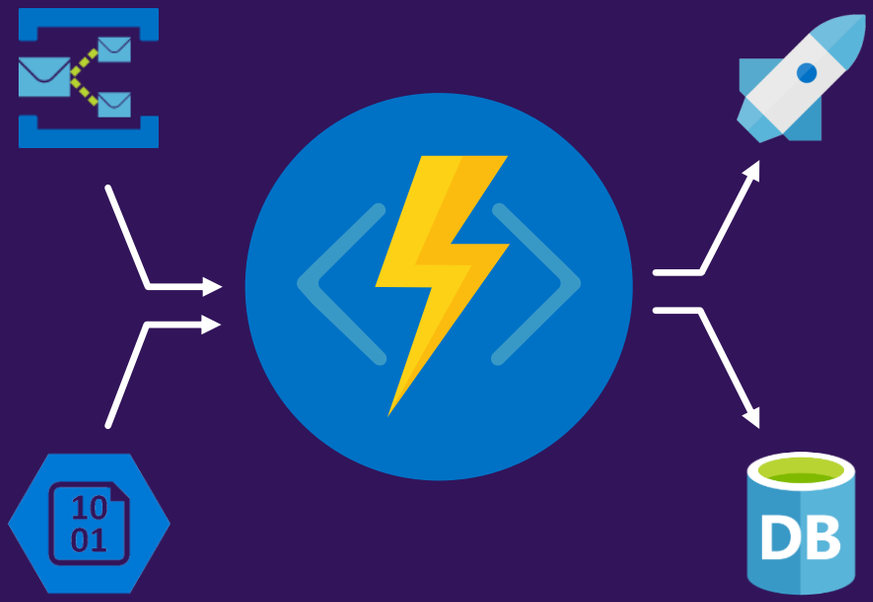
Serverless Computing with Azure Functions

David Gristwood
david.gristwood@microsoft.com
@ScroffTheBad



Microsoft

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.

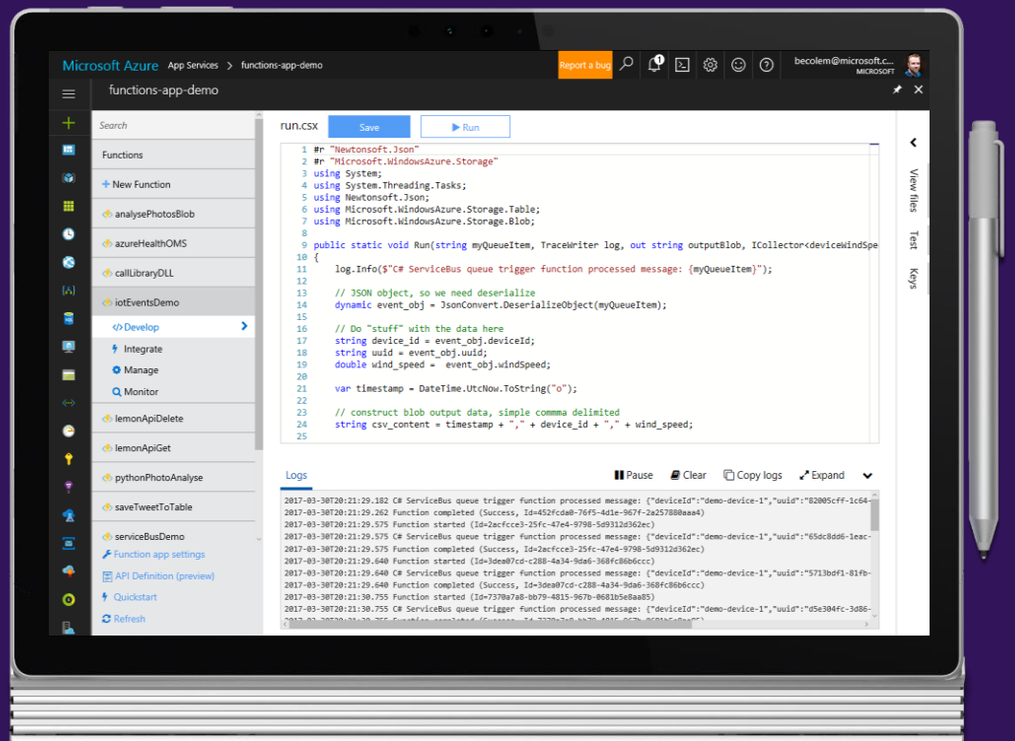
Azure Serverless Resources:

