Avaliação de desempenho de APIs Rest em NodeJS e Python

Alisson Allebrandt, Diego Henrique Schmidt

O desenvolvimento de APIS Rest dentro da área de software é um conceito inserido no paradigma de desenvolvimento em 2000 por Roy Fielding. Deste então o seu uso tem se intensificado devido a sua facilidade de comunicação com outras aplicações, permitindo realizar alterações no estado da aplicação de forma simples, rápida e segura. Dentre alguns pontos que devem ser considerados na construção de uma API Rest está a linguagem de programação que será utilizada, buscando escolher uma que possua desempenho satisfatório. Diante disso, este artigo irá comparar rotas de uma API Rest nas linguagens Node JS e Python buscando avaliar qual das duas possui um melhor desempenho de resposta, e quantidade de requisições aceitas em períodos de tempo variados.

I. PLANEJAMENTO

Para a realização da avaliação de desempenho escolheu-se as linguagens de programação Node JS e Python, criando um modelo de API Rest, ambas construídas com 4 rotas de comunicação, com métodos GET, POST, PUT e DELETE. Mais detalhes serão explicados durante as próximas seções.

O objetivo desta avaliação de desempenho é medir a performance de duas API Rest que possuem os mesmos métodos de entrada e retorno, porém construídas em duas linguagens diferentes, visando detectar qual das linguagens possui uma resposta melhor e consegue processar o maior número de requisições em um determinado período de tempo. As linguagens foram eleitas devido a sua popularidade no mercado, estando como primeiras do rank na pesquisa realizada pela RedMonk[2] em janeiro de 2020.

Para auxiliar no processo de avaliação será utilizada a ferramenta JMEETER[1], criada especificamente para realização de testes de stress em aplicações, apis e web services em geral. Com esta ferramenta pretende-se avaliar as requisições feitas para as APIS, tendo como resultado uma interpretação de qual tecnologia trabalha melhor quando submetidas a uma grande quantidade de requisições

As métricas que serão levadas em consideração na avaliação será o throughput (envio de requisições/segundo) tempo médio de resposta, porcentagem de erros no envio, quantidade máxima de requisições enviadas para cada método e quantidade total de requisições enviadas.

Para avaliar as métricas, a técnica de avaliação utilizada será a técnica de medição, buscando apresentar resultados realísticos de envio de requisições para as duas diferentes apis que serão consumidas no servidor. Por meio desta técnica será possível medir qual das duas tecnologias possui uma maior performance quanto ao suporte de requisições simultâneas, consumindo menos poder computacional para isso.

Nesta avaliação será aplicada uma carga de trabalho com base em quantidade de usuários simultâneos realizando requisições, e o tempo de envio total para o estresse da API. Realizando medições em 6 cenários diferentes. A análise dos dados coletados será realizada por meio da comparação entre os resultados obtidos das duas APIS, para cada cenário separadamente e a apresentação deste dados se dará por meio de tabelas, médias aritméticas e gráficos de linha para demonstração da latência nas respostas de cada requisição.

II. DESCRIÇÃO DO EXPERIMENTO

Esta seção irá descrever as configurações de Hardware e Software utilizados para a execução da avaliação de desempenho. Para hospedagem da aplicação de teste Jmeeter e APIS em Node JS e Python foi utilizado a plataforma em Cloud do Google.

A. Hardware

O ambiente de aplicação foi montando visando um menor impacto nos testes durante a troca de tecnologias entre Node JS e Python, dessa forma criou-se uma máquina virtual e dentro da mesma utilizou-se um container em docker com as mesmas configurações para ambos os testes, mudando apenas a porta de comunicação entre uma API e outra. As configurações da máquina virtual no nível de aplicação podem ser vistas na tabela 1.

Tabela 1 - Hardware da VM de aplicação

Sistema OS	Debian GNU/Linux 10
Processador	Intel Xeon 2.20GHz (2 cores)
Ram	7680MB
HD	10GB

O ambiente de testes foi montado separadamente para a instalação do Jmetter, buscando evitar qualquer interferência com o ambiente de aplicação durante os testes. As configurações de hardware da máquina virtual deste ambiente podem ser vistas na tabela 2.

Tabela 2 - Hardware da VM de testes

Sistema OS	Windows Server 2012 R2
Processador	Intel Xeon 2.20GHz (2 cores)
Ram	7680MB
HD	50GB

B. Software

No que diz respeito ao software instalado em cada uma das máquinas virtuais, na vm de aplicação foi instalado os pacotes e bibliotecas conforme lista 1.

Lista 1 - Softwares e bibliotecas instaladas na VM de aplicação.

