Imagen en blanco y negro

Descripción generada automáticamente con confianza baja

Laboratorio 9

Escalamiento en Azure con Máquinas Virtuales, Sacale Sets y Service Plans

url repositorio:

<https://github.com/ARSW2023-2/ARSW_Lab09.git>

Integrantes:

Luisa Fernanda Bermúdez Girón

Karol Daniela Ladino Ladino

Squad:

Inside Out

Profesor:

Javier Iván Toquica Barrera

Curso:

ARSW – 1

Fecha De Entrega:

17-11-2023

1. **Introducción**

El presente laboratorio tiene como objetivo explorar y comprender las capacidades de escalabilidad en Microsoft Azure, utilizando máquinas virtuales, Scale Sets y Service Plans. A través de una serie de pasos detallados, exploraremos tanto la escalabilidad vertical como horizontal para abordar los desafíos asociados con la ejecución eficiente de una aplicación que calcula números de la secuencia de Fibonacci. Este ejercicio nos permitirá analizar el rendimiento del sistema bajo diferentes condiciones de carga y evaluar estrategias de escalabilidad para mejorar la eficiencia y la capacidad de respuesta.

En la primera parte, nos centraremos en la escalabilidad vertical, explorando el impacto de cambiar el tamaño de una máquina virtual en Azure para abordar la carga concurrente de peticiones de la aplicación Fibonacci. Luego, avanzaremos hacia la escalabilidad horizontal, implementando un balanceador de carga y desplegando varias máquinas virtuales en diferentes zonas de disponibilidad para distribuir la carga de manera eficiente. Este enfoque nos permitirá comparar y contrastar los resultados obtenidos mediante la escalabilidad vertical y horizontal, analizando el rendimiento, la eficacia y los costos asociados.

A lo largo del laboratorio, se realizarán mediciones de consumo de CPU, tiempos de respuesta y pruebas de carga con Newman para evaluar la efectividad de las estrategias de escalabilidad implementadas. Este análisis detallado nos proporcionará información valiosa sobre cómo optimizar y gestionar recursos en entornos de nube para cumplir con los requisitos de rendimiento y escalabilidad de las aplicaciones.

1. **Desarrollo del laboratorio**

**Dependencias**

Cree una cuenta gratuita dentro de Azure. Para hacerlo puede guiarse de esta [documentación](https://azure.microsoft.com/es-es/free/students/). Al hacerlo usted contará con $100 USD para gastar durante 12 meses.



**Parte 0 – Entendiendo el escenario de calidad**

Adjunto a este laboratorio usted podrá encontrar una aplicación totalmente desarrollada que tiene como objetivo calcular el enésimo valor de la secuencia de Fibonnaci.

**Escalabilidad** Cuando un conjunto de usuarios consulta un enésimo número (superior a 1000000) de la secuencia de Fibonacci de forma concurrente y el sistema se encuentra bajo condiciones normales de operación, todas las peticiones deben ser respondidas y el consumo de CPU del sistema no puede superar el 70%.

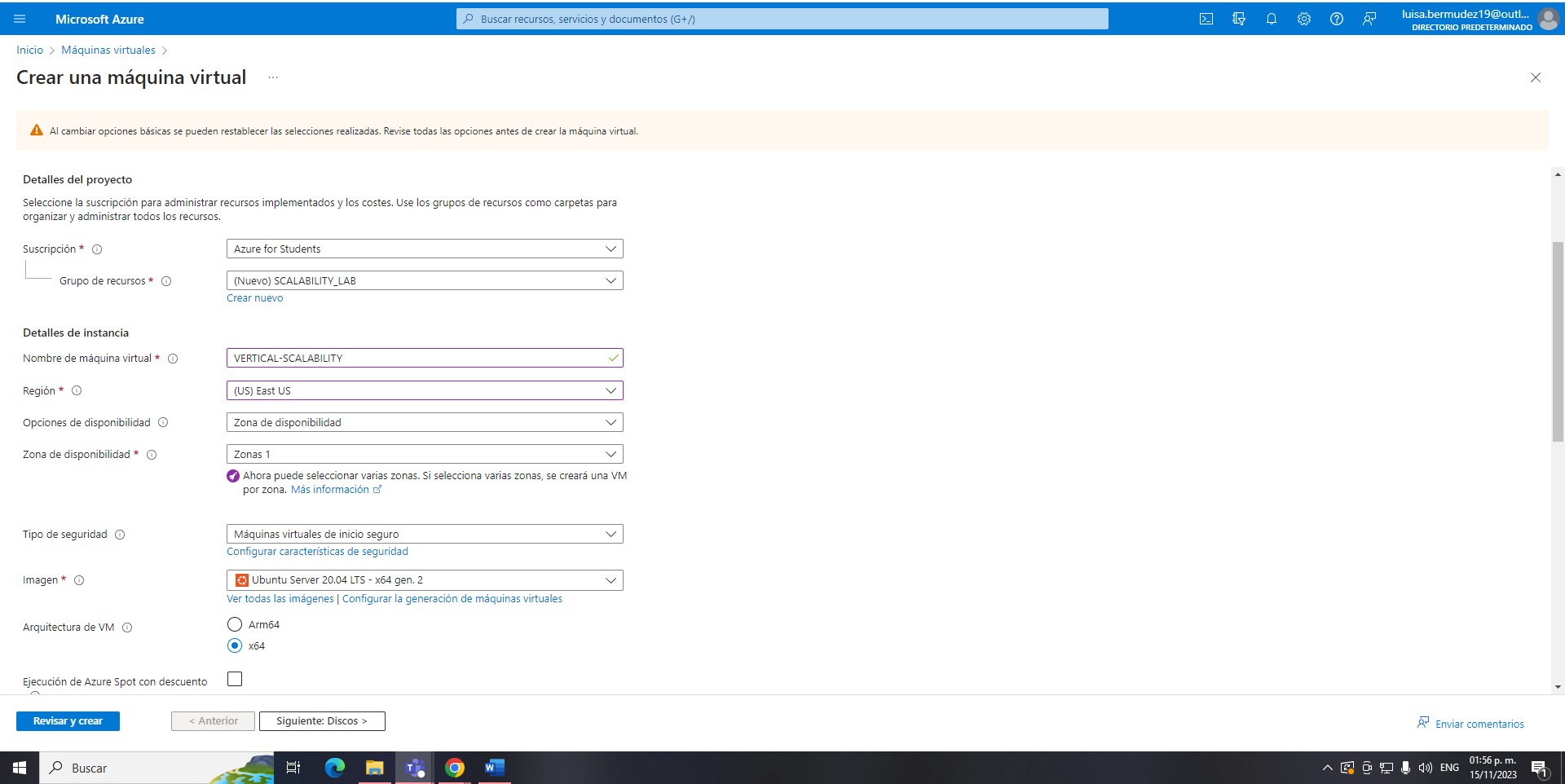
* Tenemos una clase llamada FibonacciService la cual proporciona un método estático llamado getNthNumberInSequence. Este método utiliza la biblioteca big-integer para calcular el enésimo número en la secuencia de Fibonacci. Adicional a esto vemos que el código incluye manejo de casos especiales para 0 y 1, y utiliza un bucle para calcular la secuencia para valores mayores. EL resultado se devuelve como una cadena.
* También contamos con FibonacciApp, el cual crea un servidor que ofrece servicios relacionados con la secuencia de Fibonacci, accesibles a través de rutas específicas:

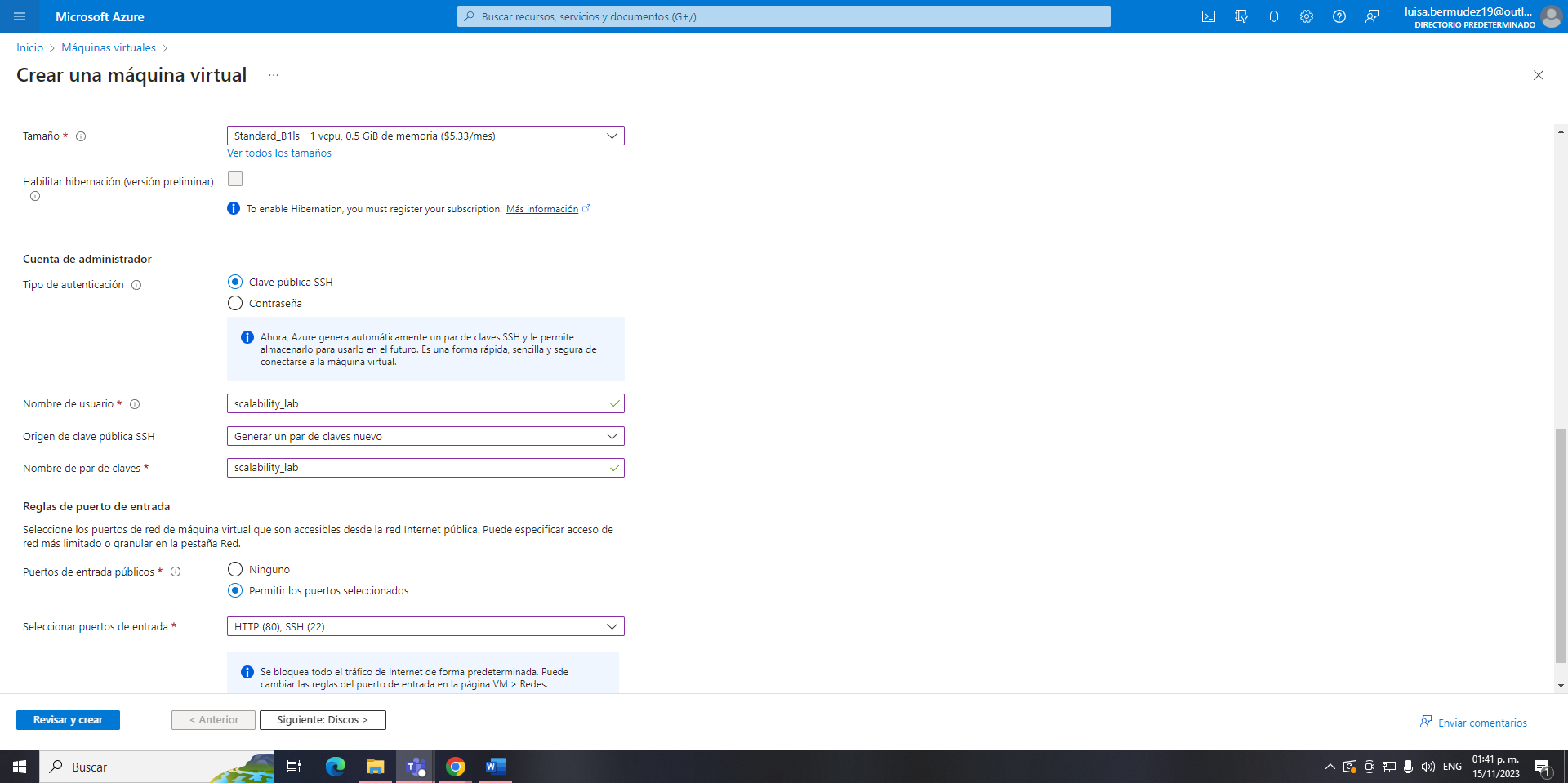
1. **/fibonacci/:nthIn:** utiliza el servicio Fibonacci para calcular el enésimo número en la secuencia y devuelve el resultado como una respuesta al cliente
2. **/:** responde con "Hello World".

