1. **Choose the best Azure service to automate your business processes**

Objective:

Evaluate Azure services for integration and process automation scenarios

Azure provides several different ways to host and execute code or workflows without using VMs including Azure functions, Microsoft Power Automate, Azure Logic Apps, Azure WebJobs.

**Common business issues.**

In business, one way to guarantee high-quality products and service to customers is to design and implement strict business processes. Such processes may involve multiple steps, people, and software packages.

Problems arise when it merges a second business or integrates with a partner organization. How can administrators integrate the separate processes used in the two organizations, which may have been implemented using different software?

Business processes modeled in software are called **workflows.** Azure includes 4 different technologies that you can use and integrate systems

* Logic apps
* Microsoft power automate
* Web jobs
* Azure functions

They have some similarities:

* They can all accept inputs. An input is a piece of data or a file that is supplied to the workflow
* They can all run actions. An action is a simple operation that the workflow executes and may often modify data or cause another action to be performed.
* They can all include conditions. A condition is a test, often run against an input, that may decide which action to execute next.
* They can all produce outputs. An output is a piece of data or a file that is created by the workflow.

In addition, workflows created with these technologies can either start based on a schedule or they can be triggered by external event.

**Design-first technologies.**

When business analysts discuss and plan business process, they may draw a flow diagram on paper. With logic apps and Microsoft power automate, you can take a similar approach to designing a workflow. They both include user interfaces in which you can draw out the workflow. We call this approach a design first approach.

**Logic apps**

It’s a service within Azure, to automate, orchestrate and integrate disparate components of a distributed application. By using the design-first approach in Logic apps, you can draw out complex workflows that model complex business processes. The following screenshot shows the logic apps designer and design canvas that you use to define your workflow.

Alternatively, you can create or edit a workflow in JSON notation by using the code view.

One reason why Logic apps is so good at integration is that over 200 connectors are included. A connector is a Logic Apps component that provides an interface to an external service. For example. Twitter connector allows you to send and retrieve tweets, office 365 connector lets you manage your email, calendar, contacts. Logic Apps provides hundreds of pre-built connectors that you can use to create your apps. If you have an unusual or unique system that you want to call from logic apps, you can create your own connector if your system exposes a REST API.

**Microsoft Power Automate**

Is a service to create workflows even when you have no development or IT pro experience.

You can create workflows that integrate many different components by using the website or mobile app.

There are four different types of flow you can create:

* **Automated.** Starts by a trigger from some event. For example, the event could be the arrival of a new tweet, or a new file being uploaded.
* **Button.** Runs a repetitive task with a single click from your mobile device.
* **Scheduled.** Runs on a regular basis such like once a week, on a specific date, or after 10 hours
* **Business process.** Models a business process such as the stock ordering process or the complaints procedure. The flow process can have: notification to required people; with their approval recorded; calendar dates for steps, recorded time of flow steps.

Microsoft Power Automate provides an easy-to-use design surface to create flows of the above types.

Under the hood, Microsoft power automate is built on logic apps. Meaning that Power automate supports same range of connectors and actions.

**Design-first technologies compared**

Microsoft Power Automate is more appropriate for use by non-technical staff. If your workflow designers are IT professionals, Logic apps are usually a better fit.

**Code-first technologies**

The developers on your team will likely prefer to write code when they want to orchestrate and integrate different business applications into a single workflow. This is when you need more control over the performance of the workflow or need to write custom code as part of the business process. For such cases, Azure includes Web Jobs and Functions.

**Web Jobs and the Web Jobs SDK**

The Azure app service is cloud based hosting service for web applications, mobile back-ends and RESTful APIs. These applications often need to perform some kind of background task. For example, when photo is uploaded, you may need to generate a smaller thumbnail photograph.

Web Jobs are part of the Azure App Service that you can use to run a program or script automatically. There are two kinds of Web Job:

* **Continuous.** Run in a continuous loop. For example, you could use a continuous Web job to check a shared folder for a new photo.
* **Triggered.** Run when you manually start them or on a schedule.

You can write code in different languages, like PowerShell, Bash. Alternatively, write program in Php, Python, Node.js or Js. Sdk 3.x supports .NET Core.

You can also program a WebJob by using the .NET framework or the .NET Core Framework. In this case, you can use WebJobs SDK to make the task easier. The SDK includes a range of classes, which reduce the amount of code required to interact with the Azure App Service.

WebJobs SDK only supports C# and the NuGet package manager.

**Azure functions**

An azure function is a simple way for you to run small pieces of code in the cloud, without having to worry about the infrastructure required to host that code. You can write Function in C#, Java, JS, Python or any of the languages listed. In addition, you can only pay for the time when the code runs.

