FaaSpot

Release

August 02, 2017

Contents

FaaSpot is a SaaS service that enables you to run Functions as a Service (FaaS), and reduce your monthly Cloud expenses.

- You need us if you are using an expensive, powerful machine to run a small set of tasks multiple times.
- With FaaSpot you can replace the expensive machine with a much smaller one, and use FaaSpot to run your functions.
- The cost of FaaSpot is predetermined. You know how much money you'll pay at the end of the month.
- With FaaSpot you'll get scale out-of-the-box for less, much less, money.

```
$ fas spots add --wait
$ fas functions samples --hello-world
$ fas functions create hello --file hello-world.py
$ fas functions run hello --parameters "name=user1" --wait
```

CHAPTER 1

User Guide

Overview

What Is FaaSpot?

FaaSpot is a SaaS service that enables you to run Functions as a Service (FaaS), and reduce your monthly Cloud expenses. The concept provides the ability for you to run your functions on a FaaSpot spot. A FaaSpot spot is a VM that can run your functions. You can run multiple concurrent functions on one spot. Nevertheless, the more spots that you have, the more concurrent functions you'll be able to run. The cool thing about FaaSpot, is that you pay for each spot that you have, regardless of the number of functions that you run on it, and it actually uses AWS spots to reduce costs.

Requirements

1. You need a FaaSpot token. If you haven't already obtained a FaaSpot API key, send us a request at info@faaspot.com.
2. If you're using the python-client, python-client version compatibility is: Python 2.7, 3.3-3.5.

Note: Communication with the `FaaSpot` webserver is encrypted and secured.

You can access FaaSpot using the FaaSpot client from the VM, directly from the Python script (as a library), or by using the proprietary REST API. The FaaSpot's command-line interface is the default method for interacting with FaaSpot and managing your functions.

Limitations

1. FaaSpot currently supports only python scripts.
2. The python script needs a main function with a specific format.
3. The function run time must be less than a minute.

Installation

You can install the FaaSpot Client via pip:

```
$ pip install faaspot
```

Note: After installing the FaaSpot client you'll have both the command-line interface (CLI) and a python-client library. The CLI client name is `fas` and the python-client library name is `faaspot`.

To upgrade the FaaSpot client version, run: `pip install faaspot --upgrade`.

For info about the CLI, go to the *CLI* page. For info about the python library, go to the *functions* page.

Setting Up The Environment

To use the FaaSpot client, you need to configure your profile. This configuration will apply to both the CLI and the python-client library. Create a FaaSpot profile that contains your FaaSpot token credentials.

```
$ fas profiles create --token MY_API_TOKEN
```

Note: The `fas profiles create` command will create a global configuration file located at ~/.faaspot folder, which contain the connection configuration to FaaSpot.

You can also manually edit the `~/.faaspot/conf.yaml` file.

Note: `fas` supports tab completion in bash shells via the argcomplete package.

To enable it, add

```
eval "$(register-python-argcomplete fas)"
```

to the `~/.basrc` file. You can add it manually or by running

```
$ echo -e '\neval "$(register-python-argcomplete fas)"\n' >> ~/.bashrc
```

and then run `source ~/.bashrc`. You can test that it works by running

```
$ fas sp<TAB>
```

It should complete to

```
$ fas spots
```

The Basics

This page provides a quick introduction to FaaSpot.

FaaSpot Spots

A FaaSpot spot is a VM that can run your functions. This VM is totally managed by FaaSpot. You can add or remove spots using the FaaSpot API. For more info, please read the *Spots* page.

Function

A FaaSpot function is a package of function to run, with optional meta-data of secrets that are needed for the function run. For more info, please read the *functions* page.

Making a Request

FaaSpot exposes a set of REST APIs. You can call the APIs manually, using cURL for example, or you can use the FaaSpot client, which wraps the HTTP calls with a user friendly interface.

