SrNo	Questions
1	Build a serverless architecture using AWS Lambda and API Gateway to allow users to upload images, process them (e.g., resize or apply filters), and then retrieve the processed images through the API Gateway. Provide step-by-step instructions for implementing this system.
2	Design a serverless email subscription service that captures user email addresses through an API Gateway, triggers a Lambda function to store them in an Amazon DynamoDB table, and sends a confirmation email using Amazon Simple Email Service (SES). Test the system's functionality using Postman.
3	Design a serverless data aggregation and reporting system using AWS Lambda and API Gateway. Users should be able to submit data via the API Gateway, and a Lambda function should aggregate and store the data in Amazon Redshift or a similar data store. Users can then retrieve reports via the API Gateway.
4	Build a serverless URL shortening service using AWS Lambda and API Gateway. When users submit a long URL via the API Gateway, a Lambda function should generate a short URL and redirect users to the original URL when they access the short URL. Test the functionality using Postman.
5	You are tasked with designing a VPC in AWS, and you've chosen the CIDR block 172.31.0.0/16 for your private subnet. Calculate and provide the total number of available IP addresses in this CIDR block for your private subnet.
6	In your AWS VPC, you've allocated the CIDR block 192.168.0.0/20. Calculate the number of available IP addresses for both the public and private subnets if you plan to divide this CIDR into two equal-sized subnets.
7	You are designing a network infrastructure for a company that requires three subnets for different purposes, each with unique CIDR blocks. Calculate and provide the total number of available IP addresses in each of the following CIDR blocks: 10.0.0.0/24, 192.168.1.0/28, and 172.16.0.0/23.
8	You've been allocated a block of IP addresses in the format 203.0.113.0/24. Calculate and provide the total number of available IP addresses in this block. If you need to expand the range to accommodate more devices, what is the new CIDR block if you add an additional /25 subnet?
9	You have multiple smaller CIDR blocks in the format 192.168.1.0/26, 192.168.1.64/26, and 192.168.1.128/26. Calculate and provide the total number of available IP addresses when these CIDR blocks are combined into a supernet.
10	Suppose you've set up a budget named "Monthly AWS Costs" with a budgeted amount of $1,000. You've chosen to receive email notifications when 80% of the budget is consumed. If your AWS costs exceed $800 in a month, you will receive an email notification to alert you of the increased spending.
11	You've implemented the order processing system, but your e-commerce website has experienced a sudden surge in orders. Explain how you can configure the components (Lambda, SQS, DynamoDB) to handle increased workload and ensure the system remains scalable and reliable.
12	With above questions - Discuss error handling and retry strategies for the order processing system. How can you ensure that failed processing attempts are retried and that error notifications are sent to a designated team for further investigation?
13	You're responsible for managing the budget for the order processing system. Describe how you can set up cost monitoring, budget alerts, and usage reports to keep track of the expenses associated with Lambda, SQS, and DynamoDB. What are some best practices for staying within budget while maintaining system performance?
14	Describe how to set up a DynamoDB trigger that invokes a Lambda function when changes occur in a specific DynamoDB table. Explain the different trigger types, such as stream-based and batch-based, and provide an example use case for each.
15	Write an AWS Lambda function in Python to process incoming JSON files and store them in an Amazon DynamoDB table. Provide a sample JSON file for testing.
16	Set up an S3 bucket for storing incoming JSON files. Configure the bucket with appropriate access policies and versioning.Configure an S3 event to trigger the Lambda function whenever a new JSON file is uploaded. Create an SNS topic to notify stakeholders when file processing is complete. Set up subscriptions for relevant email addresses.