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

Code

Events + data

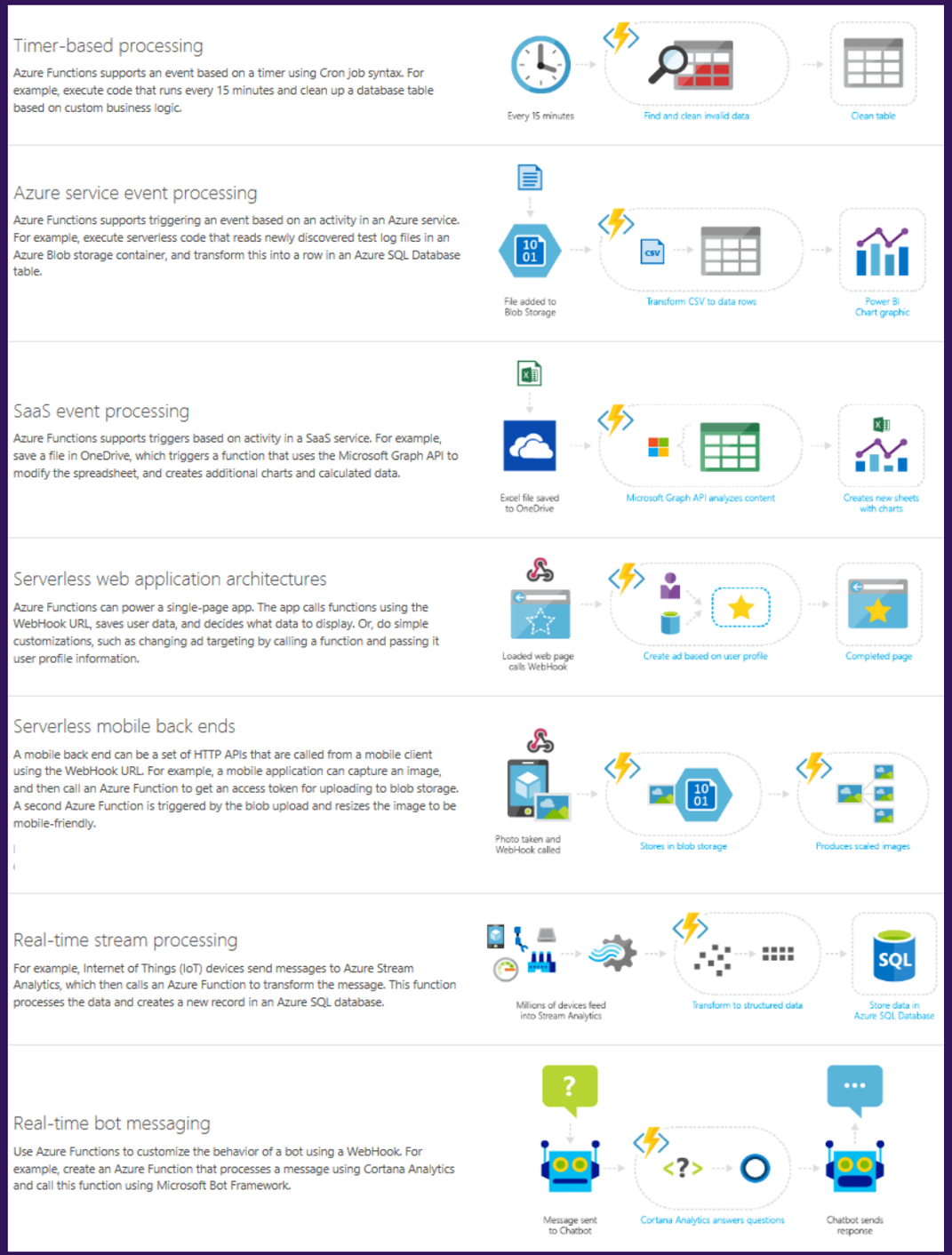
Azure Functions



- Serverless compute - “Function as a Service”
- Trigger on events & external services / feeds
- Pay only per execution
- Choice of languages
- Open source runtime

