

# Getting started with Azure Python functions



Katia Gil Guzman

[Follow](#)

Aug 7, 2019 · 10 min read



## What are Azure functions?

Functions allow to run **serverless code**, code that you run only when you need it and that doesn't need a whole infrastructure to live in. You can decide that a function is triggered when hitting an HTTP endpoint, or when new files are added into a blob storage for instance. It might always perform the same task, or it might do something different depending on your inputs. Basically, you can write code that does (almost) anything you can think of and host it as an Azure Function.

...

Read more stories this month when you [create a free Medium account](#).



For more information on Azure functions, take a look at the documentation.

• • •

*In this tutorial, we'll learn how to create an HTTP triggered function and receive and return data. To illustrate the use of external libraries and show how the code can be segmented, we'll create a function that performs text processing, and we'll finish by publishing it to Azure.*

## 1. Create a new function app

The documentation explains well how to get started quickly with Azure Python Functions, but if you prefer to stay here with me (I'm honored), **here are the steps to setting up an Azure Python Functions project:**

- Install Python 3.6
- Install Azure Functions Core Tools:

```
npm install -g azure-functions-core-tools
```

- Install the Azure CLI version 2.x or a later version.

If you don't already have an Azure subscription, you can sign up for a free trial [here](#).

Note that only Python 3.6 is supported for now, so make sure you have this version installed (it won't work with Python 3.7).

• • •

**Next step is creating a new virtual environment.** Even if it's not mandatory, you

Read more stories this month when you [create a free Medium account](#).



To create a virtual environment, navigate to where you want to set up your project and use this Bash command:

```
python3.6 -m venv .venv  
source .venv/bin/activate
```

Or this Powershell command on Windows:

```
py -3.6 -m venv .venv  
.venv\scripts\activate
```

*If you're on Windows and you can't activate your virtual environment (you get the error "The file is not digitally signed. You cannot run this script on the current system"), set your Execution Policy to Remote Signed to be able to run the script:*

*Open a Powershell command line in admin mode and run*

```
Set-ExecutionPolicy RemoteSigned
```

*You should now be able to activate your virtual environment, and you'll see that (.env) precedes every line in your command prompt.*

Inside your virtual environment, **create a new Functions project by running:**

```
func init FunctionProjTest
```

Then select Python as worker runtime.

Read more stories this month when you [create a free Medium account.](#)



Functions app can contain several functions (up to 200 for the regular consumption plan, 20 for premium and app service plans). But remember all of them share the same environment, so if you want to have several functions that don't share anything and use completely different libraries, it might make more sense to create several Functions projects.

To **create a new function**, navigate inside the Functions project folder and run

```
func new
```

Select the HTTP trigger template as it's the type of functions that we will cover in this tutorial, and follow the prompts to name your function. If you have no particular interest in being original you can use the same name as me: *func\_test*

**And... Congrats! You have now created a new Azure function**, and you can already test what the HTTP trigger template does.

Run it locally using the command

```
func host start
```

and make a GET request by either navigating to the corresponding URL in your browser or using Postman.

The URL is provided in the command line when you start your function, and it looks like this: [http://localhost:7071/api/func\\_test](http://localhost:7071/api/func_test)

The template is designed to say 'Hello' to a given name, so to test it properly, append a query parameter, like this: [http://localhost:7071/api/func\\_test?name=Katia](http://localhost:7071/api/func_test?name=Katia)

## 2 Receive and return data

Read more stories this month when you [create a free Medium account](#).



Let's take a look at `__init__.py` (inside your function folder), which was generated automatically as part of the HTTP Trigger template. This file contains the main method of our function, which is executed when the function is triggered.

