Upgrading to brainCloud from PlayFab

Cloud Scripts

Table of Contents

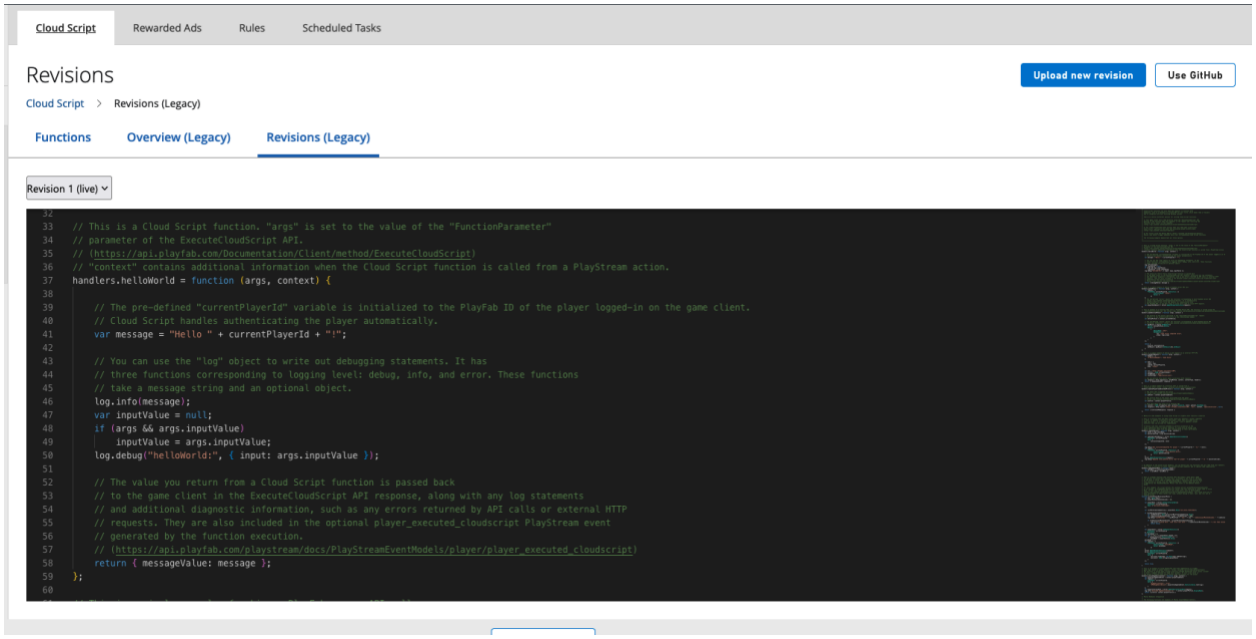
Introduction

Backend platforms provide developers with numerous features out of the box, but customizing these features for specific game needs can be challenging. To address this, many platforms offer tools for feature customization through cloud code, allowing developers to enhance existing features, create new ones, and integrate with third-party technologies.

Both PlayFab and brainCloud offer customizable cloud code (known as Cloud Script in PlayFab). However, they differ significantly in managing the creation, maintenance, and debugging of these scripts. In this section, we'll explore migrating your PlayFab cloud scripts to brainCloud and examine the additional functionalities brainCloud offers for using cloud code.

Hello World

When developers create a new title in PlayFab, it includes example code, with the first custom function often being a simple “Hello World.” We'll start by recreating this function in brainCloud.

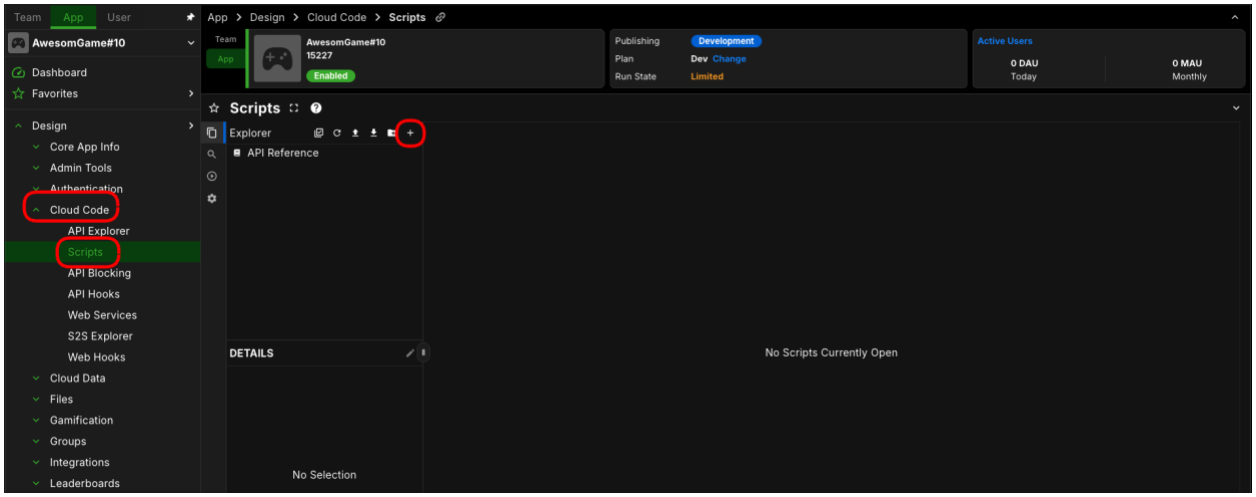


Breaking down this function there are three things going on:

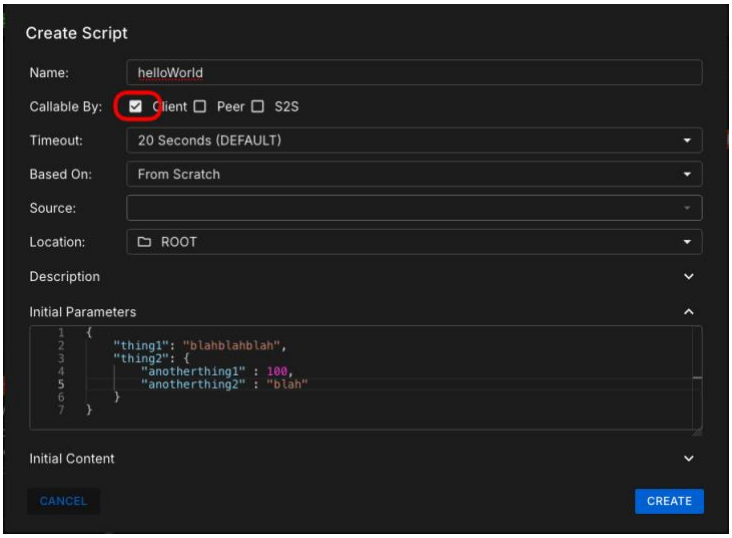
1. Declare the message string object.
2. Add logging to the script.
3. Log any inputs passed to the script.

4. Return the message string object.

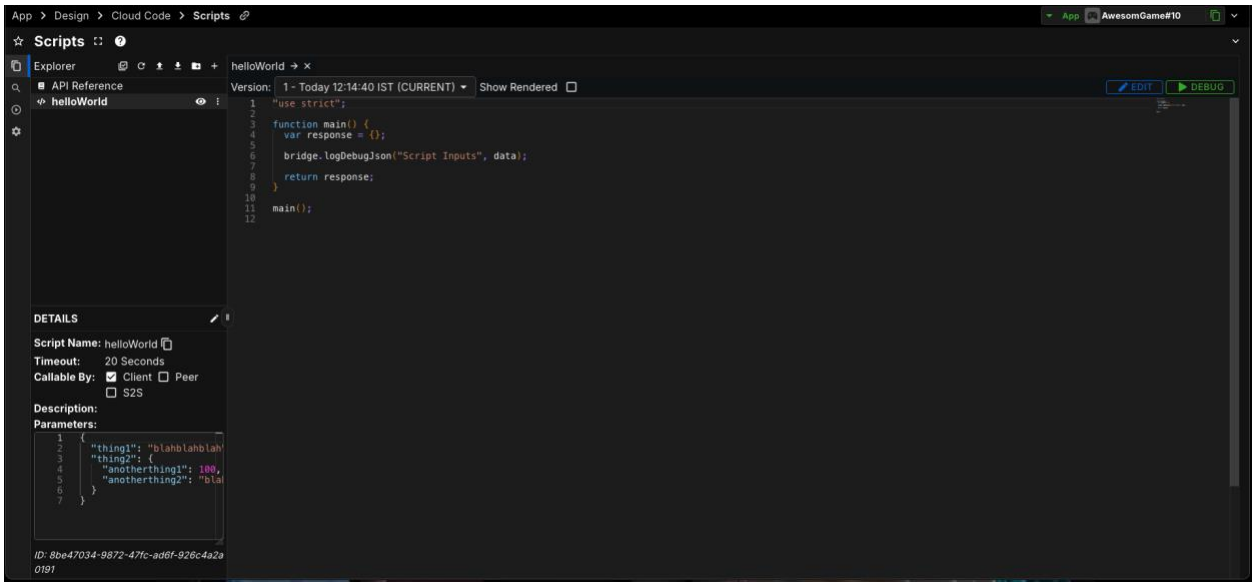
To begin, create a new cloud code script in brainCloud by navigating to the Design -> Cloud Code menu.



Click the [+] button to create a new script and give it a name, such as "helloWorld." You can leave the other attributes at their default settings for now. We will explore some of these later on, or you can read here for more information.



The key attribute here is **Callable By**, which should be set to "Client" to allow calls from the game client. For testing, you can add dummy initial parameters, which are optional but useful for debugging. Once created, your script will open in the in-portal IDE.



As we continue through this guide, we'll explore different parts of the IDE, but for now, focus on the main window where you edit your code. When executed, the script calls the `main()` function, so all code should be placed here. Unlike PlayFab, which requires handler functions, each script in brainCloud is written and triggered independently.

> Note
>
> Both PlayFab and brainCloud use JavaScript EC6, with brainCloud utilizing the Rhino engine, ensuring no compatibility issues with basic JS functionality during migration.

Initially, you'll notice one of the requirements from the PlayFab "Hello World" function is already added to the script by default.

```
bridge.logDebugJson("Script Inputs", data);
```

This line logs all input parameters to the script, which you'll observe during testing. It's called using the `bridge` object. The cloud-code bridge provides access to various services and functions within brainCloud, which we'll utilize for more complex functionality in future examples.

Next, let's add our "Hello World" code to the brainCloud script.

