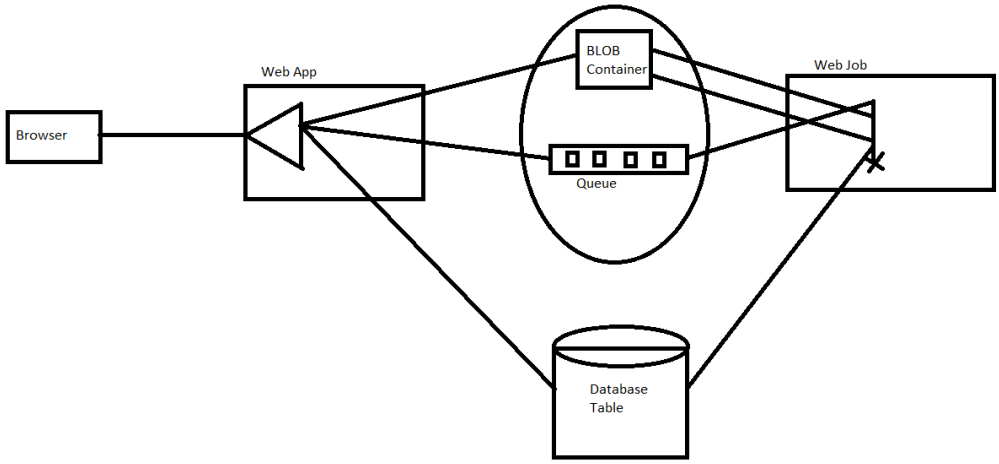
Agenda: Implement Azure Functions

- About Azure Functions.
- Azure Webjobs vs Azure Functions
- Create and Configure an Azure Function
- Create an event processing function
    - Timer Triggered Function
    - Blog Storage Triggered Function
- Implement an Azure-connected functions
    - Adding message to Storage Queue
    - Connecting to SQL Database
- Creating Azure Functions through Visual Studio
- Azure Durable Functions
- Walkthrough: Creating Image Thumbnail

About Azure Functions

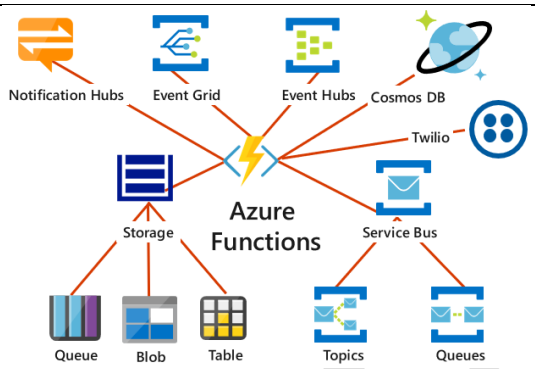


- Azure Functions is a **serverless compute service** that enables you to run code on-demand without having to explicitly provision or manage infrastructure.
- Use Azure Functions to run a script or piece of code in response to a **variety of events**.
- Functions can make development even more productive, and you can use your development language of choice, such as C#, Java, Node.js, Python or PHP.

The following service integrations are supported by Azure Functions:

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA.
http://www.deccansoft.com | http://www.bestdotnettraining.com

Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

• Azure Cosmos DB • Azure Event Hubs • Azure Event Grid • Azure Notification Hubs • Azure Service Bus (queues and topics) • Azure Storage (blob, queues, and tables) • On-premises (using Service Bus)	

The following are a common, *but by no means exhaustive*, set of scenarios for Azure Functions.

If you want to...	**then...**
Build a web API	Implement an endpoint for your web applications using the **HTTP trigger**
Process file uploads	Run code when a file is uploaded or changed in **blob storage.**
Build a serverless workflow	Chain a series of functions together using **durable functions.**
Respond to database changes	Run custom logic when a document is created or updated in Cosmos DB
Run scheduled tasks	Execute code on pre-defined timed intervals (Timer trigger)
Create reliable message queue systems	Process message queues using Queue Storage, Service Bus, or Event Hubs
Analyze IoT data streams	Collect and process data from IoT devices
Process data in real time	Use Functions and SignalR to respond to data in the moment

How much does Functions cost?

Azure Functions has following kinds of pricing plans:

- **Consumption plan**
  - Billing is based on number of executions, execution time, and memory used. Billing is aggregated across all functions within a function app.
  - When your function runs, Azure provides all of the necessary computational resources.
  - You don't have to worry about resource management, and you only pay for the time that your code runs.
  - Your app scales additional instances of the Functions host when needed to handle load, and scaled down when code stops running.
  - Every execution can use max up to **1.5GB Memory** and **One CPU.**

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA.
http://www.deccansoft.com | http://www.bestdotnettraining.com

Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

- **App Service plan**
  - Run your functions just like your web, mobile, and API apps.
  - When you are already using App Service for your other applications, you can run your functions on the same plan at no additional cost.
- **Premium plan**
  - Premium plan provides features, such as premium compute instances(one core, two core, and four core instances), Predictable Pricing, the ability to keep instances warm indefinitely, Unlimited execution duration, and VNet connectivity.
  - In the premium plan, your plan size will determine the available memory and CPU for all apps in that plan on that instance.

Function app timeout duration (in minutes):

Plan	Runtime Version	Default	Maximum
Consumption	1.x	5	10
Consumption	2.x	5	10
Consumption	3.x (preview)	5	10
App Service	1.x	Unlimited	Unlimited
App Service	2.x	30	Unlimited
App Service	3.x (preview)	30	Unlimited

Note: Regardless of the function app timeout setting, **230 seconds** is the maximum amount of time that an **HTTP triggered** function can take to respond to a request. This is because of the default idle timeout of Azure Load Balancer.

https://docs.microsoft.com/en-us/azure/azure-functions/functions-scale

Go to FunctionApp → Function app settings → Edit **host.json**

```
// Set functionTimeout to 10 minutes
{
  "functionTimeout": "00:10:00"
}
```

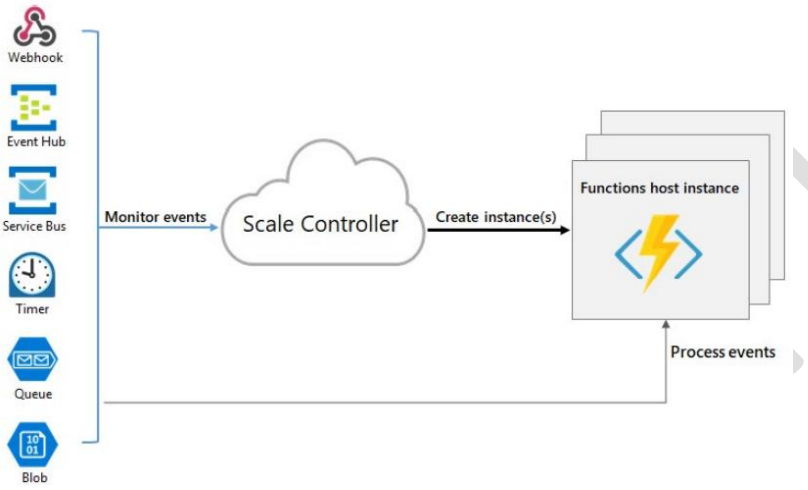
Runtime Scaling:

- Azure Functions uses a component called the **scale controller** to monitor the rate of events and determine whether to scale out or scale in. For example, when you're using an Azure Queue storage trigger, it scales based on the queue length and the age of the oldest queue message.

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA.
http://www.deccansoft.com | http://www.bestdotnettraining.com

Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

- When the function app is scaled out, additional resources are allocated to run **multiple** instances of the **Azure Functions host**. Conversely, as compute demand is reduced, the scale controller removes function host instances. The number of instances is eventually scaled down to zero when no functions are running within a function app.



Scaling can vary on a number of factors, and scale differently based on the trigger and language selected. There are a few intricacies of scaling behaviors to be aware of:

- A single function app only scales up to a **maximum of 200 instances**. A single instance may process more than one message or request at a time though, so there isn't a set limit on number of concurrent executions.
- For **HTTP triggers**, new instances will only be allocated at most once every **1 second**.
- For **non-HTTP triggers**, new instances will only be allocated at most once every **30 seconds**.

Functions vs. WebJobs

We can discuss Azure Functions and Azure App Service WebJobs together because they are both code-first integration services and designed for developers. They enable you to run a script or a piece of code in response to various events, such as new Storage Blobs or a WebHook request.

Here are their similarities:

- Both are built on Azure App Service and enjoy features such as source control, authentication, and monitoring.
- Both are developer-focused services.
- Both support standard scripting and programming languages.
- Both have NuGet and NPM support.

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA.
http://www.deccansoft.com | http://www.bestdotnettraining.com

Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

Functions is the **natural evolution** of WebJobs in that it takes the best things about WebJobs and improves upon them.