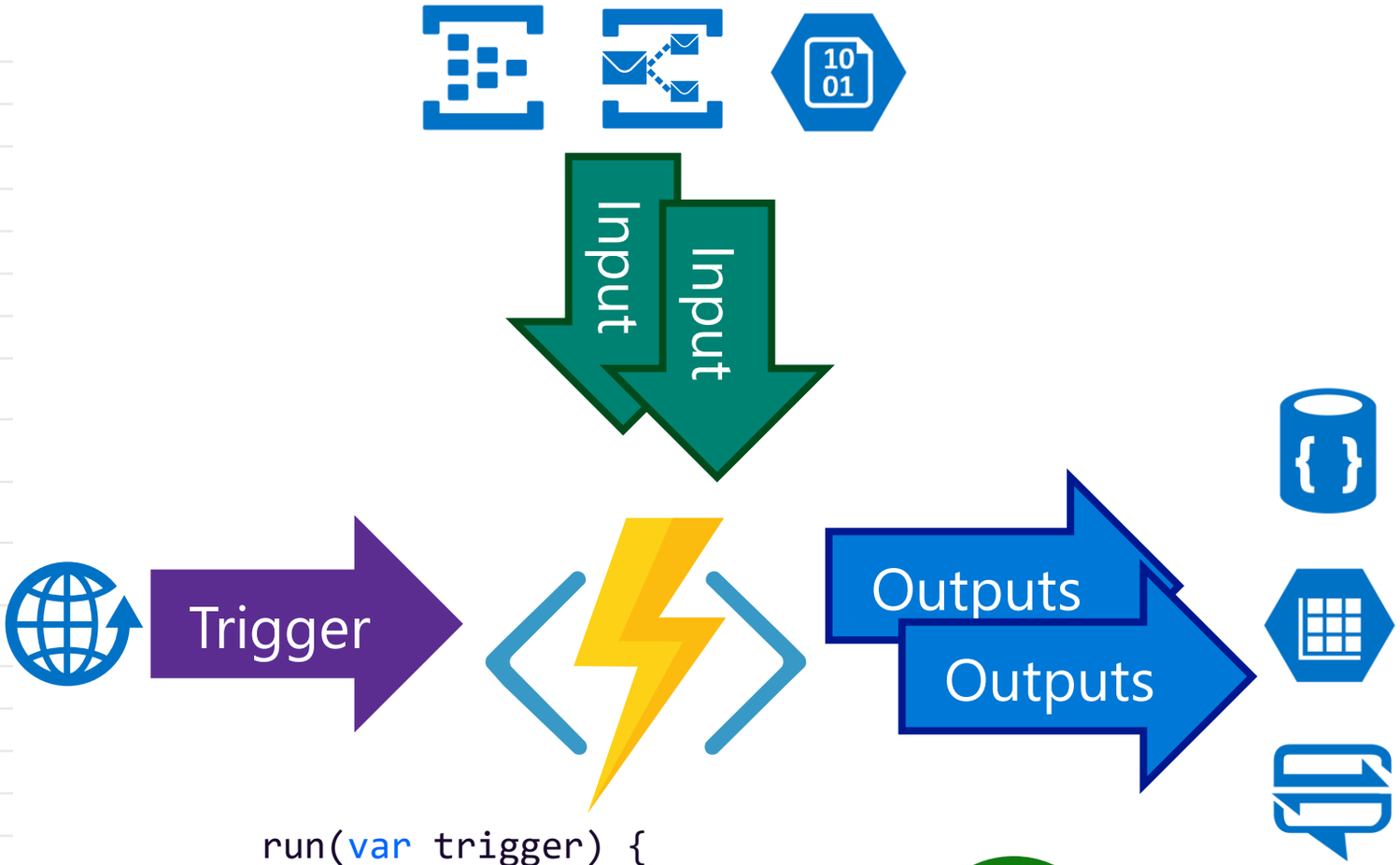
Common Serverless Use Cases

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



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	✔		✔		✔




```
run(var trigger) {
  var input = trigger;
  var output = doThing(input);
  return response;
}
```