When you create an Azure Function, you can start by writing the code for it in the portal. Alternatively, if you need source control, you may use GitHub or Azure DevOps.

To create Azure Function, choose from the range of templates. The following list is a sample of some of the templates available to you:

* HttpTrigger. When you want the code to execute in response to a request sent through the HTTP protocol
* TimeTrigger. When you want the code to execute according to a schedule.
* BlobTrigger. When you want the code to execute when a new blob is added to an Azure Storage account.
* CosmosDbTrigger. When you want the code to execute in response to new or updated documents in a NoSql database.

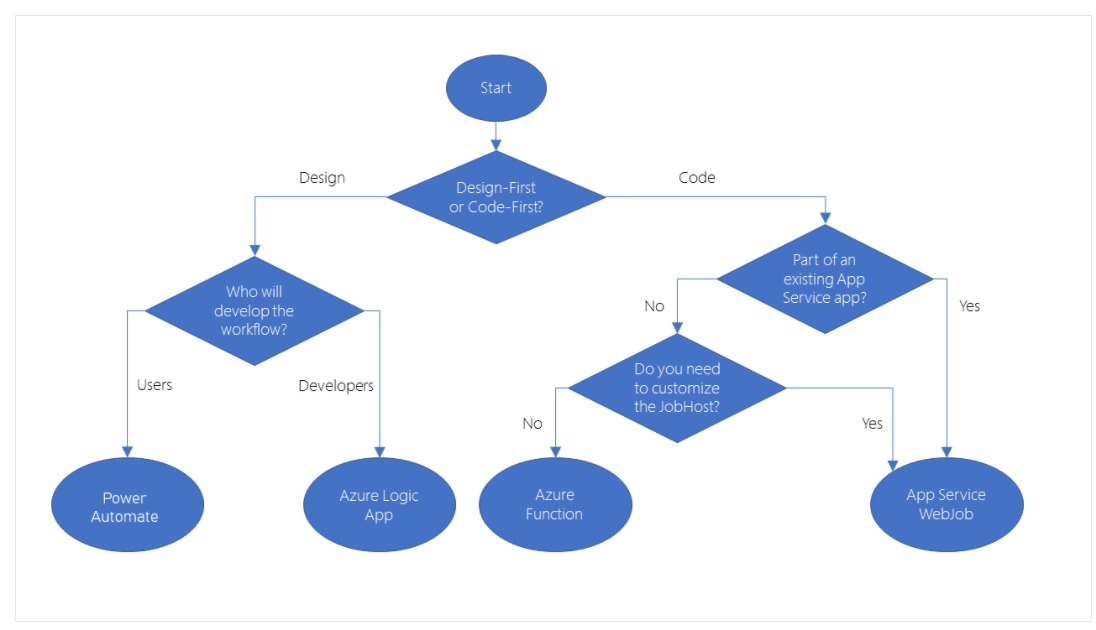
Azure functions can integrate with many different services. These services can trigger your function, or send data input to your function, or receive data output from your function.

**Code-first technologies compared**

In most cases, simple administration and more flexible coding model provided by Azure functions may lead you to chose them in preference to webjobs. However, you may choose WebJobs for the following reasons:

* You want the code to be part of an existing App Service application and to be managed as part of that application, for example in the same Azure DevOps environment.
* You need close control over the object that listens for events that trigger the code.

**Analyze the decision criteria**



First question to ask is if you want GUI or by writing code.

Valid reasons for using a design-first tool:

* People who design the workflow have no coding experience.
* Later designers and users can consult the graphical design to clearly understand how the workflow proceeds

Code-first tool, because:

* People who design the workflow are developers and prefer to work entirely in code
* You want the details of a workflow to be hidden from non-coders.

**Choosing a design-first technology**

The principal question here is who will design the workflow: will it be developers or users?

In logic apps, its designed for people with development skills.

In power automate, its designed for users who have a good understanding of the business process but no coding skills.

**Choosing a code-first technology**

Azure functions should be your default choice (ability to develop test code in the browser, pay per use price model, wider ranges of supported languages and trigger events).

There are situations when WebJobs might be a better choice:

* You have an existing Azure app service application, and you want to model the workflow within the application. This requirement means that the workflow can also be managed as part of the application, for example in Azure DevOps environment.
* You have specific customizations, for example custom retry policy for calls to external systems.

**Mixing technologies**

Remember that there is no requirement for you to use the same technology for different workflows: if your requirements differ, you are likely to reach a different answer at the end of your decision-making process. Furthermore, you can also call one workflow from another. For example, a workflow implemented in Power Automate can easily call another that is built as an Azure Function.

One reason to mix the technologies used in your business processes would be to give users control over a small section of complete workflow.