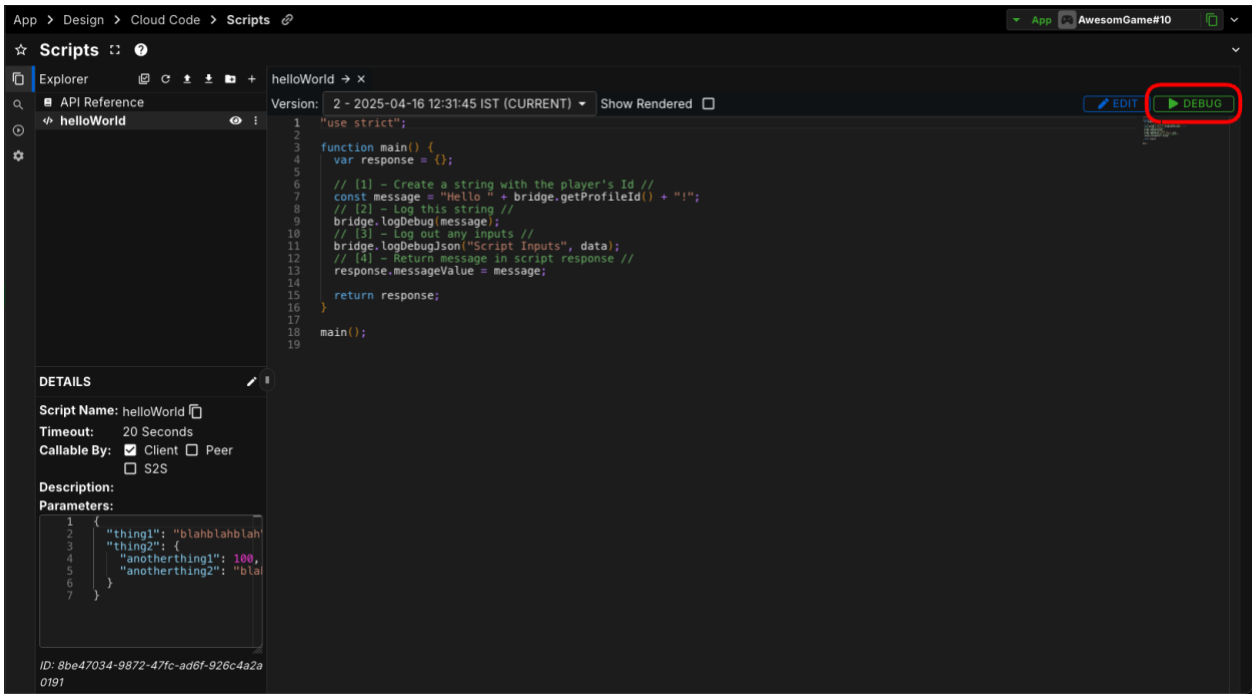
```
"use strict";

function main() {
```

```
var response = {};


// [1] - Create a string with the player's Id //
const message = "Hello " + bridge.getProfileId() + "!";
// [2] - Log this string //
bridge.logDebug(message);
// [3] - Log out any inputs //
bridge.logDebugJson("Script Inputs", data);
// [4] - Return message in script response //
response.messageValue = message;


return response;
}


main();
```

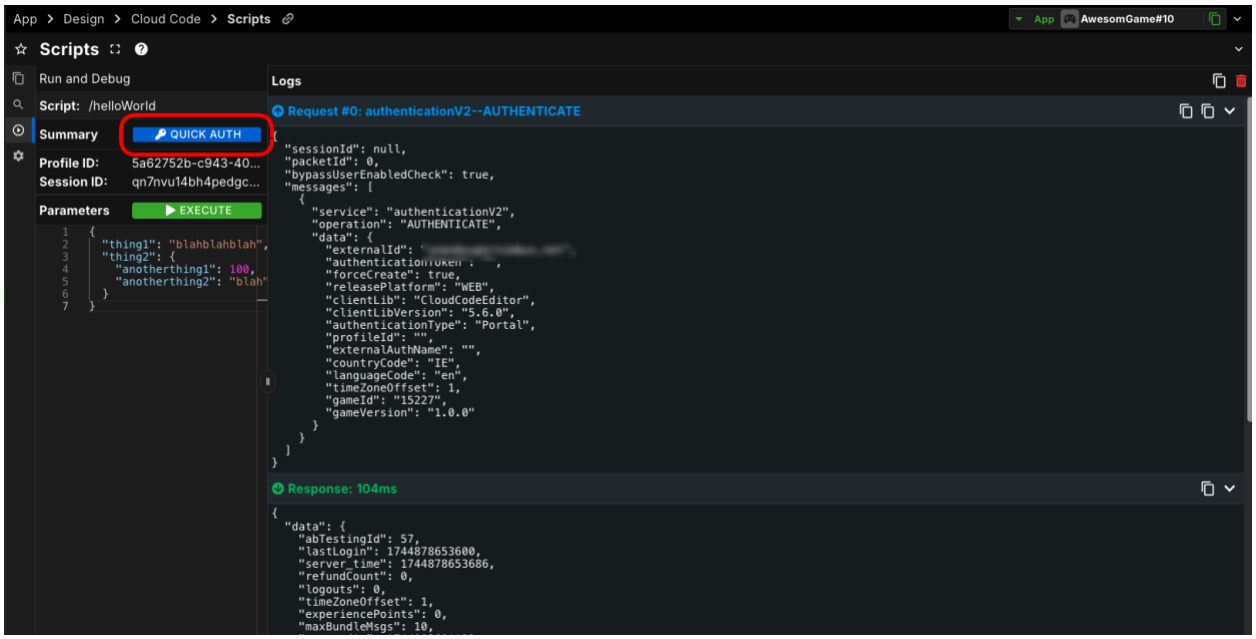
Now let's test our script.

Testing Cloud Scripts

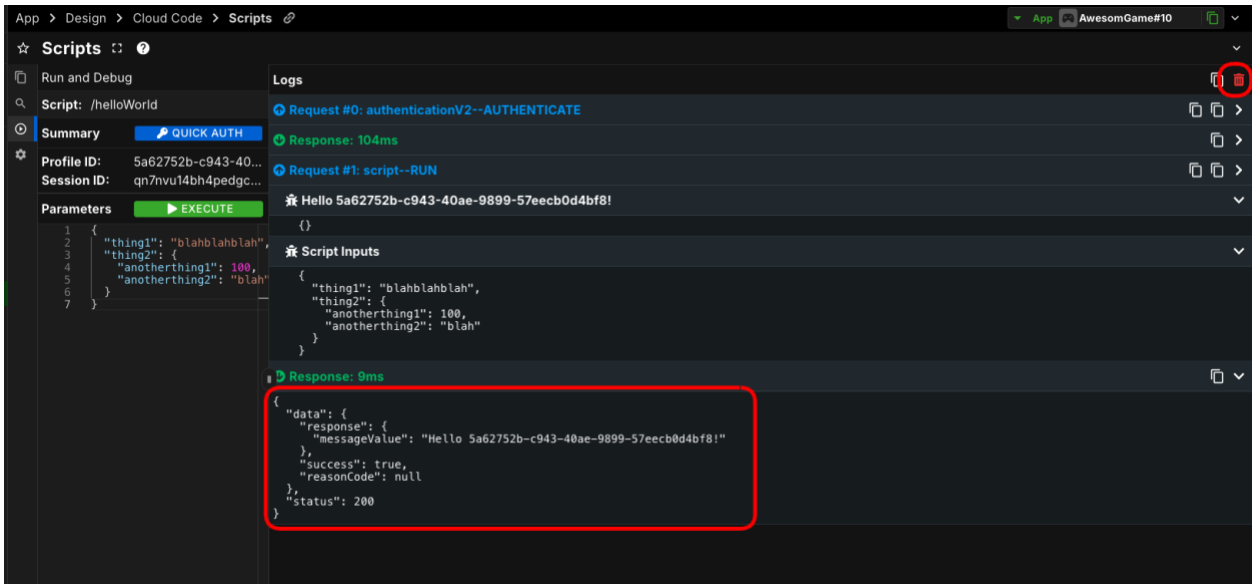
In PlayFab, you typically test cloud scripts from the player explorer dashboard. However, in brainCloud, you can test directly within the portal's IDE. Click the DEBUG button to access the debugger.



In this window, you'll find buttons for QUICK AUTH and EXECUTE, with parameters listed below the execute button. First, authenticate a user to run the script. brainCloud allows you to authenticate a user representing the developer through the portal. Click the QUICK AUTH button to do this.



Your request will appear in the blue log tab, and the server response in the green log tab. Once authenticated, the Profile Id and Session ID fields will automatically populate, indicating an active session for the player. You can now execute your script. Add any custom parameters you want to test and click the EXECUTE button.



Under the white tabs, you'll see the debug logs from your script, and the response message again appears in green. You can clear these logs using the trash icon in the top-right corner of the debugger window.

Calling Scripts From The Client

Calling cloud code scripts from the client in brainCloud is similar to PlayFab. You can call the script by name and provide callbacks for success and failure. The example below is for a C# Unity project, but other examples are available here.

```
string scriptName = "path/to/script1";

string scriptData = "{\"arg1\":42,\"arg2\":true,\"arg3\":\"example\"}";

SuccessCallback successCallback = (response, cbObject) =>

{

   Debug.Log(string.Format("Success | {0}", response));

};

FailureCallback failureCallback = (status, code, error, cbObject) =>

{

   Debug.Log(string.Format("Failed | {0}  {1}  {2}", status, code, error));
```

```
};

_bc.ScriptService.RunScript(scriptName, scriptData, successCallback, failureCallback);
```

Two main differences between PlayFab and brainCloud are evident:

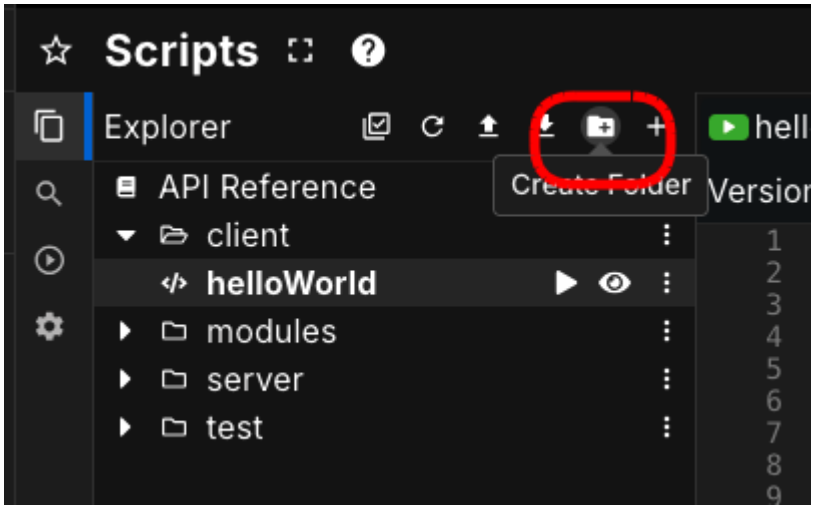
1. The script's name requires the path in your script explorer.

2. Instead of passing an anonymous object, brainCloud requires a JSON string. Therefore, you'll need a JSON serializer for your migration, such as Newtonsoft.Json for Unity.

Managing Cloud Code Scripts

We'll explore more through the course of this guide, but for now, let's focus on managing scripts. If you have many cloud script functions in PlayFab, you'll need to migrate multiple function handlers, converting them into brainCloud scripts. This includes custom functions and handlers for automation rules or webhooks, necessitating organized scripts and reusable modules.

brainCloud allows script organization through the script explorer, where you can create folders to group scripts efficiently.

