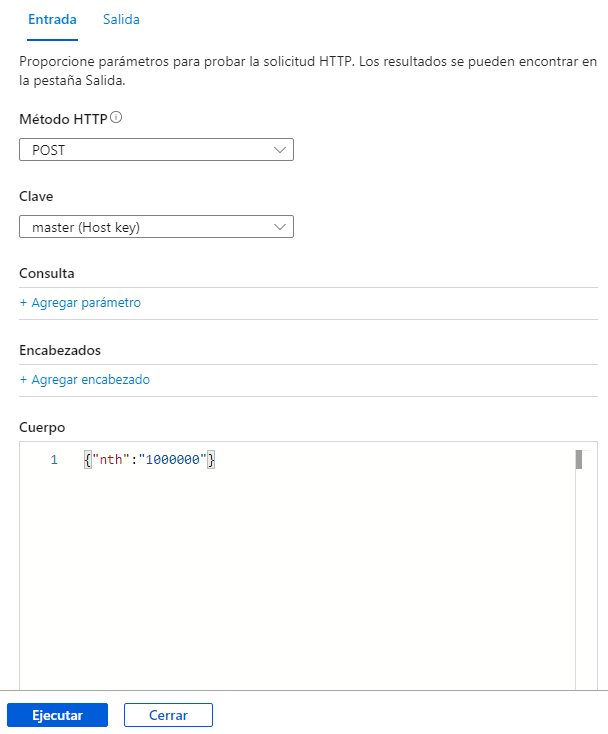
**Lab 10 ARSW**

**Camilo Fajardo – Andrea Durán**

**Punto 4**

Prueba de function en el portal de Azure

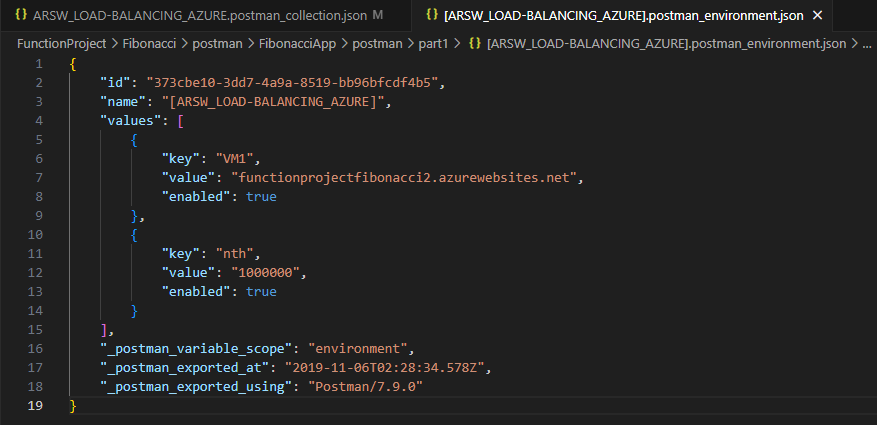


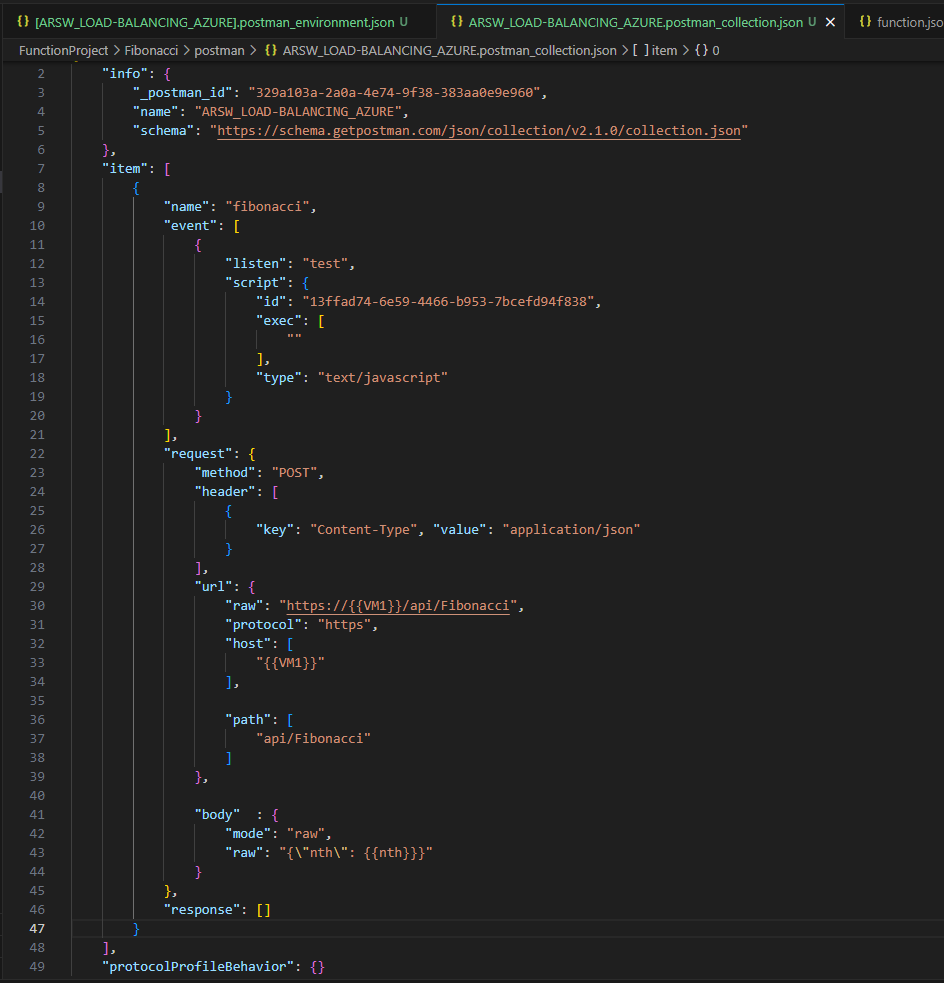
**Punto 5**

**Cambio del JSON:**

En resumen, los cambios fueron:

* Modificación del valor de VM1 en el archivo de environment, cambiándolo por "functionprojectfibonacci2.azurewebsites.net" que hace referencia al servicio de Azure
* Cambio del verbo HTTP (ahora se usa POST), la URL, el host y el protocolo (ahora se usa https) en al archivo en el request del archivo collection
* Creación de una etiqueta “body” en donde especificamos cual será el cuerpo del POST en el archivo collection.
* Creación de un header diciendo que el contenido del cuerpo será JSON.