1. **Create serverless logic with azure functions**

With serverless computing, your cloud provider manages the provisioning and maintenance of the infrastructure letting you focus completely on building the app logic. Azure functions is a key component of the serverless computing offering from Azure and enables you to run peaces of code or functions, written in programming language of your choice, in the cloud.

**What is serverless compute?**

Serverless compute can be thought of as a function as a service (FaaS), or a microservice that is hosted on a cloud platform. Your business logic runs as functions, and you don’t have to manually provision or scale infrastructure. The cloud provider manages infrastructure. Your app is automatically scaled out or down depending on load. Azure has Azure logic apps and Azure functions; this will be our focus in this module.

**What is azure functions?**

Azure functions are a serverless application platform. It enables developers to host business logic that can be executed without provisioning infrastructure. Good scalability and you are charged only for the resources used. You can write code in many different languages. Support for NuGet is also included, so you can use popular libraries.

**Benefits of a serverless compute solution**

Serverless compute is great option for hosting business logic code in the cloud. You get automatic scaling, you have no servers to manage, and you are charged based on what is used. Here are some additional characteristics of serverless solution for you to consider:

* Avoids over-allocation of infrastructure. Scaling up or down automatically.
* Stateless logic. Function instances are created and destroyed on demand. If state is required, it can be stored in an associated storage service.
* Event driven. They run only in response to an event (trigger), such as receiving an http request, or message being added to a queue. You configure a trigger as part of the function definition. Allows you to declare where data comes from and where it goes. No need to write code to watch queues, blobs, hubs, and so on. You can focus purely on the business logic.
* Can be used in traditional compute environments.

**Drawbacks of a serverless compute solution**

Serverless compute will not always be the appropriate solution for hosting business logic. Here are a few characteristics of functions that may affect your decision to host services in serverless compute:

* Execution time. Functions have timeout of 5 minutes (max10). If function requires more than 10 minutes to execute, you can host it on a VM. Additionally, if your service is initiated trough an http request and you expect an http response, timeout is restricted to 2.5 minutes.
* Execution frequency. If you expect your function to be executed continuously by multiple clients, it would be smart to estimate the usage and calculate the cost of using functions accordingly. It might be cheaper to host your service on a VM.

While scaling, only one function app instance can be created every 10 seconds, for up to 200 total instances. Keep in mind, each instance can service multiple concurrent executions, so there is no set limit about how much traffic a single instance can handle. Different types of triggers have different scaling requirements, so research your choice of trigger and investigate limits.

**Azure function triggers**

* Blog storage. Starts when a new or uploaded blob is detected.
* Azure Cosmos DB. Start a function when inserts and updates are detected.
* Event grid. Starts a function when an event is received from Event Grid.
* HTTP. Starts a function with an HTTP request.
* Microsoft Graph Events. Starts a function in response to an incoming webhook from the Microsoft Graph. Each instance of this trigger can react to one Microsoft graph resource type
* Queue Storage. Starts a function when a new item is received on a queue. This queue message is provided as input to the function.
* Service bus. Starts a function in response to messages from a Service Bus queue.
* Timer. Starts a function on a schedule.

**Azure function bindings**

A binding Is a declarative way to connect data and services to your function. Bindings interact with various data sources, which means you don’t have to write code in your function to connect to data sources and manage connection, platform takes care of that complexity for you as part of the binding code. Each binding has a direction. Code reads data from input bindings and writes data to output bindings. Each function can have zero or more bindings.

A trigger is a type of input binding that can initiate execution of some code.

**Function templates**

Azure offers 20~ templates. Once created you can further customize.

Test in 2 ways. Azure portal or trigger yourself (like, send http message if trigger is http)

1. **Execute an Azure function with triggers**

In this module, you will:

* Determine which trigger works best for your business needs
* Create a timer trigger to invoke a function on a consistent schedule
* Create an HTTP trigger to invoke a function when an HTTP request is received
* Create a blob trigger to invoke a function when a blob is created or updated in Azure Storage

Azure Functions app doesn’t do work until something tells it to execute. For example, we could create an Azure function to send out a reminder text message to our customers before an appointment. If we don’t tell the function when it should run, our customers will never receive a message.

Every function must have exactly one trigger associated with it. If you want to execute a peace of logic that runs under multiple conditions, you need to create multiple functions that share the same core function code.

**Time trigger**

Something that executes at a certain time. (Weekly, daily, hourly)

To create a time trigger, you need to supply two pieces of information.

* A timestamp parameter name, which is simply an identifier to access the trigger in code.
* A Schedule, which is CRON expression that sets the interval for the timer.

**What is a CRON expression?**

A Cron expression is a string that consists of six fields that represent a set of times.

