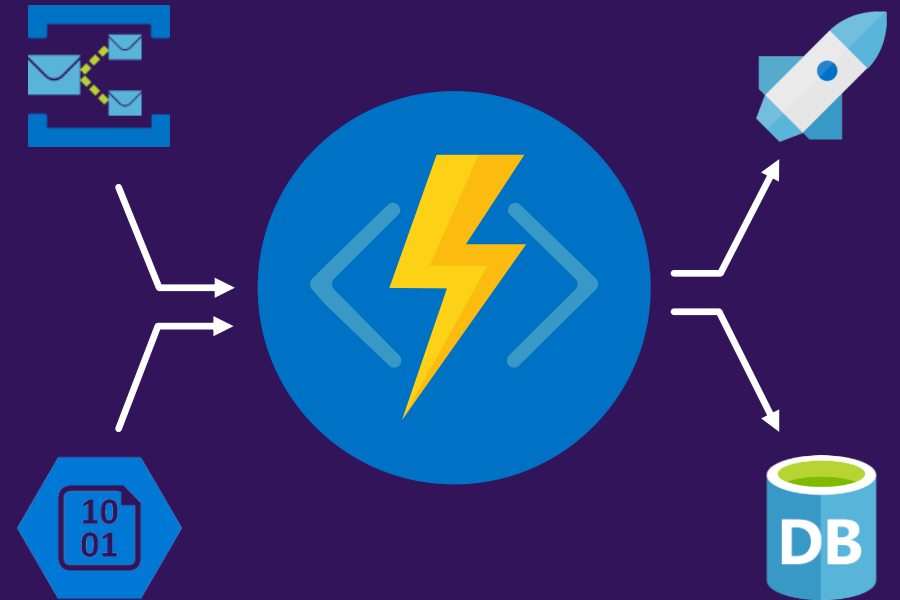


Serverless Computing with Azure Functions

David Gristwood

david.gristwood@microsoft.com

@ScroffTheBad



Serverless applications on Azure



Benefit from a fully managed service

Spare your teams the burden of managing servers. By utilising fully managed services, you can focus on your business logic and avoid administrative tasks. With serverless architecture, you simply deploy your code and it runs with high availability.



Scale flexibly

Serverless compute scales from nothing to handle tens of thousands of concurrent functions almost instantly (within seconds), to match any workload, and without requiring scale configuration – it reacts to events and triggers in near-real time.



Only pay for resources you use

With serverless architecture, you only pay for the time your code is running. Serverless computing is event-driven, and resources are allocated as soon as they're triggered by an event. You're only charged for the time and resources it takes to execute your code – through sub-second billing.

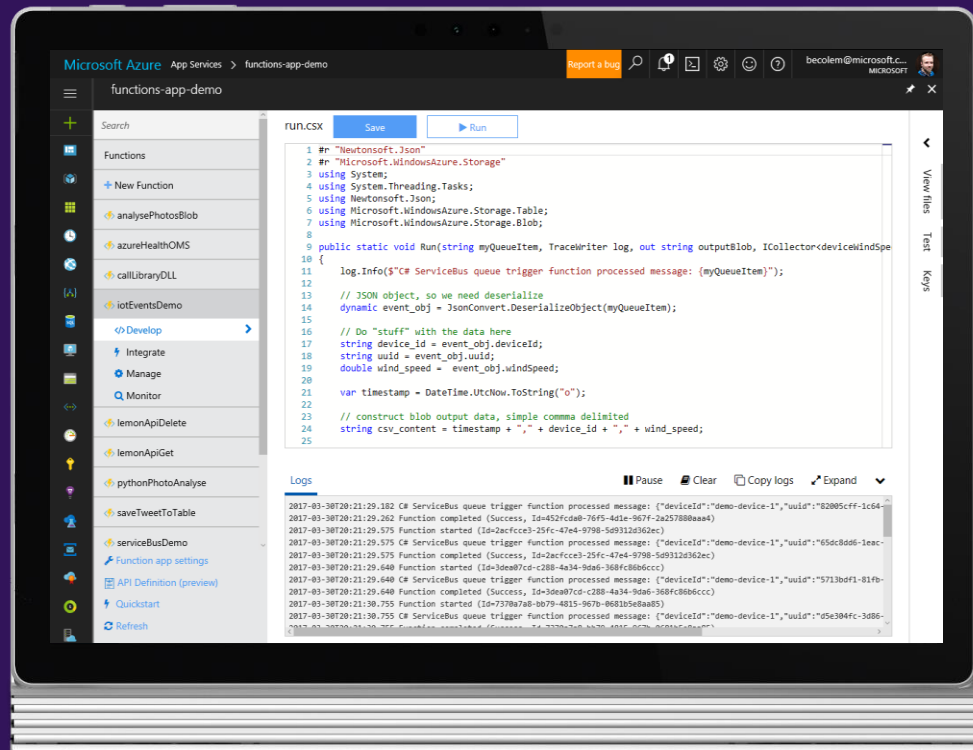
Azure Serverless Resources:

Functions, *Storage, Cosmos DB, Event Grid, Service Bus, Logic Apps, Stream Analytics, Event Hub, Bot Service, Cognitive Services, ...*

Code

Events + data

Azure Functions



Serverless compute - "Function as a Service"

Trigger on events & external services / feeds

Pay only per execution

Choice of languages

Open source runtime

Common Serverless Use Cases

- Scheduled maintenance tasks
- Data ingestion / transform
- Handle REST API calls
- Integration logic and “glue”
- Monitoring / watchdogs
- Manage alerts
- Auto Scaling
- Chatbots

Timer-based processing

Azure Functions supports an event based on a timer using Cron job syntax. For example, execute code that runs every 15 minutes and clean up a database table based on custom business logic.



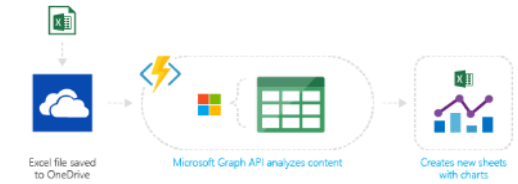
Azure service event processing

Azure Functions supports triggering an event based on an activity in an Azure service. For example, execute serverless code that reads newly discovered test log files in an Azure Blob storage container, and transform this into a row in an Azure SQL Database table.



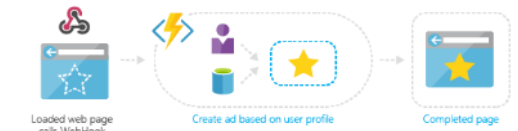
SaaS event processing

Azure Functions supports triggers based on activity in a SaaS service. For example, save a file in OneDrive, which triggers a function that uses the Microsoft Graph API to modify the spreadsheet, and creates additional charts and calculated data.



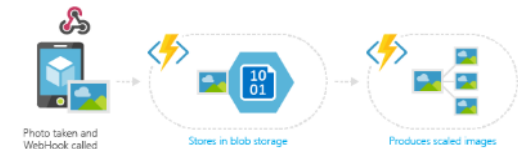
Serverless web application architectures

Azure Functions can power a single-page app. The app calls functions using the WebHook URL, saves user data, and decides what data to display. Or, do simple customizations, such as changing ad targeting by calling a function and passing it user profile information.



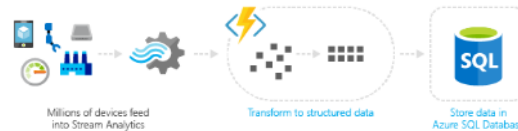
Serverless mobile back ends

A mobile back end can be a set of HTTP APIs that are called from a mobile client using the WebHook URL. For example, a mobile application can capture an image, and then call an Azure Function to get an access token for uploading to blob storage. A second Azure Function is triggered by the blob upload and resizes the image to be mobile-friendly.



Real-time stream processing

For example, Internet of Things (IoT) devices send messages to Azure Stream Analytics, which then calls an Azure Function to transform the message. This function processes the data and creates a new record in an Azure SQL database.



