Пресметување во облак

Дарио Митев (192007)

Резиме

Со ова истражување треба да се споредат преформансите на истата апликација хостирана на два различни начини: со користетње на архитектура базирана на сервер и serverless архитектура. За таа цел користен е Azure како cloud provider, поточно Azure App Services и Azure Functions.

Клучни зборови: NodeJS | Azure | Azure App Services | Azure functions | Server | Serverless.

Вовед

Пресметките без сервер добиваат популарност во последниве години како алтернатива на традиционалните архитектури базирани на сервери. Во модел без сервер, cloud провајдерот се грижи за обезбедување и скалирање на инфраструктурата, дозволувајќи им на програмерите да се фокусираат на пишување код без да се грижат за основната инфраструктура. Од друга страна, во архитектурата базирана на сервер, програмерите треба сами да управуваат и да ги одржуваат серверите. Иако пристапот без сервер има многу предности, како што се заштеда на трошоци и зголемена скалабилност, тоа можеби не е најдобриот избор за сите апликации. Апликациите со тешки пресметковни барања или долготрајни задачи може да имаат корист од архитектурата базирана на сервер, додека оние со непредвидлив сообраќај може да бидат подобро прилагодени за архитектура без сервер. Сепак, изборот помеѓу архитектурата базирана на сервер и без сервер ќе зависи од специфичните барања и карактеристики на апликацијата што се развива.

За апликацијата

Се работи за едноставна full-stack апликација, каде корисникот треба да прикачи текстуален фајл во кој се запишани рандом цели броеви од 0 до 1024, тој фајл да го прати до API-то (до серверот или до Azure function), каде што се прави пресметка на 2 соседни броеви и тој резултат се запишува во низа. Назад корисникот добива нова датотека со пресметаните вредности и времето за кое е извршена таа операција. Апликацијата е изработена во React и NodeJS, а серверот е Express.

Зошто NodeJS?

High performance: Поради V8, NodeJS е идеален за градење на апликации со високи перформанси, скалабилни апликации итн.

Non-blocking I/O: NodeJS може да се справи со многу истовремени (concurrent) конекции, при тоа користејќи минимални ресурси.

Packages & libraries: NodeJS има голема и активна заедница која има развиено широк опсег на пакети и библиотеки кои лесно може да се интегрираат во апликациите.

Single language development: API (backend) и UI (frontend) користат Javascript.

Избор на Cloud Provider

Се одлучив за хостирање на апликациите да го користам Azure. Според мене Azure има најдобра документација и со инсталирање на една екстензија во VS Code може сите работи да се прават преку едиторот. Исто така и самиот deploy на апликациите е прилично едноставен. За хостирање на API-то и UI користев App Services, додека за serverless се користи Azure functions.

Server-based архитектура (Azure App Services)

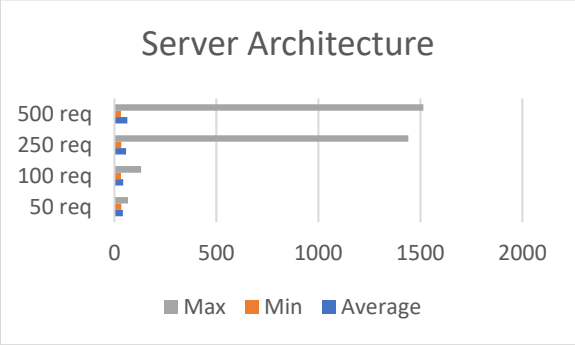
Потребно е да се креираат 2 Azure App Services веб апликации. На едната ќе биде хостирано API-то, т.е. Express сервер кој ќе се справува со повиците, а на другата ќе биде хостирана React апликација, т.е. корисничкиот интерфејс. Бидејќи овие две апликации ќе имаат различен домен, потребно е да се конфигурира CORS (Cross-Origin Resource Sharing), за да може React апликацијата да има пристап до API-то (т.е. да не добиваме 502 bad gateway).

Резултати од прв дел

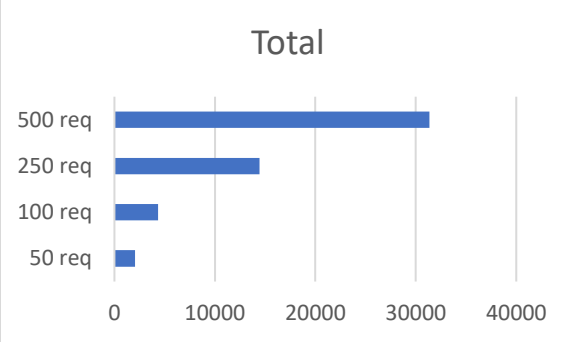
За тестирање на API-то користев Insomnia и Apache JMeter. Текстуалниот фајл што го праќам содржи 200000 броеви (од 0-1024). Одговорите со времињата од секој request ги запишував во XML документ и потоа со Javascript функција ги пресметував вкупното, просечното, минималното и максималното време. Направив неколку различни тестови и ги добив следните резултати:

Num. req	Total (ms)	Min (ms)	Max (ms)	Avg (ms)
1	46.983	46.983	46.983	46.983
50	2047.092	32.795	65.84	40.942
100	4350.747	32.575	131.56	43.507
250	14445.56	33.58	1140.7	57.782
500	31370.52	32.509	1515.392	62.741

Табела 1 – Вкупно, минимално, максимално и просечно време на извршување за различен број на истовремени повици кај архитектура со сервер



Слика 1 – Споредба на минимално, максимално и просечно време на извршување за 50, 100, 250 и 500 истовремени (server).