The order of six fields in Azure is: {second} {minute} {hour} {day} {month} {day of the week}

Example that executes every five minutes looks like: 0 \*/5 \* \* \* \*

Special characters:

* **\*** Selects every value in a field. In week this means every day
* **,** Separates items in a list. A comma “1,3” in the day of the week means just Mondays and Wednesdays
* **-** Specifies a range. 10-12in hour field means 10, 11 and 12
* **/** Specifies an increment. \*/10 in minutes means an increment of every 10 minutes.

Break down example. 0 in seconds, selects first possible value.

\*/5 means select every fifth value 0-59

Remaining fields are all \*

When put all together, means the first second of every fifth minute of every hour, of every day, of every month

**Execute an Azure function with an HTTP request**

Http request is a common operation on most platforms and devices. Azure allows us to quickly create a piece of logic to execute when an http request is received.

**What is an HTTP trigger?**

It’s a trigger that executes a function when it receives an Http request. Http triggers have many capabilities and customizations, including:

* Provide authorized access by supplying keys.
* Restrict which HTTP verbs are supported.
* Return data back to the caller.
* Receive data trough query string parameters or through the request body.
* Support URL route templates to modify the function URL.

When you create an HTTP trigger, you need to select a programming language, provide a trigger name, and select an Authorization level.

**What is an HTTP trigger Authorization level?**

An HTTP trigger Authorization level is a flag that indicates whether an incoming HTTP request needs an API key for authentications.

There are 3 levels:

* Function
* Anonymous
* Admin

The Function and Admin levels are “key” based. To send an HTTP request, you must supply a key for authentication. There are two types of keys: function and host. The difference between these keys is their scope. Function keys are specific to a function. Host keys apply to all functions inside the function app. If your Authorization level is set to Function, you can use either a function or a host key. If Authorization level is set to Admin, you must supply a host key.

The anonymous level means that authentication is not required. This exercise uses anonymous auth level.

1. **Chain Azure Functions together using input and output bindings**

Suppose you run social networking site. You allow to upload images to be posted on their profile. To reduce workload on the web server, you want to create serverless back end using Azure functions to process this data. You want to create image thumbnail and save it to permanent storage.

With bindings, developers interact with other data sources and services without worrying about how the data flows to and from their function.

**Input and output binding types**

Accessing and processing data are key tasks in many software solutions. Consider some of these scenarios:

* You’ve been asked to implement a way to move incoming data from Blob storage to Cosmos DB.
* You want to post incoming messages to a queue for processing by another component in your enterprise.
* Your service needs to grab gamer scores from a queue and update an online scoreboard.

All these examples are about moving data. The data source and destinations differ from scenario to scenario. But the pattern is similar. You connect to a data source, and you read and write data. Azure Functions helps you integrate with data and services by using bindings.

**What is binding?**

In Azure functions, bindings provide a declarative way to connect to data within your code. They make it easier to integrate with data streams consistently in a function. You can have multiple bindings providing access to different data elements. You can connect to your data sources without having to code specific connection logic.

**Types of bindings**

Two types of bindings you can use with functions:

* Input binding. Connect to a data source. Function can read data from these inputs.
* Output binding. Connects to a data destination. Our function can write data to these destinations.

There are also triggers, which are special types of input bindings that cause function to run.

**Types of supported bindings**

A binding type can be used as an input, an output, or both. For example, a function can write to a Blob Storage output binding, but a Blob Storage update could trigger another function.

Some common binding types follow:

* Blob storage
* Azure Service Bus queues
* Azure Cosmos DB
* Azure Event hubs
* External files
* External tables
* Http Endpoints

These types are just a sample.

**Binding properties**

Three properties are required in all bindings. You may have to supply additional properties based on the type of binding and storage you are using:

* **Name**. Defines the function parameter trough which you access the data (for example, in a queue input binding, this is the name of the function parameter that receives the queue message content)
* **Type**. Identifies the type of binding (for example, the type of data or service we want to interact with)
* **Direction**. Indicates the direction data is flowing (for example, input or output.)
* **Connection** (most types, but not all). Provides the name of an app setting key that contains the connection string. Bindings use connection string stored in app settings to keep secrets out of the function code. This makes your code more configurable and secure.

**Read data with input bindings**

To connect to a data source, you need to configure an input binding. An input binding makes it possible to write minimal code to create a message. You don’t need to write code for tasks, such as opening storage connection. Azure Functions take care of those tasks for you.

**Input binding types**

Azure chain functions can have multiple types of input. However, not all types support both input and output. You’ll use an input function whenever you want to ingest data of that type. Here, we’ll look at the input function types that support input bindings and when to use them.

