

LABORATORIO No.9



ELABORADO POR:

SAMUEL ROJAS YOPASA

MAURICIO ALEJANDRO MONROY JIMÉNEZ

PROFESOR

DIEGO ANDRES TRIVIÑO GONZALEZ

ARQUITECTURA DE SOFTWARE







13-11-2024

Parte 1

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

Se crean 6 recursos junto con la VM:

- Dirección IP pública.
- Clave SSH.
- Grupo de seguridad de red.
- Red virtual.
- Interfaz de red.
- Disco.

<input type="checkbox"/>	 VERTICAL-SCALABILITY-ip	Dirección IP pública
<input type="checkbox"/>	 VERTICAL-SCALABILITY-key	Clave SSH
<input type="checkbox"/>	 VERTICAL-SCALABILITY-nsg	Grupo de seguridad de red
<input type="checkbox"/>	 VERTICAL-SCALABILITY-vnet	Red virtual
<input type="checkbox"/>	 vertical-scalability413	Interfaz de red
<input type="checkbox"/>	 VERTICAL-SCALABILITY_OsDisk_1_c0b8a750368d4e7a95f84cd377411920	Disco

2. ¿Brevemente describa para qué sirve cada recurso?

- **Dirección IP pública:** Permite que la máquina virtual sea accesible desde Internet. Es la dirección que se usa para conectarse a la máquina virtual de forma remota.
- **Clave SSH:** Es una clave de acceso segura para autenticar la conexión con la máquina virtual, especialmente en sistemas Linux. Funciona como una alternativa más segura a las contraseñas.
- **Grupo de seguridad de red (NSG):** Es un conjunto de reglas de firewall que controla el tráfico de entrada y salida para la máquina virtual. Permite definir qué tipos de tráfico están permitidos o bloqueados.

- **Red virtual (VNet):** Es una red privada dentro de Azure donde se ubica la máquina virtual. Permite que los recursos en Azure se comuniquen de forma segura entre sí.
 - **Interfaz de red (NIC):** Conecta la máquina virtual a la red virtual. Define cómo la máquina virtual se comunica dentro de la red y con otros recursos.
 - **Disco:** Es el almacenamiento físico para la máquina virtual, donde se guardan el sistema operativo, los datos de la aplicación y otros archivos.
3. ¿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?

Cuando se ejecuta el comando `npm FibonacciApp.js` en la sesión SSH, la aplicación queda vinculada a esa sesión de terminal. Esto significa que cuando se cierra la conexión SSH, la sesión finaliza y también se termina cualquier proceso vinculado a ella, incluida la aplicación. La regla de puerto de entrada (Inbound Port Rule) es esencial para permitir que el tráfico de la red externa acceda a la máquina virtual. Sin esta regla, el firewall de Azure bloquea el acceso al puerto, impidiendo que la aplicación sea accesible desde el navegador o cualquier cliente externo. Al definir esta regla, se especifica que Azure debe permitir el tráfico hacia ese puerto desde el exterior, habilitando así el acceso al endpoint de la aplicación.

4. Adjunte tabla de tiempos e interprete ¿por qué la función tarda tanto tiempo?

```

> async function time(value){
  const start = performance.now();
  const answer = await fetch("http://13.82.188.197:3000/fibonacci/" + value);
  const end = performance.now();

  console.log("Valor: " + value + " Tiempo: " + (end - start) + "ms.");
}
< undefined
> time(1000000)
< ▶ Promise {<pending>}
Valor: 1000000 Tiempo: 11582.099999999627ms. VM291:6
> time(1010000)
< ▶ Promise {<pending>}
Valor: 1010000 Tiempo: 11789.5ms. VM291:6
> time(1020000)
< ▶ Promise {<pending>}
Valor: 1020000 Tiempo: 12065.299999999882ms. VM291:6
> time(1030000)
< ▶ Promise {<pending>}
Valor: 1030000 Tiempo: 19936.599999999627ms. VM291:6
> time(1040000)
< ▶ Promise {<pending>}
Valor: 1040000 Tiempo: 12674ms. VM291:6

```

```

> time(1050000)
< ▶ Promise {<pending>}
Valor: 1050000 Tiempo: 12980.699999999255ms. VM291:6
> time(1060000)
< ▶ Promise {<pending>}
Valor: 1060000 Tiempo: 13237.900000000373ms. VM291:6

```

```

< ▶ Promise {<pending>}
valor: 1070000, tiempo: 13581.199999988079 ms.
> |

```

```

> time(1080000)
< ▶ Promise {<pending>}
valor: 1080000, tiempo: 13807.399999976158 ms.
> time(1090000)
< ▶ Promise {<pending>}
valor: 1090000, tiempo: 14093.300000011921 ms.

