**Question 1:** What is a Data Lake? Explain its benefits and how it might benefit a client.

A data lake is a highly scalable and flexible repository for storing raw unstructured or semi-structured data before any processing or analytics have been performed. Data in a data lake is often recently ingested, and the data lake often serves as a “staging area” for data before clear, well-defined use cases and processing pipelines are fleshed out and implemented. This contrasts with a “data warehouse”, which is a scalable repository for storing processed, structured data that is ready to serve immediate business needs. Data lakes and data warehouses are *complimentary* technologies, and often data from a data lake is fed through a processing pipeline where a structure or schema is applied to the data, and then ultimately stored in a data warehouse for future querying.

Data lakes are useful when it becomes impractical to process data *as it is generated.* In today’s “data-driven economy”, there are often simply not enough human resources available to process the enormous volumes of data businesses are generating in real time. Data lakes provide an immediate staging area to hold data as it is generated, so potentially valuable data is not lost. This gives businesses and their data workers a “time buffer” within which to implement processing infrastructure to utilize this data and discover new, potentially unforeseen, use cases for the data. The alternative is to simply discard the data, which equates to lost business value and missed business opportunity.

**Question 2:** What are the pros and cons of serverless architectures?

Serverless computing services like AWS Lambda and Azure Functions are the latest innovation in “cloud native computing”. As with any technology, serverless computing architectures come with benefits and drawbacks. The foremost benefit is the agility and flexibility it gives to software developers and the applications they build. With a serverless computing architecture, one does not need to worry about or manage the underlying infrastructure a given piece of code runs on. This burden is left to the cloud provider, allowing businesses to utilize the “practically infinite” scalability of the provider. Functions within an application become fundamentally decoupled from one another and need not even be written in the same language, giving developers huge flexibility in how applications are written and designed. Finally, most “Function as a Service” serverless computing offerings provide broad integration with the ecosystem of other services available in typical cloud computing environments, allowing developers to focus on writing software without managing the cumbersome details of the infrastructure surrounding their application logic (databases, API routing, load balancing, etc.)

The primary benefit of serverless computing architectures (offloading management of the computing infrastructure) could also be viewed as its primary drawback. With a serverless computing architecture, you exchange *convenience* for *control.*  For certain applications, control of the underlying infrastructure may be a hard requirement, in which case serverless computing architectures simply aren’t suitable. Additionally, while on the surface serverless computing architectures seem to increase *simplicity,* they often end up increasing *complexity*. Serverless computing architectures are inherently distributed, and as the scope of a given application built with such an architecture grows the distributed nature of the application can make maintenance and debugging of an application complex and time consuming.

**Question 3:** Please provide a diagram of the ETL pipeline from Section 1.5 using serverless AWS services. Describe each component and its function within the pipeline.

See the docs/figures folder in this repo for a Visio diagram of the entire architecture, as well as PDFs of each of the individual pages in the Visio (for those without access to Visio). Totally happy to write up a description of everything, but that would certainly exceed two pages and I think it would be more efficient if I could discuss with the team verbally ☺.

**Question 4:** Describe the process of deploying infrastructure as code on AWS or Azure and why this is a good method of architecture delivery for clients.

At a *very* high level, the process looks something like the following:

1. Choose an appropriate IaC tool for your organization and use case, of which there are many (Terraform, CloudFormation, Azure ARM Templates, Ansible). I generally prefer tools like Terraform and Ansible because they are cloud agnostic and portable across cloud providers. Tools like CloudFormation and Azure ARM Templates suffer from “vendor lock in” to some extent.
2. Plan and design the desired state of your infrastructure. This part is arguably the most work because you need to answer all the hard questions. What problem are we trying to solve? What infrastructure components are most appropriate for solving the problem? How do these infrastructure components communicate with one another? How will we secure them? I find a good plan goes a long way and will certainly benefit you in the long run over jumping directly into writing some code to deploy “some stuff” in whatever tool you picked in step 1.
3. Declare/describe the state you carefully planned in step 2 in the syntax of whatever tool you choose. If you were thorough in step 2, this part should just be nailing down the syntax, writing clean, modular code, and working through any gotchas with whatever IaC tool you chose (and there are *always* gotchas).
4. Click the magic deploy button! When it doesn’t work the way you thought it would (rarely does on the first try), return to step 2 and repeat.

Using the Infrastructure as Code approach is a good method of architecture delivery because:

1. Infrastructure deployed using IaC tools is repeatable and consistent. Fewer human hands on the deployment process equates to fewer manual errors.
2. With the state of your infrastructure described in code, it is naturally version controllable. Version control admits an iterative, collaborative process to designing and refining your infrastructure.
3. The code describing your infrastructure serves a dual purpose as a source of truth and documentation for your infrastructure. No more mystery networks and shadow IT. Just look at the code!
4. Once the code is written and properly modularized, deployment of new infrastructure is faster and less error prone than following a manual process.
5. With the right IaC tools, your infrastructure deployment methodology becomes portable across IaaS platforms.