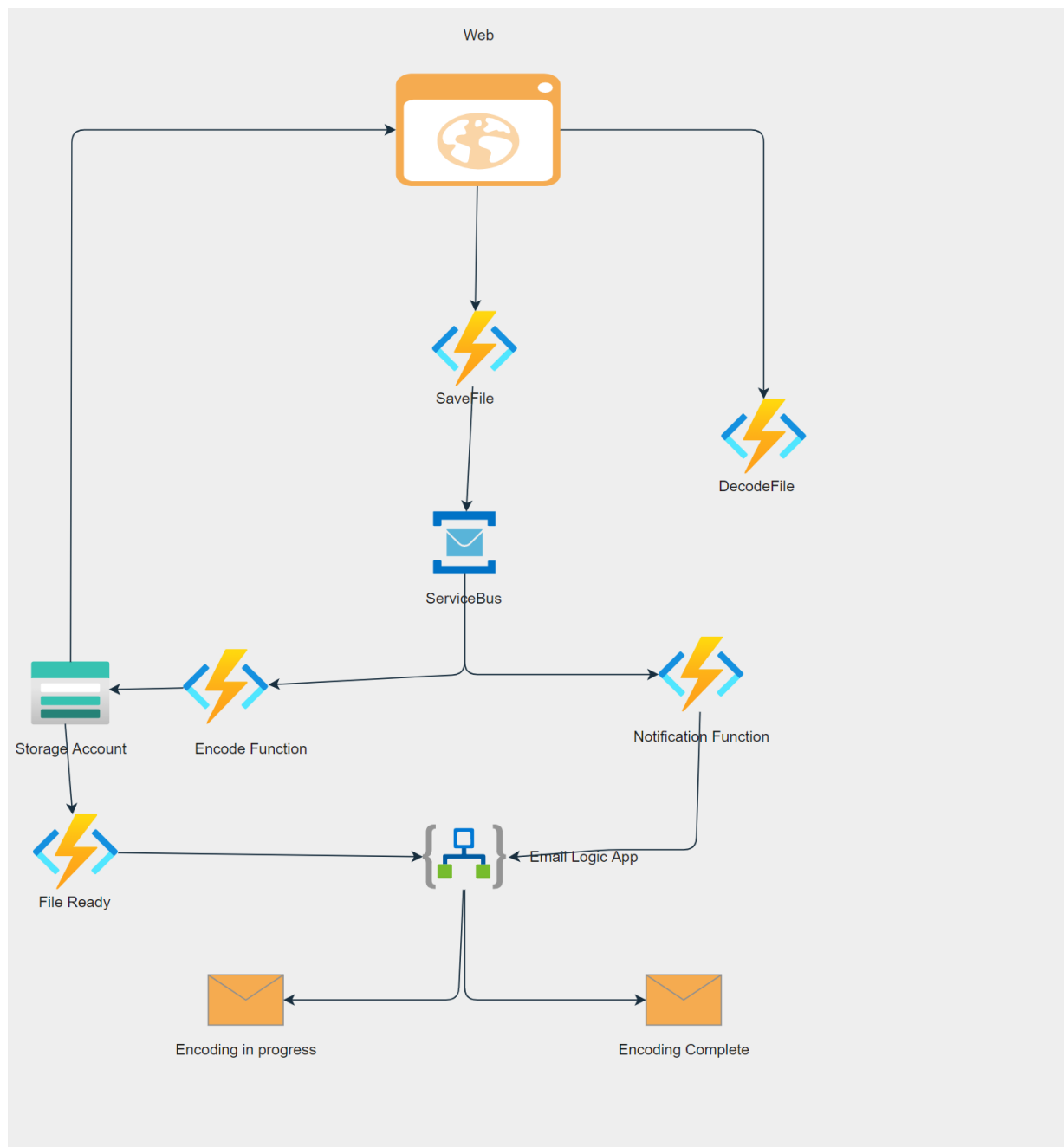


Introduction to Azure Serverless services

Azure Serverless services promote a Software as a Service approach to cloud resources and deployment. They take away the worry about the platform or the runtime environment where your resources reside and run and let you focus on the functionality of the resources themselves.

In order to make an introduction into some of the available services we created a demo application which uses the architecture presented in the diagram below.



The application is a simple “File Encoder” application which will allow the user to enter some text and an email address then the text introduced will be uploaded encoded to a file and the user will receive an email with the URL where to access the file and the encoding key. Then he can return to the application to decode the file using the key.

Services used:

Storage Account:




The first service that we used is the “Storage Account”. It provides us with a platform agnostic solution to storage. It has multiple storage solutions which range from containers (storage for blobs – files; this is what we will be using for this example), tables for SQL like data storage and queues for message transfer.

For our application we will use it in 2 ways. First, we will use it to store the encoded files for each user. Secondly, it will be used to store the files for our web interface.

In order to create it we go through the following steps:

- Go to create resource and select Storage Account
- Select the subscription and the resource group to use
- Give it a name, select the region and standard performance (for our use case it is enough)
- The rest can be left default so go ahead and create it


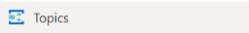


Now we have to configure it to suit our needs:

- Go to the “Containers” blade and  create a new container called “files” with “Blob” access level
- Go to the Static Website blade  and enable it
- Enter for index and error path “index.html” and save the “Primary Endpoint” somewhere
- Go to the CORS blade  and configure any needed CORS settings
- If we now go back to “Containers” we see a new container “\$web” appeared; it will be used to serve the static website files which we will put there later
- Go to access keys and save the “key1” connection string somewhere

Service Bus:

The Service Bus is azure's solution for serverless message passing, providing us with an advanced message passing solution that supports queues, topics (pub/sub model) with filtering rules. Under one service bus namespace there can be multiple queues (single input/output) and topics – can have more subscriptions for one topic so they are one input multiple outputs.