It looks like this:

```
1 import logging
2
3 import azure.functions as func
4
5
6 def main(req: func.HttpRequest) -> func.HttpResponse:
7     logging.info('Python HTTP trigger function processed a request.')
8
9     name = req.params.get('name')
10    if not name:
11        try:
12            req_body = req.get_json()
13        except ValueError:
14            pass
15        else:
16            name = req_body.get('name')
17
18    if name:
19        return func.HttpResponse(f"Hello {name}!")
20    else:
21        return func.HttpResponse(
22            "Please pass a name on the query string or in the request body",
23            status_code=400
24        )
```

`__init__.py` hosted with ❤ by GitHub

[view raw](#)

As you can see, it tries to get the name from the query parameters, and if it doesn't find it, tries to get it from the request body. Note that it's possible to use `req_body['name']` instead of `req_body.get('name')`.

This means that we can also test sending the function a POST request

Read more stories this month when you [create a free Medium account](#).



```
{  
  "name": "You can use a fake name if you don't like yours"  
}
```

The screenshot shows the Postman interface. At the top, it says 'POST' and 'http://localhost:7071/api/func\_test'. Below that, there are tabs for 'Params', 'Authorization', 'Headers (4)', 'Body (1)', 'Pre-request Script', and 'Tests'. The 'Body' tab is selected, showing a JSON payload: { "name": "Katia" }. A red arrow points from the text 'Careful here!' to the 'JSON (application/json)' dropdown menu. In the bottom right corner, the status bar shows 'Status: 200 OK'.

As expected, we get back the same response as what we got with the GET request.

• • •

So with all that we've seen how to get a request parameters and body, which is already pretty cool.

But let's go a bit further:

What happens if you want to send images or other files to your function?

Well instead of using `req.get_json()` you can use `req.get_body()`.

Let's try:

Read more stories this month when you [create a free Medium account](#).



```

3
4     file_sent = None
5
6     try:
7         file_sent = req.get_body()
8     except ValueError:
9         pass
10
11    if file_sent:
12        return func.HttpResponse(file_sent)
13    else:
14        return func.HttpResponse(
15            "Please pass a file in the request body",
16            status_code=400
17        )

```

`_init_.py` hosted with ❤ by GitHub[view raw](#)

*Note that when you save the code after changes have been made, the function restarts automatically.*

Now create a new txt file with whatever you want in it (if you're not inspired click here) and make a new POST request to your function with the txt file as request body. What you should get back is the text contained in your file:

The screenshot shows the Postman application interface. At the top, there's a header bar with 'POST' selected, the URL 'http://localhost:7071/api/func\_test', and buttons for 'Send' and 'Save'. Below the header are tabs for 'Params', 'Authorization', 'Headers (9)', 'Body (1)', 'Pre-request Script', 'Tests', 'Cookies', 'Code', and 'Comments (0)'. The 'Body' tab is active, showing a file named 'text.txt' attached. Under the 'Body' tab, there are tabs for 'Pretty', 'Raw', 'Preview', 'Text', and a file icon. The 'Text' tab is selected, displaying the contents of the 'text.txt' file. The response status at the bottom right is 'Status: 200 OK Time: 530ms Size: 1.83 KB Save Response'.

```

1  Ut enim ipsum dolor sit amet, consectetur adipiscing elit. Sed dignissim, turpis eu ornare pharetra, massa urna faucibus eros, ut bibendum ipsum dolor in purus.
2  Pellentesque vitae fermentum augue. Nulla viverra nisi sit amet ipsum ultrices gravida a nec mauris. Aliquam hendrerit leo quis ultrices vulputate. Sed commodo,
3  orci at mollis bibendum, diam sem viverra mauris, convallis rutrum est risus at lectus. Suspendisse potenti. Nulla venenatis consectetur lectus, sed imperdiet
mauris iaculis nec. Suspendisse id lectus nec purus ultrices porttitor. Donec vitae interdum ex, non pellentesque ligula. Nunc ac porttitor neque, vitae
interdum libero. Aenean placerat non nulla quis feugiat. Integer ullamcorper felis orci, non laoreet sapien pulvinar eu. Fusce semper, elit id tempus egestas,
magna elit efficitur magna, eu tincidunt justo magna posuere eros. Etiam nec lectus sed lacus egestas aliquet. Nullam sagittis, sapien non maximus dictum, ante
neque auctor nisi, at pellentesque neque augue a leo. Aliquam egestas imperdiet neque.
2
3  Nam risus sem, volutpat in vestibulum eget, placerat sit amet urna. Sed ac diam sit amet urna varius sagittis. Praesent augue ipsum, consequat vitae arcu et,
tincidunt venenatis nibh. Morbi a lacus dolor. Sed rutrum, dui a faucibus eleifend, mi nisi suscipit sapien, ac ullamcorper mi ex et metus. Mauris imperdiet
pretium sapien, in dignissim nisi pellentesque eget. Suspendisse non nisi purus. Praesent commodo leo sollicitudin purus condimentum egestas. Suspendisse
elementum nibh tellus, quis porttitor libero tempus non. Nulla facilisi. Donec efficitur sagittis euismod. Mauris elementum odio eget nibh commodo, a cursus
ligula dignissim. Nulla scelerisque ex quis orci faucibus feugiat.

