Serverless in Azure

VAIBHAV GUJRAL

About me

13+ years of experience across designing and developing enterprise-class applications

Microsoft Certified Azure Solutions Architect Expert

Currently working for Kiewit as a Cloud Architect

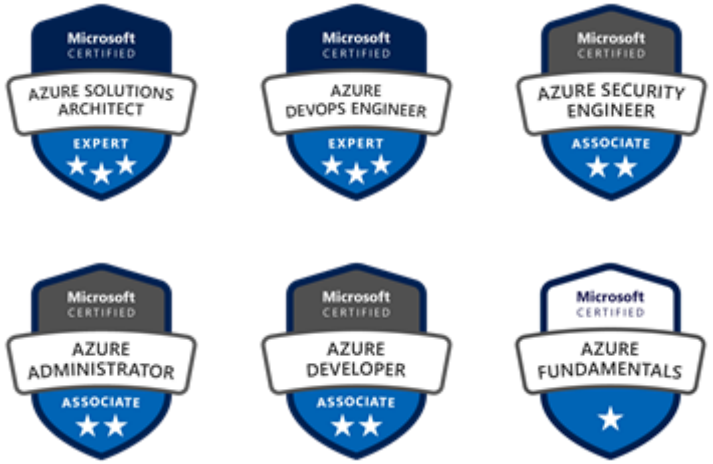
Based out of Omaha, NE

Speaker | Blogger

http://www.vaibhavgujral.com

@vabgujral

gujral.vaibhav@gmail.com



Agenda

What is Serverless?

Serverless offerings in Azure

Azure Functions

- Triggers and Bindings
- Tooling
- Pricing
- Durable Functions

Azure Logic Apps

- Triggers and Connectors
- Pricing

What is Serverless?

Doesn’t mean No-Server

Rather, think of it as Less-Server

Abstraction of servers, infrastructure, and operating systems

Function-as-a-Service (FaaS)

- Execute independent code pieces as functions
- Examples-
  - Azure Functions
  - AWS Lambda
  - Google Cloud Functions

Integration Platform as a Service (iPaaS)

- Azure Logic Apps

Serverless offerings in Azure

Serverless Functions

Serverless Kubernetes

Serverless Workflows

Serverless application environment

Serverless API Gateway

Serverless SQL Database

Azure Functions

What are Azure Functions?

Serverless compute service that can run code on-demand

Run small pieces of code in Azure (call them as “functions”)

Provides out of the box templates for some of the most common scenarios

Useful in common scenarios like –

- Connecting to Storage
- Image processing
- Exposing HTTP based APIs
- IoT
- Running a script or code in response to a variety of events etc

Azure Functions is a serverless evolution of Azure WebJobs

Azure Functions Features

Choice of Language –

- Supports C#, F#, JavaScript, Java, Powershell, Python, Typescript

Pay per use (only for the time the code is executed)

Nuget and NPM support

Integrated Security – Oauth support for Http-triggered functions

Seemless integration with other Azure Services

Flexible Development

Azure Functions runtime is open-source

Azure Functions Runtime

Experience Azure Functions before committing to the cloud

Two components

- Azure Functions Runtime Management Role
- Azure Functions Runtime Worker Role

Runtime version	Release level[1]	.NET version
3.x	GA	.NET Core 3.1
2.x	GA	.NET Core 2.2
1.x	GA[2]	.NET Framework 4.6[3]

[1] GA releases are supported for production scenarios.
[2] Version 1.x is in maintenance mode. Enhancements are provided only in later versions.
[3] Only supports development in the Azure portal or locally on Windows computers.

Azure Functions Runtime Versions

- Language Support

Language	1.x	2.x	3.x
C#	GA (.NET Framework 4.7)	GA (.NET Core 2.2)	GA (.NET Core 3.1)
JavaScript	GA (Node 6)	GA (Node 8 & 10)	GA (Node 10 & 12)
F#	GA (.NET Framework 4.7)	GA (.NET Core 2.2)	GA (.NET Core 3.1)
Java	N/A	GA (Java 8)	GA (Java 8)
PowerShell	N/A	GA (PowerShell Core 6)	GA (PowerShell Core 6)
Python	N/A	GA (Python 3.6 & 3.7)	GA (Python 3.6 & 3.7)
TypeScript	N/A	GA[1]	GA[1]

[1]Supported through transpiling to JavaScript.

Azure Functions Integrations

Functions can be integrated with various Azure and 3rd party services

These services can either trigger the function execution or serve as input/output for function code

Following services can be integrated with Azure Functions:

- Azure CosmosDB
- 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)
- Twilio (SMS messages)

What are Triggers?

One of the Merriam-Webster's definition says –

- "A Trigger is something that acts like a mechanical trigger in initiating a process or reaction"

Defines how a function is invoked

Out-of-the-box templates to trigger execution of an Azure function

A function can have exactly one trigger

A trigger can have an associated data, which is usually the payload that triggered the function

Binding direction for triggers in always in

Supported Triggers

HTTPTrigger

TimerTrigger

CosmosDBTrigger

BlobTrigger

QueueTrigger

EventGridTrigger

EventHubTrigger

ServiceBusQueueTrigger

ServiceBusTopicTrigger

What are Bindings?

Declarative way to make data from external services available to function code

Bindings are optional

Two types of bindings

- Input
- Output

A function can have multiple input and output bindings

Input and output bindings use in and out binding directions

Some bindings support special binding direction – inout

For runtime version 2.x, binding extensions are provided in NuGet packages, and to register an extension, package needs to be installed.