```

Valor	Tiempo (ms)
1000000	11582.09
1010000	11789.5
1020000	12065.29

1030000	19936.59
1040000	12674
1050000	12980.69
1060000	13237.90
1070000	13581.19
1080000	13807.39
1090000	14093.30

La función que calcula el número en la secuencia de Fibonacci es la siguiente:

```

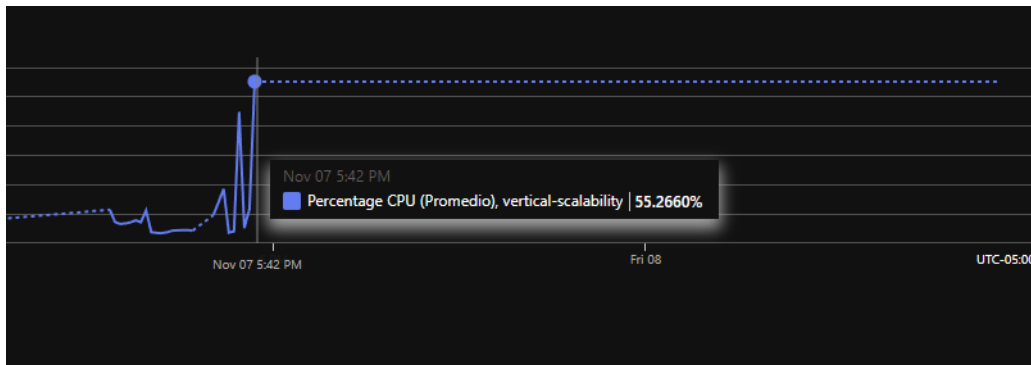
1  var bigInt = require("big-integer");
2
3  class FibonacciService {
4  static getNthNumberInSequence(nth) {
5      console.log(`The app is going to calculate ${nth} number in Fibonacci Sequence.`)
6
7      let nth_1 = bigInt.one;
8      let nth_2 = bigInt.zero;
9      let answer = bigInt.zero;
10
11     if (nth < 0)
12         throw 'must be greater than 0'
13     else if (nth === 0)
14         answer = nth_2
15     else if (nth === 1)
16         answer = nth_1
17     else {
18         for (var i = 0; i < nth - 1; i++) {
19             answer = nth_2.add(nth_1)
20             nth_2 = nth_1
21             nth_1 = answer
22         }
23     }
24
25     return answer.toString()
26 }
27 }
28
29 module.exports = FibonacciService

```

En esta implementación, se utiliza una estrategia iterativa para calcular el número de Fibonacci. La complejidad del algoritmo es $O(n)$, ya que requiere una cantidad de operaciones lineal en relación con el n -avo valor. Esto significa que, a medida que la variable nth se incrementa, el número de iteraciones también aumenta linealmente, causando que la función tome cada vez más tiempo en completar el cálculo. Esto es

especialmente notorio para valores grandes de la variable n , dado que las operaciones de suma y asignación en números de gran tamaño (manejados por el módulo big-integer) requieren cada vez más procesamiento de la CPU.

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



Al observar el consumo de CPU en la máquina virtual, notamos que aumenta significativamente para valores altos en la secuencia de Fibonacci. Esto se debe a que los números en la secuencia crecen notoriamente y cada nuevo término requiere la suma de números cada vez más grandes. A medida que el número en la secuencia se incrementa, la función debe realizar más iteraciones y trabajar con cifras de gran tamaño, lo que exige más capacidad de procesamiento. En consecuencia, el cálculo de números altos en la secuencia de Fibonacci consume más CPU, ya que se realizan múltiples operaciones de suma sobre números extensos, lo que es computacionalmente costoso.

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

	executed	failed
iterations	10	0
requests	10	0
test-scripts	10	0
prerequest-scripts	0	0
assertions	0	0
total run duration: 1m 55.9s		
total data received: 2.09MB (approx)		
average response time: 11.5s [min: 11.4s, max: 11.7s, s.d.: 81ms]		

- Tiempos de ejecución de cada petición: El tiempo promedio de respuesta para cada solicitud fue de 11.5 segundos, con un mínimo de 11.4 segundos y un máximo de 11.7 segundos. Esto indica que la aplicación tiene un tiempo de respuesta constante, lo que sugiere que el cálculo de Fibonacci para los valores probados es estable pero bastante intensivo en CPU.
- Si hubo fallos documéntelos y explique: No hubo fallos en las solicitudes, ya que todas las iteraciones, solicitudes y scripts se ejecutaron exitosamente.