17	You are tasked with designing a text processing system for your company. The system needs to process incoming text data, perform analysis, and store the results in a scalable and efficient manner. To achieve this, you will leverage various AWS services. Requirements: 1. AWS Lambda Functions: Create AWS Lambda functions for text processing and analysis. These functions should take incoming text data, perform custom processing or analysis, and store the results. 2. Amazon S3: Set up an Amazon S3 bucket where incoming text files are stored. Configure event triggers to invoke the Lambda functions whenever new files are uploaded. 3. Amazon SQS: Implement an SQS queue to manage text processing requests, ensuring reliable processing and scaling when the volume of text data increases. 4. Amazon SNS: Establish SNS topics to notify stakeholders when text processing is complete, delivering notifications to relevant email addresses or other endpoints. 5. Amazon DynamoDB: Create a DynamoDB table to store metadata and analysis results from processed text data, allowing for efficient retrieval and query operations. 6. Security: Implement proper security measures, including IAM roles and permissions, to safeguard data and processing resources. Ensure that access to resources is restricted appropriately. 7. Monitoring and Logging: Set up AWS CloudWatch for monitoring and logging to maintain visibility into system operations. Configure alarms and log retention policies for effective monitoring.
18	You are tasked with designing a secure and efficient system for processing sensitive JSON files in a company. The company receives a constant stream of JSON files that contain sensitive information. The files must be processed, stored securely, and analyzed. The following requirements have been defined: 1. AWS Lambda: Develop AWS Lambda functions for JSON processing and analysis. These functions will securely store the JSON files in an Amazon DynamoDB table with encryption enabled. 2. Amazon S3: Set up an S3 bucket with versioning enabled where incoming JSON files are securely stored. Configure event triggers to invoke Lambda functions whenever new files are uploaded. 3. Amazon SQS: Implement an SQS queue to manage file processing requests, ensuring reliable processing and scaling when the volume of files increases. Ensure that the queue is encrypted and access is restricted. 4. Amazon SNS: Establish SNS topics to notify stakeholders when file processing is complete, delivering notifications to relevant email addresses. Ensure that SNS topics are encrypted in transit. 5. Amazon DynamoDB: Create a DynamoDB table to store metadata and analysis results from processed JSON files, allowing for efficient retrieval and query. Encrypt the DynamoDB table at rest. 6. Security: Implement proper security measures, including IAM roles and permissions, to safeguard data and processing resources. Enable AWS Key Management Service (KMS) encryption for Lambda, S3, SQS, SNS, and DynamoDB. 7. Monitoring and Logging: Set up CloudWatch for monitoring and logging to maintain visibility into system operations. Enable CloudTrail for auditing API actions. Create CloudWatch Alarms for important metrics.

19	You are tasked with deploying a scalable and fault-tolerant web application for a media streaming company. The application includes both a back-end component. For this scenario, you will focus on using AWS Elastic Beanstalk to achieve the company's requirements: 1. Environment Configuration: Create an Elastic Beanstalk environment suitable for a Python and Flask-based backend application. Ensure the environment includes a web server. 2. High Availability: Set up the environment to be highly available by spanning it across multiple AWS Availability Zones. Describe the process and configurations needed for achieving fault tolerance. 3. Security and Scalability: Implement security best practices by configuring security groups and custom VPC (not in a default VPC). Additionally, set up autoscaling policies for the environment to handle traffic spikes effectively. 4. Monitoring and Metrics: Configure AWS CloudWatch to monitor the application's performance and set up alarms for critical metrics. Provide insights into which metrics are essential and the threshold values for the alarms. 5. Deployment Strategy: Define a manual deployment strategy, perform two deployments - in the first, deploy a basic Python Flask app, and in the second, make some changes redeploy to demonstrate deployment strategy.
20	You are responsible for deploying a scalable and fault-tolerant web application for a media streaming company. The application includes both a back-end component based on Python and Flask. Your task is to design and implement an AWS Elastic Beanstalk environment that meets the company's requirements: 1. Environment Configuration: Create an Elastic Beanstalk environment suitable for a Python and Flask-based backend application. Ensure the environment includes a web server. Explain the necessary configurations for the environment. 2. High Availability: Set up the environment to be highly available by spanning it across multiple AWS Availability Zones. Describe the process and configurations needed for achieving fault tolerance. What considerations should be taken into account when deploying the application across AZs? 3. Security and Scalability: Implement security best practices by configuring security groups and a custom VPC (not in a default VPC). Additionally, set up autoscaling policies for the environment to handle traffic spikes effectively. Describe the security group rules and autoscaling configurations. 4. Monitoring and Metrics: Configure AWS CloudWatch to monitor the application's performance and set up alarms for critical metrics. Provide insights into which metrics are essential and the threshold values for the alarms. How would you set up custom CloudWatch alarms for your Flask application? 5. Deployment Strategy: Define a manual deployment strategy and perform two deployments: - In the first deployment, deploy a basic Python Flask app to Elastic Beanstalk. - In the second deployment, make some changes to the Flask app and demonstrate the deployment strategy. Explain the steps you take during the deployments.