```

Read more stories this month when you [create a free Medium account](#).

If you try and just pass json as the HTTP response, you'll find yourself with an annoying error:

```
def main(req: func.HttpRequest) -> func.HttpResponse:
    logging.info('Python HTTP trigger function processed a request.')
    file_sent = None

    try:
        file_sent = req.get_body()
    except ValueError:
        pass

    if file_sent:
        return func.HttpResponse({
            "message": "Hello",
            "text": str(file_sent)
        })
    else:
        return func.HttpResponse(
            "Please pass a file in the request body",
            status_code=400
        )
```

[06/08/2019 15:17:37] INFO: Received FunctionInvocationRequest, request ID: ab08d78f-59c7-4d76-a62c-0d7f7bbff070e, function ID: 102425e6-2af9-4177-978d-ccdbf122f193, invocation ID: 86eb5749-3250-41a5-ad5b-f16e07cb89b9  
[06/08/2019 15:17:37] Python HTTP trigger function processed a request.  
[06/08/2019 15:17:37] Executed 'Functions.func\_test' (Failed, Id=86eb5749-3250-41a5-ad5b-f16e07cb89b9)  
[06/08/2019 15:17:37] System.Private.CoreLib: Exception while executing function : Functions.func\_test. System.Private.CoreLib: Result: Failure  
Exception: TypeError: response is expected to be either of str, bytes, or bytearray, got dict  
Stack: File "C:\Users\kagilguz\AppData\Roaming\npm\node\_modules\azure-functions-core-tools\bin\workers\python\deps\azure\functions\_worker\dispatcher.py", line 288, in \_handle\_invocation\_request  
 self.\_run\_sync\_func(invocation\_id, fi.func, args)  
 File "C:\Users\kagilguz\AppData\Local\Programs\Python\Python36\lib\concurrent\futures\thread.py", line 56, in run  
 result = self.fn(\*self.args, \*\*self.kwargs)  
 File "C:\Users\kagilguz\AppData\Roaming\npm\node\_modules\azure-functions-core-tools\bin\workers\python\deps\azure\functions\_worker\dispatcher.py", line 347, in \_run\_sync\_func  
 return func(\*\*params)  
 File "C:\dev\python\functions\functionTest\FunctionProjTest\func\_test\\_\_init\_\_.py", line 19, in main  
 "text": str(file\_sent)  
 File "C:\Users\kagilguz\AppData\Roaming\npm\node\_modules\azure-functions-core-tools\bin\workers\python\deps\azure\functions\\_http.py", line 84, in \_\_init\_\_  
 self.\_set\_body(body)  
 File "C:\Users\kagilguz\AppData\Roaming\npm\node\_modules\azure-functions-core-tools\bin\workers\python\deps\azure\functions\\_http.py", line 114, in \_set\_body  
 f'response is expected to be either of '

