Using Azure Functions REST and KUDU API for custom requirements

Azure function is an amazing tool on Azure Cloud that has not only allowed scalability, and a modular architectural alternative for the native apps, but also a clean and wise way to migrate several overhead tasks out of our own main server and deploy it on cloud, making our app nearly ‘Server less’. While using Azure Function is not ABCD, it is neither even Rocket science. Azure function has number of features and use cases; out of which many conventional ones are clearly explained and briefed in their [official documentations](https://docs.microsoft.com/en-us/azure/azure-functions/). So, this post is neither about the basics of Azure functions nor about its utility in conventional ways. This post will describe several other use cases which are not mentioned anywhere or are very vaguely mentioned. Instead of Azure Portal and Azure Functions CLI, this post would rather focus on the API used for various operations of Azure Functions. The basic set of API for Azure Function are mentioned in their official Github Repository [here](https://github.com/projectkudu/kudu/wiki/Functions-API).

(Prerequisites

It is expected that reader should know about

1. Azure cloud, Apis, Postman and C#.
2. The basic understanding about azure function, its working and triggers.)

Let’s get started,

Every app’s business logic is different and so are the requirements associated with them. Here are **two** use cases of azure functions for few custom (yet might frequently originating) requirements.

“My Azure Function is Timer Triggered, but I also want to invoke it manually – through my app”

This requirement basically combines the ability to have two triggers in one function, first, the Time Trigger and second the Http Trigger (or manual trigger).

Why Not the Easy Answer? Simply make two different functions, one, Http Triggered and second. Timer Triggered!

Quick reason:

* Suppose there are 20 timer triggered functions which I want to trigger manually as well, the total functions which I will have to maintain will become 40!
* Thus new function – means more code maintenance complexity, extra bucks as well as extra implementation time.
* On the contrary, having an ability to invoke a time triggered function manually easily abolishes all the potential issues.

*Okay, so what’s another approach?*

We can attain this using azure function REST API. (Thanks to Tom Sun, who discovered this by himself and answered [here](https://stackoverflow.com/a/46949126/2828434) saving the precious hours of research and complexity.)

Assuming we have a Time Triggered Azure Function deployed on portal, which we want to invoke via our app. We can do so by this API URL format,

https://{{azurefunctionapp}}.azurewebsites.net/admin/functions/{{functionname}}

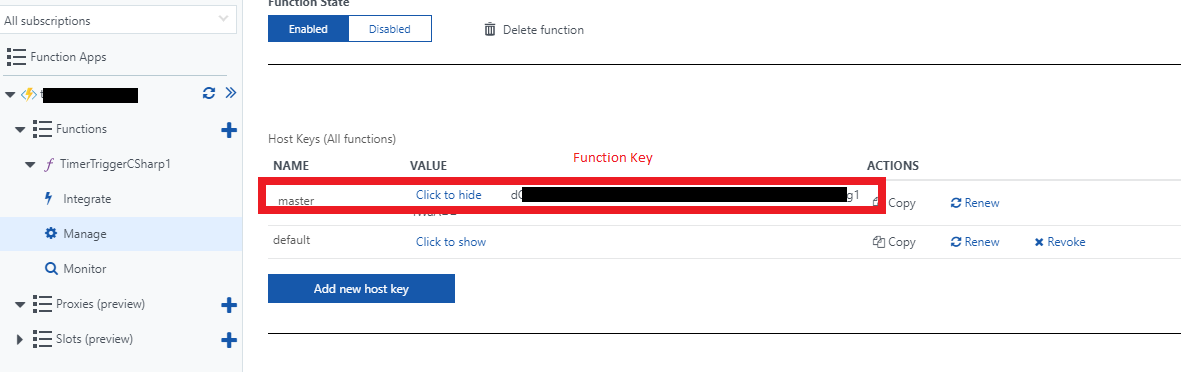
where:

azurefunctionapp is the App Name of Azure Function and

Functionname is the Function Name of Time Triggered Azure Function.

We also need to send an authentication key as a proof. For Authentication, we require X-Functions-Key which can be obtained by following these steps:

* Login to Portal and navigate to the function.
* Click ‘Manage’ tab inside your Function blade, you’ll see two grid containing Host Keys (with name \_master and/or default).
* Copy any one of the key – and the copied one is exactly your ‘X-Functions-Key’



