Software Architecture: Assignment 3

AWS Deploy and Stress Testing, group 6

Subject:

Software Architecture

Teacher:

José Luis Assadi

Students:

Manuel Bentjerodt

Nicolás Gebauer

Jorge Plaza de los Reyes

Delivery date:

August 30

1. USED ARCHITECTURES

For this activity, a Flask application was integrated on three different architectures, availables thanks to AWS. The first consisted in integrating the application to an EC2 instance; the second was to split the database with the application, leaving the main app in an EC2 instance and the DB in an RDS service; and the third was the same way as the second one, with the addition of Varnish as Reverse-Proxy, integrated in another EC2 instance.

1.1. EC2

AWS EC2 is a cloud computing service, that is, it allows the use of virtual machines in the cloud for the management and deployment of user services and applications. These virtual machines are represented as instances. The instances created allow the selection of dedicated resources for each one of them, such as CPU usage, memory, capacity, network usage, load balancing, etc. It also has security options and both private and public connectivity, allowing the use for development environments. Another point to mention is the monitoring window per instance, which provides more detailed resource usage in terms of time, allowing a better optimization of the instantiated service.

1.2 RDS

AWS RDS is a relational database management service by Amazon, that simplifies implementation, scalability, operability and setup tasks. It also allows the support of several database engines, such as: MySQL, PostgreSQL, Oracle, etc. The instance created with RDS, connects to the one created with EC2 through the connectivity of the same VPC. Once linked, the database is separated from the main server and acts under the RDS instance, which has the ability to perform the CRUD operations sent by EC2. In addition to the relief of resource usage for the EC2 instance, other benefits of using AWS RDS include: storage management, backups, scalability + load balancing and security.

1.3 VARNISH REVERSE PROXY

Varnish is a Reverse Proxy open-source service, meaning that it works by serving cached content to users, and so, reducing the load on the backend and DB servers. This service acts by redirecting requests to the main server, i.e. all requests sent are redirected to the reverse proxy endpoint and then passed to the main application or directly to the database. That is where the usefulness of this service lies, in its use of the cache for the responses of the requests sent, which if found in the cache, are automatically responded to and sent to the database, without the need for processing of the main server, saving CPU and memory usage. Thus, with the most frequent and repeated requests, large amounts of resources are saved, which in large numbers can mean the downtime of the service. However, as a consideration to mention, dynamic content is not processed by the reverse proxy and is sent directly to the main application, also, it is the capacity of the service cache and the amount of requests sent in short periods of time, which can cause bottlenecks and call rejections.

As described above Varnish acts in the same way with our application in EC2 and its database in RDS, where the connection between the three is done through the endpoints of each one.

2. RESULTS

For load testing of the Flask application, it was deployed three times according to the requested specifications. Each EC2 instance had a t2.micro configuration, which consists of 1 vCPU along with 1 GB of RAM, using the Amazon Linux 2023 image. Below are three tables, one for each deployment, presenting relevant testing information. The testing involved varying quantities of requests to the application over a 5-minute period.

2.1. EC2 (Deploy 1)

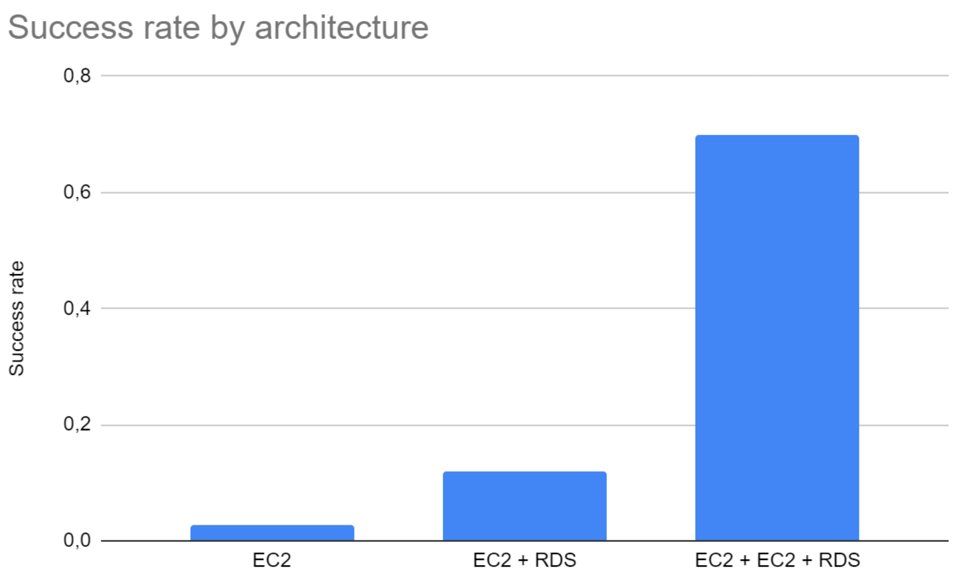
ENDPOINT	/		/authors		/books	
Request / 5m	CPU %	MEMORY %	CPU %	MEMORY	CPU %	MEMORY %
1	0,3	6,9	28,6	6,9	30,6	5,9
10	0,3	6,9	99,7	6,9	54	9,2
100	0,7	6,9	91	13,5	92	12,5
1000	1	6,9	93	33,4	100	42,4
5000	1	6,9	100	35	100	42
10000	1	6,9	100	35	100	42