There are multiple ways to send requests to FaaSpot:

- **Command Line Interface (CLI)**. For usage info, please read the *CLI* page.
- **python-client**. For usage info, please read the *python client* page.
- **HTTP Requests**. You can manually send HTTP requests to the REST API. We'll cover the usage in the API sections in the documentation.

Async Requests

Some of the requests are executed as non-blocking. When you run a non-blocking request, the result will be an execution ID. In that case, you'll need to run another request to retrieve that execution status, in order to see if it's still in progress, or completed.

You can send blocking requests that will block the client until there is a response. In the CLI, you can use the `--wait` parameter.

Concurrent Requests

You can send multiple requests concurrently, and send a single bulk request. This is a more advanced topic.

Quickstart

This page provides quick introductory examples for using FaaSpot. If you have not already installed FaaSpot, head over to the *Installation* section.

Get A Spot

The first thing that you need to run functions on FaaSpot is a FaaSpot spot.

You can easily add, remove, and get a list of your spots using the CLI:

```
$ fas spots add --wait
$ fas spots list
$ fas spots remove --wait
```

To see how to add spots using an HTTP request or using the python-client, go to the *spots* page.

Get A Function

To make life easier, we have some built-in samples that come with the `fas` CLI. The samples are very simple python scripts, which can run on FaaSpot as a quick test, or be used as a reference. The samples are:

- **hello-world function**. The hello-world function receives a name as input, and replies with hello.
- **get-content function**. The get-content function receives a URL as input, downloads it's content, and returns it.
- **sleep function**. The sleep function receives optional input that indicates the sleep time, and then returns a string.

All the samples are available using the `fas functions samples` command:

```
$ fas functions samples -h
usage: fas functions samples [-h] [--hello-world | --get_content | --sleep]
                             [-v]

optional arguments:
  --hello-world  Generates a hello-world function
  --get-content  Generates a get-content function
  --sleep        Generates a sleep function
  -h, --help     Shows this help message and exits
  -v, --verbose  Increases output verbosity. -v will print debug messages. -vv
                 will additionally print 3rd-party info
```

The following command will create a *hello-world.py* script, which we will use later on and create a FaaSpot Function from it:

```
$ fas functions samples --hello-world
```

Create A Function

If you don't already have a *FaaSpot spot*, you need to *get one*, in order to run the samples.

Now you need to create a *function*. To create a function, you just need a python script that contains a function to run. If you run the command from the section above, you already have a sample hello-world python script.

To create a function from the python script, you need to use the `functions create` API:

```
$ fas functions create hello --file hello-world.py
```

This command creates a new function, named hello, which contain the hello_world.py file. To see how to create a new function using an HTTP request or using the python-client, go to the *function* page.

Run The Function

Now that you have a spot and a function, you're ready to run the function. You can run the function using the CLI:

```
$ fas functions run hello --parameters "name=user1" --wait
```

We used the *–wait* parameters, so the command will wait until the function completes, and will return the function result, and not the execution ID.

To see how to run the function using HTTP request or using the python-client, go to the *run function* section.

Command Line Interface (CLI)

Overview

The FaaSpot command-line interface (CLI) is the default method for interacting with FaaSpot, to manage your functions.

For installation instructions, please refer to the *Installation* section.

Usage

You can access the CLI by running the `fas` command in your terminal. Use `cfy -h` to display a list of all the commands and their descriptions.

```
$ fas -h
usage: fas [-h] [--version] {spots,functions,executions,profiles} ...

positional arguments:
  {spots,functions,executions,profiles}
                        Manages the FaaSpot account
    spots               Manages spots
    functions           Manages functions
    executions          Manages executions
    profiles            Manages profiles

optional arguments:
  -h, --help            Shows this help message and exits
  --version             Show the version number and exits
```

Verbosity Level

Any command in the CLI has the option to show you the command logs. You find this option in the help message for the command, for example: `fas spots list -h`

The verbosity levels are:

- **-v** Print debug messages.
- **-vv** Additionally print 3rd-party info log messages (of utils that are being used by `fas`).
- **-vvv** Additionally print 3rd-party debug log messages.