The right way to do it is using `json.dumps`, so go ahead and import `json` of your file before updating your code:

```
1  import logging
2  import json
3  import azure.functions as func
4
5  def main(req: func.HttpRequest) -> func.HttpResponse:
6      logging.info('Python HTTP trigger function processed a request.')
7
8      file_sent = None
9
10     try:
11         file_sent = req.get_body()
12     except ValueError:
13         pass
14
15     if file_sent:
16         return func.HttpResponse(
17             json.dumps([
18                 {
19                     "text": str(file_sent),
```

Read more stories this month when you [create a free Medium account](#).

```
23     else:
24         return func.HttpResponse(
25             "Please pass a file in the request body",
26             status_code=400
27     )
```

\_init\_.py hosted with ❤ by GitHub

[view raw](#)

Now that we know how to receive and return data with Python functions, let's dive into the most interesting part: **develop anything you want!**

### 3. Segment your code & Use external libraries

If your code requires using external libraries, you can **install them the same way you would with a regular project**: just run

```
pip install NAME_OF_LIBRARY
```

at the root of your Functions project (not in the folder of the function itself).

For the sake of this tutorial, we're going to install two text/natural language processing libraries: nltk and spaCy, along with a spaCy module.

Go ahead and run (after stopping the execution of your function project):

```
pip install nltk
pip install spacy
python -m spacy download en_core_web_sm
```

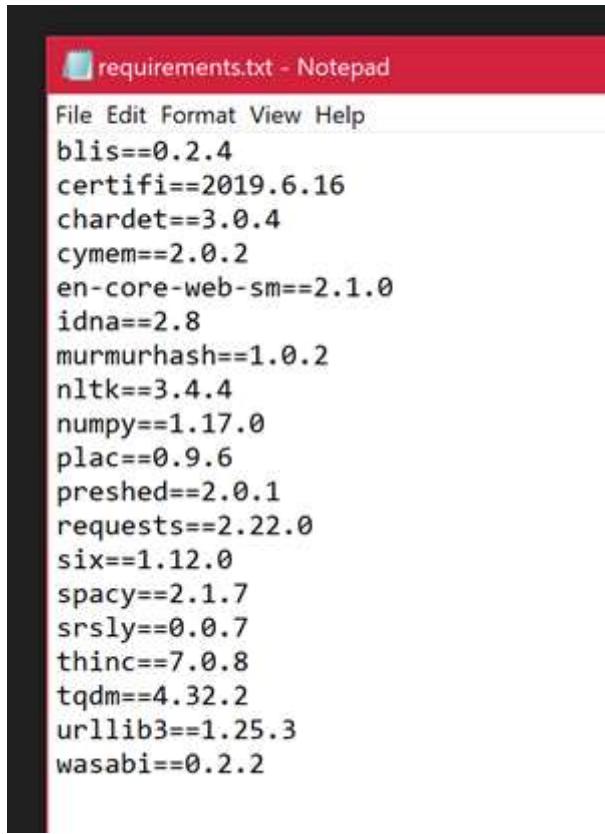
Once you've done that, run

```
pip freeze > requirements.txt
```

Read more stories this month when you [create a free Medium account](#).



If you open it, you'll see that something's wrong:

A screenshot of a Windows Notepad window titled "requirements.txt - Notepad". The window contains a list of Python package dependencies with their specific versions. The text is as follows:

```
File Edit Format View Help
blis==0.2.4
certifi==2019.6.16
chardet==3.0.4
cymem==2.0.2
en-core-web-sm==2.1.0
idna==2.8
murmurhash==1.0.2
nltk==3.4.4
numpy==1.17.0
plac==0.9.6
preshed==2.0.1
requests==2.22.0
six==1.12.0
spacy==2.1.7
srsly==0.0.7
thinc==7.0.8
tqdm==4.32.2
urllib3==1.25.3
wasabi==0.2.2
```

Guessed it? The spaCy module wasn't recorded properly (there's a *en-core-web-sm* something in there but that's not what we want) — you'll need to manually add the module to your requirements:

Just delete the *en-core-web-sm==2.1.0* line and put this URL at the end of *requirements.txt*:

[https://github.com/explosion/spacy-models/releases/download/en\\_core\\_web\\_sm-2.1.0/en\\_core\\_web\\_sm-2.1.0.tar.gz](https://github.com/explosion/spacy-models/releases/download/en_core_web_sm-2.1.0/en_core_web_sm-2.1.0.tar.gz)

The same goes for other libraries you may want to use that do not have a “regular” installation.

