ScriptPilot

Usage & Maintenance Manual

18 October 2021

Version 1.0

CONTENTS

OVERVIEW

ScriptPilot is a zero-running-costs* web application acting as a standalone platform to power user-defined scripts. The scripts can be invoked locally or through external third-party services in order to perform computational work as defined by the script author. For the backend, ScriptPilot relies on a service called AWS Lambda, which is part of AWS (Amazon Web Services) to store and execute scripts. This means that nearly all AWS Lambda concepts apply to ScriptPilot. Any scripts and changes made in ScriptPilot will be reflected in the AWS Lambda account linked to ScriptPilot. Overall, ScriptPilot provides the following key functionality on top of what AWS Lambda provides:

1. User-friendly UI to manage scripts. The AWS Lambda UI contains much more complex functionality, which is stripped down to the basics. Both may be used in tandem depending on a developer's comfort level with either.
2. API Key generation. ScriptPilot is capable of randomly generating and storing encrypted API keys for each script and its particular versions. API keys can then be used for authentication when invoking a script externally.

Scripts operate in a serverless manner; meaning their code is only being executed by a computer in the cloud (hosted by AWS) whenever invoked.

*Under reasonably high usage levels estimated for a single small-sized organisation.

USAGE

Authentication

Users arriving at https://www.scriptpilot.codebycarlos.co.uk/ are greeted by a login page along with login buttons. There is no sign up required. Users can log in directly with either a LinkedIn account or an email.

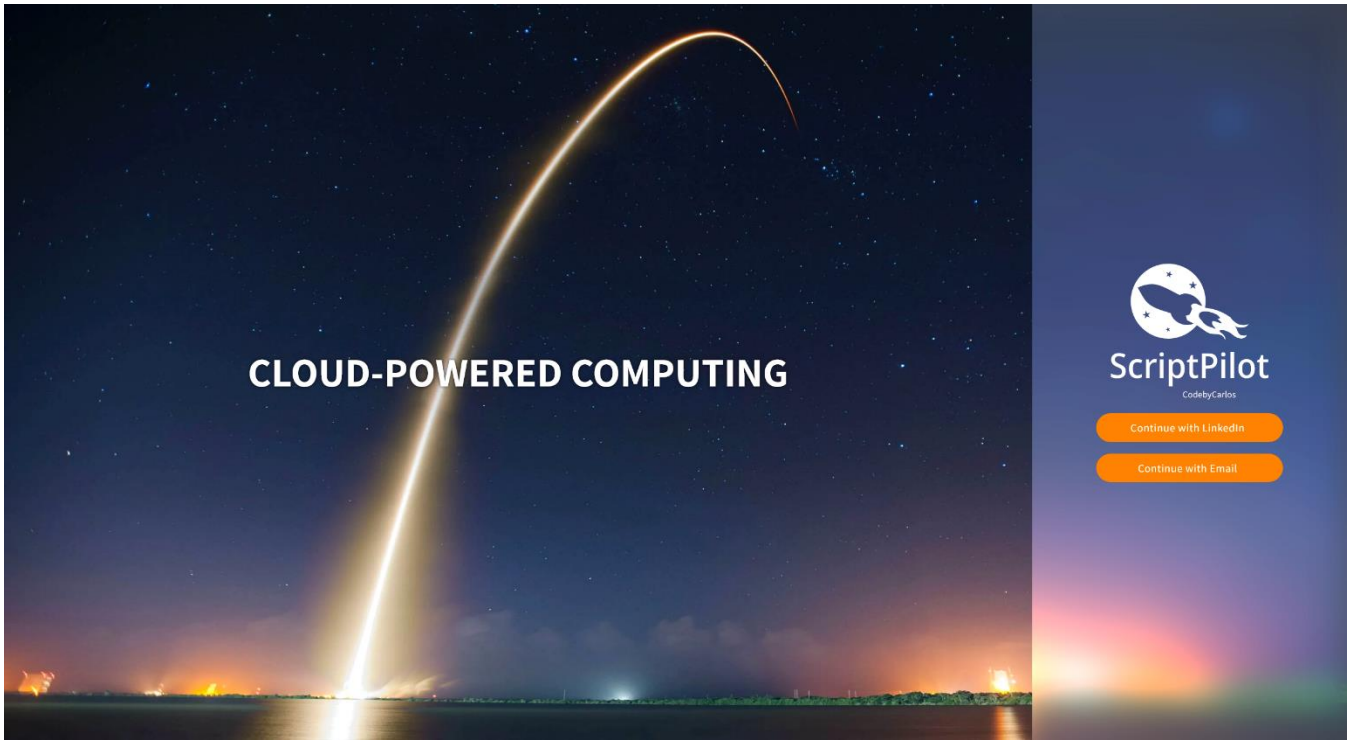

Figure 1 Login page

View Scripts

Upon logging in, users are presented with a paginated list of all scripts. Hovering over the column names will bring up three dots – this is a context menu that allows for hiding columns, sorting, and search functionality. The same can be done for each script in order to interact with them, with the possibility to invoke them from this page.

