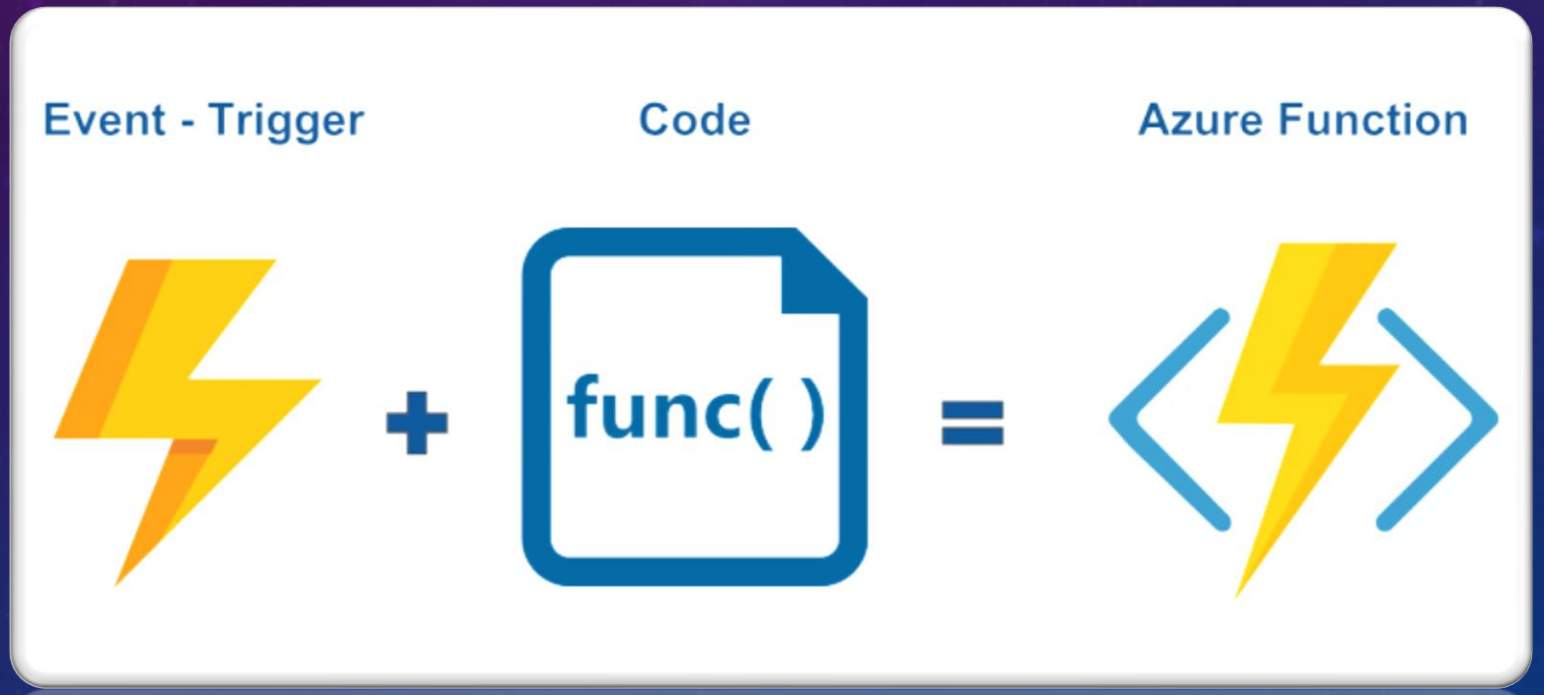
FUNCTION APPS

RESEARCH STORY

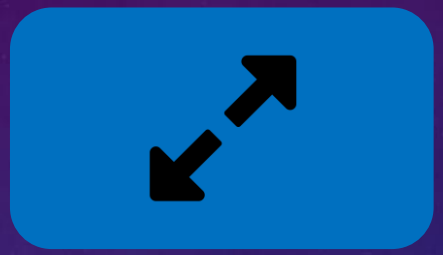
What is Azure Functions

Execute a piece of code in response to an event



Serverless

No worrys about
infrastructure
Only code matters



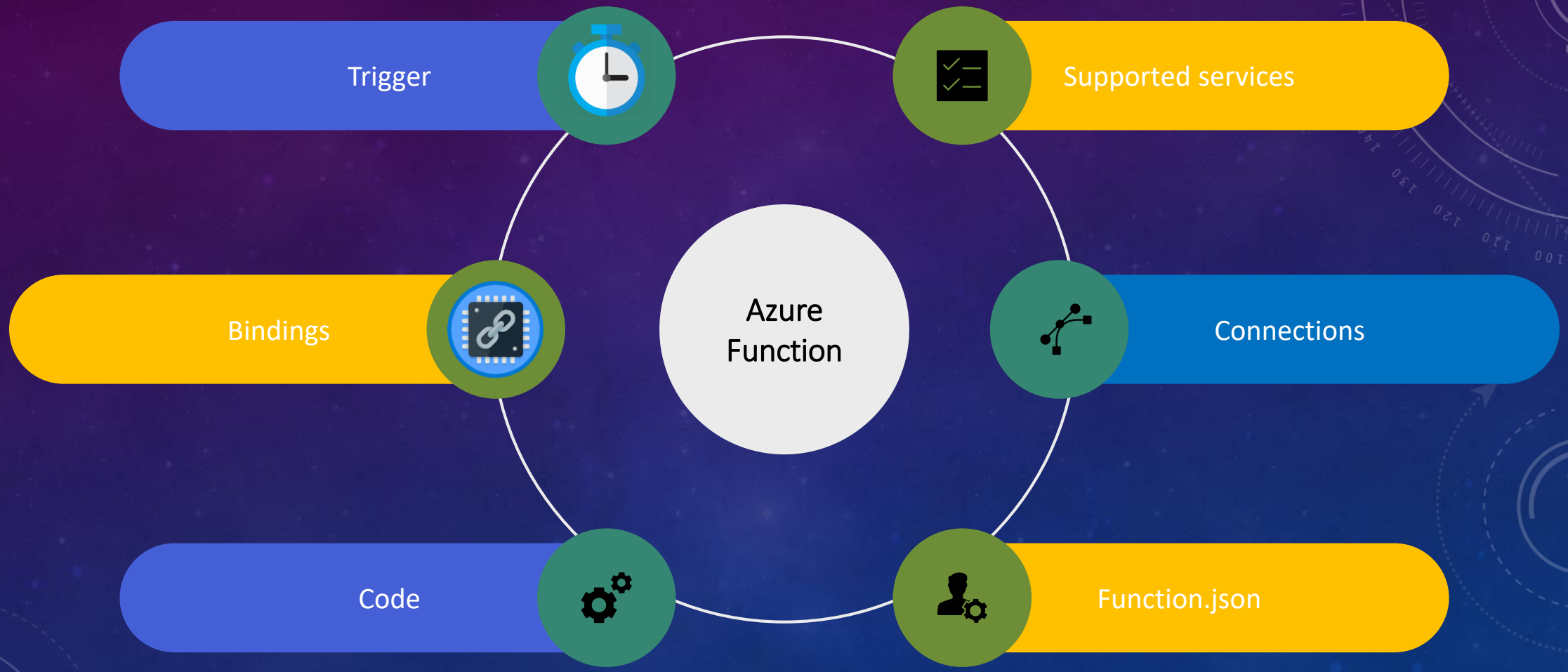
Auto scalling
Event driven

Pay on use

Scenarios

If you want to...	then...
Build a web API	Implement an endpoint for your web applications using the HTTP trigger
Process file uploads	Run code when a file is uploaded or changed in blob storage
Build a serverless workflow	Chain a series of functions together using durable functions
Respond to database changes	Run custom logic when a document is created or updated in Cosmos DB
Run scheduled tasks	Execute code on pre-defined timed intervals
Create reliable message queue systems	Process message queues using Queue Storage, Service Bus, or Event Hubs
Analyze IoT data streams	Collect and process data from IoT devices
Process data in real time	Use Functions and SignalR to respond to data in the moment

Function app



Function app

Triggers

1 Trigger per function

Trigger can be an input

Choose trigger when creating function

Bindings
(Unique to Azure functions)

Trigger

Input Binding

Output Binding

Supported Services

Azure function has support for a list of azure services that it can use as triggers, inputs and outputs.