The improvements include:

- Streamlined dev, test, and run of code, directly in the browser.
- Built-in integration with more Azure services and 3rd-party services like GitHub WebHooks.
- Pay-per-use, no need to pay for an App Service plan.
- For existing customers of App Service, running on App Service plan still possible (to take advantage of under-utilized resources).
- Automatic, dynamic scaling.
- Integration with Logic Apps.

Creating an Azure Function in Portal

Create a Function App

1. +Create a resource → Compute → Function App
2. App name = DssDemoFunctions, Hosting Plan: Consumption Plan, Runtime Stack = .NET Core, Storage Account: <Create New> → Create

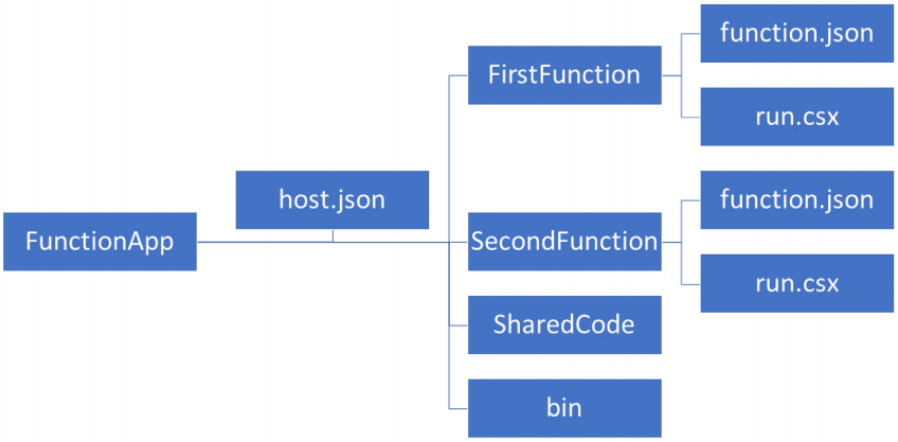
Create a Function

3. Function App → Click Functions + → **HTTP trigger**,
4. Name = SayHello,
5. Authorization level = Function / Anonymous / Admin

Authorization level controls whether the function requires an **API key** and which key to use; Function uses a function key; Admin uses your master key. The function and master keys are found in the **'keys'** management panel on the portal, when your function is selected.

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA.
http://www.deccansoft.com | http://www.bestdotnettraining.com

Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)



Note: The host.json file contains runtime-specific configurations and is in the root folder of the function app. A bin folder contains packages and other library files that the function app requires.

Following is the function auto generated in C#

```
#r "Newtonsoft.Json"

using System.Net;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Primitives;
using Newtonsoft.Json;

public static async Task<IActionResult> Run(HttpRequest req, ILogger log)
{
    log.LogInformation("C# HTTP trigger function processed a request.");

    string name = req.Query["name"];

    string requestBody = await new StreamReader(req.Body).ReadToEndAsync();
    dynamic data = JsonConvert.DeserializeObject(requestBody);
    name = name ?? data?.name;

    return name != null
        ? (ActionResult)new OkObjectResult($"Hello, {name}")
```


Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA.
http://www.deccansoft.com | http://www.bestdotnettraining.com

Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

```
    : new BadRequestObjectResult("Please pass a name on the query string or in the request body");
}
```

Test the function:

1. In your new function, click **</> Get function URL** and copy the **Function URL.**

   ```
   https://dssdemofunc.azurewebsites.net/api/HttpTriggerJS1?code=HaAojLFnYG05Fa0hMtocj7ymoCExasRWW64BaWtbQGx/O9DvSoDv8A==
   ```

2. Paste the URL for the HTTP request into your browser's address bar. Append the query string `&name=<yourname>` to this URL and execute the request.
3. View the function logs at the bottom of the screen (click up arrow at the bottom of the screen).
4. You can make changes (specially to context.res.body) to the function as needed and test the same.
5. You can also test the function from Right Handside panel: Test (expand to test)

Timer Triggered Function

1. Function App → Click Functions **+** → Choose a template: **TimerTrigger – C#** → Name your function = TimerTriggerCSharp, Schedule= 0 */1 * * * * (CRON expression that schedules your function to run every minute) → Create
2. If required we can update the **timer schedule**: Integrate tab → {seconds} {minutes} {hour} {day} {month} {dayofweek}
   - To trigger once every hour = 0 0 */1 * * *
   - To trigger once every five minutes: 0 */5 * * * *
   - To trigger once at the top of every hour: "0 0 * * * *
   - To trigger once every two hours: 0 0 */2 * * *
   - To trigger once every hour from 9 AM to 5 PM: 0 0 9-17 * * *
   - To trigger At 9:30 AM every day: 0 30 9 * * *
   - To trigger At 9:30 AM every weekday: 0 30 9 * * 1-5

3. When a timer trigger function is invoked, the timer object is passed into the function. The following JSON is an example representation of the timer object.

```
{
  "Schedule":{    },
  "ScheduleStatus": {
```

```
    "Last":"2016-10-04T10:15:00.012699+00:00",
    "Next":"2016-10-04T10:20:00+00:00"
  },
  "IsPastDue":false
}
```

Example:

```
using System;
public static void Run(TimerInfo myTimer, TraceWriter log)
{
  if (myTimer.IsPastDue)
  {
    log.Info("Timer is running late!");
  }
  log.Info(myTimer.ScheduleStatus.Last.ToString());
  log.Info($"C# Timer trigger function executed at: {DateTime.Now}");
}
```

Blob Trigger

1. Function App → Click Functions **+** → Choose a template: **BlobTrigger** for your desired language. **Name your function**, Path and Storage account connection and then click **Create**.
2. Click **Integrate**,
    a. Blob parameter **name**: The variable name used in function code for the blob.
    b. **Path**: A path that specifies the container to monitor, and optionally a blob name pattern = **"mycontainer/{name}"**

    Examples of Path:

    - **input/original-{name}**: This path would find a blob named ***original-Blob1.txt*** in the ***input*** container, and the value of the name variable in function code would be **Blob1.txt**
    - ***samples/{name}.png***: only *.png* blobs in the *samples* container will trigger the function. If **demo.png** is uploaded into the samples container, value of name parameter would be **demo**
3. Function App → Click Functions **+** → Choose a template: **BlobTrigger** for your desired language. **Name your function**, Path and Storage account connection and then click **Create**.

```
public static void Run(Stream myBlob, string name, ILogger log)
{
   log.LogInformation($"C# Blob trigger function Processed blob\n Name:{name} \n Size: {myBlob.Length} Bytes");
}
```

4. Click **Integrate**,
    a. Blob parameter **name**: The variable name used in function code for the blob.
    b. **Path**: A path that specifies the container to monitor, and optionally a blob name pattern = **"mycontainer/{name}"**
5. Go to Storage Account and Create a container, Name=mycontainer
6. Test the function:
    a. Expand your storage account, **Blob containers**, and **mycontainer**. Click **Upload** and then **Upload files...**.
    b. In the **Upload files** dialog box, click the **Files** field. Browse to a file on your local computer, such as an image file, select it and click **Open** and then **Upload**.
    c. Go back to your function **logs** and verify that the blob has been read.

Queue Trigger - Input and Output Parameter Binding

Go to Storage Account and Create a container, Name=employees and upload files 1.txt and 2.txt (1 and 2 are employee ids when you are posting a json object into queue)

Run.csx

```
using System;

public class Employee
{
   public int Id {get;set;}
   public string Name {get;set;}
}

public static void Run(Employee emp, string empDetails, out string employeeUpdatedDetails, ILogger log)
{
   log.LogInformation($"C# Queue trigger function processed: {emp.Id} - {emp.Name}");
   log.LogInformation(empDetails);
```


Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA.
http://www.deccansoft.com | http://www.bestdotnettraining.com

Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

```
    employeeUpdatedDetails = empDetails + " (Name=" + emp.Name + ")";
}
```

Function.json

```
{
 "bindings": [
  {
   "name": "emp",
   "type": "queueTrigger",
   "direction": "in",
   "queueName": "myqueue-items",
   "connection": "AzureWebJobsStorage"
  },
  {
   "name": "empDetails",
   "direction": "in",
   "type": "blob",
   "path": "employees/{Id}.txt",
   "connection": "AzureWebJobsStorage"
  },
  {
   "name": "employeeUpdatedDetails",
   "direction": "out",
   "type": "blob",
   "path": "employees/{Name}.txt",
   "connection": "AzureWebJobsStorage"
  }
 ]
}
```

Add messages to an Azure Storage queue using Functions

In Azure Functions, input and output bindings provide a declarative way to connect to external service data from your function. In this topic, learn how to update an existing function by adding an output binding that sends messages to Azure Queue storage.