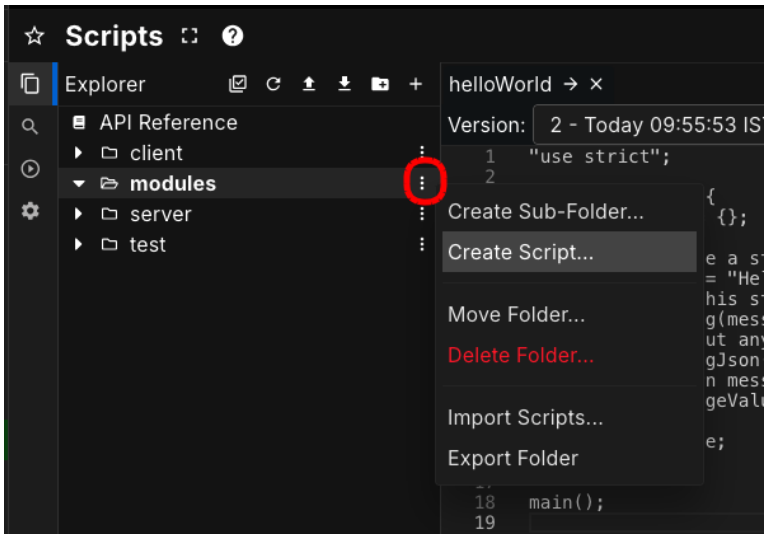


Modules

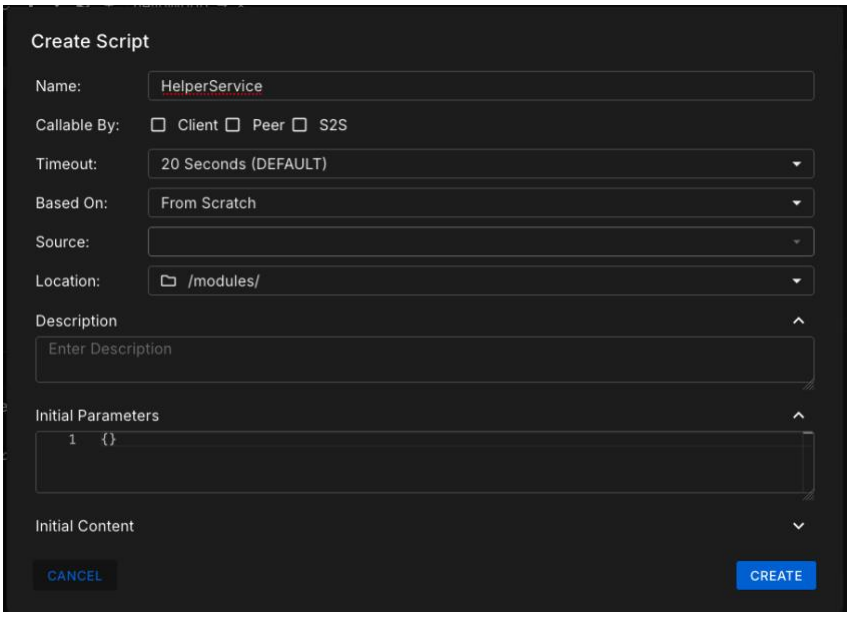
In most code projects, both frontend and backend, grouping functions for similar services or features into cohesive scripts or classes is essential. brainCloud facilitates this by referencing modules within your custom scripts.

Here's a simple example: Create a `HelperService` script with a function to return a random number within a given range; a common use case for such scripts.

Start by creating a new script in a folder called “Modules” and name it “HelperService.”



This script doesn't need to be callable by the server or client, so you can disable that option. Initial parameters aren't necessary either.



In this script, you can define your function directly.

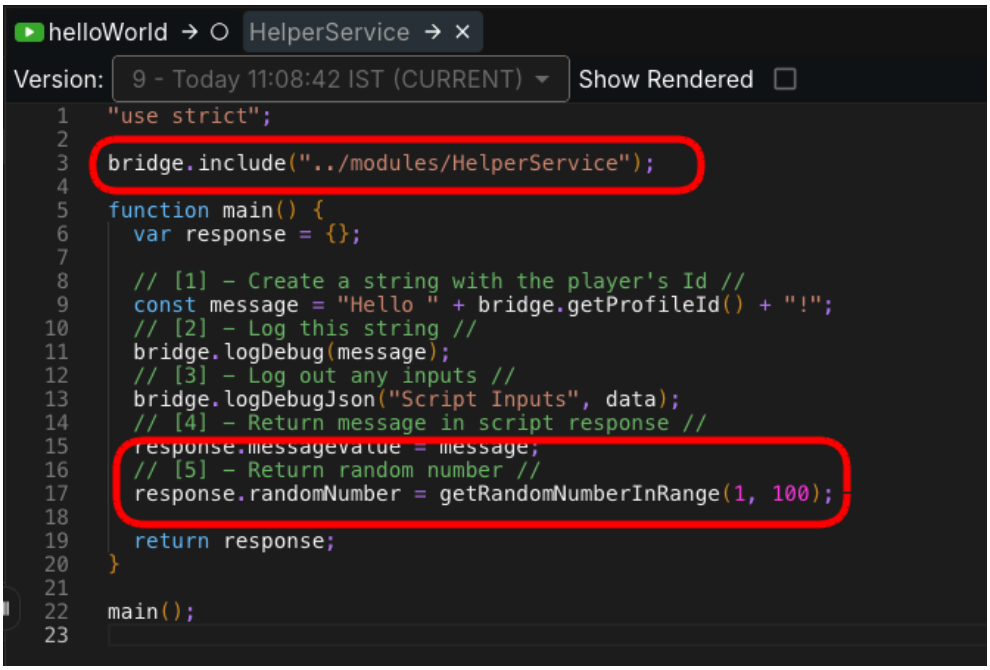
```
/**
* Generates a random integer within a specified range.
*
* @param   {number} min -  The minimum value of the range (inclusive).
* @param   {number} max -  The maximum value of the range (inclusive).
* @returns {number}        A random integer between min and max, inclusive.
*/
function getRandomNumberInRange(min, max) {
 return Math.floor(Math.random() * (max - min + 1)) + min;
}
```

Now, let's reference this function from another custom script. For simplicity, we'll use the "Hello World" function we worked on earlier. We aim to return a random number in the response, along with the “Hello” message.

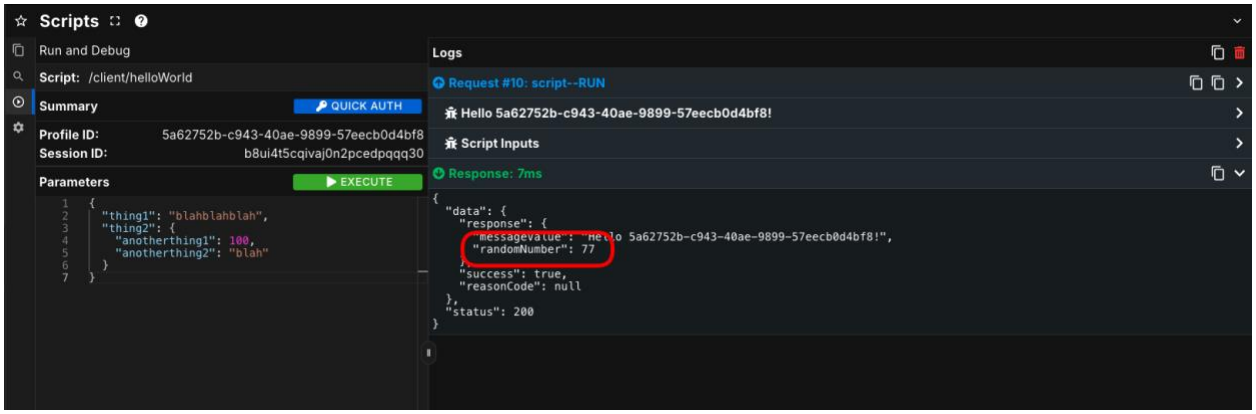
Add an `include` call at the top of the script, before the `main()` function.

> Note
> Previously, we moved the `helloWorld` script into the “client” directory, and the `HelperService` script is in the “modules” directory. Ensure the correct path in the `bridge.include` call so the

script can locate the module.



Now, if you use the debugger, you'll see the random number returned in the response, indicating the module loaded correctly.

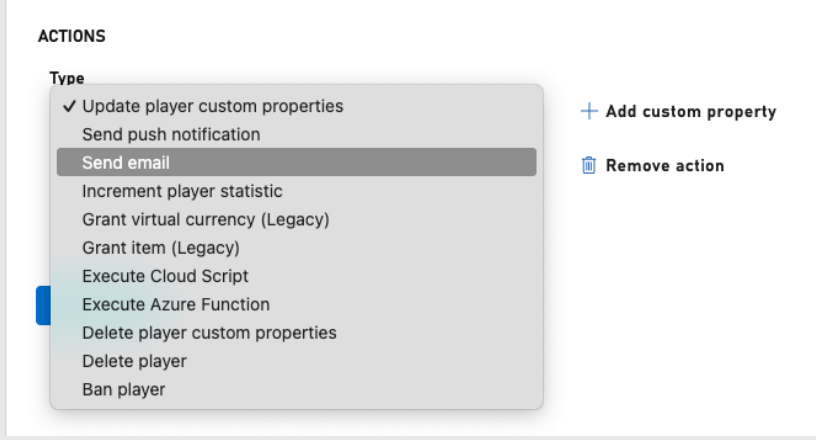


This is just a quick example of using modules or classes with the include feature. You can adapt it to fit your needs as you migrate your code.

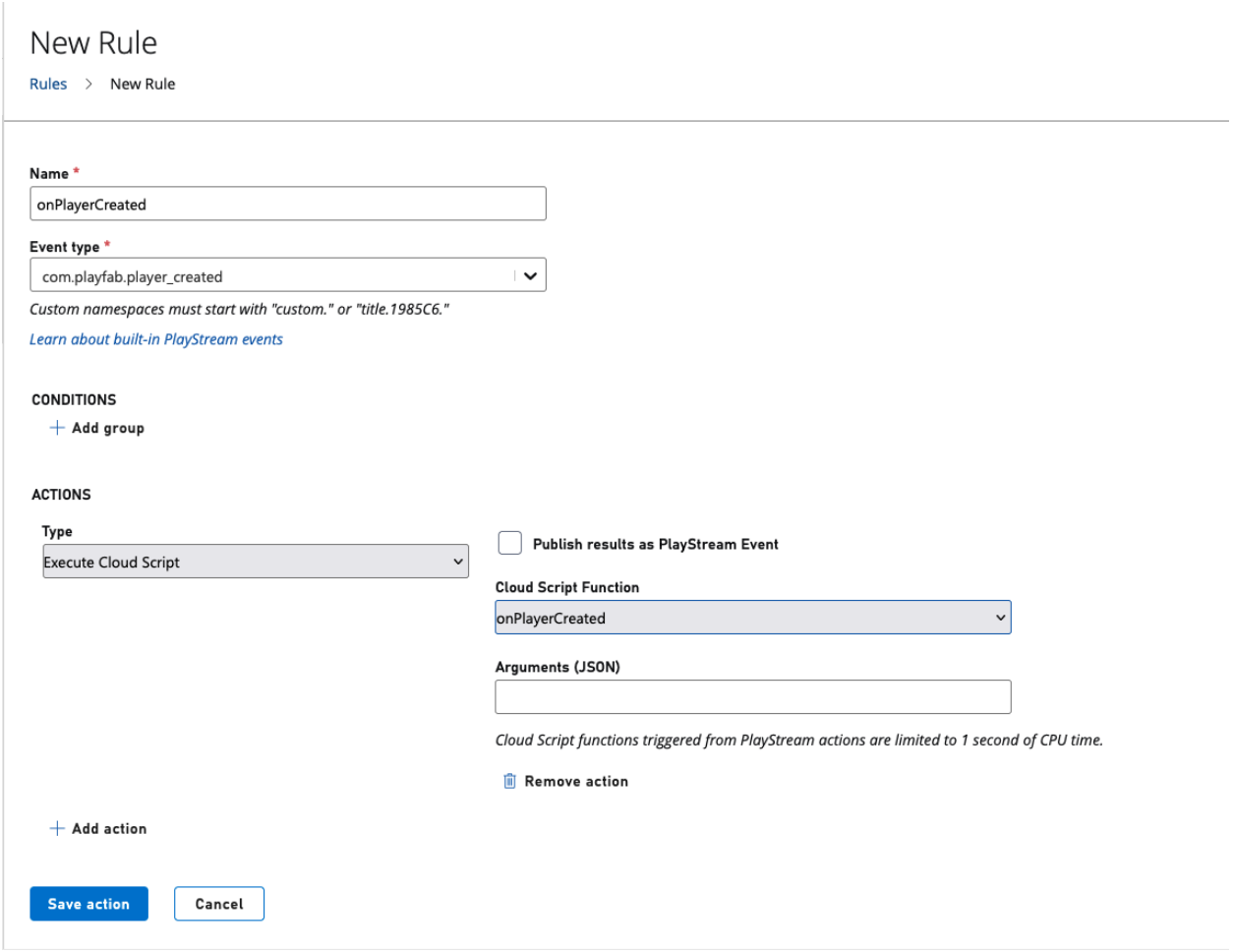
Rules, Web Hooks & API Hooks

Automation rules in PlayFab allow developers to act upon server events. brainCloud offers a similar feature called API Hooks. You can compare available hooks for PlayFab and brainCloud in their respective documentation here and here.