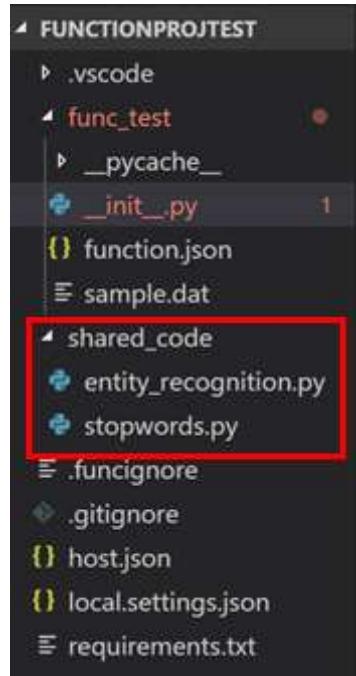
• • •

Read more stories this month when you [create a free Medium account](#).



use SpaCy to find named entities in a text.

Now your project tree should look like this:



I created a new folder at the root of the Functions project so that it could be used by several functions inside the project (even though here I only have one), but you can also create the files and folders you want inside a function folder, it's up to you really.

In `__init__.py`, import the files you just created like this:

```
from ..shared_code import entity_recognition
from ..shared_code import stopwords
```

If your files are in the same folder, use this instead:

```
from . import entity_recognition
from . import stopwords
```

Read more stories this month when you [create a free Medium account](#).



```

2 nltk.download("stopwords")
3 from nltk.corpus import stopwords
4
5 def remove_stop_words(text):
6     STOPLIST = stopwords.words('english')
7     text = ' '.join([word for word in text.split() if word not in STOPLIST])
8     return text

```

stopwords.py hosted with ❤ by GitHub

[view raw](#)

And this in *entity\_recognition.py* to return a text containing named entities recognized in an input text:

```

1 import spacy
2 import en_core_web_sm
3
4 def get_entities(text):
5
6     nlp = en_core_web_sm.load()
7     doc = nlp(text)
8     entities = ""
9
10    for ent in doc.ents:
11        entity = "/ %s - %s /" % (ent.text, ent.label_)
12        entities = entities + entity
13
14    return entities

```

entity\_recognition.py hosted with ❤ by GitHub

[view raw](#)

Now all we need to do is update *\_\_init\_\_.py* to call our new modules:

```

1 import logging
2 import json
3 import azure.functions as func
4 from ..shared_code import entity_recognition
5 from ..shared_code import stopwords
6

```

Read more stories this month when you [create a free Medium account.](#)



```
12
13     try:
14         file_sent = req.get_body()
15     except ValueError:
16         pass
17     else:
18         text = str(file_sent)
19
20     processed_text = stopwords.remove_stop_words(text)
21     entities = entity_recognition.get_entities(text)
22
23     if file_sent:
24         return func.HttpResponse(
25             json.dumps([
26                 "processedText": processed_text,
27                 "entities": entities
28             ]),
29             status_code=200
30         )
31     else:
32         return func.HttpResponse(
33             "Please pass a file in the request body",
34             status_code=400
35         )
```

`_init_.py` hosted with ❤ by GitHub

[view raw](#)

Let's test it (remember to run `func host start` first) — this is the text I used:

*Azure Functions is a solution for easily running small pieces of code, or “functions,” in the cloud. You can write just the code you need for the problem at hand, without worrying about a whole application or the infrastructure to run it. Functions can make development even more productive, and you can use your development language of choice, such as C#, Java, JavaScript, Python, or PHP. Pay only for the time your code runs and trust Azure to scale as needed. Azure Functions lets you develop serverless applications on Microsoft Azure.*