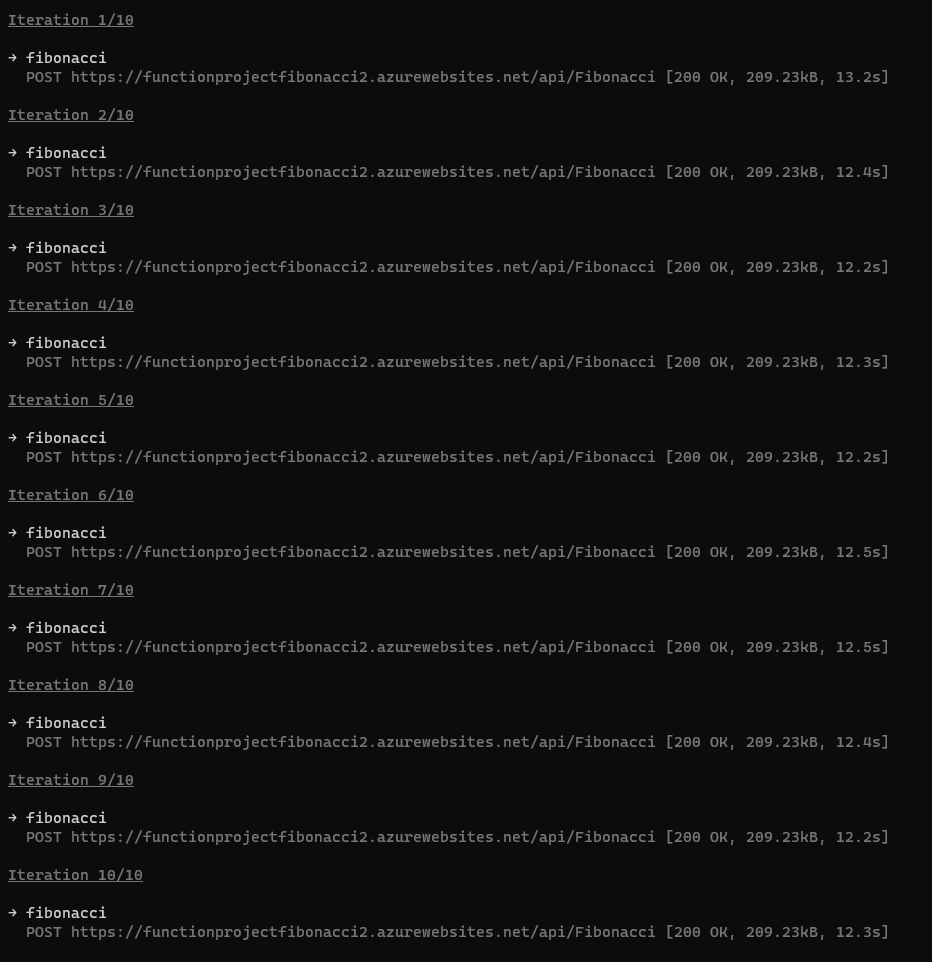


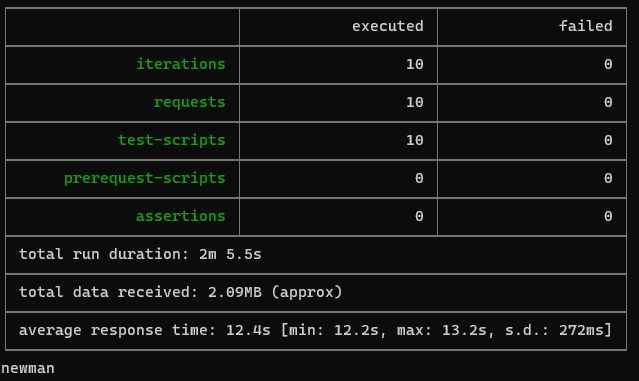


**Envío de 10 peticiones concurrentes:**

Corremos el siguiente comando de tal manera que se envíen 10 peticiones concurrentes:

“newman run ARSW\_LOAD-BALANCING\_AZURE.postman\_collection.json -e [ARSW\_LOAD-BALANCING\_AZURE].postman\_environment.json -n 10”

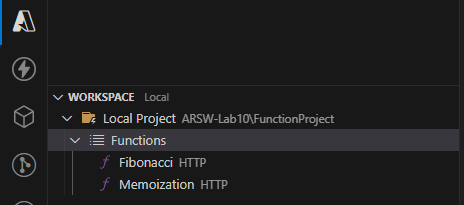


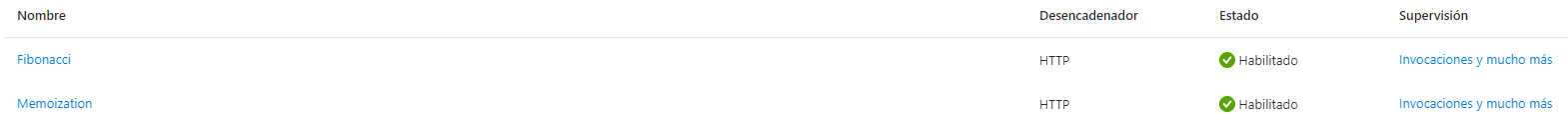


Al ver los resultados de la Function App al hacer 10 llamados concurrentes en el componente, podemos decir que el tiempo de ejecución en comparación a las máquinas virtuales del laboratorio pasado es mucho mejor, esto se podría atribuir a la propiedad Serverless que tienen estas Function App, en donde los recursos que la App necesite se le asignaran dinámicamente en su runtime.

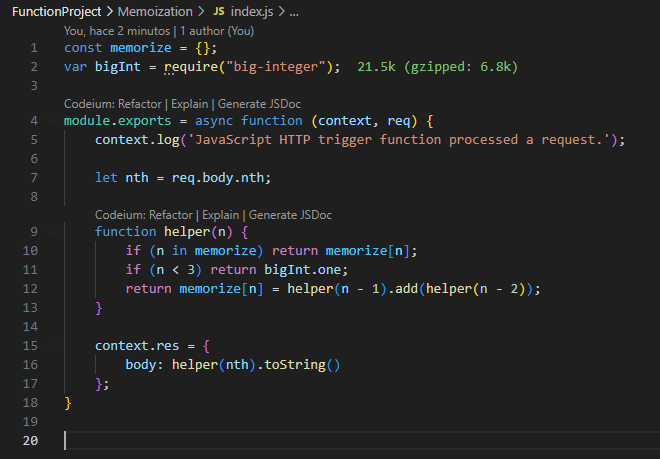
**Punto 6**

Creamos una nueva fuction en el proyecto que llamamos memoization:





Para usar la técnica momization usamos la siguiente implementación, donde en esencia, guardamos los resultados anteriores. Esto lo hacemos en el archivo index.js de la función:

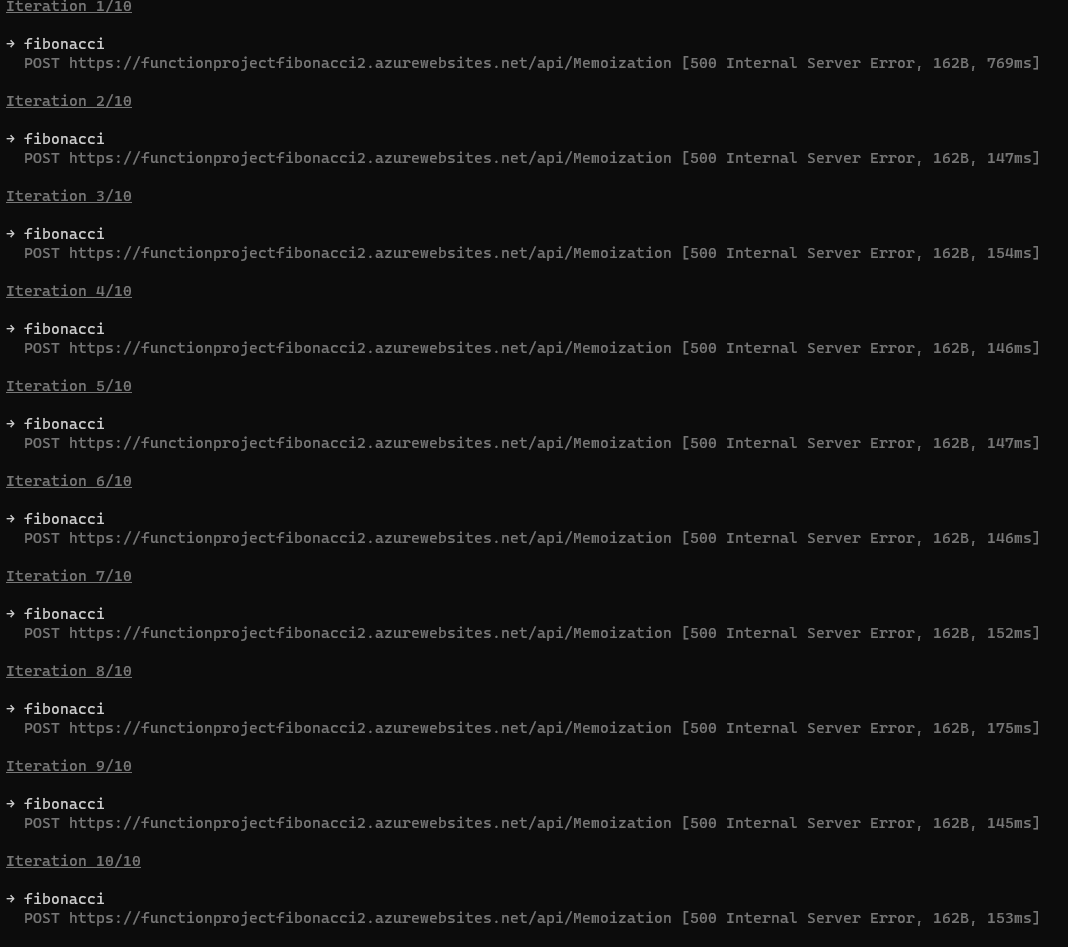


Creamos los archivos de postman y lo único que hacemos es modificar la URL de collection