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

Tamaño de VM	vCPU	RAM(GiB)	Discos de datos	E/S máxima por segundo
B2ms	2	8	4	1920
B1ls	1	0.5	2	320

- **B2ms:**
 - **Rendimiento:** Puede manejar cargas de trabajo moderadas sin problemas de latencia. Adecuada para aplicaciones web, bases de datos pequeñas y servicios de backend.

- **Costo:** Más cara, pero balanceada para necesidades de producción básica con buena escalabilidad.
 - **Casos de uso:** Servidores web, backend, y aplicaciones que requieran respuesta estable bajo carga.
- **B1ls:**
- **Rendimiento:** Bajo; no ideal para aplicaciones concurrentes o que requieran mucha memoria, ya que experimenta latencia en tareas intensivas.
 - **Costo:** Muy económica, ideal para pruebas y desarrollo ligero, pero no recomendable para producción.
 - **Casos de uso:** Automatización de scripts, entornos de prueba y desarrollo básico.

Ambas VMs usan el sistema de créditos "Burstable" para picos temporales de rendimiento, pero B2ms soporta cargas elevadas por más tiempo que B1ls.

8. ¿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?

Aumentar el tamaño de la máquina virtual (VM) en este escenario puede ayudar a reducir el tiempo de respuesta debido a que se proporcionan más recursos de CPU. Sin embargo, esta no sería una solución óptima a largo plazo. La razón es que la función que calcula el número en la secuencia de Fibonacci está implementada de forma ineficiente, utilizando un algoritmo con complejidad de tiempo $O(n)$, lo que consume mucha CPU, especialmente para valores grandes como 1,000,000.

Aunque aumentar el tamaño de la VM puede mejorar temporalmente el rendimiento, la solución real consiste en optimizar la función de cálculo. Por ejemplo, utilizando técnicas más eficientes como la programación dinámica o la fórmula de Binet, se podría reducir significativamente el tiempo de ejecución y

hacer que la aplicación sea más escalable sin depender únicamente de recursos de hardware más grandes.

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

Cuando cambia el tamaño de la VM, la infraestructura recibe más recursos, como CPU y memoria, lo que puede mejorar el rendimiento de la aplicación. Sin embargo, también implica un mayor consumo de recursos y costos adicionales. Aumentar el tamaño de la VM tiene un límite y no soluciona problemas de eficiencia en la aplicación, lo que podría generar un ciclo de aumento de costos sin mejorar el rendimiento a largo plazo.

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

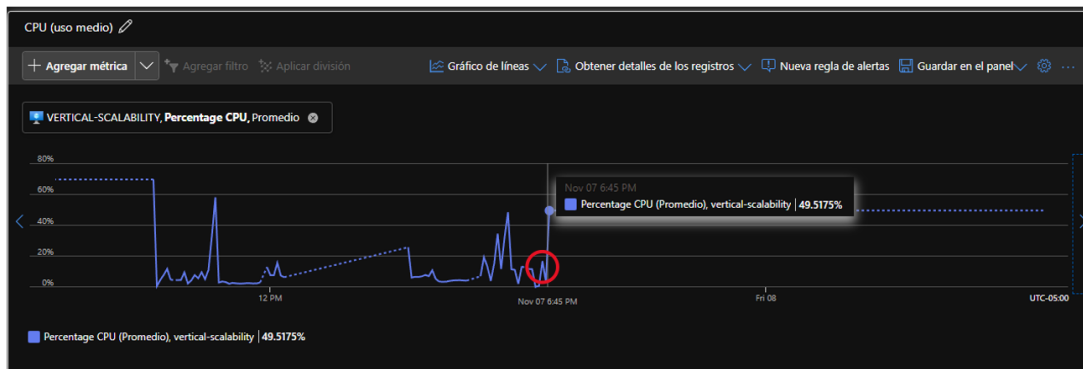
```

> time(1000000)
< ▶ Promise {<pending>}
valor: 1000000, tiempo: 9012.3000000011921 ms.
> time(1010000)
< ▶ Promise {<pending>}
valor: 1010000, tiempo: 8968.5 ms.
> time(1020000)
< ▶ Promise {<pending>}
valor: 1020000, tiempo: 9185.8000000011921 ms.
> time(1030000)
< ▶ Promise {<pending>}
valor: 1030000, tiempo: 9361.4000000035763 ms.
> time(1040000)
< ▶ Promise {<pending>}
valor: 1040000, tiempo: 9540.8000000011921 ms.
> time(1050000)
< ▶ Promise {<pending>}
valor: 1050000, tiempo: 9722.199999988079 ms.
> time(1060000)
< ▶ Promise {<pending>}
valor: 1060000, tiempo: 9985.8000000011921 ms.
> time(1070000)
< ▶ Promise {<pending>}
valor: 1070000, tiempo: 10099.5 ms.
> time(1080000)
< ▶ Promise {<pending>}
valor: 1080000, tiempo: 10378.9000000035763 ms.
> time(1090000)
< ▶ Promise {<pending>}
valor: 1090000, tiempo: 10575.699999988079 ms.