And here's the result :

Read more stories this month when you [create a free Medium account](#).



```

1  {"processedText": "b'Azure Functions solution easily running small pieces code, \"functions,\" cloud. You write code need problem hand, without worrying whole application infrastructure run it. Functions make development even productive, use development language choice, C#, Java, JavaScript, Python, PHP. Pay time code runs trust Azure scale needed. Azure Functions lets develop serverless applications Microsoft Azure.'", "entities": "/ Javascript - ORG // Python - ORG // PHP - ORG // Microsoft Azure - ORG /*]"}

```

As you can see, the stopwords have been removed and our entity recognition code picked up a few named entities.

• • •

Now that you know how to do this — **you can do pretty much anything you would normally do in Python** 😊

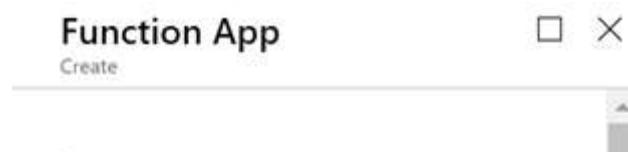
Perfect, so **your function is now ready and running locally**. That's great if you want to keep it to yourself. **But if you want others to be able to access it, you'll want to publish it**. Which is — as you might expect — what we'll learn below.

## 4. Publish

**Publishing your Functions project means an image of your virtual environment will be created and deployed to a Function App in Azure**, and since recently the build is done on the server side, so you don't need to install anything.

There are 2 ways to create a Function App: you can either use the Azure CLI and do it from your command line, or do it from the portal. I'll show how to do it from the portal since the documentation covers the former.

From the Azure portal, click on *Create a Resource* and create a new Function App:



Read more stories this month when you [create a free Medium account](#).

The screenshot shows the Azure portal's configuration page for a new function app. Key settings include:

- Resource Group:** Create new, named "katia-test-function".
- OS:** Linux.
- Publish:** Code (selected).
- Hosting Plan:** Consumption Plan.
- Location:** West Europe.
- Runtime Stack:** Python.
- Storage:** Create new, named "katiatestfuncti8a6a".
- Application Insights:** katia-test-function.

At the bottom, there are "Create" and "Automation options" buttons.

Choose a unique name, select Linux as OS, choose to publish Code, and select Python as the Runtime Stack. For the Hosting Plan and Location it's up to you but if you want your Function App to be cheap, then go for the Consumption Plan.

• • •

Once it's deployed, go back to your command prompt (still in the Functions project folder, and stop the execution of your Functions project). Make sure you're logged into

Read more stories this month when you [create a free Medium account](#).



```
az login
```

and set your subscription with

```
az account set --subscription "SUBSCRIPTION_ID"
```

Now you can either do a local or remote build, but depending on binary dependencies needed in your project, the remote build might fail. In this tutorial, we use a spaCy module which has a different behavior so we'll use the local build approach.

Install Docker and make sure it is running, then run

```
func azure functionapp publish APP_NAME --build-native-deps
```

(APP\_NAME being the name of the Function App you just created on Azure)

You should see something like this (don't be surprised if it takes a while):

```
(venv) PS C:\dev\python\functions\testGA\FunctionProjTest> func azure functionapp publish katia-test-function --build-native-deps
Getting site publishing info...
Deleting the old .python_packages directory
Running 'docker pull mcr.microsoft.com/azure-functions/python:2.0.12493+python3.6-buildenv'...done
Running 'docker exec -t 5f52cf chmod +x ./python_docker_build.sh'..done
Running 'docker exec -t 5f52cf /python_docker_build.sh'.....done
Running 'docker cp 5f52cf:"./.python_packages/" "C:\dev\python\functions\testGA\FunctionProjTest\.python_packages"'.....done
C:\dev\python\functions\testGA\FunctionProjTest>
```

If it fails for no apparent reason, increase the memory allocated to Docker (Right click on Docker icon > Settings > Advanced).

