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.

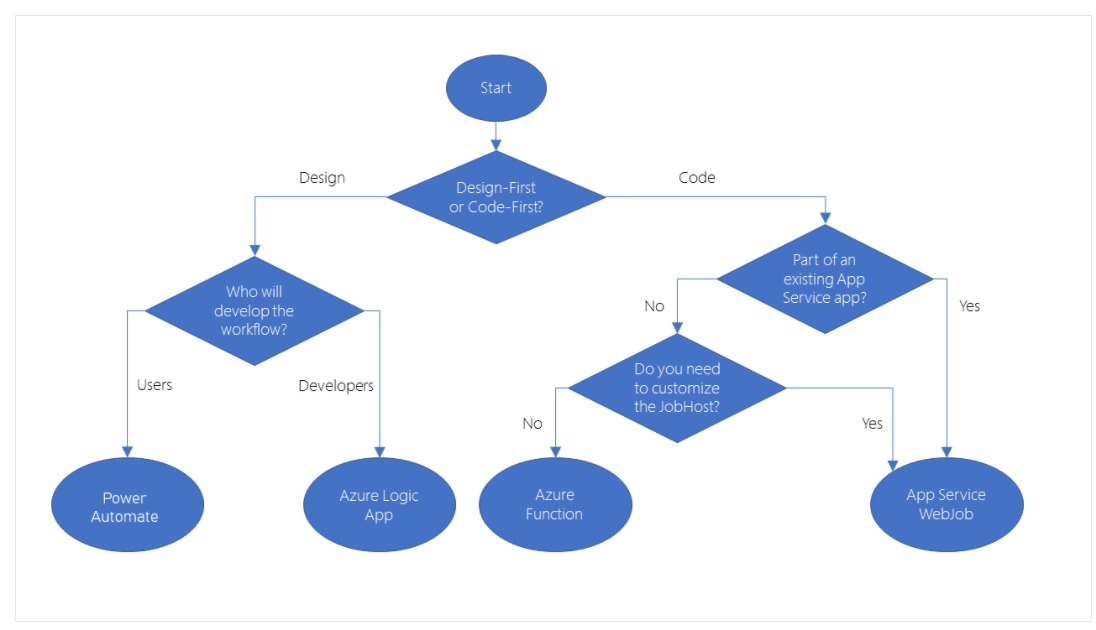
Azure functions can integrate with many different services. These services can trigger your function, or send data input to your function, or receive data output from your function.

**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,

1. **Build serverless apps with Go**

One of the biggest reasons to use Azure Functions is the low cost.

Azure Application Insights has a separate billing model. It might generate charges if it’s enabled in your azure Functions Application.

Using Azure Functions doesn’t mean that you don’t need to care about infrastructure. You just have fewer decisions to make about where your application runs. Server resources are provided on demand. As demand varies, the application automatically scales up and down.

Custom handlers allow you to bring almost any language to Azure functions.

**What are custom handlers?**

At its core, a custom handler is a web server. The web server receives events from the Functions host. You then have an opportunity to write code in your preferred language to respond to the events.

With custom handlers, you can use any language that supports HTTP primitives. That’s nearly any language.

**Important concepts and flow**

Azure functions have three central concepts that are important to understand:

* **Triggers**. A trigger is an event that begins running a function. Examples of common triggers include HTTP requests, new queue messages, and changes to a database. Selecting the right trigger is important to deciding how your function runs.
* **Bindings**. Bindings are helper code that connects your function to another cloud service. Both input and output bindings are available to pass data in and out of your function.
* **Functions host**. Controls the application event flow. As the host captures events, it invokes the handler and is responsible for returning a function’s response.

**Web app**

To use a custom handler, you need to author a web application. After you’ve written your application and compiled it, you need to configure the Azure functions host so it knows how to use it.

For you app to work with HTTP privatives, you need to configure a few things:

* Listen to a custom handler port
* Configure the default executable path
* Enable request forwarding
* Copy of the handler’s response

**Hard shit.**

1. **Choose a messaging model in Azure to loosely connect your services**

Azure provides several technologies that you can use to communicate more reliably, including Storage queues, Event Hubs, Event Grid, and Service Bus. This module shows you how to choose the best technology for your communication task.

Reliable messaging is often a critical problem.

First, to choose a messaging tool, you must understand about a communication whether it sends **messages** or **events**. This knowledge will help you choose the appropriate Azure service to use.

**What is a message?**

In the terminology of distributed applications, messages have the following characteristics:

* Message contains raw data, produced by one component, that will be consumed by another component
* Message contains data itself, not just a reference to that data
* The sending component expects the destination component to process the message content in a certain way. The integrity of the overall system may depend on both the sender and receiver doing a specific job.

**What is an event?**

Events are lighter weight than messages, most often used for broadcast communications. The components sending the event are known as publishers, and receivers are known as subscribers.

With events, receiving components will generally decide in which communications they are interested, and will subscribe to those events. The subscription is managed by an intermediary, like Azure Event Grid or Azure Event Hubs. When publishers send an event, the intermediary will route that event to interested subscribers. This pattern is known as a “publish-subscribe architecture”. It’s not the only way to deal with events, but it is the most common.

Event characteristics:

* Lightweight notification that indicates that something happened
* May be sent to multiple receivers, or to none
* Often intended to fan out, or have many subscribers for each publisher
* Publisher of the event has no expectation about the action a receiving component takes
* Events are discrete units and unrelated to other events
* Some events are part of related and ordered series.