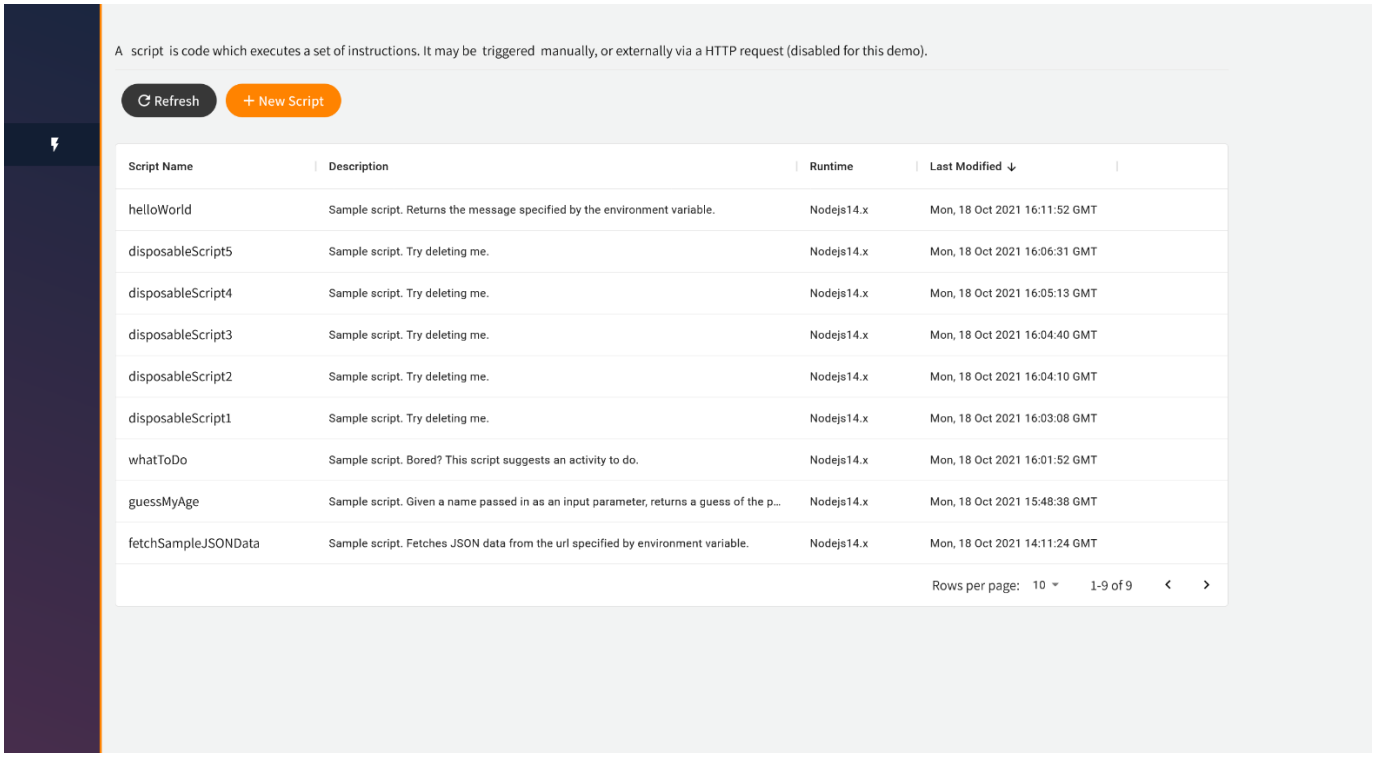

Figure 2 List of scripts

Add Script

To start a script, click the New Script button after logging in.

Figure 3 New Script button

Adding a new script involves three parts: Configuration, Environment Variables, and Code.