* Azure blob storage. Blob storage bindings allow you to read from a blob.
* Azure Cosmos Db. Input bindings use the SQL API to retrieve one or more Cosmos DB documents and pass them to the input parameter of the function.
* Mobile Apps. Input binding loads a record from a mobile table endpoint and passes it into your function.
* Azure table storage. Read data and work with Azure table storage.

These are not all, but just common examples.

**What is a binding expression?**

There are several types of binding expressions, including:

* App settings
* Trigger filename
* Trigger metadata
* JSON payloads
* New GUID
* Current date and time

Most binding expressions are identified by being wrapped in curly braces. However, app setting binding expressions are wrapped in percent signs. For example, if the blob output binding path is %Environment$/newblob.txt, and the environment app setting value is Development, a blob is created in the development container.

Input bindings allow you to connect your function to a data source. You can connect to several types of data sources, and the parameters for each vary. To resolve values from input sources, use binding expressions in the function.json file, in function parameters, or in code.

**Write data with output bindings**

Output binding types:

* Blob storage. You can use the blob output binding to write blobs.
* Azure cosmos db. Lets you write a new document to a Cosmos Db database using SQL API
* Event hubs. To write an event to an event stream.
* Http. Http output binding to respond to the Http request sender. Requires Http trigger, allows to customize the response.
* Microsoft Graph. Allow to write to files in OneDrive, modify excel data, send email trough outlook
* Mobile apps. Writes new record to mobile apps table
* Notification hubs. You can send push notifications with notification hubs output bindings.
* Queue Storage. Use Azure Queue Storage output binding to write messages to queue.
* Send Grid. Send emails using SendGrid bindings.
* Service Bus. Use azure Service bus output binding to send queue or topic messages.
* Table Storage. Se Azure Table Storage output binding to write to a table in Azure Storage account.
* Twilio. Send text messages with Twilio.

To create a binding as output, define direction as out.

1. **Create a long-running serverless workflow with Durable Functions**

Durable functions is an extension of Azure functions that lets you perform long-lasting, stateful operations in Azure. Azure provides infrastructure for maintaining state information. You can use durable functions to orchestrate a long-running workflow. You get all the benefits of a serverless hosting model, while letting durable functions framework to take care of activity monitoring, synchronization, and runtime concerns.

**What is durable functions?**

Durable functions allows you to implement complex stateful functions in a serverless-environment.

**Durable functions**

Is an extension of Azure Functions. Whereas Azure functions operate in a stateless environment, durable functions can retain state between function calls. This approach enables you to simplify complex stateful executions in a serverless environment.

Durable functions scales as needed, provides a cost-effective means of implementing workflows in the cloud. Some benefits of Durable functions include:

* They enable you to write event driven code. Durable function can wait asynchronously for one or more external events, then perform a series of tasks in response to those events.
* You can chain functions together. You can implement common patterns such as fan-out/fan-in, which uses one function to invoke others in parallel, then accumulate the results.
* You can orchestrate and coordinate functions and specify the order in which functions should execute.
* The state is managed for you. You don’t have to write your own code to save state information for a long-running function.

Durable functions allows you to define stateful workflows using an orchestration function. It provides these extra benefits:

* You can define workflows in code. You don’t need to write a JSON description or use a workflow design pool
* Functions can be called sync and async. Output from the called functions is saved locally in variables and used in subsequent function calls.
* Azure checkpoints the progress of a function automatically when the function awaits. Azure may choose to dehydrate the function and save its state while the function waits, to preserve resources and reduce costs. When the function starts running again, Azure will rehydrate it and restore its state.

**Function types**

You can use three durable function types: Client, Orchestrator, and Activity:

* Client. These are entry point for creating an instance of a Durable Functions orchestration. They can run in response to an event from many sources, such as new HTTP request arriving, a message being posted to a message queue, an event arriving in the event stream. You can write them in any of the supported languages.
* Orchestrator. These functions describe how actions are executed, and the order in which they are run. You write orchestration logic in code (C# or Js).
* Activity. These functions are basic units of work in durable function orchestration. An activity function contains the actual work performed by the tasks being orchestrated.

**Application patterns**

You can use durable functions to implement many common workflow patterns. These include:

* Function chaining. In this pattern, execution happens for a sequence of functions in a specified order. The output of one function is applied to the input of the next function in the sequence. The output of the final function is used to generate a result.
* Fan out/fan in. This pattern runs multiple functions in parallel and waits for all the functions to finish. You can aggregate the results of the parallel executions or use them to compute a result.
* Async Http API. This pattern addresses the problem of coordinating state of long-running operations with external clients. An Http call can trigger the long-running action. Then it can redirect the client to a status endpoint.
* Monitor. This pattern implements a recurring process in a workflow, possibly looking for a change in state. For example, you could use this pattern to poll unit specific conditions are met.
* Human interaction. This combines automated process that also involve some human interaction. A manual process within an automated process is tricky because people aren’t as highly available and as responsive as most computers. Human interaction can be incorporated using timeouts and compensation logic. An approval process is an example of a process that involves human interaction.