```

Valor	Tiempo (ms)
1000000	9012.3
1010000	8968.5
1020000	9185.8
1030000	9361.4

1040000	9540.8
1050000	9722.19
1060000	9985.8
1070000	10099.5
1080000	10378.9
1090000	10575.69



Sí, hubo mejora tanto en el consumo de CPU como en los tiempos de respuesta al cambiar de una VM B1ls a una VM B2ms.

La reducción en los tiempos de respuesta es evidente, ya que la VM B2ms tiene más recursos que la B1ls. La B2ms ofrece 2 vCPUs y 8 GB de RAM, frente a 1 vCPU y solo 0.5 GB de RAM en la B1ls. Estos recursos adicionales permiten una mejor distribución de la carga de trabajo, reduciendo el tiempo de procesamiento para cálculos intensivos como los de la secuencia de Fibonacci.

Además, la capacidad de E/S por segundo también mejora en la B2ms, lo que contribuye a un mejor rendimiento general en la ejecución de la aplicación. Aunque la mejora en los tiempos de respuesta es notable, la eficiencia del algoritmo sigue siendo un factor limitante, lo que explica que la reducción en los tiempos no sea tan drástica.

El consumo de CPU también se redujo, ya que la B2ms, con más vCPUs y mayor RAM, maneja la carga de forma más eficiente que la B1ls.

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

000

	executed	failed
iterations	10	0
requests	10	3
test-scripts	10	0
prerequest-scripts	0	0
assertions	0	0
total run duration: 3m 18.5s		
total data received: 1.46MB (approx)		
average response time: 22.6s [min: 9.4s, max: 28.4s, s.d.: 8.5s]		

→ fibonacci
GET http://13.82.188.197:3000/fibonacci/1000000

	executed	failed
iterations	10	0
requests	10	3
test-scripts	10	0
prerequest-scripts	0	0
assertions	0	0
total run duration: 4m 5.7s		
total data received: 1.46MB (approx)		
average response time: 30.6s [min: 18.7s, max: 1m 6.1s, s.d.: 18.7s]		

	executed	failed
iterations	10	0
requests	10	3
test-scripts	10	0
prerequest-scripts	0	0
assertions	0	0
total run duration: 4m 5.7s		
total data received: 1.46MB (approx)		
average response time: 30.8s [min: 9.4s, max: 56.7s, s.d.: 15.4s]		

	executed	failed
iterations	10	0
requests	10	4
test-scripts	10	0
prerequest-scripts	0	0
assertions	0	0
total run duration: 4m 15.1s		
total data received: 1.25MB (approx)		
average response time: 36.6s [min: 9.4s, max: 56.5s, s.d.: 15.8s]		

No, de hecho, fue peor el comportamiento del sistema al aumentar la cantidad de ejecuciones paralelas en Postman a 4. El rendimiento empeoró considerablemente. Las duraciones de ejecución fueron de **1:36, 4:15, 4:05 y 3:18** minutos, todas con tiempos de respuesta más altos en comparación con los obtenidos con la VM de tamaño **B1ls** en pruebas anteriores. Además, se presentaron fallos en las solicitudes, con entre **3 y 4 fallos de las 10 solicitudes realizadas** en cada prueba. Estos resultados sugieren que el sistema tiene limitaciones de rendimiento para manejar múltiples peticiones concurrentes, lo que provoca una sobrecarga en el consumo de CPU y posibles fallos en el procesamiento de solicitudes.

PARTE 2

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?