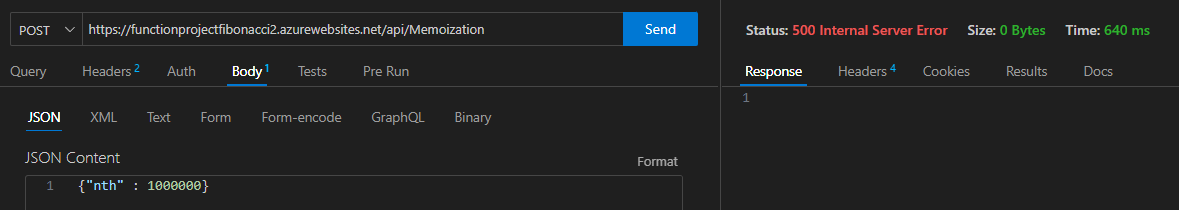
Procedemos a hacer las pruebas pertinentes

* Primeras pruebas

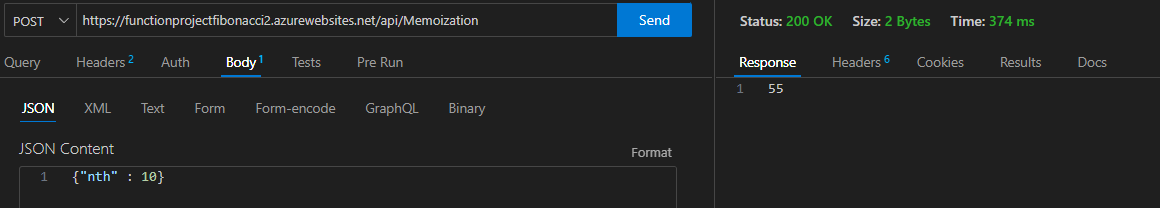


Como podemos ver en la imagen, todos los los request arrojan un error 500, que nos indican un error de servidor. Esto se puede deber a la falta de recursos para hacer las peticiones, por lo que comprobamos con un nth más pequeño.

Salida con nth = 1000000

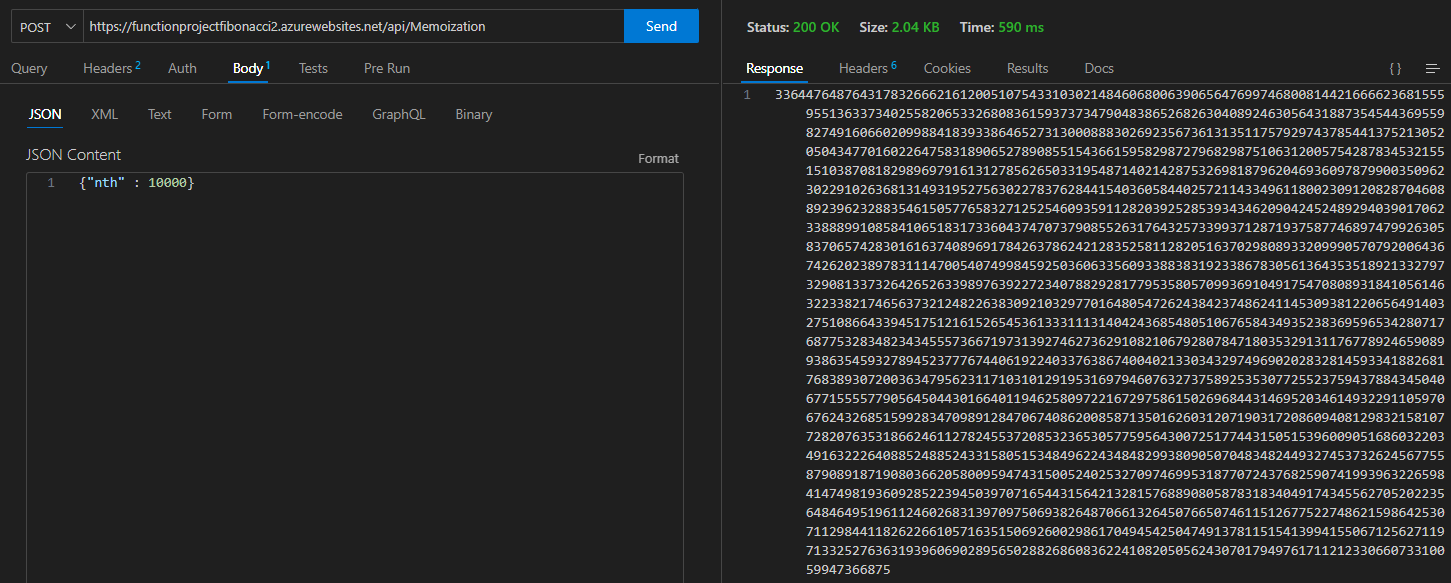


Salida con nth = 10

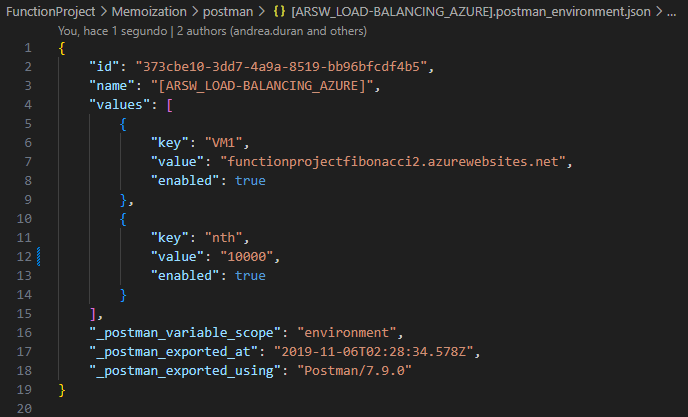


Por lo tanto, podemos decir que las peticiones fallan cuando el número que estamos pidiendo es muy grande. Según nuestra investigación, el problema es que JavaScript tiene un límite de profundidad en la recursividad de alrededor de 10000 llamados, por lo que probamos hacer un llamado con el número 10000 desde Newman para ver el desempeño del nuevo algoritmo.