X-Functions-Key for function

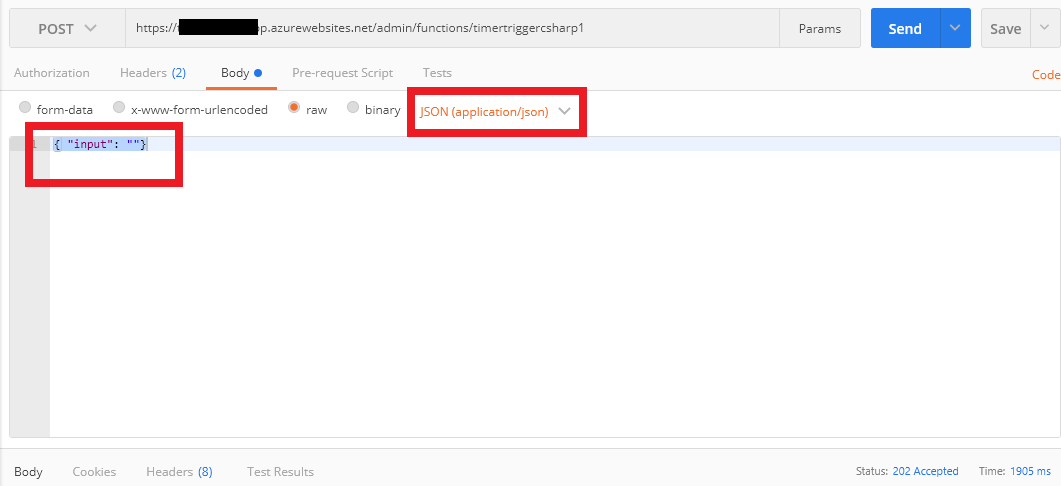
Great! We have API URL and an authentication key – let’s fire!

* *Distort Space Time Continuum,* I mean, open Postman
* Make the Request ‘POST’, paste the API Url in Request URL region,
* Add header “x-functions-key” and hit Send

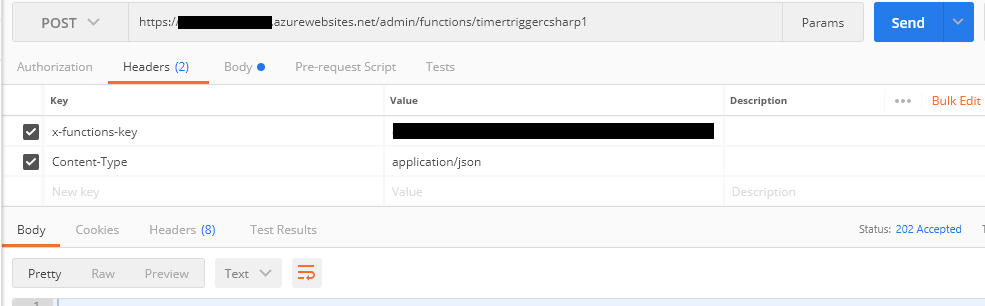
Boom, *it still didn’t reply 202!*

What we need to add along with this request is an empty Application/Json body.

* Go to Body Tab> Select radio button labelled raw
* Chose Application/Json and add {“input”:””}
* Hit send again, and you will get ‘202 Accepted’ response.



Empty input as Application/Json



Trigger function using API

*So did my function really run?*

To verify, head back to portal and verify from the ‘Monitor’ tab. You’ll notice, your function did run and it even didn’t affect any of your original time triggers neither function. Sweet, isn’t it?

“I want to change the scheduled Time for my Timer Triggered Azure Function dynamically via my app”

Each timer triggered azure function has a CRON schedule in its ‘function.json’ file (or in Integrate tab in portal) on which the function triggers. What if the CRON schedule is dynamic and you want it to set dynamically via your app. Say, for example, an Email Sending code deployed on Azure was scheduled to run at 3.00 am every day, but now it needs to run at specific different times on different days.

Why not the Easy Answer? Use other schedulers and deploy it on your server like Quartz OR whenever need arise, open portal and change schedule OR change ‘function.json’ and deploy every time!

Quick Reasons:

* If the one who is going to decide and set schedule is the one who is not aware about the backend processes (and azure function) but simply wants his app to work, then in this case, the easy answer won’t work.
* Using schedulers like Quartz is definitely a reasonable idea but it has its own limitations. It does not trigger on scheduled time when server goes to inactive state (sleep state). Therefore a server less approach like Azure Function fits in well.

Okay, so what’s another approach?

As mentioned above, function.json is the file which stores the CRON schedule for the function. What if somehow, we could replace this file via API with the new file with new schedule? Yes, this is possible using KUDU APIs. (Thanks to Alexey Rodionov, who shared his answer [here](https://stackoverflow.com/a/45575629/2828434) and made me aware that something like KUDU API even exists.)