[**Parte 1 - Escalabilidad vertical**](https://github.com/DiegoATG/ARSW_LOAD-BALANCING_AZURE#parte-1---escalabilidad-vertical)

1. Diríjase a el [Portal de Azure](https://portal.azure.com/) y a continuación cree una máquina virtual con las características básicas descritas en la imágen 1 y que corresponden a las siguientes:

* Resource Group = SCALABILITY\_LAB
* Virtual machine name = VERTICAL-SCALABILITY
* Image = Ubuntu Server
* Size = Standard B1ls
* Username = scalability\_lab
* SSH publi key = Su llave ssh publica





1. Para conectarse a la VM use el siguiente comando, donde las x las debe remplazar por la IP de su propia VM

ssh [scalability\_lab@xxx.xxx.xxx.xxx](mailto:scalability_lab@xxx.xxx.xxx.xxx)

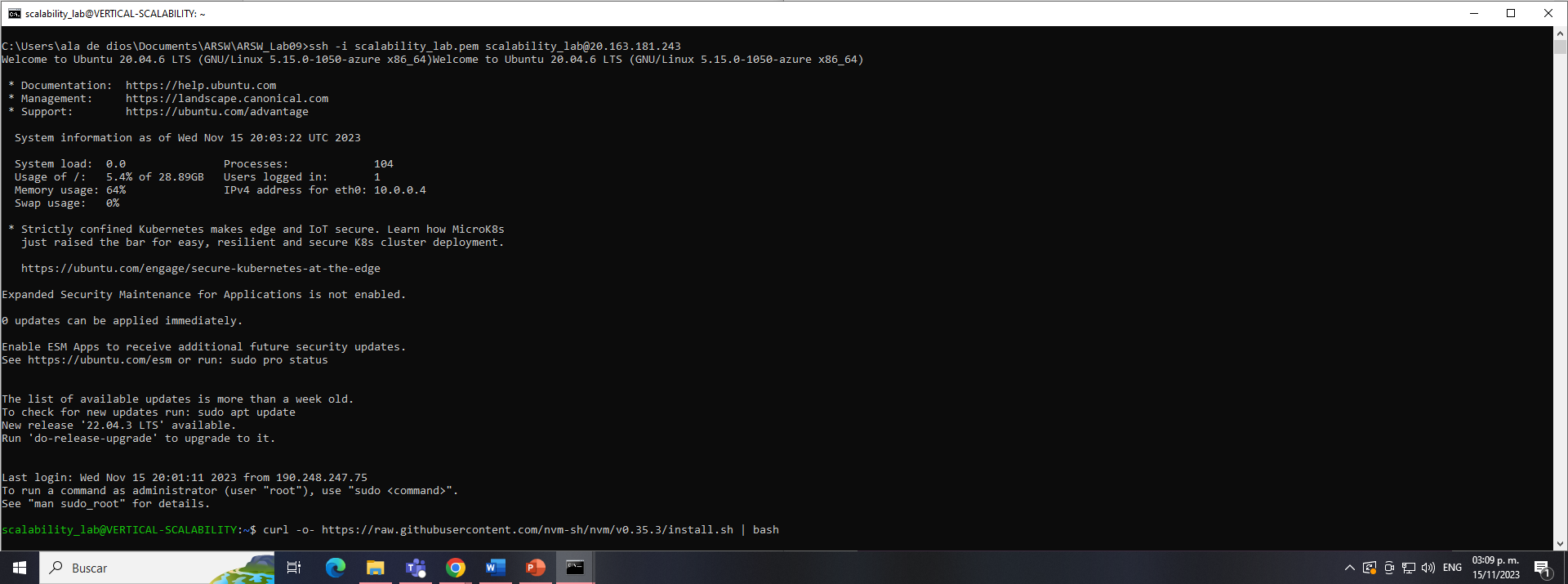
Para conectarnos a la maquina ejecutamos el siguiente comando, el cual contiene la llave privada:

* ssh -i scalability\_lab.pem [scalability\_lab@20.163.181.243](mailto:scalability_lab@20.163.181.243)

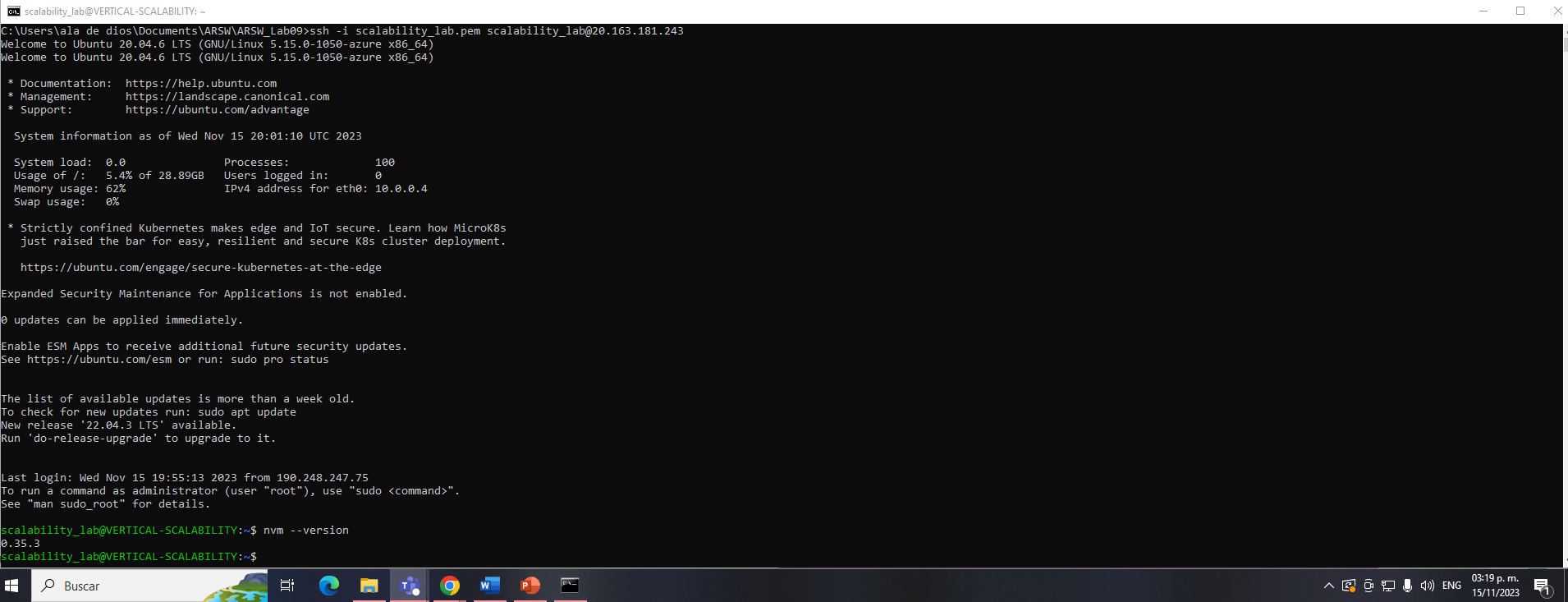
1. Instale node, para ello siga la sección Installing Node.js and npm using NVM que encontrará en este [enlace](https://linuxize.com/post/how-to-install-node-js-on-ubuntu-18.04/).

* Realizamos la instalación del script NVM. Para esto ejecutamos el comando:

curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.35.3/install.sh | bash



* Verificamos que el nvm se haya instalado correctamente. Para esto ejecutamos el comando nvm --version.

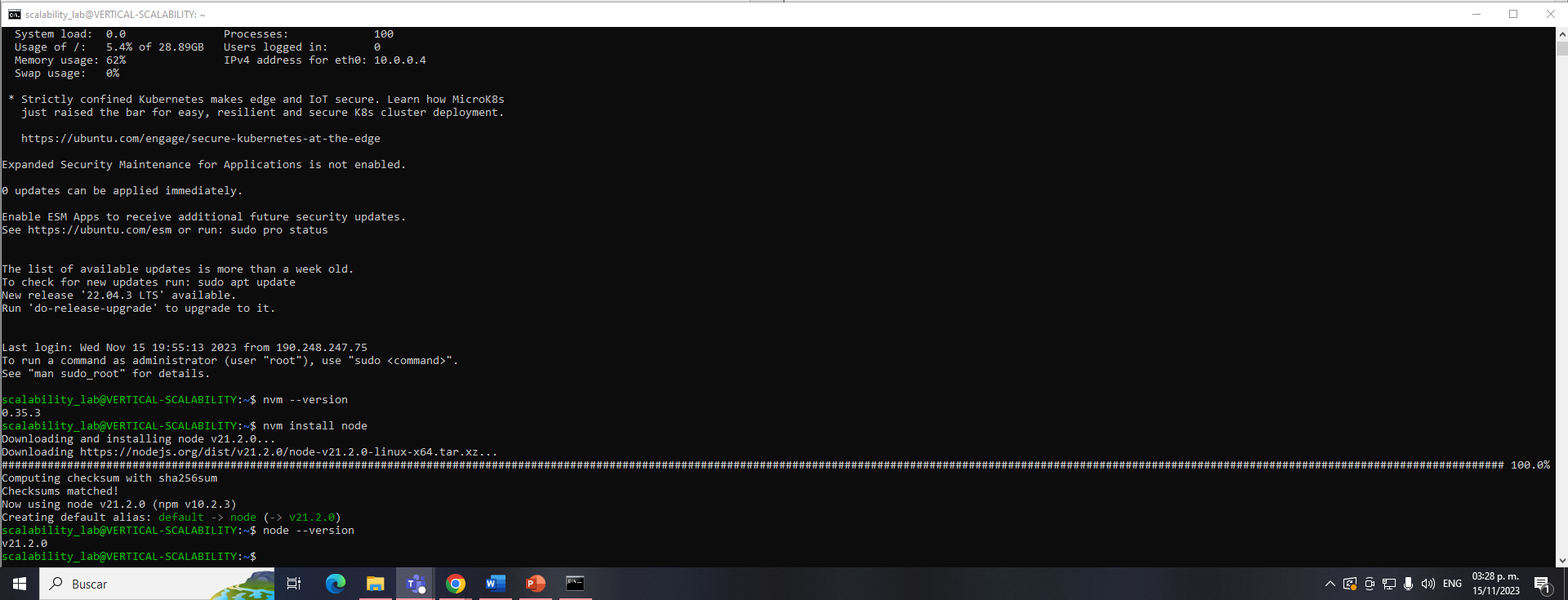


* Luego de instalar nvm , instalamos la ultima versión de Node,js. Para esto ejecutamos el comando nvm install node



