75.73 - Arquitectura de Software

Trabajo Práctico N°1

Mayo 2020

Alumno	**Padrón**	**Mail**
Jorge Anca	82399	janca@fi.uba.ar
M. Eugenia Mariotti	96260	mmariotti@fi.uba.ar

Objetivo

El objetivo principal es familiarizarse con algunas tecnologías para la medición de un sistema bajo carga y analizar diversos aspectos que impactan en los atributos de calidad. Asimismo otro objetivo es probar, o al menos sugerir, qué cambios se podrían hacer para mejorarlos.

Tecnologías utilizadas

Para realizar el presente trabajo práctico se utilizaron las siguientes tecnologías:

- Node.js (+ Express)
- Python (+ Flask y Gunicorn)
- Docker
- Docker-compose
- Nginx
- Artillery
- cAdvisor
- StatsD
- Graphite
- Grafana

Sección 1

En esta sección nos dedicaremos a realizar pruebas de performance sobre los endpoints *ping*, *timeout* e *intensivo* con el objetivo de comparar sus capacidades de respuesta según la tecnología utilizada. Para este trabajo se decidió utilizar las tecnologías *node.js* y *python* para dicho trabajo comparativo.

Las pruebas buscarán analizar la capacidad de respuesta de los endpoints y evaluar atributos básicos de calidad, como disponibilidad y performance, a partir de la comparación de métricas en distintos escenarios de prueba, que diseñaremos para tal fin. Los escenarios de prueba, consistirán en simulaciones de uso con variaciones en los recursos disponibles, cantidad de carga y tiempos de exposición a la prueba.

Endpoints

El trabajo pone a prueba 3 endpoints distintos:

1. **Timeout**: responde luego de un período de timeout de 5 segundos.
2. **Ping**: responde instantáneamente.
3. **Intensivo**: realiza un busy wait de 500ms y luego responde.

Configuraciones

A continuación, presentaremos cada uno los endpoints descritos con los resultados obtenidos para las siguientes configuraciones:

1. Único servidor de python.
2. Único servidor de node.
3. Servidor de python replicado 3 veces.

Escenarios

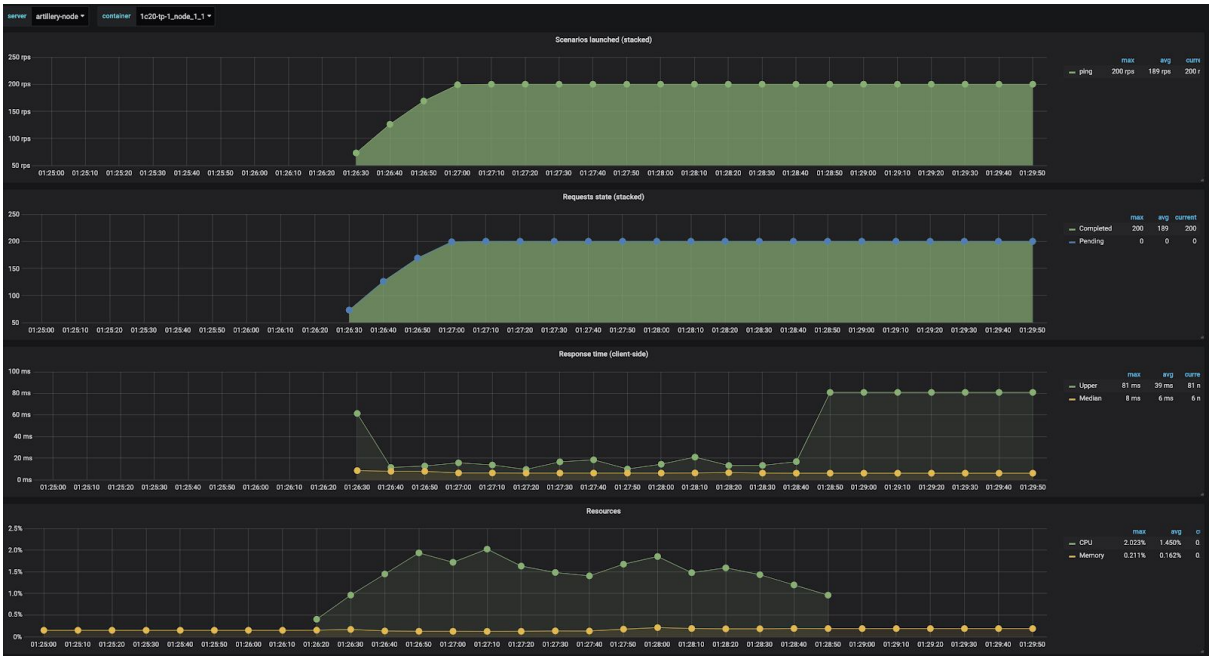
Se diseñaron 2 escenarios de mediana y alta intensidad para poner a prueba el endpoint intensivo en las distintas configuraciones. Para los endpoints de *ping* y *timeout* utilizaremos escenarios cortos y de menor intensidad, de forma tal de utilizar estos casos como testigos.

