**Assumptions made:**

I have assumed that every single component in the architecture operates asynchronously.

I have also assumed the capacity of the web applications developed by the company can be scaled horizontally easily to match the capacities of the other modules.

I have assumed that the Business Intelligence analytics are not overly complex such that Amazon Athena would suffice.

I have also assumed that image compression algorithms are done as part of the company’s software code. Any retrieval of the image would be routed to the company’s code to undo the lossy compression.

**Key Principles**

1. Separation of Concerns
2. Highly Cohesive, Loosely Coupled
3. Event-driven
4. Distributed system
5. ACID (Atomicity, Consistency, Isolation, and Durability)

**To address the key principles:**

1. Every module in the architecture diagram has a single responsibility. The client interface is responsible for collecting images uploaded by customers and returning processed images to them. It is not involved in any other process. The Kafka stream is solely responsible for ensuring the continuity of the stream and acting as a load balancer. The Amazon S3 Temporary Storage is different from the Amazon S3 Archived Storage because they have different responsibilities. The Temporary Storage is like a holding point for the processed images while the Archived Storage is the place where processed and analyzed images go. This distinction of responsibilities make identifying problems easier.
2. The Amazon ecosystem is highly compatible with each other from Apache Kafka to the S3 systems and the Lake Formation and Athena. This cohesiveness ensures that the building of the architecture would be smoother, and they would work better together with less problems. They are also loosely coupled because they are all separate modules. This lack of dependency means that the architecture is less fragile and that modules can be easily replaced. The Image Processing software code can be replaced and optimized without fear of affecting the performance of the rest of the architecture.
3. The entire architecture has a clear top-down event-driven architecture. The client calls would result in a series of event in the Kafka stream that would streamline the image processing process before sending the images back to the client and to the database. After the image is sent to the database, it will be temporarily held at the temporary storage. Once green light is given by the lake formation, it will be routed to Amazon Athena as a batch before it is routed to the archived storage. Amazon Athena would then inform the company through the company interface. At the same time, it allows the company to create an event by modifying how the Amazon Lake and Athena should function from the other way around.
4. The entire architecture built in a distributed fashion which allows vertical and horizontal scalability. Even if each single module like the AWS storage or the Kafka stream was to be improved, the architecture would still stand. Even if we were to increase the number of servers to process images (once the business grows in size), the architecture would still be able to work. Even if any of the server collapses, Kafka stream would still be able to balance the load to other servers, ensuring that our services remain reliable and available.
5. Because of how the image is passed around in the architecture, the transaction will be cancelled or rejected if any of the step fails. This ensures lack of ambiguity in the state of the database. Also, the images in the database and the one returned to the client would also be consistent since no changes would be made to the images in the database (other than analysis). Amazon S3 databases are also extremely durable, ensuring that data is properly preserved even in power outages, etc.