**How to choose messages or events**

For each communication, consider the following question: Does the sending component expect the communication to be processed in a particular way by the destination component?

If the answer is yes, choose a message. If the answer is no, you may be able to use events.

**Choose a message-based delivery with queues**

You want to deliver the details about something directly to the app whenever someone adds music to their collection.

Azure offers two solutions for the problem:

* Azure queue storage
* Azure service bus

**What is azure queue storage**

Queue storage is a service that uses Azure Storage to store large numbers of messages that can be securely accessed from anywhere in the world using a simple Rest-based interface. Queues contain millions of messages, limited only by the capacity of the storage account that owns it.

**What is Azure Service Bus Queue**

Service bus is a message broker system intended for enterprise applications. These apps often utilize multiple communication protocols, have different data contracts, higher security requirements, and can include both cloud and on-premises services. Service bus is built on top of a dedicated messaging infrastructure designed for exactly these scenarios

Both these services are based on the idea of a queue, which holds sent messages until the target is ready to receive them.

**What are Azure service bus topics**

Azure Service Bus topics are like queues but can have multiple subscribers. When a message is sent to a topic instead of a queue, multiple components can be triggered to do their work.

Internally, topics use queues. When you post to a topic, the message is copied and dropped into the queue for each subscription. The queue means that the message copy will stay around to be processed by each subscription branch even if the component processing that subscription is too busy to keep up.

Benefits of queues:

* Increased reliability. At times of high demand, messages can wait until a destination component is ready to process them.

**Message delivery guarantees**

These guarantees can take different approaches.

* At least once delivery. In this approach, each message is guaranteed delivery to at least one of the components that retrieve messages from the queue. Note, however, that in certain circumstances, it is possible that the same message may be delivered more than once. If there are two instances of a web app retrieving messages from a queue, ordinary each message goes to only one of those instances. However, if one instance takes a long time to process the message, and a time-out expires, the message may be sent to the other instance as well. Your web app code should be designed with this possibility in mind.
* At most once delivery. In this approach, each message is not guaranteed for delivery, and there is a small chance that it may not arrive. However, there is no chance that the message will be delivered twice. This is sometimes referred to as automatic duplicate detection.
* First in first out. If your distributed application requires that messages are processed in precisely the correct order, you must choose a queue system that includes a FIFO guarantee.

Having understood that the communication strategy for the architecture should be a message, you must choose whether the use of azure storage queues or azure service buss.

Use service bus topics if you:

* Need multiple receivers to handle each message

Use service bus queues if you:

* Need an at-most-once delivery guarantee
* Need first in first out guarantee
* Need to group messages into transactions
* Want to receive messages without polling the queue
* Need to provide a role-based access model to queues
* Need to handle messages larger than 64kb
* Queue size will not grow larger than 1tb
* Want to publish and consume batches of messages.

Use Queue storage if you

* Need an audit trail of all messages that pass through the queue
* Expect the queue to exceed 1tb in size
* Want to track progress for processing a message inside of the queue.

**Choose Azure Event Grid**

Many applications use publish-subscribe model to notify distributed components that something happened, or that some object changed.

**What is Azure Event Grid?**

Azure event grid is fully managed event routing service running on top of Azure service fabric. Event grid distributes events from different sources, such as Azure blob storage accounts, or azure media services, to different handlers, such as Azure functions or webhooks. Event Grid was created to make it easier to build event-based and serverless applications on Azure.

Event Grid supports most Azure services as a publisher or subscriber and can be used with third-party services. It provides a dynamically scalable, low-cost, messaging system that allows publishers to notify subscribers about a status change. The following illustration shows Azure Event Grid receiving messages from multiple sources and distributing them to event handlers based on subscription.

There are several concepts in Azure Event Grid that connect a source to a subscriber:

* Events. What happened
* Event sources. Where the event took place
* Topics. The endpoint where publishers send events
* Event subscriptions. The endpoint or built-in mechanism to route events, sometimes to multiple handlers. Subscriptions are also used by handlers to filter incoming events intelligently.
* Event handlers. The app or service reacting to the event.

Event sources send events to the event grid and event grid forward relevant events to the subscribers.

Event grid use topics to decide which events to send to which handlers. Event sources tag each event with one or more topics, and event handlers subscribe to the topics they are interested in.

Application, table, Word

Description automatically generated

**What is an event?**

Events are data messages passing through event grid that describe what has taken place. Each event is self-contained, can be up to 64kb, contains several pieces of information based on a schema defined by event grid.

**What is an event source?**

Event sources are responsible for sending events to Event Grid. Each event source is related to one or more event types.

For example, Azure storage is the event source for blob created events. IoT Hub is the event source for device created events.

Event publisher is the user or organization that decides to send events to Event Grid. Event source is the specific service generating the event for that publisher.

**What is an event topic?**

Event topics categorize events into groups. Topics are represented by a public endpoint and are where the event source sends events to. Larger solutions will create a custom topic for each category of related events, while smaller solutions might send all events to a single topic.

Topics are divided into **system** topics and custom **topics**.

**System topics**

Built-in topics provided by Azure services. You don’t see system topics in your azure subscription because the publisher owns the topics, but you can subscribe to them. To subscribe, you provide information about the resource you want to receive events from. If you have access to the resource, you can subscribe to its events.