Note: To directly use HTTP requests, you can run a command using `-vvv`. It will show you the HTTP requests that have been used.

FaaSpot Python-Client

Overview

The python client uses the same config as the CLI, which means that you need to set your config only one time. You can set the config using the CLI or the python client.

For installation instructions, please refer to the *Installation* section.

Spots

Overview

A FaaSpot spot is a VM that can run your functions. You can run multiple concurrent functions on one spot. Nevertheless, the more spots that you have, the more concurrent functions you'll be able to run.

The cool thing about FaaSpot, is that you pay for each spot that you have, regardless to the number of functions that you run on it. Having said that, to run a function in FaaSpot you will need at least one FaaSpot spot.

Add a Spot

To add a spot to your spot's pool, you need to use the spots API:

CLI

You can manage your spots using the `fas` CLI:

```
$ fas spots add --wait
```

Python

You can manage your spots using the `fas` python-client:

```
from faaspot import Faaspot
faaspot = Faaspot()
faaspot.spots.add(wait=True)
```

HTTP Request

You can manage your spots using the direct HTTP requests:

```
$ curl -X PUT --header "Authorization: Token MY_TOKEN" https://api.faaspot.com:443/v1/spots/
```

Note: What is the `wait` argument?

By default, requests run in the background, in a synchronized manner. This means that the request will return the execution ID. You can then check that execution status (completed or not), using the execution ID.

When using the FaaSpot client, you can run the command in a synced manner (wait until you receive a response), using the `wait` argument.

When using the HTTP request to add a spot, you'll need to check the execution status manually. To see how to do it, go to the *executions* page.

List Spots

To retrieve a list of all the spots that you have, use the spots list API:

CLI

You can manage your spots using the `fas` CLI:

```
$ fas spots list
```

Python

You can manage your spots using the `fas` python-client:

```
from faaspot import Faaspot
faaspot = Faaspot()
faaspot.spots.list()
```

HTTP Request

You can manage your spots using the direct HTTP requests:

```
$ curl -X GET --header "Authorization: Token MY_TOKEN" https://api.faaspot.com:443/v1/spots/
```

Remove a Spot

To remove a spot from your spot's pool, you need to use the spots API:

CLI

You can remove one spot from your spots pool using the CLI:

```
$ fas spots remove --wait
```

Python

You can remove one spot from your spots pool using the python-client:

```
from faaspot import Faaspot
faaspot = Faaspot()
faaspot.spots.remove(wait=True)
```

HTTP Request

You can remove one spot from your spots pool using a direct HTTP request:

```
$ curl -X DELETE --header "Authorization: Token MY_TOKEN" https://api.faaspot.com/v1/spots/
```

This API will return the execution ID of the spot removal task. To get the execution status of that task, you will need to query the execution status. You can see how to do it in the *executions* page.

Note: What is the `wait` argument?

By default, requests run in the background, in a synchronized manner. This means that the request will return the execution ID. You can then check that execution status (completed or not), using the execution ID.

When using the FaaSpot client, you can run the command in a synced manner (wait until you receive a response), using the `wait` argument.

When using the HTTP request to remove a spot, you'll need to check the execution status manually. To see how to do it, go to the *executions* page.

Refresh a Spot IP Address

The spots are actual VMs, with a public IP address. Sometimes, there is a need to give the spots a new IP, not a specific IP, just a different one. You can do it using a FaaSpot refresh_ip request.

CLI

You can refresh the IP address of your spots, using the CLI:

```
$ fas spots refresh_ip --wait
```

The refresh_ip command parameters:

- (Optional) **- -ip** Specifies which spot IP to refresh. Default is to refresh all spots IPs.
- (Optional) **- -wait** Boolean parameter, whether to wait for completion. Default is False.

Python

You can refresh the IP address of your spots, using the python-client:

```
from faaspot import Faaspot
faaspot = Faaspot()
faaspot.spots.refresh_ip(wait=True)
```

The refresh_ip command parameters:

- (Optional) **ip** Specifies which spot IP to refresh. Default is to refresh all spots IPs.
- (Optional) **wait** Boolean parameter, whether to wait for completion. Default is False.