Connections

Configuration within functions

Managed identity

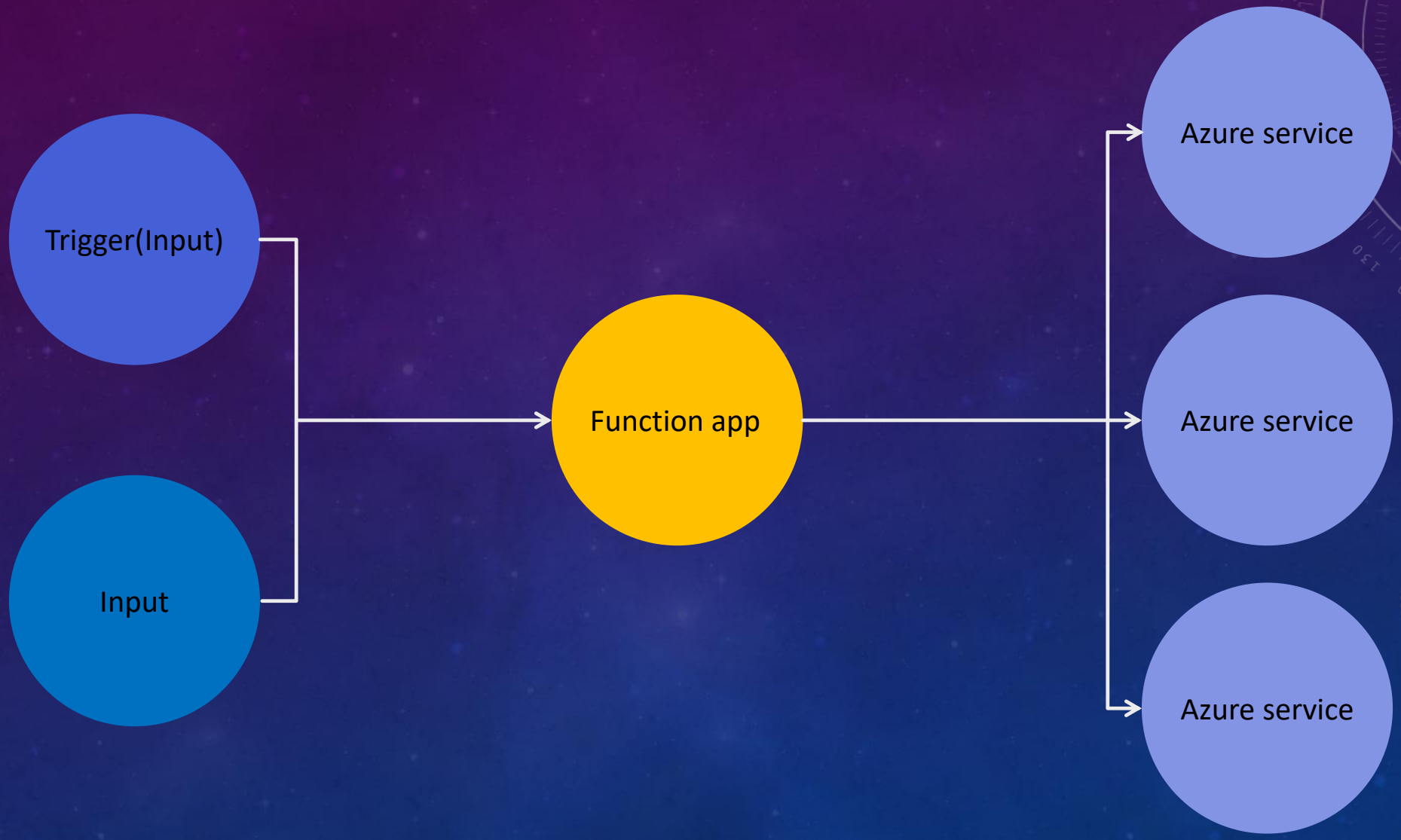
Key Vault

Code/ Function.json

Code file that contains executed code

Function.json is where you define bindings.

Voorbeeld


Trigger(Input)
Input
Function app
Azure service
Azure service
Azure service

Supported services

Type	1.x	2.x and higher[1]	Trigger	Input	Output
Blob storage	✓	✓	✓	✓	✓
Azure Cosmos DB	✓	✓	✓	✓	✓
Azure SQL (preview)		✓		✓	✓
Dapr[3]		✓	✓	✓	✓
Event Grid	✓	✓	✓		✓
Event Hubs	✓	✓	✓		✓
HTTP & webhooks	✓	✓	✓		✓
IoT Hub	✓	✓	✓		
Kafka[2]		✓	✓		✓
Mobile Apps	✓			✓	✓
Notification Hubs	✓				✓
Queue storage	✓	✓	✓		✓
RabbitMQ[2]		✓	✓		✓
SendGrid	✓	✓			✓
Service Bus	✓	✓	✓		✓
SignalR		✓	✓	✓	✓
Table storage	✓	✓		✓	✓
Timer	✓	✓	✓		
Twilio	✓	✓			✓

Bindings

- Input / Output
- No connections in the code
- Declared as parameter in the code
- Easy to connect with other services

Language	Triggers and bindings are configured by...
C# class library	decorating methods and parameters with C# attributes
Java	decorating methods and parameters with Java annotations
JavaScript/PowerShell/Python/TypeScript	updating function.json (schema)

Example scenario	Trigger	Input binding	Output binding
A new queue message arrives which runs a function to write to another queue.	Queue*	*None*	Queue*
A scheduled job reads Blob Storage contents and creates a new Cosmos DB document.	Timer	Blob Storage	Cosmos DB
The Event Grid is used to read an image from Blob Storage and a document from Cosmos DB to send an email.	Event Grid	Blob Storage and Cosmos DB	SendGrid
A webhook that uses Microsoft Graph to update an Excel sheet.	HTTP	*None*	Microsoft Graph

Example code

FunctionAppPython > RsServiceBusQueueTrigger > {} function.json > ...

```json
{
  "scriptFile": "__init__.py",
  "bindings": [
    {
      "name": "input",
      "type": "serviceBusTrigger",
      "direction": "in",
      "queueName": "rsqueueinput",
      "connection": "RsServiceBusHvA_SERVICEBUS"
    },
    {
      "name": "output",
      "direction": "out",
      "type": "serviceBus",
      "queueName": "rsqueueoutput",
      "connection": "RsServiceBusHvA_SERVICEBUS"
    }
  ]
}
```

FunctionAppPython > RsServiceBusQueueTrigger > __init__.py > ...

```python
import logging
import azure.functions as func

def main(input: func.ServiceBusMessage,
         output: func.Out[str]):
    logging.info('Python ServiceBus queue trigger processed message: %s',
                 input.get_body().decode('utf-8'))
    output.set(input.get_body().decode('utf-8') + ' toevoeging ServiceBus')
```

Return binding

C# Copy

```csharp
[FunctionName("QueueTrigger")]
[return: Blob("output-container/{id}")]
public static string Run([QueueTrigger("inputqueue")]WorkItem input, ILogger log)
{
    string json = string.Format("{{ \"id\": \"{0}\" }}", input.Id);
    log.LogInformation($"C# script processed queue message. Item={json}");
    return json;
}
```