Trigger & Binding Definition

Defined in function.json file.

```
{
  "bindings": [
    {
      "name": "order",
      "type": "queueTrigger",
      "direction": "in",
      "queueName": "myqueue-items",
      "connection": "MY_STORAGE_ACCT_APP_SETTING"
    },
    {
      "name": "$return",
      "type": "table",
      "direction": "out",
      "tableName": "outTable",
      "connection": "MY_TABLE_STORAGE_ACCT_APP_SETTING"
    }
  ]
}
```

Trigger & Binding Definition

```
public static class QueueTriggerTableOutput
{
    [FunctionName("QueueTriggerTableOutput")]
    [return: Table("outTable", Connection = "MY_TABLE_STORAGE_ACCT_APP_SETTING")]
    public static Person Run(
        [QueueTrigger("myqueue-items", Connection = "MY_STORAGE_ACCT_APP_SETTING")]JObject order,
        TraceWriter log)
    {
        return new Person() {
                PartitionKey = "Orders",
                RowKey = Guid.NewGuid().ToString(),
                Name = order["Name"].ToString(),
                MobileNumber = order["MobileNumber"].ToString() };
    }
}

public class Person
{
    public string PartitionKey { get; set; }
    public string RowKey { get; set; }
    public string Name { get; set; }
    public string MobileNumber { get; set; }
}
```

Binding Expressions

Expressions that resolve to values from various sources.

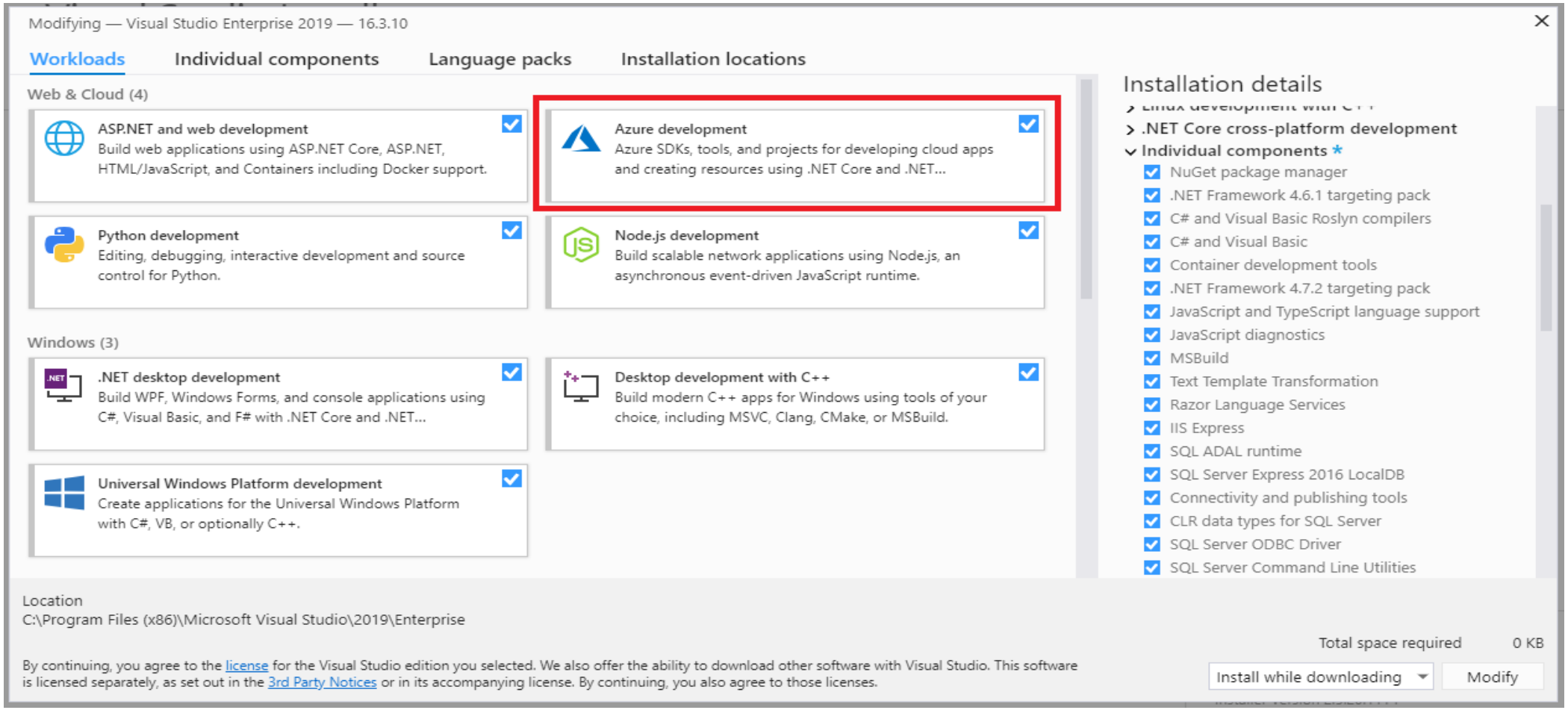
Most expressions are identified by wrapping them in curly braces.

