**Advantages**

1. **Cost-Effective**
   * With AWS Lambda, you only pay for the compute time you consume (by the millisecond), making it cost-effective for infrequent or unpredictable workloads.
2. **Automatic Scaling**
   * Lambda automatically scales with incoming requests, handling thousands of concurrent executions, which is ideal for applications with unpredictable traffic patterns.
3. **No Server Management**
   * Since Lambda is fully managed by AWS, there’s no need to provision, manage, or scale physical servers or virtual machines, reducing maintenance overhead.
4. **Integration with AWS Ecosystem**
   * Lambda seamlessly integrates with a variety of AWS services like S3, DynamoDB, Kinesis, SNS, and API Gateway, enabling flexible and powerful workflows.
5. **Event-Driven Architecture**
   * Lambda can be triggered by events from multiple sources, supporting real-time, event-driven applications without the need for polling or constant monitoring.
6. **Faster Deployment**
   * With the serverless model, deployment is quick, as you’re only deploying small pieces of code. This can improve development speed, especially for smaller, modular functions.
7. **Improved Resilience and Fault Tolerance**
   * AWS manages fault tolerance for Lambda functions, running instances across multiple Availability Zones by default, which can lead to better resilience.
8. **Environment-Friendly**
   * Lambda uses shared server resources, reducing the need for dedicated servers and potentially reducing energy consumption and carbon footprint.

**Disadvantages**

1. **Execution Time Limits**
   * Lambda functions have a maximum execution time of 15 minutes, which can be a constraint for long-running processes or batch jobs that need more time.
2. **Cold Start Latency**
   * Lambda functions can experience latency when they haven’t been used for a while (cold start), which may affect performance for latency-sensitive applications.
3. **Limited Control Over Execution Environment**
   * The runtime environment is managed by AWS, meaning you have limited control over the underlying infrastructure, including operating system and runtime updates.
4. **Restricted Storage**
   * Lambda functions have a temporary disk space limit of 512 MB, which can be limiting for applications that require more storage during execution.
5. **Resource Limits and Concurrency**
   * AWS imposes limits on function memory (up to 10 GB) and concurrent executions. Although limits can be raised, they could impact applications that require extensive computing resources.
6. **Vendor Lock-In**
   * Using Lambda can lead to vendor lock-in as code is often written to work specifically with AWS services. Migrating Lambda functions to another cloud provider may require significant rewrites.
7. **Debugging and Monitoring Complexity**
   * Monitoring, debugging, and logging can be more complex compared to traditional servers, as debugging distributed and event-driven systems require additional tools and practices.
8. **Limited Language and Dependency Support**
   * While Lambda supports multiple languages (e.g., Python, Node.js, Java, Go), some libraries or frameworks may not work as expected due to Lambda’s runtime constraints.
9. **Cost for High-Volume, Continuous Workloads**
   * For applications with high and consistent workloads, Lambda can be more expensive than traditional servers, as Lambda pricing is optimized for short and sporadic executions.