**Custom topics**

Application and third-party topics.

**What is an event subscription?**

Event subscriptions define which events on a topic event handler wants to receive. A subscription can also filter events by their type or subject, so you can ensure an event handler only receives relevant events.

**What is an event handler?**

Sometimes referred to as an event subscriber is any component that can receive events from Event Grid. For example, Azure functions can execute code in response to the new song being added to the Blob storage account.

**Use Event Grid when you need these features**

* Simplicity. It is straightforward to connect sources to subscribers in Event Grid.
* Advanced filtering. Subscriptions have close control over the events they receive from a topic.
* Fan out. You can subscribe to unlimited number of endpoints to the same events and topics.
* Reliability. Event Grid retries event delivery for up to 24 hours for each subscription.
* Pay-per-event. Pay only for the number of events that you transmit.

**Choose Azure Event hubs**

There are certain applications that produce a massive number of events from almost as many sources.

**What is Azure event hubs?**

**Partitions**

Event hubs receives communications, it divides them into partitions. Partitions are buffers into which the communications are saved.

Choose event hubs if:

* You need support authenticating many publishers
* You need to save a stream of events to data lake or blob storage
* You need aggregation or analytics on your event stream
* You need reliable messaging or resiliency.

**Implement message-based communication workflows with Azure service bus**

In a sense, the sender and receiver of a message are often coupled by strict data contract.

**How to choose a communications technology**

Consider following questions:

* Is the communication an event? If so, consider using Event grid or Event Hubs.
* Should a single message be delivered to more than one destination? If so, use Service Bus topic, otherwise, use queue.

**Choose Service Bus queues if:**

* You need an at-most-once delivery guarantee
* You need a FIFO guarantee
* You need to group messages into transactions
* You want to receive messages without polling the queue
* You need to provide role-based access to the queues
* You need to handle messages larger than 64 KB but smaller than 256 KB for the standard tier or 100 MB for the premium tier
* Your queue size will not grow larger than 80 GB
* You would like to be able to publish and consume batches of messages

**Choose queue storage if:**

* You need a simple queue with no particular additional requirements
* You need an audit trail of all messages that pass through the queue
* You expect the queue to exceed 80 GB in size
* You want to track progress for processing a message inside of the queue

Source components and destination components both need two pieces of information to connect to a queue in a Service Bus namespace:

* The location of the service bus namespace, also know as an endpoint. The location is specified as a fully qualified domain name within the servicebus.windows.net domain.
* An access key. Service bus restricts access to queues or topics by requiring a valid access key.

Talking about a topics, you can set filters on subscriptions

Filters can be one of three types:

* Boolean filters. The TrueFilter ensures that all messages are sent to the topic are delivered to the current subscription. The FalseFilter ensures that none of the messages are delivered to the current subscription (this effectively blocks or switches off the subscription)
* SQL filters. Specifies a condition by using the same syntax as a where clause in a SQL query. Only messages that return True when evaluated against this filter will be delivered to the subscribers.
* Correlation Filters. A correlation filter holds a set of conditions that are matched against the properties of each message. If the property in the filter and the property on the message have the same value, it is considered a match.

1. **Communicate between applications with Azure Queue storage**

Direct communication between the components of a distributed application can be problematic because it might be disrupted when network bandwidth is low or when demand is high.

High traffic causes problems and so the plan is to use a queue to eliminate the direct link between the front-end apps and your middle-tier web service.

**Why use queues?**

A queue increases resiliency by temporarily storing waiting messages. At times of low or normal demand, the size of the queue remains small because the destination component removes messages from the queue faster than they are added.

At times of high demand, the queue may increase in size, but messages are not lost. The destination component can catch up and empty the queue as demand returns to normal.

Autoscale feature is available on Azure virtual machine scale sets and many other features. Autoscaling responds to demand quickly, but not instantaneously. By contrast, azure queue storage instantaneously handles high demand and storing messages until processing resources are available.

**Identify a queue**

When you have a storage account, lets look how to access the queue.

To access the queue, you need three pieces of information:

* Storage account name
* Queue name
* Authorization token

This information is used by both the senders and receivers of queue.

Queue name must be unique within your storage account but doesn’t need to be globally unique.

**Access authorization**

Every request to the queue must be authorized, there are several options:

* Azure active directory. You can use role-based authentication and identify specific clients based on AAD credentials.
* Shared key. Sometimes referred to as an account key, this is an encrypted key signature associated with the storage account. Every storage account has two of these keys that can be passed with each request to authenticate access. Using this approach is like using a root password, it provides full access to the storage account.
* Shared access signature. Or SAS, is a generated URI that grants limited access to objects in your storage account to clients. You can restrict access to specific resources, permissions, and scope to a date range to automatically turn off access after a period.

In queues, get and delete are separate values. The arrangement handles potential failures in the receiver and implements a concept called at least once delivery. After the receiver gets a message, the message remains in the queue but is invisible for 30 seconds. If the receiver crashes or experiences a power failure during processing, then it will never delete the message from the queue. After 30 seconds, the message will reappear in the queue and another instance of the receiver can process it to completion.

**Enable reliable messaging for Big Data applications using Azure Event Hubs.**

Connect sending and receiving applications with Event hubs so you can handle extremely high loads without losing data.