- Azure ofrece principalmente dos tipos de balanceadores de carga:
 - **Balanceador de carga público (Public Load Balancer):** Este tipo se utiliza para distribuir el tráfico entrante desde Internet hacia recursos en la nube, como máquinas virtuales.
 - **Balanceador de carga interno (Internal Load Balancer):** Este balanceador distribuye el tráfico en la red interna de Azure. Se usa principalmente para aplicaciones internas que no necesitan estar expuestas a Internet, como bases de datos o aplicaciones backend dentro de una red privada.

Estos balanceadores operan en la capa 4 del modelo OSI (protocolo TCP/UDP) y distribuyen las conexiones según la configuración y reglas que se establezcan.



- **SKU (Stock Keeping Unit):** Se refiere a las diferentes variantes o versiones del balanceador de carga que determinan sus características y capacidades. Azure ofrece tres SKUs para balanceadores de carga:
 - **Basic SKU:** Ideal para aplicaciones con necesidades simples de balanceo y para entornos de desarrollo y pruebas. Ofrece capacidades limitadas, como un número máximo de reglas de balanceo y conexiones concurrentes.

- **Standard SKU:** Ofrece mayores capacidades y características avanzadas, como alta disponibilidad y soporte para zonas de disponibilidad. Además, tiene una mayor capacidad de manejo de tráfico, SLA superior y mejores opciones de monitoreo y diagnóstico.
- **Gateway SKU:** Aunque no es un balanceador de carga tradicional, Azure Application Gateway se puede considerar en esta categoría, ya que proporciona balanceo de carga a nivel de aplicación (capa 7). Este SKU está diseñado para enrutamiento avanzado y manejo de tráfico HTTP/HTTPS, incluyendo afinidad de sesión y balanceo basado en URL.

Choose the SKU that's best for you



Standard Load Balancer

Equipped for load-balancing network layer traffic when high performance and super-low latency are needed.

Standard Load Balancer routes traffic within and across regions, and to availability zones for high resiliency.

[Learn more >](#)



Gateway Load Balancer

Easily deploy and scale virtual appliances with Gateway Load Balancer.

Enables scenarios that need service-chaining such as analytics, DDoS protection, firewall, and more.

[Learn more >](#)



Basic Load Balancer

Supports small-scale applications that don't need high availability or redundancy.

[Learn more >](#)

- Un balanceador de carga necesita una IP pública para permitir que las solicitudes entrantes desde Internet lleguen a los recursos de la red en Azure. Esta IP pública actúa como un punto de entrada para los clientes externos, permitiendo que el tráfico llegue a las máquinas virtuales (o cualquier otro recurso) de forma segura y eficiente.

En el caso de un balanceador de carga público, esta IP es necesaria para que el tráfico entrante pueda ser distribuido adecuadamente entre los recursos. La IP pública asociada permite a los usuarios acceder a los servicios alojados en las máquinas virtuales desde cualquier ubicación externa a la red de Azure.

2. ¿Cuál es el propósito del Backend Pool?

El Backend Pool es el conjunto de recursos (como máquinas virtuales o instancias de aplicaciones) que recibirán el tráfico distribuido por el balanceador. Su propósito principal es definir los recursos detrás del balanceador de carga que

procesarán las solicitudes entrantes. Cuando una solicitud llega al balanceador de carga, este distribuye el tráfico entre las instancias del Backend Pool de acuerdo con las reglas de balanceo configuradas. El Backend Pool agrupa los recursos que responderán al tráfico, permitiendo escalabilidad y redundancia, ya que varias instancias pueden manejar el tráfico, mejorando así la disponibilidad y rendimiento de la aplicación.

3. ¿Cuál es el propósito del Health Probe?

El Health Probe (o sondeo de estado) permite monitorear la disponibilidad de las instancias en el Backend Pool para asegurar que solo se envíe tráfico a instancias saludable, verifica periódicamente si las instancias están respondiendo correctamente en un puerto y protocolo específicos, como HTTP o TCP.

Cuando una instancia no responde correctamente al Health Probe, el balanceador de carga la considera no saludable y deja de enviarle tráfico. Esto garantiza que solo los recursos operativos reciban solicitudes, manteniendo así la continuidad del servicio y mejorando la disponibilidad y fiabilidad de la aplicación. Una vez que una instancia no saludable vuelve a pasar las verificaciones del Health Probe, puede volver a recibir tráfico.

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?

