SUTD 2022 Term 7 50.046 Homework 2

Hand-out: 17 Feb

Hand-in: 24 Feb (Thursday 23:59)

Name: James Raphael Tiovalen Student ID: 1004555

Question 1. [10pt] Please compare a serverless architecture with a server-based architecture.

1) [2pt] What is a serverless architecture in the cloud? How is it different from a server-based architecture?

Answer: A serverless architecture is a way to build and run applications and services without having to manage infrastructure. The applications and services still run on servers, but the server management is done by the cloud provider/vendor (from provisioning to scaling and maintenance). It is different from a server-based architecture since application developers can do their work without having to worry about servers at all. Instead, developers can purchase backend and server services on a flexible "pay-as-you-go" basis.

2) [6pt] Please list 3 strength and 3 weakness points of a serverless architecture.

Answer: The strengths of a serverless architecture include:

- Decreased time to market: To deploy a working version of an application, a serverless infrastructure eliminates the requirement to upload code to servers or perform any backend configuration. Developers may upload code and deliver a new product in a matter of minutes. Because the application is not a single monolithic stack but rather a collection of functions provided by the vendor, they can upload code all at once or one function at a time. This also allows developers to update, modify, correct, or add new features to a program quickly. Developers do not need to make changes to the entire application; instead, they can update one function at a time.
- Enhanced scalability: As the user base or consumption develops, applications built on a serverless infrastructure will scale automatically. If several instances of a function are required, the vendor's servers will launch, execute, and stop them as needed, often using containers. As a result, a serverless application will be capable of handling a large number of requests just as well as a single request from a single user. On the other hand, a program running on a set quantity of server capacity can be overwhelmed by a rapid spike in demand.
- Lower cost: Developers are only billed for the resources they use. The code only runs when the serverless application requires backend functionalities, and it automatically

scales up as needed. Provisioning is dynamic, precise, and instantaneous. In a traditional 'server-full' architecture, on the other hand, developers must forecast how much server capacity they will need in advance and then acquire that capacity, whether they end up using it or not.

The weakness points of a serverless architecture include:

- Security issues: It may not be able to completely assess suppliers' security when they operate the entire backend, which can be an issue for applications that handle personal or sensitive data. Serverless services will frequently run code from multiple clients on a single server at any given time because companies are not allotted their own distinct physical servers. Such multitenancy has the potential to degrade application performance and expose data if multi-tenant servers are not configured properly.
- Vendor lock-in: Allowing a vendor to handle all of an application's backend services increases the application's dependency on that provider. Setting up a serverless architecture with a single vendor can make it harder to swap vendors later, especially when each provider has slightly distinct capabilities and workflows.
- More difficult debugging process: It's tough to simulate the serverless environment to see how code will behave after it's deployed. Because developers have little access into backend processes and the application is separated up into distinct, smaller functions, debugging is more difficult.

3) [2pt] Find one example for the serverless and non-serverless option respectively from AWS storage services.

Answer: One possible example for the serverless AWS data storage service listed here: https://docs.aws.amazon.com/whitepapers/latest/serverless-multi-tier-architectures-api-gateway-lambda/serverless-data-storage-options.html is Amazon Simple Storage Service (Amazon S3), while one possible example for the non-serverless option of the AWS data storage service listed here: https://docs.aws.amazon.com/whitepapers/latest/serverless-multi-tier-architectures-api-gateway-lambda/non-serverless-data-storage-options.html is Amazon Relational Database Service (Amazon RDS).

Question 2. [4pt] We learned that a relational database must follow the ACID principle (Atomicity, Consistency, Isolation, Durability). In NoSQL, another principle BASE (Basically Available, Soft state, Eventual consistency) is adopted.

1) [2pt] What are the differences between ACID and BASE principles?

Answer: The ACID model provides a strongly consistent system, while the BASE model provides high availability. The ACID consistency model is usually used by RDBMS, while the BASE consistency model is usually used by NoSQL database systems that are willing to compromise strong consistency for the sake of high availability even in the presence of network partitions, which could realistically and frequently occur in real-life production systems, as dictated by the CAP theorem.

2) [2pt] And use one database example to illustrate how the BASE principle is applied.
Answer: One database example would be Amazon DynamoDB. Amazon DynamoDB provides eventual consistency, which means that the latest/recent update/write might not always be reflected in subsequent reads. DynamoDB uses single-leader replication, whereby clients would send their write requests to the leader node, which would then determine the order in which writes would be processed, and the follower nodes apply the leader's writes in the same order. Thus, clients that subsequently read from the replicas/follower nodes might not have transparently seen the latest update yet as it is still propagated through the network from the leader node to the follower nodes.

Question 3. [6pt] AWS provides multiple EBS volume types for different workloads. Please help find the proper EBS volume type for the following use cases. Explain your choice.

1) [2pt] Amazon EBS volume type can be expanded to 64,000 IOPS and 1,000 MB/s throughputs and offers a tiered pricing structure.

Answer: The appropriate EBS volume type is **io2**, under the Provisioned IOPS SSD category. This is because as stated in the corresponding link, for both io1 and io2, the maximum IOPS per volume is 64,000 and the maximum throughput per volume is 1,000 MiB/s. However, as stated in this page: https://aws.amazon.com/ebs/pricing/, only io2 provides a tiered pricing structure.

2) [2pt] Amazon EBS volume type provides a baseline of 3,000 IOPS and 125 MB/s throughputs and can scale volume size and performance independently.

Answer: The appropriate EBS volume type is **gp3**, under the General Purpose SSD category. This is because in the link, it is stated that the gp3 volume type can deliver a consistent baseline of 3,000 IOPS and 125 MiB/s, and it can scale volume size and performance independently, as long as the ratio of provisioned IOPS to provisioned volume size is not more than 500 IOPS per GiB. Meanwhile, for gp2, the baseline performance scales linearly at 3 IOPS per GiB of volume size. Thus, gp2 is not able to scale volume size and performance independently.

3) [2pt] Amazon EBS volume types support attachment to multiple Amazon EC2 instances at the same time. (There can be multiple answers.)

Answer: The appropriate EBS volume type is either **io1, io2, or io2 Block Express**, under the Provisioned IOPS SSD category. This is because, as stated here: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-volumes-multi.html, Amazon EBS Multi-Attach is available for io1 volumes in the Regions: us-east-1, us-west-2, eu-west-1, and ap-northeast-2, and it is available in all Regions that support io2 and io2 Block Express volume types.

-- END --