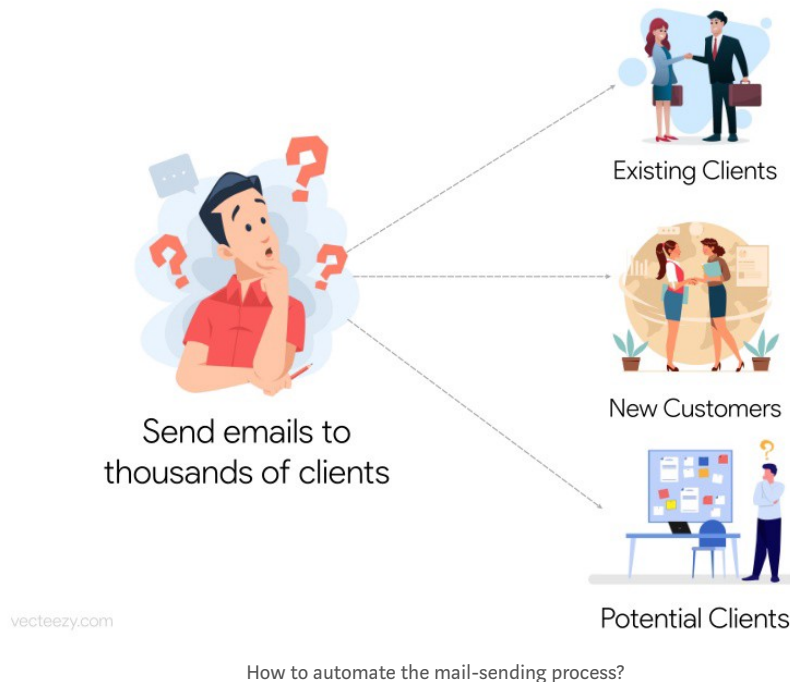


# Problem statement

All businesses, whether they are growing or planning on expanding their customer base, need a marketing strategy. This is achieved by sending promotional emails to potential clients and existing clients from time to time. A requirement on the same lines was put forth by my business.



However, it is tedious due to multiple reasons:

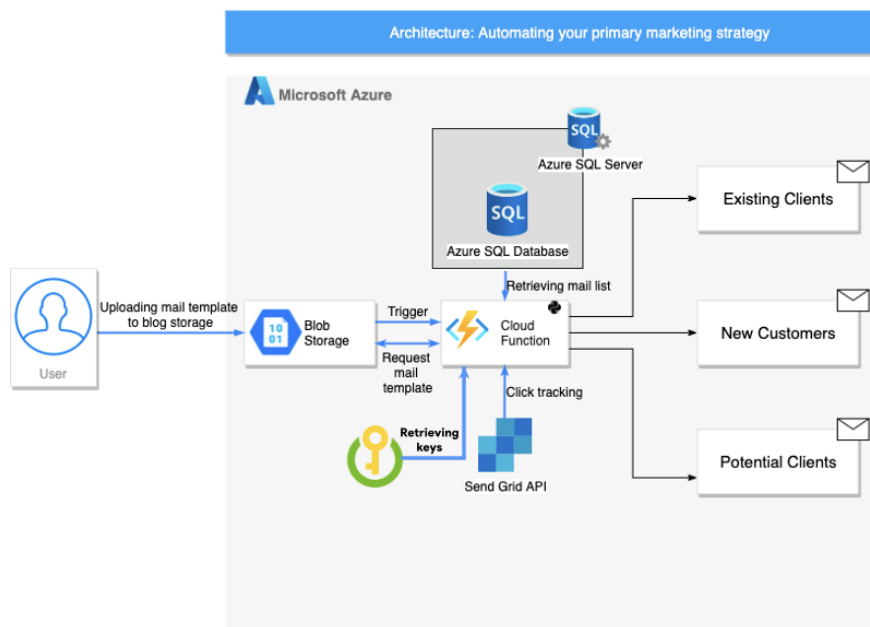
- The target list might consist of thousands of recipients.
- The list might change from time to time depending on the content that is being promoted by the business.
- Follow-up without a proper record is very difficult.
- Manually sending emails to each and every client is painstaking

- Using a third-party bulk mailing service is difficult to manage and very expensive.

## Solution



Architecture Diagram:



Architecture Diagram

Due to business restrictions, I cannot provide the entire code base. However, I can provide the exact steps to replicate the whole automation process based on your requirement.

