Domain 1

Network Connectivity Strategies

AWS Direct Connect vs. AWS VPN vs. Transit Gateway

1. **AWS Direct Connect**

    - **Purpose**: Provides a dedicated, private connection between on-premises data centers and AWS.
    - **Performance**: Offers stable, low-latency, and high-bandwidth connections, allowing consistent throughput.
    - **Security**: Private connectivity ensures data doesn't traverse the public Internet, enhancing security.
    - **Use Cases**: Ideal for data-intensive applications, hybrid cloud setups, and regulatory compliance.

2. **AWS VPN**

    - **Purpose**: Establishes a secure, encrypted connection over the public Internet between on-premises networks and AWS VPCs.
    - **Performance**: Latency can vary as it relies on Internet traffic; generally lower bandwidth than Direct Connect.
    - **Security**: Utilizes IPsec for encryption, ensuring secure communication over untrusted networks.
    - **Use Cases**: Suitable for smaller data transfers, temporary connections, and when budgets are tight.

3. **AWS Transit Gateway**

    - **Purpose**: Acts as a central hub to connect multiple VPCs and on-premises networks.
    - **Performance**: Optimizes routing and reduces the need for multiple connections, potentially enhancing network performance.
    - **Scalability**: Simplifies management by allowing multiple VPCs to connect through a single gateway; ideal for large, multi-region architectures.
    - **Use Cases**: Best for complex architectures with many VPCs, requiring centralized traffic management.

Summary

- Use **Direct Connect** for high-performance, stable connections.
- Choose **VPN** for secure, encrypted Internet connections, especially for smaller workloads.
- Implement **Transit Gateway** for efficient management of multiple VPCs and connectivity in expansive architectures.

Domain 2

Compute & Serverless Architectures

EC2 Auto Scaling vs. Lambda Provisioning

1. Scalability

- **EC2 Auto Scaling:**
  - Scales EC2 instances up or down based on predefined metrics (CPU, memory, etc.).
  - Requires time to spin up/down instances, making it less responsive to sudden spikes.
- **Lambda:**
  - Automatically scales instantly based on incoming requests.
  - Handles unpredictable traffic seamlessly, as it can manage thousands of concurrent executions.

2. Cost Management

- **EC2 Auto Scaling:**
  - Pay for running instances, regardless of usage.
  - Costs can accumulate during idle periods, depending on instance types.
- **Lambda:**
  - Pay only for execution time and requests.
  - More cost-effective for sporadic workloads due to a consumption-based pricing model.

3. Management

- **EC2 Auto Scaling:**
  - Requires more management overhead (instance health checks, AMI maintenance).
  - Infrastructure management such as OS updates, scaling policies, etc.
- **Lambda:**
  - Fully managed service; no infrastructure management required.
  - Simplifies deployments, updates, and maintenance.

4. Performance

- **EC2 Auto Scaling:**
  - Better suited for long-running applications that need consistent performance.
  - Initialization overhead can lead to latency during scale-up events.

- **Lambda:**
    - Optimal for short-lived tasks; may experience cold starts, impacting latency.
    - Excellent for event-driven architectures with variable workloads.

5. Use Cases

- **EC2 Auto Scaling:**
    - Best for predictable scaling of applications like web servers, databases, and services with steady workloads.
- **Lambda:**
    - Ideal for microservices, APIs, real-time data processing, and event-driven applications.

Conclusion

For unpredictable traffic patterns, **Lambda** often provides better scalability and cost efficiency, while **EC2 Auto Scaling** may be preferable for stable, long-running workloads. The choice depends on specific application needs and usage patterns.