Event Hubs is a big data streaming platform and event ingestion service. It can receive and process millions of events per second. Data sent to an event hub can be transformed and stored by using any real-time analytics provider or batching/storage adapters. It can also be configured to scale dynamically, when required, to handle increased throughput.

Event hubs is cloud-based, event-processing service that can receive and process millions of events per second.

An entity that sends data to your event hub is called a publisher, and an entity that reads data from an event hub is called a consumer, or a subscriber. Event hub sits in between. This decoupling helps to manage scenarios where the rate of event production nis much higher than the consumption.

**Events**

An event is a small packet of information (a datagram) that contains a notification. Events can be published individually or in batches, but a single publication can’t exceed 1 mb.

Event publishers are any app or device that can send out events using either HTTPS, Advances message queuing protocol, or Apache Kafka.

Event subscribers are apps that use one of two supported programmatic methods to receive and process events from an event hub.

* EventHubReceiver. A simple method that provides limited management options
* EventProcessorHost. An efficient method that we’ll use later in this module.

**Consumer groups**

An event hub consumer group represents a specific view of an event hub data stream. By using separate consumer groups, multiple subscriber apps can process an event stream can process an event stream independently, without affecting other apps. However, the use of many consumer groups isn’t a requirement, and for many apps, the single default consumer group is sufficient.

**Create and configure an event hub**

There are two main steps to create a new event hub. First step is to define Event Hubs namespace. The second step is to create an event hub in that namespace.

**Minimum Event Hub requirements for configuring an application?**

To configure an application to send messages to an Event Hub, provide the following information, so that the application can create connection credentials:

* Event hub namespace name
* Event hub name
* Shared access policy name
* Primary shared access key

To configure an application to receive messages from an Event Hub, provide the following information, so that the application can create connection credentials:

* Event hub namespace name
* Event hub name
* Shared access policy name
* Primary shared access key
* Storage account name
* Storage account connection string
* Storage account container name

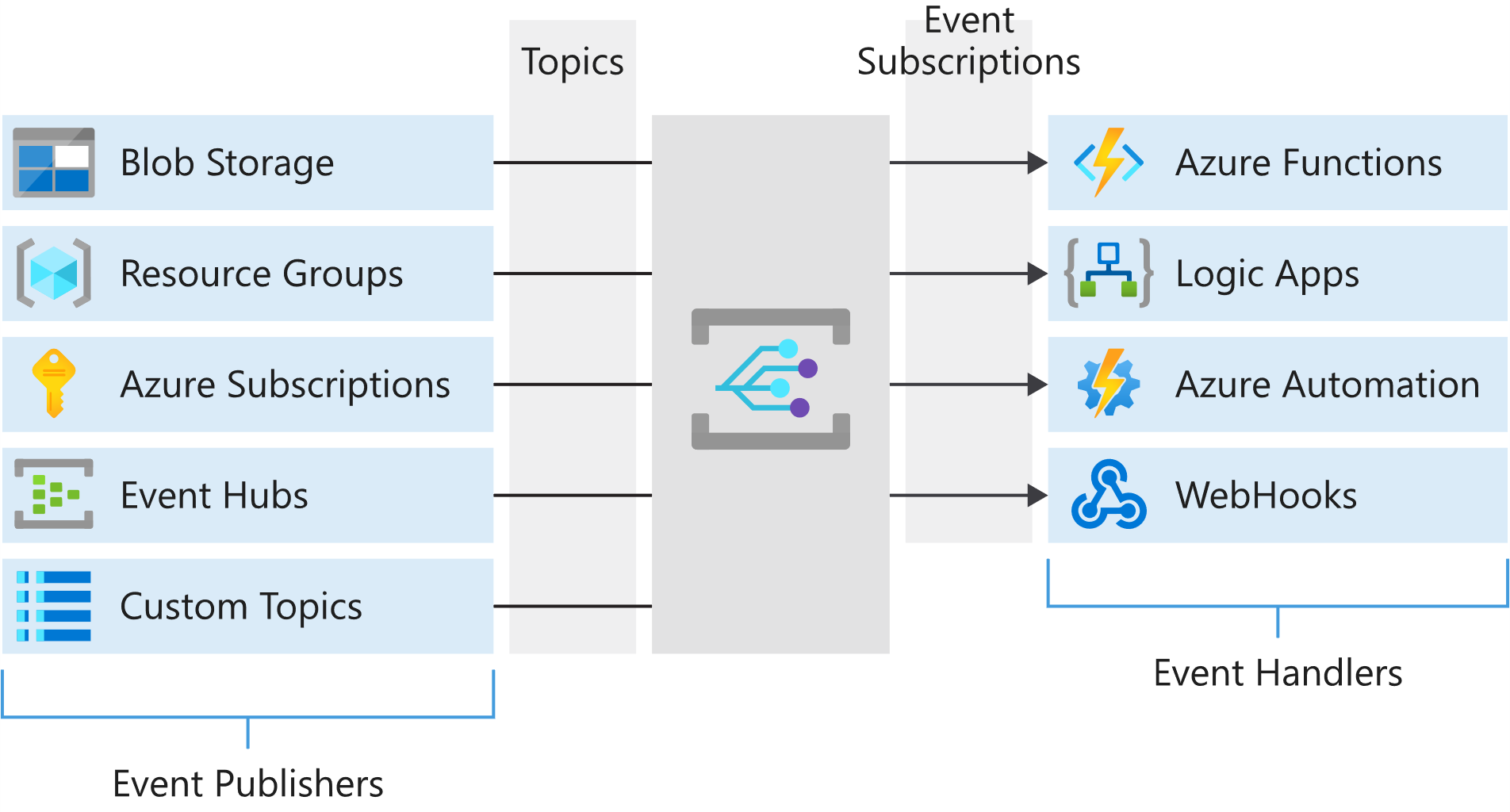
If you have a receiver application that stores messages in Azure Blob storage, you’ll also need to configure a storage account.