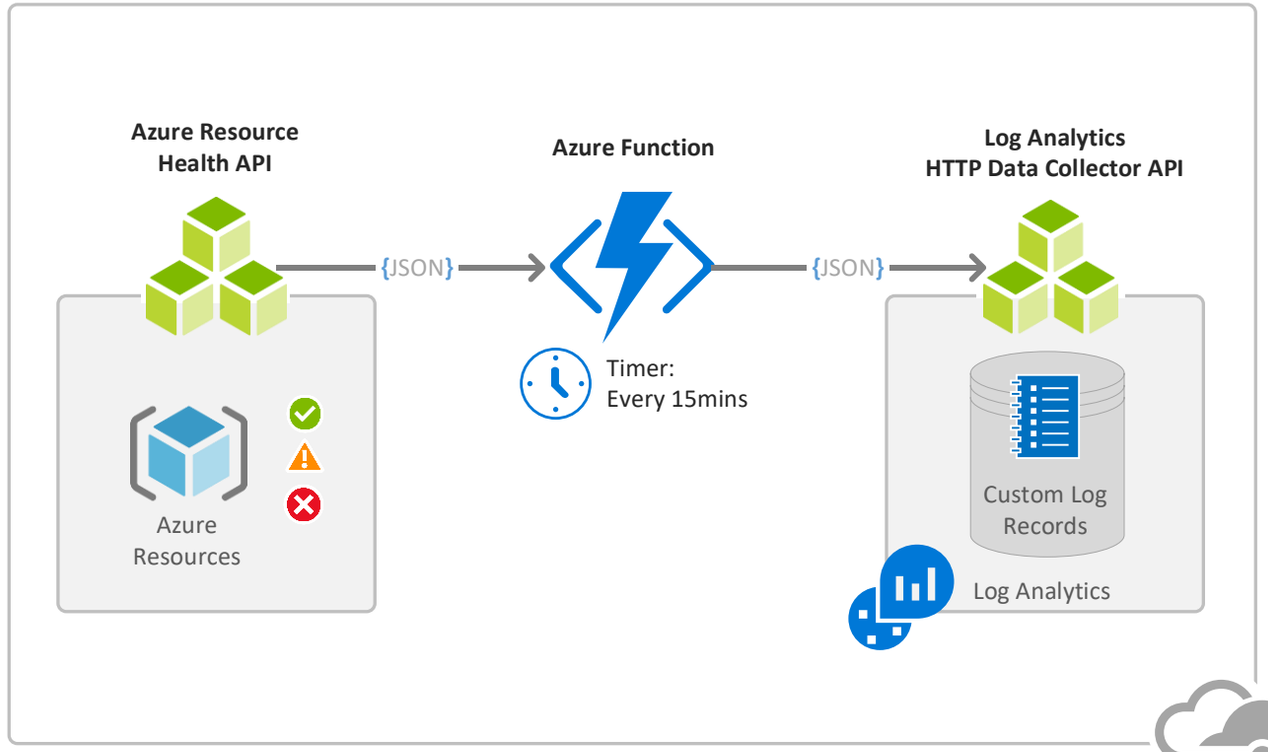
Triggers & Bindings

- 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

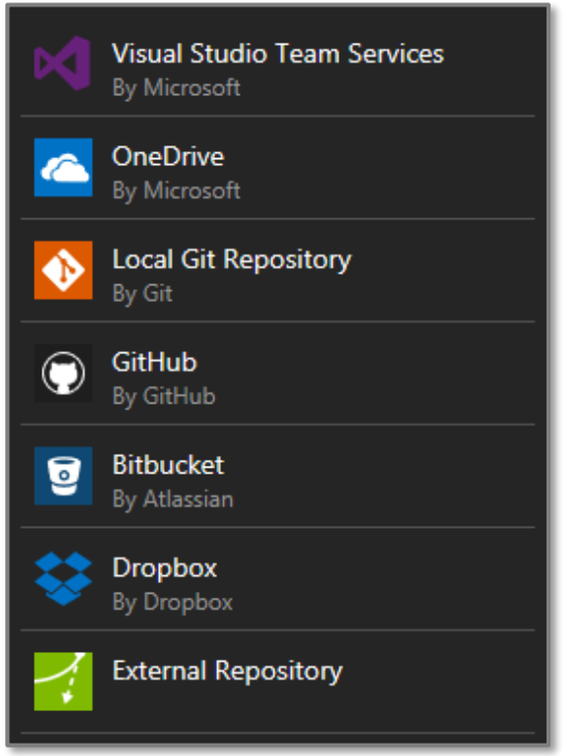