Ping

Fase	Duración (seg)	Arrival Rate	Ramp To	Arrival Count
Ramp	30	5	20	-
Linear Ramp	120	20	-	-

1. Único servidor de python

Gráfico comparativo: Scenarios Launched vs. Requests State vs. Response Time vs. CPU and Memory Usage



2. Único servidor de node

Gráfico comparativo: Scenarios Launched vs. Requests State vs. Response Time vs. CPU and Memory Usage

3. Resúmenes de Corrida

	Único Servidor de Node	Único Servidor de Python	Servidor de Python (3 réplicas)
Scenarios launched	2776	2791	2786
Scenarios completed	2776	2791	2786
Requests completed	2776	2791	2786
Mean response/sec	18.4	18.52	18.75
Response time (msec)			
mínimo	3.2	3.5	3.2
máximo	79.6	18	22.4
mediana	4.6	5	4.7
p95	6.1	6.7	6.2
p99	9.9	9.8	9.9

Scenario counts			
ping	2776 (100%)	2791 (100%)	2786 (100%)
Codes			
200 *OK*	2776 (100%)	2791 (100%)	2786 (100%)
502 *Bad Gateway*	-	-	-
504 *Gateway Timeout*	-	-	-
Errors			
ECONNRESET	-	-	-
ESOCKETTIMEDOUT	-	-	-

4. Análisis de Resultados

Podemos observar que para el caso de este endpoint tenemos resultados favorables tanto en el caso del servidor de Python como en el de Node, ya que todas las requests efectuadas por el tests retornan una respuesta favorable (200: OK) y no se detectaron errores, timeouts o reset de conexión. Como primer análisis podemos observar que la cantidad de respuestas promedio por segundo es ligeramente mayor para el servidor de Python que para el servidor de Node. Cuando agregamos réplicas, este número aumenta pero no dramáticamente.

Por otra parte, algo que también podemos notar es que, para el servidor de Node, la variación entre el mínimo y máximo tiempo de respuesta obtenidos es considerablemente mayor que para el servidor de Python, donde el máximo tiempo de respuesta se mantiene alrededor de los 20 milisegundos tanto con replicación como sin ella.

Timeout

Fase	Duración (seg)	Arrival Rate	Ramp To	Arrival Count
Ramp	30	5	20	-
Linear Ramp	120	20	-	-

1. Único servidor de python

Gráfico comparativo: Scenarios Launched vs. Requests State vs. Response Time vs. CPU and Memory Usage

2. Único servidor de node

Gráfico comparativo: Scenarios Launched vs. Requests State vs. Response Time vs. CPU and Memory Usage