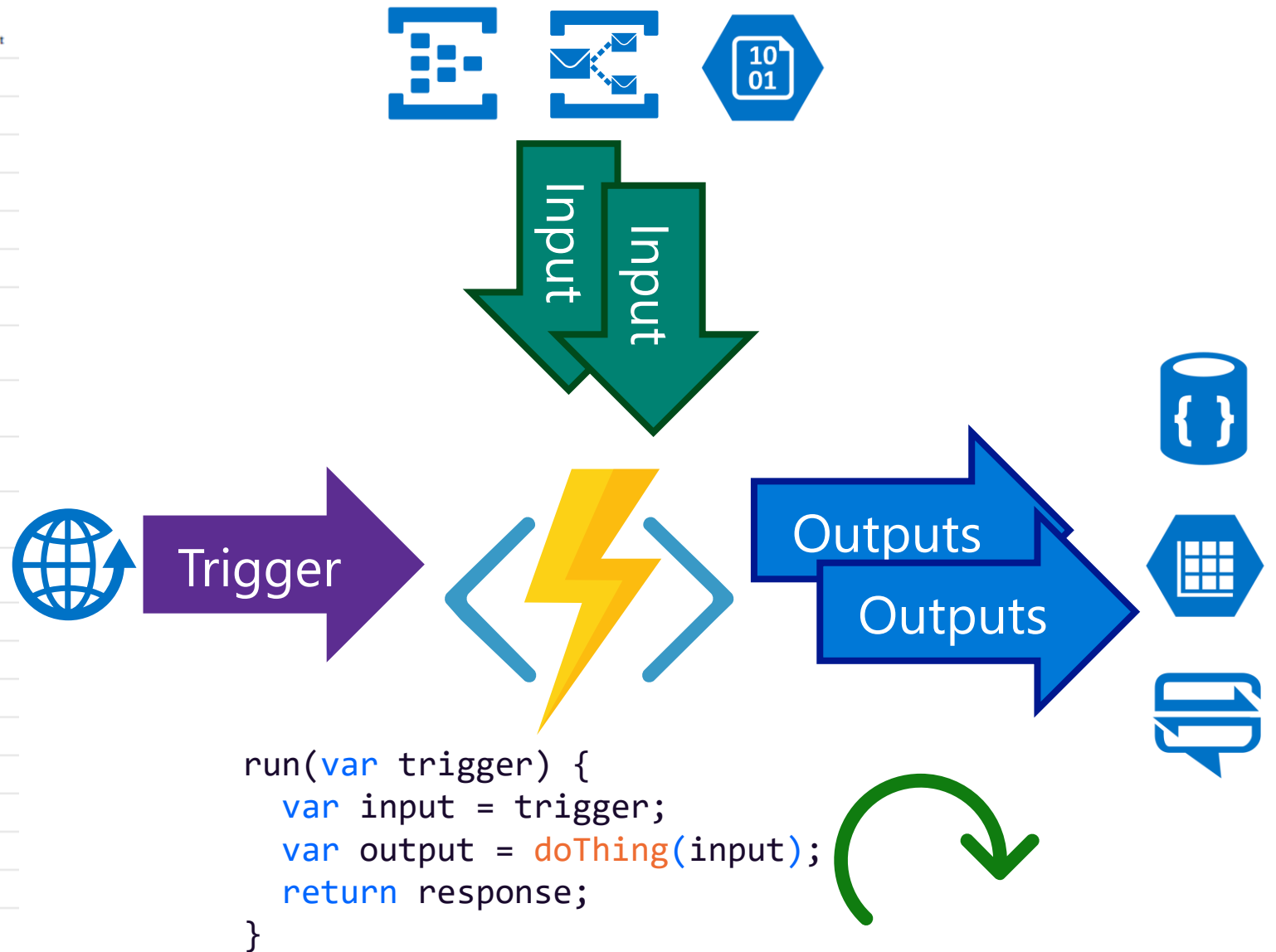
Real-time bot messaging

Use Azure Functions to customize the behavior of a bot using a WebHook. For example, create an Azure Function that processes a message using Cortana Analytics and call this function using Microsoft Bot Framework.