While PlayFab supports various actions upon event triggers, brainCloud allows only custom code execution. However, similar functionalities, like adding items, currency, or updating stats, can be achieved through code.

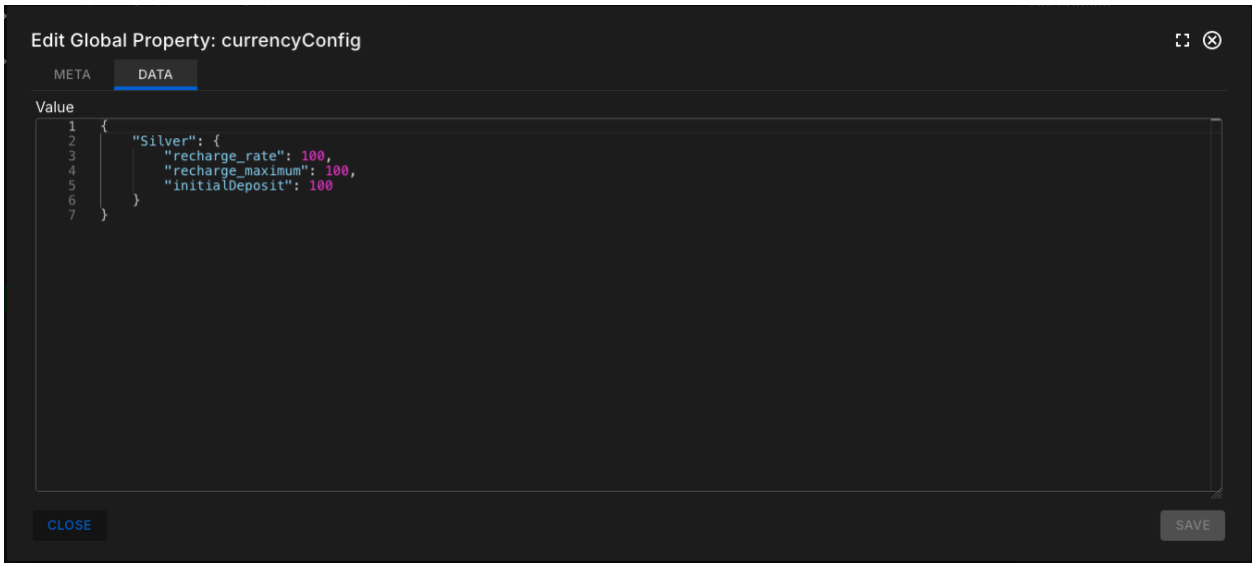


Let's look at a simple example where an action is performed when a player creates their account for the first time.

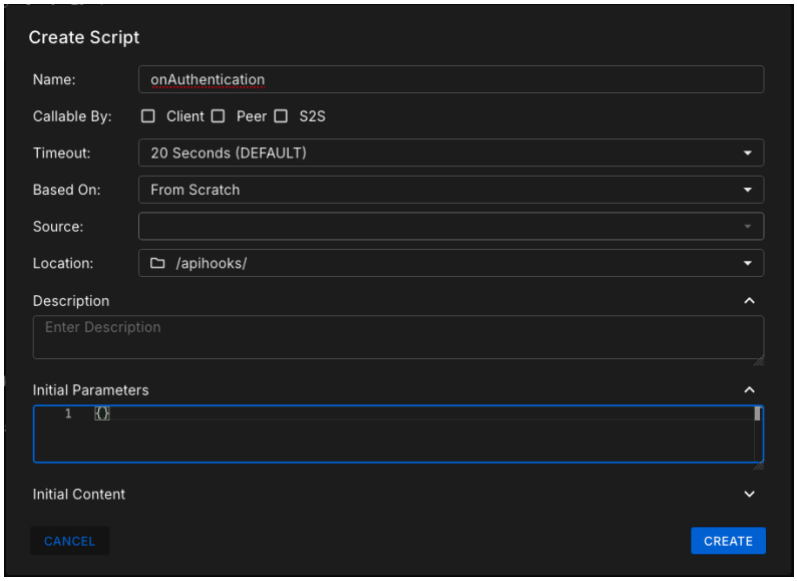


In this example, we'll deliver starting currency to a player upon account creation. This feature, discussed in the Virtual Currency guide allows PlayFab to configure initial deposits for all new players.

Begin by defining the currency amount using Global Properties. You can follow from the tutorial linked above. Ensure the "currencyConfig" property schema includes the "initialDeposit" parameter.



Next, create the post-hook script. Name it "onAuthentication" and add it to a new folder called "apiHooks." Ensure it's not callable from the client or server by disabling this option.



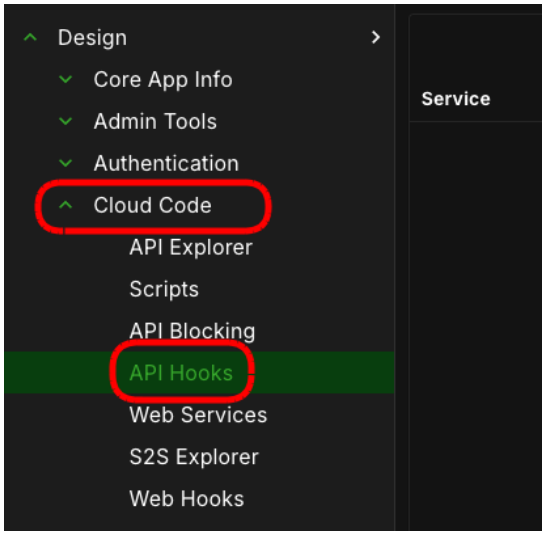
Since this script is invoked after authentication, we'll check the "newUser" flag passed from the authentication request. We'll load the global properties, deliver the initialDeposit to the user, and update the account balance response data to reflect the new balance in the authentication response.

```
"use strict";
```

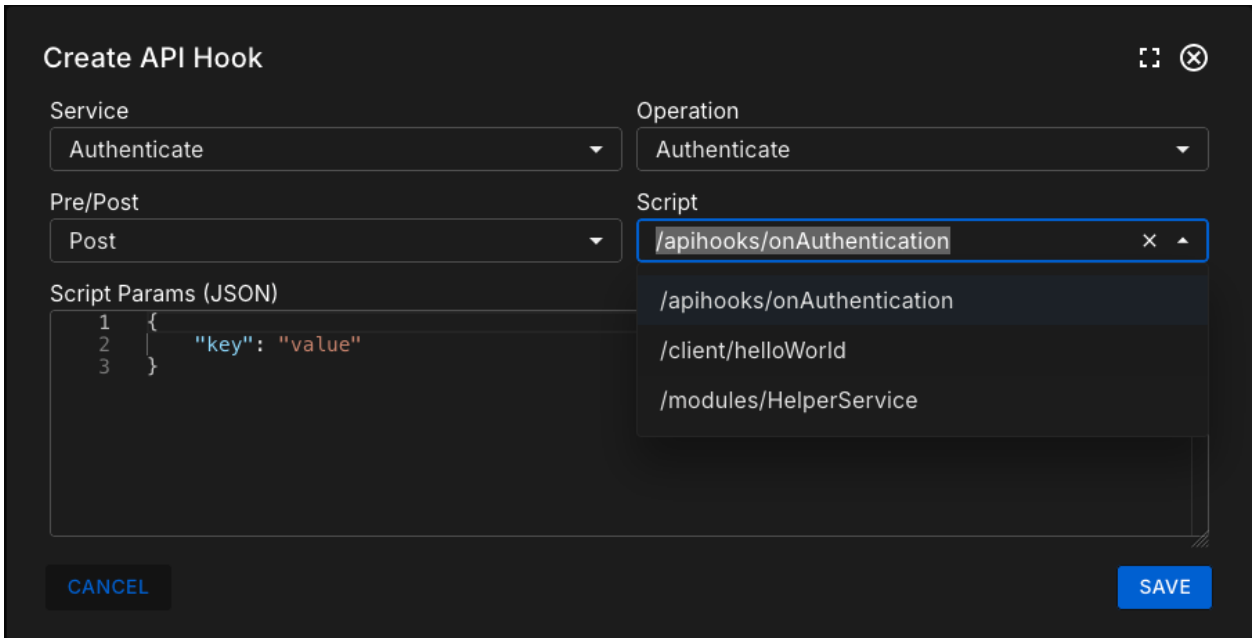
```
function main() {


 let response = {
   status: 200,
   data: data.message,
 };


 if(response.data.newUser == "true"){


   const propertyName = "currencyConfig";
   // [1] - Load Global Property //
   const globalAppProxy = bridge.getGlobalAppServiceProxy();
   const propResult = globalAppProxy.readSelectedProperties([propertyName]);


   if (propResult.status != 200) {
     // handler errors //
   }
   const currencyConfig = JSON.parse(propResult.data[propertyName].value);
   bridge.logDebugJson(propertyName, currencyConfig);


   // [2] - Deliver Virtual Currencies //
   const virtualCurrencyProxy = bridge.getVirtualCurrencyServiceProxy();
   Object.keys(currencyConfig).forEach(vcKey => {


     let vcAmount = currencyConfig[vcKey].initialDeposit;


     const awardVCResult = virtualCurrencyProxy.  awardCurrency(vcKey, vcAmount);
     if (awardVCResult.status != 200) {
       // handler errors //
     }
     // update authentication response //
```

```
    response.data.currency[vcKey].balance += vcAmount;

    response.data.currency[vcKey].awarded += vcAmount;

  });

}


return response;

}


main();
```

Save this script, then add it to the post-authentication API Hook. You can find API Hooks in the Design -> Cloud Code menu.



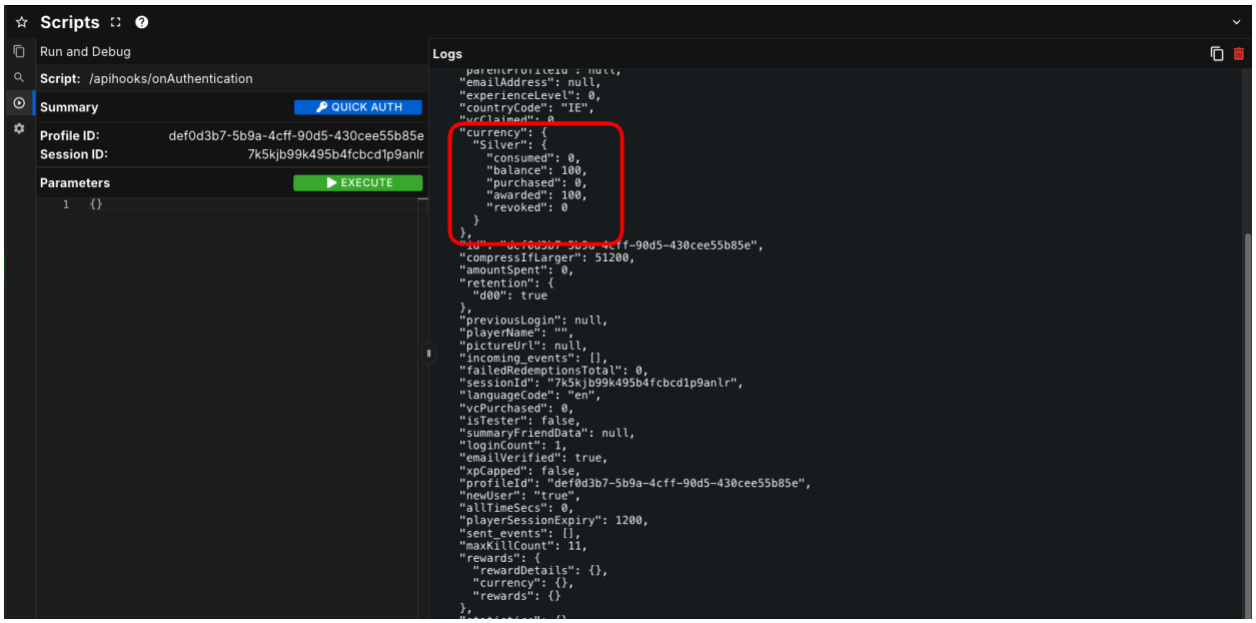
Create a new API Hook for the Authenticate service and the Authenticate operation. Set it as a post-hook, and select the "onAuthentication" script from the drop-down menu.