Comienzo prueba: 19:41:42 hs - Fin prueba: 19:44:18 hs

3. Servidor de python replicado 3 veces

Gráfico de Latencia Máxima

Comienzo prueba: 16:54:12 hs - Fin prueba: 16:58:29 hs

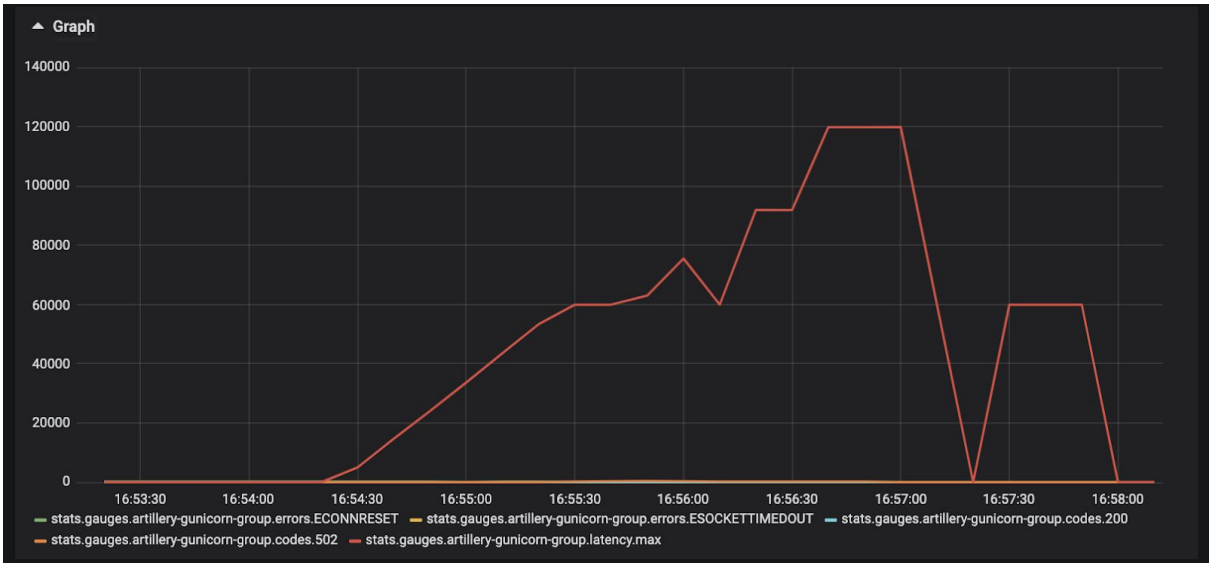


Gráfico de Códigos de Respuesta

Comienzo prueba: 16:54:12 hs - Fin prueba: 16:58:29 hs

4. Resúmenes de Corrida

	Único Servidor de Node	Único Servidor de Python	Servidor de Python (3 réplicas)
Scenarios launched	2747	2772	2764
Scenarios completed	2747	1376	2217
Requests completed	2747	1376	2217
Mean response/sec	17.68	11.55	10.55
Response time (msec)			
min	4997.3	5013.5	2.5
máx	5015.4	91910.2	91901.7
median	5000.3	59935.7	3.7
p95	5003.2	62384.7	59936.8
p99	5005.3	91646.4	64359.5
Scenario counts			
timeout	2747 (100%)	2772 (100%)	2764 (100%)
Codes			
200 *OK*	2747 (100%)	12 (0.43%)	38 (1.38%)

502 *Bad Gateway*	-	-	2179 (78.83%)
504 *Gateway Timeout*	-	1364 (49.21%)	-
Errors			
ECONNRESET	-	1396 (50.36%)	542 (19.61%)
ESOCKETTIMEDOUT	-	-	5 (0.18%)

5. Análisis de Resultados

Para este caso, podemos ver una distinción muy clara entre el servidor de Node y el de Python. Ni siquiera el endpoint replicado de Python responde con la misma capacidad que el servidor de Node, en el cual casi no se observa variación en la cantidad de respuestas promedio/segundo respecto del endpoint *ping*. Para el servidor de Python, sin embargo, vemos que este número cayó a la mitad, aún cuando se agregan las réplicas.

Por otra parte, vemos que el servidor de Node mantiene sus respuestas alrededor de los 5 segundos, mientras que para Python, el tiempo máximo de respuesta llega a los 91 segundos. Estos tiempos de respuesta, van en línea con los errores por *socket timeout* y *connection reset* que podemos observar en su resúmen de corrida y en los gráficos, donde vemos que a medida que pasa el tiempo, los códigos de error y la latencia de las respuestas aumentan.

El motivo por el cual obtenemos resultados tan diferentes es porque, en el caso de Python, tenemos un único thread dedicado a atender las requests. Es decir, una request no será atendida hasta que no se termine de procesar la anterior. Entonces, si tenemos un endpoint como el de *timeout*, donde la request tarda 5 segundos en ser procesada, el endpoint quedará bloqueado e imposibilitado de atender requests durante todo ese tiempo. En cambio en el servidor de Node, el timeout no bloquea la ejecución, sino que la tarea se delega al kernel (que puede ejecutar la tarea en background usando multithreading), lo cual permite al servidor seguir atendiendo requests.