Continue the "**Create a HTTP triggered function"** Function

1. Add an output binding: Click Integrate and **+ New output**, → Azure Queue storage → Select
2. Set Message parameter name: outputQueueItems, Queue name=**myqueue-items**, Storage account connection:<select storage account> → Save
3. **Update the function code → Save**

```
public static async Task<HttpResponseMessage> Run(HttpRequestMessage req, ICollector<string>
outputQueueItems, TraceWriter log)
{
   ....
   outputQueueItems.Add("Name passed to the function: " + name);
   outputQueueItems.Add("Name passed to the function: " + name);
   outputQueueItems.Add("Name passed to the function: " + name);
   log.Info("System time: " + System.DateTime.Now.ToLongTimeString());
   return name = . . .
}
```

4. Test the function
    a. Run the function and Check the logs to make sure that the function succeeded. A new queue named outqueue is created in your Storage account by the Functions runtime when the output binding is first used.
    b. Connect to the storage account queue and verify the new queue and message you added to it.

Connect to SQL Database

1. Create an SQL Database and copy its connection string.
2. Navigate to your function app you created
3. Select **Platform features → Application settings.**
4. Scroll down to **Connection strings** and add a connection string "sqldb_connection"
5. Save
6. In your function app, select the timer-triggered function

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA.
http://www.deccansoft.com | http://www.bestdotnettraining.com

Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

```
#r "System.Configuration"
#r "System.Data"

using System;
using System.Configuration;
using System.Data.SqlClient;

public class Employee
{
   public int Id{get;set;}
   public string Name {get;set;}
}

public static void Run(Employee emp, ILogger log, ExecutionContext context)
{
   var str = Environment.GetEnvironmentVariable("sqldb_connection");
// Note: Functions written in Portal cannot read data from ConnectionStrings Section. They can only read from ApplicationSettings Sections

   log.LogInformation(str);
   using (SqlConnection conn = new SqlConnection(str))
   {
      conn.Open();
      var text = $"insert into Emp values({emp.Id},'{emp.Name}')";
      using (SqlCommand cmd = new SqlCommand(text, conn))
      {
         // Execute the command and log the # rows affected.
         var rows = cmd.ExecuteNonQuery();
         log.LogInformation($"{rows} rows were inserted");
      }
   }
}
```

Azure Function in C# using Command Line Interface

Configure Local Environment

- .NET 6.0 SDK
- Azure Functions Core Tools version 4.x.
- One of the following tools for creating Azure resources:
  - Azure CLI version 2.4 or later.
  - The Azure Az PowerShell module version 5.9.0 or later.

Steps:

1. Create a local function project

```
func init LocalFunctionProj --dotnet
cd LocalFunctionProj
```

2. Add a function to your project.

```
func new --name HttpExample
```

3. Run the function locally

```
func start
```

4. Test the function

   Open the HTTPTrigger URL in browser.

Publishing an Function App

5. Create the function app in Azure (use cloud shell)

```
az functionapp create --resource-group <ResourceGroup Name> --consumption-plan-location <Location> --runtime dotnet --functions-version 4 --name <FunctionAppName> --storage-account <StorageAccount>

az login --tenant sandeepsonideccansoft.onmicrosoft.com
az account list
az account set --subscription <SubscriptionName>
az account show
az functionapp create --resource-group Demo-rg --consumption-plan-location eastus --runtime dotnet --functions-version 4 --name dssfuncdemoapp1 --storage-account dssdemostorage321
```

6. At command prompt: Deploy the function project to Azure

```
func azure functionapp publish <FunctionAppName>
```

7. Run the following command to view near real-time streaming logs:

```
func azure functionapp logstream <FunctionAppName>
```

8. Test the function

   Open the HTTPTrigger URL in browser.

Same Example as what we did in Portal

```
using System;
using Microsoft.Azure.Functions.Worker;
using Microsoft.Extensions.Logging;

namespace LocalFunctionProj
{
  public class Employee
  {
    public int Id {get;set;}
    public string Name {get;set;}
  }

  public class QueueTriggerDemo
  {
    private readonly ILogger _logger;

    public QueueTriggerDemo(ILoggerFactory loggerFactory)
    {
      _logger = loggerFactory.CreateLogger<QueueTriggerDemo>();
    }

    [Function("QueueTriggerDemo")]
    [BlobOutput(blobPath: "employees/{Name}.txt")]
    public string Run([QueueTrigger("demoqueue", Connection = "AzureWebJobsStorage")] Employee emp,
            [BlobInput(blobPath: "employees/{Id}.txt")]string resumeContent)
    {
      _logger.LogInformation($"C# Queue trigger function processed: {emp.Id} - {emp.Name}");
```


Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA.
http://www.deccansoft.com | http://www.bestdotnettraining.com

Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

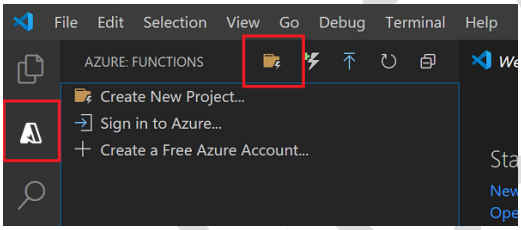
```
        _logger.LogInformation("Content: " + resumeContent);
        return resumeContent + " - Updated";
    }
  }
}
```

Post the message into the queue

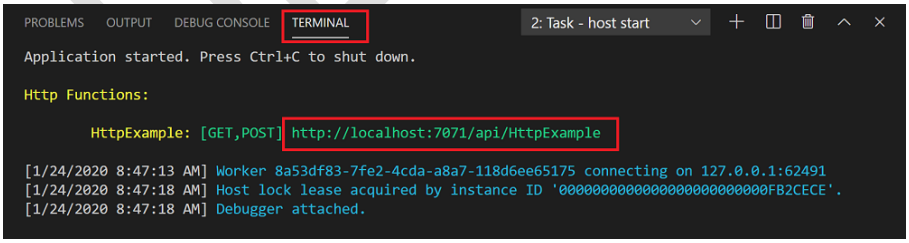
```
{
  "Id":1,"Name":"Sandeep"
}
```

1. Azure Function using VS Code

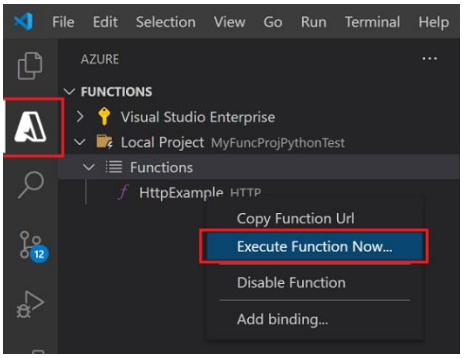
2. In VSCode add C# extension: https://marketplace.visualstudio.com/items?itemName=ms-dotnettools.csharp
3. In VSCode add Azure Functions extension: https://marketplace.visualstudio.com/items?itemName=ms-azuretools.vscode-azurefunctions
4. Choose the Azure icon in the Activity bar, then in the **Azure: Functions** area, select the **Create new project...** icon.



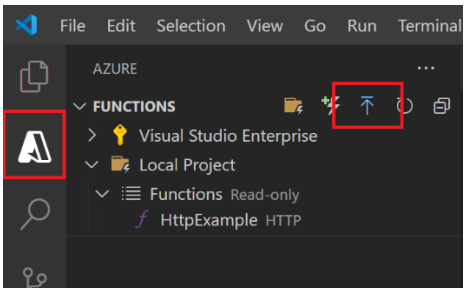
5. To call your function, **press F5 to start** the function app project. Output from Core Tools is displayed in the **Terminal** panel.



With Core Tools running, go to the **Azure: Functions** area. Under **Functions**, expand **Local Project** > **Functions**. Right-click (Windows) or Ctrl - click (macOS) the HttpExample function and choose **Execute Function Now...**.



6. Choose the Azure icon in the Activity bar, then in the **Azure: Functions** area, choose the **Deploy to function app...** button.