- La Load Balancing Rule define cómo el balanceador de carga distribuirá el tráfico entre las instancias del Backend Pool. Especifica el puerto y el protocolo de entrada (como HTTP en el puerto 80 o TCP en el puerto 443), el puerto de destino en las instancias de backend y el comportamiento de enrutamiento del tráfico.
- La persistencia de sesión asegura que todas las solicitudes de un cliente específico se envíen a la misma instancia de backend durante una sesión, lo

cual es importante en aplicaciones que necesitan que los datos del usuario se mantengan en la misma instancia mientras navega.

Existen principalmente dos tipos de sesión persistente en Azure:

- **Sin afinidad de sesión (None):** Las solicitudes de un cliente pueden dirigirse a cualquier instancia del Backend Pool. Este tipo ofrece mayor escalabilidad porque permite distribuir el tráfico de manera uniforme sin restricciones de sesión.
- **Cliente IP (Client IP):** Dirige todas las solicitudes de una misma IP de cliente a la misma instancia. Esto es útil cuando se necesita que la sesión del usuario sea persistente en la misma instancia, sin embargo, disminuye la flexibilidad de la distribución.
- **Cliente IP y Protocolo (Client IP and Protocol):** Además de considerar la IP del cliente, también mantiene la afinidad según el protocolo y puerto de origen. Es útil cuando se necesita un nivel adicional de persistencia.

Persistencia de la sesión ⓘ	Ninguno ▾
Tiempo de espera de inactividad (minutos) ⓘ	Ninguno
Habilitar el restablecimiento de TCP	IP de cliente
	IP y protocolo del cliente

5. ¿Qué es una Virtual Network? ¿Qué es una Subnet? ¿Para qué sirven los address space y address range?

- Una Virtual Network (VNet) en Azure es una red privada que permite a los recursos, comunicarse entre sí de forma segura y eficiente. Funciona como una red lógica aislada en la nube donde se pueden definir subredes, controlar el tráfico, establecer conexiones con redes locales y gestionar la seguridad de la red. Las VNets permiten extender una red empresarial a la nube y son la base para la conectividad y administración de redes.
- Una Subnet (subred) es una división lógica dentro de una VNet que permite segmentar el espacio de direcciones de la red en secciones más pequeñas.

Cada subred tiene un rango de direcciones IP específico dentro del espacio de direcciones de la VNet y puede configurarse con sus propias reglas de seguridad y de acceso. Al dividir una VNet en subredes, se pueden organizar y aislar recursos para mejorar la seguridad y la administración del tráfico dentro de la red.

- **Address Space (espacio de direcciones):** Define el rango total de direcciones IP que la VNet usará. Este espacio determina el conjunto completo de direcciones IP disponibles dentro de la red virtual y abarca todas las subredes que se creen en ella.
- **Address Range (rango de direcciones):** Es el subconjunto de direcciones dentro del Address Space que se asigna a una subred en particular.

6. ¿Qué son las Availability Zone y por qué seleccionamos 3 diferentes zonas? ¿Qué significa que una IP sea zone-redundant?

- Las Availability Zone son ubicaciones físicas distintas dentro de una región, diseñadas para proporcionar alta disponibilidad y redundancia. Cada zona está aislada de las demás, lo que significa que, si una zona experimenta una falla, las otras zonas seguirán funcionando de manera independiente, ayudando a garantizar que las aplicaciones y servicios en la nube sean más resilientes y tengan menos riesgo de interrupciones.

Se seleccionan 3 zonas de disponibilidad para desplegar la aplicación para aumentar la disponibilidad y la tolerancia a fallos. Con tres zonas, los servicios y recursos pueden distribuirse entre ellas, lo que garantiza:

- Alta disponibilidad
- Escalabilidad y resiliencia
- Redundancia geográfica

- Una IP zone-redundant es una dirección IP que está asociada con varios recursos desplegados en diferentes Availability Zones dentro de una región, lo que garantiza:
 - **Alta disponibilidad de la IP:** Si una zona de disponibilidad experimenta una falla, la IP continuará funcionando desde otra zona. Así, no se pierde conectividad, ya que el tráfico puede ser redirigido a las instancias de servicios en las zonas restantes.
 - **Balanceo automático de carga:** Las direcciones IP zone-redundant son típicamente asociadas con servicios como Azure Load Balancer, los cuales automáticamente distribuyen el tráfico entre diferentes zonas, de manera que los usuarios no noten interrupciones, incluso si alguna zona falla.

7. ¿Cuál es el propósito del Network Security Group?