**React to state changes in your Azure services by using event grid**

In complex cloud environment, you might need to respond to events from many different sources both automatically and manually.

You might want to capture changes in virtual machines. You’ll use azure event grid to receive events that will happen to virtual machines and Azure logic apps to send emails that alert your team of any changes.

**What is an event grid?**

Event grid aggregates all your events and provides routing from any source to any destination. Event grid is a service that manages the routing and delivery of events from many sources and subscribers. This process eliminates the need for polling, and results in minimized cost and latency. Event publishers and subscribers are decoupled by using the publisher/subscriber pattern. 

**Capabilities**

* Its simple. Point and click in Azure portal to add and collet your events from Azure resources
* It can filter events. Filter events so that handlers receive only relevant events.
* It supports multiple subscribers. Attach multiple handlers to a single event from a single source.
* It’s reliable. Take advantage of 24-hour retries to ensure events are delivered.
* Its throughput is high. Handle a high volume of events, in the range of millions per second.
* It has built-in events. Use built-in events to get started quickly and easily.
* It supports custom events.

**Event sources and event handlers**

Azure storage is the event source for blob-created events.

Sources can be configured from anywhere.

Some examples of event handlers within Azure are:

* Azure functions
* Azure logic apps
* Azure automation
* Azure event hubs
* Azure service bus

1. **Choose data storage approach in Azure**

Data comes in different shapes or sizes, and no single storage solution fits all data. Key factors to consider are: how to classify your data, how your data will be used, and how you can get the best performance for your application.

**Structured data**

Sometimes referred to as relational data, is data that adheres to a strict schema, so all the data has the same fields or properties. The shared schema allows this type of data to be easily searched with query languages such as SQL.

Structured data is often stored in database tables with rows and columns.

**Semi-structured data**

Less organized than structured data, band is not stored in a relational format, as the fields do not neatly fit into tables, rows, and columns. Contains tags that make the organization and hierarchy of the data apparent – for example, key/value pairs. Its non-relational, or NoSQL data. The expression and structure of the data in this style is defined by a serialization language.

**Common formats**

* Xml
* JSON
* Yaml

**Unstructured data**

Often delivered in files, such as photos or videos.

* Media files, such as photos, videos and audio files
* Office files, such as word documents
* Text files
* Log files

**Create an Azure Storage account**

**What is azure storage?**

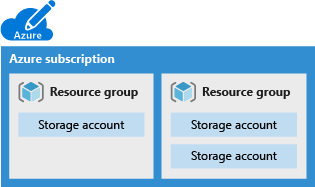
Azure provides many ways to store your data, including multiple database options like Azure SQL Database, Azure cosmos Db, Azure Table storage. Azure offers multiple ways to store and send messages, such as Azure Queues and Event Hubs. You can even store loose files using services like Azure Files or Azure blobs.

Azure groups four of these data services together under the name Azure Storage. The four services are Azure Blobs, Azure Files, Azure Queues, and Azure Tables.

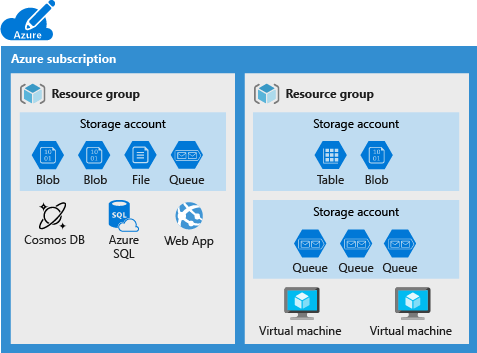
**What is a storage account?**

Is a container that groups a set of Azure storage services together. Only data services from Azure storage can be included in a storage account. One storage account can have multiple same type storage services (like, two queue services, 3 blob services, etc.)

Combining data services into a single storage account enables to manage them as a group. The settings you specify when you create the account, or any changes that you make after creation, apply to all services in the storage account.

A storage account is an azure resource and is part of a resource group.

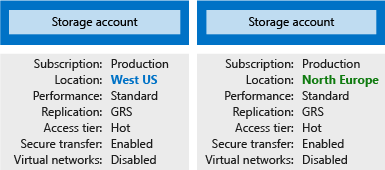
Other Azure data services, such as Azure SQL and Azure Cosmos DB, are managed as independent Azure resources and cannot be included in a storage account.



**Storage account settings**

* Subscription. Subscription that will be billed for the services in the account.
* Location. The datacenter that will store the services in the account.
* Performance. Determines the data services you can have and type of hardware discs used to store the data.
* Replication. Determines the strategy used to make copies of your data to protect against hardware failure.
* Access tier. Controls how quickly you will be able to access the blobs in ta storage account. Hot gives quicker access than cool, but at increased cost.
* Secure transfer required. Enable requires HTTPS, while disabled allows HTTP
* Virtual networks. A security feature allows inbound access request only from the virtual networks you specify.