* Verificamos la instalación de Node.js. Para esto ejecutamos el siguiente comando

node --version



* Luego instalamos dos versiones más, la última versión LTS y la versión 8.10.0



* Enumeramos las versiones instaladas de Node.js. Para esto ejecutamos el comando

nvm ls

Imagen que contiene Rectángulo

Descripción generada automáticamente

1. Para instalar la aplicación adjunta al Laboratorio, suba la carpeta FibonacciApp a un repositorio al cual tenga acceso y ejecute estos comandos dentro de la VM:

|  |
| --- |
| git clone <your\_repo>  cd <your\_repo>/FibonacciApp  npm install |

Texto

Descripción generada automáticamente

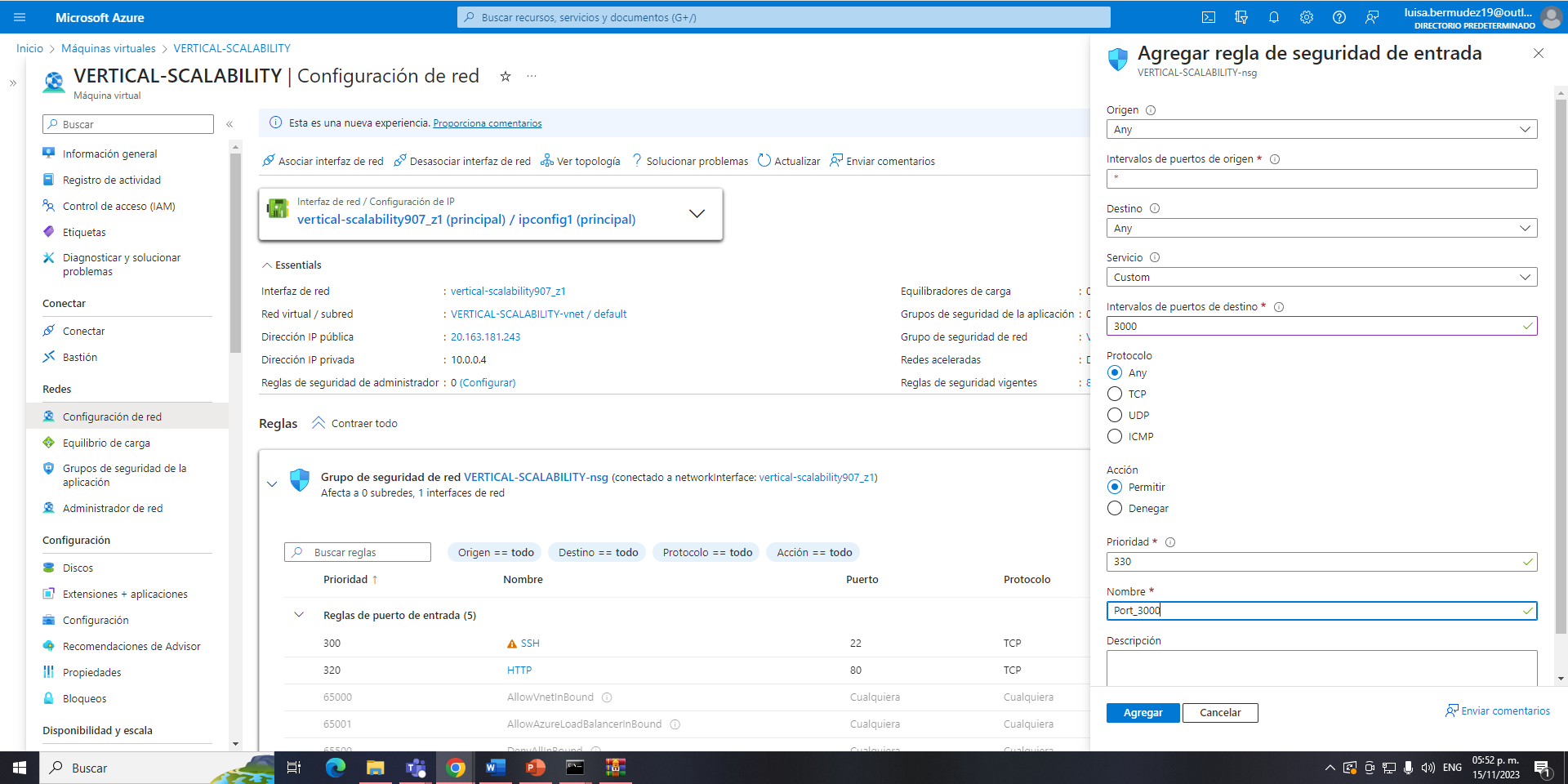
1. Para ejecutar la aplicación puede usar el comando npm FibinacciApp.js, sin embargo una vez pierda la conexión ssh la aplicación dejará de funcionar. Para evitar ese compartamiento usaremos forever. Ejecute los siguientes comandos dentro de la VM.

|  |
| --- |
| node FibonacciApp.js |

Imagen que contiene Texto

Descripción generada automáticamente

1. Antes de verificar si el endpoint funciona, en Azure vaya a la sección de Networking y cree una Inbound port rule tal como se muestra en la imágen. Para verificar que la aplicación funciona, use un browser y user el endpoint http://xxx.xxx.xxx.xxx:3000/fibonacci/6. La respuesta debe ser The answer is 8.



Al probar en el browser con <http://20.163.181.243:3000/fibonacci/6>, obtenemos como resultado 8

Interfaz de usuario gráfica, Texto, Aplicación, Word

Descripción generada automáticamente

1. La función que calcula en enésimo número de la secuencia de Fibonacci está muy mal construida y consume bastante CPU para obtener la respuesta. Usando la consola del Browser documente los tiempos de respuesta para dicho endpoint usando los siguientes valores:

* 1000000

Como se puede observar en la imagen el tiempo de respuesta fue de 15.42s

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

* 1010000

Como se puede observar en la imagen el tiempo de respuesta fue de 16.03s

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

* 1020000

Como se puede observar en la imagen el tiempo de respuesta fue de 16.28s

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

* 1030000

Como se puede observar en la imagen el tiempo de respuesta fue de 16.68s

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

* 1040000

Como se puede observar en la imagen el tiempo de respuesta fue de 16.98s

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

* 1050000

Como se puede observar en la imagen el tiempo de respuesta fue de 17.56s

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

* 1060000

Como se puede observar en la imagen el tiempo de respuesta fue de 17.95s

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

* 1070000

Como se puede observar en la imagen el tiempo de respuesta fue de 18.24s

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

* 1080000

Como se puede observar en la imagen el tiempo de respuesta fue de 18.63s

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

* 1090000

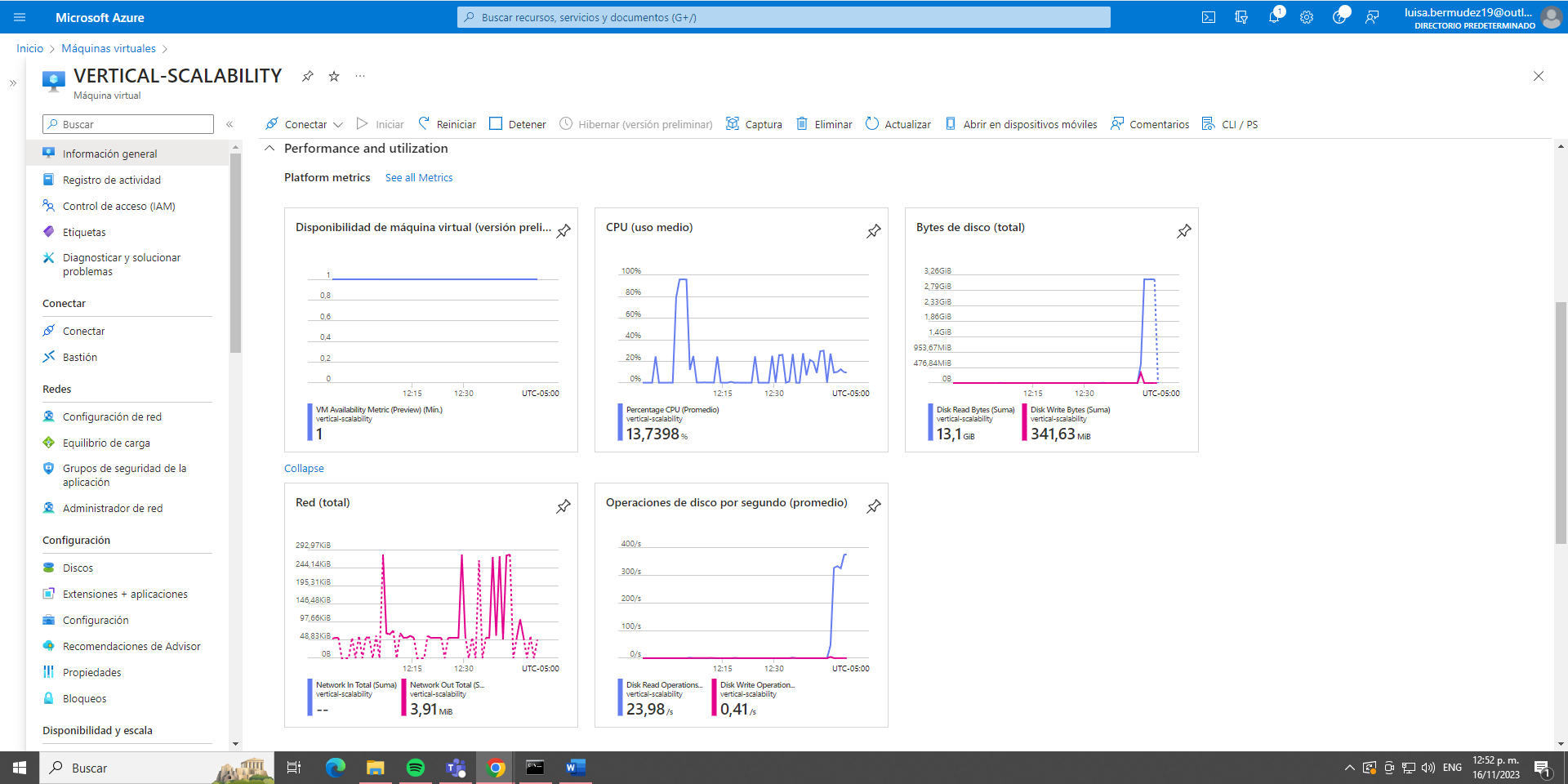
Como se puede observar en la imagen el tiempo de respuesta fue de 18.96s

