

# Developing with Azure Functions



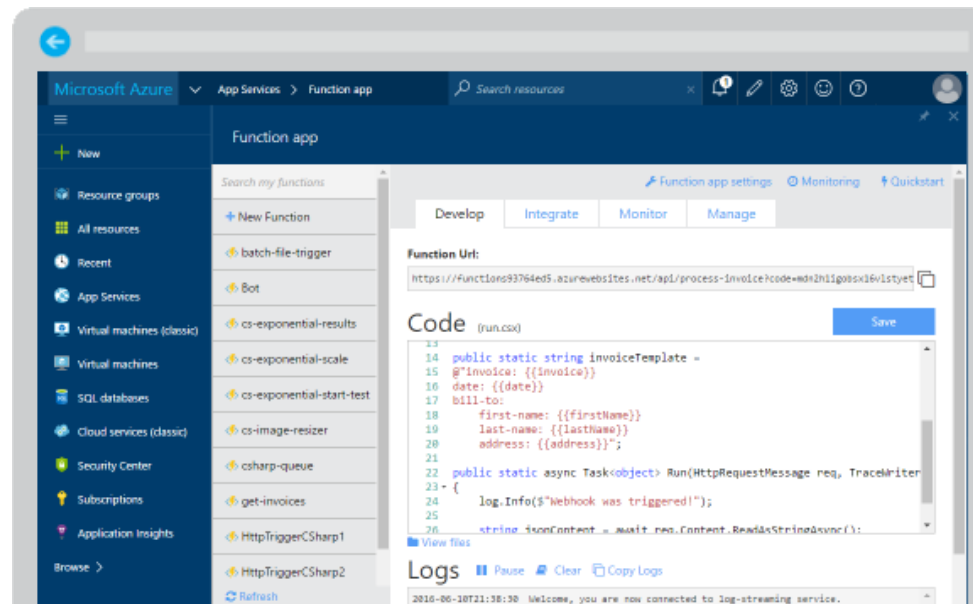
# Agenda

- Introduction to Developing with Azure Functions
- Creating and Testing Azure Functions in the Azure Portal
- Using Azure Functions to Create a Custom Web API
- Configuring Security and Cross-Origin Resource Sharing
- Calling Azure Functions from SPFX Web Parts
- Developing Azure Function in Visual Studio using C#



# Azure Functions

Create a “serverless” event-driven experience that extends the existing Azure App Service platform by building “nanoservices” that can scale based on demand



# Dual abstraction

- Serverless compute abstracts away the compute
- Bindings abstract away services you interact with

Other Services



Business Logic

Serverless PaaS



# Supported Languages and Tools

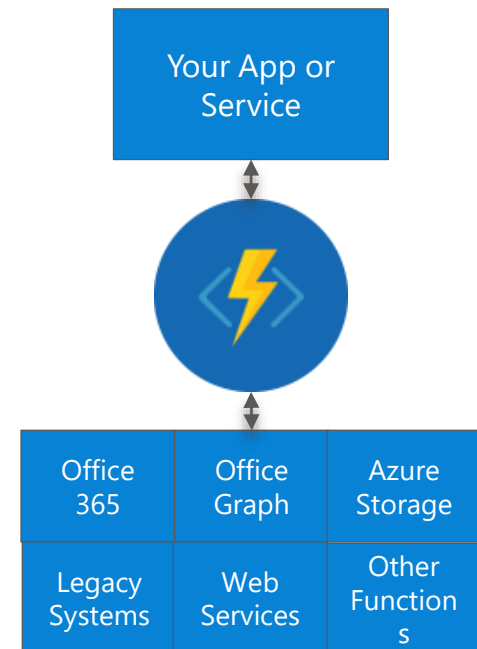
Create functions in JavaScript, C#, Python, and PHP, as well as scripting options such as Bash, Batch, and PowerShell, that can be triggered by virtually any event in Azure, 3rd party services, or on premise systems

The screenshot displays the Azure Functions portal interface. On the left, a sidebar contains configuration options: 'Runtime version: Latest (~0.4)', 'Memory Size' (with a slider), 'Features' (including 'Continuous Integration' and 'Authentication/Authorization'), 'CORS', and 'API definition'. The main content area features a blue banner titled 'The faster way to functions' with the text: 'Write any function in minutes - whether to run a simple job that cleans up a database or to build a more complex architecture. Creating functions is easier than ever before, whatever your chosen OS, platform, or development method. No install required.' Below the banner, it says 'Get started quickly with a premade function' and '1) Choose a scenario:'. Three scenario icons are shown: 'Timer' (clock icon), 'Data processing' (database icon), and 'Webhook + API' (code icon). At the bottom, there are three blue buttons: 'Authentication', 'Configure CORS', and 'Configure API metadata'.