**How many storage accounts do you need?**

You need one storage account for each group of settings that you want to apply to your data.

The number of storage accounts you need is typically determined by your data diversity, cost sensitivity, and tolerance for management overhead.

**Data diversity**

Do you have data that is specific to a country or region? If so, you might want to store the data in a datacenter in that region or country for performance and compliance reasons. You will need one storage account for each geographical region.

Separating proprietary data and public data will also require separate storage accounts.

**Cost sensitivity**

Storage account by itself has no financial cost. However, the settings you choose for the account do influence the cost of services in the account. Geo-redundant storage costs more than locally redundant storage.

You can use multiple storage accounts to reduce costs. For example, you could partition your data int critical and non-critical categories. You could place critical data into geo-redundant storage and put non-critical data in a different storage account with locally redundant storage.

**Account settings**

The storage account settings we’ve discussed apply to data services in the account. Now we’ll discuss settings that apply to the account itself, rather than to the data stored in the account:

* Name
* Deployment model
* Account kind.

**Name**

Globally unique, only lowercase letters and digits and be between 3 and 24 chars.

**Deployment model**

* Resource manager. The current model that uses the Azure Resource Manager API
* Classic. Legacy offering that uses the Azure service management API

Key difference is their support for grouping. Resource manager adds the concept of a resource group.

Microsoft recommends that you use Resource manager for all new resources.

**Account kind**

* StorageV2 (general purpose v2). Current offering that supports all storage types and lates features
* Storage (general purpose v1). Legacy kind that supports all storage types but may not support all features
* Blob storage. Legacy kind allows only block blobs and append blobs.

**Choose a tool**

Storage account creation is usually a one-time operation, then the portal is the most common choice.

In the cases where you need automation, then you may use scripting and azure client or azure power shell.

**Connect an app to azure storage**

Microsoft storage is a managed service that provides durable, secure, and scalable storage in the cloud. Let’s break these terms down.

* Managed. Handles maintenance and any critical problems for you.
* Durable. Ensures that your data is safe in the event of transient hardware failures.
* Secure. All data written to azure storage is encrypted by the service.
* Scalable. Designed to be massively scalable to meet the data storage and performance needs.

**Data types**

* Blobs. Massively scalable object store for text and binary data.
* Files. Managed file shares for cloud or on-premises deployments.
* Queues. Messaging store for reliable messaging between application components.
* Table storage. A NoSQL store for schema-less storage of structured data.

**Blobs**

Blob storage is ideal for:

* Serving images or documents directly to a browser
* Storing files for distributed access
* Streaming video and audio
* Storing data for backup and restoration, disaster recovery, and archiving.
* Storing data for analysis by an on-premises or azure-hosted service.

Azure supports three kinds of blobs:

* Block blobs. Used to hold text or binary files.
* Page blobs. Used to hold random-access files. Are used primarily as the backing storage for the VHDs.
* Append blobs. Made up of blocks but optimized for append operations. For example, you might write all of your trace logging to the same append blob for an application running on multiple VMs.

**Files**

Used for many common scenarios:

* Storing shared configuration files for VMs
* Log files such as diagnostics
* Shared data between on-premises applications and azure VMS.

**Security access keys**

Each storage account has two unique access keys that are used to secure the storage account. If your app needs to connect multiple storage accounts, your app will require an access key for each storage account.

1. **Secure your azure storage account**

Azure storage accounts can create authorized apps in active directory to control access to the data in blobs and queues. This authentication approach is the best solution for apps that use blob storage or queue storage.

For other storage models, clients can use a shared key, or shared secret. The client embeds the shared key in the http authorization header of every request, and the storage account validates the key.

**Storage account keys**

In Azure storage accounts, shared keys are called storage account keys. Azure creates two of these keys. Keys give access to everything in the account.

Because these keys are powerful, use them only with trusted in-house applications that you control completely.

As a best practice, you shouldn’t share storage account keys with external third-party applications. If these apps need access to your data, you’ll need to secure their connections without using storage account keys.

**Shared access signature**

For untrusted clients, use a shared access signature (SAS). It’s a string that contains a security token that can be attached to a URI. With this key, you may specify constraints, such as the permissions and the time range of access.

You can give a customer a SAS token, for example, so they can upload pictures to a file system in blob storage. You can give a web app permission to read those pictures. In both cases, you allow only the access that the application needs to do the task.

**Types of shared access signatures**

* Service level SAS to allow specific access to specific resources in a storage account. For example, to allow app to retrieve a list of files in a file system
* Account level SAS. Allows to access anything that a service-level can allow, plus additional resources and abilities.

You’d typically use a SAS for a service where users read and write their data to your storage account.

**Storage application data with Azure blob storage**

Blobs give you object storage in the cloud and an API that lets you build apps to access the data.

**What are blobs?**

Blobs are files for the cloud. Apps work with blobs in much the same way as they would work with files on a disk. Apps work with blobs in much the same way as they would work with files on a disk, like reading and writing data. However, unlike a local file, you can reach blobs from anywhere with an internet connection.

Azure blob storage is unstructured, meaning that there are no restrictions on the kinds of data it can hold. For example, a blob can hold a pdf document, a JPG image, a JSON file, video content, and more. Blobs aren’t limited to common file formats. A blob could contain gigabytes of binary data streamed from a scientific instrument, an encrypted message for another application, or data in a custom format for an app you’re developing.