Interfaz de usuario gráfica, Aplicación, Word

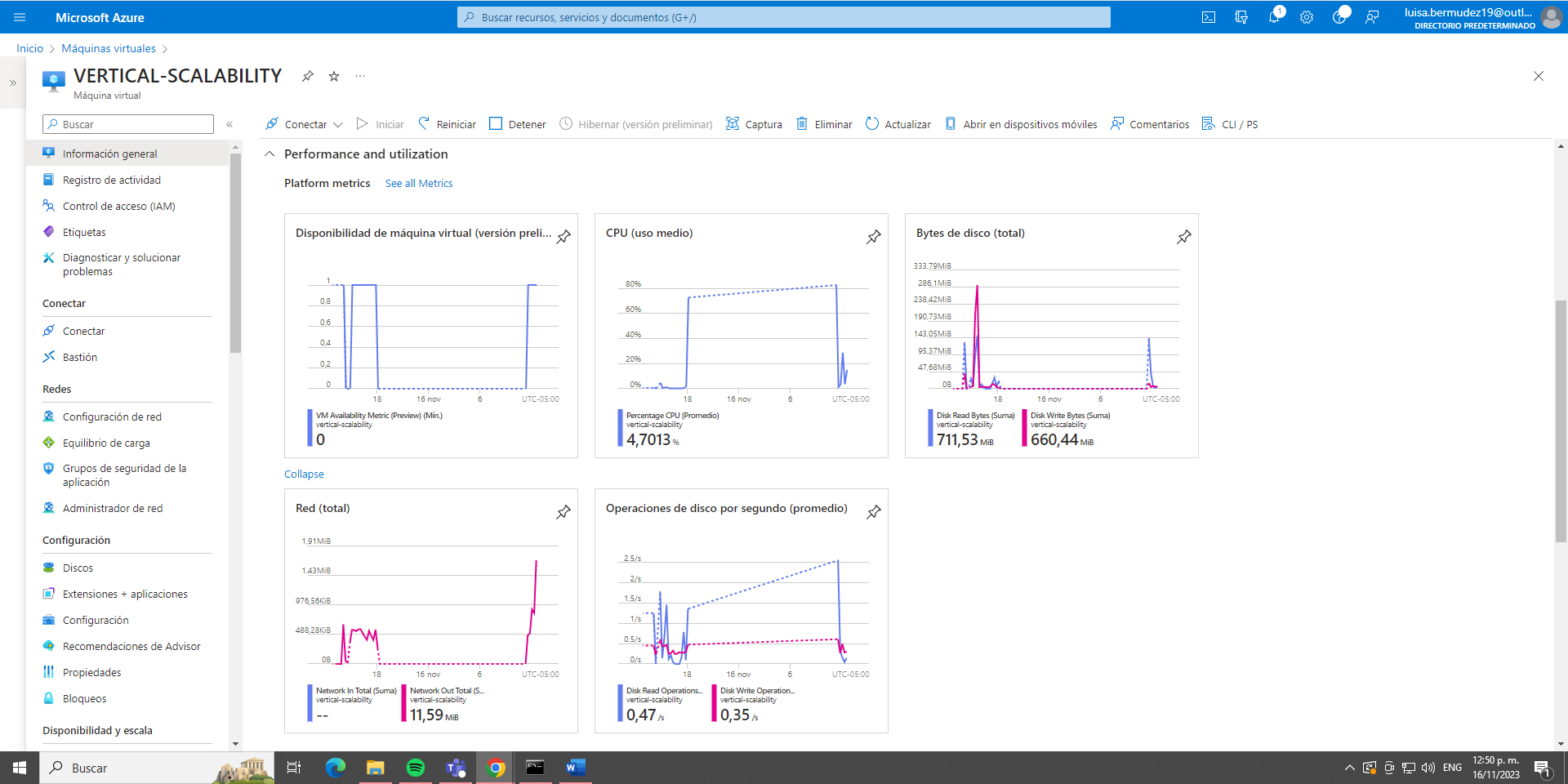
Descripción generada automáticamente

1. Diríjase ahora a Azure y verifique el consumo de CPU para la VM. (Los resultados pueden tardar 5 minutos en aparecer).

Métricas de la ultima hora

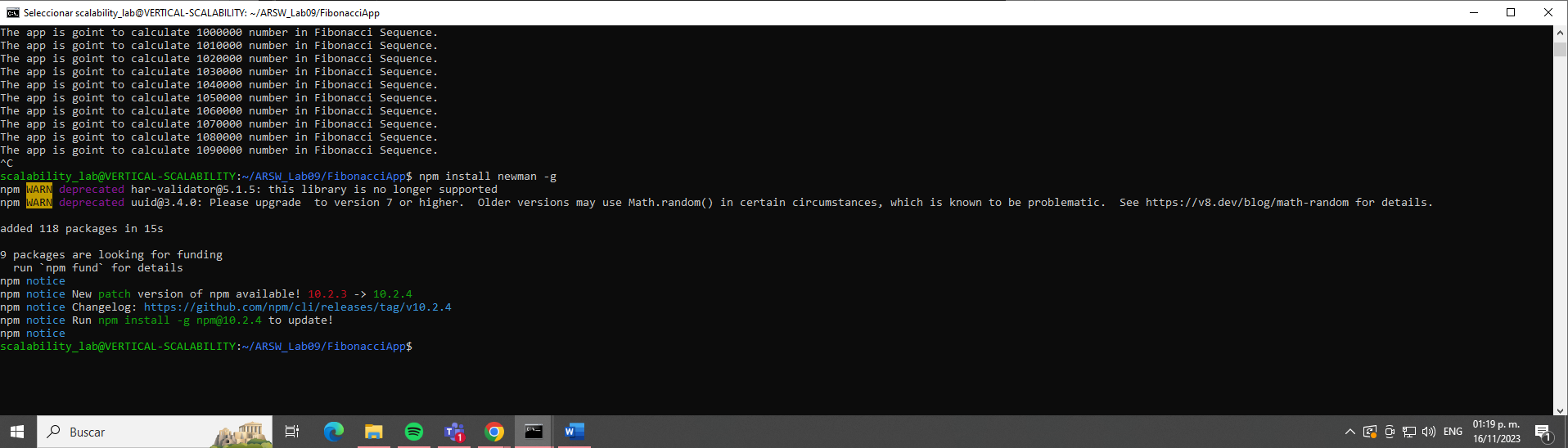


Métricas de la ultimas 24 horas



1. Ahora usaremos Postman para simular una carga concurrente a nuestro sistema. Siga estos pasos.

* Instale newman con el comando npm install newman -g. Para conocer más de Newman consulte el siguiente [enlace](https://learning.getpostman.com/docs/postman/collection-runs/command-line-integration-with-newman/).