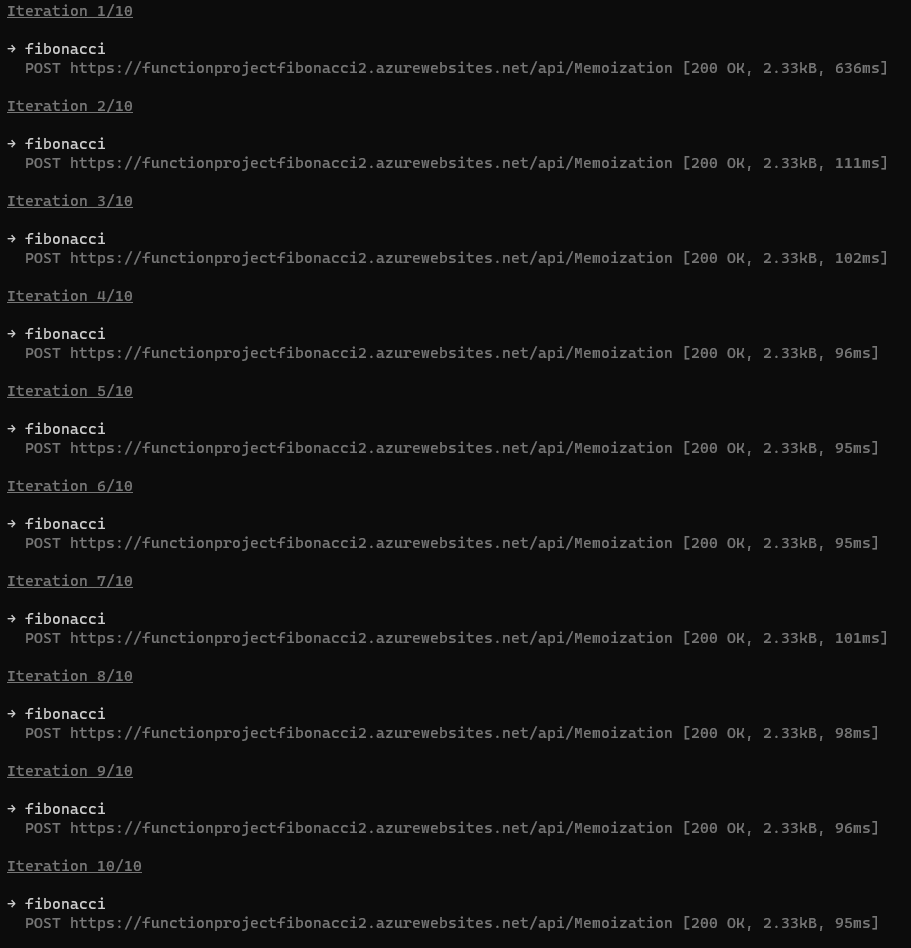
Probamos este nth y vemos que funciona

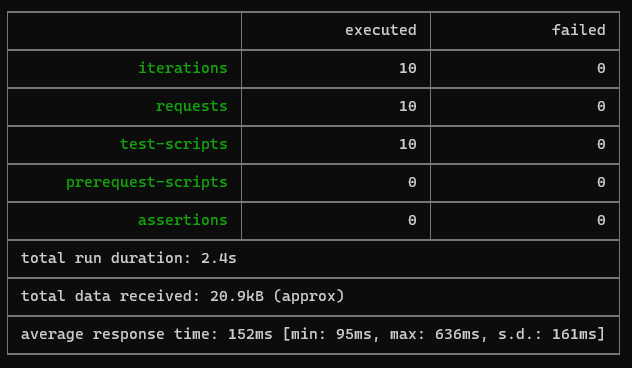


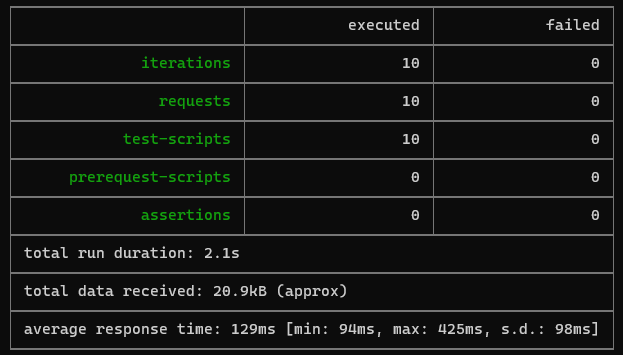
Así que lo cambiamos en el postman en el archivo de environment



Procedemos a hacer las pruebas con postman y vemos que todo es correcto

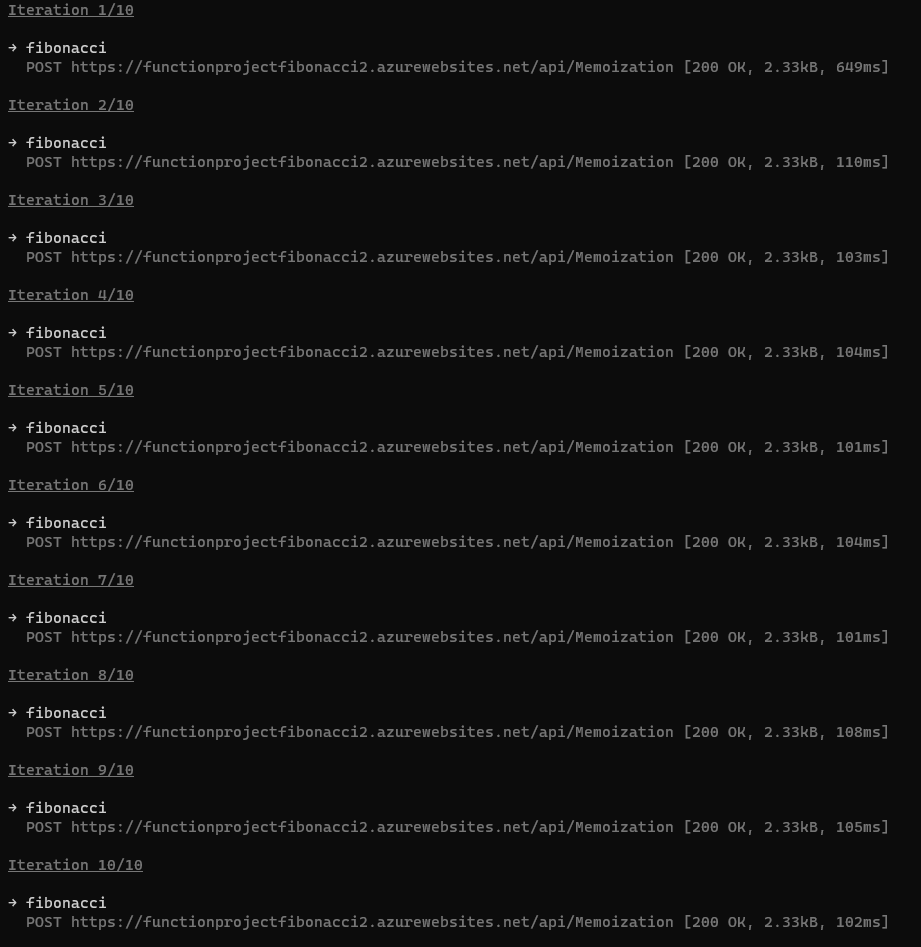


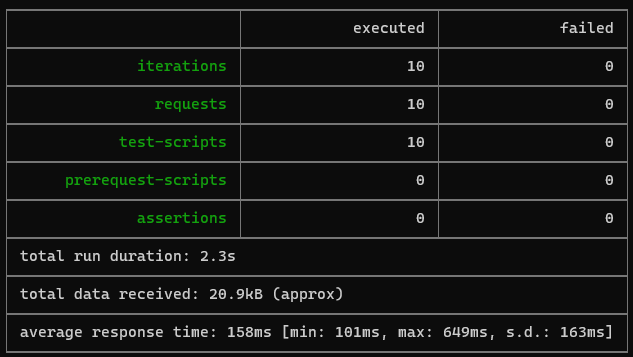




Corrimos varias veces el comando y en su reporte, vimos que la primera petición a la API es la que se demora más, mientras que las siguientes tienen una mejora considerable en su tiempo de ejecución.

* Pruebas después de haber esperado 5 minutos





Podemos ver como al pasar el tiempo el primer llamado de la función crece en 200 milisegundos, demorándose el mismo tiempo que se demoró en el primer llamado hace 5 minutos.