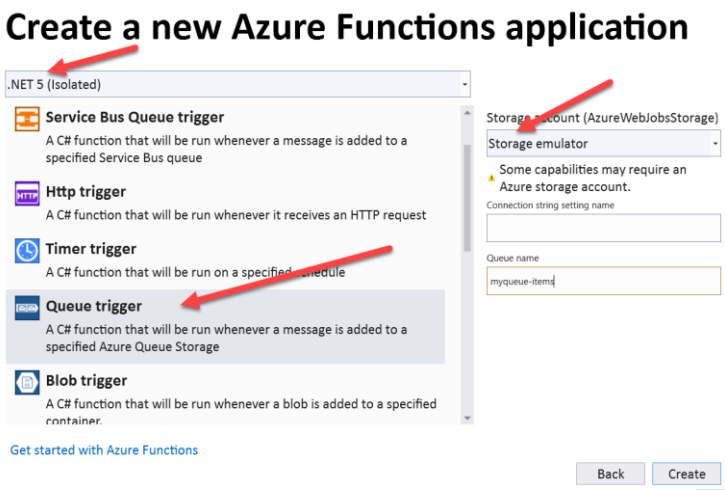


Create a Java Function in Azure using VS Code:

https://docs.microsoft.com/en-us/azure/azure-functions/create-first-function-vs-code-java

Writing Function in Visual Studio.NET

1. In Azure Portal Create Storage Account
2. Create a New Project
    - VS.NET → File → New Project → Cloud → Azure Functions, Name="DemoFunctionApp"



3. Edit the function as below

```
using System;
using Microsoft.Azure.Functions.Worker;
using Microsoft.Extensions.Logging;

namespace DemoFunctionApp
{
    public class Employee
    {
        public int Id { get; set; }
        public string Name { get; set; }
    }

    public static class StorageQueueDemo
    {
        [Function("StorageQueueDemo")]
        [BlobOutput(blobPath: "employees/{Name}.txt")]
        public static string Run(
            [QueueTrigger("myqueue-items", Connection = "")] Employee emp,
            [BlobInput(blobPath: "employees/{Id}.txt")] string empDetails,
            FunctionContext executionContext)
        {
```


Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA.
http://www.deccansoft.com | http://www.bestdotnettraining.com

Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

```
            var logger = executionContext.GetLogger("StorageQueueDemo");
            logger.LogInformation($"C# Queue trigger function processed: {emp.Id} - {emp.Name}");
            logger.LogInformation(empDetails);
            logger.LogInformation("This is v2");
            return empDetails + " (Name=" + emp.Name + ")";
        }
    }
}
```

4. Run the function locally.
5. Goto Azure Portal and post the message in Queue by name "myqueue-items".
6. See that the new blob as created in employees container.
7. Publish the Azure Function

Create a Java Function in Azure using VS Code:

https://docs.microsoft.com/en-us/azure/azure-functions/create-first-function-vs-code-java

Employee.java

```
package com.sandeep;


public class Employee
{
   private String name;
   public void setName(String name)
   {
      this.name = name;
   }
   public String getName()
   {
      return name;
   }
   private String id;
   public void setId(String id)
```


Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA.
http://www.deccansoft.com | http://www.bestdotnettraining.com

Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

```
    {
        this.id = id;
    }
    public String getId()
    {
        return id;
    }
}
```

QueueTriggerJava1.Java

```
package com.sandeep;

import com.microsoft.azure.functions.annotation.*;
import com.microsoft.azure.functions.*;

/**
 * Azure Functions with Azure Storage Queue trigger.
 */
public class QueueTriggerJava1 {
    @FunctionName("QueueTriggerJava1")
    @BlobOutput(name = "blobOutput", dataType = "String", path = "emps/{name}.txt", connection =
"AzureWebJobsStorage")
    public String run(
            @QueueTrigger(name = "emp", queueName = "demo-queue", connection = "AzureWebJobsStorage")
Employee emp,
            @BlobInput(name = "blobInput", dataType = "String", path = "emps/{id}.txt", connection =
"AzureWebJobsStorage") String blobInput,
            final ExecutionContext context) {
        context.getLogger().info("Java Queue trigger function processed a message: " + emp.getId() + " - " +
emp.getName() + " - " + blobInput);
        return "This is file of " + emp.getName();
    }
```


Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA.
http://www.deccansoft.com | http://www.bestdotnettraining.com

Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

```
}
```

Azure Durable Functions

Durable Functions are an **extension** of Azure Functions that lets you write **stateful functions** in a serverless environment.

The extension lets you define **stateful workflows** by writing **orchestrator functions** and **stateful entities** by writing **entity functions** using the Azure Functions programming model.

Behind the scenes, the extension manages state, checkpoints, and restarts for you, allowing you to focus on your business logic.

Benefits:

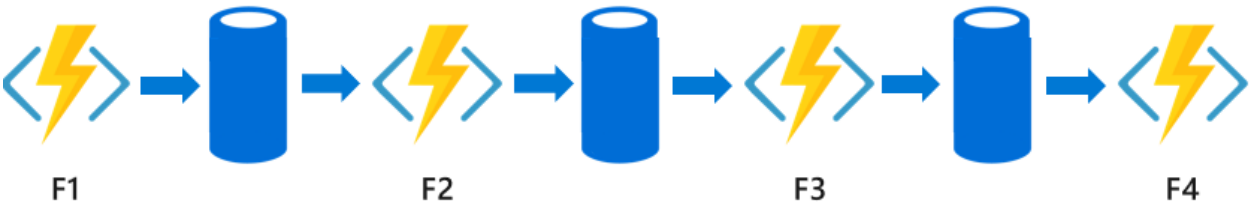
- You can define your workflows in code. Unline in Logic Apps No JSON schemas or designers are needed.
- Other functions can be called both **synchronously and asynchronously**. Output from called functions can be saved to local variables.
- Progress is automatically checkpointed when the function awaits. Local state is never lost when the process recycles or the VM reboots.

Durable Function scenario

1. Function Chaining
2. Fan-out/fan-in
3. Async HTTP APIs
4. Monitoring
5. Human Interaction
6. Aggregator (stateful entities)

Durable Function scenario – Chaining

Function chaining refers executing a sequence of functions in a particular order. Often, the output of one function needs to be applied to the input of another function



Programming in VS.NET

1. Visual Studio → New → Project → Visual C# → Cloud → Select Azure Function, Name = "MyDurationFunctionDemo"
2. Select Version = Azure Function 2.X (.NET Core), **Template** = **Empty**, Storage Account = <Azure Storage Account> → OK
3. Right click on Project → Add → New Azure Function → Select **Durable Functions Orchestration** → OK
4. A new durable function is added to the app
    - Method Name = RunOrchestrator, FunctionName = <file-name>
    - Method Name = SayHello, FunctionName = <file-name>_Hello
    - Method Name = HttpStart, FunctionName = <file-name>_HttpStart

In this example, the values `F1`, `F2`, `F3`, and `F4` are the names of other functions in the same function app. You can implement control flow by using normal imperative coding constructs.

```
using System.Collections.Generic;
using System.Net.Http;
using System.Threading.Tasks;
using Microsoft.Azure.WebJobs;
using Microsoft.Azure.WebJobs.Extensions.DurableTask;
using Microsoft.Azure.WebJobs.Extensions.Http;
using Microsoft.Azure.WebJobs.Host;
using Microsoft.Extensions.Logging;

namespace MyVSFuncDemoApp
{
  public static class GetCountriesTotalPopulation
  {
    [FunctionName("GetCountriesTotalPopulation")]
    public static async Task<int> RunOrchestrator(
      [OrchestrationTrigger] IDurableOrchestrationContext context)
    {
      var countries = new List<string>();

      countries.Add(await context.CallActivityAsync<string>("GetCountry", "Tokyo"));
      countries.Add(await context.CallActivityAsync<string>("GetCountry", "Seattle"));
      countries.Add(await context.CallActivityAsync<string>("GetCountry", "London"));
```

```
    int totalPopulation = await context.CallActivityAsync<int>("GetTotalPopulation", countries);
    return totalPopulation;
}

[FunctionName("GetCountry")]
public static string GetCountry([ActivityTrigger] string city, ILogger log)
{
    log.LogInformation($"City: {city}.");
    System.Threading.Thread.Sleep(5000);
    if (city == "Tokyo")
        return "Japan";
    else if (city == "Seattle")
        return "Australia";
    else
        return "GB";
}

[FunctionName("GetTotalPopulation")]
public static int GetTotalPopulation([ActivityTrigger] List<string> countries, ILogger log)
{
    int population = 0;
    foreach (var country in countries)
    {
        population += 1000;
    }
    return population;
}

[FunctionName("GetCountriesTotalPopulationTrigger")]
public static async Task<HttpResponseMessage> HttpStart(
    [HttpTrigger(AuthorizationLevel.Anonymous, "get", "post")] HttpRequestMessage req,
    [DurableClient] IDurableOrchestrationClient starter,
```

```
        ILogger log)
    {
        // Function input comes from the request content.
        string instanceId = await starter.StartNewAsync("GetCountriesTotalPopulation", null);
        log.LogInformation($"Started orchestration with ID = '{instanceId}'.");
        return starter.CreateCheckStatusResponse(req, instanceId);
    }
  }
}
```

Test the function locally:

5. Run the Project (Ctrl + F5). You might get the Windows Security Alert → Check both private and public → Allow access

6. Publish the project to Azure

Note: When you click **Run from a package file (recommended)**, your function app will be deployed using Zip Deploy with Run-From-Package mode enabled. This is the recommended way of running your functions, and will result in better performance.