Intensivo

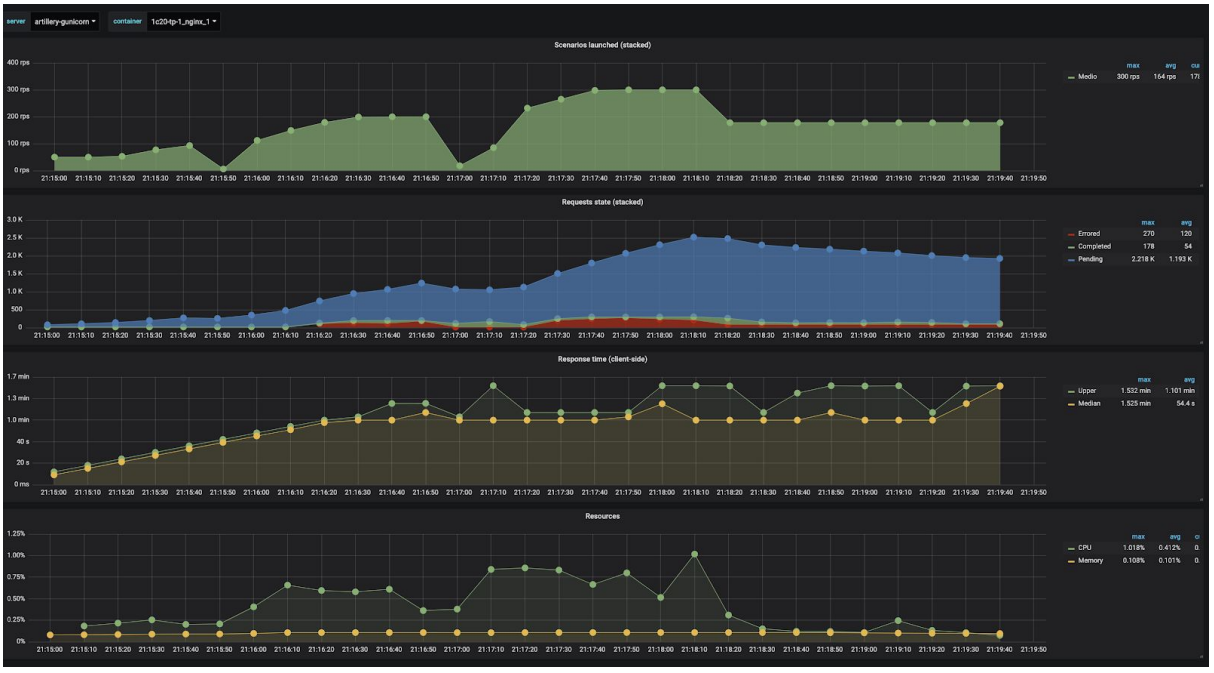
Para este endpoint usaremos los escenarios de prueba de intensidad **media** y **alta**. Se presentan los resultados de ambos aplicados al endpoint *intensivo*, en sus distintas configuraciones, a continuación. Luego de presentar los resultados y gráficos obtenidos para ambos escenarios, realizaremos un análisis de los mismos al final de esta sección.

1. Escenario Intensidad Media

Fase	Duración (seg)	Arrival Rate	Ramp To	Arrival Count
Warm Up	30	5	-	-
Linear Ramp	30	5	10	-
Pause	10	-	-	-
Linear Ramp 2	30	10	20	-
Plane	30	20	-	-
Pause 2	15	-	-	-
Linear Ramp 3	30	20	30	-
Medium Plane	40	30	-	-

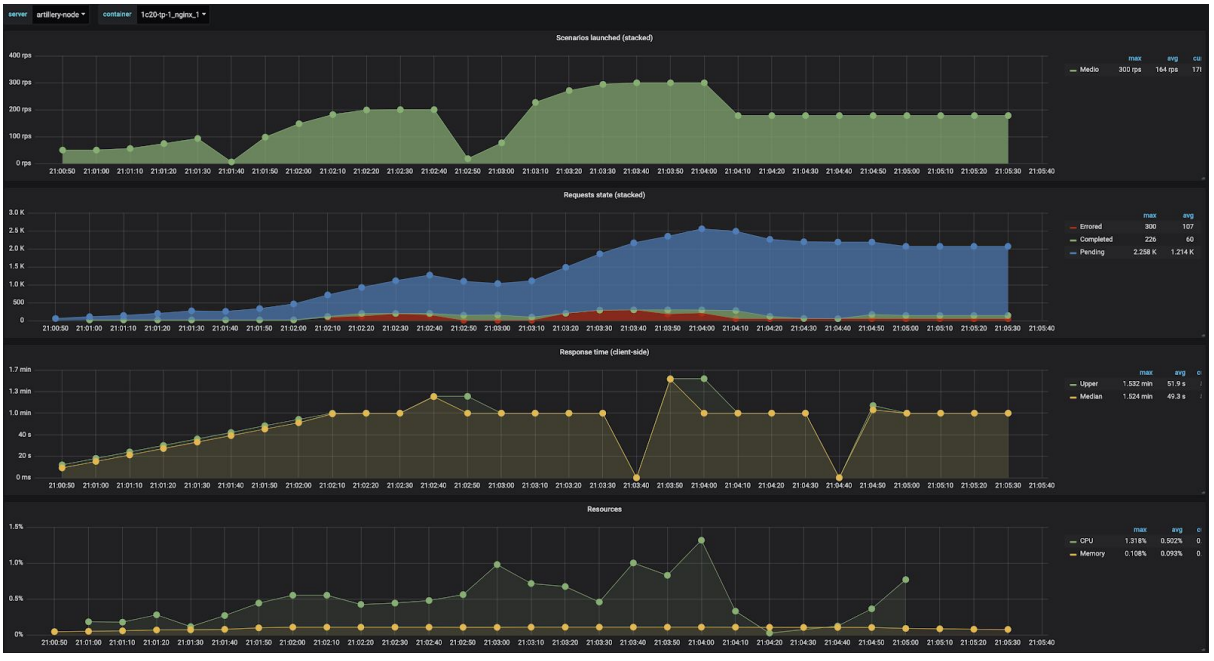
1.1. Único servidor de python

Gráfico comparativo: Scenarios Launched vs. Requests State vs. Response Time vs. CPU and Memory Usage