Aunque 200 milisegundos no parezcan mucho en este caso podríamos atribuirle ese tiempo extra a la pérdida de la memoria debido a la propiedad de Function TimeOut que tienen las Function Apps, en donde cuando no se hacen peticiones por un tiempo, la memoria que esté usando el componente será borrada.

**Preguntas**

* ¿Qué es un Azure Function?

Azure Functions es un servicio de computación sin servidor que permite ejecutar código en respuesta a eventos sin tener que aprovisionar ni administrar explícitamente una infraestructura. En lugar de preocuparse por la infraestructura, puedes centrarte en el código que se ejecuta en respuesta a los eventos que se producen en tu aplicación o en los datos que se encuentran en tus servicios. Azure Functions es ideal para tareas que necesitan escalar en respuesta a la demanda de eventos, como la manipulación de archivos en Azure Storage, el procesamiento de mensajes en una cola de Azure Service Bus, o la respuesta a solicitudes HTTP.

* ¿Qué es serverless?

Serverless es un modelo de ejecución de aplicaciones en la nube en el que los desarrolladores no tienen que preocuparse por la administración de servidores. En lugar de eso, pueden centrarse en escribir código y el proveedor de la nube se encarga de la infraestructura subyacente, incluyendo la administración de servidores, el escalado automático, la monitorización y la facturación. Azure Functions es un ejemplo de un servicio sin servidor.

* ¿Qué es el runtime y que implica seleccionarlo al momento de crear el Function App?

El runtime de una Azure Function es el entorno de ejecución en el que se ejecuta el código de la función. Al seleccionar un runtime, estás especificando el lenguaje de programación en el que se escribe tu código de función y la versión del entorno de ejecución que se usará para ejecutar ese código. Los runtimes disponibles para Azure Functions incluyen .NET, Node.js, Python, Java, PHP y PowerShell. Además, en el run time también incluimos el Azure Function Host, el cual es el responsable de manejar el ciclo de vida de las funciones.

* ¿Por qué es necesario crear un Storage Account de la mano de un Function App?

Azure Functions requiere un Storage Account para almacenar el código de la función, las claves de acceso y otros datos relacionados con el servicio. Cuando tu Function App se aloja en el plan de consumo o en el plan premium, tu código de función y los archivos de configuración se almacenan en Azure Files en el Storage Account vinculado. Si eliminas este Storage Account, el contenido se elimina y no se puede recuperar. Por lo tanto, es importante gestionar cuidadosamente el acceso a los Storage Accounts utilizados por las Function Apps.

* ¿Cuáles son los tipos de planes para un Function App? ¿En qué se diferencias?, mencione ventajas y desventajas de cada uno de ellos.

Azure Functions ofrece tres tipos de planes de alojamiento:

1. Plan de consumo: En este plan, las instancias del host de Azure Functions se agregan y eliminan dinámicamente en función del número de eventos entrantes. La facturación se basa en el número de ejecuciones, el tiempo de ejecución y la memoria utilizada. Este plan es ideal para aplicaciones con tráfico irregular o para aplicaciones que no necesitan estar en línea todo el tiempo.
2. Plan Premium: Este plan proporciona características adicionales como instancias siempre listas y precalentadas para reducir los inicios fríos en las funciones. Las instancias siempre listas son instancias preasignadas que no se ven afectadas por el escalado, y las instancias precalentadas son un búfer a medida que escalas debido a eventos HTTP. La facturación para el plan Premium se basa en el número de núcleos y la memoria asignada a través de las instancias. Este plan es ideal para aplicaciones con tráfico constante o para aplicaciones que necesitan estar en línea todo el tiempo.
3. Plan Dedicado (App Service): Este plan es similar al plan Premium, pero ofrece un mayor control y aislamiento. Puedes escalar manualmente agregando más instancias de VM o habilitar el escalado automático. También puedes escalar verticalmente eligiendo un plan de App Service diferente. Este plan es ideal para aplicaciones que necesitan un alto rendimiento y un control total sobre el entorno de ejecución.

* ¿Por qué la memoization falla o no funciona de forma correcta?

La memoization es una técnica de optimización que se utiliza para almacenar los resultados de las funciones costosas para que, si la función se llama de nuevo con los mismos argumentos, pueda devolver el resultado almacenado en lugar de volver a calcularlo. Sin embargo, la memoization no siempre funciona de forma correcta por varias razones:

* Espacio de memoria limitado: La memoization almacena los resultados de las funciones en la memoria. Si tu aplicación tiene un espacio de memoria limitado, puede que no puedas almacenar todos los resultados de las funciones.
* Cambios en los datos: Si los datos que tu función utiliza para calcular un resultado cambian, el resultado almacenado en caché ya no será válido. En este caso, la función debería volver a calcular el resultado, pero la memoization podría hacer que se devuelva el resultado en caché obsoleto.
* Funciones no puras: Si tu función tiene efectos secundarios o depende del estado externo, la memoization podría dar resultados incorrectos. Por ejemplo, si tu función incrementa un contador cada vez que se llama, la memoization podría hacer que el contador se reinicie a cero en cada llamada, lo que podría llevar a resultados incorrectos.
* ¿Cómo funciona el sistema de facturación de las Function App?

Azure Functions utiliza un modelo de facturación basado en el consumo. Esto significa que pagas por la cantidad de recursos que consumen tus funciones, en lugar de pagar por un número fijo de instancias de máquinas virtuales o un número fijo de horas de tiempo de CPU.

En el plan de consumo, la facturación se basa en tres factores:

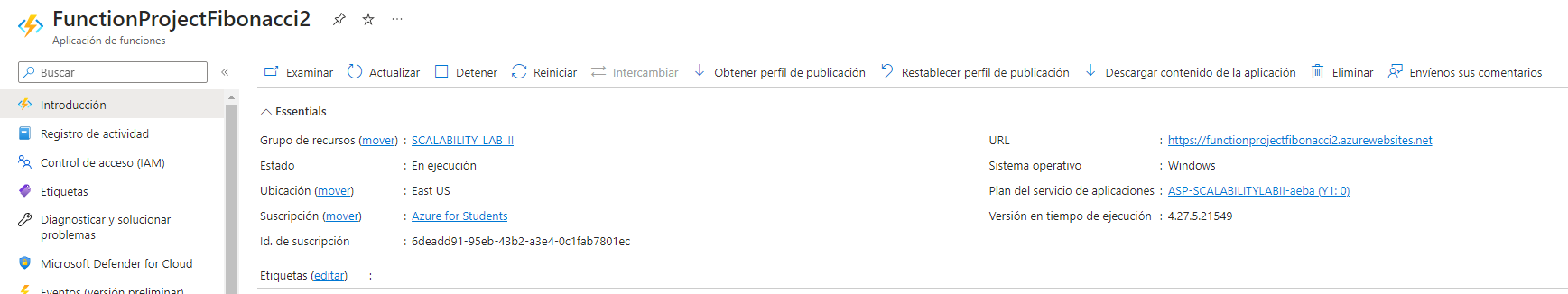
1. Número de ejecuciones: Cada vez que se llama a tu función, se cuenta como una ejecución.
2. Tiempo de ejecución: Cada ejecución de tu función se mide en milisegundos. Pagas por el tiempo total de ejecución de todas tus funciones en un período de facturación.
3. Memoria utilizada: Azure Functions se factura en función de la cantidad de memoria que utiliza tu función. Si tu función utiliza más memoria, pagas más.

Además, también hay costos asociados con el uso de otros servicios de Azure, como Azure Storage y Azure Logic Apps, que podrían ser utilizados por tus funciones.

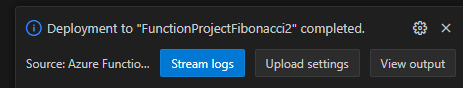
En el plan Premium, la facturación se basa en el número de núcleos y la memoria asignada a través de las instancias. Este plan es ideal para aplicaciones con tráfico constante o para aplicaciones que necesitan estar en línea todo el tiempo.