El Network Security Group es un conjunto de reglas de seguridad que controlan el tráfico de red hacia y desde los recursos en una red virtual. Las reglas definidas en un NSG se utilizan para permitir o denegar el tráfico entrante y saliente de los recursos, como máquinas virtuales, instancias de base de datos o cualquier otro servicio dentro de una VNet.

Propósito:

- **Control de acceso a nivel de red:** Permite establecer reglas de firewall a nivel de red para controlar qué tráfico está permitido o bloqueado en función de criterios como direcciones IP de origen y destino, puertos y protocolos.
- **Seguridad y segmentación:** Ayuda a segmentar el tráfico entre diferentes recursos dentro de una red, como entre subredes y a proteger esos recursos de accesos no autorizados o ataques externos.
- **Control granular de tráfico:** Se compone de reglas que permiten especificar detalladamente:
 - Dirección IP de origen y destino.

- Puertos a los que se permite o deniega acceso.
- Protocolos como TCP, UDP o ICMP.
- Dirección de tráfico.
- **Seguridad en capas:** Proporcionan una capacidad de control de acceso basada en capas, permitiendo configurar reglas específicas para diferentes segmentos de la infraestructura.

8. Informe de newman 1 (Punto 2)

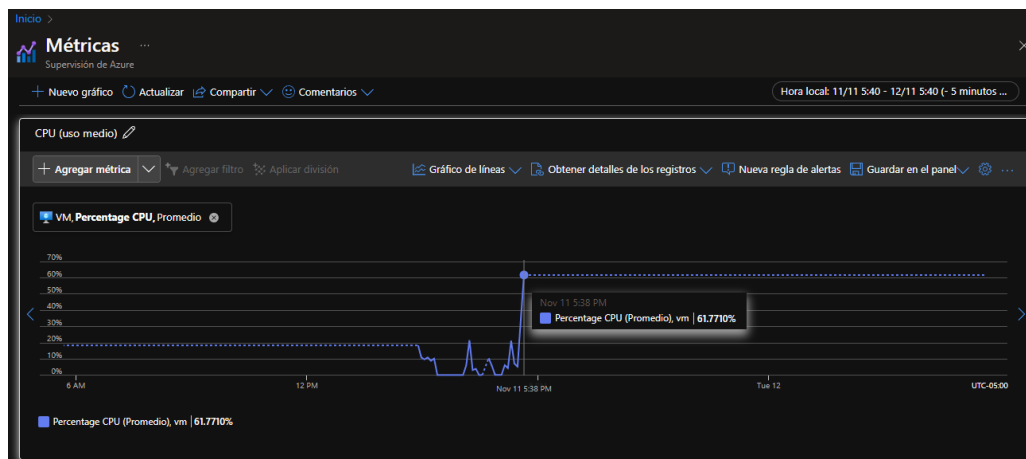
Respuestas newman máquina B2ms 2 peticiones

	executed	failed
iterations	10	0
requests	10	4
test-scripts	10	0
prerequisite-scripts	0	0
assertions	0	0
total run duration: 1m 36.2s		
total data received: 1.25MB (approx)		
average response time: 10.9s [min: 8.7s, max: 17.6s, s.d.: 3.8s]		

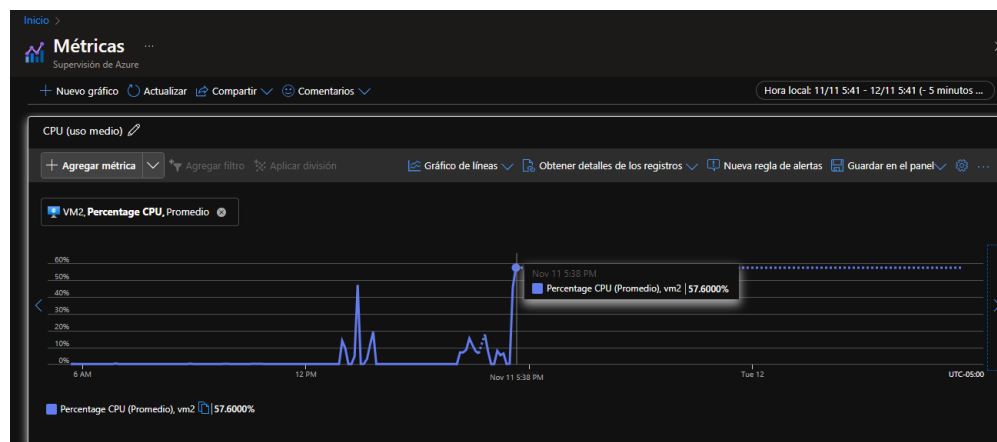
Respuestas newMan balanceador de carga 2 peticiones

```
vm1@VM1: ~/ARSW/Fibonacc x + v
GNU nano 4.8 [ARSW_LOAD-BALANCING_AZURE]
{
  "id": "373cbe10-3dd7-4a9a-8519-bb96bfcd4b5",
  "name": "[ARSW_LOAD-BALANCING_AZURE]",
  "values": [
    {
      "key": "loadbalancer",
      "value": "74.178.111.69",
      "enabled": true
    },
    {
      "key": "nth",
      "value": "1000000",
      "enabled": true
    }
  ],
  "_postman_variable_scope": "environment",
  "_postman_exported_at": "2019-11-08T01:10:51.170Z",
  "_postman_exported_using": "Postman/7.9.0"
}
```