JSON Copy

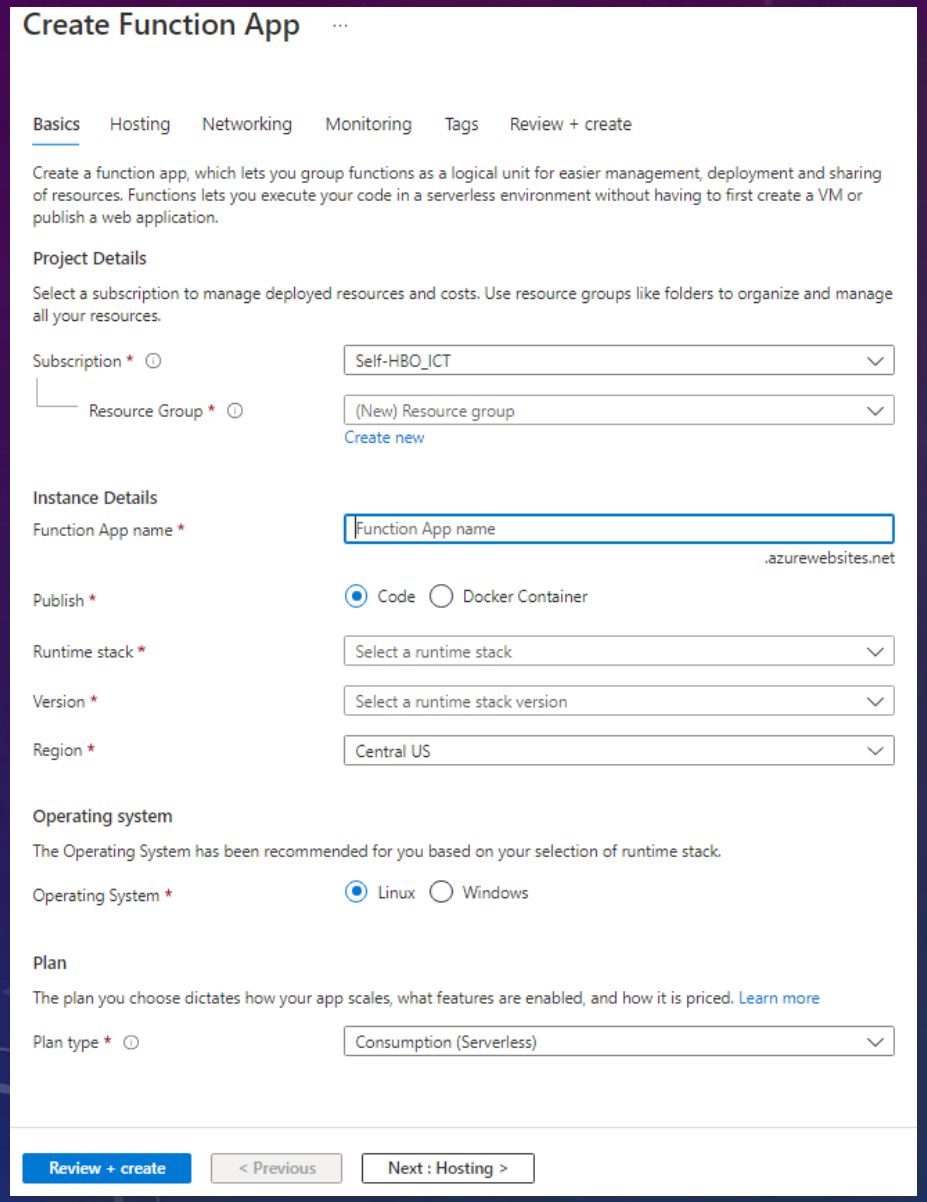
```json
{
    "name": "$return",
    "type": "blob",
    "direction": "out",
    "path": "output-container/{id}"
}
```

Here's the Python code:

Python Copy

```python
def main(input: azure.functions.InputStream) -> str:
    return json.dumps({
        'name': input.name,
        'length': input.length,
        'content': input.read().decode('utf-8')
    })
```

Creating a Function App

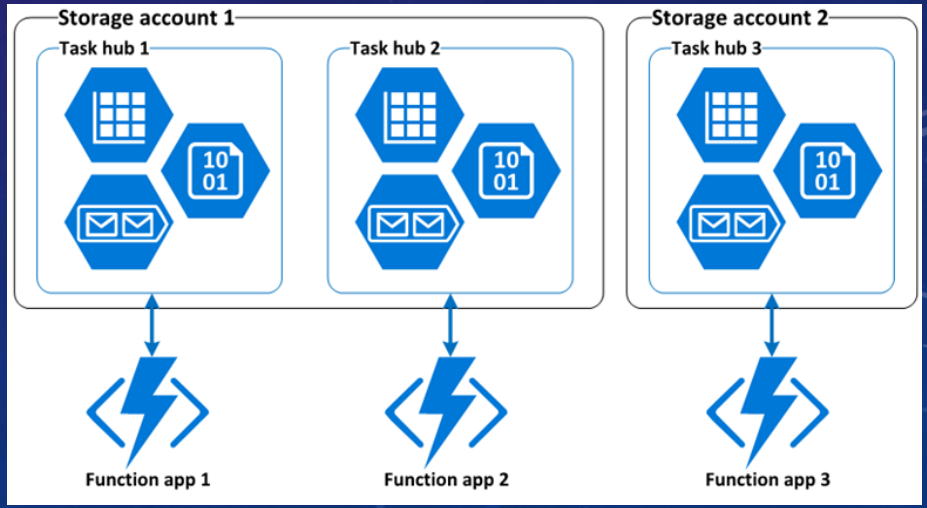


- Python, C#, Java, Node, Powershell
- Serverless, Premium

Storage account

Storage service	Functions usage
Azure Blob Storage	Maintain bindings state and function keys. Also used by task hubs in Durable Functions.
Azure Files	File share used to store and run your function app code in a Consumption Plan and Premium Plan. Azure Files is set up by default, but you can create an app without Azure Files under certain conditions.
Azure Queue Storage	Used by task hubs in Durable Functions and for failure and retry handling by specific Azure Functions triggers.
Azure Table Storage	Used by task hubs in Durable Functions.

When creating a function app, you must create or link to a general-purpose Azure Storage account that supports Blob, Queue, and Table storage. This is because Functions relies on Azure Storage for operations such as managing triggers and logging function executions



Creating a Function

Create function

Instructions will vary based on your development environment. Learn more

Development environ... VS Code

Install dependencies

Before you can get started, you should install Visual Studio Code. You should also install NodeJS which includes npm. This is how you will obtain the Azure Functions Core Tools. If you prefer not to install Node, see the other installation options in our Core Tools reference.