Blobs are usually not appropriate for structured data that needs to be queried frequently.

Blobs are frequently used in combination with databases to store non-quarriable data. For example, an app with database of user profiles could store profile pictures in blobs. Each user record in the database would include the name or URL of the blob containing the user’s picture.

Blobs are used for data storage in many ways across all kinds of apps and architectures:

* Apps that need to transmit large amounts of data using messaging system that supports only small messages. These apps can store data in blobs and send the blob URLs in messages.
* Blob storage can be used like a file system for storing and sharing documents and other personal data.
* Static web assets like images can be stored in blobs and made available for public download as if they were files on a web server.
* Many Azure components use blobs behind the scenes. For example, Azure Cloud Shell stores your files and configuration in blobs, and virtual machines uses blobs for hard-disk storage.

**Storage accounts, containers, metadata**

In blob storage, every blob lives inside a blob container. You can store an unlimited number of blobs in a container and an unlimited number of containers in a storage account. Containers can only store blobs, not other containers.

Blobs and containers support metadata in the form of name-value string pairs. Your apps can use metadata for anything you like: a human-readable description of a blob content can be displayed by the app, a string that your app uses to determine how to process the blob’s data, and so on.

Take in mind, that blob storage does not provide any mechanism for searching or sorting blobs by metadata.

**Blob storage API and client libraries.**

The blob storage API is rest-based and supported by client libraries in many popular languages.

**Design a storage organization strategy**

When designing an app that needs to store data, it’s important to think about how the app is going to organize data across storage accounts, containers, and blobs.

**Storage accounts**

A single storage account is flexible enough to organize your blobs however you like, but you should use additional storage accounts as necessary to logically separate costs and control access data.

**Containers and blobs**

The nature of your app and the data it stores should drive your strategy for naming and organizing containers and blobs.

Apps using blobs as part of a storage scheme that includes database often don’t need to rely heavily on organization, naming, or metadata to indicate anything about their data. The app will determine where blobs are stored and the kind of data they contain.

Other apps may use Azure blob storage more like a personal file system, where container and blob names are used to indicate meaning and structure. Blob names in these kinds of apps will often look like traditional file names and include file name extensions like .jpg to indicate what kind of data they contain. They’ll use virtual directories to organize blobs and will frequently use metadata tags to store information about blobs and containers.

**Naming limitations**

**Public access containers as security boundaries**

By default, all blobs require authentication to access.

Enabling public access is important for scalability because data downloaded directly from Blob Storage doesn’t generate any traffic in your server-side app. However, anyone who know their storage URLS can download blobs in a container configured for public access without any kind of authentication or auditing. Never put blob data in public container that you don’t intend to share publicly.

**Blob name prefixes (virtual directories)**

Technically, containers are flat and do not support any kind of nesting or hierarchy. But if you give blobs hierarchical names that look like file paths “finance/budgets/2017/q1.xls”, the api’s listing operation can filter results to specific prefixes.

This feature is often called virtual directories, because some tools and client libraries use it to visualize and navigate blob storage as if it was file system. Each folder navigation triggers a separate call to list the blobs in that folder.

Using names that are like file names for blobs is a common technique for organizing and navigating complex blob data.

**Blob types**

* Block blobs. Composed of blocks of different sizes and can be uploaded independently and in parallel. Writing to a block blob involves uploading data to blocks and committing them to the blob.
* Append blobs. Support only appending new data (not updating or deleting existing data), but very efficient at it. Great for scenarios like storing logs or writing streamed data.
* Page blobs. Designed for scenarios that involve random-access reads and writes.

Block blobs are the best choice for most scenarios that don’t specifically call or append or page blobs.

**Containers**

Unlike storage account creation, container creation is a lightweight activity that makes sense to perform from within an app. It’s not uncommon for apps to create and delete containers as part of their work.

**Workflow**

The typical workflow for apps that use Azure Blob storage is as follows:

* Retrieve configuration. At startup, load the storage account configuration, typically a storage account connection string.
* Initialize client. To initialize Azure Storage Client library, use the connection string. This creates the objects the app will use to work with the Blob storage API.
* Use. To operate on containers and blobs, make API calls with the client library.

Storage account connection strings include the account key. Consider account key a secret and store it securely.

**Initialize the blob storage object model**

In Azure Storage SDK for .NET, the standard pattern for using blob storage is as follows.

* Instantiate a new blobServiceClient object and provide the connection string to your storage account.
* To get a BlobContainerClient, call GetBlobContainerClient on the BlobServiceClient with the name of the container.

None of this initialization code makes calls over the network. This means that some exceptions that occur because of incorrect information won’t be thrown until later. For example, if an incorrectly formatted connection string is supplied, exception will be thrown immediately. However, if the connection string points to a storage account that does not exist, no exception will be thrown until you attempt an operation against the storage account.

**Create containers at startup**

To create container when your app starts or when the app first tries to use a container, call CreateIfNotExistsAsync on a BlobContainerClient.

It won’t throw an exception if the container already exists, but it does make a network call to Azure Blob Storage. Call it once during initialization, not every time you try to use a container.