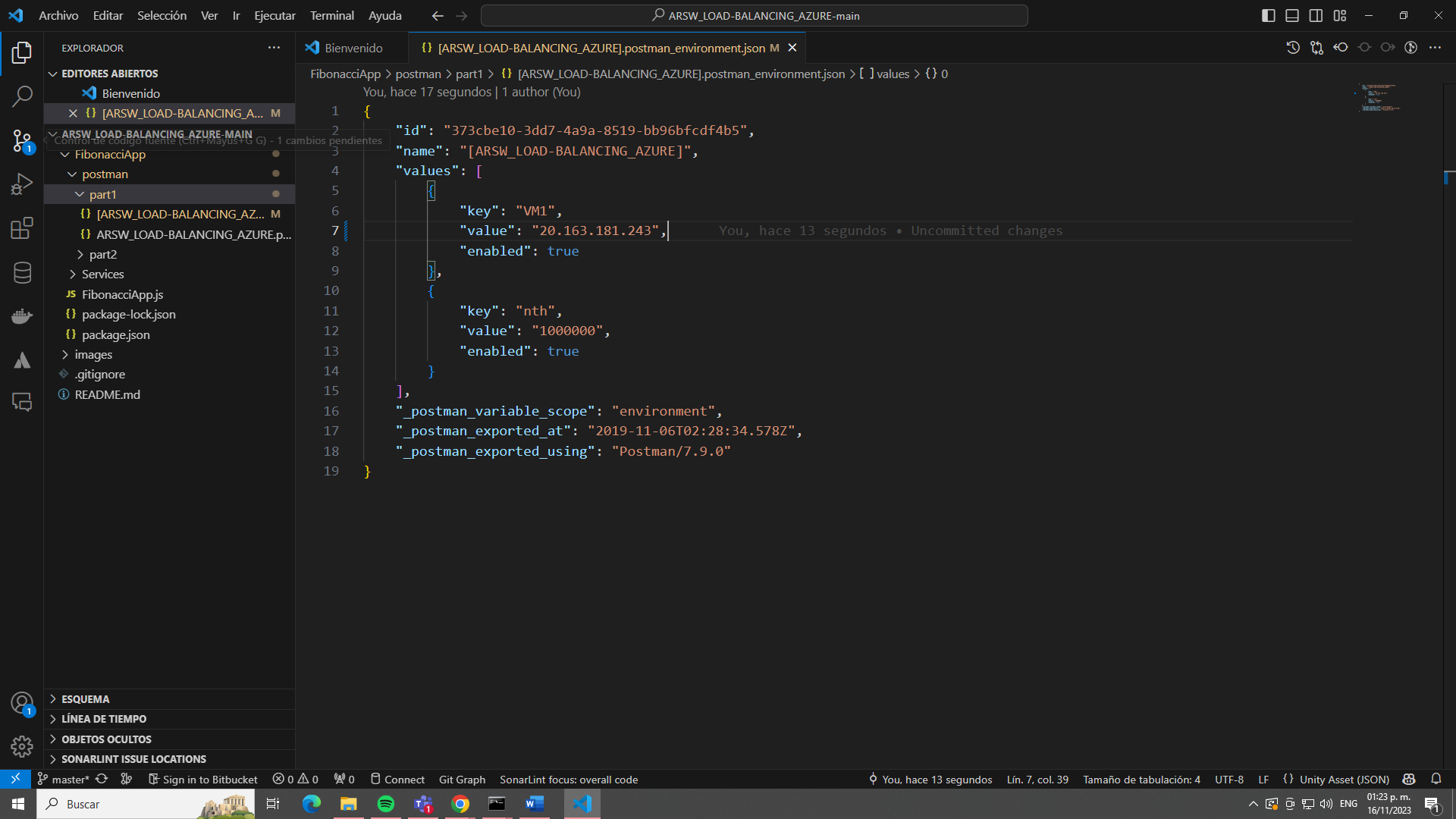


* Diríjase hasta la ruta FibonacciApp/postman en una maquina diferente a la VM.

Interfaz de usuario gráfica

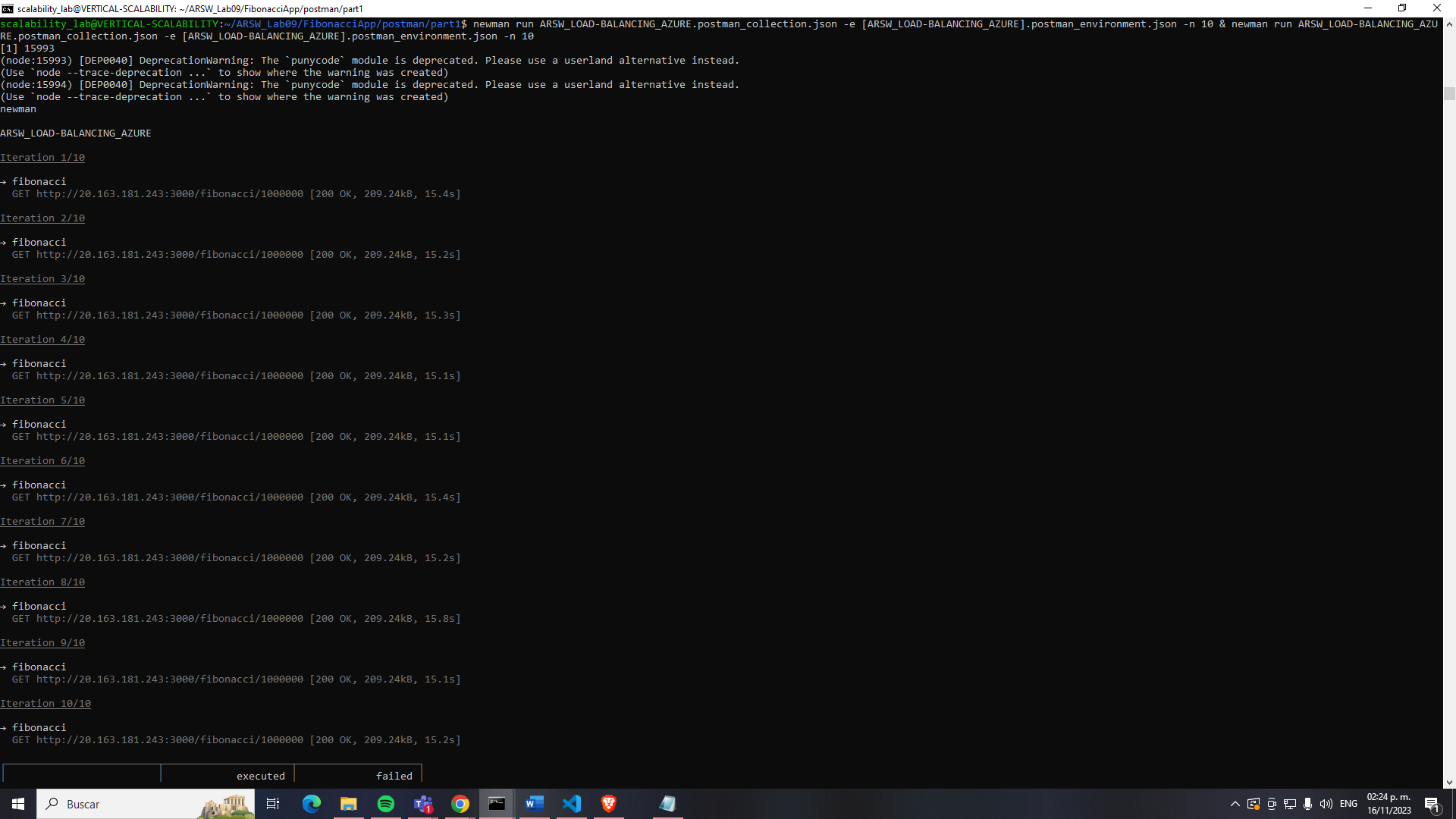
Descripción generada automáticamente con confianza media

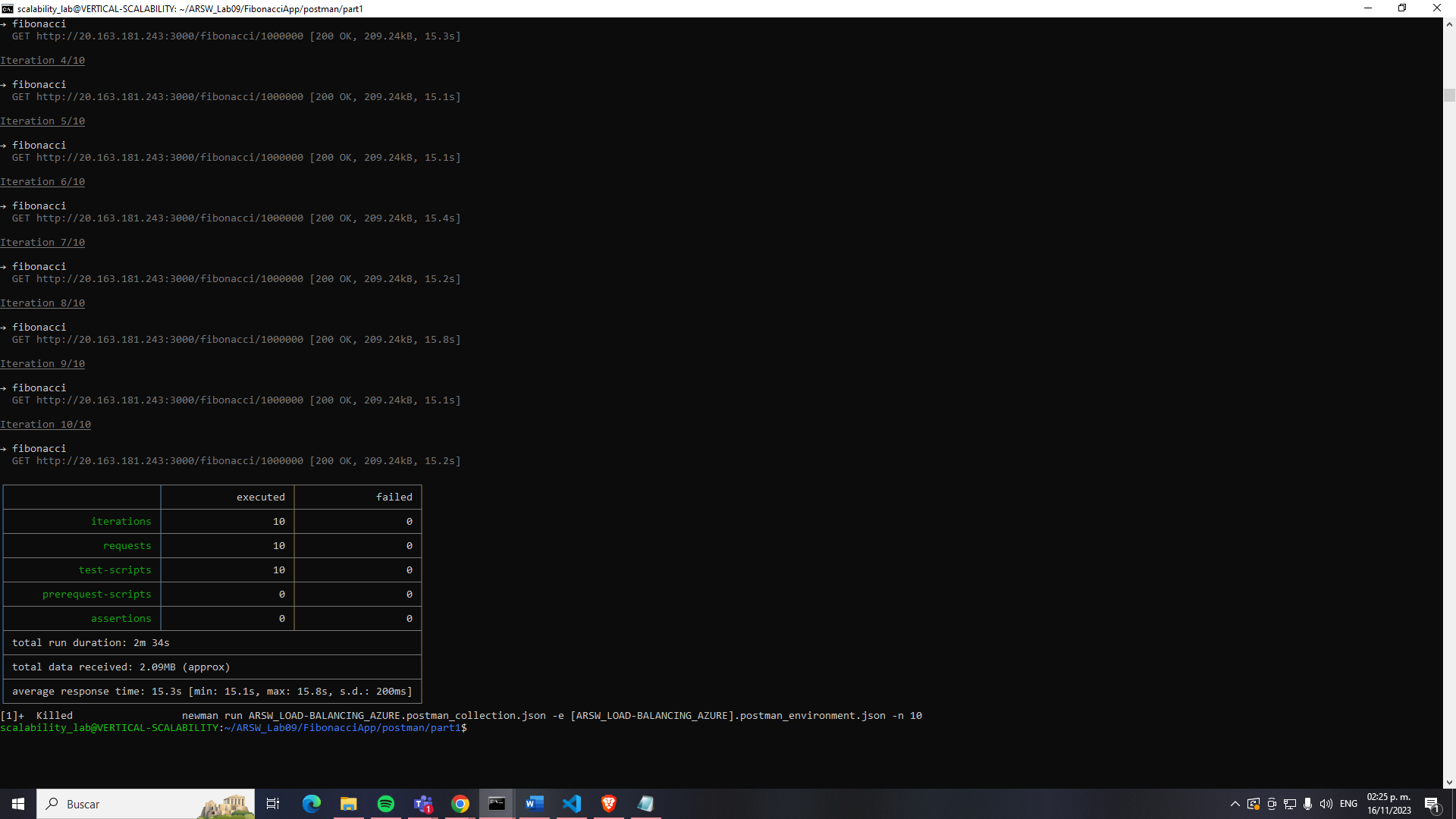
* Para el archivo [ARSW\_LOAD-BALANCING\_AZURE].postman\_environment.json cambie el valor del parámetro VM1 para que coincida con la IP de su VM



* Ejecute el siguiente comando.