Now that everything is set up, test it by returning to the debugger. As mentioned earlier, authenticate the user to execute scripts. There is no need to execute the script directly; just click the quick auth button.

Note

Since we already created the default user, you'll need to delete that user for this code to execute. Refer to the section here <link> for instructions on deleting players from the brainCloud portal.



You will see the balance updated in the response information. Verify the code by running the quick auth request again to ensure the VC is only delivered once, upon the player's initial creation.

This example shows how the API Hook can be adapted to perform various actions based on the wide range of API Hooks available in brainCloud.

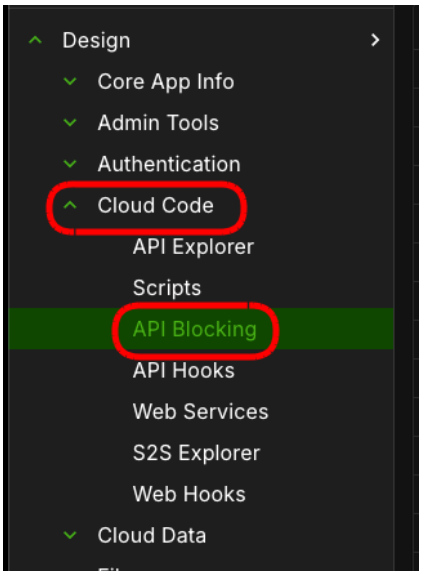
Web Hooks

PlayFab offers Web Hooks, which allow developers to reach external endpoints using HTTP requests when specific actions are triggered, similar to automation rules.

If your game uses this PlayFab feature, it can be replicated with brainCloud's Web Services. Alternatively, use cloud code and HTTP requests. An example is provided in the section on Azure Functions below.

API Access Policy & Blocking APIs

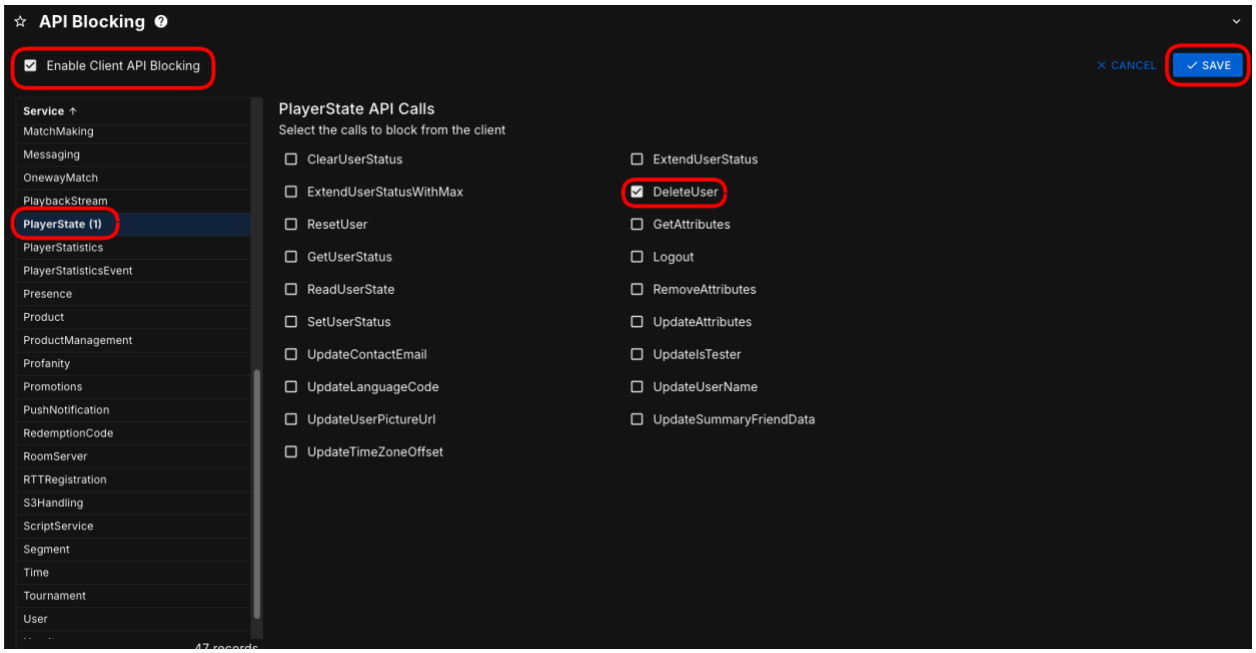
PlayFab allows developers to restrict access to certain APIs to prevent hacking or malicious use, as all out-of-the-box APIs are client-accessible by default. In PlayFab, this requires an admin call and custom code. However, in brainCloud, this can be managed directly from the portal using the API Blocking options found in the Design -> Cloud Code menu.



Lets take a look at a similar API to the *DeleteCharacterFromUser* API described in the PlayFab docs.

Note brainCloud does not have a direct 'character' feature, but we do cover how to replicate this functionality using Shared Accounts in another tutorial here <link>.

brainCloud does provide a DeleteUser API, callable from the client by default. To block this API, first enable Client API Blocking, then select the PlayerState service, choose the API, and click save.

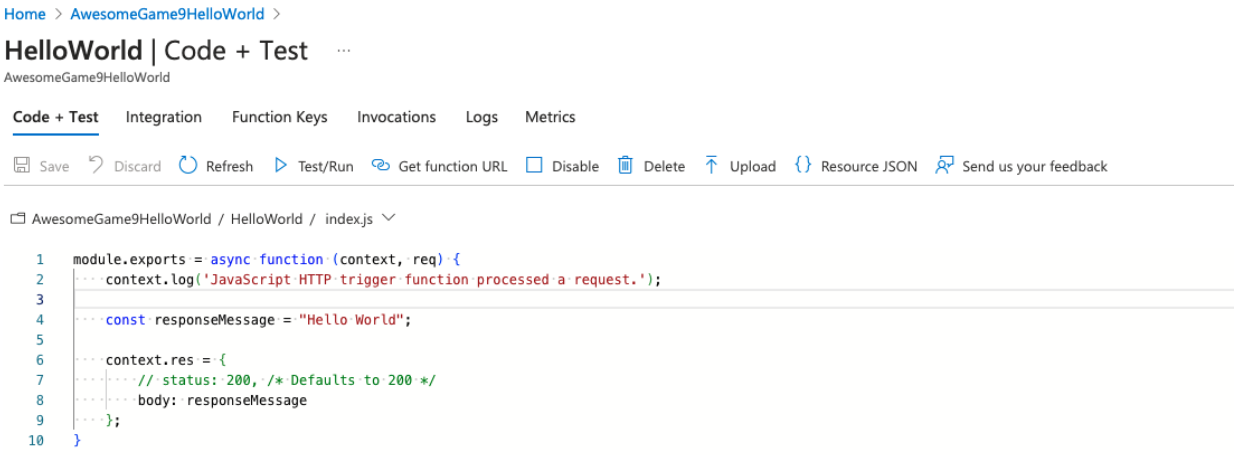


That's it! You can block any API from client execution using this method.

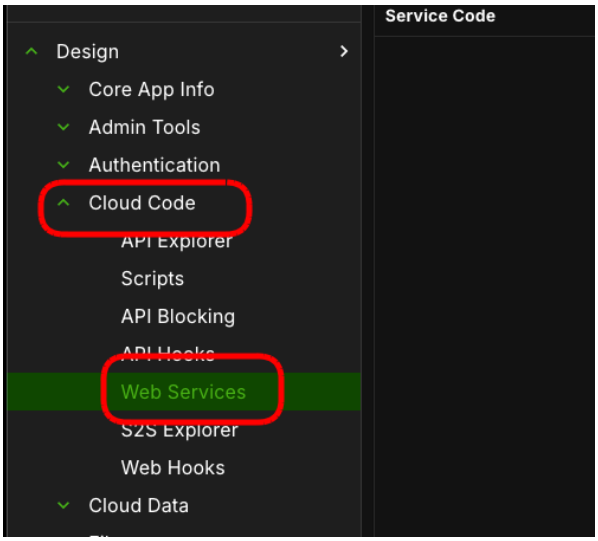
Azure Functions

PlayFab allows developers to execute Azure Functions instead of out-of-the-box cloud scripts. While brainCloud doesn't directly integrate with Azure Functions, you can achieve this using Web Services and HTTP requests from within your cloud code scripts.

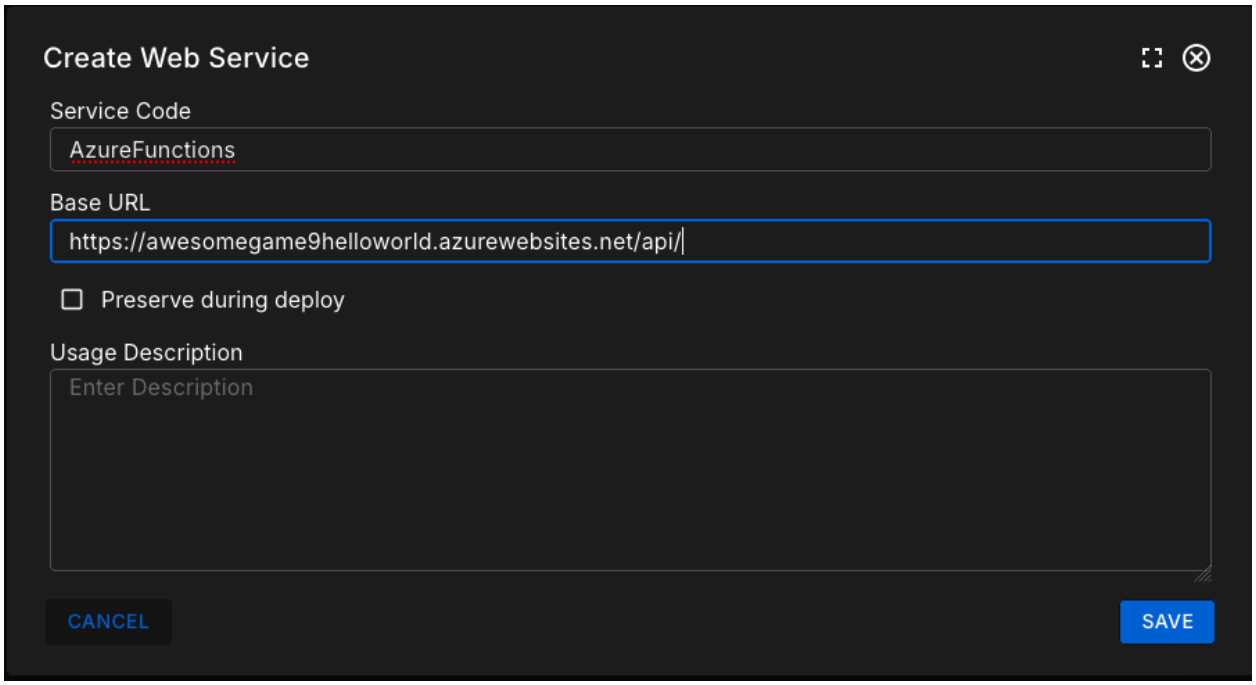
Here's how to set it up in brainCloud using a simple Hello World example as an Azure Function.