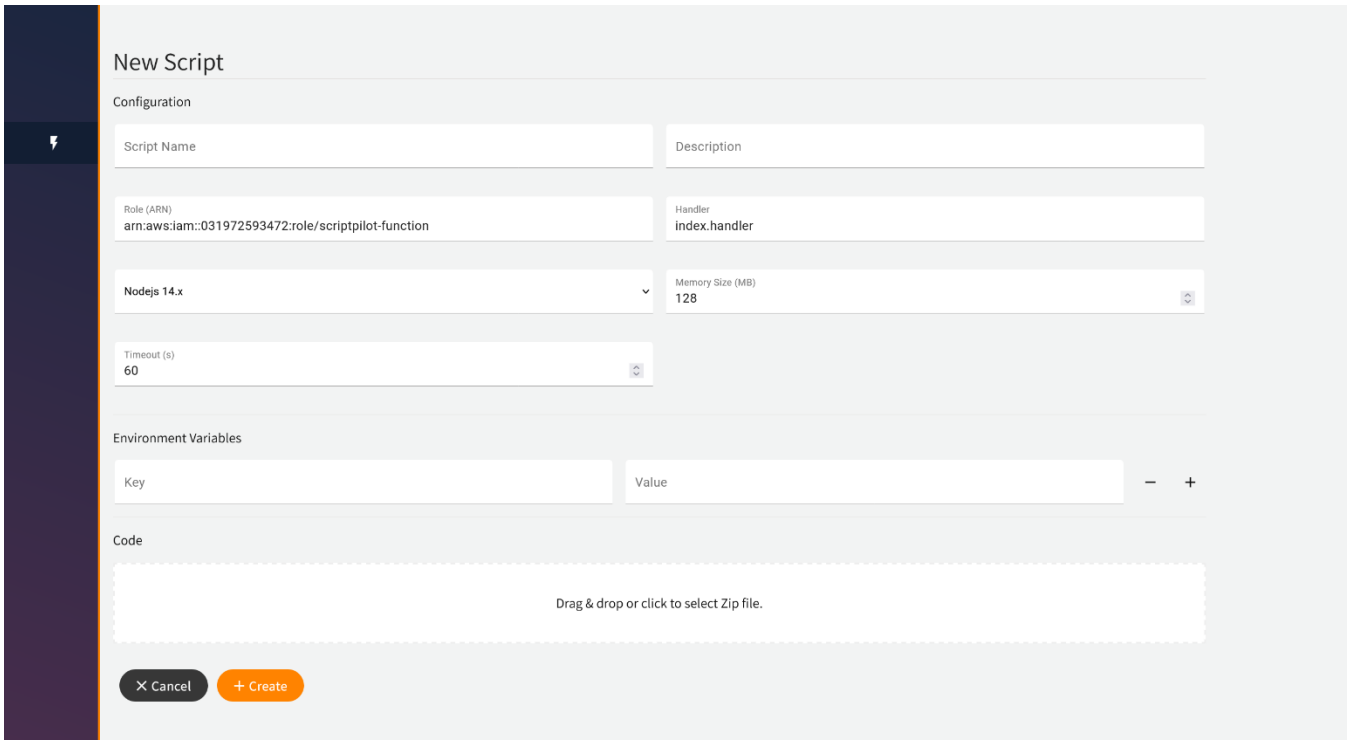

Figure 4 New Script form

Configuration: these are key defining attributes of a script.

- Name: must be unique.
- Description: a brief description of what the script does.
- Role (ARN): automatically filled in with the default role (should be sufficient for all cases unless incorporating additional AWS services in the future). This controls what permissions the script has within the AWS ecosystem (i.e. being able to invoke other scripts).
- Handler: this is the starting point of the script. When uploaded, a script's folder may contain numerous files breaking up code (this is good), this just tells the computer where to start. The format is *filename.functionName*. In the case of Node.js scripts, for example, this means the code folder should contain a file named index.js in the main directory, along with an exported function inside of it named handler.
- Runtime: scripts may be written in a variety of different languages and environment types.

- Memory Size: how much RAM (Random Access Memory) is temporarily allocated towards invoking a script, up to 10 GB. The more memory that is allocated, the more powerful the CPU being used. For regular scripts, 128 MB should be more than enough. More memory intensive scripts may benefit from a higher allocation of memory (i.e. 256 MB or 1024 MB), resulting in faster execution times (testing at different memory sizes is recommended). Any memory increase that does not result in a considerable increase in performance should be avoided; higher memory allocation uses up more of the computational resources provided by the AWS Lambda free-tier (see Usage > Notable Limitations).
- Timeout: should the script run for any longer than the specified timeframe in seconds, the invocation will be halted. Note that this does not override any of the limitations listed (see Usage > Notable Limitations).

Environment Variables: these are key-value pairs of data that are directly and globally accessible as variables within a script's code (in the case of Node.js, these would be accessed by executing *process.env.VARIABLE_NAME*).