If for other projects you only use standard libraries, you can build remotely by running:

```
func azure functionapp publish APP_NAME --build remote
```

Read more stories this month when you [create a free Medium account](#).



```

Running 'docker run -rm f2aff4 mkdir -p ./home/site/wwwroot'...done
Running 'docker exec -t f2aff4 cp ./app.zip ./home/site/wwwroot'...done
Running 'docker cp "C:\Users\kagilguz\AppData\Local\Temp\f2ahl34.xi4\_" f2aff4:"/home/site/wwwroot"'...done
Running 'docker cp "C:\Users\kagilguz\AppData\Local\Temp\tmpDC7C.tmp" f2aff4:"/worker_packages.txt"'...done
Running 'docker cp "C:\Users\kagilguz\AppData\Local\Temp\tmpDC58.tmp" f2aff4:"/python_docker_build.sh"'...done
Running 'docker exec -t f2aff4 chmod +x /python_docker_build.sh'...done
Running 'docker exec -t f2aff4 chmod +x /python_bundle_script.py'...done
Running 'docker exec -t f2aff4 /python_docker_build.sh'...done
-----
Running 'docker cp f2aff4:/app.zip C:\Users\kagilguz\AppData\Local\Temp\hrbsz1pl.tdk'...done
running 'docker kill f2aff4'...done
Preparing archive...
uploading 118,25 MB [=====] (please wait a little in the request body)
upload completed successfully.
deployment completed successfully.
Syncing triggers...
Functions in katia-test-function:
  func_test - [httpTrigger]
    Invoke url: https://katia-test-function.azurewebsites.net/api/func_test?code=EcRHCK0jaJYw3BXdPFL79rxw4gdFu1msk6g3C1YfNaNle6ej8Pg==

(.venv) PS C:\dev\python\functions\functionTest\FunctionProjTest> .

```

We are (at least I am — if it didn't work for you, read through the steps again and make sure you followed them properly) almost done!

• • •

To make sure everything works as expected, open Postman again and just replace the endpoint with the invoke URL provided after deployment (with the code parameter, otherwise you'll get an *Unauthorized Access* error).

The screenshot shows a Postman interface with the following details:

- Method:** POST
- URL:** https://katia-test-function.azurewebsites.net/api/func\_test?code=EcRHCK0jaJYw3BXdPFL79rxw4gdFu1msk6g3C1YfNaNle6ej8Pg==
- Body:** (Binary file named text.txt attached)
- Response Status:** 200 OK
- Response Body:** (Pretty printed JSON)

```

1  [{"processedText": "b'Azure Functions solution easily running small pieces code, \"functions,\" cloud. You write code need problem hand, without worrying whole application infrastructure run it. Functions make development even productive, use development language choice, C#, Java, JavaScript, Python, PHP. Pay time code runs trust Azure scale needed. Azure Functions lets develop serverless applications Microsoft Azure.'", "entities": "/ JavaScript - ORG // Python - ORG // PHP - ORG // Microsoft Azure - ORG /"}]

```

**Yay! We're now ready to use this Python function anywhere!**

Here's an example of how to use this in a JS web app:

```

1   fetch(functionURL, {
2     method: 'POST',
3     ...

```

Read more stories this month when you [create a free Medium account](#).

```
8     "filename":filename
9   })
10  })
11 .then(res => res.json())
12 .then((data) => {
13   console.log(data);
14 })
15 .catch((error) => {
16   console.error(error);
17 });
});
```

fetch\_function.js hosted with ❤ by GitHub

[view raw](#)

Hope you find Azure functions as useful as I do 😊

## Resources

[Azure Functions Introduction](#)

[Create an HTTP Triggered Azure Python Function Quickstart](#)

[Finished project on GitHub](#)

[Python](#)

[Serverless](#)

[Functions As A Service](#)

[Azure Functions](#)

[Python Functions](#)

[About](#) [Help](#) [Legal](#)

[Get the Medium app](#)



Read more stories this month when you [create a free Medium account](#).

