**Activity 2**

kgulbarg

**Q 0.1)**

“Infrastructure is fluid" means that the cloud environment is always changing. Resources such as virtual machines, containers, storage, and even network settings can be added, removed, resized, or moved automatically. This happens based on demand, performance, or failure. For example, an app might scale up during high traffic or move containers to a different server if one fails. Because of this, applications should not expect the infrastructure to stay the same. They need to be designed to keep working even when the environment, i.e., the infrastructure, around them changes.

**Q 0.2)**

"Failure is constant" means that in big cloud systems, something could always be going wrong somewhere. There are numerous services, machines, and networks working together that it’s very common for some parts to crash or slow down. This might be a suspended connection, a service timing out, or a disk failing. Even if each part is reliable, the size of the system means failures are normal. That’s why cloud-native applications are built to be resilient — they use tools such as retries, backups, and failover systems so they can keep working even when something breaks.

**Q 1)**

In Figure 1, the parts labeled M1 through M6 and A1 through A5 are microservices because these are small, focused services created and managed by the development team. Each one does a specific job, such as collecting data or analyzing it. They are separate from each other and can be changed or scaled without affecting the rest of the system.

The other parts, such as AWS Simple Queue Service (SQS), DynamoDB, and CosmosDB, are appliances because these are cloud tools that the team uses but does not control. They are provided and managed by the cloud provider. The team can use them for storing data or sending messages between services, but they cannot change how they work. They are reliable and useful, but they are not built or owned by the application team.

**Q 2)**

Logs: One challenge is collecting logs from all the different microservices and cloud tools. Each service might use a different logging system or format, which makes it hard to collate them. Also, cloud services such as SQS or DynamoDB may only give limited logging or send logs to different places. It becomes difficult to track a full request or event using logs when everything is scattered or inconsistent.

Metrics: Getting clear and useful metrics can also be hard due to the lack of standard formats across the system. Microservices can send custom measures, but managed cloud services mostly only give basic ones, and they may use different formats. It can be tricky to interpret all this data, especially when services are added or removed due to scaling. A sudden change in traffic or the number of services can make the metrics confusing or misleading if not handled properly.

Traces: Tracing a request from start to finish is probably the hardest part. Distributed tracing depends on every service being set up correctly to pass along trace data. Microservices might not all be instrumented the same way. Without something such as a shared tracing system, it’s very hard to see where a problem has occurred or how long each step took. This makes debugging and performance tuning much more difficult.

For all three of these pillars, the main concern comes down to lack of agreement on the depth and format of the observability outputs. However, this is a centralization approach which is against the distributed spirit of cloud.