Node JS, versão v10.20.1
NPM, versão 6.14.4
Express, versão 4.17.1
Python, versão 3.8.3
Flask, versão 1.1.2
uWSGI, versão 2.0.18
Request, versão 2019.4.13
Jinja2, versão 2.11.2

Na VM de testes o único software instalado foi o Jmeeter na versão 5.2, e também o Java JDK versão 14.0.1 que é uma das dependências necessárias para a correta execução do software.

III. Análise e apresentação dos resultados

Nesta seção serão apresentados os principais resultados obtidos durante a avaliação de desempenho entre as duas API estudadas. O restante dos resultados e informações de desempenho podem ser encontrados no repositório público do github desta avaliação.

As medições foram realizadas para ambas as API em 6 cenários diferentes, porém neste artigo serão colocados os 2 principais, conforme Tabela 3.

Tabela 3 - Cenários de teste

Cenário 1	100 usuários por 60 segundos
Cenário 2	300 usuários por 5 minutos

Conforme os cenário levantados podemos ver os resultados obtidos durante a execução dos testes com a API em Node JS conforme tabela 4.

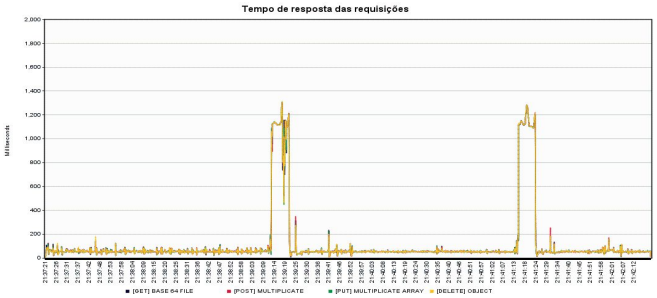
Tabela 4 - Conjunto de resultados obtidos para API em Node JS

Cenário 1				
Método	Total/Req	Média/Req/Seg	Max/Req/Seg	% Erro
GET	88183	16	84	0%
POST	88151	16	84	0%
PUT	88217	16	82	0%
DELETE	88103	16	76	0%
Cenário 2				
Método	Total/Req	Média/Req/Seg	Max/Req/Seg	% Erro
GET	399614	56	9130	0.65%
POST	399541	56	9131	0.65%
PUT	399457	56	9132	0.65%
DELETE	399386	56	9122	0.65%

Na tabela 4 é possível acompanhar o Total/Req (total de requisições enviadas), Média/Req/Seg (média de envios por segundo), Max/Req/Seg(Máximo de envios em 1 segundo) e a porcentagem de erro, separados pelos métodos GET, POST, PUT e DELETE.

O throughput médio, com base em avaliação do cenário 2, ficou na faixa de 1,5 ms, conforme pode ser visto na figura 1.

Figura 1 - Throughput das requisições



Os testes para a API de Python foram aplicados sob o mesmo cenário apresentado na Tabela 3. Os resultados obtidos são apresentados na Tabela 5.

Tabela 5 - Conjunto de resultados obtidos para API em Python

Cenário 1				
Método	Total/Req	Média/Req/Seg	Max/Req/Seg	% Erro
GET	49169	36	3083	31,40%
POST	49133	18	2005	68,54%
PUT	49116	37	2649	31,44%
DELETE	49087	19	2381	68,52%
Cenário 2				
Método	Total/Req	Média/Req/Seg	Max/Req/Seg	% Erro
GET	284875	84	72288	45,35%
POST	284798	75	28576	53,51%
PUT	284722	80	72284	45.44%
DELETE	284646	74	24077	53,62%

É possível verificar que a quantidade de erros retornados é alta, isso se deve aos limites da VM utilizada para os testes, pois esta, possui uma quantidade de 2 cores para processamento e os erros são retornados quando a VM utilizada no teste atinge o seu máximo de recurso.

IV. Conclusão

Com base na avaliação de desempenho das APIs em NodeJS e Python, foi possível notar grandes diferenças no processamento das requisições, é possível notar que, a API em NodeJS, com uma VM de somente 2 cores, conseguiu processar as requisições de 100 usuários simultaneamentes, sem em momento algum ter requisições não respondidas. Já a API em python, mostrou necessitar uma VM com muito mais recurso, pois, com teste de 100 usuários por segundo, o tempo de resposta da API foi maior, assim como, a quantidade de erros é muito maior se comparada a API em NodeJS.

De modo geral, com base nos trechos de código escritos para a API em Node JS e Python, é possível constatar que a API em Node JS consegue gerenciar melhor os recursos disponíveis evitando disponibilidade de resposta de requisições, fato esse que não aconteceu na API em Python. Porém, pode ser que existam outras bibliotecas em python para gerenciamento de requisições http que consigam gerenciam melhor essa carência encontrada com a biblioteca Flask.

Referências

[1] https://jmeter.apache.org/
[2] https://redmonk.com/sogrady/2020/02/28/language-rankings-1-20/