2.2. EC2 + RDS (Deploy 2)

ENDPOINT	/		/authors		/books	
Request / 5m	CPU %	MEMORY %	CPU %	MEMORY %	CPU %	MEMORY %
1	0,3	8	41,7	8	47,5	8
10	0,3	8	41,2	17,1	46	19
100	0,7	8	98	15,4	99	22
1000	1	8	99	15,5	99	24
5000	3	8	100	15	100	26
10000	3	8	100	16	100	24

2.3 EC2(Proxy) + EC2 + RDS

ENDPOINT	/		/authors		/books	
Request / 5m	CPU %	MEMORY %	CPU %	MEMORY %	CPU %	MEMORY %
1	0,3	8	38	8	40	8
10	0,3	8	42	8	45	8
100	0,3	8	40,5	8	48,7	8
1000	1,3	8	40,9	8	46,5	8
5000	1	8	41	8	46,8	8
10000	1,3	8	40,9	8	44,5	8



3. ANALYSIS

3.1 EC2 (DEPLOY 1)

For the “/” endpoint, both CPU and memory usage remain relatively low, even with 10,000 requests over 5 minutes.
For the /authors endpoint, there is a noticeable spike in CPU utilization as requests increase, hitting 93% with 1,000 requests. However, memory remains stable.
The “/books” endpoint displays behavior akin to “/authors”, with CPU peaking at 100% at 1,000 requests, and maintaining a high percentage even with 10,000 requests.

3.2. EC2 + RDS (Deploy 2)

CPU and memory usage on the “/” endpoint consistently stay low.
For the “/authors” endpoint, the CPU maxes out (100%) with just 100 requests, suggesting this endpoint has significant resource consumption when accessing the database.
Similarly, the "/books” endpoint hits 100% CPU usage at 100 requests, pointing to a similar pattern as “/authors”.

3.3 EC2(Proxy) + EC2 + RDS

Again, the load on the / endpoint remains low, indicating that this endpoint is efficient and possibly lacks costly operations.
Both the /authors and /books endpoints show similar and stable CPU and memory utilization, even as requests ramp up. This suggests that the proxy setup might be distributing the load effectively or that this setup just handles the load more efficiently overall.

4. DISCUSSIONS

Resource Utilization and Scalability: The results demonstrate varying degrees of CPU and memory utilization based on the deployment configuration and endpoints. CPU spikes, especially those observed in the /authors and /books endpoints, may indicate inefficiencies in how the application processes requests or retrieves data, especially from the RDS. Another hypothesis is that as the number of requests grows, certain queries or processes that were trivial with fewer requests become significant bottlenecks with a larger volume.

Reaching 100% CPU Utilization: Once CPU utilization hits 100%, the system has reached its processing capacity. When a server's CPU is maxed out, several issues arise:

> **Response Times**: The application will likely experience dramatically increased response times. This leads to a poor user experience, as users may face significant delays or even time-outs.
>
> **Potential Crashes**: Continual operation at 100% CPU can cause the application to become unresponsive or even crash. This is a severe scenario, especially in production environments where uptime and reliability are critical.
>
> **Queuing**: New incoming requests might get queued up, waiting for processing power to become available. This queue can grow, and if it becomes too extensive, new requests might be dropped or rejected.
>
> **Overheating & Hardware Stress**: Prolonged high CPU usage can lead to overheating and potentially reduce the hardware's lifespan.

Memory Stability vs. CPU Spikes: It's noteworthy that while CPU usage sees significant spikes, memory usage remains relatively stable across most configurations. This may suggest that the application's resource strain is more computationally intensive rather than data-intensive. Processes such as data parsing, computation, or non-optimized queries might be the culprits.

Proxy's Impact: The configuration involving a proxy shows more even distribution of resources, which could be due to multiple factors:

Load Distribution: If the proxy distributes incoming requests to multiple backend servers, it can effectively balance the load and prevent any single server from being overwhelmed.

Caching: If the proxy caches frequent responses, it reduces the need to process those requests on the main application server or access the database repeatedly.

5. CONCLUSIONS

The “/” endpoint is efficient across all deployment configurations and handles large loads without significant resource strains.

The “/authors” and “/books” endpoints are more resource-intensive, especially when interfacing directly with a database on RDS. These endpoints might benefit from optimizations, whether in the application or in the database query itself.

The setup involving a proxy seems to handle the load better than the other two configurations. However, more data would be helpful to understand why. For instance, is the proxy distributing the load across multiple instances? Is it caching responses?

Overall, a deeper dive into the “/authors” and “/books” endpoints would be recommended to pinpoint the costly operations and consider optimizations either at the application level or the database level. Additionally, caching techniques or load balancing might be worth considering to handle high request volumes without exhausting resources.