Expressions using AppSettings are wrapped in percent signs

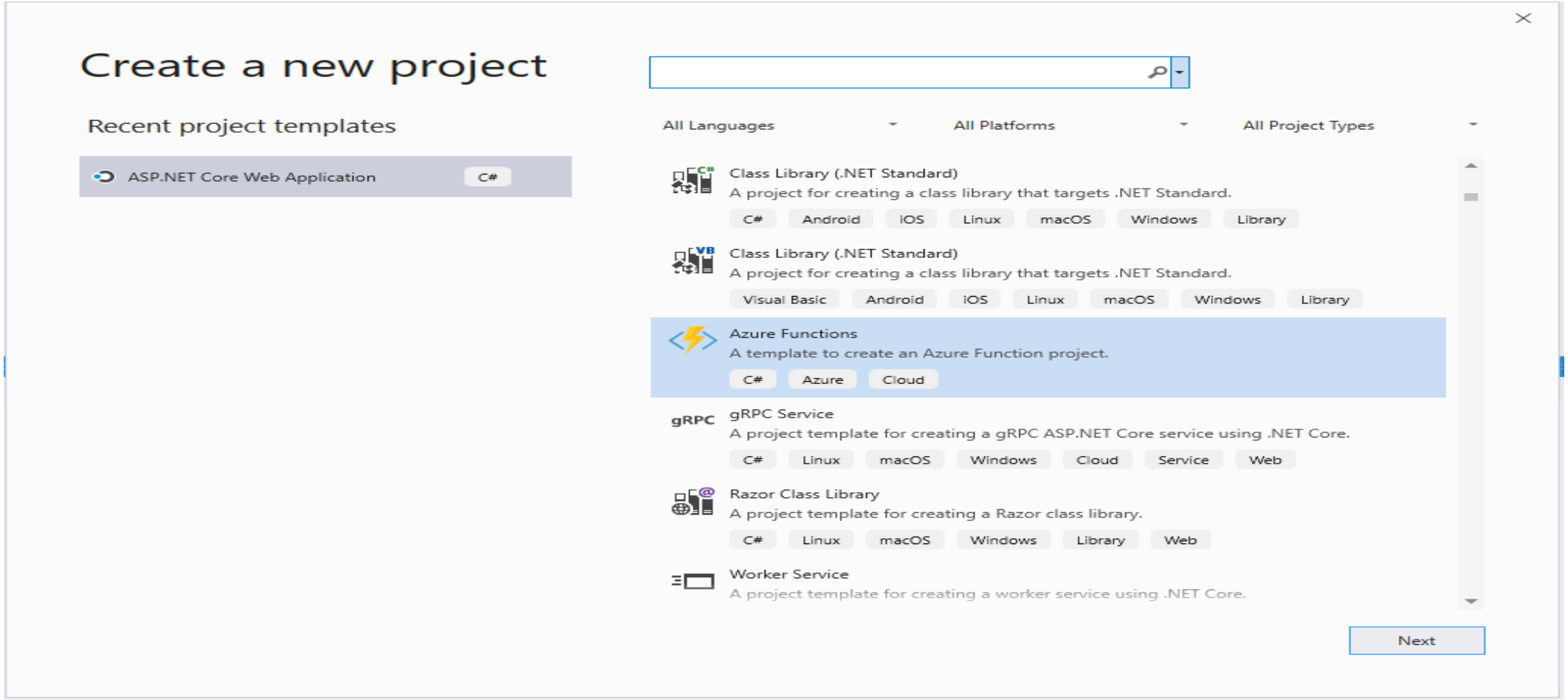
Types –

- App Settings
- Trigger File Names
- Trigger Metadata
- JSON Payloads
- New Guid ({rand-guid})
- Current Date and time ({DateTime})