Run the following command to install the Core Tools package:

```
npm install -g azure-functions-core-tools@4 --unsafe-perm true
```

Next, install the Azure Functions extension for Visual Studio Code. Once the extension is installed, click on the Azure logo in the Activity Bar. Under **Azure: Functions**, click **Sign in to Azure...** and follow the on-screen instructions.

Create an Azure Functions project

Click the **Create New Project...** icon in the **Azure: Functions** panel.

You will be prompted to choose a directory for your app. Choose an empty directory.

You will then be prompted to select a language for your project. Choose .

Create a function

Click the **Create Function...** icon in the **Azure: Functions** panel.

You will be prompted to choose a template for your function. We recommend HTTP trigger for getting started.

Run your function project locally

Press **F5** to run your function app.

The runtime will output a URL for any HTTP functions, which can be copied and run in your browser's address bar.

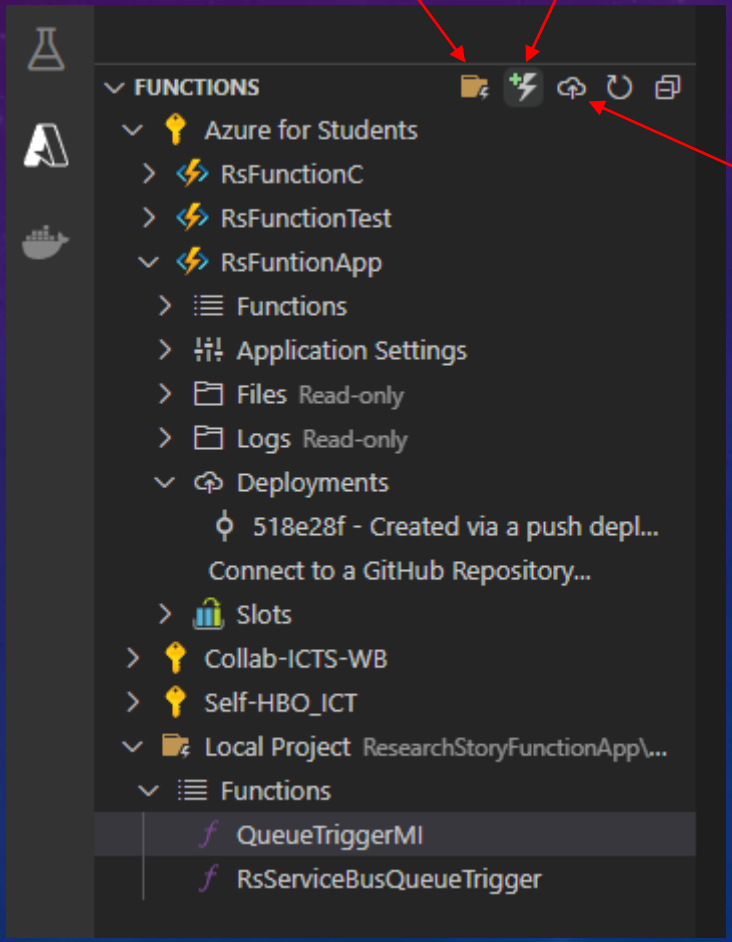
To stop debugging, press **Shift + F5**.

Creating and deploying a Function

Create project

Create function

Deploy to azure



Connections Configuration

Settings

- Configuration
- Authentication
- Application Insights
- Identity
- Backups
- Custom domains
- TLS/SSL settings