I will present the above-mentioned automation in **5 steps**:

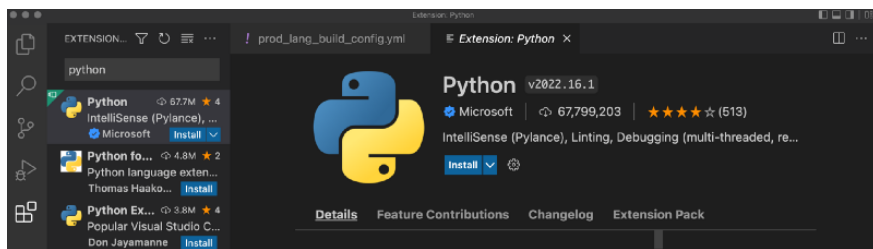
1. Writing the Python code which consumes the SendGrid API
2. Deploying the code to Azure Functions

3. Creating an Azure SQL Database and establishing a connection with the function
4. Creating and storing a mail template in Azure Blob Storage
5. Testing the workflow to send bulk emails to multiple recipients

## Step 1:

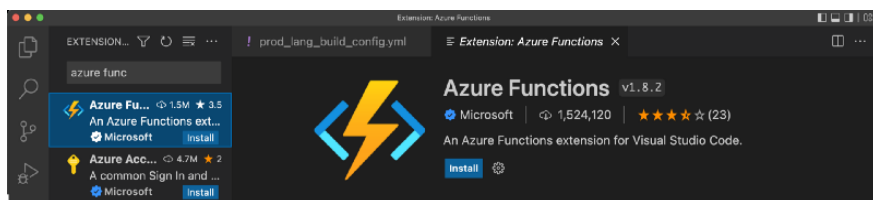
Pre-requisites:

- An Azure account with an active subscription. Create an account for free.
- The Azure Functions Core Tools version 3.x.
- Python versions that are supported by Azure Functions.
- Visual Studio Code on one of the supported platforms.
- The Python extension for Visual Studio Code.



Python VS Code extension

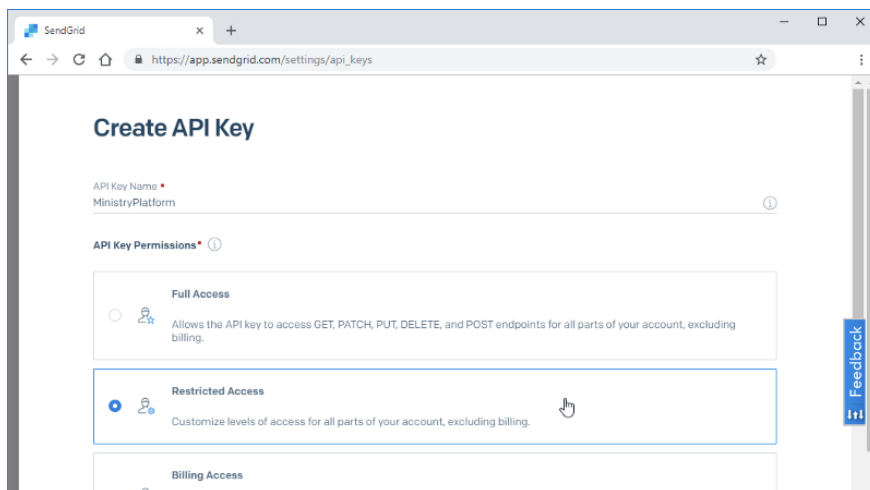
- The Azure Functions extension for Visual Studio Code.