Java

Developing Functions

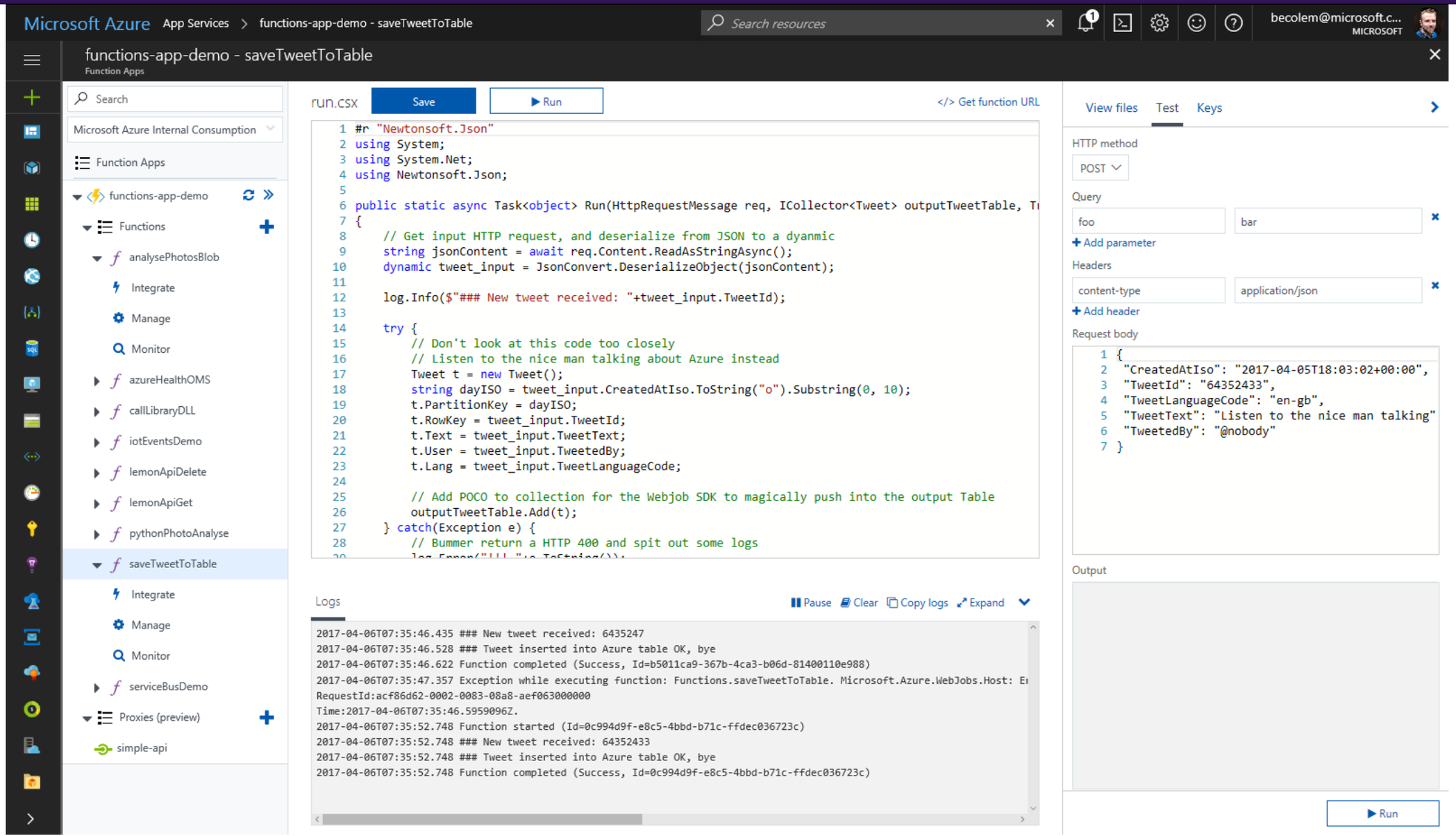
Use Azure portal & in-browser IDE for getting started, prototyping & testing