For our example we will configure a service bus with 1 topic with 2 subscriptions. Let's go through the steps to create it.


- Go to create resource and select Service Bus
- Select the Subscription and Resource Group
- Give it a name and select a location
- For the Pricing Tier select Standard (needed to use topics)
- Go ahead and create it
- Now to configure it
- Go to the Service Bus you created
- Click the Create topic  button at the top
- Give it a name and create it
- Now either go to the “Topics”  blade or select topics from the middle section and select your topic
- Now click on the create a subscription  button, give it a name (this will be used for in progress notifications), set the max delivery count to 5 (doesn't matter for this example, this is the retry count) and set it to never auto-delete
- Do the same for another subscription (will be used to start an encoding)
- Now go back to the service bus namespace and go to the “Shared Access Policies” blade  and click on the default key (or you can create a new one) and save the PrimaryConnectionString somewhere
- Also save the name of the topic and subscriptions somewhere

This should be all needed in order to configure the service bus. If you want to, you can also go ahead and download a program called “Service Bus Explorer” where you would use the connection string you saved from the key to connect to the service bus to manage it and browse through it if needed.

Logic App:

Azure's Logic App is a cloud-based platform for creating and running automated workflows that integrate your apps, data, services, and systems. With logic apps you can orchestrate and automate workflows using triggers and actions. While there are a lot of complex things you can do with logic apps (and really nice ones in the IoT space) we are going to use some simple connectors and actions in order to trigger emails to be sent.

Let's configure our logic app by taking the following steps:

- Go to create resource, search and select Logic App
- Select Consumption, the Subscription and the Resource Group
- Give it a name and select the region
- Go ahead and create it
- Go to the logic app you just created
- Go to the designer blade  Logic app designer (if you are not taken there automatically)
- As a starting point select or search for HTTP and select a HTTP trigger (When a HTTP request is received)
- For the JSON Schema populate it with the JSON below and set the method as POST
- Add a new step, search for "Control" and select the condition action
- Inside the condition block a new panel should automatically open where you can choose to use the outputs from the HTTP request.
- Go ahead and select the notification
- For the condition select "is equal to" and for the value input "true" (no quotes)
- This was the configuration to check if we are sending a notification request or a full request with the file and the encoding key
- Now for both of the branches resulting from the condition block we will use the same action: search for "Gmail" (or your email provider) and select SendEmailV2
- Login with the email address (this will be the sender of the email)
- Set as the "to" parameter the recipient field from the http request
- Configure the body as you like

If you want to, you can test your logic app by either calling the URL directly (if you click on the HTTP trigger you can now see the URL) or by using the "Run Trigger" button

JSON schema of request:

```
{
  "properties": {
    "encodingKey": {
      "type": "string"
    },
    "fileUrl": {
      "type": "string"
    },
    "notification": {
      "type": "boolean"
    },
    "recipient": {
      "type": "string"
    }
  },
  "type": "object"
}
```

Azure Functions:

Azure Functions are the star of Azure's serverless compute services. In reality they are mini applications (some even call them micro services but they are not the same) that run inside a function app (one function app can have many functions). They are again as everything else mentioned here platform agnostic (you still have to choose the operating system), you can see their reports but by default you don't know and don't care about on which platform they run on.

Another major improvement that functions bring is the minimalism approach. From the start you don't have all the boilerplate code of a regular c# application or API. All of the tools are there available to use if you need them but they are not baked in.

Azure functions work based on event triggers. Those triggers can be of many types but here we will present 3 of the more popular ones.

First we have the HTTP trigger. It is, as you can guess from the name, triggered by an http request so you can build mini APIs using it. You can specify everything you need like route, method, authorization and path parameters from the start.

Next we have the Service Bus trigger. It is connected to a service bus queue or topic subscription (in our case this is what we are using) and it is triggered whenever there is a new message in the queue/topic subscription. After processing, the message can be completed (to be deleted), abandoned (for it to be retried) or deadLettered (to be send for investigation in a separate queue to be checked for issues).


The last trigger that we are using is the “BlobTrigger”. It is connected to a storage account container and will trigger the function whenever a new “blob” (file) is uploaded to the container and we will receive the information about the blob.

To extend the functionality of the functions we can also use bindings to automatically initialize and minimize the code needed to “chain” the functions together. In our example we have used output bindings for the service bus to use a message sender to publish the message to the topic.

Our application uses 5 functions:

- 2 HTTP trigger functions
- 2 Service Bus trigger functions
- 1 Blob trigger function

Let’s see how to create a function app where to host these functions:



- Go to create resource, search and select function app
- Select the subscription and resource group
- Give it a name
- For our example we will be publishing code
- Select a runtime stack and version (for our example we are using .Net 3.1)
- Go to hosting
- Select the storage account you created prior
- Leave everything else default and create your function
- Now go to the configured function app
- Go to the CORS blade to allow the web to call our function later  CORS
- Add the primary endpoint of the storage account to the allowed origins (and localhost if you want for local development)

Now you can create your functions and add logic to them. You can use the examples provided on GitHub but you will need to add the configuration (connection strings and names that we told you to save/remember prior) in a file `local.settings.json`.

After you are happy with your work you can deploy your function to the function app using your IDE or azure CLI tools.

Web:

As a web-ui you can use any implementation and aspect you would like as long as it can be “published” to a directory. You can also use the sample provided in the repository but don’t forget to change the URL from the `httpService`.

In order to deploy it online you can go to the storage account you created prior to the “\$web” container. Inside of it you can upload  the files  for the user interface and then you can access them using the “Primary Endpoint” for the storage account.

Another option would be to use the azure CLI tools to deploy directly from the command line.