Code: this should be a zipped file containing all of the code (including any module/package dependencies) necessary for the script to run. This code is directly uploaded to AWS through ScriptPilot for storage and there is a limitation to that storage size as well as the script's file size (see Usage > Notable Limitations). Code bundlers or minimizers such as Webpack may be used to reduce the file size by up to 99% and even provide a minor performance boost as there is less code the computer needs to load. The name chosen for the zipped folder does not matter. Ensure only the contents of the script are zipped in order to maintain the file structure (i.e. the folder containing the script files should not be zipped as this adds an additional directory level).

Upon successfully adding a script, it will show up in the scripts' list.

Add Script Version

Every script has a version referred to as $LATEST. This version of the script can have its details edited at any time. It is recommended this version be reserved only for testing purposes. When deploying scripts in production, a separate version should be published; these have their configuration, code, and environment variables locked. This means that another service relying on the script won't break if a new incompatible version is released. Each version represents a copy of that script frozen in time. An API key can be generated for each version individually (including the $LATEST version).

To add a version, click the Add Version button in the script's details page.

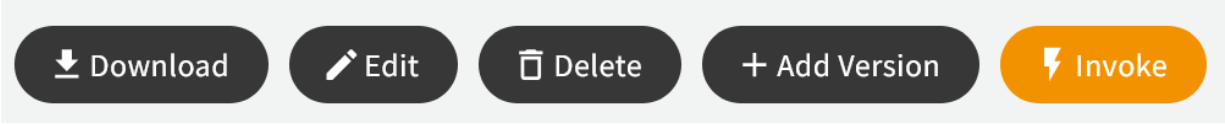

Figure 5 Add Version button

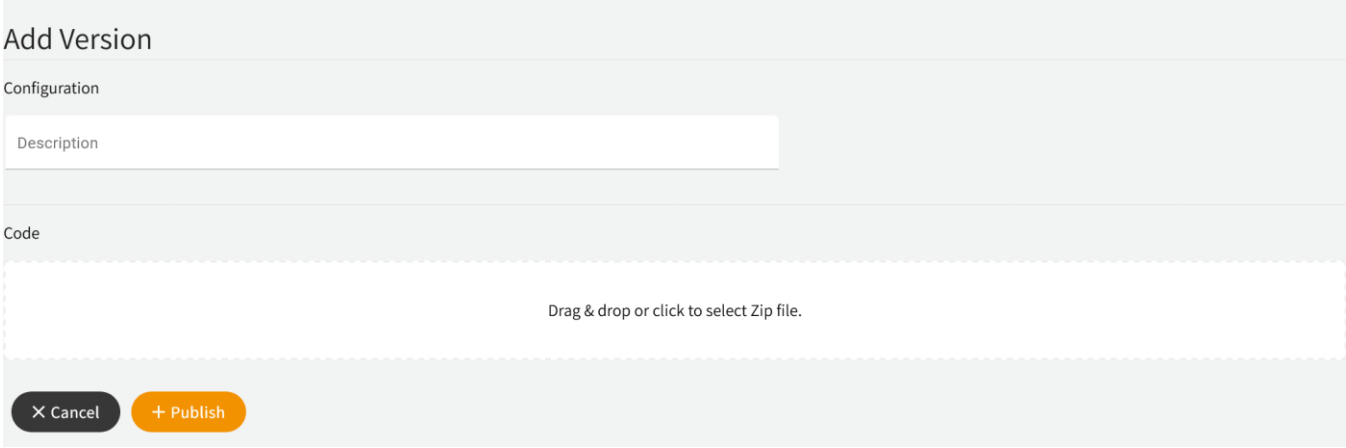

Figure 6 Add Version form

Once published, the new script version will have a version number automatically assigned. All versions of a script can be viewed at the bottom of the script details page.

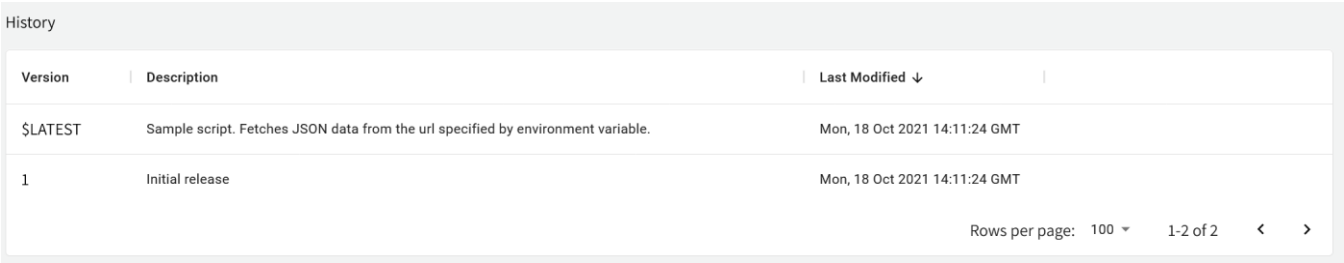

Figure 7 View of a script's versions

Edit Script

A scripts configuration and environment variables can be edited with the Edit button in the script's details page. Editing a script's details will only have the changes applied to the $LATEST version. Previously created script versions will be unaffected and are not editable.