**Comparison with logic apps**

Durable functions and logic apps are both azure services that enable serverless workload. Durable functions is intended as a powerful serverless compute option to run custom logic. Azure logic apps is better suited for integrating Azure services and components. Key differences:

* Development. Azure durable functions (code first). Azure logic apps (design-first).
* Connectivity. Azure durable functions (Dozen built-in binding types.). Azure logic apps (Large collection of connectors. Enterprise integration pack for b2b).
* Actions. Azure durable functions (Each activity is Azure function). Azure logic apps (Large collection of ready-made actions).
* Monitoring Azure durable functions (Azure application insights). Azure logic apps (Azure portal).
* Management. Azure durable functions (Rest api, powershell, visual studio). Azure logic apps (Azure portal, Rest api, powershell, visual studio).

**Design a workflow based on Durable functions.**

You can use durable functions to orchestrate a long-running workflow as a set of activities. You can map each step in the process to a function type, and each task to an activity.

We need to consider the cases when a step in the process does not complete in time and needs to be escalated. Escalation steps are useful to the business, as they move a task along when a deadline has been reached. They ensure tasks are completed and not forgotten.

Durable functions lets you implement long-running workflows without requiring that you maintain state information manually. Azure provides the infrastructure.

1. **Develop, test, and public Azure Functions by using Azure Functions Core Tools**

Azure Functions Core Tools are command-line utilities that let you develop and run functions locally, then publish them on azure.

**What are the Azure functions core tools?**

Azure functions core tools are a set of command-line tools that you can use to develop and test Azure Functions on your local computer.

Primary purpose is to:

* Generate the files and folders you need to develop functions on your local computer
* Run functions locally so you can test and debug them
* Publish your functions to Azure.

Core Tools are packaged as a single command line utility named “func”. If you run “func” from the command line without any other commands, it will display version information and a usage guide.

Core tools are stand alone utilities, not an extension of a larger integrated development environment or tool. Use any text editor you like to write code and modify a configuration. In practice, you’ll also need the Azure Cli or Azure PowerShell to sign in to Azure and carry out other management tasks.

Visual studio Azure functions extension for visual studio code are built on top of core tools.

**Local development vs Azure portal development**

In most cases, azure portal doesn’t support modifying functions that you develop locally. Once you start using a local development workflow based on Core Tools, you can’t use the Azure portal to make changes to your functions.

**Function apps and functions projects**

Every function published to Azure belongs to a function app, which is a collection of one or more functions that azure publishes together into the same environment. All functions in function app share a common set of configuration values. Build them all for the same language runtime. A function app is an Azure resource that can be configured and managed independently.

When you develop functions locally, you work within a functions project. The project is a folder that contains the code and configuration files that define your functions. A functions project on your computer is equivalent to a function app in Azure, and can contain multiple functions that use the same language runtime.

To start developing locally on your computer, you need to create a functions project folder that’s organized correctly. Every new function you add to the project requires additional code and a configuration. They must be complete and correctly structured, or your functions won’t run.

If you want, you can become familiar with the names, contents and structure of the files and create them yourself. Doing this is time-consuming and error prone.

With azure functions core tools, you’ll never need to create them yourself.

To create a new functions project, run ‘func init’ on the command line.

When you create functions project, the files included in the project folder depend on the language runtime you select. However, the two most critical project files are always present:

* **Host.json** stores runtime configuration files, such as logging options, for the function app. The settings stored in this file are used both when running functions locally and in Azure.
* **Local.settings.json** stores configuration values that only apply to the function app when it’s run locally with the Core tools. This file contains two kinds of settings
  + **Local runtime settings.** Used to configure the local functions runtime itself.
  + **Custom application settings.** You add andconfigure them based on your app’s needs. All the functions in the app can access and use them.

Projects that func init generates don’t have any functions in them. Let’s find out how to add one.

Running func new in a functions project folder will create a new function and all the files you need to get started developing.

**Run functions locally**

Functions aren’t programs that can be run on their own: they must be hosted. The function host is what powers everything outside of your function code: it loads the configuration, listens for triggers and http requests, starts the worker process, writes log output, and more. In Azure, function apps run the function host automatically when they start.

You can use Core tools to run your own instance of the functions host and try out locally before you publish them. You can teste out your functions by making real HTTP calls to them without the need for Azure resources.