1. **Introduction to Azure virtual machines**

Azure VMs are one of several types of on-demand, scalable computing resources that Azure offers. With VMs, you have total control over the configuration and can install anything you need to perform the work. You don’t need to purchase physical hardware when you need to scale or extend your datacenter. Finally, Azure provides additional services to monitor, secure, and manage updates and patches to the OS.

Azure offers Infrastructure as a service.

Things to think about:

* Start with the network
* Name the VM
* Decide the location for the VM
* Determine the size of the VM
* Understanding the pricing model
* Storage for the VM
* Select an operating system

**Starting with the network**

Virtual networks (VNets) are used in Azure to provide private connectivity between Azure Virtual Machines and other Azure services. VMs and services that are part of the same virtual network can access one another. By default, services outside the virtual network cannot connect to services within the virtual network. However, you can configure the network to allow access for external service.

If Vnet will be connected to other Vnets, you must select IP address ranges that are not overlapping.

**Plan each VM deployment**

* What does the server communicate with?
* Which ports are open?
* Which OS is used?
* How much disk space is in use?
* What kind of data does this use? Are there restrictions with not having it on-premises?
* What sort of CPU, memory and disk I/O load does the server have? Is there burst traffic to account for?

**Name the VM**

Good naming convention includes:

* Environment. Dev, prod, QA.
* Location. UW (US west), UE (US east)
* Instance. 01. 02.
* Product or service. Service
* Role. SQL, web, messaging.

For example, devusc-webvm01.

**What is an azure resource?**

An azure resource is manageable item in azure? Just like physical computer in your datacenter, VMs have several elements that are needed to do their job:

* The VM itself
* Storage account for the disks
* Virtual network (shared with other VMs and services)
* Network interface to communicate on the network
* Network Security Group(s) to secure the network traffic
* Public internet address (optional).

Azure will create all these resources if necessary, or you can supply existing ones as part of the deployment process.

**Other options available to create and manage azure VMs**

* Azure resource manager
* Azure PowerShell
* Azure cli
* Azure Rest api
* Azure client sdk
* Azure vm extensions
* Azure automation services

You can create template, deploy in test environment, if something is not right, redeploy with different settings, and when works, use template for dev and prod and just use different resources (storage, etc.)

**Backup your virtual machines**

Azure backup can be used for wide range of data backup scenarios, such as:

* Files and folders on windows OS machines (physical or virtual, local or cloud)
* Application-aware snapshots
* Popular Microsoft server workloads such as Microsoft SQL server, Microsoft SharePoint, Microsoft Exchange
* Native support for Azure Virtual Machines, both windows or linux
* Linux and Windows 10 client machines

**Resources used in a Linux VM**

**What is SSH?**

Secure shell is an encrypted connection protocol that allows secure sign-ins over unsecured connections.

Two ways to authenticate an SSH connection, username and password, or SSH key pair.

More secure way is public-private key pair, known as SSH key.

There are two parts to an SSH key pair: a public key and private key.

* Public key is placed on your Linux VM or any other service that you wish to use without public key cryptography. Can be shared with anyone.
* Private key is what you present to verify your identity when you make an SSH connection.

You can use same public-private key pair to access multiple Azure VMs and services.

An image is a template that’s used to create a VM.

**Use RDP to connect to Windows Azure virtual machines.**

When you have a new Windows virtual machine, we need to install our custom software onto it. There are several options to choose from:

* Remote desktop protocol (RDP)
* Custom scripts
* Custom VM images (With the software preinstalled)

Simplest approach for windows VMs: Remote desktop.

**Align requirements with cloud types and service models in Azure**

Azure supports three approaches to deploying cloud resources – public, private, and the hybrid cloud.

**Public cloud**

Public clouds ae the most common way of deploying cloud computing. Services are offered over the public internet and available to anyone who wants to purchase them. Cloud resources such as servers and storage are owned and operated by a third-party cloud service provider and delivered over the internet.

**Why public cloud?**

Public clouds can be deployed faster than on-premises infrastructures and with an almost infinitely scalable platform.

Examples of why you would use public cloud:

* Service consumption trough on-demand or subscription model.
* No up-front investment of hardware. No requirement to purchase, manage, and maintain on-premises hardware and application infrastructure. Cloud service provider is held responsible for all management and maintenance of the system.
* Automation. Quickly provision infrastructure resources using a web portal, scripts, or via automation.
* Geographic dispersity. Store data near your users in desired locations without having to maintain your own datacenters.
* Reduced hardware maintenance. Service provider is responsible for hardware maintenance.

**Private cloud**

Consists of computing resources used exclusively by users from one business or organization. It can be physically located at your organization’s on-site datacenter, or it can be hosted by a third-party service provider.

**Why private cloud?**

* Pre-existing environment.
* Legacy applications.
* Data sovereignty and security. Political borders and legal requirements may dictate where data can physically exist.
* Regulatory compliance / certification.

**Hybrid cloud**

A hybrid cloud is a computing environment that combines a public cloud and a private cloud by allowing data and applications to be shared between them.

Example, database could be hosted in an on-premises datacenter and the website could be hosted in the public cloud. A VPN is used between the on-premises datacenter and the public cloud. This scenario would be considered a hybrid cloud.

**Why hybrid cloud?**

* Existing hardware investment.
* Regulatory requirements. When regulation requires that the data needs to remain at a physical location
* Unique operating environment.
* Migration. Move workloads to the cloud over time.