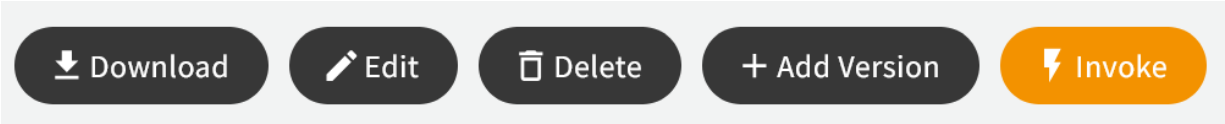

Figure 8 Edit button

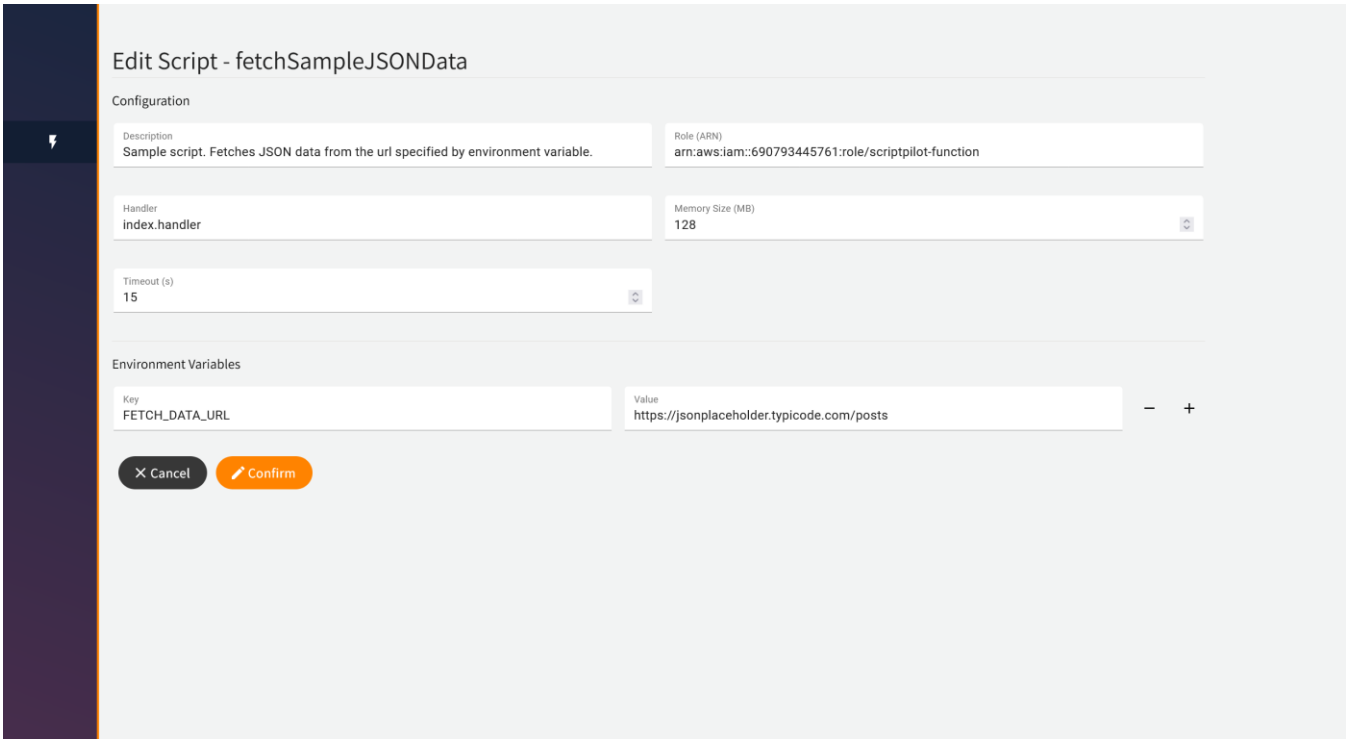

Figure 9 Edit Script form

Delete Script

A script and its associated code can be deleted with the Delete button in the script's details page. Deleting the $LATEST version of a script will also delete all versions and their code while deleting a specific version will not affect any other versions.