7. After the deployment is complete, make a note of the **Site URL** value, which is the address of your function app in Azure.
8. **Test the function in Azure**: http://<APP_NAME>.azurewebsites.net/api/<FUNCTION_NAME>_HttpStart

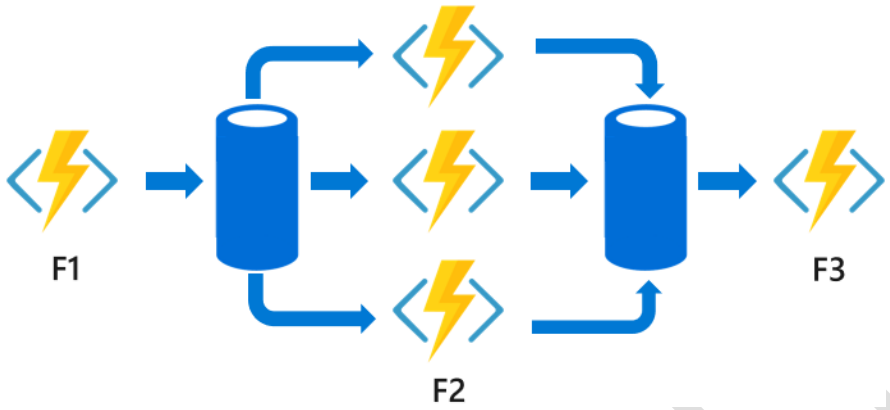
Durable Function scenario - Fan-out/fan-in

Fan-out/fan-in refers to the pattern of executing multiple functions in parallel, and then waiting for all to finish. Often, some aggregation work is done on results returned from the functions.

With normal functions, fanning out can be done by having the function send multiple messages to a queue. However, fanning back in is much more challenging. You'd have to write code to track when the queue-triggered functions end and store function outputs. The Durable Functions extension handles this pattern with relatively simple code.

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA.
http://www.deccansoft.com | http://www.bestdotnettraining.com

Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)



Example 2: Right click on Project → Add → New Azure Function → Select **Durable Functions Orchestration** → OK

The orchestrator function does the following:

1. Takes a rootDirectory value as an input parameter.
2. Calls a function to get a recursive list of files under rootDirectory.
3. Makes multiple parallel function calls to upload each file into Azure Blob Storage.
4. Waits for all uploads to complete.
5. Returns the sum total bytes that were uploaded to Azure Blob Storage.

Following functions are used in sample below:

- BackupSiteContent
- BackupSiteContent_GetFileList
- BackupSiteContent_CopyFileToBlob

```
using Microsoft.Azure.WebJobs;
using Microsoft.Azure.WebJobs.Extensions.Http;
using Microsoft.Extensions.Logging;
using Microsoft.WindowsAzure.Storage.Blob;
using System.IO;
using System.Linq;
using System.Net.Http;
using System.Threading.Tasks;
```

```
public static class DurableFunctionDemo
{
   [FunctionName("BackupSiteContent")]
   public static async Task<long> RunOrchestrator([OrchestrationTrigger] DurableOrchestrationContext
backupContext)
   {
      string rootDirectory = backupContext.GetInput<string>()?.Trim();
      if (string.IsNullOrEmpty(rootDirectory))
      {
         rootDirectory = Directory.GetParent(typeof(DurableFunctionDemo).Assembly.Location).FullName;
      }
      string[] files = await
backupContext.CallActivityAsync<string[]>("BackupSiteContent_GetFileList",rootDirectory);
      var parallelTasks = new Task<long>[files.Length];
//Fan-Out
      for (int i = 0; i < files.Length; i++)
      {
         parallelTasks[i] = backupContext.CallActivityAsync<long>("BackupSiteContent _CopyFileToBlob",files[i]);
      }
//Fan-In
      await Task.WhenAll(parallelTasks);
      long totalBytes = parallelTasks.Sum(t => t.Result);
      return totalBytes;
   }

   [FunctionName("BackupSiteContent_GetFileList")]
   public static string[] GetFileList([ActivityTrigger] string rootDirectory,ILogger log)
   {
      log.LogInformation($"Searching for files under '{rootDirectory}'...");
      string[] files = Directory.GetFiles(rootDirectory, "*", SearchOption.AllDirectories);
      log.LogInformation($"Found {files.Length} file(s) under {rootDirectory}.");
      return files;
   }
```

```
  [FunctionName("BackupSiteContent_CopyFileToBlob")]
  public static async Task<long> CopyFileToBlob([ActivityTrigger] string filePath,Binder binder,ILogger log)
  {
    long byteCount = new FileInfo(filePath).Length;
    // strip the drive letter prefix and convert to forward slashes
    string blobPath = filePath
      .Substring(Path.GetPathRoot(filePath).Length)
      .Replace('\\', '/');
    string outputLocation = $"backups/{blobPath}";
    log.LogInformation($"Copying '{filePath}' to '{outputLocation}'. Total bytes = {byteCount}.");
    // copy the file contents into a blob
    using (Stream source = File.Open(filePath, FileMode.Open, FileAccess.Read, FileShare.Read))
    using (Stream destination = await binder.BindAsync<CloudBlobStream>(new BlobAttribute(outputLocation,
FileAccess.Write)))
    {
      await source.CopyToAsync(destination);
    }
    return byteCount;
  }

  [FunctionName("BackupSiteContent_HttpStart")]
  public static async Task<HttpResponseMessage> HttpStart(
    [HttpTrigger(AuthorizationLevel.Anonymous, "get", "post")]HttpRequestMessage req,
    [OrchestrationClient] DurableOrchestrationClient starter,
    ILogger log)
  {
    // Function input comes from the request content.
    string instanceId = await starter.StartNewAsync("BackupSiteContent ", "D:\\Temp\\Web1");
    log.LogInformation($"Started orchestration with ID = '{instanceId}'.");
    return starter.CreateCheckStatusResponse(req, instanceId);
  }
}
```

Note:

- The automatic **checkpointing** that happens at the await call on Task.WhenAll ensures that a potential midway crash or reboot doesn't require restarting an already completed task.
- In rare circumstances, it's possible that a crash could happen in the window after an activity function completes but before its completion is saved into the orchestration history. If this happens, the activity function would re-run from the beginning after the process recovers.
- You might be wondering why you couldn't just put this code directly into the orchestrator function. You could, but this would break one of the fundamental rules of orchestrator functions, which is that they should never do I/O, including local file system access.

Run the Orchestration by sending HTTP Request to HttpStart method

```
http://localhost:7071/api/BackupSiteContent_HttpStart
```

From the URL, copy the value of "statusQueryGetUri" and open the same in browser and see the value of "output".

OR

You can as well start the orchestration directly by sending the following HTTP POST request.

```
POST http://localhost:7071/orchestrators/BackupSiteContent
Content-Type: application/json
Content-Length: 20

"D:\\home\\LogFiles"
```

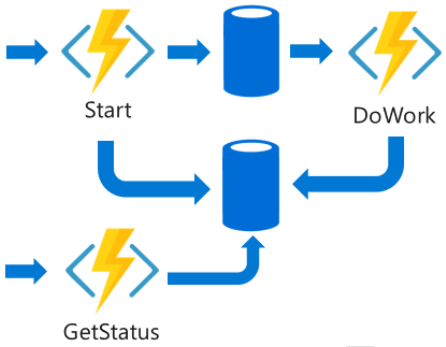
Complete example with explanation is here: https://docs.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-cloud-backup

Durable Function scenario - Async HTTP APIs

The third pattern is all about the problem of coordinating the state of long-running operations with external clients. A common way to implement this pattern is by having the long-running action triggered by an HTTP call, and then redirecting the client to a status endpoint that they can poll to learn when the operation completes.

Durable Functions provides built-in APIs that simplify the code you write for interacting with long-running function

executions. After an instance is started, the extension exposes webhook HTTP APIs that query the Orchestrator function status.



The following example shows the REST commands to start an Orchestrator and to query its status.