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





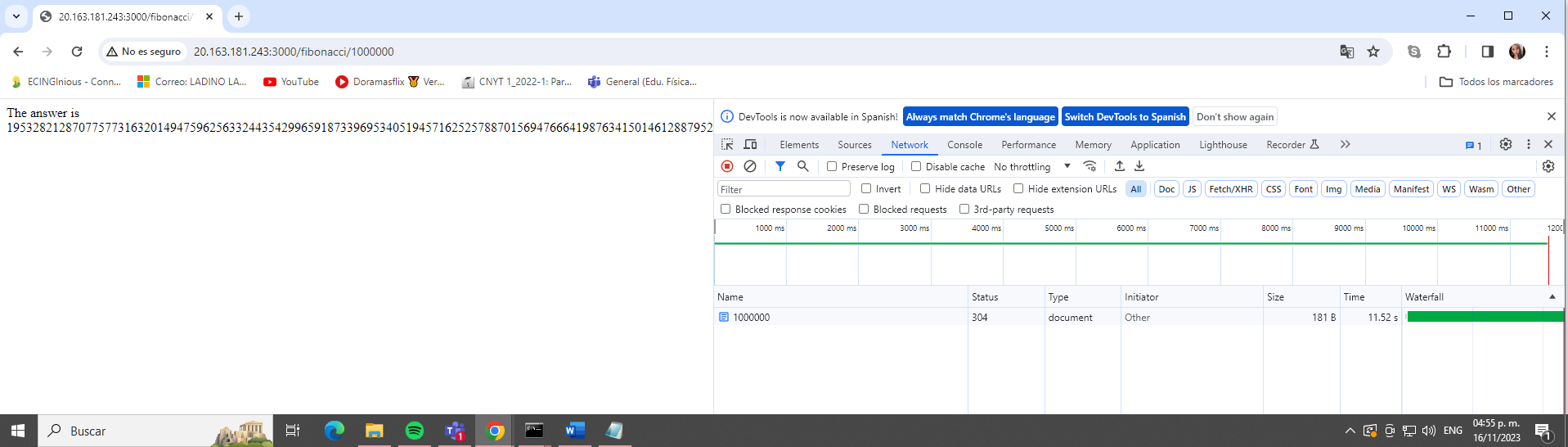
1. La cantidad de CPU consumida es bastante grande y un conjunto considerable de peticiones concurrentes pueden hacer fallar nuestro servicio. Para solucionarlo usaremos una estrategia de Escalamiento Vertical. En Azure diríjase a la sección size y a continuación seleccione el tamaño B2ms.



1. Una vez el cambio se vea reflejado, repita el paso 7, 8 y 9.

* 1000000

Como se puede observar en la imagen el tiempo de respuesta fue de 11.52s



* 1010000

Como se puede observar en la imagen el tiempo de respuesta fue de 12.05s

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

* 1020000

Como se puede observar en la imagen el tiempo de respuesta fue de 12.45s

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

* 1030000

Como se puede observar en la imagen el tiempo de respuesta fue de 12.69s

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

* 1040000

Como se puede observar en la imagen el tiempo de respuesta fue de 12.89s

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

* 1050000

Como se puede observar en la imagen el tiempo de respuesta fue de 13.14s

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

* 1060000

Como se puede observar en la imagen el tiempo de respuesta fue de 13.64s

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

* 1070000

Como se puede observar en la imagen el tiempo de respuesta fue de 13.71s

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

* 1080000

Como se puede observar en la imagen el tiempo de respuesta fue de 13.74s

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

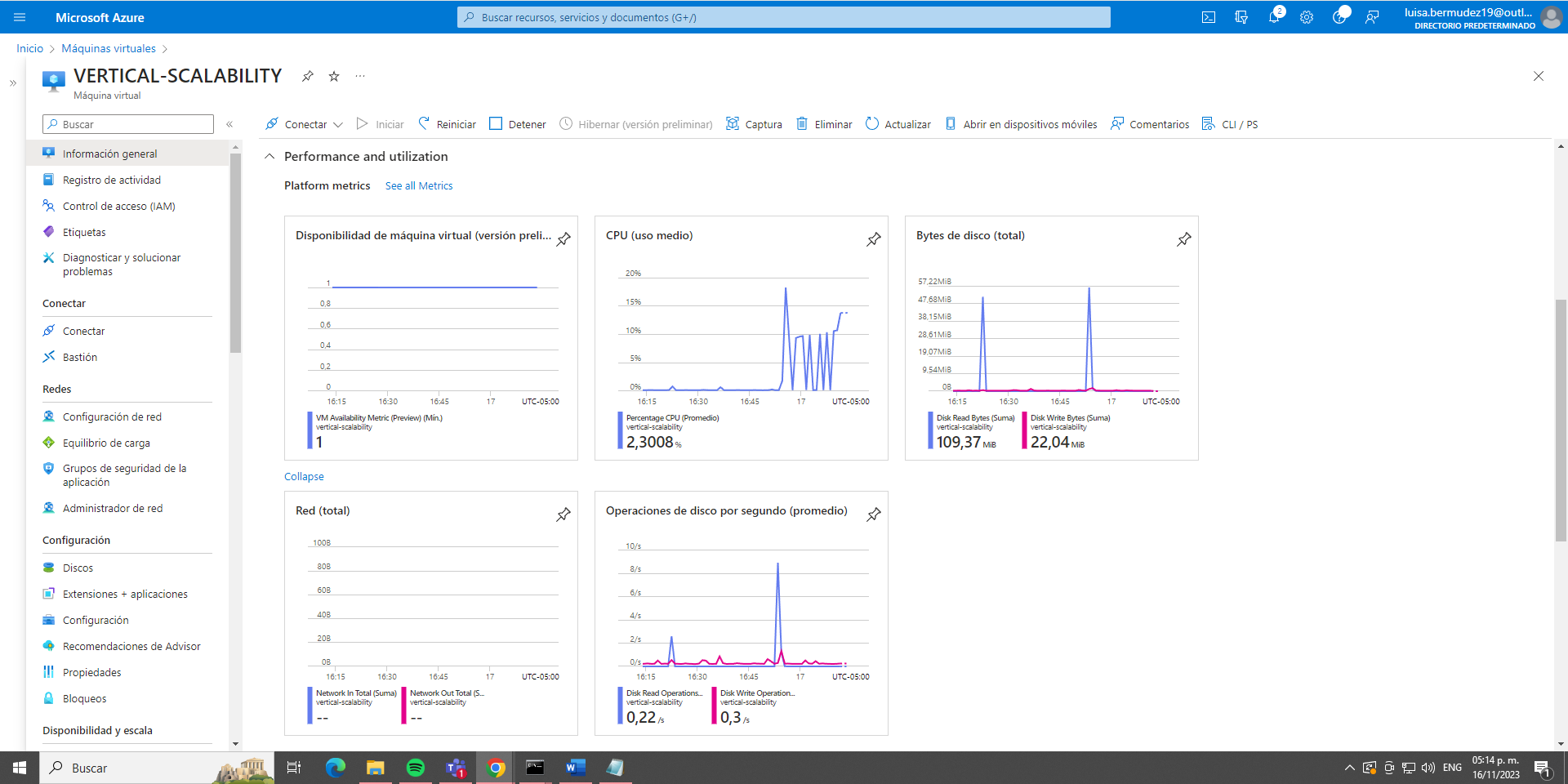
* 1090000

Como se puede observar en la imagen el tiempo de respuesta fue de 13.80s

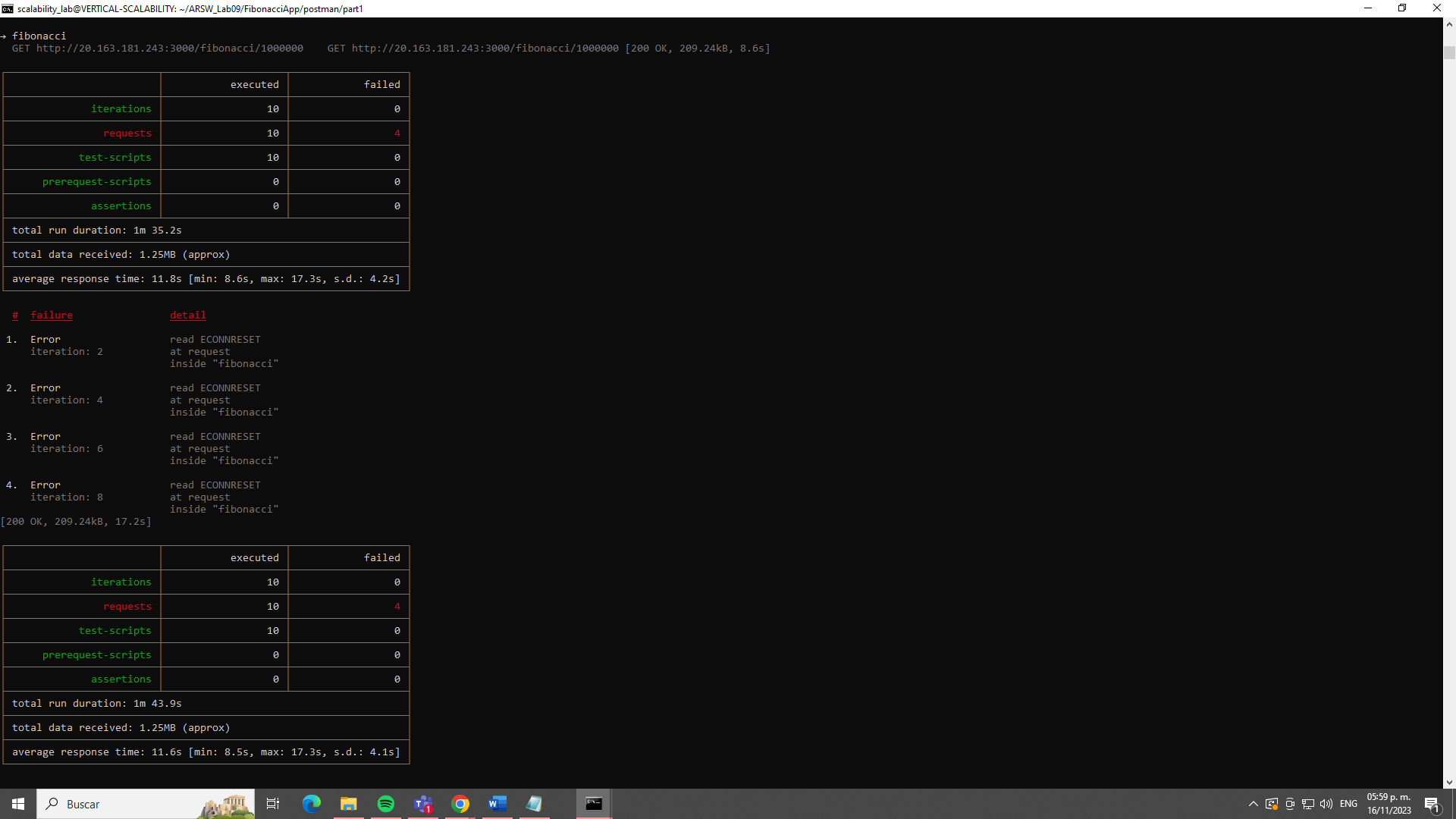
Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

* Métricas última hora



* Usando Postman para simular una carga concurrente a nuestro sistema.