21	You're tasked with setting up a scalable and secure Flask web application on AWS. The application will serve as an image uploading platform, allowing users to upload images, apply filters, and share them. Requirements: 1. Application Development: You have a Flask application codebase ready, including image uploading and filtering features. 2. Auto-Scaling: Implement an auto-scaling solution to handle varying user loads effectively. Configure scaling policies and triggers to adjust the number of application instances dynamically based on traffic fluctuations. 3. High Availability: Deploy the Flask application across multiple Availability Zones (AZs) to provide redundancy and high availability, minimizing downtime. 4. Security Measures: Implement security best practices, including securing communication over HTTPS, access controls, and safeguarding user-uploaded images. Ensure that application instances have the latest security patches. 5. Monitoring and Alerts: Set up comprehensive monitoring using AWS CloudWatch. Create meaningful alarms based on key performance metrics, such as response times, error rates, and resource utilization. Configure alerts to proactively respond to issues. 6. Amazon S3 Integration: Utilize Amazon S3 for storing user-uploaded images. Implement functionality in your Flask application to allow users to upload images to an S3 bucket and apply filters.

22	Design and implement a robust and scalable infrastructure for a Flask application that accepts JSON files, processes them, and displays structured output. Ensure high availability, security, monitoring, and logging. Use Amazon S3 for storage. Steps: 1. Flask Application Development: - Develop a Python Flask application that accepts JSON files, processes them, and displays the structured output. - Test the Flask application locally to ensure it functions as expected. 2. Create an S3 Bucket: - Log in to the AWS Management Console. - Navigate to Amazon S3 and create a new S3 bucket to store incoming JSON files. 3. Set Up an Elastic Load Balancer (ELB): - Create a new Elastic Load Balancer (ELB) to distribute incoming traffic among multiple instances. 4. Launch Configuration and Auto Scaling Group: - Create a launch configuration specifying the Flask application and other configurations. - Set up an Auto Scaling Group to launch instances based on the launch configuration. 5. Configure Autoscaling Policies: - Create scaling policies for the Auto Scaling Group to dynamically adjust the number of instances based on traffic fluctuations. 6. High Availability: - Configure the Auto Scaling Group to span across multiple Availability Zones (AZs) for redundancy and high availability. 7. Security Measures: - Implement security groups and Network Access Control Lists (NACLs) to control inbound and outbound traffic. - Ensure instances have the latest security patches using AWS Systems Manager. 8. Monitoring and Alerts: - Set up CloudWatch alarms to monitor key performance metrics (CPU utilization, request latency, etc.). - Configure alarms to trigger notifications when predefined thresholds are breached. 9. Logging and Tagging: - Implement CloudWatch Logs to monitor application logs for troubleshooting. - Apply resource tags to manage and organize AWS resources efficiently. 10. Testing and Verification: - Upload sample JSON files to the S3 bucket and ensure the Flask application processes them correctly. - Generate traffic to the application to trigger autoscaling and verify its effectiveness.
23	You are tasked with deploying a Python Flask web application that offers a service for image processing and storage. The application needs to be scalable, secure, and well-monitored. You must set up the infrastructure to achieve this. Tasks: 1. Flask Application Deployment: Deploy a basic Python Flask application on AWS. Ensure it's accessible over HTTP. You can use a sample Flask app for this purpose. 2. Autoscaling Configuration: Set up autoscaling for the Flask application. Define a scaling policy that adds or removes instances based on CPU utilization. Test the autoscaling by simulating traffic load on the application. 3. High Availability Setup: Deploy the Flask application across two AWS Availability Zones (AZs) to ensure high availability. Configure a load balancer to distribute incoming traffic evenly between instances in different AZs. 4. Security Measures: Implement security best practices. Ensure that communication with the Flask application is secure over HTTPS. Implement access controls to restrict unauthorized access. Create a security group to control incoming and outgoing traffic to the instances. 5. Monitoring and Alerts: Configure CloudWatch to monitor the Flask application. Set up alarms for key performance metrics such as CPU utilization, memory usage, and request latency. Define actions to be taken when alarms are triggered. 6. Logging and Storage: Set up logging for the Flask application using CloudWatch Logs. Ensure that logs are stored securely and can be easily retrieved for troubleshooting. Implement Amazon S3 for storing processed images from the application. 7. Testing and Cleanup: Test the entire setup by simulating different traffic loads, monitoring the application's performance, and verifying that scaling and security measures work as expected. After successful testing, clean up all AWS resources to avoid unnecessary charges.