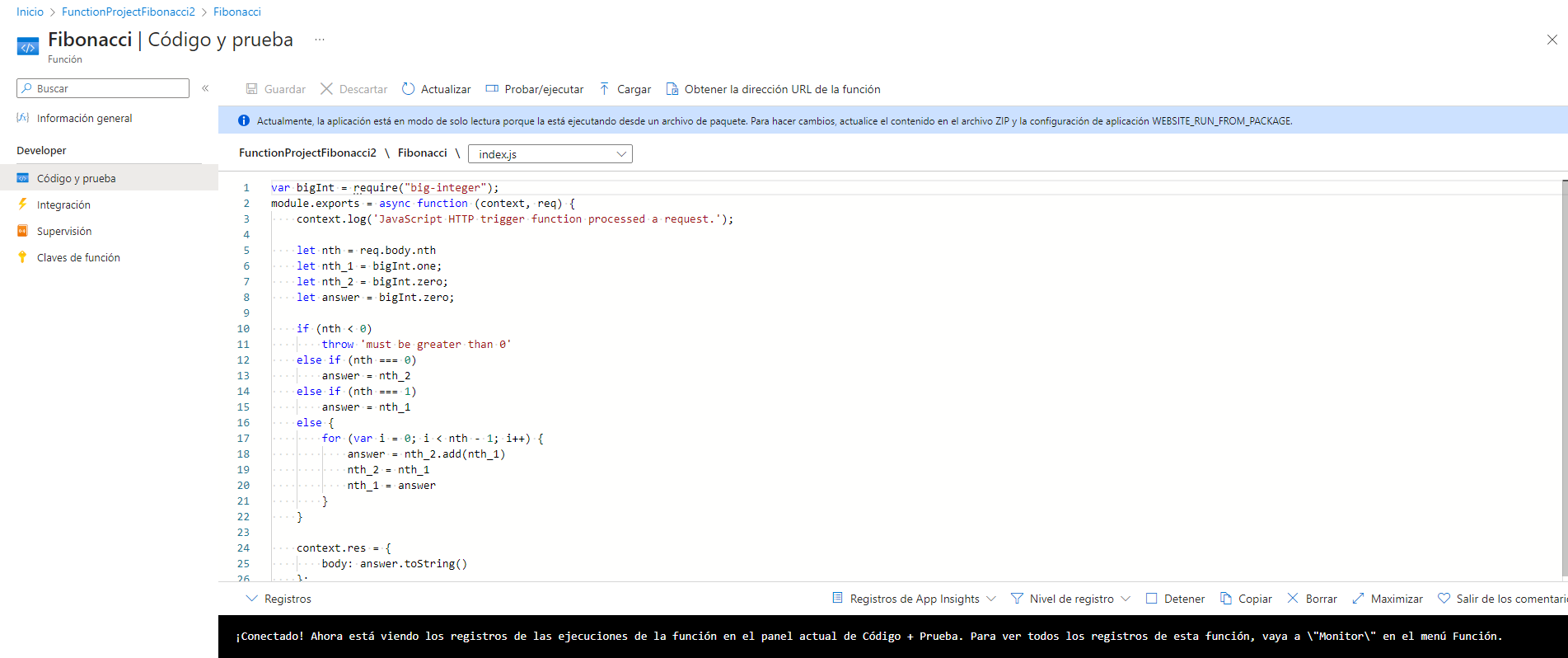
El plan Dedicado (App Service) es similar al plan Premium, pero ofrece un mayor control y aislamiento. Puedes escalar manualmente agregando más instancias de VM o habilitar el escalado automático. También puedes escalar verticalmente eligiendo un plan de App Service diferente. Este plan es ideal para aplicaciones que necesitan un alto rendimiento y un control total sobre el entorno de ejecución