1.2. Único servidor de node

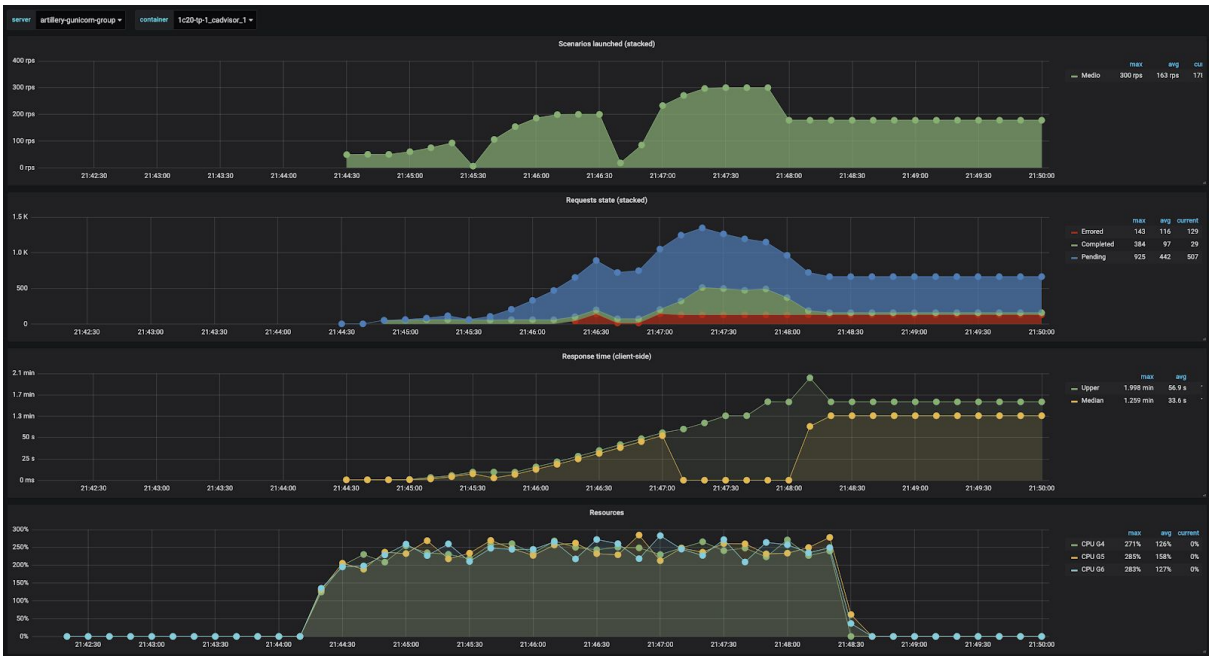
Gráfico comparativo: Scenarios Launched vs. Requests State vs. Response Time vs. CPU and Memory Usage



1.3. Servidor de python replicado 3 veces

Gráfico comparativo: Scenarios Launched vs. Requests State vs. Response Time vs. CPU Usage

Comienzo prueba: 21:44:13 - Fin prueba: 21:48:23



1.4. Resúmenes de Corrida

	Único Servidor de Node	Único Servidor de Python	Servidor de Python (3 réplicas)
Scenarios launched	3370	3393	3417
Scenarios completed	1569	1596	2942
Requests completed	1569	1596	2942
Mean response/sec	12.21	11.03	13.87
Response time (msec)			
min	523.9	508.8	2.9
máx	91931.8	91936.1	91937.1
median	59935.9	59935.7	13.4
p95	75561.4	91474.3	75295.2

p99	91837.1	91836.3	91824.2
Scenario counts			
medio	3370 (100%)	3393 (100%)	3417 (100%)
Codes			
200 *OK*	194 (5.76%)	197 (5.81%)	959 (28.07%)
502 *Bad Gateway*	-	54 (1.59%)	1983 (58.03%)
504 *Gateway Timeout*	1375 (40.80%)	1345 (39.64%)	-
Errors			
ECONNRESET	1801 (53.44%)	1797 (52.96%)	473 (13.84%)
ESOCKETTIMEDOUT	-	-	2 (0.06%)