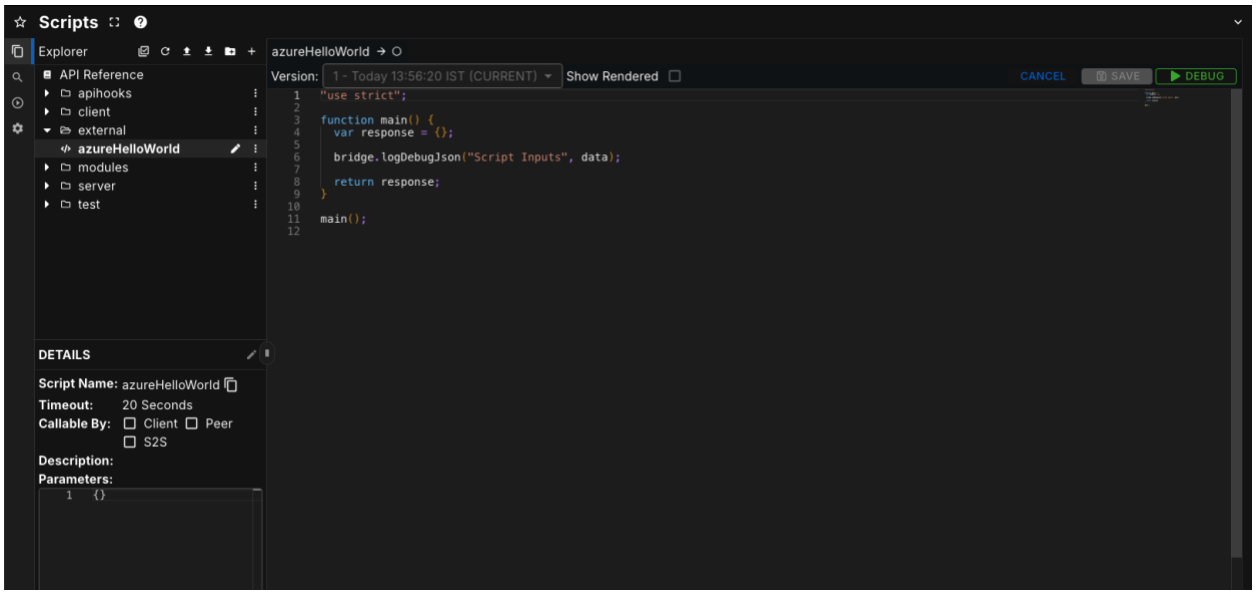
First, register the Azure Function App as a brainCloud Web Service. You can find Web Services in the Design -> Cloud Code menu.



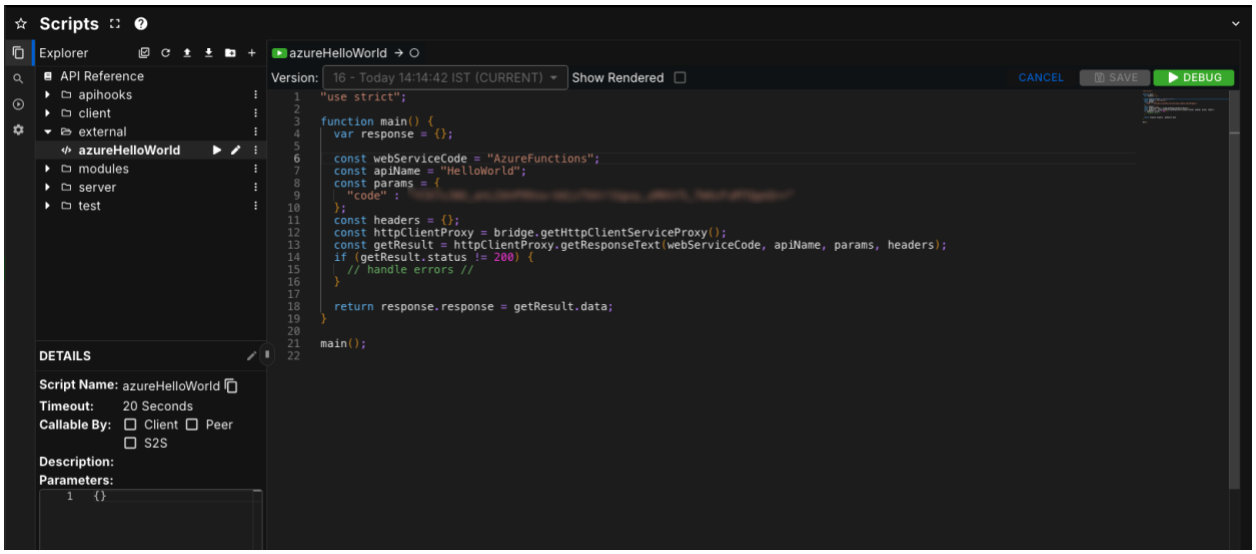
Create a new Web Service with the route of your Azure Functions App. You don't need to include the function name or key here if you have multiple functions. These can be added in your custom script.



Next, create a new script, and place it in a folder called "external." Name the script "azureHelloWorld." You may allow the script to be called from the client, as PlayFab Azure Functions are typically client-invoked or used by event hooks.



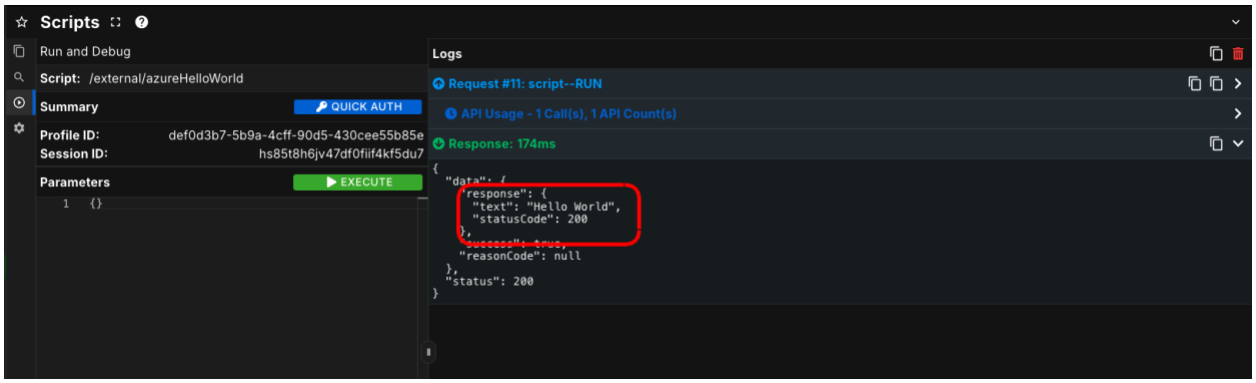
Add the following code to your new script.




```
"use strict";



function main() {

 var response = {};



 const webServiceCode = "AzureFunctions";

 const apiName = "HelloWorld";

 const params = {

   "code" : "<azure-function-code>"

 };

 const headers = {};

 const httpClientProxy = bridge.getHttpClientServiceProxy();

 const getResult = httpClientProxy.getResponseText(webServiceCode, apiName, params,
headers);

 if (getResult.status != 200) {

   // handle errors //

 }



 return response.response = getResult.data;
```

```
}

main();
```

Next, you can test this script using the debugger. You should see your Azure function's payload in the response data.



Using this feature, you can maintain your Azure Functions within the Azure environment while triggering these scripts from the brainCloud client, as demonstrated earlier.

If your Azure Functions are written in Node.js, it might be possible to migrate all code to brainCloud to keep it centralized.

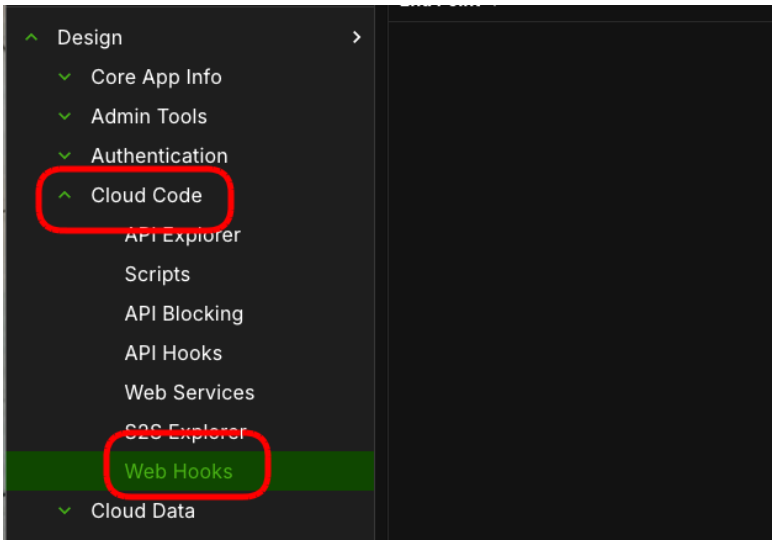
External Services & REST Requests

PlayFab allows API access via REST, which can be used by clients through HTTP requests. brainCloud offers a Server-to-Server REST API intended for back-office, admin tools, or posting player data post-validation by an external server.

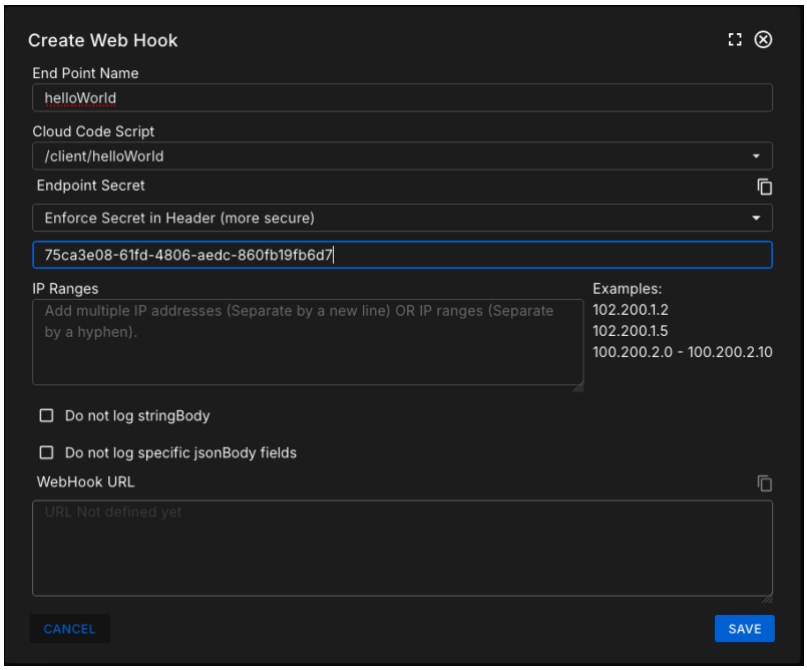
However, third-party services can execute actions on brainCloud using Web Hooks.

> Note
>
> It's important to distinguish between brainCloud's and PlayFab's Web Hooks. PlayFab's Web Hooks reach out to external servers when an action is triggered, while brainCloud's Web Hooks allow external servers to trigger actions on brainCloud.