* Informe
  + Creación Fuction

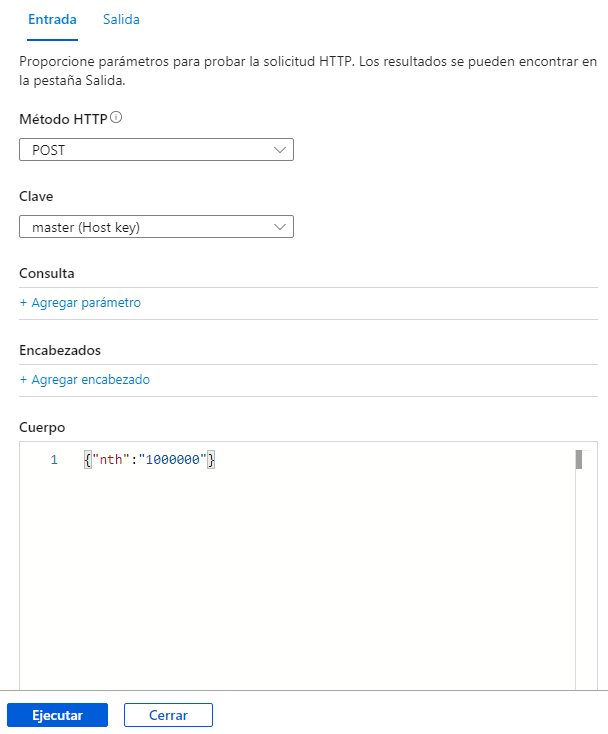


* + Despliegue en Azure

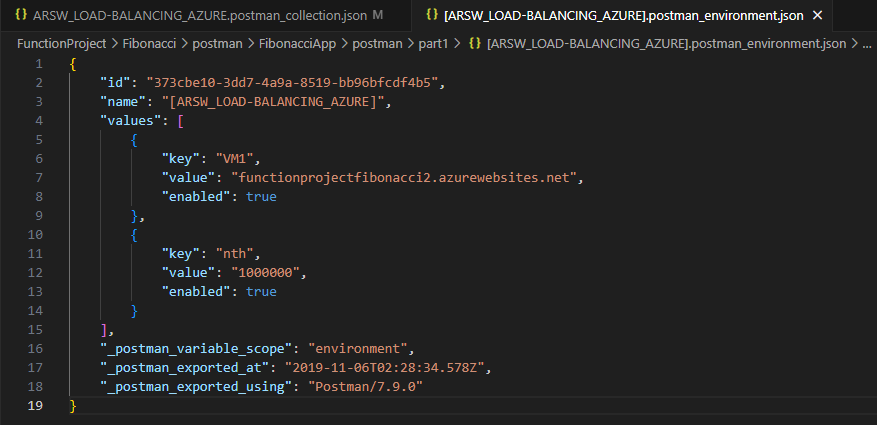


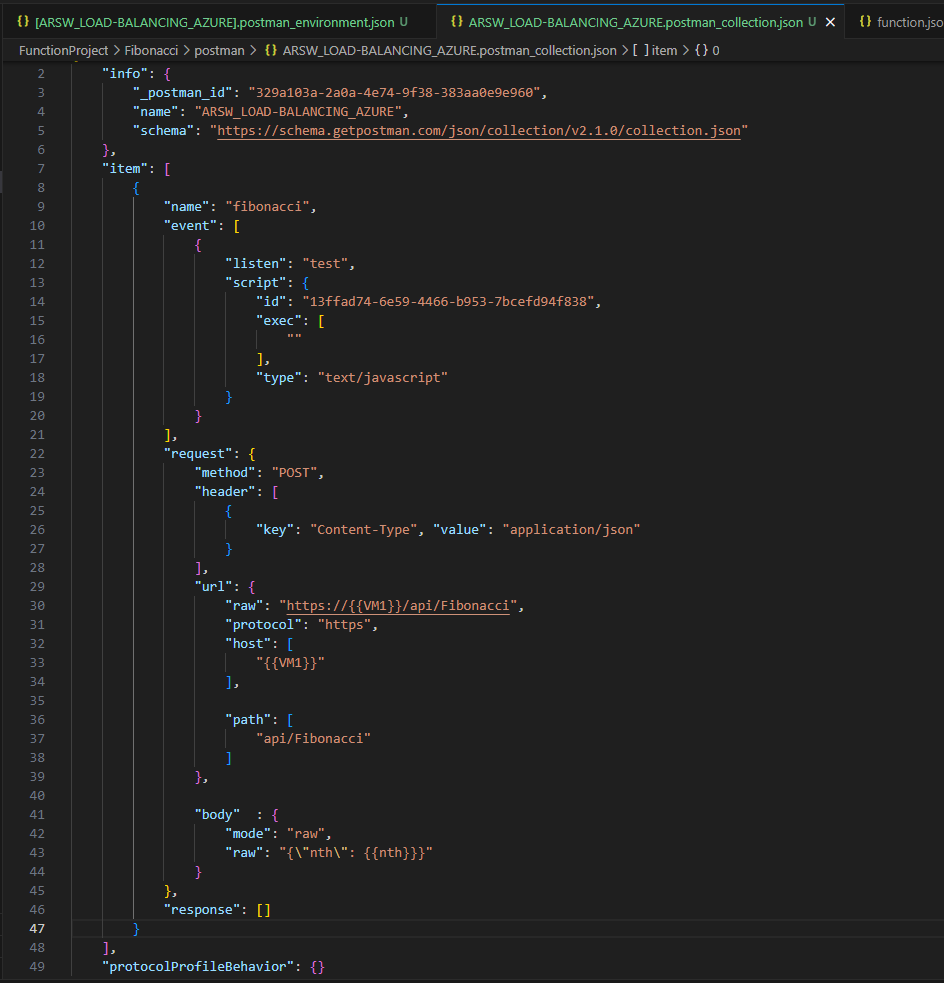


* + Prueba de function

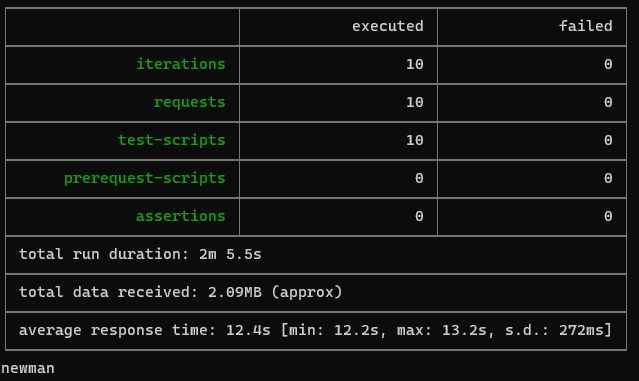


* + Cambio del JSON

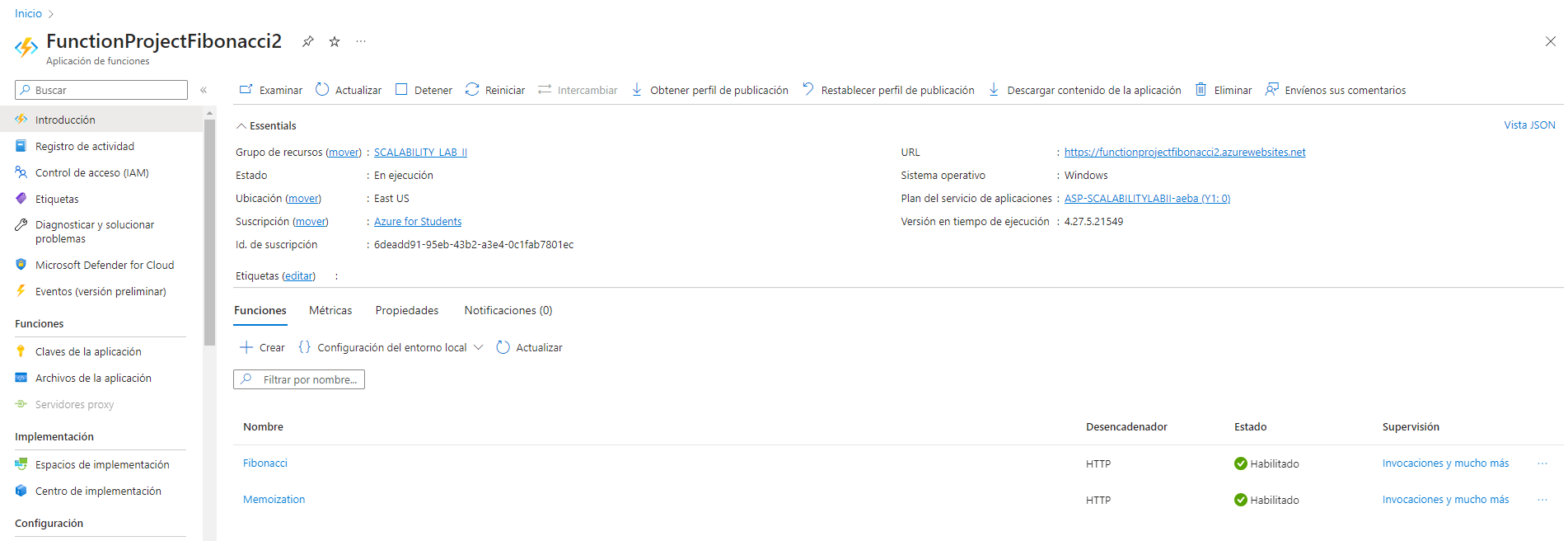




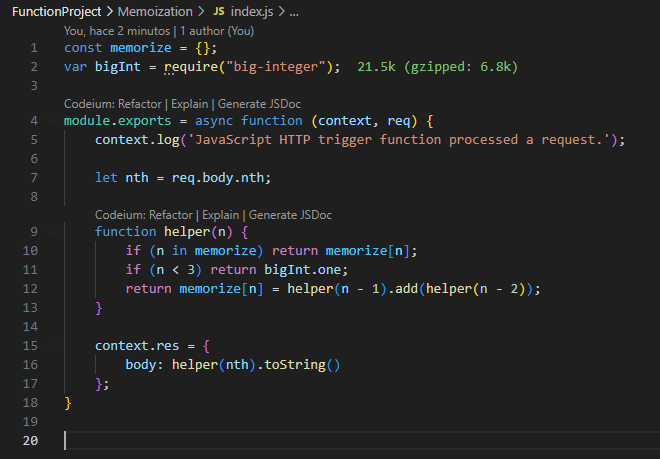
* + Resultado del envío de 10 peticiones concurrentes



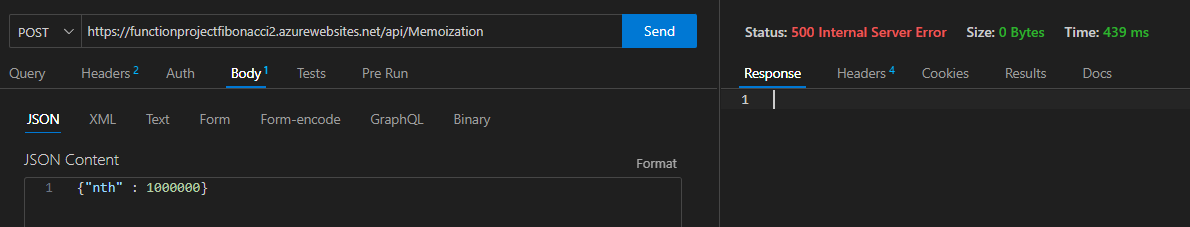
* + Creación de otra función que use memoization