Triggers & Bindings

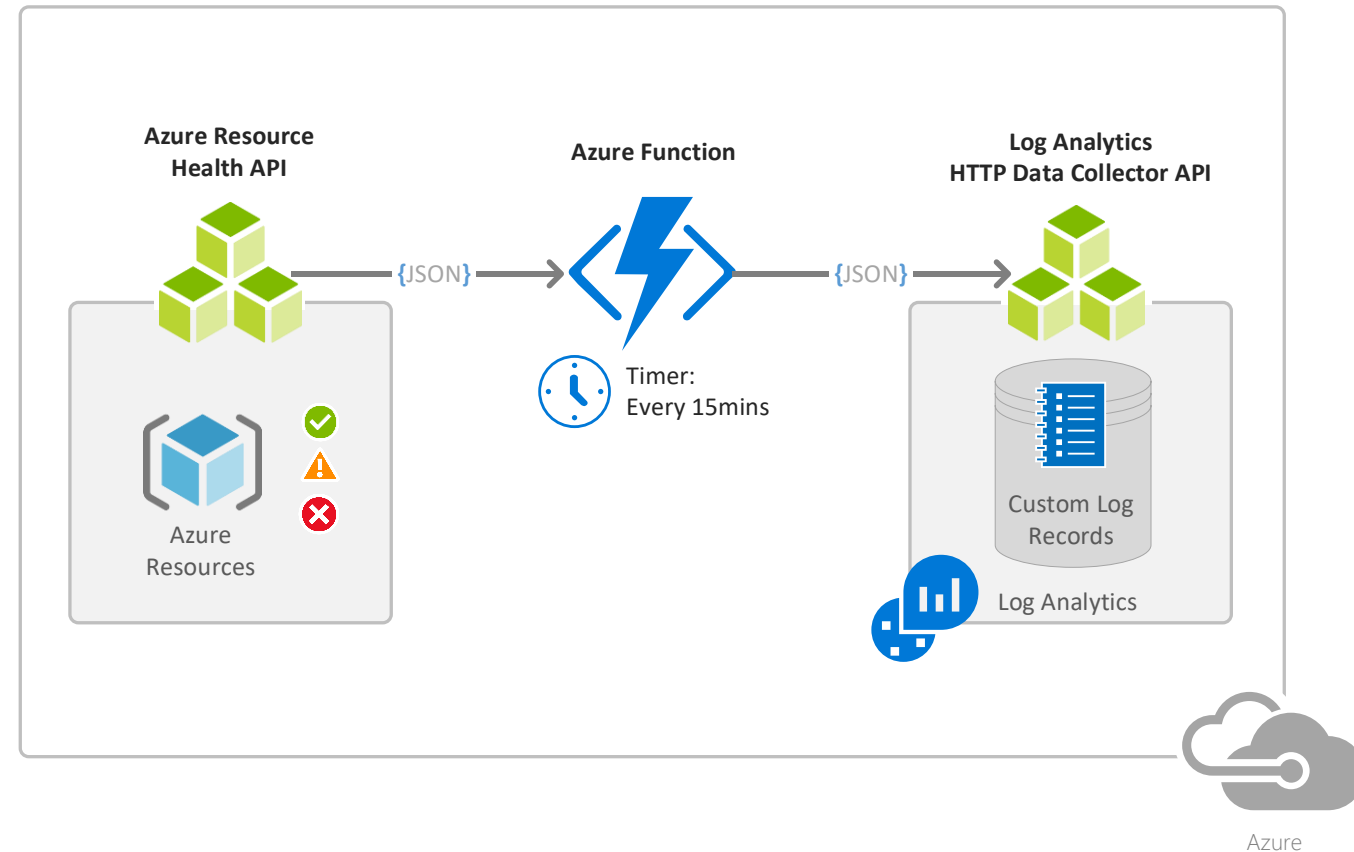
Type	1.x	2.x	Trigger	Input	Output
Blob Storage	✓	✓ ¹	✓	✓	✓
Cosmos DB	✓	✓	✓	✓	✓
Event Grid	✓	✓	✓		
Event Hubs	✓	✓	✓		✓
External File ²	✓			✓	✓
External Table ²	✓			✓	✓
HTTP	✓	✓ ¹	✓		✓
Microsoft Graph Excel tables		✓		✓	✓
Microsoft Graph OneDrive files		✓		✓	✓
Microsoft Graph Outlook email		✓			✓
Microsoft Graph Events		✓	✓	✓	✓
Microsoft Graph Auth tokens		✓		✓	
Mobile Apps	✓	✓		✓	✓
Notification Hubs	✓				✓
Queue storage	✓	✓ ¹	✓		✓
SendGrid	✓	✓			✓
Service Bus	✓	✓	✓		✓
Table storage	✓	✓ ¹		✓	✓
Timer	✓	✓	✓		
Twilio	✓	✓			✓
Webhooks	✓		✓		✓