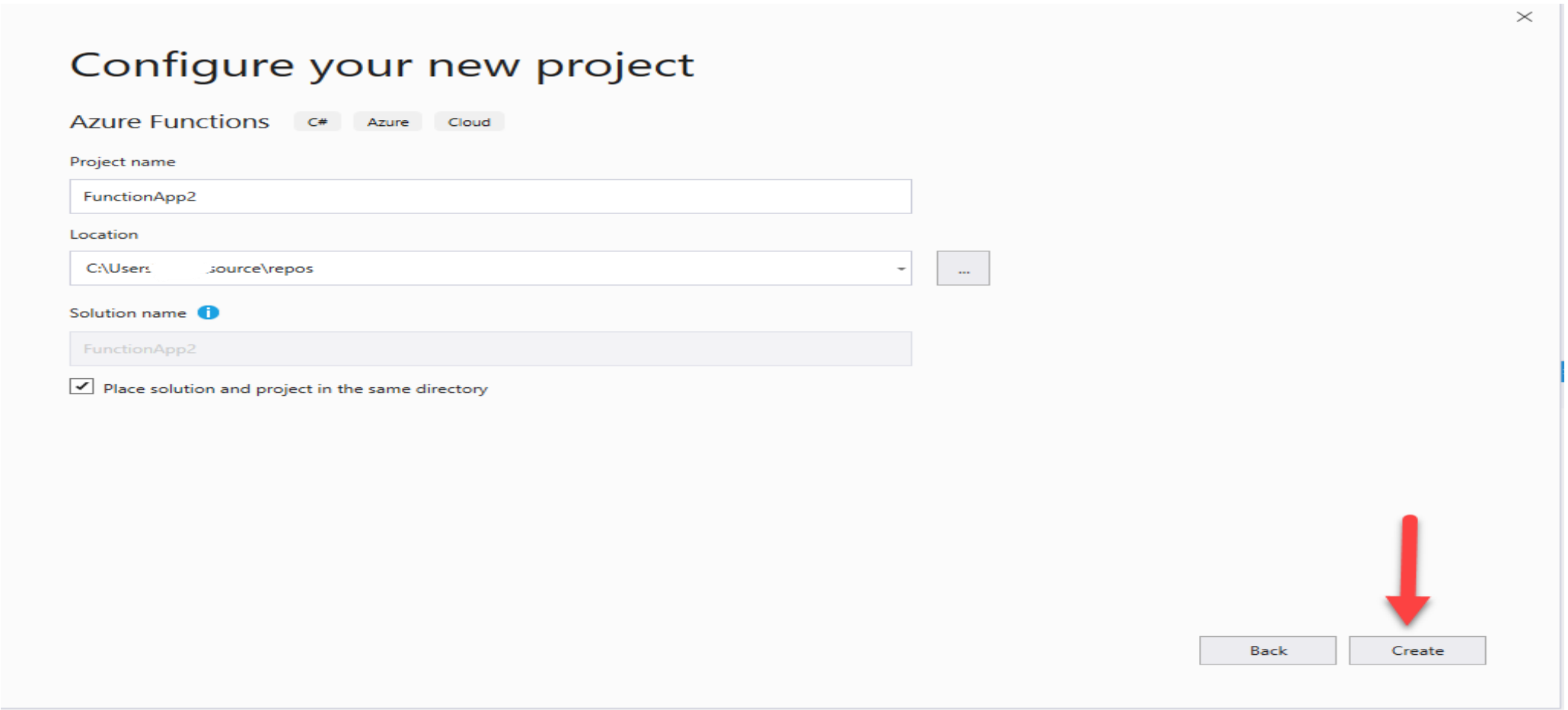
Azure Function Tooling – Visual Studio



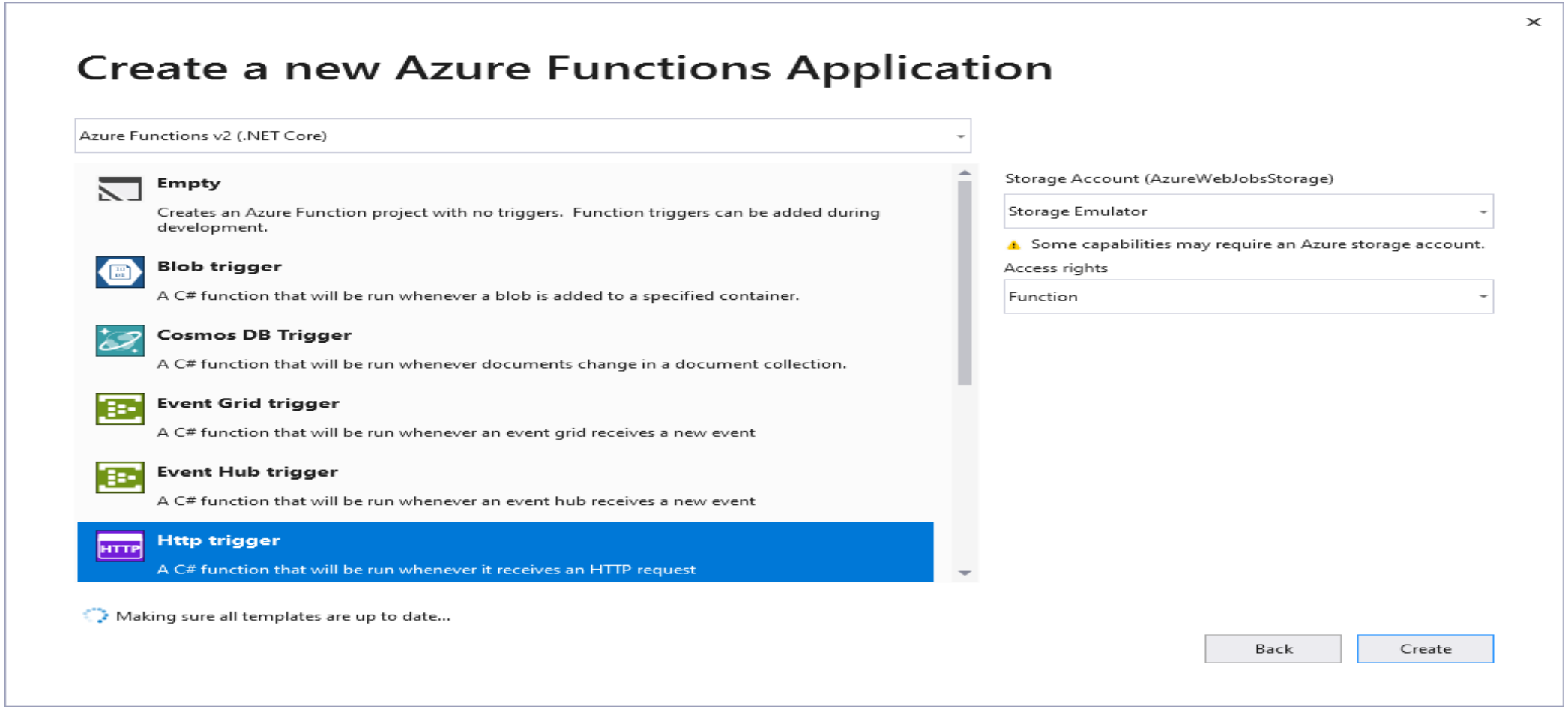
Azure Function Tooling – Visual Studio



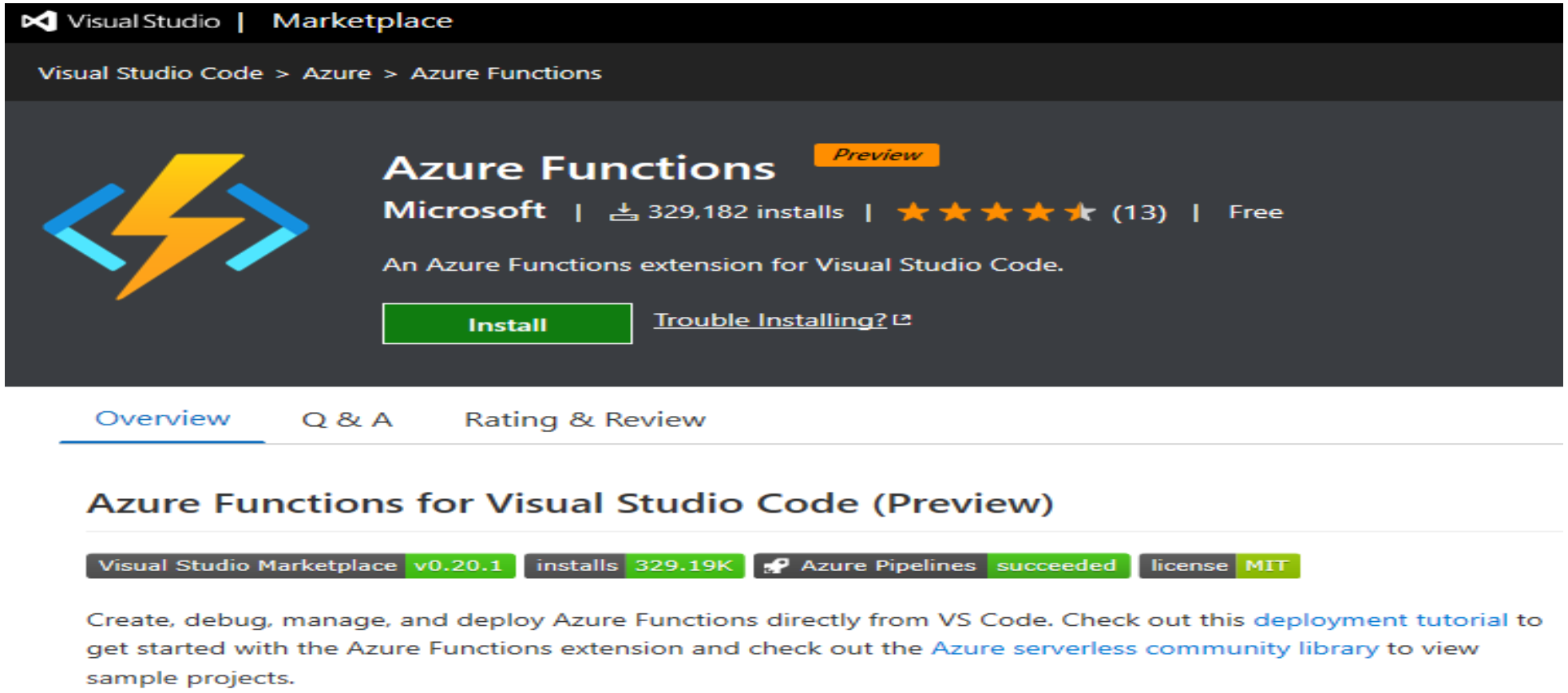
Azure Function Tooling – Visual Studio



Azure Function Tooling – Visual Studio



Azure Function Tooling – VS Code



Azure Functions Core Tools

Azure Functions Core Tools

- Develop/test/debug/deploy functions on local computer from command prompt or terminal

v1.x

- Supported on Windows

v2.x/3.x

- Supports Windows, macOS and Linux
- Needs .Net core 2.0/3.0 and Node.js (including npm)

Azure Functions Pricing

Three pricing plans:

Consumption Plan

- Pay for the time the code is executed
- Default Hosting plan

Premium Plan

- Similar to consumption plan
- Perpetually warm instances to avoid any cold start
- VNET connectivity

Dedicated Plan

- Same as app service plan for a web app
- Enable always on

Consumption Plan

Pay for the time that the code runs

Automatically allocates compute power when code is running