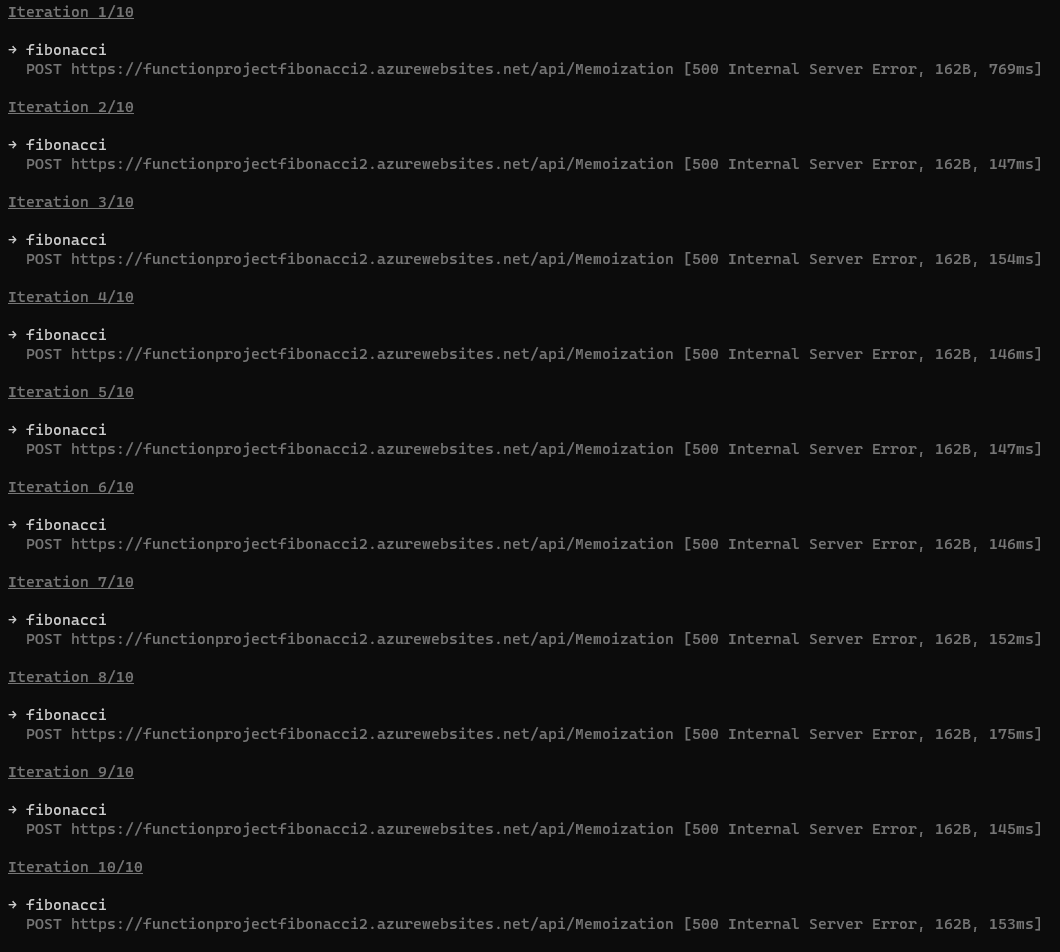


* + Cambio del index.js para que use la técnica memoization

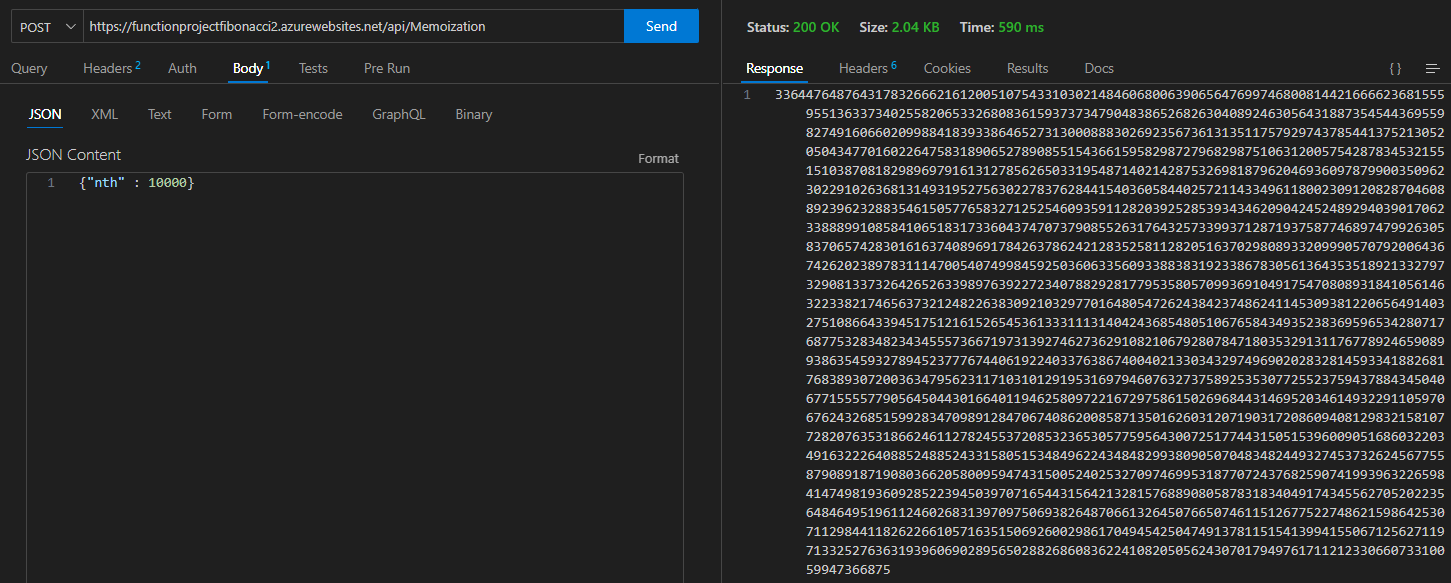


* + Pruebas iniciales: Error 500

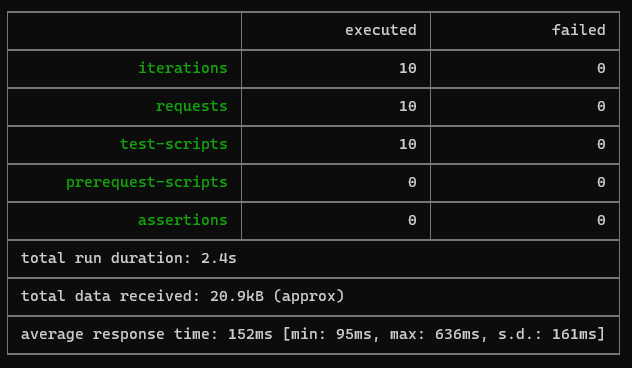


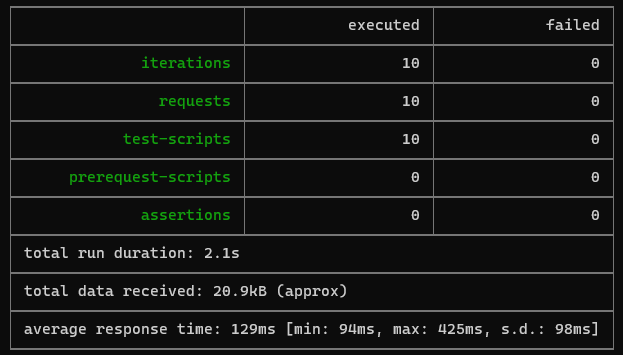


* + Por límite de profundidad de recursividad de Javascript se cambia nth por 10000



* + Resultados exitosos al hacer las pruebas con postman. Podemos ver que a medida que aumentan las iteraciones de las peticiones, el tiempo de ejecución va disminuyendo





* + Resultados exitosos después de 5 minutos. Podemos ver que el tiempo aumenta a la misma cantidad que tuvo la primera iteración hace 5 minutos