Filter application settings

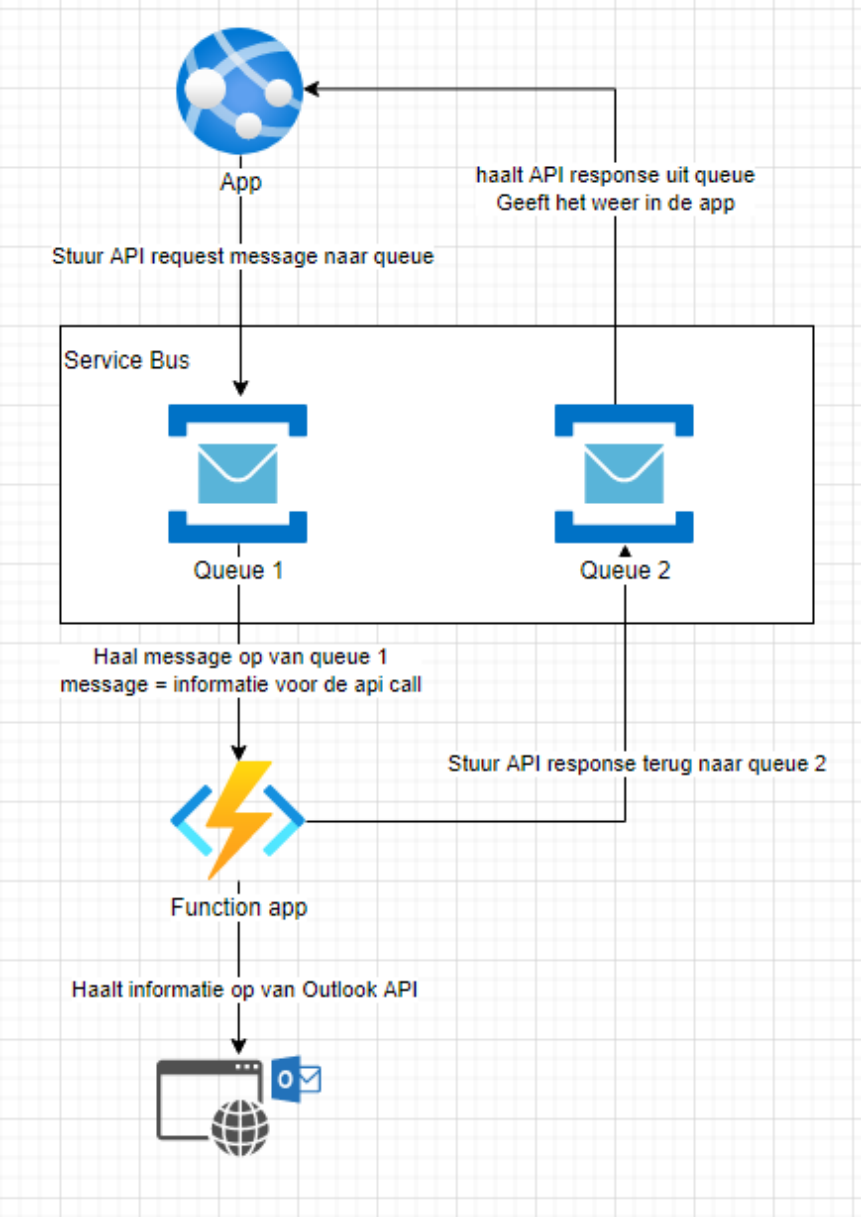
Name	Value
APPINSIGHTS_INSTRUMENTATIONKEY	Hidden value. Click to show value
APPLICATIONINSIGHTS_CONNECTION_STRING	Hidden value. Click to show value
AzureWebJobsStorage	Hidden value. Click to show value
FUNCTIONS_EXTENSION_VERSION	Hidden value. Click to show value
FUNCTIONS_WORKER_RUNTIME	Hidden value. Click to show value
RsServiceBusHvA_SERVICEBUS	Endpoint=sb://rsservicebushva.servicebus.windows.net/;SharedAccessKeyName=RootMa
WEBSITE_CONTENTAZUREFILECONNECTIONSTRING	Hidden value. Click to show value
WEBSITE_CONTENTSHARE	Hidden value. Click to show value

Connections Configuration

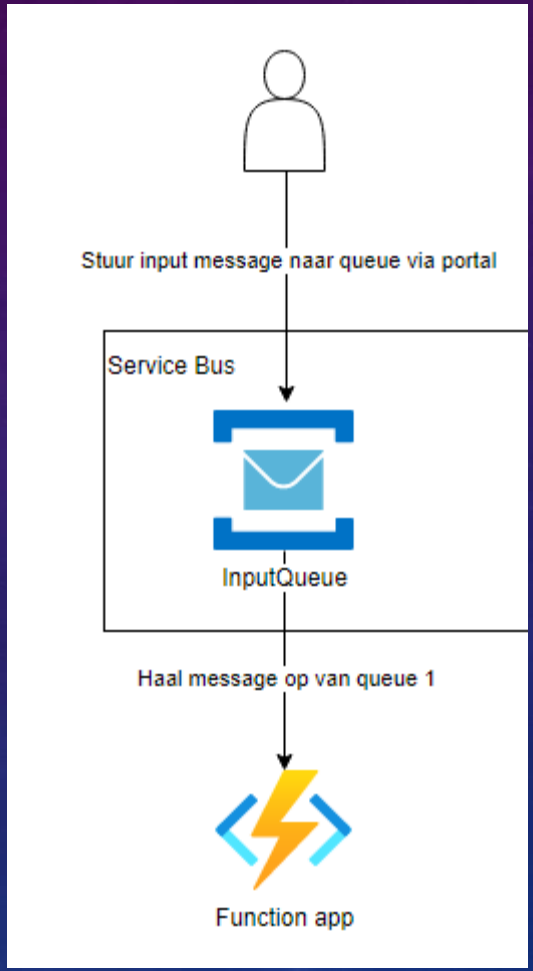
FunctionAppPython > RsServiceBusQueueTrigger > {} function.json > ...

```
1  {
2    "scriptFile": "__init__.py",
3    "bindings": [
4      {
5        "name": "input",
6        "type": "serviceBusTrigger",
7        "direction": "in",
8        "queueName": "rsqueueinput",
9        "connection": "RsServiceBusHvA_SERVICEBUS"
10     },
11     {
12       "name": "output",
13       "direction": "out",
14       "type": "serviceBus",
15       "queueName": "rsqueueoutput",
16       "connection": "RsServiceBusHvA_SERVICEBUS"
17     }
18   ]
19 }
20
```

Project voorbeeld



Default voorbeeld



Code + Function.json Default

RsServiceBusQueueTrigger > {} function.json > ...

```json
{
  "scriptFile": "__init__.py",
  "bindings": [
    {
      "name": "msg",
      "type": "serviceBusTrigger",
      "direction": "in",
      "queueName": "rsqueueinput",
      "connection": "RsServiceBusHvA_SERVICEBUS"
    }
  ]
}
```

RsServiceBusQueueTrigger > __init__.py > ...