Adds or removes Functions Host (Function App) instances based on the number of events that its functions are triggered on

- Each instance of function host is limited to 1.5 GB of memory
- All functions within same function host share the same resources and are scaled together

Function code files are stored on Azure File shares

Function times out after configurable period of time (functionTimeout property)

- Default – 5 mins
- Maximum – 10 mins

Premium Plan

Similar to Consumption plan

Perpetually warm instances to avoid any cold start

VNet connectivity

Unlimited execution duration

Premium instance sizes (one core, two core, and four core instances)

More predictable pricing

High-density app allocation for plans with multiple function apps

Scaling under consumption plan

Scale Controller

- Monitors the rate of events and determines whether to scale out or scale in
- Uses heuristics for each trigger type

The unit of scale is a function app

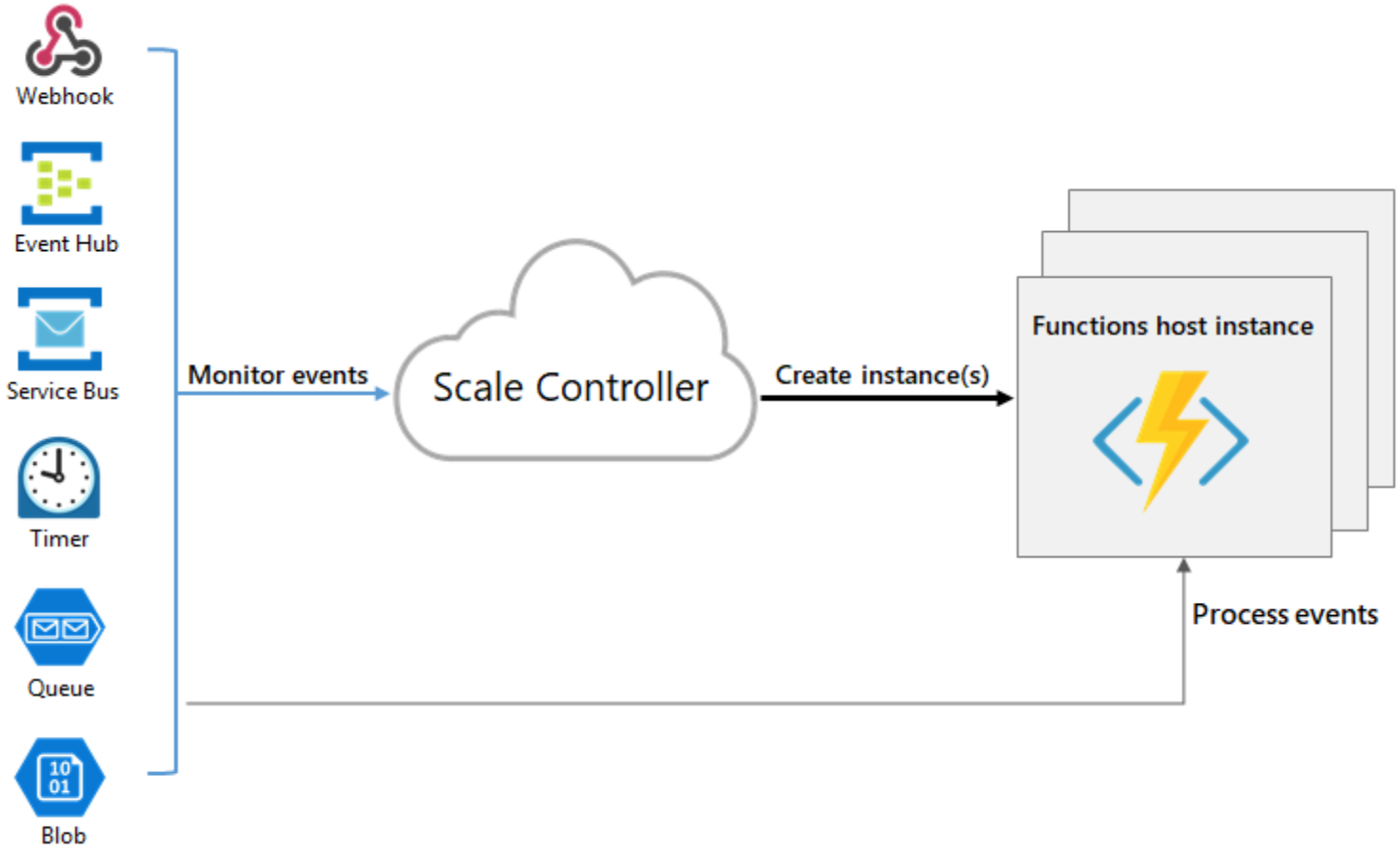
When the function app is scaled out or down, resources are de/allocated to run multiple instances of the Azure Functions host.