To start functions host locally, run ‘func start’ from a functions project folder. At the end of the output, Core Tools will display local URLs so you can use to call each of your functions.

To publish a functions project run ‘func azure function public <appname>’ from the functions project folder. App name is the name of the target function app in azure, not the name of your project folder, which can be different.

Core tools don’t ask you to sign in to azure. Instead, they access your subscriptions and resources by loading your session information from the Azure CLI or Azure powershell.

So, you need Azure CLI or Azure PowerShell.

**Things to know**

* The core tools don’t validate or test your functions code during publishing.
  + Make sure to use func start to do some testing before you publish.
* When you publish, any functions already present in the target app are stopped and deleted before the contents of your project are deployed.

1. Create and test a simple Azure Function locally with Visual studio

Azure provides three versions of the runtime environment required to run Azure Functions:

* Version 1 (V1) uses the .NET Framework 4.7
* Version 2 (v2x) runs using .NET Core 2
* Version 3 (v3x) contains JavaScript and .NET changes.

V2 triggers enable you to develop and host a trigger in different environments. V2 triggers can only be created using Windows, so use v2 triggers whenever possible.

An Azure Function App stores management information, code, and logs in Azure storage. Create a Storage Account to hold this data. Storage account must support Azure Blob, Queue, Files, and Table storage. Use a general Azure Storage account for this purpose. You specify which storage account to use for the function using the dialog previously shown.

Azure function – you might want to limit the ability to run this function to selected groups of users. You protect an Azure Function by specifying the access rights required to trigger the function. Azure Function triggered by an HTTP request supports three levels of access rights:

* Anonymous. No authentication is required, and any user can trigger the function.
* Function. The http request must provide a key that enables the Azure function runtime to authorize the request. You create this key separately, and you can maintain it using the Azure portal.
* Admin. This is like Function since the user must specify the key with the HTTP request. The difference is that the key is an admin key. This key can be used to access any function in the function app. You can create this key separately.

If you’re creating function triggered by events other than HTTP requests, you’re required to provide a connection string and other details necessary for the function app to access the resource triggering the event. For example, if you’re writing a function triggered by a Blob Storage event, you must specify the connection string for the corresponding Blob Storage account.

**Structure of an Azure Function**

Azure function is implemented as a static class. The class provides a static, asynchronous method named Run, which acts as the entry point for the class.

Parameters passed to the Run method provide the context for the trigger. In the case of an HTTP trigger, function receives an HttpRequest object. This object contains the header and body of the request. You can access the data in the request using the same techniques available in any HTTP app. The attributes applied to this function specify the authorization requirements, and the HTTP operations to which the Azure function responds.

In all cases, an Azure Function is passed an ILogger parameter. The function can use this parameter to write log messages, which the function app will write to storage for later analysis.

An Azure function also contains metadata that specify the type of the trigger, security requirements, and any other specific information requirements. You can modify metadata using the HttpTrigger, BlobTrigger, or other trigger attributes, as shown in the examples. The FunctionName attribute that precedes a function is an identifier for the function used by the function app. The name doesn’t have to be the same as the name of the function, but it’s good practice to keep then synchronized to avoid confusion.

**Test an Azure Function app locally**

You can use the Visual debugger to build and test function app locally.

If it’s a HTTP trigger, then when debugging you’ll see the localhost endpoint to which the function is currently attached.

**Publish a simple Azure function**

An Azure function runs in the cloud in the context of an Azure Function App. Function App is a container that specifies the operating system and resources, such as the memory, computing power, disk space, for running an azure function. Azure function app also provides the public URL for running your function. Behind the scenes, an Azure Function app is a collection of one or more virtual machines (VMs) running a web server. When you publish an Azure function, you deploy it to these VMs.

Azure Visual studio Publish wizard requires that you either have access to the Azure Functions App that will host your functions, or you have an Azure subscription that you can use to create an Azure Functions App as part of the publication process.

**Continuous deployment**

Azure function makes it easy to deploy your function app using App Service continuous integration. Azure functions integrate with numerous deployment sources. The following are supported:

* Bitbucket
* Dropbox
* External repository (Git or Mercurial)
* Git local repository
* GitHub
* OneDrive
* Azure DevOps

You can configure continuous deployment from Azure portal, using the Deployment center feature.

**Zip deployment**

Azure function can be deployed from a zip file using push deployment, either from Azure CLi, or by using REST interface.

**Unit testing**

Create xUnit project.

Add Microsoft.AspNetCore.Mvc. The test project will create a mock HTTP environment. The classes required for doing this are in this package.