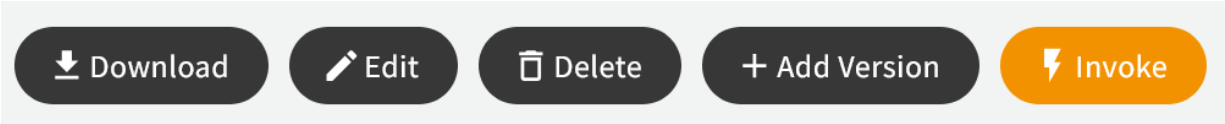

Figure 10 Delete button

Attempting to delete a script will trigger a confirmation pop up. Upon confirming, the deletion will be completed and the script will disappear from the scripts' list.

Figure 11 Deletion of script confirmation pop up

Enable Script API Key

In order to invoke a script externally, it needs to have an API key enabled. To do so, click the API Key button in the script's details page. The API Key will then be generated for that specific script version, encrypted, stored, and displayed in the script's details.

Figure 12 API Key button

Invoke Script

Invoking a script executes its code and returns a response. Currently, there are two methods of invoking a script: manually through the ScriptPilot app, or externally through an HTTP request.

MANUAL INVOCATION

From a script's details page, click the Invoke button.

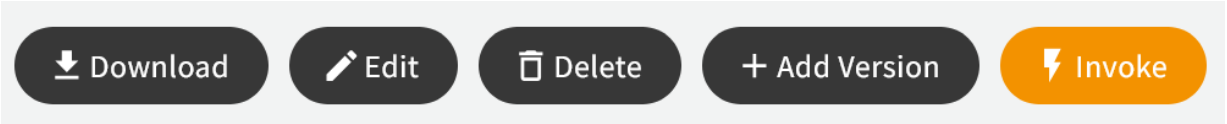

Figure 13 Invoke button

A panel will appear showing the invocation panel. There are two sections:

Input: the input section defines the type of invocation that will be performed. A collection of key-value pairs of data can be passed into the script as parameters, similar to environment variables but instead defined at the moment of invocation (this is useful, say, for a script that acts based on a given record ID number). Within the script, these values are accessed through an event object (for example, event.*keyName*). Furthermore, the invocation type must be set. During synchronous invocation, a script's code will be executed and a response result returned. Whereas an Asynchronous invocation will just have the script's code executed without waiting for a response. There are differing timeout restrictions based upon the type of invocation chosen (see Usage > Notable Limitations).

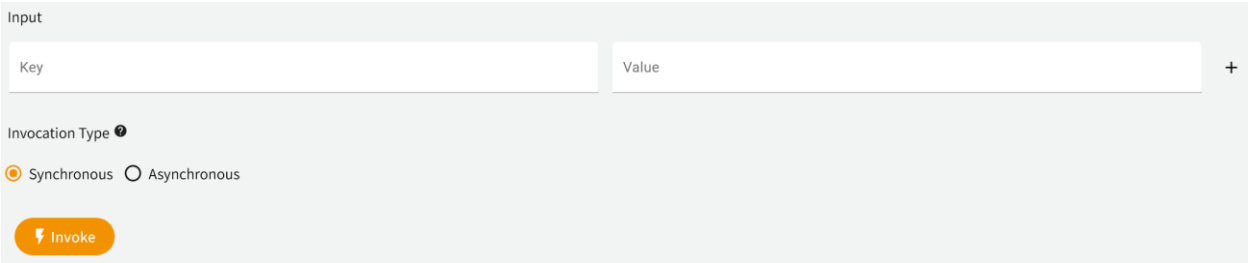

Figure 14 Script invocation input

Output: upon clicking the Invoke button, an output section is visible. It includes a code response if the invocation type was set to synchronous, and an indicator identifying if the execution itself of the script was successful (please note that this is different to logic errors returned within the script's response).