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

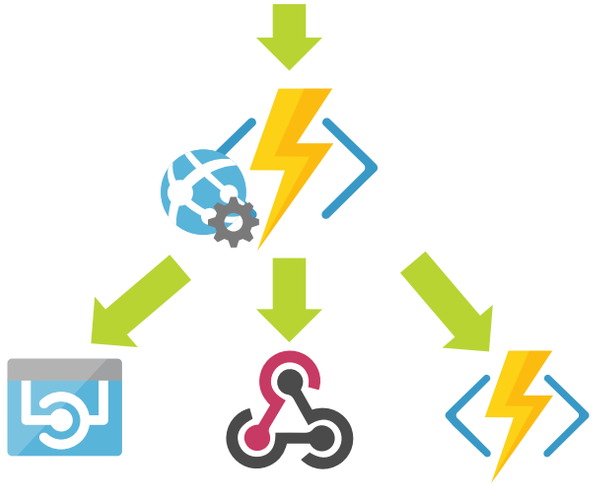


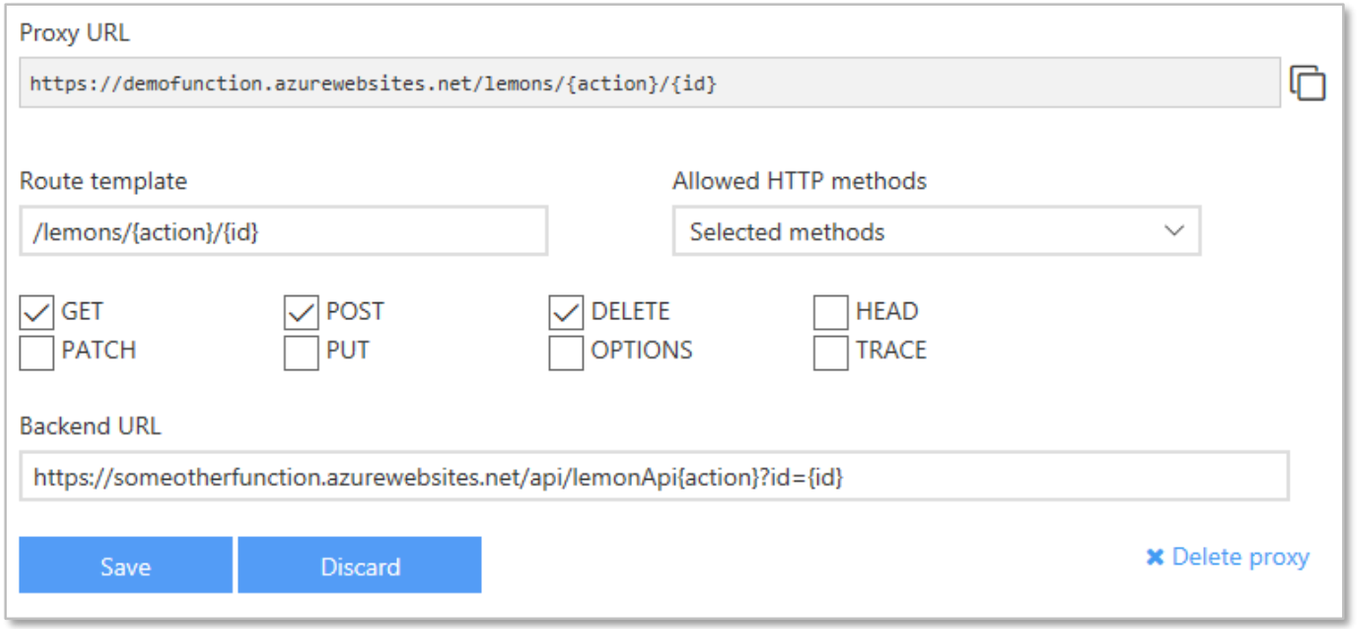
In Browser IDE



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





DURABLE FUNCTIONS

PREVIEW

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

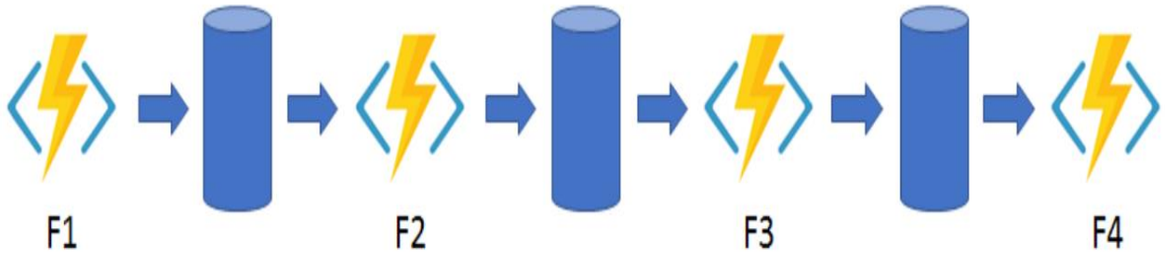
- They are stateful workflows **authored in code**.
- They can *synchronously* and *asynchronously* **call other functions** and **save output to local variables**.
- They **automatically checkpoint** their progress, so that local state is never lost