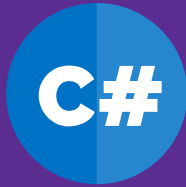
Triggers & Bindings

- However you are **not limited** to the out of the box input & output bindings
- You can write code to do any processing you wish:
 - Make HTTP calls to REST APIs
 - Connect to database, run SQL
 - Load an external DLL / library
 - Connect to external services
 - SSH / FTP / SCP

Example Custom Integration Azure Health to OMS Log Analytics



Language Support



C#



F#



**Java
Script**

Fully Supported Languages



**Power
Shell**

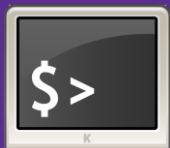


Python



PHP

Preview / Experimental Languages



Bash



Batch



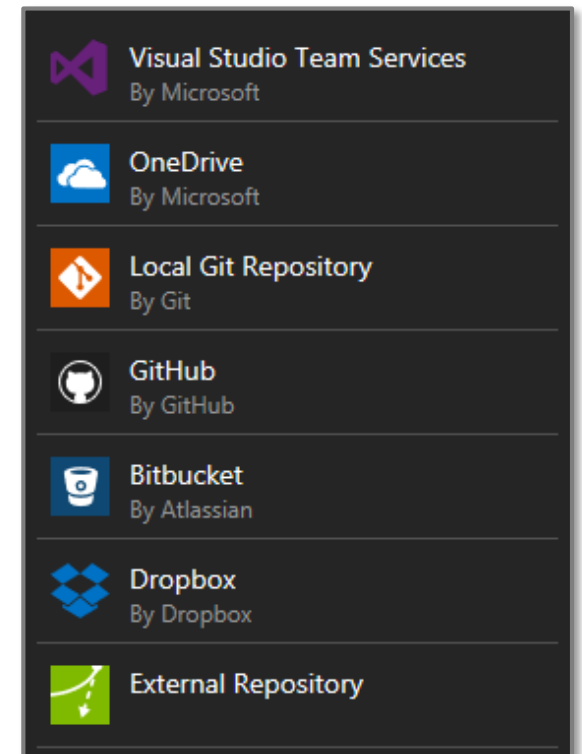
Java

Developing Functions

Use Azure portal & in-browser IDE for getting started, prototyping & testing

Use continuous deployment for real world usage

- Use any IDE or code editor
- Number of source control sources:
 - Github / Git
 - Bitbucket
 - Visual Studio Team Services
 - Dropbox / OneDrive



In Browser IDE

The screenshot displays the Microsoft Azure In Browser IDE interface. The top navigation bar shows the user is logged in as 'becolem@microsoft.c...' and the current project is 'functions-app-demo - saveTweetToTable'. The left sidebar contains a search bar and a list of function apps, with 'saveTweetToTable' selected. The main editor area shows the C# code for the 'run.csx' file, which is a function that takes an HTTP request and returns a JSON response. The code includes comments and uses the 'Newtonsoft.Json' library for JSON handling. The right sidebar contains a 'Test' tab with a 'POST' method, a query string 'foo=bar', a header 'content-type: application/json', and a request body containing a JSON object with tweet details. Below the test tab is an 'Output' section with a 'Run' button. At the bottom, a 'Logs' section shows the execution history of the function, including timestamps, log messages, and function completion status.