HTTP Request

You can refresh the IP address of your spots, using direct HTTP requests:

```
$ curl -X PATCH --header "Authorization: Token MY_TOKEN" https://api.faaspot.com/v1/spots/ \
--data '{"refresh_ip": "all"}'
$ curl -X PATCH --header "Authorization: Token MY_TOKEN" https://api.faaspot.com/v1/spots/ \
--data '{"refresh_ip": "SPOT_IP_TO_REFRESH"}'
```

This API will return the execution ID of the spot refresh_ip task. To get the execution status of that task, you will need to query the execution status. You can see how to do it in the *executions* page.

functions

Overview

In FaaSpot, function is a combination of:

- The **function code** that you want to run.
- (Optional) A **Context file**, which might contain secrets credentials that you need to be accessible in the function, but don't want to send in every functions run command.

The Function

The function is basically a python script that contains a main function in a specific format. If the python script doesn't have a main function in the required format, it will not run.

The script must contain a `main` function, that receives 2 parameters:

- **args** A dictionary that contains the arguments to the function. These are the parameters that are sent to the function to run command.
- **context** A dictionary that contains the Context variables, which you set in the function creation command.

The very basic template of the python script:

```
def main(args, context):
    return ''
```

The Context

The Context is a dictionary that might contain secret data to be sent to all of the specific function instances. You can create a function without it, in which case the context dict will be empty.

A very basic Context file might look like this:

```
access_key_is = "XYZ"
secret_access_key = "123"
```

Create function

To create a new function, you need to use the `functions create` API. You can see info about the API arguments in the *overview* section.

CLI

You can create a new function using the `CLI`, for example:

```
$ fas functions create FUNCTION_NAME --file FILE_PATH
```

The create command parameters:

- **–file** - Path to he file code
- (Optional) **–context-file** Path to the context file
- (Optional) **–wait** Boolean parameter. Whether to wait for completion. Default is False.

Python

You can create a new function using the `fas` python-client, for example:

```
from faaspot import Faaspot
faaspot = Faaspot()
faaspot.functions.create(FUNCTION_NAME, file=FILE_PATH)
```

The create function parameters:

- **–file** - Path to he file code
- (Optional) **context-file** Path to the Context file
- (Optional) **wait** Boolean parameter. Whether to wait for completion. Default is False.

HTTP Request

You can create a new function using direct HTTP requests:

```
$ curl -X PUT --header "Authorization: Token MY_TOKEN" --header "Content-Type: application/json"
--data '{"name": "FUNCTION_NAME", "code": "THE_CODE", "context": "THE_CONTEXT"}'
https://api.faaspot.com/v1/functions/
```

The code argument is mandatory, the context is optional. You must provide the actual code and context in UTF-8 format. To encode your text to UTF-8 format, you can use this on-line converter, or use python:

```
from six.moves.urllib.parse import quote
encoded_str = quote(str_to_encode.encode("utf-8"))
```

Run a function

To run a function, you need to use the `functions run` API.

CLI

You can run a function using the `CLI`, for example:

```
$ fas functions run hello --parameters "name=user1" --wait
```

The functions run command parameters:

- **name** The name of the function to run
- (Optional) **–wait** Boolean parameter. Whether to wait for completion. Default is False.

To run a function with multiple parameters, add them in the parameter argument, for example:

```
$ fas functions run FUNCTION_NAME -p "param_1=value_1, param_2=value_2" --wait
```

As you can see in the example above, you can use `-p` as a shortcut for `--parameters`

In this example, we run the `fas functions run` command in a blocked manner (wait until you have a response), using the `--wait` parameter.

By default, without `--wait`, the command will run in a non-blocking manner, and the `fas functions run` command will return the execution ID of the task. You can then check that status of the execution task (completed or not), using the `fas executions get` command. You can read about the execution API in the *execution* page.