> curl -X POST http://localhost:7071/api/orchestrators/ChainingExample -H "Content-Length: 0" -i

HTTP/1.1 202 Accepted Content-Type: application/json

{ "id":"**428fff4c9331491485c44229c297ea47** ", ... }

> curl http://localhost:7071/runtime/webhooks/durabletask/instances/**428fff4c9331491485c44229c297ea47** -i

HTTP/1.1 **202** Accepted Content-Type: application/jso

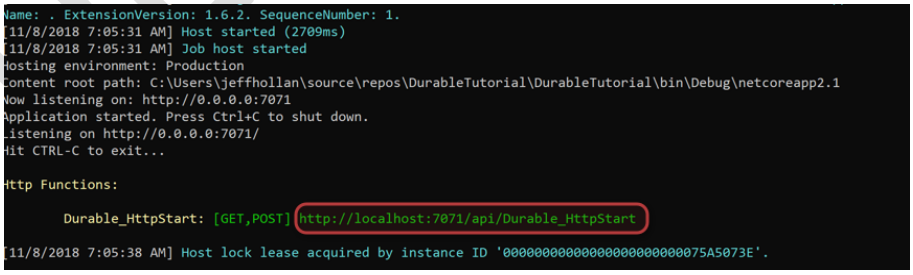
{ "runtimeStatus":"**Running**", "lastUpdatedTime":"2017-03-16T21:20:47Z", ... }

> curl http://localhost:7071/runtime/webhooks/durabletask/instances/**428fff4c9331491485c44229c297ea47** -i

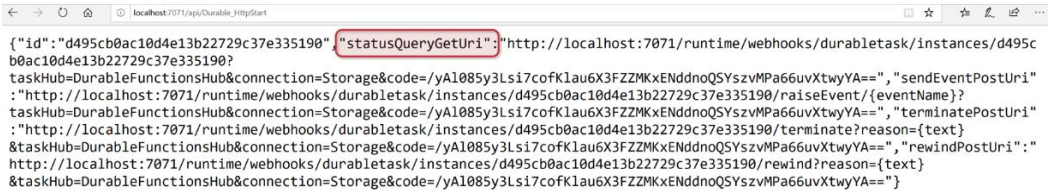
HTTP/1.1 **200** OK Content-Length: 175 Content-Type: application/json

{ "runtimeStatus":"**Completed**", "lastUpdatedTime":"2017-03-16T21:20:57Z", ... **output**: "somedata"}

9. From the output in console window: Copy the **URL of Durable_HttpStart**



10. Paste the URL in browser. The following shows the response in the browser to the local GET request returned by the function:



The response is the initial result from the HTTP function letting us know the durable orchestration has started successfully. It is not yet the end result of the orchestration. The response includes a few useful URLs. For now, let's query the status of the orchestration.

11. Copy the URL value for **statusQueryGetUri** and pasting it in the browser's address bar and execute the request.

    The request will query the orchestration instance for the status. You should get an eventual response that looks like the following. This shows us the instance has completed, and includes the outputs or results of the durable function.


```
{
  "instanceId": "d495cb0ac10d4e13b22729c37e335190",
  "runtimeStatus": "Completed",
  "input": null,
  "customStatus": null,
  "output": [
    "Hello Tokyo!",
    "Hello Seattle!",
    "Hello London!"
  ],
  "createdTime": "2018-11-08T07:07:40Z",
  "lastUpdatedTime": "2018-11-08T07:07:52Z"
}
```


Durable Function scenario - Monitoring

The monitor pattern refers to a flexible recurring process in a workflow—for example, **polling until certain conditions are met**. A regular timer-trigger can address a simple scenario, such as a periodic cleanup job, but its interval is static and managing instance lifetimes becomes complex. **Durable Functions enables flexible recurrence**

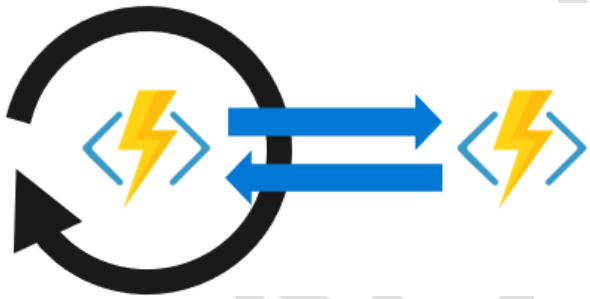
Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA.
http://www.deccansoft.com | http://www.bestdotnettraining.com

Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

intervals, task lifetime management, and the ability to create multiple monitor processes from a single orchestration.

An example would be reversing the earlier async HTTP API scenario. Instead of exposing an endpoint for an external client to monitor a long-running operation, the long-running monitor consumes an external endpoint, waiting for some state change.

Using Durable Functions, multiple monitors that observe arbitrary endpoints can be created in a few lines of code. The monitors can end execution when some condition is met, or be terminated by the DurableOrchestrationClient, and their wait interval can be changed based on some condition (that is, exponential backoff). The code on the next slide implements a basic monitor.



```
using System;
using System.Net.Http;
using System.Threading;
using System.Threading.Tasks;
using Microsoft.Azure.WebJobs;
using Microsoft.Azure.WebJobs.Extensions.DurableTask;
using Microsoft.Azure.WebJobs.Extensions.Http;
using Microsoft.Extensions.Logging;

public static class MonitoringFunction
{
  [FunctionName("MonitorJobStatus")]
  public static async Task Run(
    [OrchestrationTrigger] IDurableOrchestrationContext context)
  {
    int jobId = context.GetInput<int>();
    int pollingInterval = 5; //5 Seconds
```

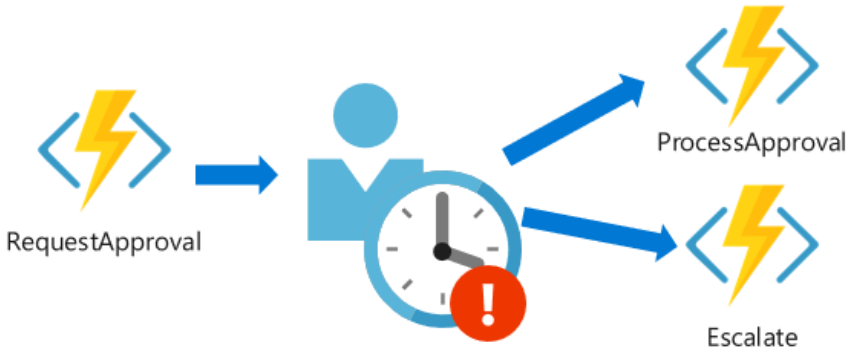
```
    DateTime expiryTime = DateTime.Now.AddMinutes(5);
    while (context.CurrentUtcDateTime < expiryTime)
    {
        var jobStatus = await context.CallActivityAsync<string>("GetJobStatus", jobId);
        if (jobStatus == "Completed")
        {
            // Perform an action when a condition is met.
            await context.CallActivityAsync("SendAlert", "some-guid");
            break;
        }
        // Orchestration sleeps until this time.
        var nextCheck = context.CurrentUtcDateTime.AddSeconds(pollingInterval);
        await context.CreateTimer(nextCheck, CancellationToken.None);
    }
    // Perform more work here, or let the orchestration end.
}

[FunctionName("SendAlert")]
public static void SendAlert([ActivityTrigger] string toAddress, ILogger log)
{
    Console.WriteLine("Send Alert to: " + toAddress);
}

[FunctionName("GetJobStatus")]
public static string GetJobStatus([ActivityTrigger] int jobId, ILogger log)
{
    Console.WriteLine("Getting job status.");
    string status = System.IO.File.ReadAllText("d:\\status.txt");
    Console.WriteLine("Status fetched from file: " + status);
    return status;
}

[FunctionName("MonitorJobStatus_HttpStart")]
```

```
  public static async Task<HttpResponseMessage> HttpStart(
    [HttpTrigger(AuthorizationLevel.Anonymous, "get", "post")] HttpRequestMessage req,
    [DurableClient] IDurableOrchestrationClient starter,
    ILogger log)
  {
    // Function input comes from the request content.
    string instanceId = await starter.StartNewAsync<int>("MonitorJobStatus", null, 100000);

    log.LogInformation($"Started orchestration with ID = '{instanceId}'.");

    return starter.CreateCheckStatusResponse(req, instanceId);
  }
}
```

When a request is received, a new orchestration instance is created for that job ID. The instance polls a status until a condition is met and the loop is exited. A durable timer controls the polling interval. Then, more work can be performed, or the orchestration can end. When nextCheck exceeds expiryTime, the monitor ends.

