.
- This workflow system is completely serverless.
- This service has integration with other Azure services and third party services as well.
- With Azure Logic Apps , you get a visual designer to help you design the workflow.











## Lab – Azure Notification Hubs

How to work with Azure Notification Hubs

What are notification hubs and what we are going to do in our lab

- Azure Notification Hubs can be used to enable and send notifications to multiple platforms – iOS, Android, Windows etc.
- We are going to create an Azure Notification Hub.