1. **Monitor GitHub events by using a webhook with Azure functions**

Webhooks offer a lightweight mechanism for your app to be notified by another service when something of interest happens via HTTP endpoint. You can use a webhook to trigger an Azure function, then analyze the message, to determine what happened and how to respond.

**What is a webhook?**

Webhooks are user-defined HTTP callbacks. They are triggered by some event, such as pushing code to a repo or updating a wiki page. When the event occurs, the source site makes an Http request to the UR: configure for the webhook. With Azure Functions, we can define login in a function that can be run when a webhook message is received.

One common use of webhooks in DevOps environment is to notify an Azure function that the code or configuration for an application has changed in GitHub. The message sent through the webhook contains the details of the event. You can use a webhook with a function to perform a task such as deploying the updated version of the application.

**Set up a webhook for a GitHub repository**

In GitHub, webhooks can be set up for an organization or for a specific repository. Webhook is triggered each time one or more subscribed events occur. For example, the Gollum event allows to listen for wiki creation and updates for a wiki page.

It’s a two-step process. You have to specify how you want your webhook to behave trough GitHub and what events it should listen to. Then you set up your function in Azure Functions to receive and manage the payload received from the webhook.

1. **Enable automatic updates in a web application using Azure Functions and SignalR service**

Reports stock information by fetching changes from the server based on a timer. This design is often called a polling-based design.

CORS is a HTTP feature that enables a web application under one domain to access resources in another domain. Web browsers implement a security restriction known as same-origin policy that prevents a web page from calling APIs in a different domain; CORS provides a secure way to allow one domain (origin domain) to call APIs in another domain.

You can set CORS rules individually for each of the Azure Storage services, by calling Set Blob service properties, Set File service properties, Set Queue service properties, Set Table service properties. Once you set the CORS rules for the service, then a properly authorized request made against the service from a different domain will be evaluated to determine whether it is allowed according to the rules you have specified.

**SignalR and persistent connections**

In contrast to polling, a more favorable design features persistent connection between the client and server. Establishing a persistent connection allows the server to push data to the client at will. The on-demand nature of the connection reduces network traffic and load on the server.

SignalR is an abstraction for a series of technologies that allows your app to enjoy two-way communication between the client and server.

**This one sucks.**

**Expose multiple Azure function apps as a consistent API by using Azure API management**

The Azure API management service enables you to construct an API from a set of disparate microservices.

In your online store, you’ve implemented each part of the application as a microservice – one for the product details, one for order details, and so on. A separate team manages each microservice, and each team uses continuous development and delivery to update and deploy their code regularly. You want to find a way to assemble these microservices into a single product and then manage that product centrally.

**Serverless architecture and microservices**

Microservices have become a popular approach to the architecture of distributed applications in recent years. When you build an application as a collection of microservices, you create many different small services. Each service has a defined domain of responsibility, and is developed, deployed, scaled independently. The modular architecture results in an application that is easier to understand, improve, and test. It also makes continuous delivery easier, because you change only a small part of the whole application when you deploy a microservice.

Another complementary trend in distributed software development is serverless architecture. In this approach, a host organization publishes a set of services that developers can use to run their code. Developers don’t have to concern themselves with a supporting hardware, operating systems, underlying software, and other infrastructure. Instead, the code is run in stateless computing resources that are triggered by requests.

**Azure API management**

Azure API management is a fully managed cloud service that you can use to publish, secure, transform, maintain, and monitor API. It helps organization publish APIs to external, partner, and internal developers to unlock the potential of their data and services. API management handles all the tasks involved in mediating API calls, including authorization and authentication, rate limit and quota enforcement, request and response transformation, logging and tracing, and API version management.

Because you can publish Azure function trough API management, you can use them to implement a microservices architecture. Each function implements a microservice. By adding several functions to a single API management product, you can build those microservices into an integrated distributed application. Once the application is built, you can use API management policies to implement cashing or ensure security requirements.

Microservice architectures can also present challenges, such as:

* Client apps are coupled to microservices. If you want to change the location or definition of the microservice, you may have to reconfigure or update the client app.
* Each microservice may be presented under different domain names or IP addresses. This presentation can give an impression of inconsistency.
* It can be difficult to enforce consistent API rules and standards across all microservices. For example, one team may prefer to respond with xml and another may prefer JSON.
* You’re reliant on individual teams to implement security correctly.

**How does API management help?**

* Client apps are coupled to the API expressing business logic, not the underlying technical implementation with individual microservices.
* API management acts as an intermediary. It forwards request to the right microservice, wherever it’s located and returns responses to users. Users never see the different URIs where microservices are hosted.
* You can use API management policies to enforce consistent rules on all microservices in the product. For example, you can transform all XML responses into JSON, if that’s your preferred format.
* Policies also enable you to enforce consistent security requirements,