DURABLE FUNCTIONS

PREVIEW

Function chaining

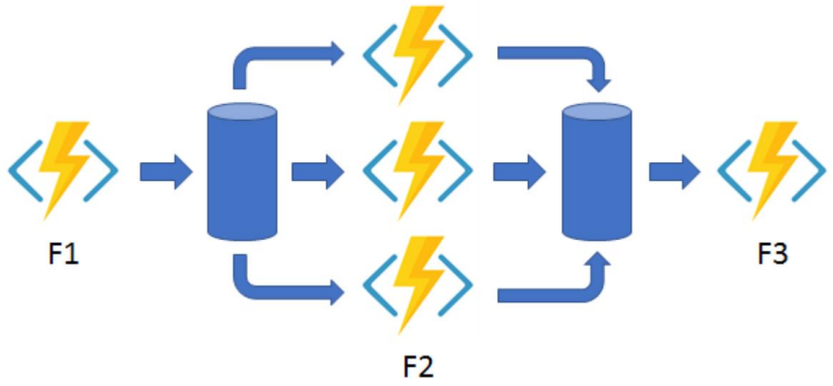
```
public static async Task<object> Run(DurableOrchestrationContext ctx)
{
    try
    {
        var x = await ctx.CallActivityAsync<object>("F1");
        var y = await ctx.CallActivityAsync<object>("F2", x);
        var z = await ctx.CallActivityAsync<object>("F3", y);
        return  await ctx.CallActivityAsync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



DURABLE FUNCTIONS

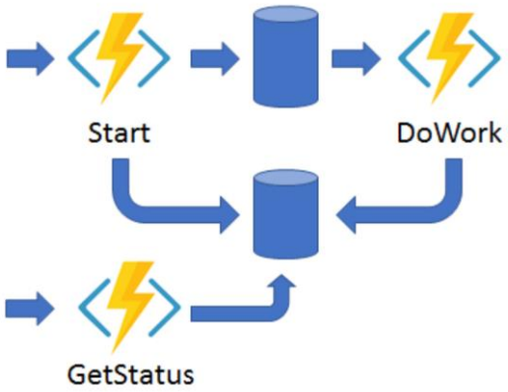
PREVIEW

Fan-out/fan-in



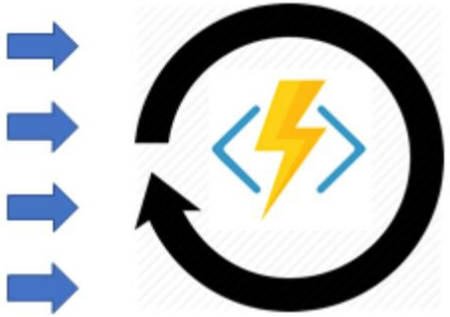
Fan-out/fan-in is the pattern of executing multiple functions in parallel, and then waiting for all to finish.

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

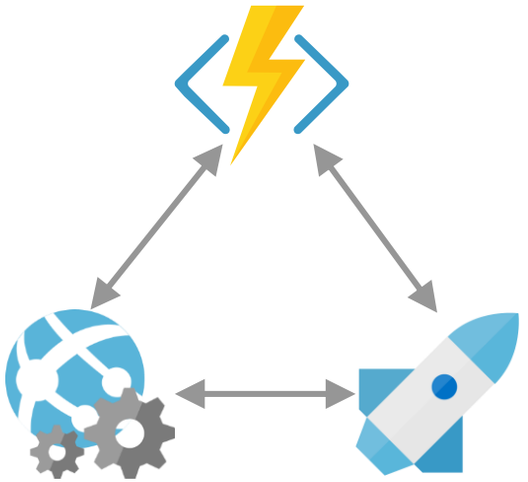
- Dedicated resources
- Same SKUs and tiers as other App Services (web apps)
- Pay a fixed hourly rate
- Share with your other PaaS apps

Choose the correct service plan for Azure Functions

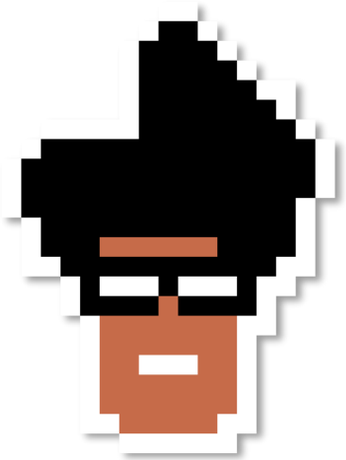
Pricing Information

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

Microsoft

aka.ms/smilr