2. Escenario Intensidad Alta

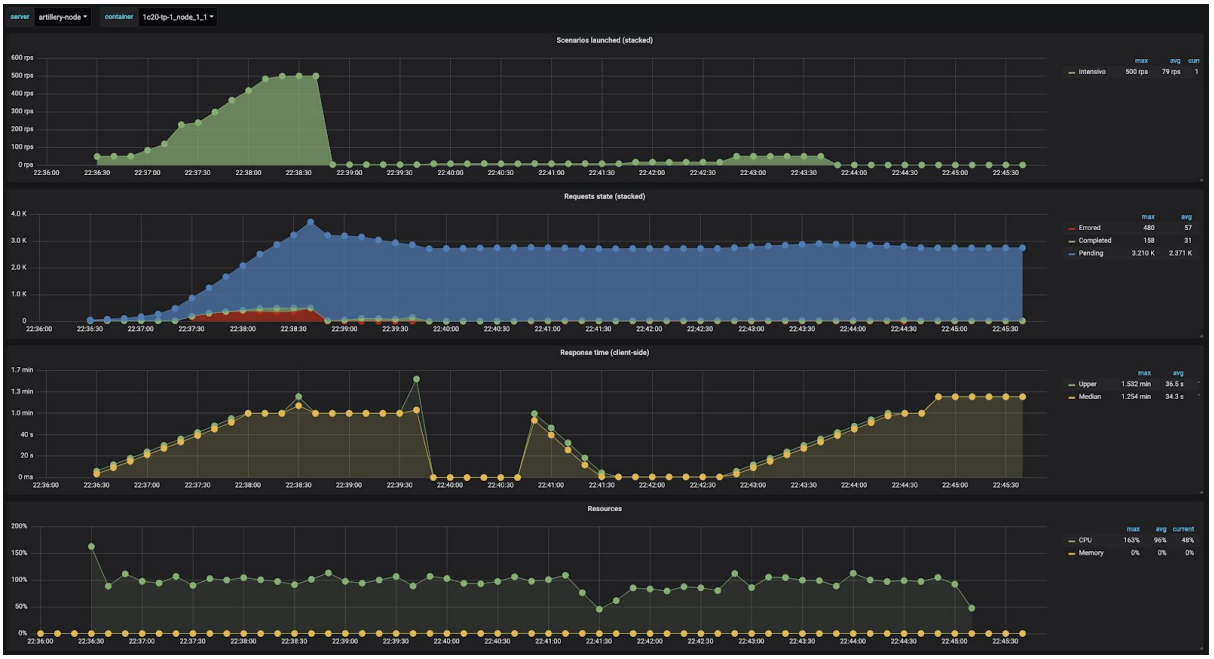
Fase	Duración (seg)	Arrival Rate	Ramp To	Arrival Count
Warm Up	30	5	-	-
Linear Ramp	80	5	50	-
High Plain	30	50	-	-
Pause	60	-	-	-
Non-linear Ramp	120	-	-	100
Non-linear Ramp 2	60	-	-	100
Active Pause	30	5	-	-
Non-linear Ramp 3	30	-	-	150

2.1. Único servidor de python

Gráfico comparativo: Scenarios Launched vs. Requests State vs. Response Time vs. CPU and Memory Usage

2.2. Único servidor de node

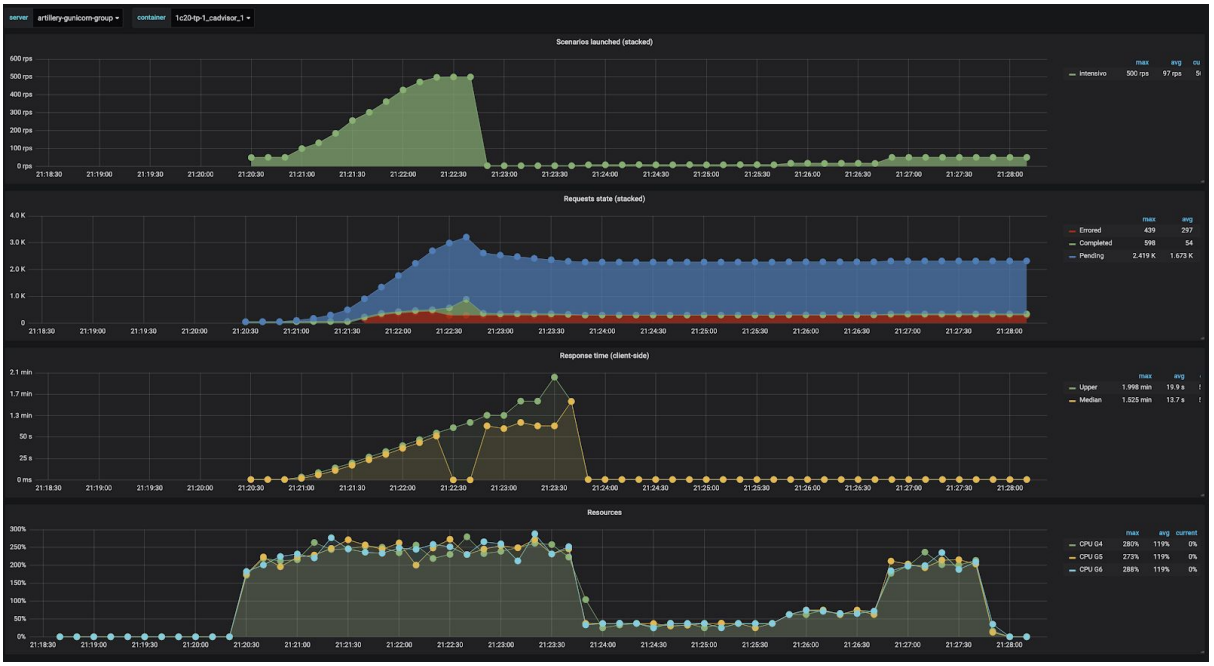
Gráfico comparativo: Scenarios Launched vs. Requests State vs. Response Time vs. CPU and Memory Usage