Azure Functions VS Code extension

## Creating the SendGrid API Key

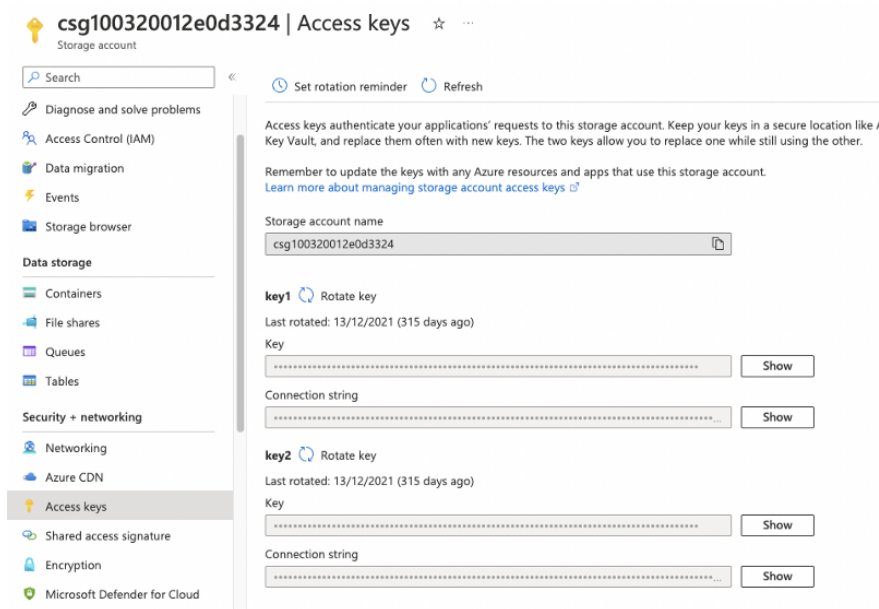
On the SendGrid portal, navigate to the settings section and create the key with the desired level of access and copy it.



Creating SendGrid API key

## Getting the Cloud Storage connection string

On your Storage Account, navigate to Access Keys and copy the connection string.



Getting Azure Storage connection string

## Substituting values in the code

Below is the code snippet for consuming the SendGrid API. I have used Azure Key Vault to securely store and retrieve the keys.

You would need to:

- Update SendGrid API Key & config.json.

- Update the Azure Storage connection string.
- Provide the Azure function access to the Azure SQL database.

```

1  import os
2  import json
3  import logging
4  import base64
5  import azure.functions as func
6  import os
7  import pyodbc
8  import struct
9  import azure.functions as func
10 import sqlalchemy
11 import sendgrid
12 from azure.storage.blob import BlobServiceClient, Blo
13 from io import BytesIO
14 from sendgrid.helpers.mail import Mail, Email, To, Co
15 from azure.identity import DefaultAzureCredential
16 from azure.keyvault.secrets import SecretClient
17
18 credential = DefaultAzureCredential()
19 client = SecretClient(vault_url=KVUri, cred=cred)
20 user = client.get_secret(uname)
21 passs = client.get_secret(passs)
22 dbname = client.get_secret(dbname)
23 hname = client.get_secret(hname)
24 pno = client.get_secret(pno)
25 skey = client.get_secret(skey)
26 scs = client.get_secret(scs)
27
28 def main(intblobstore: func.InputStream):
29     logging.info(f"Initiated by new blob: {intblobsto
30
31     connection_string = scs
32     sendgrid_api_key = skey
33     blob_service_client = BlobServiceClient.from_conn
34     container_client = blob_service_client.get_contai
35     blob_client = container_client.get_blob_client("m
36     blobstr = blob_client.download_blob().readall().d
37
38     cnf_path = os.path.join('blob-trigger-with-config
39
40     with open(cnf_path) as json_file:
41         email_details = json.load(json_file)
42
43     pool = sqlalchemy.create_engine(
44         sqlalchemy.engine.url.URL(

```

```

45         drivename="mysql+pymysql",
46         username=uname,
47         password=passs,
48         host=hname,
49         port=pno,
50         database=dbname,
51     ),
52 )
53
54 query='SELECT email_id FROM send_list'
55
56 with pool.connect() as connection:
57     statement = sqlalchemy.text(query)
58     output = connection.execute(statement)
59
60 mail_list=[]
61 for i in result:
62     mail_list.append(i[0])
63

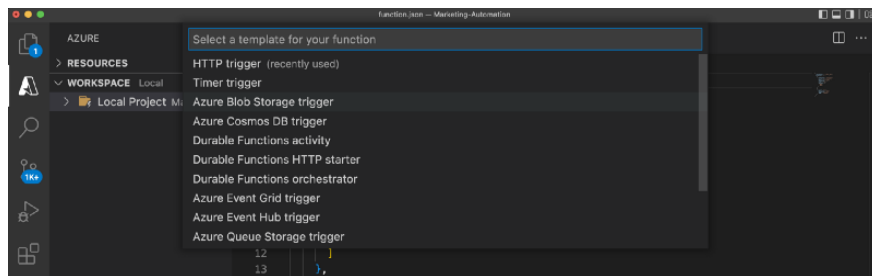
```