For example:

```
$ UUID=`fas functions run hello --parameters "name=user1"`
$ fas executions get $UUID
```

Python

You can run a function in a blocking way (wait until the execution is completed), using the python-client:

```
from faaspot import Faaspot
faaspot = Faaspot()
faaspot.functions.run(FUNCTION_NAME, {'PARAMETER_1': 'VALUE_1', 'PARAMETER_2': 'VALUE_2'}, wait=True)
```

The functions run function parameters:

- **name** The name of the function to run
- (Optional) **wait** Boolean parameter. Whether to wait for completion. Default is False.

To run the function in non-blocking way, and then check the execution status of the function, you can run:

```
from faaspot import Faaspot
faaspot = Faaspot()
execution_id = faaspot.functions.run('FUNCTION_NAME', {'PARAMETER': 'VALUE'})
execution = faaspot.executions().get(execution_id)
print execution['status']
```

HTTP Request

You can run a function using direct HTTP requests. You can run function in a blocking way (wait until the execution is completed), with `/sync/` in the url.

```
$ curl -X GET --header "Authorization: Token MY_API_TOKEN"
https://api.faaspot.com/v1/sync/functions/FUNCTION_NAME/rpc/?PARAMETER_1=VALUE_1&PARAMETER_2=VALUE_2
```

You can also run the function in non-blocking way, without `/sync/` in the url:

```
$ EXECUTION_ID_STR=`curl --header "Authorization: Token MY_API_TOKEN" https://api.faaspot.com/v1/func
$ EXECUTION_ID=`sed -e 's/^"//' -e 's/"$//' <<< "$EXECUTION_ID_STR"`
$ curl --header "Authorization: Token MY_API_TOKEN" https://api.faaspot.com/v1/executions/$EXECUTION_
```

In the above sample you can see how to run a function using an HTTP Request, and then how query the execution status of the function run task.

For the non-blocking approach, you can also use a POST request, to enable you to send the parameters in the request body, instead of in the request URL. For example:

```
$ curl -X POST --header "Content-Type: application/json" --header "Authorization: Token MY_API_TOKEN"
https://api.faaspot.com:443/v1/functions/hello/rpc/ -d '{"PARAMETER_1": "VALUE_1", "PARAMETER_2": "VA
```

Run functions In Bulk

Sometimes, you want to run the same function with different arguments. One way to do it, is to run the *run function* multiple times, each time with different arguments.

A faster way, is to use a single request, with the data of all the different arguments. The way to do that, is to use the `functions run_bulk` request. The `run_bulk` request requires a list of group-of-parameters, meaning that every item in the input list represents a call to the‘‘functions run‘‘ request, with a group-of-parameters.

CLI

You can run a bulk function using the `CLI`, for example:

```
$ fas functions run_bulk FUNCTION_NAME -p "k1=v1, k2=v2" -p "k3=v3, k4=v4"
```

The sample above, will execute two `FUNCTION_NAME` tasks. One with the arguments k1=v1, k2=v2, and another with the arguments k3=v3, k4=v4. The result of the sample above will be a list of two execution IDs.

Python

You can run a bulk function using the python-client:

```
from faaspot import Faaspot
faaspot = Faaspot()
args_list = [{'k1': 'v1', 'k2': 'v2'}, {'k3': 'v3', 'k4': 'v4'}]
id_list = faaspot.functions.run_bulk(FUNCTION_NAME, args_list)
```

HTTP Request

If you want to create a bulk run request using an HTTP request, you will need to create a POST request to: https://api.faaspot.com/v1/functions/FUNCTION_NAME/bulk_rpc/, and to add to the request body the list of the parameters, in the following format: `'[{"k1": "v1", "k2": "v2"}, {"k3": "v3", "k4": "v4"}]'`

```
$ curl -X POST --header "Content-Type: application/json" --header "Authorization: Token MY_API_TOKEN"
https://api.faaspot.com:443/v1/functions/FUNCTION_NAME/bulk_rpc/ -d '[{"k1": "v1", "k2": "v2"}, {"k3"
```

The result of the above request is a list of executions IDs of all the related function executions.