The number of instances is scaled down to zero when no functions are running within a function app.

A single function app only scales up to a maximum of 100 instances.

New instances are allocated at most once every 10 seconds.

Scaling under consumption plan



App Service Plan

Same as App Service Plan in App Services or a web app

Dedicated resources based on the chosen tier like Basic or Standard

Fixed cost

Manual scaling and Auto scaling supported

Run Function app just like a web app

Useful if app service plan already exists for a web app and function app can run at no additional cost

Always On

Under App service plan, Function App goes idle after few minutes of inactivity

Only HttpTriggers can wake up a function

Under app service plan, enable “Always on” to keep the functions running continuously and correctly

Under Consumption plan, function apps are activated automatically

When using a blob trigger on a Consumption plan, there can be up to a 10-minute delay in processing new blobs.

Monitor Azure Functions

Built-in logging mechanism based on Azure Storage

- Useful for non-prod environments with light workloads
- Can be disabled by deleting AzureWebJobsDashboard app setting

Built-in support for Application Insights

- Recommended for production workloads
- If this is enabled, built-in logging on Azure Storage should be disabled
- Telemetry data can be further queried using Application Insights Analytics
- Telemetry includes traces, requests, exceptions, customMetrics, customEvents and performanceCounters

Azure Functions – Continuous Deployment

Directly deploy through source control of your choice

Deployments are configured per-functionapp basis

If continuous deployment is enabled, the access to function code in the portal is set read-only

Azure DevOps (previously VSTS) offers full support for Azure Functions

Other ways to deploy-

- Zip Deployment
- Through ARM template
- Through deployment package

Best Practices

Avoid long running functions

- Refactor large functions into smaller function sets

Use Azure Storage queues for cross-function communication

Functions should be stateless and idempotent if possible

Re-use external connections whenever possible

Don’t mix test and production data in same function app

Use async code but avoid blocking calls

Receive messages in batch whenever possible

Write defensive functions

Demo

Durable Functions

An extension of Azure Functions

Enables stateful functions in server-less environment

Simplifies complex, stateful coordination problems in serverless applications

Provides stateful orchestration of function execution

Durable Functions

Built on top of Durable Task Framework

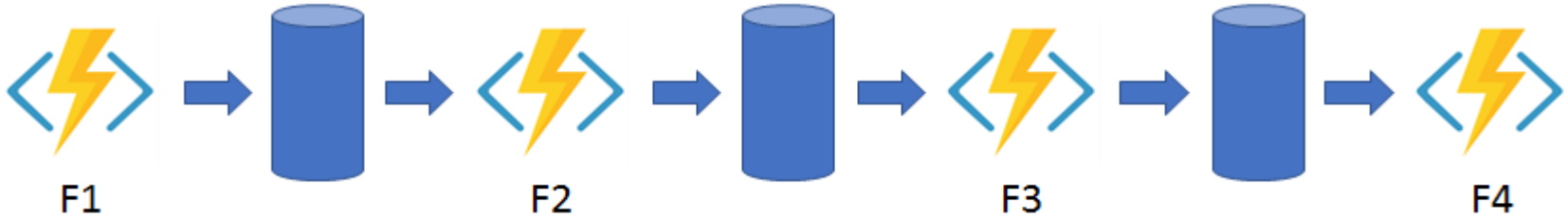
- As a serverless evolution

Some typical application patterns supported by durable functions includes –

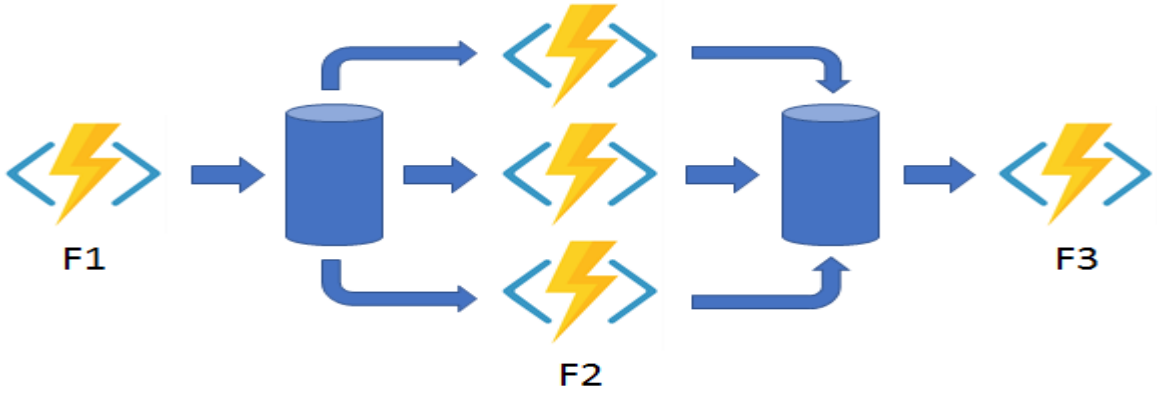
- Function Chaining
- Fan-out/fan-in
- Async Http APIs
- Monitoring
- Human Interaction