Mail Automation

## Step 2:

### Creating the function locally

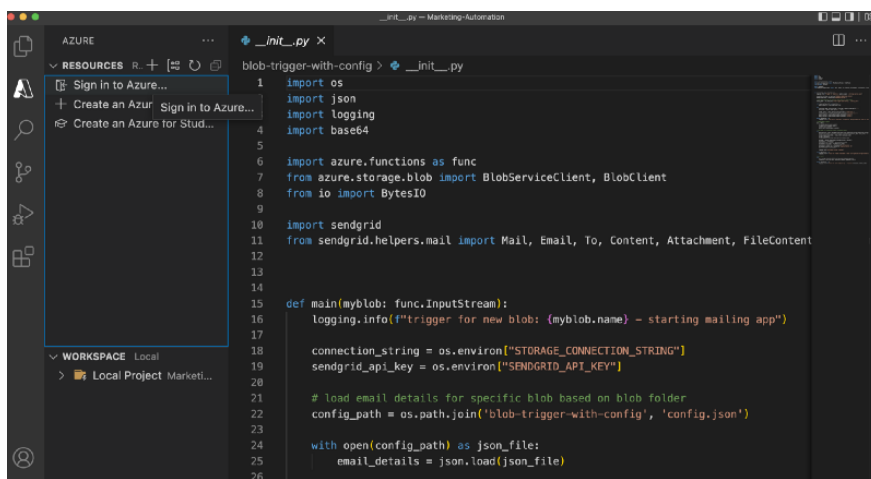
Click on your workspace's "+" icon to create a function with the Azure Blob Storage trigger.



Creating function locally with Blob Storage trigger

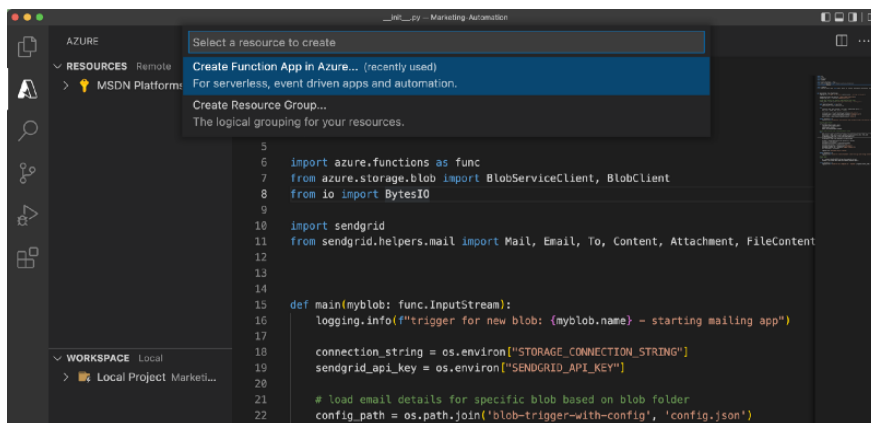
Select Python and the authorization level as anonymous (for testing).

Connect your VS code to Azure by signing in



Signing in to the Azure account

Once signed in, click on create function app.



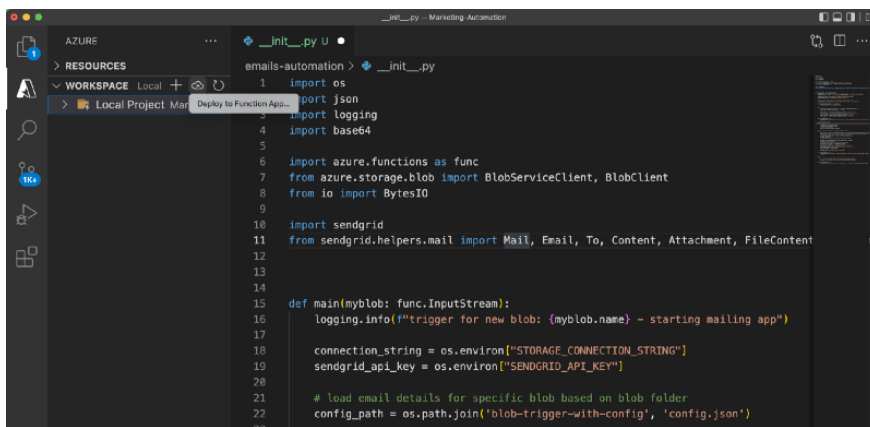
Creating a Function App in Azure

You will need to fill in the following details:

- Subscription
- Globally unique function name
- Runtime stack (Python in our case)
- Location of resources

Finally, deploy the local function to Azure.