24	You are a DevOps engineer responsible for managing the deployment pipeline and infrastructure for a web application. The application consists of a frontend and backend, both hosted on Amazon EC2 instances. Your goal is to establish a robust CI/CD pipeline and automate the deployment using AWS CloudFormation. Additionally, you need to set up monitoring and logging to ensure the health and performance of the application. Tasks: 1. CI/CD Pipeline Setup: - Implement a CI/CD pipeline using a CI/CD service of your choice (e.g., AWS CodePipeline, Jenkins). Connect it to your version control system (e.g., GitHub). - Configure the pipeline to trigger on code commits to the repository. - Set up build and deploy stages for both the frontend and backend components. 2. EC2 Instances and Autoscaling: - Launch Amazon EC2 instances for the frontend and backend using AWS CloudFormation. Define the necessary parameters, resources, and outputs in the CloudFormation template. - Implement an Auto Scaling group for the backend to ensure redundancy and scalability. Configure appropriate launch configurations and policies. 3. Code Deployment: - Automate the deployment of frontend and backend code using your CI/CD pipeline. Ensure that new code versions are deployed to EC2 instances without manual intervention. 4. Monitoring and Logging: - Configure CloudWatch Alarms to monitor key metrics (e.g., CPU utilization, network traffic) for your EC2 instances. Define meaningful thresholds and actions for each alarm. - Set up CloudWatch Logs for both frontend and backend components. Ensure that logs are centralized and easily accessible for troubleshooting. 5. Application Health Checks: - Implement custom health checks for your application. Define an endpoint that returns the health status of the frontend and backend. Configure the load balancer to use these health checks. 6. Rollback Mechanism: - Create a rollback mechanism in your CI/CD pipeline. Define conditions that trigger a rollback (e.g., failed health checks, high error rates). Ensure that the previous version is automatically restored. 7. Testing and Validation: - Test the pipeline by making code changes and observing the deployment process. Verify that the application is running as expected on the EC2 instances. 8. Clean-Up and Documentation: - Document the entire setup, including CI/CD pipeline configuration, CloudFormation template, and monitoring setup. - After successful testing, clean up any unused resources to avoid unnecessary charges.

25	You are responsible for setting up a CI/CD pipeline for a web application on AWS. The application runs on EC2 instances and is defined in CloudFormation templates (CFT). You must also ensure proper monitoring and logging of the application's performance and issues. Tasks: 1. CI/CD Pipeline Setup: Create a CI/CD pipeline using AWS CodePipeline and AWS CodeBuild. The pipeline should pull the application code from a source code repository (e.g., GitHub), build the code, and deploy it to EC2 instances. 2. EC2 Instances Deployment: Launch EC2 instances using CloudFormation templates (CFT). Define the CFT for EC2 instances, including necessary security groups, IAM roles, and user data to install and configure the application. 3. Pipeline Integration: Integrate the CI/CD pipeline with the CloudFormation stack that deploys EC2 instances. Ensure that the pipeline automatically deploys the application to new instances when changes are made to the source code. 4. Monitoring Configuration: Configure AWS CloudWatch to monitor EC2 instances. Set up custom metrics to monitor application-specific parameters, such as response times, error rates, and resource utilization. 5. CloudWatch Alarms: Set up CloudWatch alarms to alert you when certain conditions are met, such as high CPU utilization or excessive error rates. Define actions to be taken when alarms are triggered. 6. Logging and Troubleshooting: Configure EC2 instances to send application logs to CloudWatch Logs. Ensure that logs are stored securely and can be accessed for troubleshooting and analysis. 7. Pipeline Testing: Test the CI/CD pipeline by making changes to the application code and verifying that the pipeline automatically deploys the changes to EC2 instances. Monitor the process to ensure it works as expected.