Microsoft Azure App Services > functions-app-demo - saveTweetToTable

Search resources

functions-app-demo - saveTweetToTable
Function Apps

Search

Microsoft Azure Internal Consumption

Function Apps

functions-app-demo

Functions

analysePhotosBlob

Integrate

Manage

Monitor

azureHealthOMS

callLibraryDLL

iotEventsDemo

lemonApiDelete

lemonApiGet

pythonPhotoAnalyse

saveTweetToTable

Integrate

Manage

Monitor

serviceBusDemo

Proxies (preview)

simple-api

run.csx

Save

Run

</> Get function URL

```
1 #r "Newtonsoft.Json"
2 using System;
3 using System.Net;
4 using Newtonsoft.Json;
5
6 public static async Task<object> Run(HttpRequestMessage req, ICollector<Tweet> outputTweetTable, Tr
7 {
8     // Get input HTTP request, and deserialize from JSON to a dyanmic
9     string jsonContent = await req.Content.ReadAsStringAsync();
10    dynamic tweet_input = JsonConvert.DeserializeObject(jsonContent);
11
12    log.Info($"### New tweet received: "+tweet_input.TweetId);
13
14    try {
15        // Don't look at this code too closely
16        // Listen to the nice man talking about Azure instead
17        Tweet t = new Tweet();
18        string dayISO = tweet_input.CreatedAtIso.ToString("o").Substring(0, 10);
19        t.PartitionKey = dayISO;
20        t.RowKey = tweet_input.TweetId;
21        t.Text = tweet_input.TweetText;
22        t.User = tweet_input.TweetedBy;
23        t.Lang = tweet_input.TweetLanguageCode;
24
25        // Add POCO to collection for the Webjob SDK to magically push into the output Table
26        outputTweetTable.Add(t);
27    } catch(Exception e) {
28        // Bummer return a HTTP 400 and spit out some logs
29        log.Error($"!!! {e.ToString()}");
30    }
```

View files Test Keys

HTTP method

POST

Query

foo bar

+ Add parameter

Headers

content-type application/json

+ Add header

Request body

```
1 {
2   "CreatedAtIso": "2017-04-05T18:03:02+00:00",
3   "TweetId": "64352433",
4   "TweetLanguageCode": "en-gb",
5   "TweetText": "Listen to the nice man talking",
6   "TweetedBy": "@nobody"
7 }
```

Output

Run

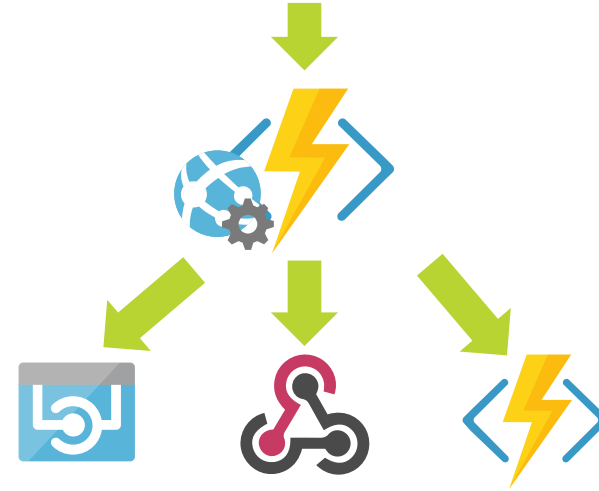
Logs

Pause Clear Copy logs Expand

2017-04-06T07:35:46.435 ### New tweet received: 6435247
2017-04-06T07:35:46.528 ### Tweet inserted into Azure table OK, bye
2017-04-06T07:35:46.622 Function completed (Success, Id=b5011ca9-367b-4ca3-b06d-81400110e988)
2017-04-06T07:35:47.357 Exception while executing function: Functions.saveTweetToTable. Microsoft.Azure.WebJobs.Host: Error
RequestId:acf86d62-0002-0083-08a8-aef063000000
Time:2017-04-06T07:35:46.5959096Z.
2017-04-06T07:35:52.748 Function started (Id=0c994d9f-e8c5-4bbd-b71c-ffdec036723c)
2017-04-06T07:35:52.748 ### New tweet received: 64352433
2017-04-06T07:35:52.748 ### Tweet inserted into Azure table OK, bye
2017-04-06T07:35:52.748 Function completed (Success, Id=0c994d9f-e8c5-4bbd-b71c-ffdec036723c)