1. Evalué el escenario de calidad asociado al requerimiento no funcional de escalabilidad y concluya si usando este modelo de escalabilidad logramos cumplirlo.

Si analizamos los tiempos obtenidos los resultados de las pruebas de escalabilidad muestran que el sistema puede escalar sin afectar significativamente el rendimiento. Ya que cuando se documentaron los tiempos en cada uno de los casos estos mejoraron en comparación con los resultados obtenidos antes de realizar el escalamiento, adicional a esto como se puede observar al usar el postman para simular la carga concurrente el promedio de tiempo de respuesta fue de 11.8 segundos y 11.6 segundos. Mientras que cuando realizamos esto sin escalamiento, el promedio de tiempo de respuesta fue de 15.35 segundos, lo que representa un aumento de aproximadamente 29%.

En base a esto podríamos decir que el modelo de escalabilidad utilizado logró cumplir el requerimiento no funcional de escalabilidad, llevando así que el sistema puede escalar sin afectar significativamente el rendimiento.

Pero si tenemos en cuenta los request que fallaron luego de realizar la escalabilidad y el motivo del fallo de estos podría pasar que el modelo de escalabilidad utilizado no logre cumplir el requerimiento no funcional de escalabilidad.

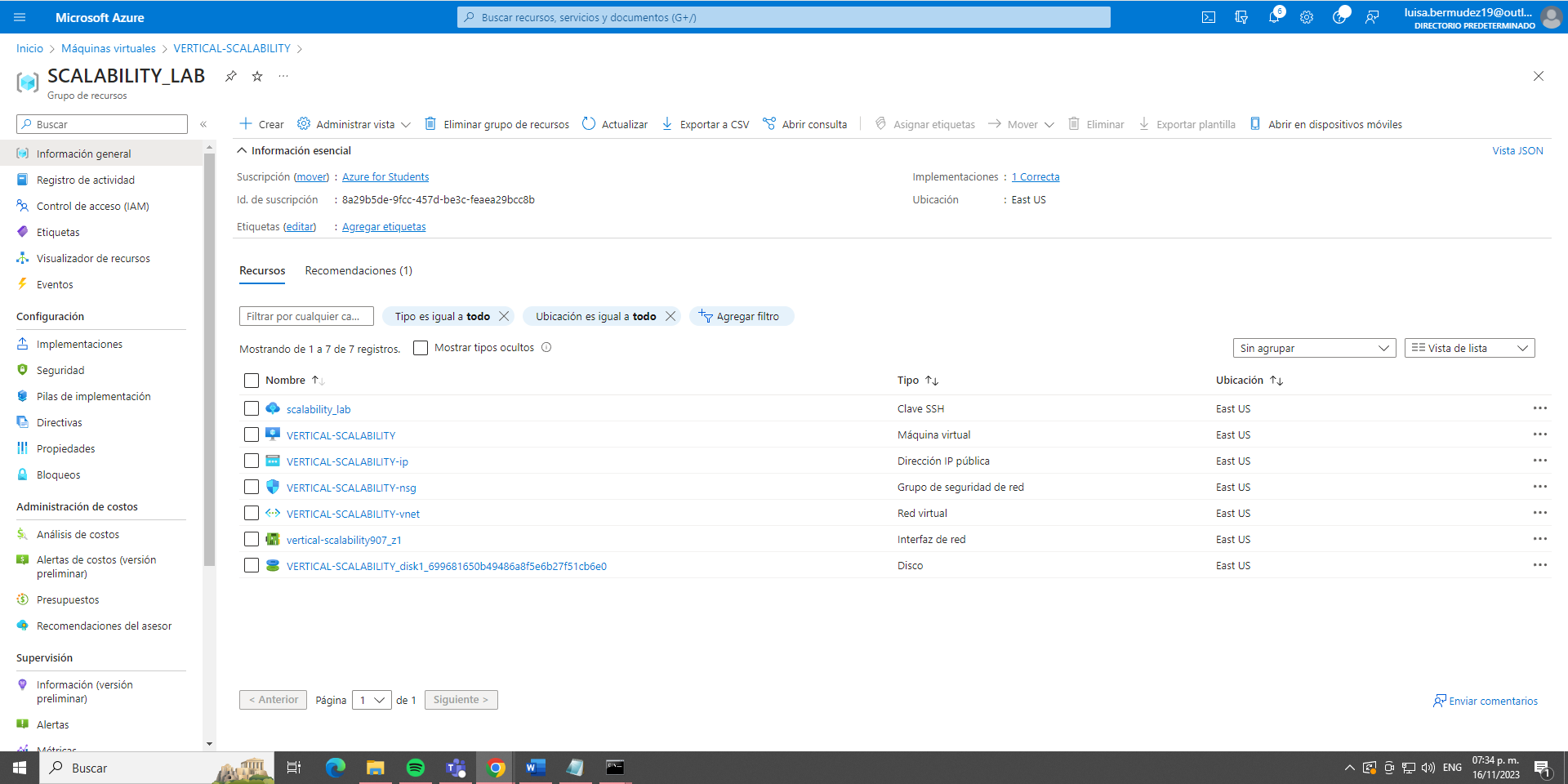
1. Vuelva a dejar la VM en el tamaño inicial para evitar cobros adicionales.

**Preguntas**

1. ¿Cuántos y cuáles recursos crea Azure junto con la VM?

Se crean 7 recursos

* Clave SSH
* Máquina virtual
* Dirección IP pública
* Grupo de seguridad de red
* Red virtual
* Interfaz de red
* Disco



1. ¿Brevemente describa para qué sirve cada recurso?

* **Clave SSH:** Se utiliza para conectarse a la máquina virtual de forma segura.
* **Máquina virtual:** Es el recurso principal de Azure que proporciona un servidor virtual en la nube.
* **Dirección IP pública:** Permite que la máquina virtual se conecte a Internet.
* **Grupo de seguridad de red:** Controla el tráfico entrante y saliente de la máquina virtual.
* **Red virtual:** Es una red privada que conecta las máquinas virtuales entre sí.
* **Interfaz de red:** Conecta la máquina virtual a la red virtual.
* **Disco:** Almacena el sistema operativo y los datos de la máquina virtual.

1. ¿Al cerrar la conexión ssh con la VM, por qué se cae la aplicación que ejecutamos con el comando npm FibonacciApp.js? ¿Por qué debemos crear un *Inbound port rule* antes de acceder al servicio?

* La conexión SSH es la que inicia la ejecución de la aplicación, por lo tanto, cuando se cierra la conexión SSH, se detiene el proceso que inició la aplicación, y con él, todos los procesos que se ejecutan como hijos de este proceso.

En el caso específico de la aplicación FibonacciApp.js, el comando npm FibonacciApp.js inicia un proceso que crea un servidor web en el puerto 3000. Por lo tanto, cuando se cierra la conexión SSH, este proceso se detiene, y con él, el servidor web. Esto significa que los clientes ya no pueden acceder a la aplicación.

* El firewall de la máquina virtual bloquea todo el tráfico de red entrante por defecto. Para que los clientes puedan acceder a la aplicación FibonacciApp.js, debemos crear una regla de entrada que permita el tráfico en el puerto 3000.

Esta regla especifica que el tráfico entrante en el puerto 3000 está permitido desde todas las direcciones IP. Esto significa que los clientes de cualquier dirección IP podrán acceder a la aplicación FibonacciApp.js.

1. Adjunte tabla de tiempos e interprete por qué la función tarda tanto tiempo.

|  |  |  |
| --- | --- | --- |
| Numero Secuencia | B1ls | B2ms |
| 1000000 | 15.42 | 11.52 |
| 1010000 | 16.03 | 12.05 |
| 1020000 | 16.28 | 12.45 |
| 1030000 | 16.68 | 12.69 |
| 1040000 | 16.98 | 12.89 |
| 1050000 | 17.56 | 13.14 |
| 1060000 | 17.95 | 13.64 |
| 1070000 | 18.24 | 13.71 |
| 1080000 | 18.63 | 13.74 |
| 1090000 | 18.96 | 13.8 |

La máquina virtual B1ls tiene menos recursos disponibles que la máquina virtual B2ms. Por lo tanto, la máquina virtual B1ls tiene menos memoria para almacenar los datos intermedios de la función de Fibonacci y menos potencia de procesamiento para realizar los cálculos necesarios, por esta razón los tiempos de la VM con B1ls tardan más tiempo.

1. Adjunte imágen del consumo de CPU de la VM e interprete por qué la función consume esa cantidad de CPU.

|  |  |
| --- | --- |
| B1ls | B2ms |
|  |  |

La función consume una mayor cantidad de CPU, debido a que como se mencionó anteriormente la máquina virtual B1ls tiene menos recursos disponibles que la máquina virtual B2ms. En las imágenes, podemos observar que la función de Fibonacci consume un promedio del 13,7 % de la CPU de la máquina virtual B1ls, mientras que cuando se construye en una máquina virtual B2ms, la función consume un promedio del 2,3 % de la CPU. Esto se debe a que la función de Fibonacci es una función recursiva que calcula el número de Fibonacci a partir de los dos números anteriores. Por lo tanto, esta función realizara varios cálculos y los almacenara.

1. Adjunte la imagen del resumen de la ejecución de Postman. Interprete:

* B1ls
* Tiempos de ejecución.

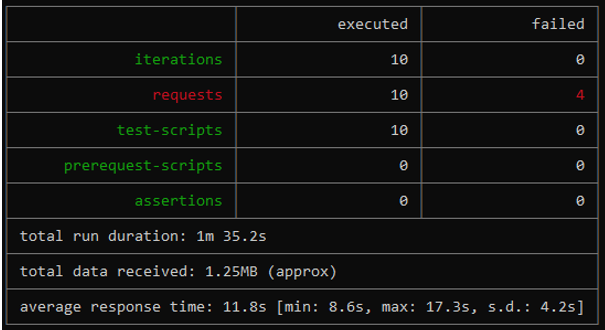
Una captura de pantalla de un celular

Descripción generada automáticamente con confianza media

* Si hubo fallos documéntelos y explique.