2.3. Servidor de python replicado 3 veces

Gráfico comparativo: Scenarios Launched vs. Requests State vs. Response Time vs. CPU Usage

Comienzo prueba: 21:20:24 - Fin prueba: 21:27:45



2.4. Resúmenes de Corrida

	Único Servidor de Node	Único Servidor de Python	Servidor de Python (3 réplicas)
Scenarios launched	4383	4313	4378
Scenarios completed	1681	1682	2452
Requests completed	1681	1682	2452
Mean response/sec	8.51	8.14	9.93
Response time (msec)			
min	502.5	503.5	2.9
máx	91924.7	91931.9	119868.2
median	59934.1	59933.9	505.4
p95	75346.4	75473.8	75162.9
p99	91594.1	91826.1	91705.5
Scenario counts			
intensivo	4383 (100%)	4313 (100%)	4378 (100%)
Codes			
200 *OK*	588 (13.42%)	595 (13.79%)	1212 (27.68%)
502 *Bad Gateway*	-	18 (0.42%)	1240 (28.32%)
504 *Gateway Timeout*	1093 (24.94%)	1069 (24.79%)	-
Errors			
ECONNRESET	2702 (61.65%)	2631 (61%)	1925 (43.97%)
ESOCKETTIMEDOUT	-	-	1 (0.02%)

3. Análisis de Resultados

En ambos casos, vemos un porcentaje similar de respuestas exitosas para los servidores de Node y Python, aproximadamente 14% del total. Solamente vemos que aumenta la capacidad de respuesta cuando agregamos replicación al servidor de Python. Además, puede observarse claramente en el gráfico de recursos para los casos con replicación, que la capacidad de recuperación de los mismos es mucho más alta (vemos que el uso de cpu baja bruscamente en la etapa de pausa) que para los endpoints sin

replicación, lo cual probablemente se relacione con que el número de requests acumuladas para este endpoint es menor.

Por otro lado, también observamos que para el endpoint con replicación el mínimo tiempo de respuesta es dramáticamente menor (2.9 milisegundos) que para los otros dos. Podemos asociar este valor a los errores de **ESOCKETTIMEDOUT** que vemos en el resumen de corrida. Lo que sucede es que frente a una sobrecarga de requests para procesar, nginx marca los upstreams como no disponibles y responde con **110: Connection timed out :**

```
nginx_1 | 2020/05/25 01:05:03 [error] 7#7: *2742 upstream timed out (110: Connection timed out) while reading response header from upstream, client: 192.168.0.1, server: , request: "GET /gunicorn_group/timeout HTTP/1.1", upstream: "http://192.168.0.7:8000/timeout", host: "localhost:5555"
nginx_1 | 2020/05/25 01:05:03 [error] 7#7: *2742 no live upstreams while connecting to upstream, client: 192.168.0.1, server: , request: "GET /gunicorn_group/timeout HTTP/1.1", upstream: "http://gunicorn_group/timeout", host: "localhost:5555"
```

La razón por la que en este escenario el servidor Node no tuvo un desempeño mejor que el de ùnico servidor Python es que en el mismo hace un uso intensivo del CPU y como ambos tienen uno sólo hilo de ejecución no pueden distribuir la carga a otros procesadores. Es decir el uso de CPU es el cuello de botella para ambos servidores. En cambio, el servidor Python replicado puede distribuir la carga más de un procesador (en este caso 3) por lo cual puede completar más pedidos.

Opcional mas de un hilo de ejecución

Creamos un servidor node en dos hilos utilizando el módulo cluster y corrimos el escenario intensivo, obteniendo los siguientes resultados :

	Único Servidor de Node	**Servidor de Node 2 hilos**	
Scenarios launched	4383	4350	
Scenarios completed	1681	1838	
Requests completed	1681	1838	
Mean response/sec	8.51	9.44	
Response time (msec)			
min	502.5	512.7.	
máx	91924.7	60234.2	

median	59934.1	60000.2	
p95	75346.4	60017.2	
p99	91594.1	60055.2	
Scenario counts			
intensivo	4383 (100%)	4350(100%)	
Codes			
200 *OK*	588 (13.42%)	881 (20.2 %)	
502 *Bad Gateway*	-		
504 *Gateway Timeout*	1093 (24.94%)	957 (22 %)	-
Errors			
ECONNRESET	2702 (61.65%)	2512(57.8 %)	
ESOCKETTIMEDOUT	-	-	

Analizando la tabla vemos que aumentó casi un 50 % los pedidos con respuesta 200 , los cuales pasaron de 588 a 881. Asimismo cabe destacar que si bien prácticamente no varió la mediana del tiempo de respuesta , sí disminuyó enormemente la varianza del mismo.