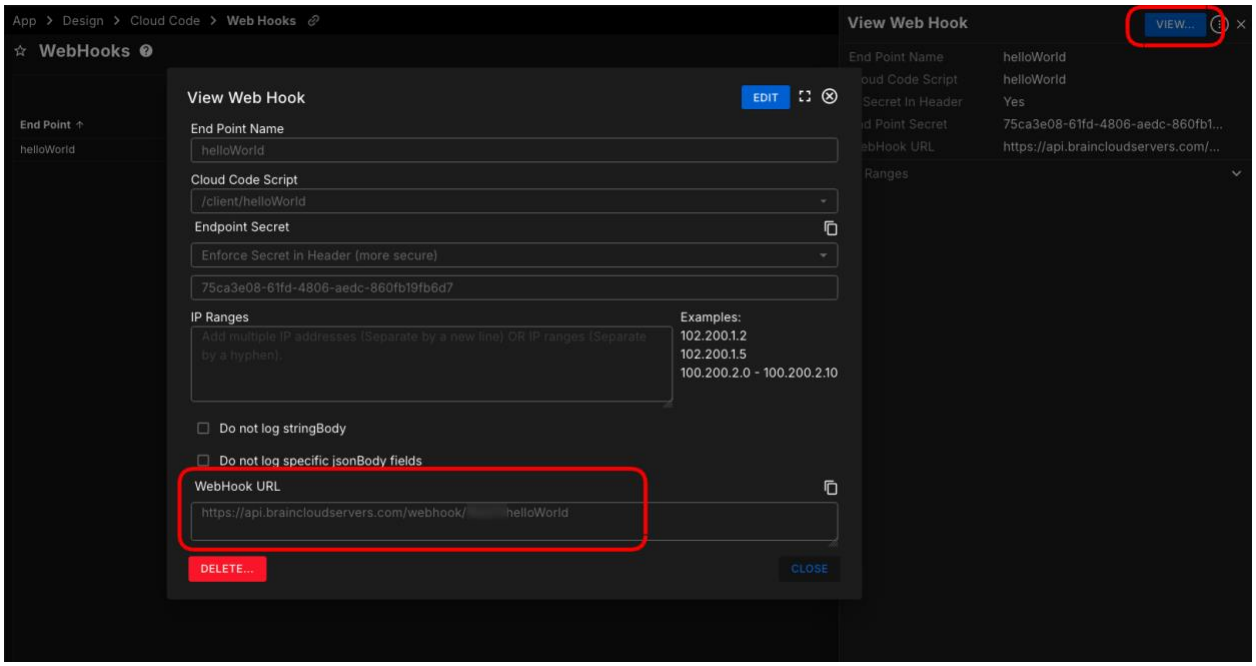
To demonstrate, let's look at a simple Hello World example. First, create a new Web Hook found in the Design -> Cloud Code menu.



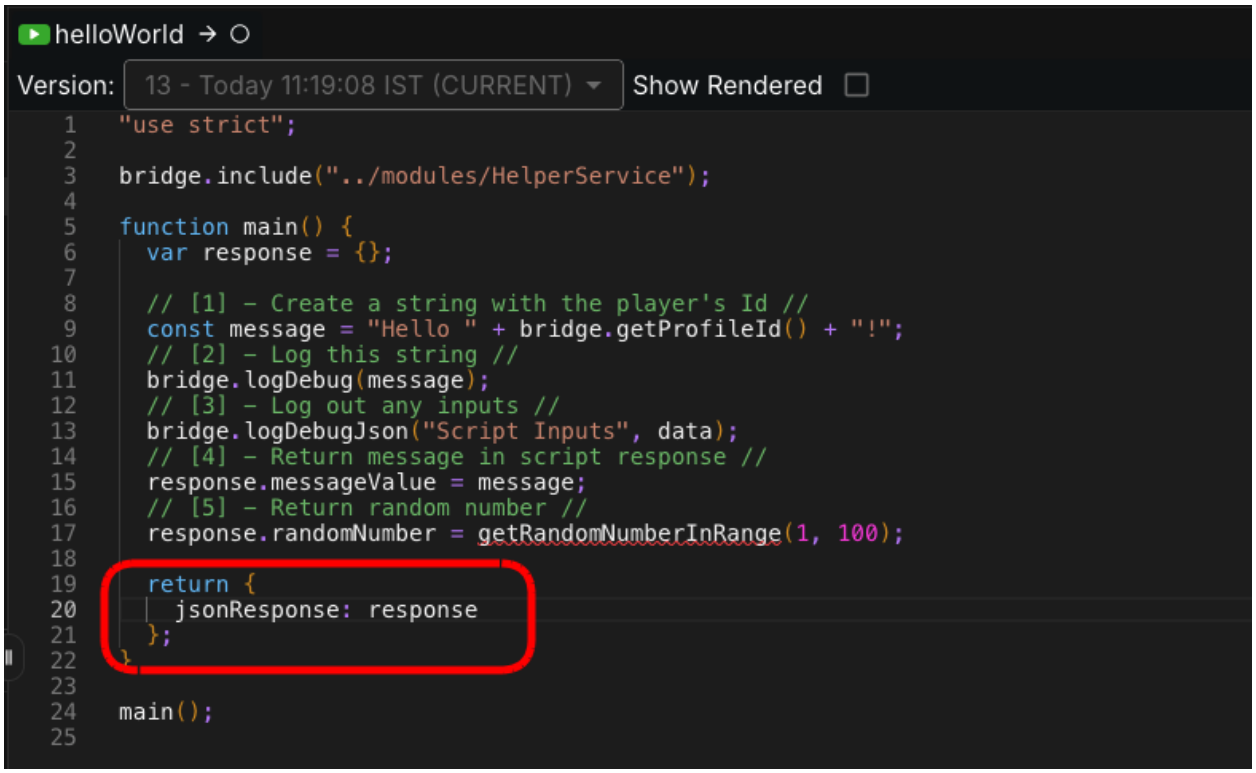
Create a new Web Hook and select the `helloWorld` script. Name the hook accordingly, as this will become the URL path used to access the Web Hook. For this example, ensure the Web Hook enforces a secret required in the request header.



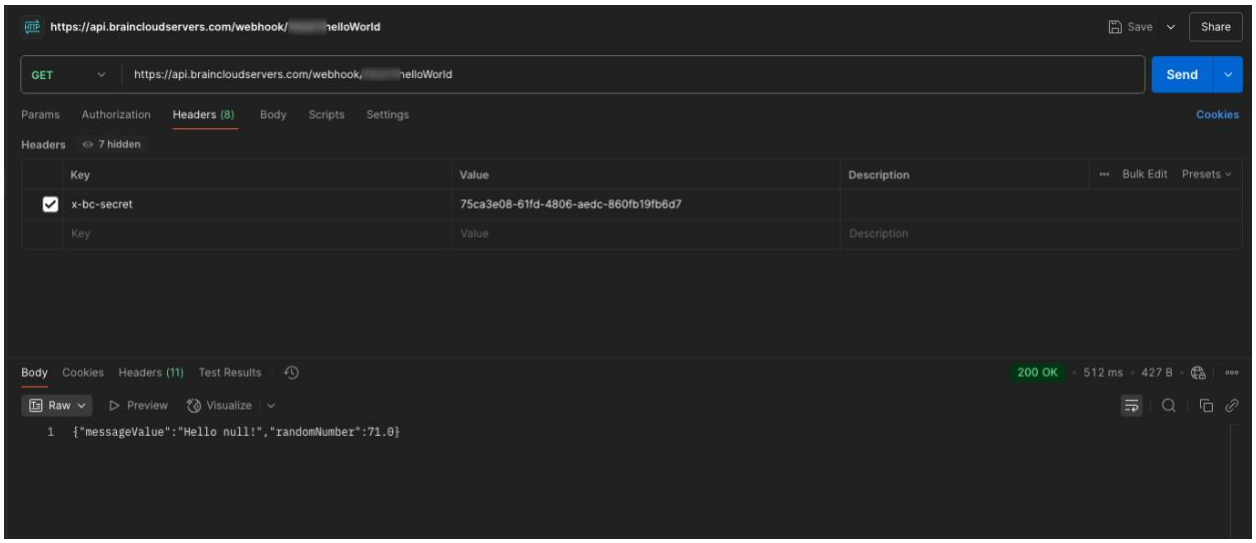
Once you save this Web Hook, a URL is automatically generated. You can view it by selecting the Web Hook endpoint and clicking the View button in the top right corner.



One more change is needed to execute the Web Hook properly. To receive a response, include a parameter called `jsonResponse` or `stringResponse` in your script. Add this to your `helloWorld` script first.



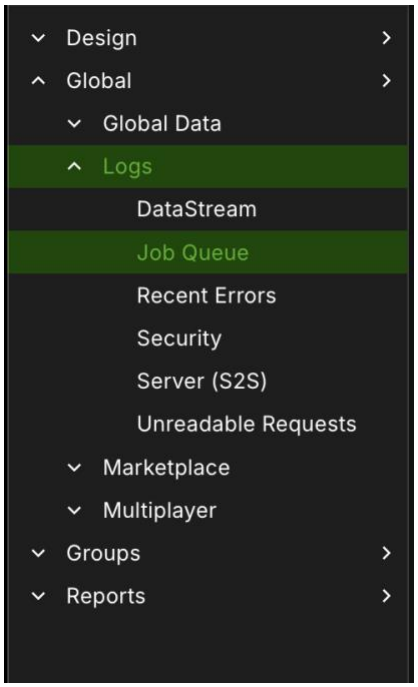
Now we can test this out using [Postman] to see if our web hook is working. Remember to add your secret to the request header with the key **x-bc-secret.**



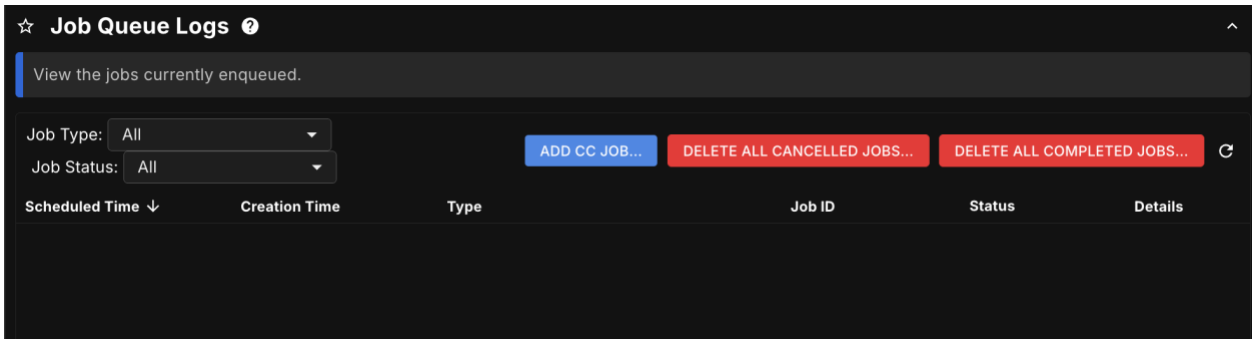
Note that the player's profile ID is returning null in the example above. This is because web hooks don't have player context as they would if you called them from the client.

Scheduled Tasks

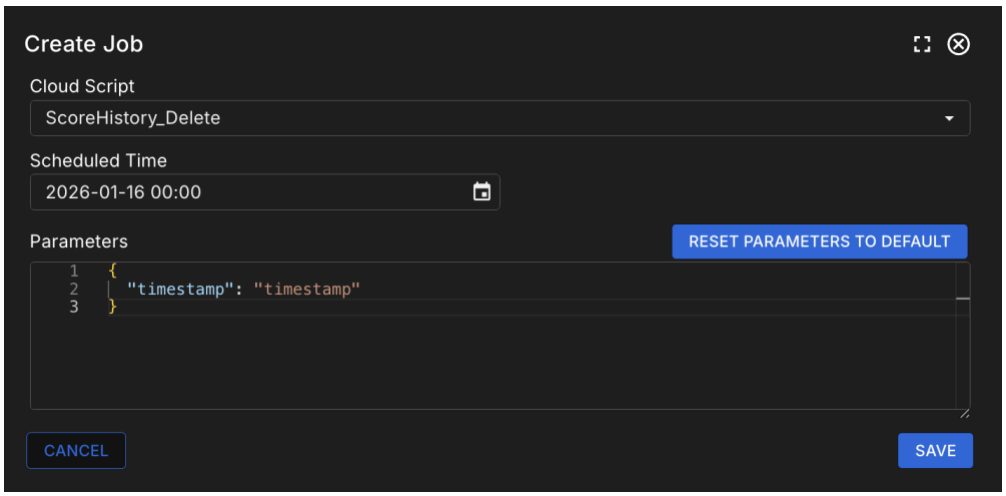
PlayFab allows developers to schedule tasks automatically using cron jobs, such as hourly or daily executions. brainCloud supports this capability directly in the brainCloud portal using a feature called scheduled jobs. To create a scheduled job navigate to the Logs –> Job Queue menu:



Click on the Add CC Job…button.