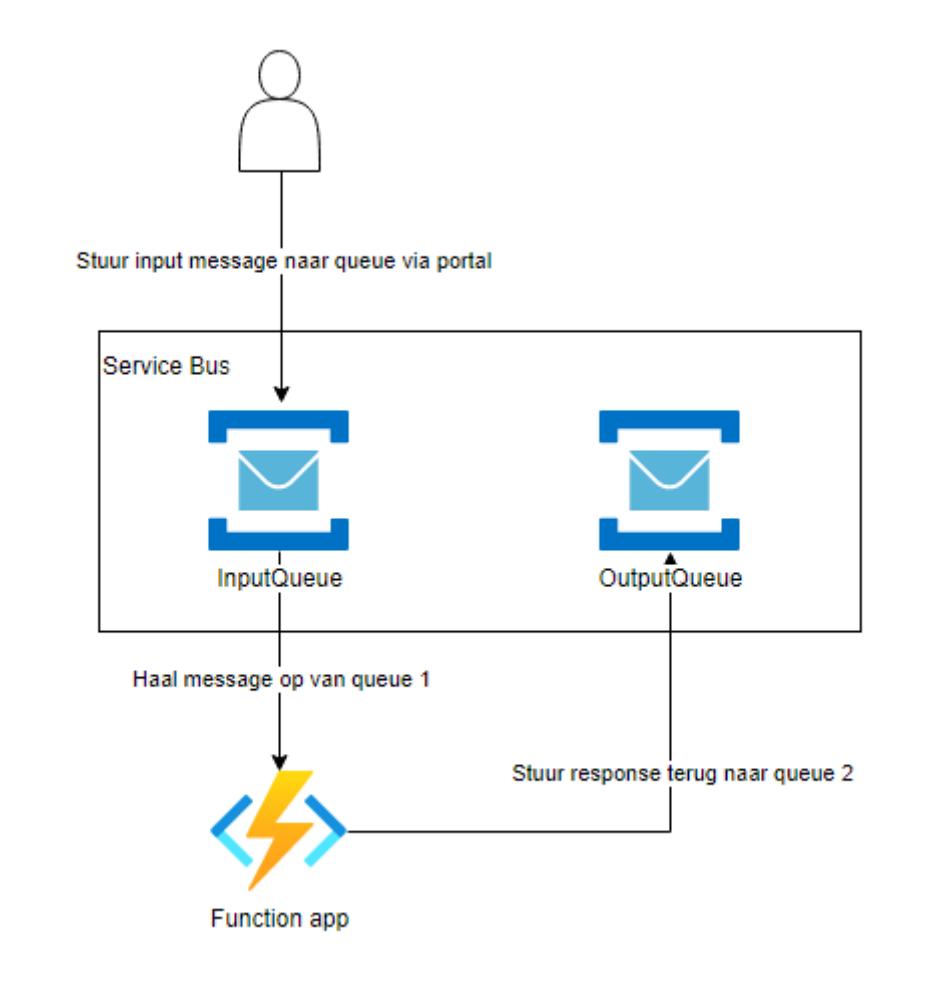
```python
import logging

import azure.functions as func


def main(msg: func.ServiceBusMessage):
    logging.info('Python ServiceBus queue trigger processed message: %s',
                 msg.get_body().decode('utf-8'))

```

Code + Function.json
Service Bus demo



Code + Function.json
Service Bus demo

```python
from azure.servicebus import ServiceBusClient, ServiceBusMessage

CONNECTION_STR = "Endpoint=sb://testservicebushva.servicebus.windows.net/;SharedAccessKeyName=RootManageSh
QUEUE_NAME = "testqueue"

# create a Service Bus client using the connection string
servicebus_client = ServiceBusClient.from_connection_string(conn_str=CONNECTION_STR, logging_enable=True)
sender = servicebus_client.get_queue_sender(queue_name=QUEUE_NAME)
# get the Queue Receiver object for the queue
receiver = servicebus_client.get_queue_receiver(queue_name=QUEUE_NAME, max_wait_time=5)

def send_single_message(sender):
    # create a Service Bus message
    singleMessage = "My name is Bob"
    message = ServiceBusMessage(singleMessage)
    # send the message to the queue
    sender.send_messages(message)
    print("Message: " + singleMessage + " deliverd.")

with sender:
    # send one message
    send_single_message(sender)

print("Done sending messages")
print("----------------------")

with receiver:
    for msg in receiver:
        print("Received: " + str(msg))
        # complete the message so that the message is removed from the queue
        receiver.complete_message(msg)
```

FunctionAppPython > RsServiceBusQueueTrigger > {} function.json > ...