Слика 2 – Споредба на вкупно време на извршување за 50, 100, 250 и 500 истовремени барања (server).

Serverless архитектура (Azure Function)

Azure Function Apps овозможуваат serverless. Најпрвин, треба да се креира нова Function App.

https://dario-192007-serverless.azurewebsites.net/

Откако ќе се креира Function App потребно е да се додаде Trigger. Има повеќе видови на trigger-и (Timer, HTTP, Blob, Queue …), оној што треба да го искористиме е HTTP. Доколку на тој HTTP Trigger му дадеме име HttpTrigger2, до истиот може да го праќаме барања со ова URL:

https://dario-192007-serverless.azurewebsites.net/api/HttpTrigger2?code=qDHDNHpcOCFeQEgZvJDh7q2Z5sylISQgEHox0dWK3_whAzFudzIH2g==

Каде што query параметарот code е достапен на Azure порталот и се користи за автентикација.

За овој тригер се пишува истиот код како и кај API-то со сервер, со некои мали разлики.

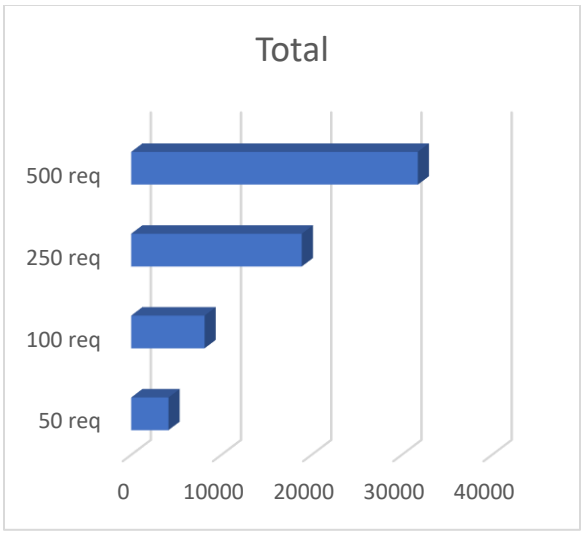
Резултати од втор дел

И овде ги направив истите тестови како кај првиот дел. Ова се резултатите:

Num. req	Total (ms)	Min (ms)	Max (ms)	Avg (ms)
1	82.856	82.856	82.856	82.856
50	4125.356	37.202	163.569	82.507
100	8126.202	40.78	172.738	81.262
250	18904.484	36.852	195.611	75.618
500	31769.377	36.612	199.469	63.666

Табела 2 – Вкупно, минимално, максимално и просечно време на извршување за различен број на истовремени повици кај serverless архитектура

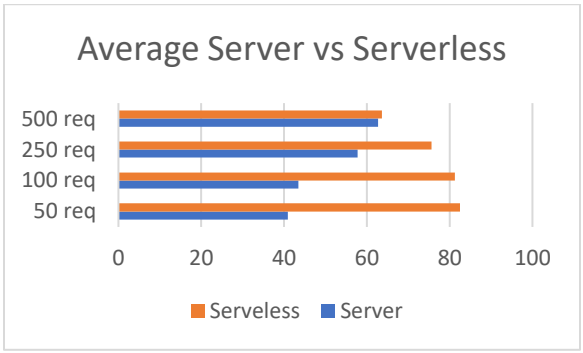
Слика 3 – Споредба на минимално, максимално и просечно време на извршување за 50, 100, 250 и 500 истовремени барања (serverless).



Слика 4 – Споредба на вкупно време на извршување за 50, 100, 250 и 500 истовремени барања (serverless).

Споредба

На *слика 5* јасно се гледа дека како што се зголемуваат бројот на барања испратени во исто време, просечното време за пресметка кај сервер архитектурата се зголемува а кај serverless се намалува.



Слика 5 – Споредба на просечно време на извршување за 50, 100, 250 и 500 истовремени барања (server vs serverless).

.

Заклучок

Од ова истражување може да се заклучи дека и двата прстапи се доста успешни за овој проблем. Доколку немаме многу сообраќај тогаш архитектурата со сервер ќе ни биде покорисна, но доколку имаме голем сообраќај, т.е. многу истовремени повици до API-то тогаш serverless е дефинитивно подобра опција. Сепак, користењето на едната или другата опција зависи од комплексноста на апликацијата и од тоа каков проблем треба да решава.

Референци

[1] Fundamentals of Azure Second Edition by **Michael Collier & Robin Shahan**

[2] Azure Documentation

[3] Microsoft Learn