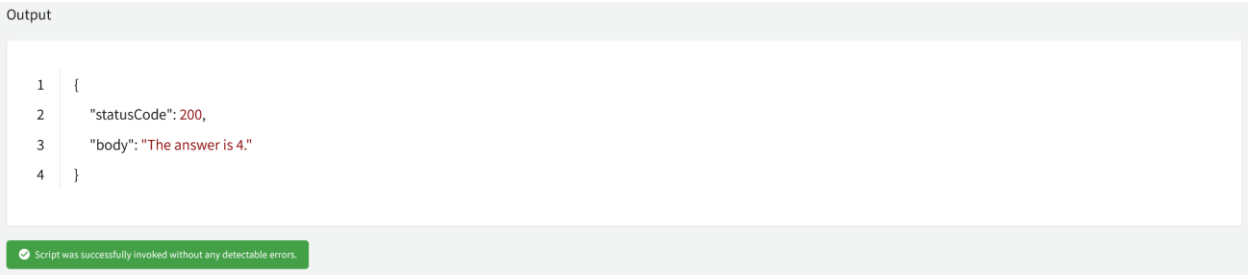

Figure 15 Script invocation output

EXTERNAL INVOCATION

An API key may be used to invoke a script externally. An HTTP POST request is made to ScriptPilot's publicly exposed API which authenticates the request, invokes the script, and returns a response. Script parameters may be passed in a JSON format as parameters of the HTTP request, under the "Payload" key (For example, {"Payload": {"key1":"value1","key2":"value2"}}. The format of the API key contains everything necessary in order to perform the invocation. Take for example the following sample API key being deconstructed:

---

https://www.scriptpilot.codebycarlos.co.uk/api/scripts/fetchSampleJSONData/versions/$LATEST/external-invoke?auth_type=apikey&apikey=RDCe0.-.A6aUOsDaomYFbpmb2wooP45dePdRbI~v~qnem~7n7Wv--yR_HisB4qbNO5Tdx-&invocation_type=synchronous

https://www.scriptpilot.codebycarlos.co.uk— base URL

/scripts/fetchSampleJSONData — specified script name

/versions/$LATEST — specified version

/external-invoke?auth_type=apikey&apikey=RDCe0.-.A6aUOsDaomYFbpmb2wooP45dePdRbI~v~qnem~7n7Wv--yR_HisB4qbNO5Tdx- — authentication code

&invocation_type=synchronous — invocation type, can be set to synchronous or asynchronous

---

Download Script Code

A script's code may be downloaded by clicking the Download button in the script's detail page.

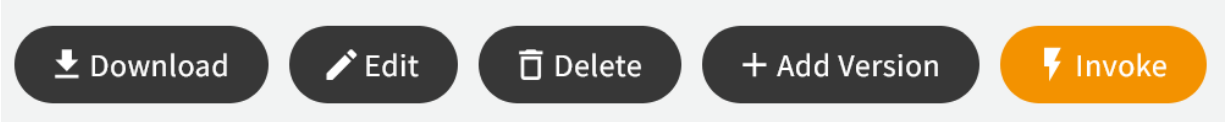

Figure 16 Download button

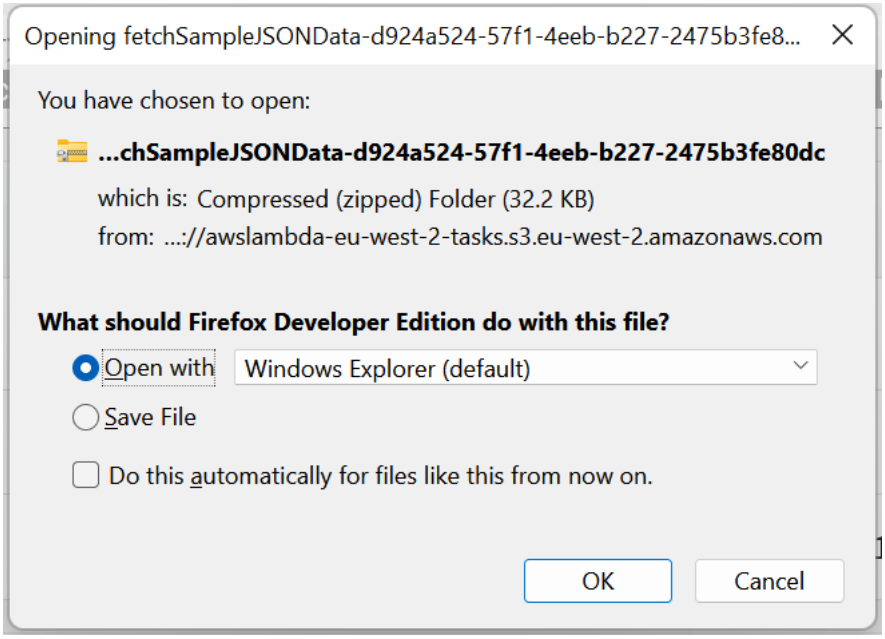

Figure 17 Downloading a script's code

Notable Limitations

SCRIPT TIMEOUT

A script's synchronous timeout is limited to 30 seconds, this is a limitation that is put in place by Heroku, ScriptPilot's web hosting platform. Synchronous scripts are those for which a user is generally waiting for a response, and therefore should be designed to maintain the lowest possible timeout anyways. Asynchronous scripts which are ideal for background and longer running tasks are limited to 15 minutes by AWS Lambda. A script that has exceeded its timeout limit will return an error.