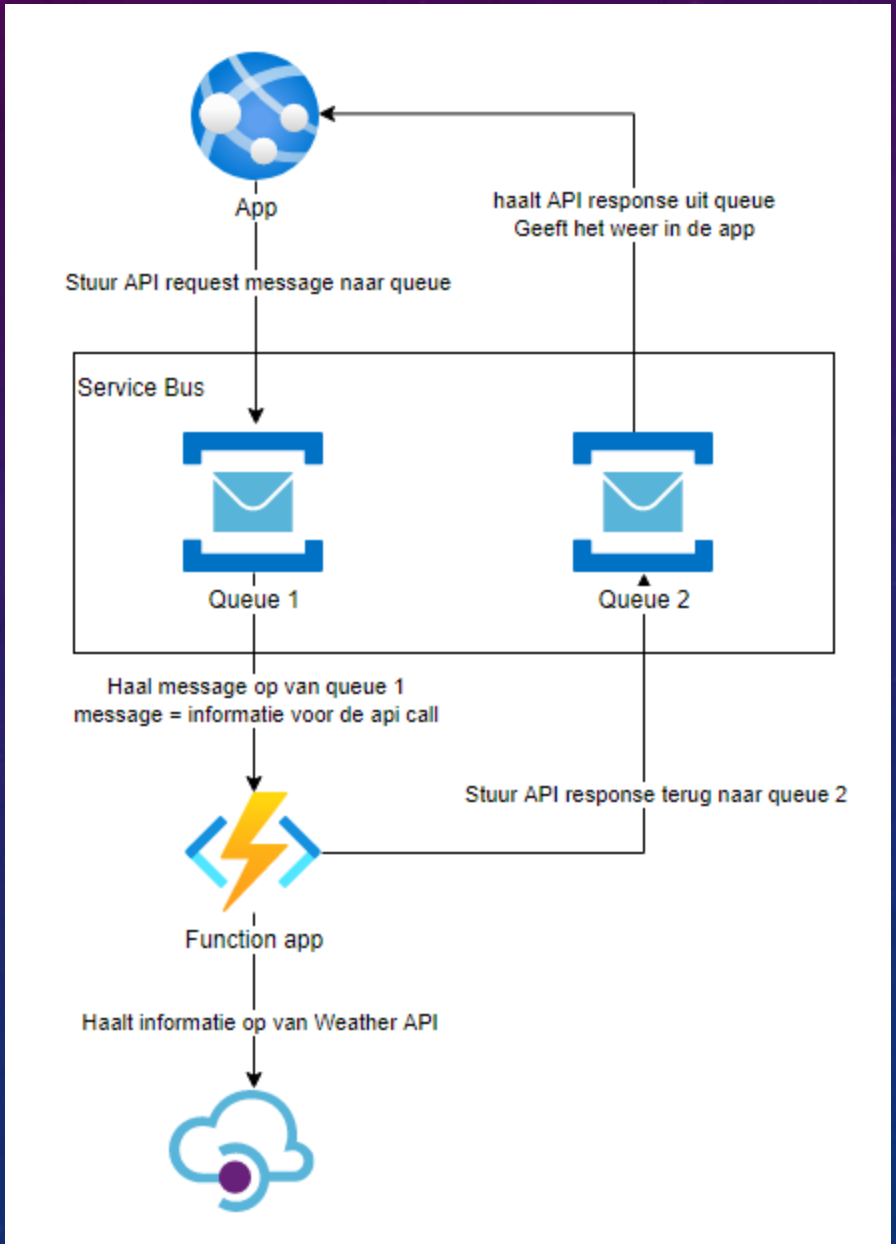
```json
{
  "scriptFile": "__init__.py",
  "bindings": [
    {
      "name": "input",
      "type": "serviceBusTrigger",
      "direction": "in",
      "queueName": "rsqueueinput",
      "connection": "RsServiceBusHvA_SERVICEBUS"
    },
    {
      "name": "output",
      "direction": "out",
      "type": "serviceBus",
      "queueName": "rsqueueoutput",
      "connection": "RsServiceBusHvA_SERVICEBUS"
    }
  ]
}
```

FunctionAppPython > RsServiceBusQueueTrigger > __init__.py > ...

```python
import logging
import azure.functions as func

def main(input: func.ServiceBusMessage,
         output: func.Out[str]):
    logging.info('Python ServiceBus queue trigger processed message: %s',
                 input.get_body().decode('utf-8'))
    output.set(input.get_body().decode('utf-8') + ' toevoeging ServiceBus')
```

Project Idea



Project Function

```python
from http.client import responses
import logging
import requests
import json
import azure.functions as func

#The main method that runs when app is triggerd
#The inputRequest is the trigger input
#The outputWeatherAPI is the output these are defined in the function.json
def main(inputRequest: func.ServiceBusMessage,
         outputWeatherAPI: func.Out[str]):
    # Log the input so we can see what it gets from the queue
    logging.info('Python ServiceBus queue trigger processed message: %s',
                 inputRequest.get_body().decode('utf-8'))
    #Create the request URL for the weather API with the input from the queue
    message =  inputRequest.get_body().decode('utf-8')
    api_url = "https://api.openweathermap.org/data/2.5/weather?q=" + message +
    #Request data from weather API with URL
    response = requests.get(api_url)
    #Log the response of the Weather API so we can see what it sends back
    logging.info(response.json())
    #Convert json to a string so that we can send it back to the second queue
    data = json.loads(response.text)
    jsonToString = json.dumps(data)
    #Set the value of output to the json to send back
    outputWeatherAPI.set(jsonToString)
```

```json
{
  "scriptFile": "__init__.py",
  "bindings": [
    {
      "name": "inputRequest",
      "type": "serviceBusTrigger",
      "direction": "in",
      "queueName": "fromwebappwheatherdata",
      "connection": "MixitAppServiceBus_SERVICEBUS"
    },
    {
      "name": "outputWeatherAPI",
      "direction": "out",
      "type": "serviceBus",
      "queueName": "fromweerapptowebapp",
      "connection": "MixitAppServiceBus_SERVICEBUS"
    }
  ]
}
```

Configuration

Name	Value
APPINSIGHTS_INSTRUMENTATIONKEY	Hidden value. Click to show value
APPLICATIONINSIGHTS_CONNECTION_STRING	Hidden value. Click to show value
AzureWebJobs.ServiceBusQueueTrigger1.Disabled	Hidden value. Click to show value
AzureWebJobsStorage	Hidden value. Click to show value
FUNCTIONS_EXTENSION_VERSION	Hidden value. Click to show value
FUNCTIONS_WORKER_RUNTIME	Hidden value. Click to show value
MixitAppServiceBus_SERVICEBUS	Hidden value. Click to show value
WEBSITE_CONTENTAZUREFILECONNECTIONSTRING	Hidden value. Click to show value
WEBSITE_CONTENTSHARE	Hidden value. Click to show value

THE END