26	You are a cloud engineer tasked with setting up a Virtual Private Cloud (VPC) to host a web application. The application will run on an Amazon EC2 instance. Your goal is to ensure that the VPC is properly configured and that the EC2 instance is launched and accessible. Tasks: 1. VPC Creation: Create a new VPC with the following specifications: VPC CIDR Block: 10.0.0.0/16 Public Subnet: 10.0.0.0/24 (us-east-1a) Private Subnet: 10.0.1.0/24 (us-east-1b) Internet Gateway: Attach an Internet Gateway to the VPC for internet access. 2. Route Table Configuration: Create two route tables - one for the public subnet and one for the private subnet. Ensure that the public subnet's route table has a route to the Internet Gateway. 3. Security Group Setup: Create a security group that allows inbound traffic on port 80 (HTTP) for the web application. Attach this security group to both the public and private subnets. 4. Key Pair Generation: Create an EC2 key pair for SSH access to the instances. Store the private key securely. 5. EC2 Instance Launch: Launch an Amazon EC2 instance in the public subnet with the following specifications: Amazon Machine Image (AMI): Amazon Linux 2 Instance Type: t2.micro Security Group: Use the security group created in step 3. Key Pair: Use the key pair generated in step 4. IAM Role: Attach an IAM role that allows basic EC2 permissions. 6. Public IP Assignment: Ensure that the EC2 instance has a public IP address. 7. Web Application Deployment: SSH into the EC2 instance and deploy a sample web application. You can use a basic HTML file or any sample application of your choice. 8. Access Verification: Access the web application via the public IP address of the EC2 instance using a web browser to verify that the deployment is successful.

27	You are a cloud engineer tasked with setting up a Virtual Private Cloud (VPC) and an Amazon EC2 instance in AWS. Your goal is to create a secure network environment and verify the correct deployment of the EC2 instance. Tasks: 1. VPC Creation: Create a new Virtual Private Cloud (VPC) using the AWS Management Console. Configure the VPC with a unique IPv4 CIDR block. Ensure that the VPC is located in a specific AWS region. 2. Subnet Setup: Create two subnets within the VPC. One subnet should be public and the other private. Associate an Availability Zone with each subnet. Define appropriate IPv4 CIDR blocks for each subnet. 3. Internet Gateway: Create an internet gateway and attach it to the VPC. Modify the route table of the public subnet to allow traffic to and from the internet via the internet gateway. 4. Security Groups: Configure security groups for the EC2 instance. Define rules for incoming and outgoing traffic to ensure that the EC2 instance is accessible over SSH (port 22) and HTTP (port 80). 5. Key Pair: Create an EC2 key pair to use for securely accessing the instance over SSH. 6. EC2 Instance Launch: Launch an Amazon EC2 instance within the public subnet. Choose a suitable Amazon Machine Image (AMI) and instance type. Use the previously created security group and key pair. Assign an Elastic IP address to the instance. 7.Connect to EC2: Connect to the EC2 instance using SSH to verify that it is running and correctly deployed. Ensure that it is accessible via its public IP address.
28	Create and Configure an EC2 Instance: 1. Launch a new EC2 instance using the AWS CLI, specifying instance type, key pair, and security group. 2. Attach an Elastic IP address to the EC2 instance. 3. SSH into the EC2 instance using the key pair.
29	Create an S3 Bucket, Upload, and Download Files: 1. Create a new S3 bucket with a unique name. 2. Upload a local file to the S3 bucket. 3. Download the file from the S3 bucket to your local machine.
30	Configure SNS and SQS for Messaging: 1. Create an SNS topic using the AWS CLI. 2. Create an SQS queue and subscribe it to the SNS topic. 3. Send a test message to the SNS topic and verify that it's delivered to the SQS queue.
31	You are tasked with deploying a static website for a small business using Amazon S3. The website will contain basic HTML, CSS, and JavaScript files. Your goal is to create an S3 bucket, configure it for static website hosting, upload the website files, and verify its accessibility.