Podemos concluir que aumentar la concurrencia con Node nos permite responder de manera similar la replicación del servidor Python en casos de acciones que requiera el uso intensivo del procesador.

Conclusiones de la sección 1

Vimos que para operaciones en la que no se requiera un uso intensivo del CPU un servidor en Node.js tiene una mayor capacidad responder a distintos pedidos y un menor tiempo de respuesta que un servidor en Python. En el caso de que ésto no se cumple no hay grandes diferencias entre los mismos.

En conclusión si necesitamos atender muchos pedidos con bajo procesamiento conviene en lo posible usar Node. En cambio si los pedidos requieren el uso intensivo del CPU se podría elegir cualquiera de las dos tecnologías replicando los servidores o correr en tantos hilos como procesadores tenga el servidor.

Sección 2

Set up

El container puede correrse invocando el siguiente comando:

```
$ docker run -it --name mystery_service -p 9090:9090 -p 9091:9091 \
           -v $(pwd)/mystery_service.properties:/opt/bbox/config.properties \
           arqsoft/bbox:202001.1
```

Análisis y Caracterización

Para caracterizar el servicio usamos un escenario similar al ping. En donde denominamos serverX al servicio que está asignado al puerto 9090 y serverY al puerto 9091.
Para correr el escenario se debe ejecutar:

- ./run-scenario.sh helloMistery serverY
- ./run-scenario.sh helloMistery serverY

Servicio #1: 9090

Para analizar este endpoint corrimos el escenario y obtuvimos los siguientes resultados de artillery:

```
All virtual users finished
Summary report @ 15:26:00(-0300) 2020-05-28
  Scenarios launched:  2786
  Scenarios completed: 1973
  Requests completed:  1973
  Mean response/sec: 10.3
  Response time (msec):
    min: 651.4
    max: 119873
    median: 41229.7
    p95: 97806.5
    p99: 115690.4
  Scenario counts:
    Hello Word: 2786 (100%)
  Codes:
    200: 1973
  Errors:
    ESOCKETTIMEDOUT: 813
```

Asimismo visualizamos los resultados mediante grafana:

Del gráfico y de los resultados de artillery podemos ver la gran cantidad de errores y la variabilidad en el tiempo de respuesta.

Sincrónico o Asincrónico?

Por la alta variabilidad de los tiempos de respuesta y la cantidad de requests erróneas obtenidas, podemos inferir que se trata de un endpoint sincronico. El hecho de que los errores sean de tipo ESOCKETTIMEDOUT sugiere que se encolaron más requests de las que el endpoint podía manejar.

Cantidad de Workers?

Podemos observar que se obtuvo una cantidad de 10.34 responses/segundo. Si tomamos el tiempo de respuesta mínimo podemos realizar el siguiente cálculo:

10.3 responses/segundo * 0.651 segundos = 6.7 responses

Con lo cual podríamos estimar un total de 7 workers.

Response Time?

El tiempo de respuesta mínimo para este endpoint fue de 651.4 milisegundos, que asumimos es el tiempo de respuesta optimo. Mientras que el tiempo de respuesta máximo fue de 2 minutos.

Servicio #2: 9091

Para analizar este endpoint corrimos el escenario y obtuvimos los siguientes resultados de artillery:

```
All virtual users finished
Summary report @ 15:46:23(-0300) 2020-05-28
  Scenarios launched:  2768
  Scenarios completed: 2768
  Requests completed:  2768
  Mean response/sec: 18.33
```

Response time (msec):
  min: 800.7
  max: 932.1
  median: 801.5
  p95: 806.7
  p99: 818.3
Scenario counts:
  Hello Word: 2768 (100%)
Codes:
  200: 2768

Vemos en este caso, a diferencia que en el caso anterior, se completaron todos los pedidos y que la varianza de los tiempos de respuestas fue menor.

Sincrónico o Asincrónico?

Por la uniformidad en los tiempos de respuesta, y la capacidad de respuesta de este endpoint, podemos inferir que se trata de un endpoint asincrónico y el 99 % de los pedidos fueron completados en menos 818.3 ms.

Response Time?

La mediana del tiempo de respuesta obtenido para este endpoint, como puede observarse en los resultados obtenidos, es de 801 milisegundos.