Durable Function scenario - Human interaction

- Many processes involve some kind of human interaction. The tricky thing about involving humans in an automated process is that people are not always as highly available and responsive as cloud services. Automated processes must allow for this, and they often do so by using timeouts and compensation logic.
- One example of a business process that involves human interaction is an approval process. For example, approval from a manager might be required for an expense report that exceeds a certain amount. If the manager does not approve within 72 hours (maybe the manager went on vacation), an escalation process starts to get the approval from someone else (perhaps the manager's manager).
- This pattern can be implemented by using an Orchestrator function. The Orchestrator would use a durable timer to request approval and escalate in case of timeout. It would wait for an external event, which would be the notification generated by some human interaction.




```
using System;
using System.Net.Http;
using System.Threading;
using System.Threading.Tasks;
using Microsoft.Azure.WebJobs;
using Microsoft.Azure.WebJobs.Extensions.DurableTask;
using Microsoft.Azure.WebJobs.Extensions.Http;
using Microsoft.Extensions.Logging;

public static class ApprovalWorkflowFunction
{
   [FunctionName("ApprovalWorkflow")]
   public static async Task Run([OrchestrationTrigger] IDurableOrchestrationContext context)
   {
      await context.CallActivityAsync("RequestApproval", "sandeepsoni@dss.com");
      using (var timeoutCts = new CancellationTokenSource())
      {
         DateTime dueTime = context.CurrentUtcDateTime.AddHours(72);
         Task durableTimeout = context.CreateTimer(dueTime, timeoutCts.Token);

         Task<bool> approvalEvent = context.WaitForExternalEvent<bool>("ApprovalEvent");
         if (approvalEvent == await Task.WhenAny(approvalEvent, durableTimeout))
         {
            timeoutCts.Cancel();
            await context.CallActivityAsync("ProcessApproval", approvalEvent.Result);
```

```
        }
        else
        {
            await context.CallActivityAsync("Escalate", "Demo Data");
        }
    }
}

[FunctionName("RequestApproval")]
public static void RequestApproval([ActivityTrigger] string toAddress, ILogger log)
{
    Console.WriteLine("Send Email to: " + toAddress);
}

[FunctionName("ProcessApproval")]
public static void ProcessApproval([ActivityTrigger] bool isApproved, ILogger log)
{
    Console.WriteLine("Approval Status: " + isApproved);
}

[FunctionName("Escalate")]
public static void Escalate([ActivityTrigger] string d, ILogger log)
{
    Console.WriteLine("Escalated: " + d);
}
[FunctionName("ApprovalWorkflow_HttpStart")]
public static async Task<HttpResponseMessage> HttpStart(
    [HttpTrigger(AuthorizationLevel.Anonymous, "get", "post")] HttpRequestMessage req,
    [DurableClient] IDurableOrchestrationClient starter,
    ILogger log)
{
    // Function input comes from the request content.
    string instanceId = await starter.StartNewAsync("ApprovalWorkflow", null);
```

```
    log.LogInformation($"Started orchestration with ID = '{instanceId}'.");

    return starter.CreateCheckStatusResponse(req, instanceId);
  }
}
```

The durable timer is created by calling ctx.**CreateTimer**. The notification is received by ctx.**WaitForExternalEvent**. And Task.WhenAny is called to decide whether to escalate (timeout happens first) or process approval (approval is received before time-out).

An external client can deliver the event notification to a waiting Orchestrator function by using either the built-in HTTP APIs

```
curl -d "true" http://localhost:7071/runtime/webhooks/durabletask/instances/{instanceId}/raiseEvent/
ApprovalEvent -H "Content-Type: application/json"
```

OR

An event can also be raised using the durable orchestration client from another function in the same function app:

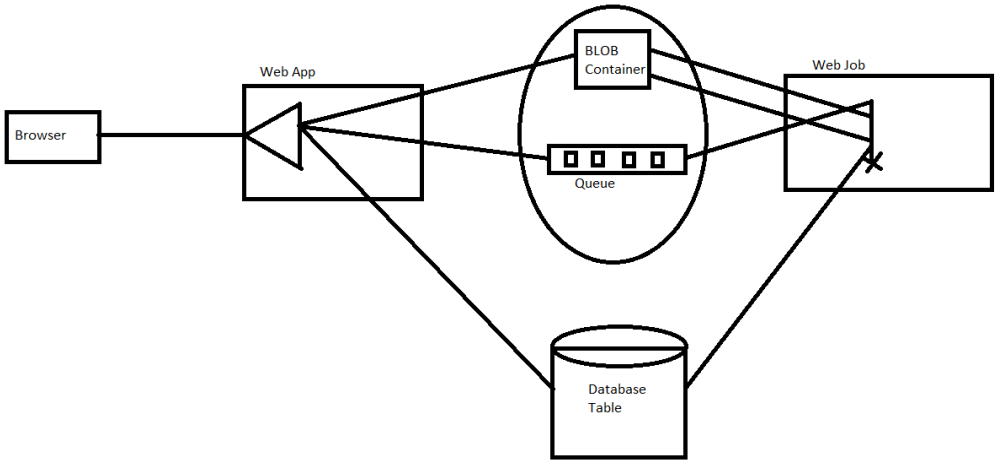
```
[FunctionName("RaiseEventToOrchestration")]
public static async Task Run(
  [HttpTrigger] string instanceId,
  [DurableClient] IDurableOrchestrationClient client)
{
  bool isApproved = true;
  await client.RaiseEventAsync(instanceId, "ApprovalEvent", isApproved);
}
```

Thumbnail Casestudy using Azure Functions

Following .NET and Azure Features will be covered in this walkthrough

1. ASP.NET Core Web Application
2. Entity Framework Code First Approach
3. Azure App Service
4. Storage Service

5. SQL Database Service
6. Azure Serverless Functions



Create a New ASP.NET MVC Application

1. Visual Studio → File → New → Project, Template = ASP.NET Web Application (.NET Framework) → MVC, Change Authentication = No Authentication, Check Host in the cloud → OK
2. Add New Project to Solution for sharing code between web application and web jobs project.
    a. File → Add → New Project → Class Library (.NET Framework) → Name="**CommonDemoLibrary**" → OK
3. Go to Tools → NuGet Package Manager → Manage Nuget Packages for Solution
4. Add reference to **NuGet Package EntityFramework** in the class library project
5. Add the following in Employee.cs (CommonDemoLibrary Project)

```
public class Employee
{
   [Key]
   public int Id { get; set; }
   public string EmpName { get; set; }
   public decimal Salary { get; set; }
   [StringLength(2083)]
   [DisplayName("Full-size Image")]
   public string ImageURL { get; set; }
   [StringLength(2083)]
   [DisplayName("Thumbnail")]
```


Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA.
http://www.deccansoft.com | http://www.bestdotnettraining.com

Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

```
    public string ThumbnailURL { get; set; }
}
public class MyDemoContext : DbContext
{
    public MyDemoContext() : base("name=MyDemoContext") { }
    public DbSet<Employee> Employees { get; set; }
}
public class BlobInformation
{
    public Uri BlobUri { get; set; }
    public string BlobName
    {
        get
        {
            return BlobUri.Segments[BlobUri.Segments.Length - 1];
        }
    }
    public string BlobNameWithoutExtension
    {
        get
        {
            return Path.GetFileNameWithoutExtension(BlobName);
        }
    }
    public int EmpId { get; set; }
}
```

6. In Web Application add reference to class library project "**CommonDemoLibrary**"
7. In Web.Config add the following

```
<connectionStrings>
  <add name="MyDemoContext" connectionString="Data Source=.\sqlexpress;Integrated
Security=True;database=DemoDb" providerName="System.Data.SqlClient"/>
 </connectionStrings>
```


Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA.
http://www.deccansoft.com | http://www.bestdotnettraining.com

Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

8. Build the Solution
9. Web App project → Right Click on Controller Folder → Add → Controller… → Select "**MVC 5 Controller with Views, using Entity Framework**" → Add
10. Select Model = Employee, DataContext Class = "DemoDbContext" → Add

    Note: This generates Controller along with Views for Add/Edit/Delete operations.
11. In Views/Employees/Create.cshtml replace the HTML for ImageURL and ThumbnailUrl with the following (FileUpload element)

```
<div class="form-group">
        <label class="control-label col-md-2" for="imageFile">Image file</label>
        <div class="col-md-10">
          <input type="file" name="imageFile" accept="image/*" class="form-control fileupload" />
        </div>
</div>
```

12. In Create.cshtml update the BeginForm as below

```
@using (Html.BeginForm("Create", "Employees", FormMethod.Post, new { enctype = "multipart/form-data"
}))
```

13. Run and test the application by inserting Employee. **Note that ImageURL and ThumbNailURL are blank.**

Storing Image as Blob and posting a message in Azure Storage:

14. Azure Portal → Create a New Storage Account
    a. Set Name = "dssemployeestorage" (all lowercase), Set Region or Affinity Group and Replication → Create
    b. Copy the Connection String of the Storage Account
15. Edit the Web.Config, edit <connectionStrings> section set

```
<connectionStrings>
        <add name="MyDemoContext" . . ./>
        <add name="AzureWebJobsDashboard"
connectionString="DefaultEndpointsProtocol=https;AccountName=dssemployeestorage;AccountKey=0z1SuZxDM+
. . ." />
        <add name="AzureWebJobsStorage"
connectionString="DefaultEndpointsProtocol=https;AccountName=dssemployeestorage;AccountKey=0z1SuZxDM+
. . ." />
```


Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA.
http://www.deccansoft.com | http://www.bestdotnettraining.com

Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

```
</connectionStrings>
```

16. Tools → NuGet Package Manager → Manage NuGet Packages for Solution → Search "Microsoft.Azure.Storage.Blob and Microsoft.Azure.Storage.Queue" → Install.
17. Add the following Employees Controller class

```
public string UploadImage(HttpPostedFileBase imageFile)
{
   //Write code here to Storage Image in Azure Blob Storage and Get the URL of image
   var storageAccount =
CloudStorageAccount.Parse(ConfigurationManager.ConnectionStrings["AzureWebJobsStorage"].ToString());

   //To Upload the Image into the Blob Container
   string blobName = Guid.NewGuid().ToString() + Path.GetExtension(imageFile.FileName);
   var blobClient = storageAccount.CreateCloudBlobClient();
   CloudBlobContainer imagesBlobContainer = blobClient.GetContainerReference("images");
   imagesBlobContainer.CreateIfNotExists();
   CloudBlockBlob imageBlob = imagesBlobContainer.GetBlockBlobReference(blobName);
   var fileStream = imageFile.InputStream;
   imageBlob.UploadFromStream(fileStream);
   fileStream.Close();
   return imageBlob.Uri.ToString();
}

private void PostMessageToQueue(int empId, string imageUrl)
{
   var storageAccount =
CloudStorageAccount.Parse(ConfigurationManager.ConnectionStrings["AzureWebJobsStorage"].ToString());
   //To create the Queue with BlobInformation
   CloudQueueClient queueClient = storageAccount.CreateCloudQueueClient();
   CloudQueue thumbnailRequestQueue = queueClient.GetQueueReference("thumbnailrequest");
   thumbnailRequestQueue.CreateIfNotExists();
   BlobInformation blobInfo = new BlobInformation() { EmpId = empId, BlobUri = new Uri(imageUrl) };
   var queueMessage = new CloudQueueMessage(JsonConvert.SerializeObject(blobInfo));
```

```
    thumbnailRequestQueue.AddMessage(queueMessage);
}
```

18. **Edit** the existing EmployeesController → **Create** method to call the above method. (Don't add new method)

```
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Create([Bind(Include = "EmpName,Salary")] Employee employee, HttpPostedFileBase
imageFile)
{
    if (ModelState.IsValid)
    {
        employee.ImageURL = UploadImage(imageFile);
        db.Employees.Add(employee);
        db.SaveChanges();
        //Convert
        PostMessageToQueue(employee.Id, employee.ImageURL);
        return RedirectToAction("Index");
    }
    return View(employee);
}
```

19. **Build and run the application. Note that this time FullSizeImage is updated with BLOB URL and ThumbNailUrl is null.**

Programming Azure Functions

20. File → Add → New Project → Visual C# → Cloud → Azure Functions, Name="**FunctionsDemoApp**"
21. In the dialog box
    a. In Dropdown select "**Azure Function v1 (.NET Framework)**"
    b. Select Queue Trigger, Expand Storage Account → Browse →
        i. In the Dialog, Select the Storage Account Created using Azure Portal → OK
    c. Leave Connection String as blank, Path = thumbnailrequest (Queue Name)
    d. OK
22. Add reference to "CommonDemoLibrary"

23. Add reference to **System.Drawing**
24. Tools → Nuget Package Manager → Manage NuGet Package for Solution… → Add reference to "**EntityFramework**"
25. Tools → Nuget Package Manager → Manage NuGet Package for Solution… → Add reference to "**Storage Account**"
26. Edit **local.settings.json** as below

```
{
  "IsEncrypted": false,
  "Values": {
    "AzureWebJobsStorage":
"DefaultEndpointsProtocol=https;AccountName=dssdemostorage;AccountKey=2mszgzSAkAu/ACJKpFQWadiNLbvX
PexYUFo6CwMkg7X3N06I3SdIevdLBkJhEOg17p+mbignhq/7GkybbIKCqA==;BlobEndpoint=https://dssdemostorage.
blob.core.windows.net/;TableEndpoint=https://dssdemostorage.table.core.windows.net/;QueueEndpoint=https:/
/dssdemostorage.queue.core.windows.net/;FileEndpoint=https://dssdemostorage.file.core.windows.net/",
    "AzureWebJobsDashboard":
"DefaultEndpointsProtocol=https;AccountName=dssdemostorage;AccountKey=2mszgzSAkAu/ACJKpFQWadiNLbvX
PexYUFo6CwMkg7X3N06I3SdIevdLBkJhEOg17p+mbignhq/7GkybbIKCqA==;BlobEndpoint=https://dssdemostorage.
blob.core.windows.net/;TableEndpoint=https://dssdemostorage.table.core.windows.net/;QueueEndpoint=https:/
/dssdemostorage.queue.core.windows.net/;FileEndpoint=https://dssdemostorage.file.core.windows.net/"
  },
  "connectionStrings": {
    "MyDemoContext": {
      "ConnectionString": "Server=tcp:dssdemoserver.database.windows.net,1433;Initial Catalog=dssdemodb;Persist
Security Info=False;User
ID=dssadmin;Password=Password@123;MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False
;Connection Timeout=30;",
      "ProviderName": "System.Data.SqlClient"
    }
  }
}
```

27. Edit **Function1.cs** as below

```
using System;
```

```
using System.Drawing;
using System.Drawing.Drawing2D;
using System.IO;
using Microsoft.Azure.WebJobs;
using Microsoft.Extensions.Logging;
using Microsoft.WindowsAzure.Storage.Blob;
using MySharedLibrary;

namespace FunctionsDemoApp
{
   public static class Function1
   {
      [FunctionName("Function1")]
      public static void GenerateThumbnail(
         [QueueTrigger("thumbnailrequest")] BlobInformation blobInfo,
         [Blob("images/{BlobName}", FileAccess.Read)] Stream input,
         [Blob("images/{BlobNameWithoutExtension}_thumbnail.jpg")] CloudBlockBlob outputBlob, ILogger log)
      {
         using (Stream output = outputBlob.OpenWrite())
         {
            ConvertImageToThumbnailJPG(input, output);
            outputBlob.Properties.ContentType = "image/jpeg";
         }

         //SqlConnection connection = new SqlConnection
         //{
         //    ConnectionString = Environment.GetEnvironmentVariable("MyDemoContext")
         //};
         //string sql = $"Update employees Set ThumbnailUrl = '{outputBlob.Uri.ToString()}' where
Id={blobInfo.EmpId}";
         //SqlCommand cmd = new SqlCommand(sql,connection);
         //connection.Open();
         //cmd.ExecuteNonQuery();
```

```
    //connection.Close();
    //log.LogInformation("Connection Closed");
    using (MyDemoContext db = new MyDemoContext())
    {
      var id = blobInfo.EmpId;
      Employee emp = db.Employees.Find(id);
      if (emp == null)
      {
        throw new Exception(String.Format("EmpId: {0} not found, can't create thumbnail", id.ToString()));
      }
      emp.ThumbnailURL = outputBlob.Uri.ToString();
      db.SaveChanges();
    }
  }

  public static void ConvertImageToThumbnailJPG(Stream input, Stream output)
  {
    int thumbnailsize = 80;
    int width;
    int height;
    var originalImage = new Bitmap(input);

    if (originalImage.Width > originalImage.Height)
    {
      width = thumbnailsize;
      height = thumbnailsize * originalImage.Height / originalImage.Width;
    }
    else
    {
      height = thumbnailsize;
      width = thumbnailsize * originalImage.Width / originalImage.Height;
    }
```

```
        Bitmap thumbnailImage = null;
        try
        {
            thumbnailImage = new Bitmap(width, height);

            using (Graphics graphics = Graphics.FromImage(thumbnailImage))
            {
                graphics.InterpolationMode = InterpolationMode.HighQualityBicubic;
                graphics.SmoothingMode = SmoothingMode.AntiAlias;
                graphics.PixelOffsetMode = PixelOffsetMode.HighQuality;
                graphics.DrawImage(originalImage, 0, 0, width, height);
            }
            thumbnailImage.Save(output, System.Drawing.Imaging.ImageFormat.Jpeg);
        }
        finally
        {
            if (thumbnailImage != null)
            {
                thumbnailImage.Dispose();
            }
        }
    }
  }
}
```

28. Test the Application Locally
    a. Run the Azure Function Project
    b. Run the WebApplication Project
        i. Add a New Employee
    c. Note that the Azure Function execute GenerateThumbnail as the message was posted into the queue
    d. Refresh the Employee list in web application and note that Thumbnail URL is available.
29. Publish the Web Application and Azure Function to Azure Portal

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA.
http://www.deccansoft.com | http://www.bestdotnettraining.com

Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)