Note: The `bulk_run` call doesn't support blocking requests. The response is a list of executions IDs. To get the executions status, you need to run the *executions get* command.

Delete functions

To delete a new function, you need to use the `functions delete` API.

CLI

You can delete a function using the `CLI`, for example:

```
$ fas functions delete FUNCTION_NAME
```

Python

You can delete a function using the python-client:

```
from faaspot import Faaspot
faaspot = Faaspot()
faaspot.functions.delete(FUNCTION_NAME)
```

HTTP Request

You can delete a function using an HTTP request:

```
$ curl -X DELETE --header "Authorization: Token MY_API_TOKEN" https://api.faaspot.com:443/v1/functio
```

Executions

Overview

When you run some command in a non-blocking way, for example: *function run*, the command will create an execution that will run in the background, and the result of the API will be the execution ID. You can then use the executions API to retrieve the execution status, or to see which executions are currently running.

Get Execution Status

To get the status of an execution, you need to use the spots API:

CLI

You can get the status of an execution using the CLI:

```
$ fas executions get EXECUTION_ID
```

Python

You can get the status of an execution using the python-client:

```
from faaspot import Faaspot
faaspot = Faaspot()
faaspot.executions.get(EXECUTION_ID)
```

HTTP Request

You can get the status of an execution using direct HTTP requests:

```
$ curl -X GET --header "Authorization: Token MY_TOKEN" https://api.faaspot.com/v1/executions/EXECUTIO
```

A result of the API should look as follows:

```
{ u'created': u'2017-07-21 15:10:52.537686+00:00',
    u'name': u'tasks.create_spot',
    u'output': None,
    u'status': u'Started',
    u'uuid': u'9cfcd860-9a5d-414f-8d2b-317d59d3f486'
}
```

Note: The CLI and the python-client support the `wait` argument, meaning that using `fas executions get EXECUTION_ID --wait` or using `faaspot.executions.get(EXECUTION_ID, wait=True)`, will wait until the execution is completed.

Get Bulk Execution Status

Sometimes, you have multiple executions running simultaneously. In that case, you might want to query the status of all of them using a single request, instead of generating an `execution get` request for each of the exceutions separately. You can achieve that using the `execution get_bulk` command.

The results will be a list of execution-statuses.

CLI

You can retrieve the statuses of multiple executions using the CLI:

```
$ fas executions get_bulk -u EXECUTION_ID_1 -u EXECUTION_ID_2
```

Python

You can retrieve the statuses of multiple executions using the python-client:

```
from faaspot import Faaspot
faaspot = Faaspot()
faaspot.executions.get([EXECUTION_ID_1, EXECUTION_ID_1])
```

HTTP Request

You can retrieve the statuses of multiple executions using direct HTTP requests:

```
$ curl -X POST --header "Content-Type: application/json" --header "Authorization: Token MY_API_TOKEN"
https://api.faaspot.com:443/v1/executions/bulk/ -d '["EXECUTION_ID_1", "EXECUTION_ID_2"]'
```

Get Executions List

To retrieve a list of the current running executions.

CLI

You can retrieve a list of the current running executions using the CLI:

```
$ fas executions list
```

The executions status command parameters:

- (Optional) **–include_completed** Boolean parameter. Whether to include completed executions. Default is False.

Python

You can retrieve a list of the current running executions using the python-client:

```
from faaspot import Faaspot
faaspot = Faaspot()
faaspot.executions.list()
```

The executions status command parameters:

- (Optional) **include_completed** Boolean parameter. Whether to include completed executions. Default is False.

HTTP Request

You can retrieve a list of the current running executions using direct HTTP requests:

```
$ curl -X GET --header "Authorization: Token MY_TOKEN" https://api.faaspot.com/v1/executions/?include
```

You can add `?include_completed=False` or `?include_completed=True` to the request, to include completed executions, or not.