# Common Scenarios

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





# Function App Templates

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

Choose a template

Language:  Scenario:

<b>BlobTrigger - C#</b> A C# function that will be run whenever a blob is added to a specified container	<b>BlobTrigger - Node</b> A Node.js function that will be run whenever a blob is added to a specified container	<b>Empty - C#</b> An empty C# function without triggers, inputs, or outputs	<b>Empty - Node</b> An empty Node.js function without triggers, inputs, or outputs
<b>EventHubTrigger - Node</b> A Node.js function that will be run whenever an event hub receives a new event	<b>Generic Webhook - C#</b> A C# function that will be run whenever it receives a webhook request	<b>Generic Webhook - Node</b> A Node.js function that will be run whenever it receives a webhook request	<b>GitHub WebHook - C#</b> A C# function that will be run whenever it receives a GitHub webhook request



# Timer Function Apps

- Run at explicitly specified intervals
  - e.g. executes once every 5 minutes
  - Configured using CRON expressions ( "0 \*/5 \* \* \* \*" )
  - Can send information to other systems, but typically don't "return" information, only write to logs
  - Great for redundant cleanup and data management
  - Great for checking state of services
  - Can be combined with other functions





# Data Processing Function Apps

- Run when triggered by a data event, such as an item being added to a queue or container
  - Typically have in and out parameters
  - Great for responding to CRUD events
  - Great for performing CRUD events
  - Great for moving content
  - Access data across services



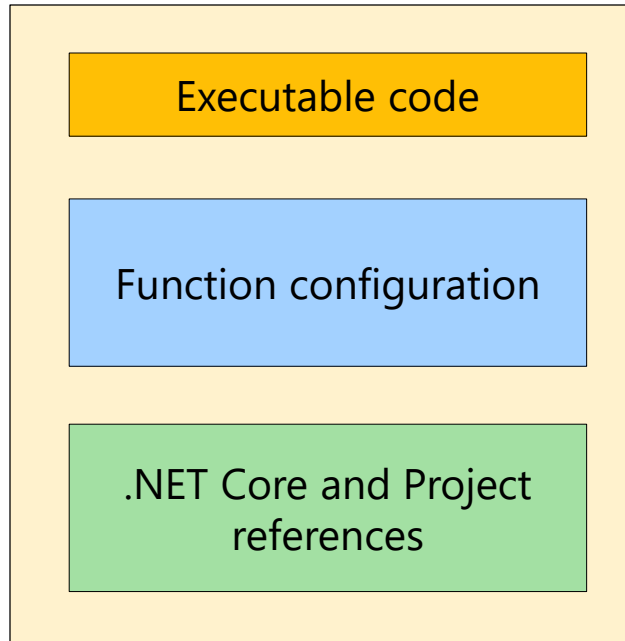
# Webhook & API Function Apps

- Triggered by events in other services
  - GitHub
  - Team Foundation Services
  - Office 365
  - OneDrive
  - Microsoft PowerApps
- Takes in a request and sends back a response
  - Often mimic Web API and legacy web services flows
  - Typically need CORS settings managed
  - Best for exposing functionality to other apps and services
  - Great for building Logic Apps



# Anatomy of a Function

- “Run” file that containing the function code
- “Function” file containing service & trigger bindings & parameters
- “Project” file containing project assembly and NuGet packages
- App Service settings, such as connection strings and API keys



# Function Bindings

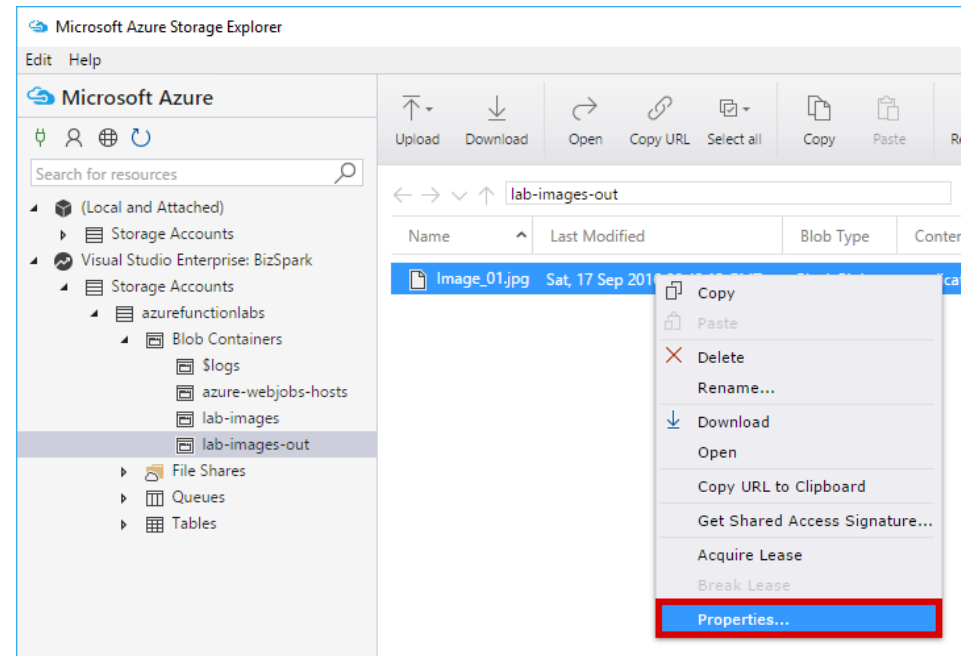
- Bindings used to create connections to and from function
  - Many bindings can be “bi-directional” as well.