Métricas VM



Métricas VM2



Tablas newman

	executed	failed
iterations	10	0
requests	10	0
test-scripts	10	0
prerequest-scripts	0	0
assertions	0	0
total run duration: 2m 29.8s		
total data received: 2.09MB (approx)		
average response time: 14.9s [min: 14.7s, max: 15.4s, s.d.: 214ms]		

	executed	failed
iterations	10	0
requests	10	0
test-scripts	10	0
prerequest-scripts	0	0
assertions	0	0
total run duration: 3m 28.6s		
total data received: 2.09MB (approx)		
average response time: 20.7s [min: 14.9s, max: 1m 9.4s, s.d.: 16.2s]		

Máquina virtual	Tiempo total de ejecución	Tiempo promedio por petición	Cantidad de peticiones enviadas	Cantidad de peticiones respondidas con éxito
B2ms	1 min 36.2s	10.2 s	10	6
VM	2m 29.8s	14.9s	10	10
VM2	3m 28.6s	20.7s	10	10

Costos:

- Escalabilidad Vertical

- Una VM B2ms (Standard B2ms): 2 vCPU, 8 GB RAM.
- Disco de Almacenamiento: 16 GB.
- Inbound Port Rule: Sin costo adicional, solo es una configuración de red.

Costo Estimado:

- VM B2ms: Aproximadamente \$60 USD al mes para uso continuo.
- Disco: Aproximadamente \$1-2 USD al mes.

Costo total mensual para Escalabilidad Vertical: Aproximadamente \$61-62 USD al mes.

- Escalabilidad Horizontal

- Dos VMs B1ls (Standard B1ls): En diferentes zonas de disponibilidad, con 1 vCPU y 0.5 GB RAM cada una.
- Balanceador de Carga: Para manejar el tráfico hacia las VMs.
- Discos para las VMs: Cada VM necesitará un disco OS.
- IP Pública para cada VM: Si se configura como estándar, hay costos adicionales.

Costo Estimado:

- Dos VMs B1ls a \$5-8 USD cada una, totalizando entre \$10-16 USD al mes.
- Balanceador de Carga Estándar: En Azure, un balanceador de carga estándar puede costar aproximadamente entre \$18-20 USD al mes**.
- Dos discos estándar HDD, aproximadamente \$3-6 USD en total.
- IP Pública Estándar en el rango de \$3-4 USD por IP en Azure.

Costo total mensual para Escalabilidad Horizontal: Aproximadamente \$40-54 USD al mes.

Infraestructura	Costo Aproximado (USD)
Escalabilidad Vertical	\$61-\$62
Escalabilidad Horizontal	\$40 - \$54

Si el costo es un factor clave y se requiere una solución más flexible y redundante, la escalabilidad horizontal es la mejor opción. A pesar de que cada VM individual tiene menos capacidad (1 vCPU y 0.5 GB RAM), el balanceador de carga y la distribución de carga entre múltiples VMs ofrecen una infraestructura más robusta para manejar grandes volúmenes de tráfico, con un costo total mensual más bajo (\$40-54 USD).

Por otro lado, si se prioriza simplicidad y rendimiento en una sola máquina, y el costo no es un obstáculo importante, la escalabilidad vertical (con la VM B2ms) es una opción sólida. Ofrece más potencia de procesamiento con 2 vCPUs y 8 GB de RAM, pero a un costo mensual más alto (\$61-62 USD).

9. Presente el Diagrama de Despliegue de la solución.