Azure Function Proxies

- Define a single API surface for multiple Function Apps
 - Independent scaling for microservice architecture
- Proxy to multiple APIs
 - Other Function apps
 - Azure API Apps
 - or other URL endpoints



Proxy URL

`https://demofunction.azurewebsites.net/lemons/{action}/{id}`

Route template

`/lemons/{action}/{id}`

Allowed HTTP methods

Selected methods

☒ GET ☒ POST ☒ DELETE ☐ HEAD

☐ PATCH ☐ PUT ☐ OPTIONS ☐ TRACE

Backend URL

`https://someotherfunction.azurewebsites.net/api/lemonApi{action}?id={id}`

Save Discard

[Delete proxy](#)

DURABLE FUNCTIONS

PREVIEW

Allows writing of *long-running, stateful* function orchestrations

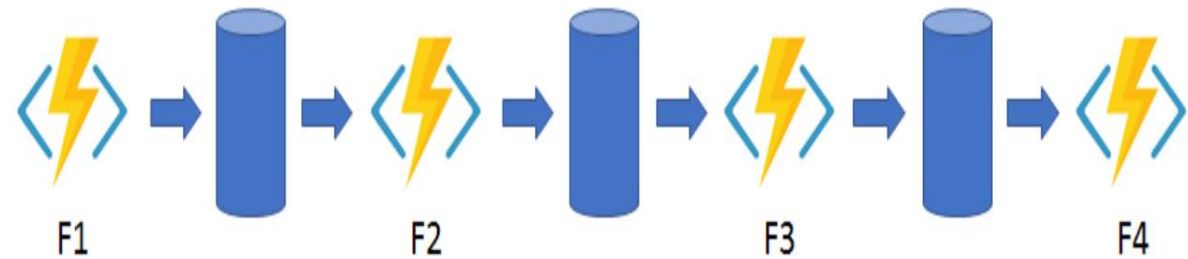
- They are stateful workflows **authored in code**.
- They can *synchronously* and *asynchronously* **call other functions** and **save output to local variables**.
- They **automatically checkpoint** their progress, so that local state is never lost

DURABLE FUNCTIONS

PREVIEW

Function chaining

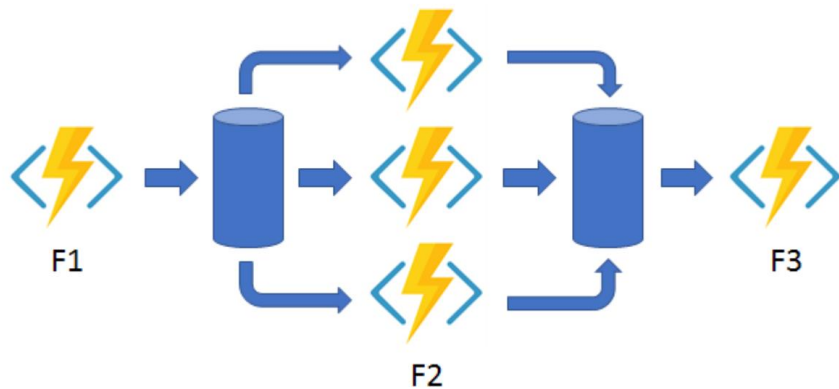
```
public static async Task<object> Run(DurableOrchestrationContext ctx)
{
    try
    {
        var x = await ctx.CallActivityAsync<object>("F1");
        var y = await ctx.CallActivityAsync<object>("F2", x);
        var z = await ctx.CallActivityAsync<object>("F3", y);
        return await ctx.CallActivityAsync<object>("F4", z);
    }
    catch (Exception)
    {
        // error handling/compensation goes here
    }
}
```