Type	Service	Trigger	Input	Output
Schedule	Azure Functions	✓		
HTTP (REST or webhook)	Azure Functions	✓		✓*
Blob Storage	Azure Storage	✓	✓	✓
Events	Azure Event Hubs	✓		✓
Queues	Azure Storage	✓		✓
Tables	Azure Storage		✓	✓
Tables	Azure Mobile Apps		✓	✓
No-SQL DB	Azure DocumentDB		✓	✓
Push Notifications	Azure Notification Hubs			✓

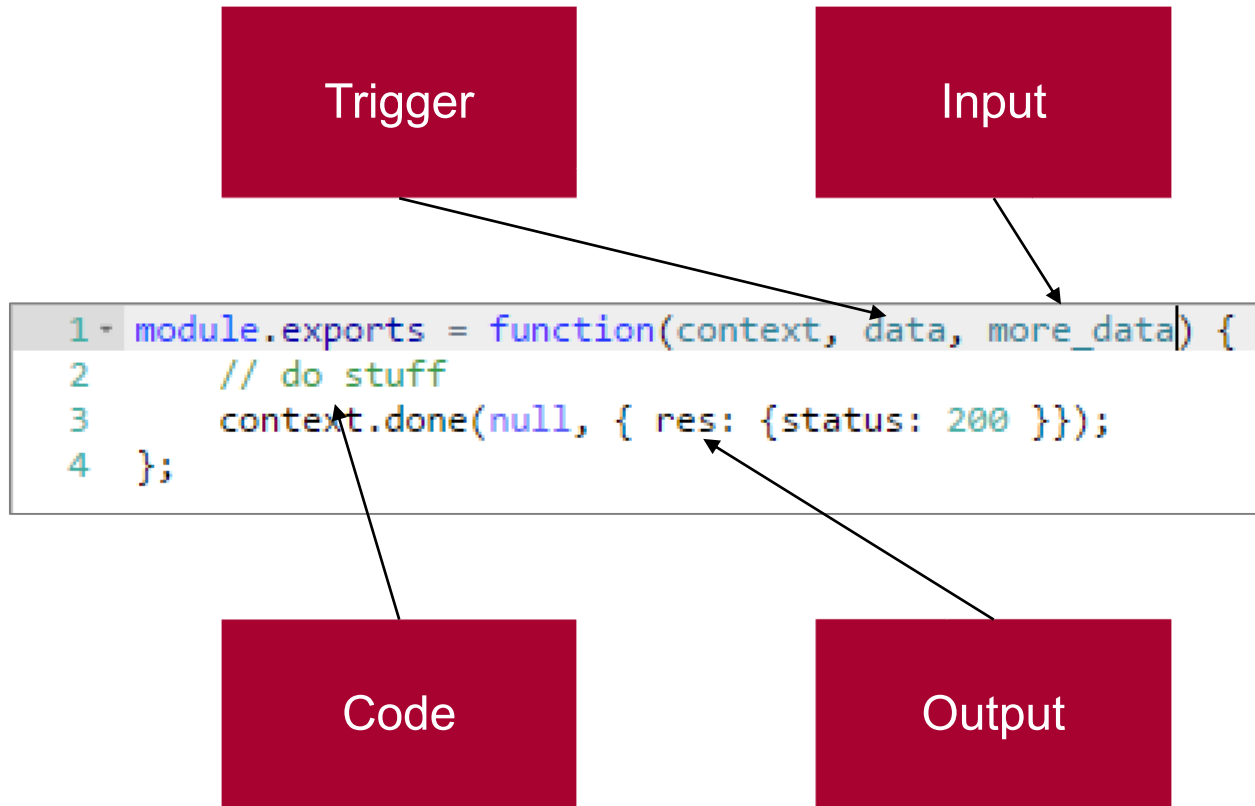


# Testing Functions

- Command-line tools
- 3<sup>rd</sup> party products such as Postman and Swagger
- Direct web calls via cURL
- Nested functions
- Microsoft Azure Storage Explorer
- Visual Studio Cloud Explorer



# Functions programming concepts



# Azure Function Development Process