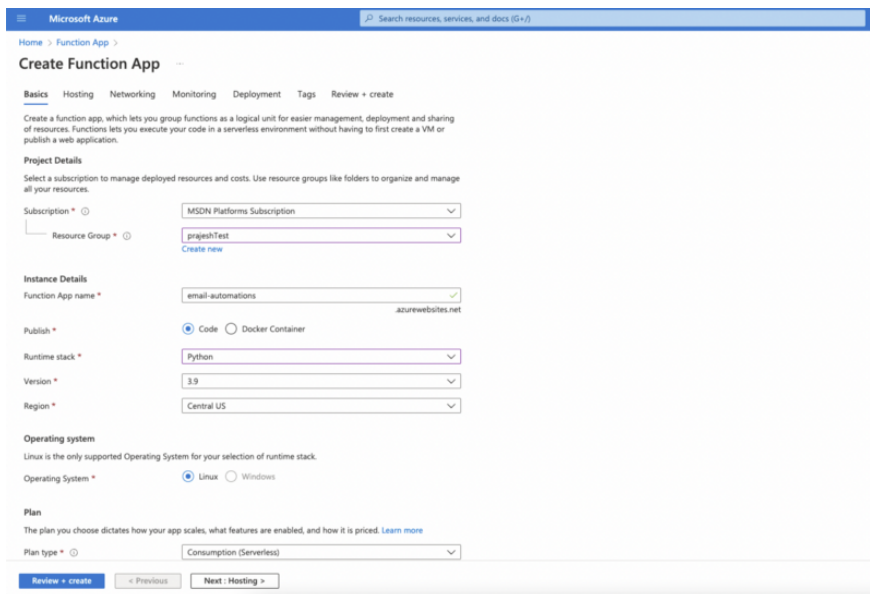


```
1 import os
2 import json
3 import logging
4 import base64
5
6 import azure.functions as func
7 from azure.storage.blob import BlobServiceClient, BlobClient
8 from io import BytesIO
9
10 import sendgrid
11 from sendgrid.helpers.mail import Mail, Email, To, Content, Attachment, FileContent
12
13
14
15 def main(myblob: func.InputStream):
16     logging.info(f"trigger for new blob: {myblob.name} - starting mailing app")
17
18     connection_string = os.environ["STORAGE_CONNECTION_STRING"]
19     sendgrid_api_key = os.environ["SENDGRID_API_KEY"]
20
21     # load email details for specific blob based on blob folder
22     config_path = os.path.join('blob-trigger-with-config', 'config.json')
```

Deploying our function to Azure Function App

Alternatively, you can create a function app from the Azure Portal.

Below are the configurations:



Microsoft Azure

Home > Function App >

### Create Function App

Basics | Hosting | Networking | Monitoring | Deployment | Tags | Review + create

Create a function app, which lets you group functions as a logical unit for easier management, deployment and sharing of resources. Functions lets you execute your code in a serverless environment without having to first create a VM or publish a web application.

**Project Details**

Select a subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription \*

Resource Group \*  [Create new](#)

**Instance Details**

Function App name \*  azurewebsites.net

Publish \* ☒ Code ☐ Docker Container

Runtime stack \*

Version \*

Region \*

**Operating system**

Linux is the only supported Operating System for your selection of runtime stack.

Operating System \* ☒ Linux ☐ Windows

**Plan**

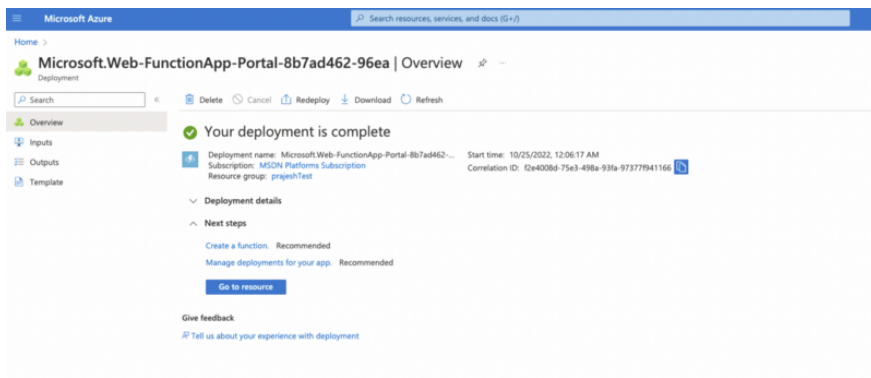
The plan you choose dictates how your app scales, what features are enabled, and how it is priced. [Learn more](#)

Plan type \*

[Review + create](#) [< Previous](#) [Next : Hosting >](#)

Function App creation from the portal

Once the deployment is successful, you will see this screen.