COMPUTING POWER

AWS Lambda provides for up to 1 million free invocation requests or 400,000 GB-seconds of computing time per month.

How fast this computing power is used up depends on the size of the RAM (Random Access Memory) used by each script, how often they are invoked, and for how long. Take for example a script that has its memory size allocation set to 256 MB, and an average invocation duration of 20 seconds (0.256 GB x 20 seconds = 5.12 GB-seconds); each invocation would use up 5.12 GB-seconds of computing time. This means that such a script could be invoked a total of approximately 78,125 times before it incurs a monthly cost.

SCRIPT STORAGE

All script code is stored within AWS servers. This has a storage size limit of **75 GB**, after which it is necessary to request a quota increase**.**

How fast this total space for scripts is used up depends on the runtime of the scripts used and their resulting file sizes. Each script is limited to **50 MB** when uploaded directly through ScriptPilot. Ideally, scripts should be kept as small as possible as larger codebase sizes take longer to initialise and execute. At 50 MB, this allows for storing 1,500 scripts. It is possible to use code bundlers and minimizers to reduce the size of scripts by up to 99%, meaning a 50 MB script instead takes up 0.5MB. At 0.5MB, this allows for storing 150,000 scripts. When a script or script version is deleted, the space taken up by its code is cleared.

APPLICATION STORAGE

ScriptPilot connects to a cloud MongoDB Atlas database for storing key data necessary for the running of the app. This has a storage size limit of **512 MB**, after which it incurs a monthly cost (see Shared cluster). The following data is stored:

User, session, and account data (Capped): Average document size is 0.0007355 MB, and the collection of documents is capped at 3.15 MB. This means that after approximately 4,282 documents, the oldest will be deleted to clear space, which is appropriate as this data is only ever needed temporarily during a user session.

API keys: Average document size is 0.0003381 MB. With the other document types and the size taken up, this leaves 508 MB remaining to be occupied by API keys for each script or script version that requires it. This means that after approximately 1,502,514 script API keys, storage capacity would be exceeded. When a script or script version is deleted, the space taken up by its API key is cleared.

WEB HOSTING

Heroku's free tier initially provides 550 hours of free web hosting each month to distribute between web apps (a month typically has up to 744). Registering a credit card to the account provides for an additional 450 hours each month. This means that hosting for one application is free of cost. When inactive for over 30 minutes, free sites are put into a sleeping state (meaning there is a slight delay upon being visited again); this has been mitigated with the one-time use of a tool called Kaffeine which pings the site periodically. For more details on Heroku's free tier, click here.

MAINTENANCE

Dependencies

ScriptPilot relies on the following services for providing its functionality.

AWS LAMBDA

A cloud computing service used for handling scripts' code storage and execution. It is used in tandem with ScriptPilot's API key management system. A single AWS Lambda account must be linked to ScriptPilot. A link is established through the application's environment variables (AWS_LAMBDA_ACCESS_KEY_ID, AWS_LAMBDA_SECRET_ACCESS_KEY). To login to AWS, click here.

MONGODB ATLAS

A cloud database storage service used for storing key application data (see Usage > Notable Limitations for a breakdown). A link is established through the application's environment variables (MONGODB_CLUSTER0_USERNAME, MONGODB_CLUSTER0_PASSWORD, MONGODB_CONNECT_URL). There are similar alternative services which may be used instead of MongoDB Atlas for the application's database needs; MongoDB Atlas was selected due to its cloud infrastructure and generous free-tier. To login to MongoDB Atlas, click here.

HEROKU

A web application hosting service used for bringing ScriptPilot online. It integrates directly with ScriptPilot's GitHub repo allowing for an effective CI/CD (Continuous Integration & Continuous Delivery) pipeline workflow. Changes pushed in production to GitHub will automatically trigger a new build to be published in production and for changes to be visible online after a few minutes. To login to Heroku, click here.

Updating Environment Variables

Environment Variables or Config Variables as they are referred to in Heroku define and alter the way the application behaves; these are similar but completely separate from environment variables that can be defined at the script level. These are often critical settings necessary for the application's proper functioning (i.e. password to connect to a database). Therefore any changes should always be done with caution. For instructions on configuring environment variables in Heroku, click here.