En este caso no se presentaron fallos.

* B2ms
* Tiempos de ejecución.



Una captura de pantalla de un celular

Descripción generada automáticamente con confianza media

* Si hubo fallos documéntelos y explique.

|  |  |
| --- | --- |
| Texto  Descripción generada automáticamente |  |

En este caso se presentó el error ECONNRESET, el cual es un error de red que se produce cuando el servidor cierra la conexión de una manera que probablemente no era normal. Esto puede ocurrir por varias razones, como:

* El servidor se ha sobrecargado y no puede atender más peticiones.
* El servidor ha detectado un error en la petición del cliente.
* El cliente ha cerrado la conexión de forma inesperada.

En esta ocasión el error ECONNRESET se produce porque la máquina virtual B2ms tiene más recursos disponibles que la B1ls, permitiendo así que se genere una mayor concurrencia en las peticiones. Sin embargo, la máquina virtual no puede responder a todas las solicitudes a tiempo, lo que provoca que el servidor cierre la conexión.

1. ¿Cuál es la diferencia entre los tamaños B2ms y B1ls (no solo busque especificaciones de infraestructura)?

* Diferencias de las especificaciones de infraestructura:

|  |  |  |
| --- | --- | --- |
| Característica | B1ls | B2ms |
| CPU | 1 | 2 |
| RAM | 0.5 | 8 |
| Discos de datos | 2 | 4 |
| E/S máxima por segundo | 320 | 1920 |
| Almacenamiento temporal | 4 | 16 |
| Costo | 3.8 | 60.74 |

* **Disponibilidad:** Los tamaños B2ms están disponibles en todos los centros de datos de Azure, mientras que los tamaños B1ls solo están disponibles en algunos centros de datos.
* **Compatibilidad:** Los tamaños B2ms son compatibles con todos los sistemas operativos, mientras que los tamaños B1ls solo son compatibles con Linux.
* **Rendimiento: Los tamaños B2ms tienen un rendimiento de línea de base garantizado, lo que significa que pueden proporcionar una potencia de procesamiento constante. Mientras que Los tamaños B1ls, no tienen un rendimiento de línea de base garantizado. El rendimiento de la CPU de estas máquinas virtuales varía según la demanda de la infraestructura de Azure.**

1. ¿Aumentar el tamaño de la VM es una buena solución en este escenario?, ¿Qué pasa con la FibonacciApp cuando cambiamos el tamaño de la VM?

En este escenario no es una buena solución aumentar el tamaño de la VM, ya que a pesar de que la función Fibonacci es una función de uso intensivo de recursos, se están presentando fallos debido a que la función no tiene un algoritmo eficiente, por lo tanto, cuando se escala el sistema y se agregan más instancias de esta se está cerrando el servidor debido a que no puede responder todas las solicitudes a tiempo. Para mejorar esto y poder aprovechar los recursos se debería optimizar la función para que esta use menos recursos llevando así a que el modelo de escalabilidad utilizado logre cumplir el requerimiento no funcional de escalabilidad.

1. ¿Qué pasa con la infraestructura cuando cambia el tamaño de la VM? ¿Qué efectos negativos implica?

Cuando cambiamos el tamaño de la maquina virtual esta se reinicia, llevando asi a que esta se detenga temporalmente, provocando:

* Una interrupción del servicio que se está ejecutando.
* Una pérdida de datos en dado caso de que se reduzca el tamaño de la VM, ya que la VM puede no tener suficiente espacio para almacenarlos. Esto puede ocurrir si la VM tiene datos que no se han utilizado en un tiempo.

1. ¿Hubo mejora en el consumo de CPU o en los tiempos de respuesta? Si/No ¿Por qué?

Si hubo una mejora en el consumo de CPU y en los tiempos de respuesta ya que:

* La función de Fibonacci consume un promedio del 13,7 % de la CPU de la máquina virtual B1ls, mientras que cuando se construye en una máquina virtual B2ms, la función consume un promedio del 2,3 % de la CPU.
* El tiempo de ejecución total de la máquina virtual B1ls es de 2m 34s, los cuales se dan para cada una de las peticiones con un promedio de 15.35s. Mientras que con la máquina virtual B2ms el tiempo de ejecución es de1m 35.2s con un promedio de 11.8s para cada una de las peticiones.

1. Aumente la cantidad de ejecuciones paralelas del comando de postman a 4. ¿El comportamiento del sistema es porcentualmente mejor?

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Captura de pantalla de computadora

Descripción generada automáticamente

Al aumentar la cantidad de ejecuciones paralelas a 4, el comportamiento del sistema no es mejor. Como podemos observar en las imágenes, el tiempo de respuesta aumenta hasta 3m 36s con un promedio de 36.4s de respuesta para cada petición, mientras que anteriormente teníamos un tiempo de ejecución de1m 35.2s con un promedio de 11.8s para cada una de las peticiones.

[**Parte 2 - Escalabilidad horizontal**](https://github.com/DiegoATG/ARSW_LOAD-BALANCING_AZURE#parte-2---escalabilidad-horizontal)

[**Crear el Balanceador de Carga**](https://github.com/DiegoATG/ARSW_LOAD-BALANCING_AZURE#crear-el-balanceador-de-carga)

Antes de continuar puede eliminar el grupo de recursos anterior para evitar gastos adicionales y realizar la actividad en un grupo de recursos totalmente limpio.

El Balanceador de Carga es un recurso fundamental para habilitar la escalabilidad horizontal de nuestro sistema, por eso en este paso cree un balanceador de carga dentro de Azure tal cual como se muestra en la imágen adjunta.

1. A continuación, cree un Backend Pool, guiese con la siguiente imágen.
2. A continuación, cree un Health Probe, guiese con la siguiente imágen.
3. A continuación, cree un Load Balancing Rule, guiese con la siguiente imágen.
4. Cree una Virtual Network dentro del grupo de recursos, guiese con la siguiente imágen.

[**Crear las máquinas virtuales (Nodos)**](https://github.com/DiegoATG/ARSW_LOAD-BALANCING_AZURE#crear-las-maquinas-virtuales-nodos)

Ahora vamos a crear 3 VMs (VM1, VM2 y VM3) con direcciones IP públicas standar en 3 diferentes zonas de disponibilidad. Después las agregaremos al balanceador de carga.

1. En la configuración básica de la VM guíese por la siguiente imágen. Es importante que se fije en la "Avaiability Zone", donde la VM1 será 1, la VM2 será 2 y la VM3 será 3.
2. En la configuración de networking, verifique que se ha seleccionado la *Virtual Network* y la *Subnet* creadas anteriormente. Adicionalmente asigne una IP pública y no olvide habilitar la redundancia de zona.
3. Para el Network Security Group seleccione "avanzado" y realice la siguiente configuración. No olvide crear un *Inbound Rule*, en el cual habilite el tráfico por el puerto 3000. Cuando cree la VM2 y la VM3, no necesita volver a crear el *Network Security Group*, sino que puede seleccionar el anteriormente creado.
4. Ahora asignaremos esta VM a nuestro balanceador de carga, para ello siga la configuración de la siguiente imágen.
5. Finalmente debemos instalar la aplicación de Fibonacci en la VM. para ello puede ejecutar el conjunto de los siguientes comandos, cambiando el nombre de la VM por el correcto

|  |
| --- |
| git clone <https://github.com/daprieto1/ARSW_LOAD-BALANCING_AZURE.git>  curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.34.0/install.sh | bash  source /home/vm1/.bashrc  nvm install node  cd ARSW\_LOAD-BALANCING\_AZURE/FibonacciApp  npm install  npm install forever -g  forever start FibonacciApp.js |

Realice este proceso para las 3 VMs, por ahora lo haremos a mano una por una, sin embargo es importante que usted sepa que existen herramientas para aumatizar este proceso, entre ellas encontramos Azure Resource Manager, OsDisk Images, Terraform con Vagrant y Paker, Puppet, Ansible entre otras.

[**Probar el resultado final de nuestra infraestructura**](https://github.com/DiegoATG/ARSW_LOAD-BALANCING_AZURE#probar-el-resultado-final-de-nuestra-infraestructura)

1. Porsupuesto el endpoint de acceso a nuestro sistema será la IP pública del balanceador de carga, primero verifiquemos que los servicios básicos están funcionando, consuma los siguientes recursos:

|  |
| --- |
| http://52.155.223.248/  http://52.155.223.248/fibonacci/1 |

1. Realice las pruebas de carga con newman que se realizaron en la parte 1 y haga un informe comparativo donde contraste: tiempos de respuesta, cantidad de peticiones respondidas con éxito, costos de las 2 infraestrucruras, es decir, la que desarrollamos con balanceo de carga horizontal y la que se hizo con una máquina virtual escalada.
2. Agregue una 4 máquina virtual y realice las pruebas de newman, pero esta vez no lance 2 peticiones en paralelo, sino que incrementelo a 4. Haga un informe donde presente el comportamiento de la CPU de las 4 VM y explique porque la tasa de éxito de las peticiones aumento con este estilo de escalabilidad.

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

**Preguntas**

1. ¿Cuáles son los tipos de balanceadores de carga en Azure y en qué se diferencian?, ¿Qué es SKU, qué tipos hay y en qué se diferencian?, ¿Por qué el balanceador de carga necesita una IP pública?
2. ¿Cuál es el propósito del Backend Pool?
3. ¿Cuál es el propósito del Health Probe?
4. ¿Cuál es el propósito de la Load Balancing Rule? ¿Qué tipos de sesión persistente existen, por qué esto es importante y cómo puede afectar la escalabilidad del sistema?.
5. ¿Qué es una Virtual Network? ¿Qué es una Subnet? ¿Para qué sirven los address space y address range?
6. ¿Qué son las Availability Zone y por qué seleccionamos 3 diferentes zonas?. ¿Qué significa que una IP sea zone-redundant?
7. ¿Cuál es el propósito del Network Security Group?
8. Informe de newman 1 (Punto 2)
9. Presente el Diagrama de Despliegue de la solución.
10. **Conclusiones**