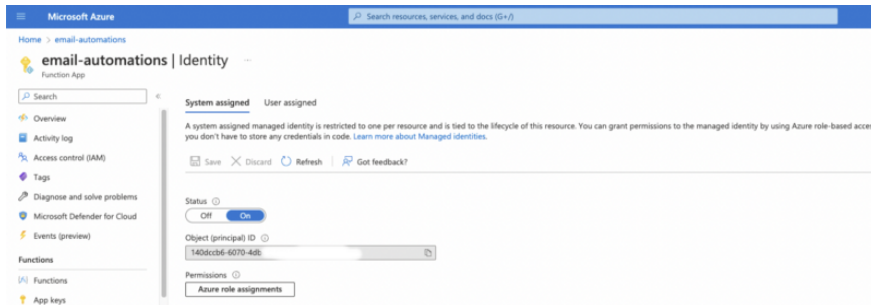


Function App deployment

### Step 3:

In order to enable the function app to connect with the Azure SQL database, you will need a system-assigned managed identity.

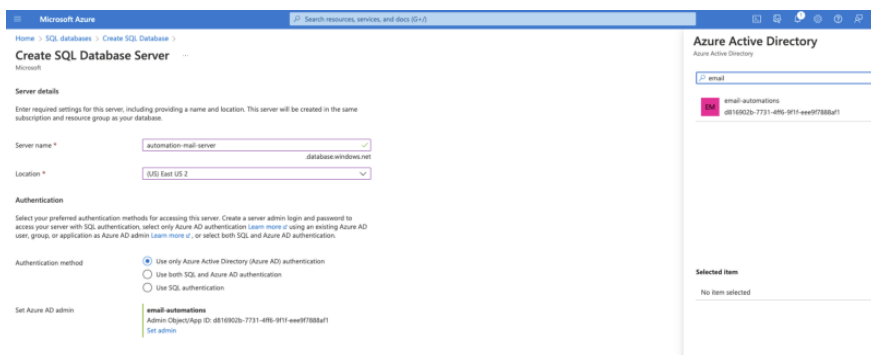
To enable it, navigate to your deployed function app. Go to the identity section, and change the status to “On”.



Enabling system-managed identity for Function App

### Creating the Azure SQL Database

Navigate to Azure SQL Databases on the portal and click on create. Create a new Azure SQL server, give it a unique name and under authentication select use only Azure AD authentication.



Creating Azure SQL Database and SQL Server

Then set the Azure AD admin. From the section on the right, select the function app's system-assigned managed identity and click on create.

Microsoft Azure

Search resources, services, and docs (G+/)

Home > SQL databases >

Create SQL Database

...

Microsoft

Authentication method

Azure Active Directory authentication only

Azure AD Admin

email-automations

Compute + storage

General Purpose - Serverless: Standard-series (Gen5), 1 vCore, 32 GB storage, zone redundant disabled

Backup storage redundancy

Locally-redundant backup storage

Networking

Allow Azure services and resources to access this server

No

Private endpoint

None

Minimum TLS version

1.2

Connection Policy

Default

Security

Identity

Not enabled

Service principal (preview)

Off

Transparent data encryption

Service-managed key selected

Advanced data security

Not now

Sql Ledger(Database)

Disabled

Digest Storage

Disabled

Additional settings

Use existing data

Blank

Collation

SQL\_Latin1\_General\_CP1\_CI\_AS

Maintenance window

System default (5pm to 8am)

Tags

Create

< Previous

[Download a template for automation](#)

Azure SQL database creation

Once the database is created, create a table and populate it with your recipient's email addresses.