And select the Cloud Code script that you want to run and when you want to run it.



If this is a job that you want to run repeatedly at a specific time (i.e. hourly, daily, weekly, etc.) you will need to reschedule the job as the first thing that you do in the Cloud Code script.

To do this download the Script Scheduler Script from the brainCloud API Docs site and add it to your app:

https://docs.braincloudservers.com/learn/cloud-code-central/handy-cloud-code-scripts/scriptscheduler-script/#docusaurus_skipToContent_fallback

Then call the script at the start of the Cloud Code script that you want to reschedule:

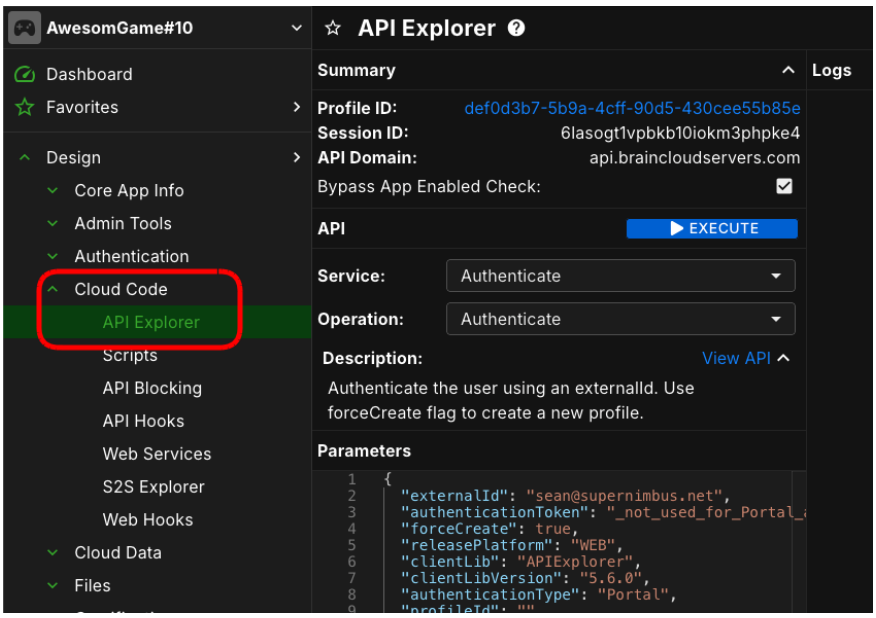
```
// First, schedule this script to run again!
// - Important to do this first - before any errors can occur!
var scriptProxy = bridge.getScriptServiceProxy();
var schedulerRetVal = scriptProxy.runScript(
                "ScriptScheduler",
           { "scriptName": "DataCacheUpdate",  "interval": 10,
"args": {}}
);
// Now continue with this script's activities!
// ...
```

If this doesn't satisfy your needs or if you want to reuse your existing Cron jobs you can also replicate that approach using Web Hooks and external cron-job tools like AWS EventBridge. This setup also allows developers to execute recurring tasks on brainCloud.

> Note
> Cron jobs can consume significant server resources, leading to performance bottlenecks at peak times. If numerous tasks are scheduled, your scripts may time out, necessitating additional code to detect and recover these tasks.
>
> Developers who need these features are advised to contact brainCloud support for guidance on safely and scalably reproducing their requirements.

API Explorer

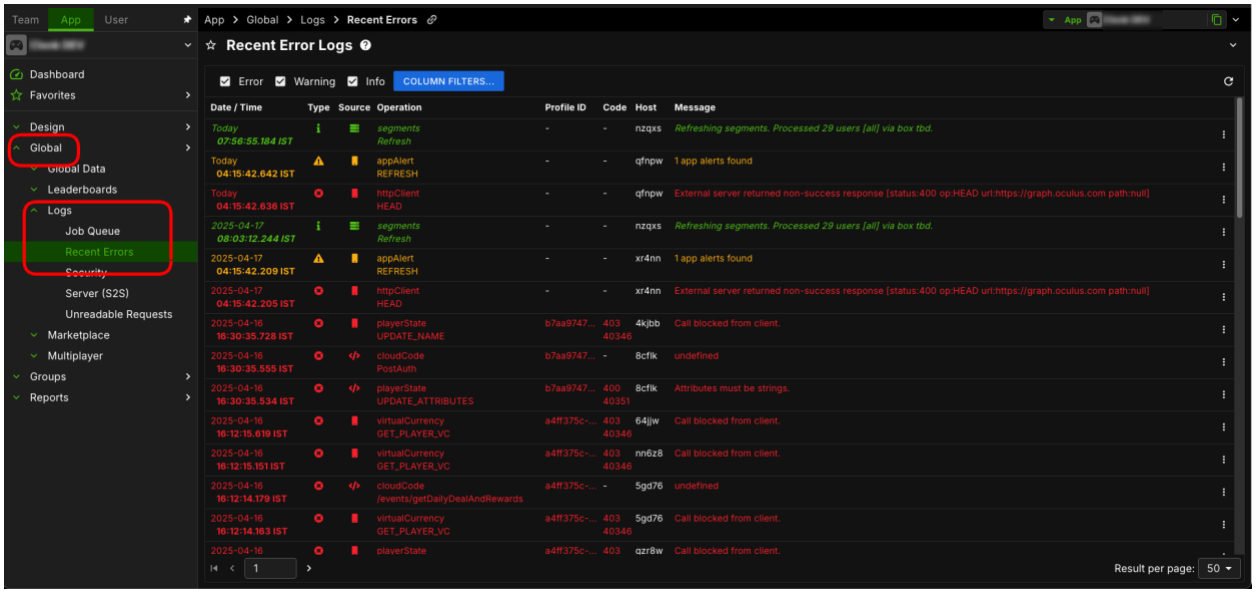
While we've explored how to execute and debug custom cloud code using the brainCloud portal IDE, you can also execute any of the out-of-the-box APIs from the brainCloud portal. This feature, called the API Explorer, is available in the Design -> Cloud Code menu.



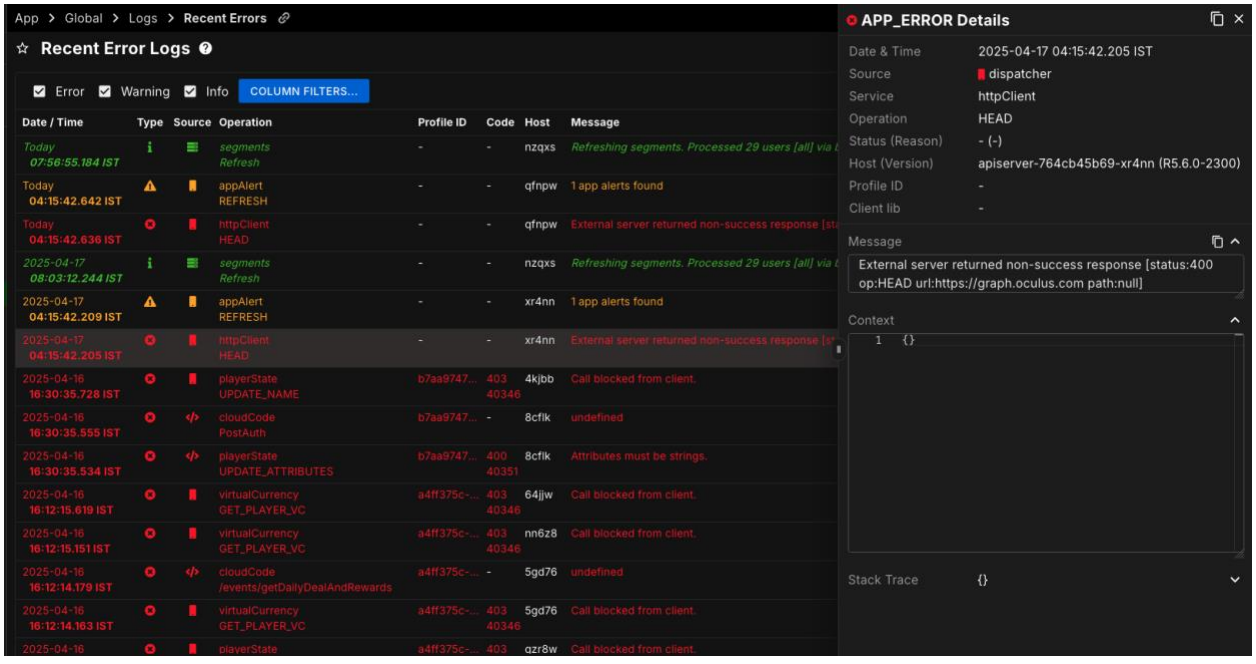
As with script debugging, authenticate with a player before executing requests. Then, select the desired service and operation to perform from the available out-of-the-box requests.

Log Explorer

brainCloud provides the ability to view logs and events occurring on your server, similar to PlayFab's PlayStream Monitor. You can find this dashboard in the Global -> Logs menu.

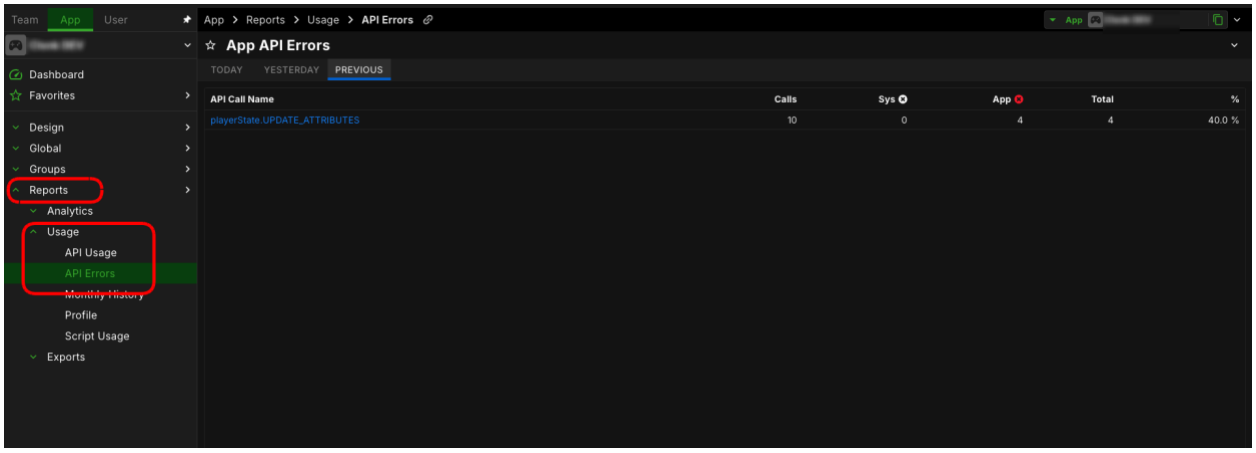


You can select one of these logs to view more information.



This is invaluable for viewing how APIs are preforming when your game goes live as its often hard to cover all test-cases while debugging through the portal.

You can also see error logs summarised by API call from the API Errors dashboard found in the Report -> Usage menu.



Clicking on the API call in this dashboard shows you all the instances of errors along with their payloads and Stack Trace of the error.