Durable Functions

Function Chaining

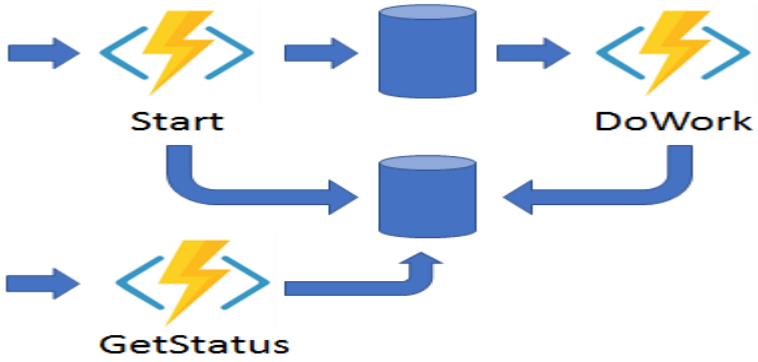


Fan-out/fan-in



Durable Functions

Async Http APIs



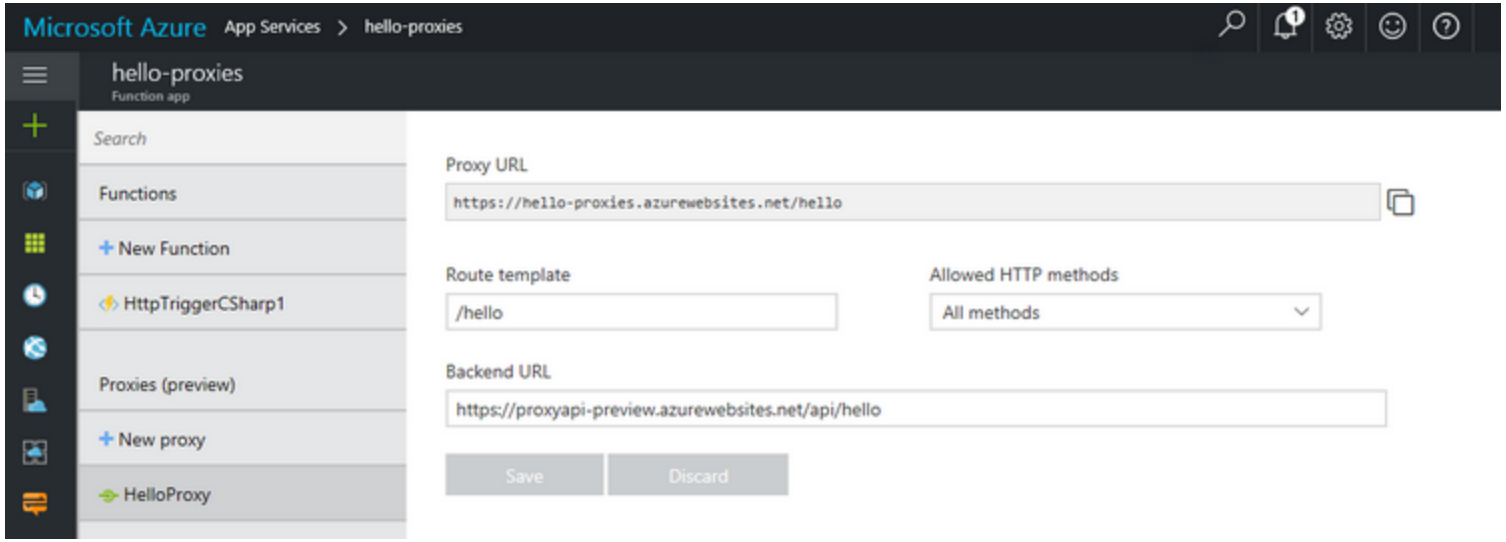
Human Interaction

Azure Function Proxies

Makes it easier to develop APIs by using Azure Functions

Define a single API surface for multiple function apps

Define an endpoint that serves as a reverse proxy to another API



Azure Logic Apps

What are Azure Logic Apps?

Integration platform as a Service (iPaaS)

Schedule, automate, and orchestrate tasks, business processes, and workflows

Design and build scalable solutions for

- App integration
- Data integration
- System integration
- Enterprise application integration (EAI)
- Business-to-Business (B2B) communication

Ready-to-use and custom connectors

Benefits

Visually build workflows with easy-to-use tools

First-class support for enterprise integration and B2B scenarios

Write once, reuse often

Built-in extensibility

Pay only for what you use

Triggers and Actions

A *trigger* is the first step in any logic app, usually specifying the event that fires the trigger and starts running your logic app.

Actions are the steps that follow the trigger and perform tasks in your logic app's workflow.

After a trigger fires, Azure Logic Apps creates an instance of your logic app and starts running the *actions* in your logic app's workflow.