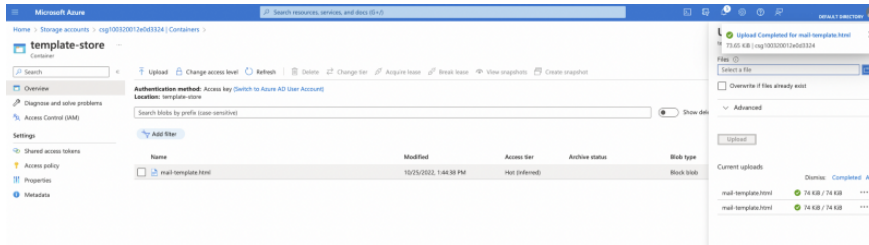
Since our function is able to connect with the Azure SQL database, we can query it directly. **All we need to do is update the database details in the Azure Key Vault.**

Our function will retrieve the recipient mail addresses and store them in a Python list. The list would then be iterated and all the mail addresses would be sent an email using the SendGrid API which was consumed in the Python code snippet.

### Step 4:

To upload the template to Azure Storage, create a storage account, then create a container, and upload the email template.

To get started, you can use open-source templates available at <https://beefree.io/>



Creating and uploading template to Azure Blob

## Step 5:

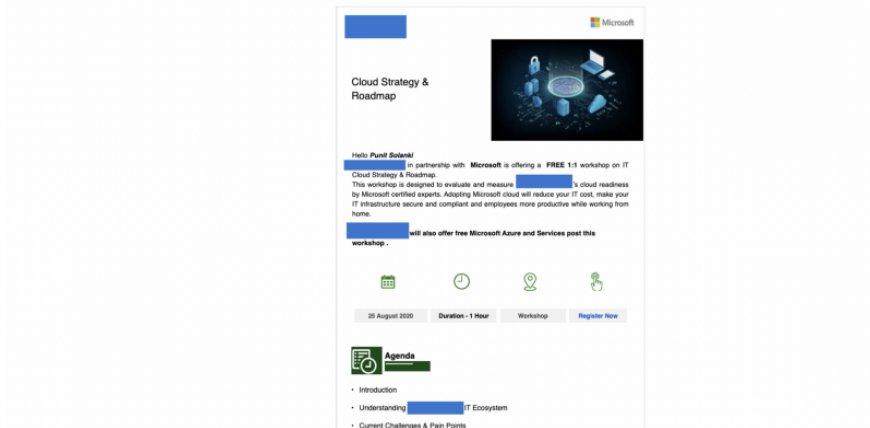
Now, for the final step, we will test our entire workflow to ensure that the configurations are correct.

Since we have created a function with a blob storage trigger, it will execute every time a new template is uploaded.

## Custom configurations:

- Implemented function-level logic to ensure the correct template is sent to the correct recipients.
- Wrote Python code to replace fields like “First Name”, “Last Name”, etc with the names of customers/clients corresponding with the email addresses. This adds a personalised touch to the email.
- Took consent from the customers/clients and ensured that they wanted to be added to the recipient list.

**Voila! This was the email which was received using this automation:**



Mail received

## Knowledge Sharing and Best Practices:

- Using system-assigned managed identity prevents the usage of keys within code which is a bad practice for production environments.
- In my environment, the SendGrid API key and other sensitive information were stored in Azure Key Vault and retrieved from there when required. This enhances the robustness of the automation
- The authorization level “anonymous” for a function app is not recommended for the production environment.

## Challenges Faced:

The automation was tricky to implement due to the multiple components involved. Integrating it with Azure SQL Database, creating a blob-triggered function using Python and implementing the logic to customize the email was challenging and fun!

Secondly, the SendGrid API offers features that can be used with Python to track clicks in the email. This was the most difficult part to implement.

## Business Benefits:

- Mailing overhead was reduced greatly
- Tracking and follow-up became efficient
- There was a 5X reduction in cost!

- **Business flourished** in months because of the improved mailing strategy
- This **solution itself was further implemented** as a business requirement for 2 clients!
- After success in the marketing aspect, this was further **implemented for internal communication between departments.**

## References:

Deploying Azure Functions: <https://learn.microsoft.com/en-us/azure/azure-functions/create-first-function-vs-code-python>

Connecting to Azure Cloud Database:  
<https://techcommunity.microsoft.com/t5/apps-on-azure-blog/how-to-connect-azure-sql-database-from-python-function-app-using/ba-p/3035595>

SendGrid Python API: <https://www.twilio.com/blog/how-to-send-emails-in-python-with-sendgrid>