DURABLE FUNCTIONS

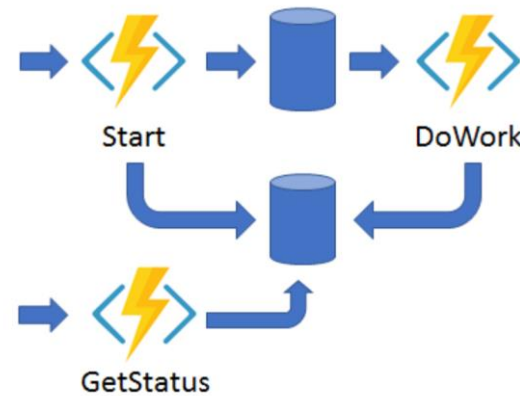
PREVIEW

Fan-out/fan-in



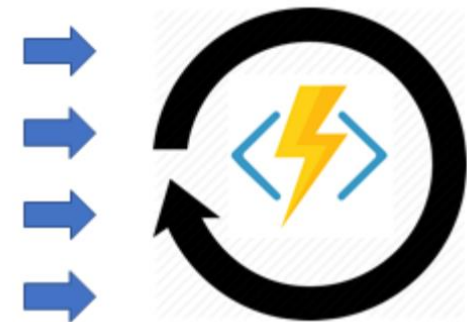
Fan-out/fan-in is the pattern of executing multiple functions in parallel, and then waiting for all to finish.

Async HTTP APIs



Coordinating the state of long-running operations with external clients

Stateful singletons



Stateful singleton pattern allows Functions to behave like reliable actors

Billing & Usage Models

Consumption Plan



- Shared resources
- Limited on execution time and other factors
- Pay per execution
 - £0.15 per million runs
 - £0.000012 per GB/s (gigabyte seconds)
- Free each month:
 - 1 million executions
 - 400,000 GB-s

App Service Plan

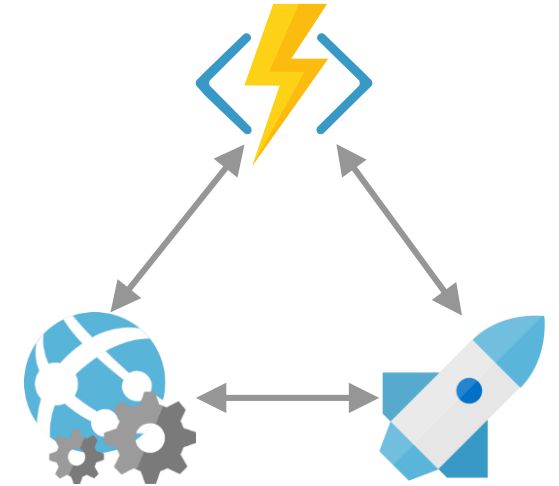


- Dedicated resources
- Same SKUs and tiers as other App Services (web apps)
- Pay a fixed hourly rate
- Share with your other PaaS apps

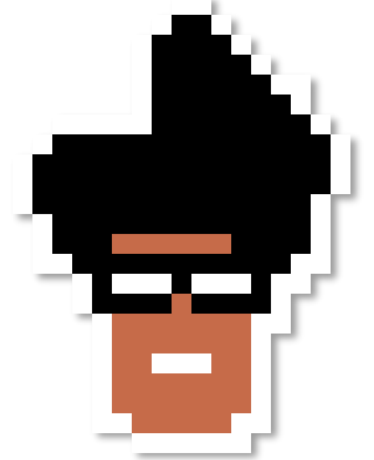
[Choose the correct service plan for Azure Functions Pricing Information](#)

Functions vs WebJobs & Logic Apps

- Logic Apps are **codeless** and **workflow** based, optimised for **integration** tasks.
 - Aimed at non-developers
- If part of your integration scenario requires highly specialized logic, use a **Function app**
- Functions are the natural evolution of WebJobs.
For very simple task scheduling on *existing* Azure Web app, you can use a **WebJob**
- Logic Apps and Functions are designed to be combined and used together



Warning!
Nerdy Stuff



[Official Guidance and Documentation](#)



**Ben's
DEMO**



aka.ms/smilr