Basically to change the schedule of Azure Function via KUDU API, we will have to follow these steps:

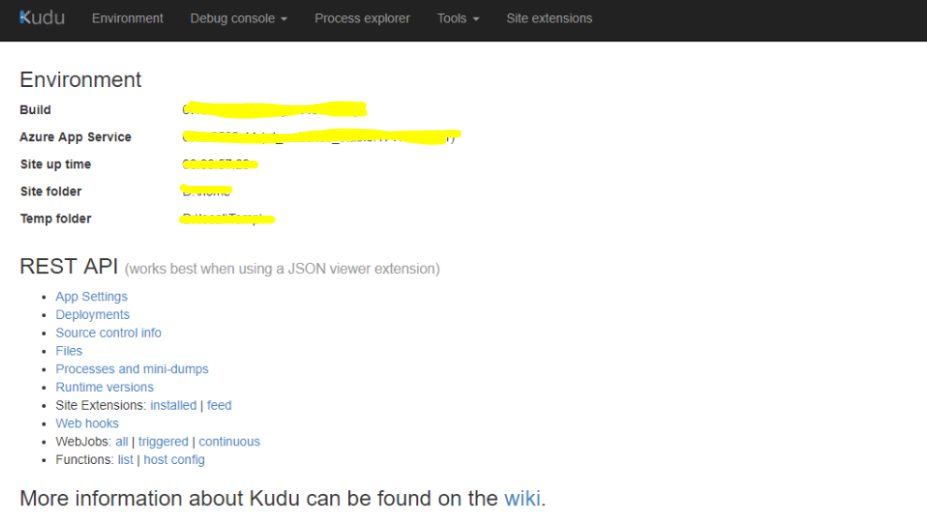
1. Replace the existing function.json file on portal with the new function.json file.
2. Make the new CRON schedule trigger in sync so that function actually triggers on new time.

Let’s do it step wise-

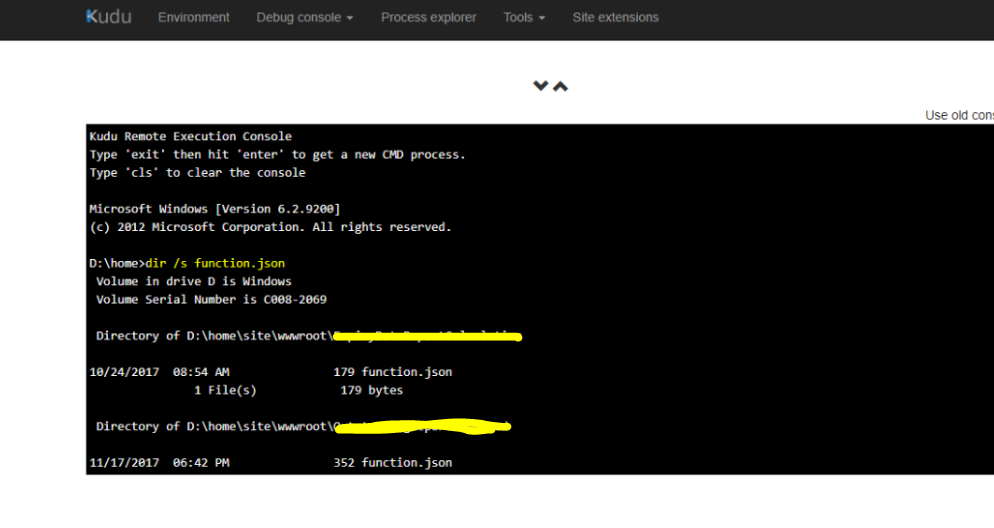
Step 1: Replace the existing function.json on portal with the new function.json file.

Here, we must know where the existing function.json is situated in cloud. To do so,

1. Open Portal, navigate to Function App> Platform Features>Advanced Tools (Kudu)
2. KUDU page will open as shown in the screen shot. Navigate to Debug Console>CMD
3. In command prompt, type command >dir \s function.json. This command will return path where in this file would be located as shown in screenshot.



The KUDU page



Path of Function.json from CMD

Now you have the path of function.json, let’s replace this file with the new file.

* Move Satellites into position, I mean open Postman.
* We are sending PUT request with this URL of this format: https://{functionAppName}.scm.azurewebsites.net/api/vfs/{pathToFunction.json}
* For authorization we will have to use Authorize Tab>Basic Auth. Fill out the Username and Password which you use to deploy the function on portal. You can get the same by downloading the publish profiles for function app.
* Along with these, we’ll have to send another parameter as header i.e Application/Json whose value is: {if-match”,”\*”}. In body new function.json will be added.
* Hit send.
* Go back to portal to verify whether function.json has been updated or not.
* So it’s still we alone who knows that function time has changed but the busy trigger handlers and queues are still unaware about it. To make them aware and sync new trigger with the function, we will have to send one more API request, this time a POST request over  https://{functionAppName}.scm.azurewebsites.net/api/functions/synctriggers

That’s all, you have just dynamically set and synced the new time for your timer triggered azure function. Cool isn’t it?

Thus these were two use cases of using Azure Function API in custom requirements. Do you have any custom requirement related to azure function and are you still in search of a suitable solution for it? Let me know by posting an issue and I’ll try to find the best possible solution.

You are most welcome to read other articles on my Github Repository [here](https://github.com/karandesai28/Learning-Saturday-Blogs).