Name: Smit Mahesh Panchal
Asurite: spanch19

SER-516 Assignment No. 4

Activity 2: Critical Inquiry

Q: What do the authors mean by “infrastructure is fluid”? In what ways can it be fluid?

Ans: The authors meaning of the characteristic that infrastructure is fluid meaning that the nature of the cloud applications is very dynamic in nature. The environment facilities of the cloud native applications are bound to change and the application must adapt to the changing and dynamic environment. The infrastructure is very prone to grow, shrink or migrate over time or based on the application needs. There are various ways that it can be fluid, some of them are as follows -

1. Containerization of the applications components or modules so that can be scalable and respond to the demand and not rely heavily on the hardware aspects.
2. These computing resources can be automatically added or removed based on the demand. So on demand computing power optimization can make the architecture more fluid in nature.
3. Migration of workloads to balance the load or for fault recovery.
4. Cloud platforms do a really good job of abstracting the physical specifications which the developers do not really care about. Thus these type of platform abstractions can also be made or leveraged in order to make infrastructure more fluid.

Q: What do the authors mean by “failure is constant”? In what ways is failure constant?

Ans: The authors meaning of failure is constant points to the fact that when working on a global scale failure is sure to happen in terms of cloud native application. The author also elaborates on this fact saying that applications which were built to function or be used on a single PC, mainframe or supercomputer, basically assuming that the underlying OS and hardware is rock solid and cannot change, So when these types of applications are ported into a cloud environment and run in a global environment the probability of them failing is very close to 100%. So essentially it says that failure is a norm and the capability of the application to recover from failures without any external intervention should be one of the core principles. The ways in which the failure is constant are as follows -

1. Network instability can be a frequent cause of failure like packet timeout or routing issues which might be unpredictable at a global scale.
2. Hiccups in datacenter like temporary hardware or VM unavailability can be causes of failure of such type of applications.
3. Software bugs causing containers crashing or causing memory leaks which causes failure in the builds and disrupts the continuous integration pipeline.

The authors provide a very brief application walkthrough based on Figure 1. Based on this walkthrough, answer the following:

1. Which aspects of Figure 1 are true microservices and which are appliances as we discussed when doing microservices?

Ans: the aspects of the Figure 1 which represent true microservices are M1-M6, A1-A5 and D1-D3 which also includes D’1. These represent microservices handling Data monitoring, Analysis and database management for the application.These all are considered true microservice as they depict the characteristics of being small, independent and loosely coupled in nature and they all have one specific job to perform when looking from a larger application perspective.

In terms of appliances, AWS simple queueing service, CosmosDB of Azure and DynamoDB of AWS are considered as appliances in figure 1. These applications are themselves considered as cloud native applications.

2. Identify the challenges in implementing an observability solution for this hypothetical application. You do not have to design a whole solution architecture, I’m asking you to consider what will be difficult to do when considering the 3 pillars of observability (so yes, structure your answer here in terms of the 3 pillars).

Ans: The challenges in implementing an observability solution for this situation are as follows -

1. Collecting and centralization of **logs** from the different microservice could be difficult in this architecture. Since there are multiple services giving output in different format and storage might also be in difficult as the developers might decide to make one database service include Azure based database and the others being AWS based so collection and standardization of logs across the whole application could be pretty difficult to implement. Thus in summary log integration and tracing microservices will be difficult.
2. Another difficulty could be **Metric** wise as we need to monitor the effectiveness or for health and performance of the system as a whole. Also how can we maintain a consistent stream system of queries and feedbacks for those so that we can measure the effectiveness of using the Azure Based database as to AWS based system so we can migrate to it if it is better. So basically what and how the metrics should be constructed and implemented or watched to determine the effectiveness of the current system with all these microservices aggregated.
3. Since all the aggregator services work in an asynchronous fashion so it can cause issues or difficulty in **Tracing** requests along all the different microservices. This is a challenge as not able to trace requests and responses across the microservices will also cause problems with determining latency which can be used as a **metric** to calculate the efficiency of the Azure database. Tracing here would be difficult as the M and A services are accessing the AWS queue system asynchronous so end to end tracing of what is going on will be difficult at a particular instance of time.