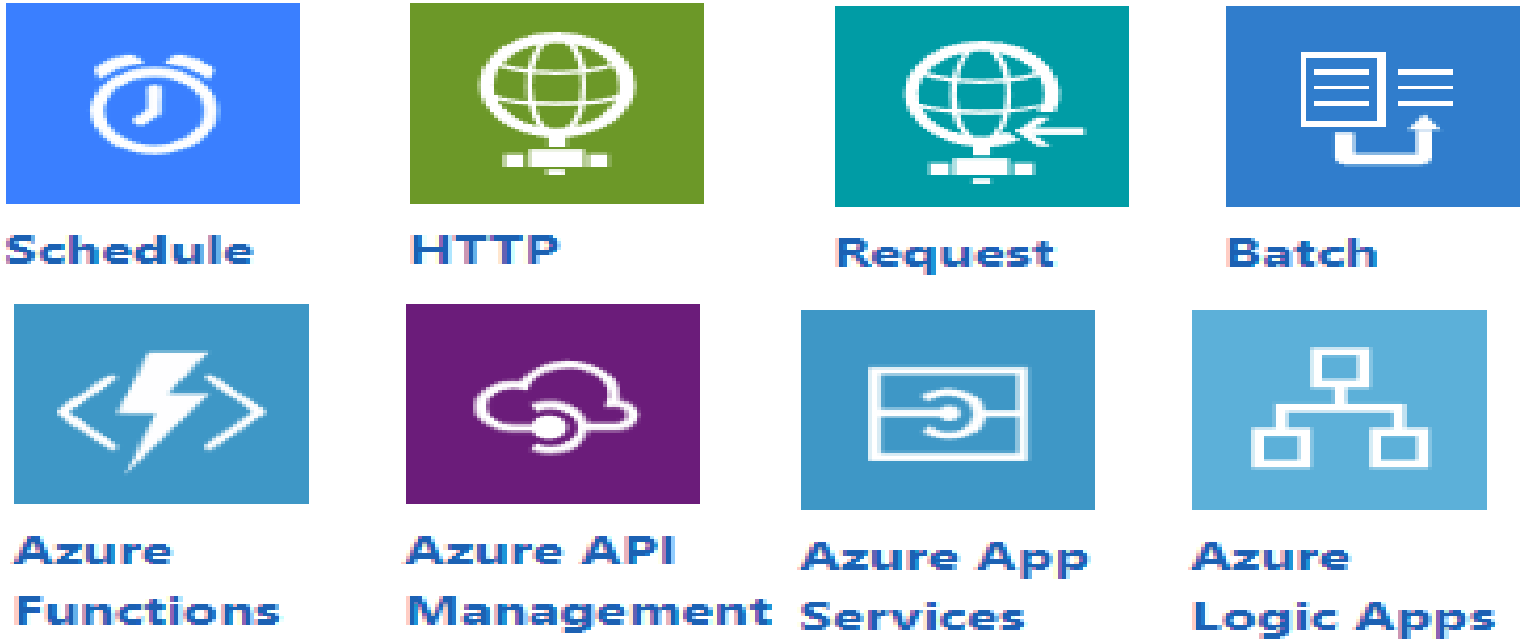
3 kinds of triggers –

1. Recurrence Trigger
2. Polling Trigger
3. Push Trigger

Every time a Logic App definition runs the triggers, action and connector executions are metered.

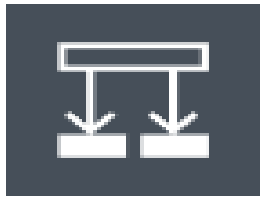
Built-in Connectors

Triggers

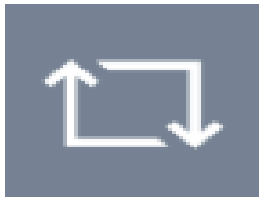


Built-in Connectors

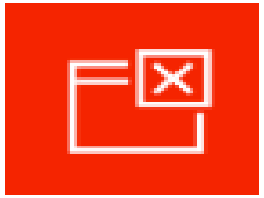
Control Workflow



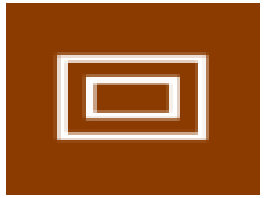
Condition



For each

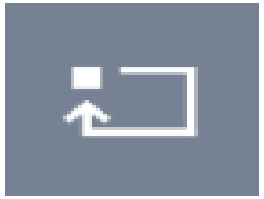


Terminate



Scope

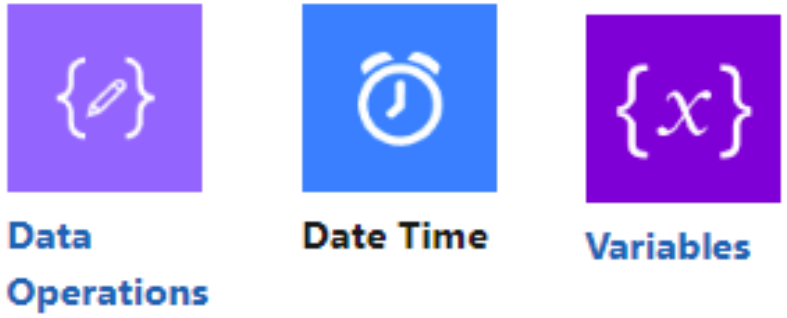
Switch



Until

Built-in Connectors

Data Manipulation



Managed API connectors

Azure Service Bus

SQL Server

SFTP

SharePoint Online

Salesforce

Twitter

Office 365 Outlook

Azure Blob Storage

Dynamics 365 CRM Online

FTP

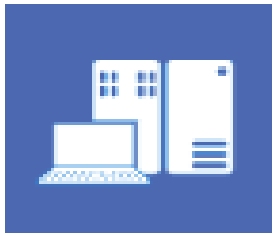
Azure Event Hubs

Azure Event Grid

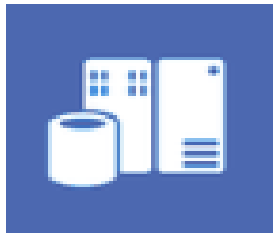
On-premises connectors

Integration Account connectors

Enterprise connectors



IBM 3270



IBM MQ

SAP

Demo

Thank You…!